SOVAR: SMART MEMORIES FOR OUT-OF-ORDER EXECUTION VLSI ARCHITECTURES

by

Gregory Ameriada Uvieghara

Memorandum No. UCB/ERL M89/41

14 April 1989

SOVAR: SMART MEMORIES FOR OUT-OF-ORDER EXECUTION VLSI ARCHITECTURES

by

Gregory Ameriada Uvieghara

Memorandum No. UCB/ERL M89/42

14 April 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Dedicated to my junior brother, Chris Israel Uvieghara
for all his support, sacrifice and dedication
to the family business.

SOVAR:Smart Memories for Out-of-Order Execution VLSI Architectures

Gregory Ameriada Uvieghara

Ph.D.

Department of Electrical Engineering
and Computer Science

ABSTRACT

Advances in semiconductor fabrication capabilities and CAD (Computer Aided Design) tools have made it feasible to design and fabricate single-chip processors. With increasing integration, more and more complexity is being put on chip to achieve higher and higher performance. One approach is to use the increasing integration to provide larger on-chip conventional memories on traditional sequential execution CPU chips. (A trend in this direction has been an increase in the use of larger and larger on-chip caches for implementing von Neumann architectures.) The opposite approach is to use the increasing chip real estate to implement less dense but more functional (smart) memories. These smart memories can be employed to support out-of-order execution architectures, which are non-von Neumann machines that attempt to achieve higher performance than traditional von Neumann architectures by waiving the sequential execution requirement. Although the advances in semiconductor technology and CAD now make it feasible to implement single-chip out-of-order execution architectures, there are several circuit design challenges that must be surmounted. These challenges relate to the smart memories that are needed for implementing out-of-order execution models. This dissertation addresses the circuit design challenges of the smart memories that are required for implementing out-of-order execution VLSI architectures and proposes some solutions.

David A. Hodges

Prof. David Albert Hodges

Chairman, Dissertation Committee

Acknowledgements

First of all, I want to proclaim my heart-felt gratitude to *Oghene* (God), my Father for His love and faithfulness. I also want to thank Him for giving me a second chance and for guiding me even when I was in rebellion. I want to thank my Savior, the Lord Jesus Christ for saving, healing and delivering me. I am grateful to God, the Holy Spirit for directing me to participate in the Ph.D. program, and for leading me throughout, especially during the chip verification , chip testing and dissertation write-up. I also thank the Holy Spirit for clearing my thoughts and for giving me creative insights into circuit design. I am thankful to *El Shaddai* (the God that is more than enough) for providing the finances even from far-away lands and for bringing me in contact with the right kinds of people at the right times.

I wish to express my sincere thanks to Rev. Kenneth Hagin for his teaching on faith that helped me tremendously during the trying moments. I thank Pastors Joe and Jean Perez, Brother Bob and Sister Cheryl Cambridge for helping me in the last crucial months with love, faith, hope, a new anointing and knowing God's will. I am especially thankful to Pastor Jean Perez for being sensitive to the Holy Spirit in helping me realize how the Ph.D. program fits in *Chukwu's* (God's) perfect will. I don't have enough space to express my thanks to Revs. Momoh, Etidia, Ejomah, Freeborn, Friesen, Copeland, Ogbeta, Chief Mosheshe and other ministers who helped me with their prayers and ministries.

I am grateful to Prof. Johnson, "Chief" Abiola and Dr. (Mrs.) Yetunde Taiwo and Prof. and Mrs. Aladekomo for "supercarving" and for christianly fellowship. I thank Kevin Kornegay, Valerie Taylor, Karen Martin, Glen Dunning, Art King and Jim Kelly for memorable moments and hours of witnessing on Telegraph Avenue. I also thank Greg Uehara for all the enriching moments of christian fellowship. I express my gratitude to Dr. Kola Aiyesimoju and Dr. Franklin Osaisai for loans and various forms of assistance.

On the academic side, I start by expressing my profound gratitude to Prof. David Hodges, my research adviser, for all his fatherlike support. He was more than a research adviser to me: he was also an intellectual Dad. I am especially thankful to him for his personal concern, his stimulation of my creativity, and his insightful, yet constructive criticisms. He stimulated my creativity by drawing the best out of me. He criticized my half-baked ideas without any alienation or friction. His personal care was such that he was able to generate funds for the research from a foreign country.

I thank Prof. Yale Patt, a qualifying exam committee member, whose insights and pressure at the early part of the research led to my first proposal. I am grateful to Prof. Robert Brodersen, qualifying exam committee chairman and dissertation committee member, for his insightful comments during the exam and for editing the dissertation. I am appreciative of the cooperation of Prof. Frederick Balderston, a member of my qualifying exam and dissertation committees. My discussion with him reminded me of my B-school days by revealing the business perspective. I thank my late MIT Professor, Prof. Kuh who encouraged me to apply to Berkeley. I would like to thank Prof. Alvin Despain for allowing me to use his DARPA MOSIS account to fabricate my chip. I am also grateful to Profs. Richard Muller and Alberto Sangiovanni-Vicentelli for all their cooperation during my 1984 health debacle.

I have special thanks for Dr. Wen-mei Hwu, who collaborated with me from the Computer Architecture viewpoint. I thank him for his careful and detailed explanation of out-of-order execution. In the area of circuit design, I am eternally grateful to Yoshinobu Nakagome (visiting Hitachi researcher; 1987/88) who implemented the Barrell Shifter. I am particularly grateful to him for his thorough circuit design review of the RAT and ALU, and the RAT noise margin tests. I appreciate all the help I received from Dr. Deog-Kyoon Jeong (D.K.). I thank him for availing me of his extensive circuit design experience, his pads, and his clock generator. I also thank him for his verification of my ideas and for helping me with various CAD tools. I am grateful to Dr. Daebum Lee for helping me with his VLSI circuit design insights, his SPUR experience, his ALU and information on the SPUR CPU. I acknowledge with

thanks the contribution to the thesis topic selection by Dev Chen, the intense review of the Abstract by Greg Uehara, the sense amplifier discussion by Sehat Sutarja, the setting up of the test bench by Ken Lutz and for various forms of technical assistance by Joey Doernberg and Phil Schrupp.

I thank Marvin Baron for helping me with immigration and financial problems, Ruth Tobey for helping me with a loan, and Genevieve Thiebaut , Beth Rhine, and Pearl Tranter for various forms of administrative assistance. I am grateful to Cheryl Craigwell for refreshing jokes and laughs.

I am indebted to my financial sponsors, DARPA and the Hitachi Central Research Laboratory, Japan.

On my family side, I start by expressing my gratitude to my late Dad posthumously, for his go-getter philosophy that has brought me this far. I am eternally grateful to my Mum, for paying very careful attention to my education in my early years of school, for being sensitive to God, the Holy Spirit in 1984 on my behalf, for supporting me throughout my Ph.D program and for forfeiting a lot for the program. I don't have enough words to thank my immediate junior brother, Israel Uvieghara for taking care of the family business, thereby freeing me to participate in the Ph.D. program without any burdens.

Finally, I thank other family members for all their help. I am grateful to Justice Akpomudjere and Chief Akamune for all their support of the family business while I was abroad pursuing the Ph.D. program. I thank Uncle Chris Ogwu, and his wife, Tees, and Malechi and Tony Ogwu for transmitting my phone messages to Warri. I thank Emma Osakwe and his wife, Mabel, for introducing me to Rev. Momoh, Meschack and Justy Ilobi for prayers and giving me their car to visit the Irri ministry, Fide and Christy Ossom for prayers and help during the 1985 crisis, and Onome, Angela, Augusta, Victor, and Mathias for their prayers and support throughout the program.

Table of Contents

1. Introduction 1

1.1. Smart Memories 1

1.2. Smart Memories for Out-of-Order Execution Architectures 3

1.3. Dissertation Outline 6

2. HPSm : An Out-of-Order Execution VLSI Architecture Case Study 8

2.1. Introduction 8

2.2. Overview 8

2.3. Major Components 9

2.4. Global Busses and Signals 11

2.5. Detailed Pipeline Timing 15

2.6. Conclusions 20

3. NT : A Smart Instruction Memory Case Study 22

3.1. Introduction 22

3.2. Memory Architecture 22

3.3. Basic Operations 24

3.4. Circuits 29

3.5. Simulation Results 32

3.5. Conclusions 33

4. RAT: A Smart Data Memory Case Study 35

4.1. Introduction 35

4.2. Memory Architecture 35

4.3. Basic Operations 36

4.4. Circuits 37

4.5. Experimental Results 48

4.6. Conclusions 54

5. Smart Memory Design Issues 56

5.1. Introduction 56

5.2. Issues and Proposed Solutions 56

5.3. Cost/Functionality Study 64

5.4. Conclusions 65

6. Conclusions 67

6.1. Future Work 67

6.2. Final Conclusions 67

CHAPTER 1

Introduction

1.

1.1. Smart Memories

Traditionally, in a digital system the logic/control section is conceptually separate from the data storage section as in Fig. 1a, with a distinct boundary separating the two sections. The memories in the data storage section are just conventional READ/WRITE memories. But when the logic/control section partially overlaps the data storage section, a new species of memory, **smart memory**, emerges as in Fig. 1b. This memory-logic hybrid derives its "smartness" from embedding logic in memory. It may be necessary to intermingle memory and logic in this way to improve the performance of the overall system. The granularity of the intermingling may be at the level of a data cell (i.e. each cell has embedded logic) or at the level of a memory subsystem (i.e. several cells share some embedded logic). So, one can visualize a "smartness" spectrum where on one end you have "full smartness" (i.e. each cell has embedded logic) and on the other end you have conventional memories (i.e. no cell has embedded logic). One can also define a logic-memory "conceptual distance" spectrum that corresponds to the "smartness" spectrum: "conceptual distance" of 0 for "full smartness" but "conceptual distance" of ∞ for conventional memories.

The content addressable memory (CAM) is a smart memory that can be used to illustrate the foregoing points. It has a logic-memory "conceptual distance" of "0". In the conventional memory, the address is given to access the data, while in the CAM, the data is given to access the address. Such a memory is used where it is necessary to know what memory words contain a certain bit pattern (like in

parallel processing, expert systems, and artificial intelligence applications). If an *N-word* RAM is used, *N* cycles would be needed to sequentially search through the *N* words of the memory to determine the memory words that match the given bit pattern. But by employing some logic in each data cell, a CAM can simultaneously match each memory word against the bit pattern in a parallel fashion. In other words, it performs parallel data search without extensive address handling. Therefore, a CAM can be viewed as an efficient hardware substitute for a RAM plus a searching algorithm. Adding a comparator logic to the RAM cell in Fig. 2a gives the CAM cell in Fig. 2b. The detailed circuits are in Figs. 3a & 3b. In Fig. 3b, the EXCLUSIVE-OR circuit is the comparator added to each RAM cell to give the CAM cell.

The video RAM (VRAM) is another smart memory that readily comes to mind. It has a non-zero logic-memory "conceptual distance" since clusters of simple data cells share the embedded logic. The VRAM is a smart memory that combines a conventional RAM with serial access registers and some control logic to support bit-mapped graphics display systems [1]. A bit-mapped graphics display system is a technique that is needed to efficiently implement real time graphic simulators, higher resolution displays, and combined text and graphics in color on a single screen. This technique provides unlimited flexibility in the images which can be displayed by allowing each pixel on the screen to be individually controlled by one or more bits of information in a bit-mapped memory. Traditional RAMs are inadequate in supporting bit-mapped graphics display systems because they do not have the necessary bandwidth to supply information for refreshing the screen while also allowing a graphics processor sufficient access to the memory to update it [2]. In comparison, the VRAM is ideally suited for bit-mapped graphics systems because of the way it uses 2 ports. The first port is a parallel port that allows data to be written to and read from the memory in a normal fashion, while the second port is connected to a high-speed internal shift register so the memory can turn out serial data at video frequencies.

In the area of computer architecture, smart memories are needed to relieve the so-called von Neumann bottleneck [3], [4]. The von Neumann bottleneck is the separation of processor from memory as in Fig. 1a and is the main determinant of the upper bound of the performance of von Neumann architectures [3]. The bottleneck is opened only slightly in very high performance von Neumann machines like the Cray-1, by employing a wide, high speed channel between the processor and memory [3]. This approach, however, does not address the problem fundamentally. The Connection Machine is a non-von Neumann machine that solves the bottleneck problem by permitting the processor and memory to co-mingle, thereby eliminating the bottleneck altogether [5]. A specific Connection Machine implementation -the CM-1- achieves this goal by linking 64K single bit processors, each with 4K bits of private memory as shown in Fig. 4 [6]. In this machine, processing is integrated into and distributed throughout the memory in such a way that an allocation of memory for any piece of data automatically allocates processing for that data. Here, the combination of 4K bits of conventional memory with 1-bit ALU and supporting logic gives a smart memory with a non-zero logic-memory "conceptual distance".

1.2. Smart Memories for Out-of-Order Execution Architectures

The out-of-order execution architecture is another non--von Neumann computing style that can employ smart memories. By executing instructions out-of-order, out-of-order execution architectures hope to achieve higher performance than traditional sequential execution machines. Out-of-order execution machines can use a centralized control like the **scoreboard** used in the CDC 6600 [7], or a decentralized control like the **Tomasulo Algorithm** used in the IBM 360/91 floating-point unit [8]. HPS (High Performance Substrate) [9] is an out-of-order execution microarchitecture that uses the latter approach. It uses data flow techniques to coordinate out-of-order execution. In other words, execution is driven by the availability of data. Operations whose operands are not ready do not block subsequent ones; rather they wait in rest areas (Node Tables) until their operands arrive. HPS employs a modified

Tomasulo Algorithm to achieve maximum concurrency while preserving any precedence. In the decentralized control approach used by HPS, some of the control is distributed throughout the memories making them "smart". For sake of clarity, let the HPS smart memories be divided into 2 classes: smart data memories (memories that store only data) and smart instruction memories (memories that store both data and instructions). The smart memories for HPS are listed below:

Type	Features
CPU SMART MEMORIES: 1) Register Alias Table (RAT) Smart Data Memory (SDM)	i) Has 3 ports for concurrent accesses by 3 function units. ii) Has 3 back-to-back cells to support Branch Prediction and Exception Handling. iii) A 5-bit multi-port CAM field for associative distribution of results into current and backup copies. iv) Maintains the data dependency information for each register entry by having a tag field. v) Does 15 accesses per cycle : 6 READs, 3 WRITEs and 6 associative WRITEs.
2) Node Table (NT) - Smart Instruction Memory (SIM)	i) Buffers microoperations awaiting data/control operands. Uses CAM for associative WRITE into waiting entries. ii) Has priority encoding scheme to schedule microoperations. iii) Discards appropriate microoperations during Branch or Exception Repair.
3) Active Instruction Table (AIT) - Smart Instruction Memory (SIM)	i) Used for detecting and retiring completed instructions. ii) Uses a content addressable

	counter to keep track of the unexecuted microoperations. iii) Keeps track of the exception status for each instruction. iv) Does sequential retirement using a shift register. v) Discards appropriate instructions during repair.
4) Memory Write Buffer (MWB) - Smart Instruction Memory (SIM)	i) Used in memory management unit. ii) Needs CAM to do memory data forwarding. iii) Does sequential retirement using a shift register. iv) Discards appropriate instruction during repair.
5) Node Cache - Smart Instruction Memory (SIM)	i) Used in the instruction unit as a smart on-chip instruction cache. ii) Uses a multi-port CAM. iii) Supports branch prediction and exception handling.
FLOATING POINT UNIT (FPU) SMART MEMORIES: 1) FPU RAT same as CPU RAT.	
CACHE CONTROLLER SMART MEMORIES: 1) Repair Stack - Smart Instruction Memory (SIM)	i) Discards appropriate memory writes.
2) Other Smart Memories - Smart Instruction Memories (SIMs)	i) Smart memories for instruction decoding and instruction unit stall signal generation.

The "smartness" of the HPS memories poses significant circuit design challenges. In the first place, the increased functionality reduces the bit density. Yet, it is imperative to have the memories

on-chip to exploit the high-speed advantage of on-chip communications. Therefore, one of the major circuit design challenges is to maximize the bit density without sacrificing the functionality. A longer cycle (because of the many operations that have to be performed per cycle), and higher power dissipation and higher peak currents (because of the multiplicity of bit lines and match lines that have to be precharged) are other problems resulting from the enhanced capability of these smart memories. Other difficulties include layout irregularity (resulting in pitch-matching problems etc.), longer (therefore more capacitive) bit lines, and increased design effort. Making these memories multi-port to support increased on-chip parallelism compounds the above problems. This dissertation explores the circuit design alternatives needed for implementing **Smart Memories for Out-of-Order Execution VLSI Architectures.** HPSm [10], a single-chip version of HPS, would be used as the reference out-of-order execution architecture.

1.3. Dissertation Outline

In Chapter 2, a brief description of HPSm would be presented to give the necessary motivation for the smart memories that will be discussed. The Node Table (NT), the HPSm Smart Instruction Memory, would be described in Chapter 3, while the Register Alias Table (RAT), the HPSm Smart Data Memory, would be described in Chapter 4. Chapter 5 gives a broad treatment of smart memory design alternatives and costs. The thesis ends with Conclusions in Chapter 6.

References

[1] Jean-Daniel Nicoud, "Video RAMs: Structure And Applications" in *IEEE Micro, February 1988.*

[2] Raymond Pinkham et. al., "A High Speed Dual Port Memory with Simultaneous Serial and Random Mode Access for Video Applications", *IEEE Journal of Solid State Circuits,* December 1984, pp.999-1007.

[3] R. Zippel "The Database Accelerator: Architecture", *SMP Internal Memo #1, MIT, April 1986.*

[4] Jon P. Wade "An Integrated Content Addressable Memory System" *Ph.D Thesis, MIT, May, 1988.*

[5] W. D. Hillis, "The Connection Machine (computer architecture for the new wave)," *MIT Artificial Intelligence Laboratory, Rept. 646,* September 1981.

[6] W. Daniel Hillis, "The Connection Machine," *Ph.D. Dissertation,* Electrical Engineering and Computer Science Department, MIT, Cambridge, MA, 1985.

[7] James E. Thornton, "Parallel Operation in the Control Data 6600," *AFIPS Proc. FJCC, pt. 2, vol. 26, 1964, pp. 33-40.*

[8] Tomasulo, R. M., "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," *IBM Journal of Research and Development, vol. 11, 1967, pp. 25-33.* Principles and Examples, McGraw-Hill, 1982.

[9] Yale N. Patt, Wen-mei Hwu, and Michael Shebanow, "HPS, A New Microarchitecture: Rationale and Introduction", *Proceedings of the 18th International Microprogramming Workshop,* Asilomar, CA, December, 1985.

[10] Wen-mei Hwu, "HPSm: Exploiting Concurrency to Achieve High Performance in a Single-chip Microarchitecture" *Ph.D. Dissertation, Computer Science Division, EECS Dept.,* University of California, Berkeley, CA. 94720, 1987.

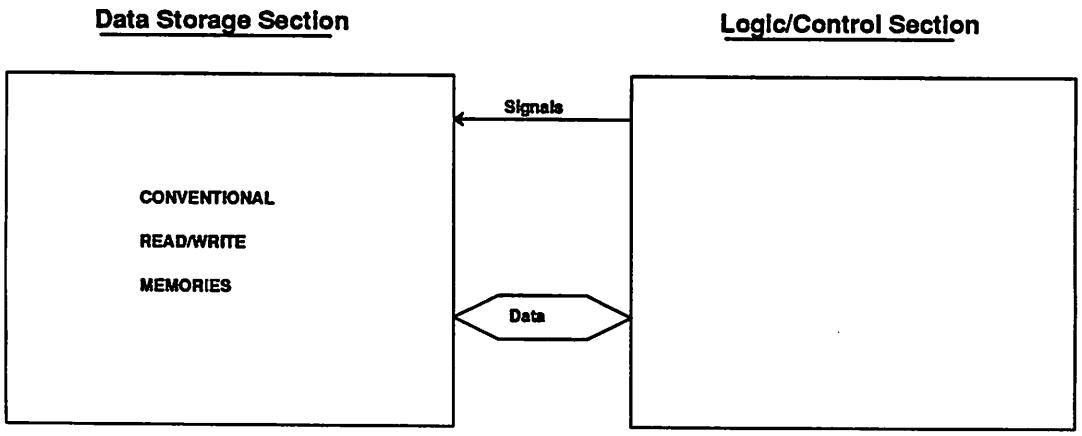


Fig. 1a : Conventional Digital System

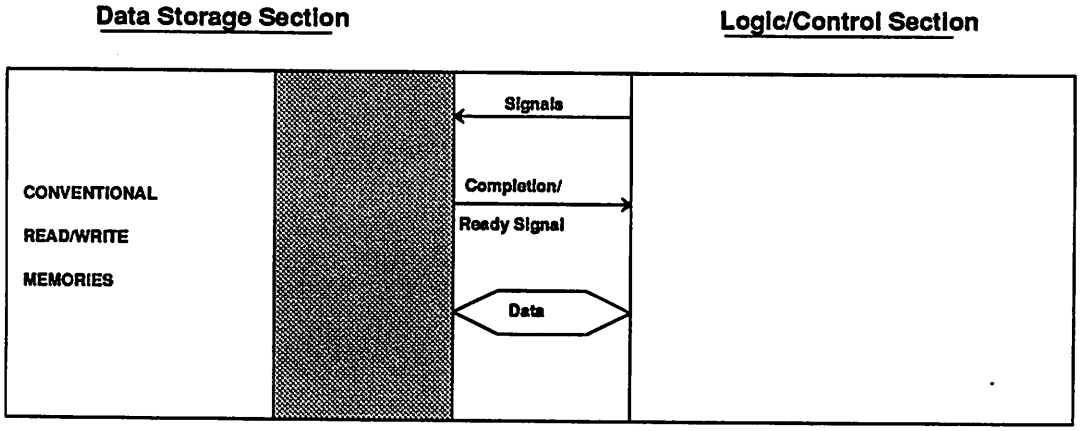


Fig. 1b: Digital System With Smart Memories

Conventional Memory

Smart Memory

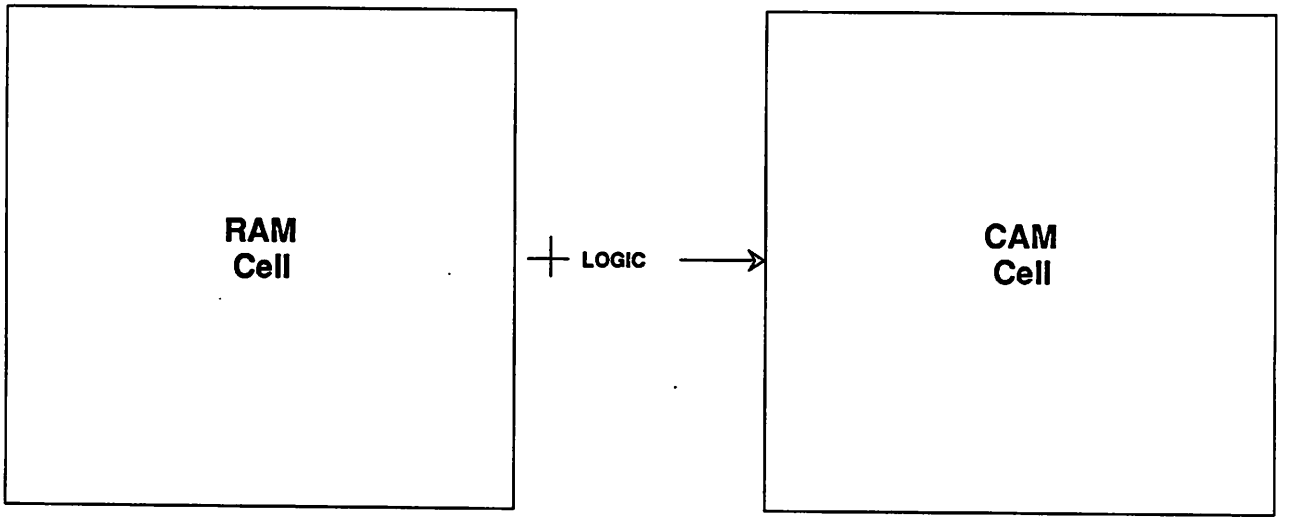


Fig. 2a

Fig. 2b

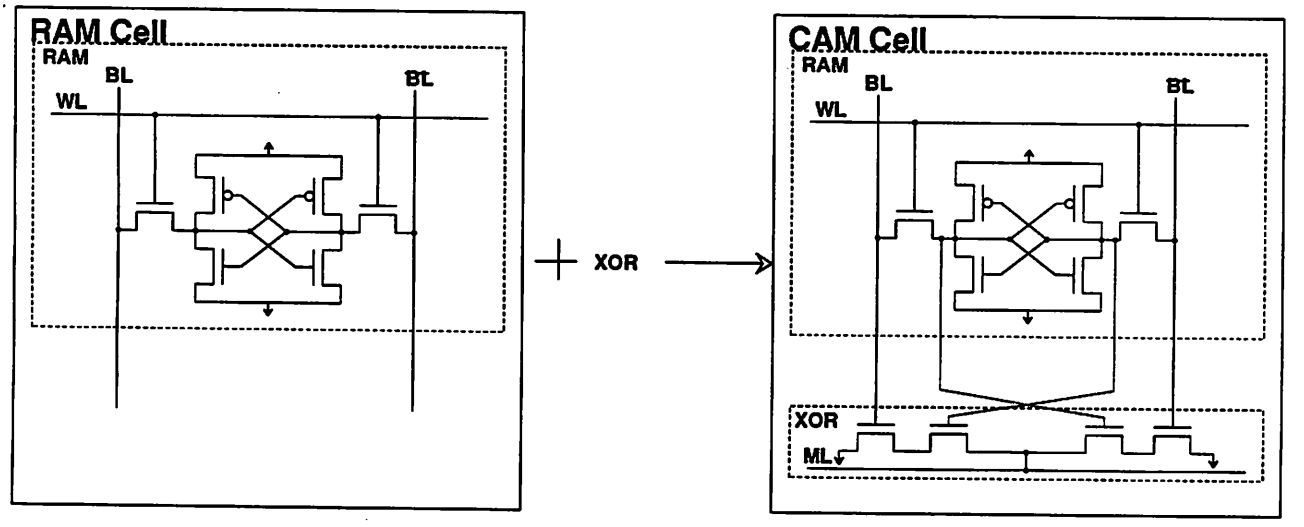


Fig. 3a

Fig. 3b

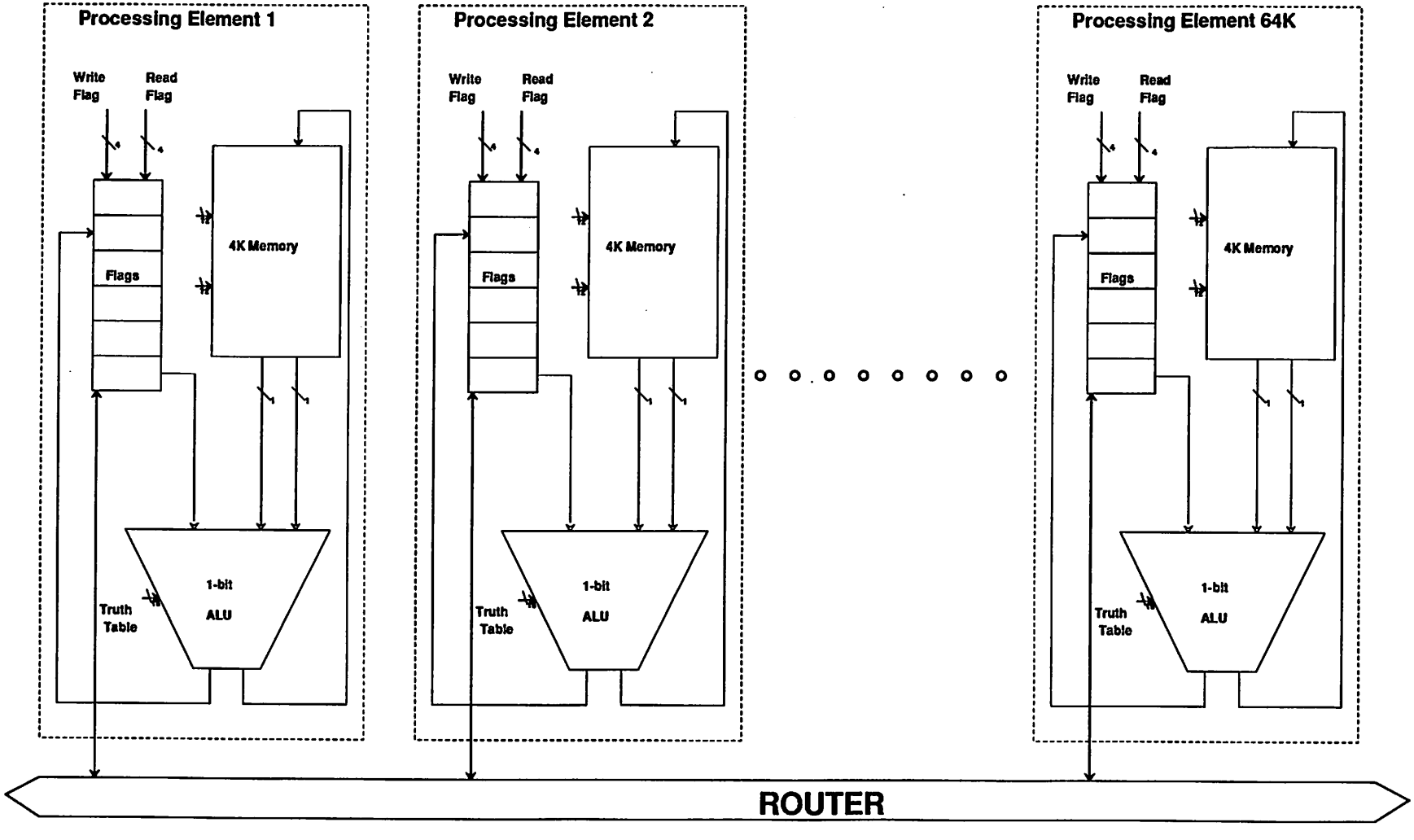


Fig. 4: CM-1 Processor Unit

CHAPTER 2

HPSm : An Out-of-Order Execution VLSI Architecture Case Study

2.

2.1. Introduction

This chapter gives the system context of the smart memories to be discussed in later chapters. It provides a qualitative description of HPSm [1] - the single-chip version of HPS - that is being used as the research vehicle for the study of **smart memories** for out-of-execution VLSI architectures. Its block diagram and data path are shown in Figs. 1 & 2 respectively.

2.2. Overview

HPSm is a data flow engine that can be viewed as a 5-stage pipeline as shown in Fig. 3 [2]. However, it must be stressed that the HPSm pipeline is unlike the classical pipeline in the sense that the execution of operations are data-driven. For this reason, Fig. 3 is termed a simplified view of the pipeline.

In the 1st stage (FETCH), a 64-bit *very long instruction word* (VLIW) [3] consisting of 3 "RISC-like" instructions (e.g ALU operation) is fetched by the IU using branch prediction. This instruction is decoded into a data dependency graph with each operation forming a node in this graph. Tags are allocated to the output result and input operands of each operation for data flow graph construction purposes. In the MERGE stage, the merger merges the decoder output (instruction data dependency graph) into the current data flow graph. The merging process uses a modified **Tomasulo Algorithm** [4] and the RAT.

The EXECUTE stage executes operations in a data-driven manner. The operations after having been decoded into data flow nodes are sent to 3 node tables : 1 node table for each function unit. Operations wait in the node tables until their operands arrive. An operation whose operands are ready is executed by the function unit attached to the node table. If more than one operation is ready, a scheduling algorithm is used to determine the oldest eligible operation for execution. After execution, the results are distributed on the distribution bus to 3 node tables (2 ALU NTs and MEM NT) and the RAT. Control information about completion is sent to the IU (so that new instructions can be brought into the engine) and the RETIRE and other control units.

Finally, in the RETIRE stage, an instruction retires from the machine after all its operations have been executed and all instructions before it have retired. Operations whose operands are not ready do not block the execution of other operations : they just wait until their operands arrive. The machine is therefore data-driven; hence, the name "data flow".

2.3. Major Components

The HPSm microarchitecture (see Chapter 5 of Reference [1]) consists of 6 major components: the instruction unit (IU), 4 smart memories (the Register Alias Table and 3 Node Tables), the function units, the Floating Point Unit (FPU) and the Memory System (see Fig. 1.). The first four components are designed for implementation in a single (11.5mm x 11.5mm, 1.6 μm) CMOS chip. This chapter will concentrate on the smart memories and how they are connected in the HPSm CPU chip. The FPU and Memory System would not be discussed; the interface to them will only be mentioned whenever appropriate.

2.3.1. Instruction Unit

The IU (Fig. 2.) fetches instructions, determines the next instruction address, performs branch prediction, executes unconditional jump and branch instructions, assigns tags to instructions and generates SAVE/REPAIR signals.

2.3.2. Node Table (Smart Instruction Memory)

For each function unit, the corresponding node table (NT) buffers the operations (e.g. ALU ADD) waiting for their operands to arrive. Using the **Tomasulo Algorithm**, it enforces data flow dependencies between operations and removes data antidependencies *. It schedules operations and discards appropriate operations during repair. This smart memory will be covered in detail in Chapter 3.

2.3.3. Register Alias Table (Smart Data Memory)

The Register Alias Table (RAT) is a smart register file that has high bandwidth to support 15 accesses per cycle. It maintains dependency information for each register entry, provides the source operand values and the tag of the operation producing the source operand values, removes data output dependencies and repairs the contents of each register entry if necessary. This smart memory will be covered in detail in Chapter 4.

2.3.4. Function Units

The function units perform branch prediction verifications, arithmetic and logic operations, data memory accesses, and coprocessor data transfers.

---

* An operation, A, has an anti data dependency on another operation, B, if both share the same register but A must read from the register before B writes into it so that A does not get the unintended value. An operation, C, has an output data dependency on another operation, D, if both must write into the same register but C must write before D does so that the operations that depend on C's result do not get D's result instead.

2.4. Global Busses and Signals

The global busses are the REGISTER ADDRESS bus (3 x 6 bits in Fig. 4), the TAG bus (3 x 5 bits in Fig. 5), the READY bus (3 x 1 bit in Fig. 6), the VALUE bus (3 x 32 bits in Fig. 7), and the CHECKPOINT IDENTIFICATION bus (3 x 1 bit in Fig. 8). The global signals are the EXCEPTION SAVE signal (1 bit), the EXCEPTION REPAIR signal (1 bit), the EXCEPTION STATUS signal (3 bits), the JUMP TARGET PENDING signal (1 bit), the BRANCH PENDING signal (1 bit), the BRANCH REPAIR signal (1 bit), the BRANCH VERIFY signal (1 bit), the NODE TABLE OVER-FLOW signal (1 bit), and the IU STALL signal (1 bit). Figure 9 summarizes the timing of the global busses and signals.

2.4.1. REGISTER ADDRESS Bus

The REGISTER ADDRESS bus connects the RAT, the IU, the NT's and the function units (Fig. 4). It is made up of 3 sub-busses; one for each function unit. Each sub-bus consists of 1 VALID bit and 5 ADDRESS bits. The VALID bit inhibits or permits an access while 5 ADDRESS bits indicate the register entry to be accessed. The bus switches before the rising edges of the 4-phase clocks and stays stable for a phase as shown in Fig. 9. In phase 1 (phase 2), the IU drives each sub-bus with the register address of the source 1 (source 2) operand for the corresponding function unit. In phase 3, each FU drives its address sub-bus with the VALID bit indicating whether the result on its VALUE sub-bus is valid, and drives the ADDRESS sub-bus with the destination address for the result. In phase 4, the IU drives each sub-bus with the destination register address for each microoperation in the VLIW.

2.4.2. TAG Bus

The TAG bus connects the IU, the RAT, the NT's and the FU's (Fig. 5). It consists of 3 5-bit sub-busses. The TAG bus switches before the rising edges of the 4-phase clocks and stays stable for a phase.

In phase 1 (phase 2), the RAT drives the TAG bus with the tags fetched from the source 1 (source 2) registers of the issued microoperations. In phase 3, the FU's drive the TAG bus with the tags of the distributed results. In phase 4, the IU drives the TAG bus with the tags of the results of the issued microoperations.

2.4.3. READY Bus

The READY bus connects the IU, the RAT, the NT's and the FU's (Fig. 6). It consists of 3 1-bit sub-busses. The READY bus switches before the rising edges of the 4-phase clocks and stays stable for a phase. In phase 1 (phase 2), if the operand 1 (operand 2) addressing mode is *register*, the RAT drives the corresponding READY bus with the READY bit fetched from the source registers. Otherwise the READY bus is driven to "1". In phase 3, the FU's drive each READY sub-bus to indicate if the corresponding VALUE sub-bus contains a valid distribution result. In phase 4, the IU drives the READY bus to "0".

2.4.4. VALUE Bus

The VALUE bus connects the IU, the RAT, the NT's and the FU's (Fig. 7). It consists of 3 32-bit sub-busses. The bus switches before the rising edges of the 4-phase clocks and stays stable for a phase. In phase 1 (phase 2), if the operand 1 (operand 2) register address is not "Register 31", the RAT drives the corresponding VALUE bus with the operand values fetched from the source registers. Otherwise, if the register address is "Register 31", the IU drives the VALUE bus with the FETCH PC value; otherwise the IU drives the bus with the sign extended result of the source register number (for *literal* mode). In phase 3, the FU's drive the VALUE bus with their evaluation results. In phase 4, the contents of the bus are *don't care*.

2.4.5. CHECKPOINT IDENTIFICATION Bus

The CHECKPOINT IDENTIFICATION bus connects the IU, the NT's and the FU's (Fig. 8). It consists of 3 1-bit sub-busses. It is used only in phases 2 and 3. In phase 2, the IU drives the bus with the CHECKPOINT IDENTIFICATION of the currently active checkpoint. In phase 3, the FU's drive the bus with the identification of the checkpoint to whose exception-repair range (see Chapter 4 of Reference [1]) the finishing microoperation belongs to.

2.4.6. EXCEPTION SAVE Signal

The EXCEPTION SAVE signal is generated by the IU and is monitored by the IU, the RAT, the NT's, and the FPU (Fig. 10). It indicates whether an exception check action is to be performed in a cycle. It switches at the rising edge of ϕ_3 (Fig. 9) and remains stable for the entire cycle. It crosses the chip boundary.

2.4.7. EXCEPTION REPAIR Signal

The EXCEPTION REPAIR signal is generated by the IU and is monitored by the IU, the RAT, the NT's, the FPU, and the memory system (Fig. 10). It indicates whether an exception repair action is to be performed in a cycle (see Chapter 4 of Reference [1]). It switches at the rising edge of ϕ_4 (Fig. 9) and remains stable for the entire cycle. It crosses the chip boundary.

2.4.8. EXCEPTION STATUS Signal

The EXCEPTION STATUS signal is generated by the function units and is monitored by the IU (Fig. 11). It consists of 3 1-bit sub-signals. It indicates what type of exceptions occured during the FU evaluation of the last cycle. It switches at the rising edge of ϕ_3 and remains stable for a phase (Fig. 9).

2.4.9. BRANCH VERIFY/REPAIR Signals

The BRANCH VERIFY/REPAIR signals are generated by the SECONDARY_ALU and monitored by the FPU, the IU and the NT's (Fig. 12). They switch at the rising edge of ϕ_3 and remain stable for the entire cycle. They cross the chip boundary.

2.4.10. BRANCH PENDING and JUMP TARGET PENDING Signals

The BRANCH PENDING and the JUMP TARGET PENDING signals are generated by the IU and monitored by the FPU, the IU and the NT's (Fig. 13). The BRANCH PENDING signal indicates whether there is a branch prediction yet to be verified by the SECONDARY_ALU (see below). The JUMP TARGET PENDING signal indicates whether there is an active microoperation which is going to write into REGISTER 31. The signals switch at the rising edge of ϕ_4 and remain stable for the entire cycle (Fig. 9). They cross the chip boundary.

2.4.11. NODE TABLE OVERFLOW Signal

The NODE TABLE OVERFLOW signal is generated by the NT's and is monitored by the instruction cache controller (Fig. 14) that is off-chip. It indicates whether any of the NT's is full. It switches at the rising edge of ϕ_1 and remains stable for the entire cycle. It crosses the chip boundary.

2.4.12. INSTRUCTION UNIT STALL Signal

The INSTRUCTION UNIT STALL signal is generated by the instruction cache controller and is monitored by the IU, the NT's, and the FPU (Fig. 15). The signal crosses the chip boundary and is set TRUE in phase 2 if at least one of the following is true:

(1) the instruction cache misses;

(2) the NODE TABLE OVERFLOW signal is TRUE;

(3) the BRANCH PENDING signal is TRUE and the instruction output latch contains a conditional branch microoperation;

(4) the JUMP TARGET PENDING signal is TRUE and the instruction output latch contains a jump instruction.

2.4.13. NEW CONDITIONAL BRANCH and NEW JUMP Signals

The NEW CONDITIONAL BRANCH and the NEW JUMP signals are generated by the instruction cache controller and monitored by both the IU and the instruction cache controller. The NEW JUMP signal indicates whether a jump microoperation has just been fetched. The NEW CONDITIONAL BRANCH signal indicates whether a new conditional branch microoperation has just been fetched.

2.4.14. NEW MICROOPERATIONS Signal

The NEW MICROOPERATIONS signal is generated by the IU and is monitored by the NT's (Fig. 16). It provides the opcodes, the destination register addresses, and the destination tags for the new microoperations in phase 4.

2.5. Detailed Pipeline Timing

2.5.1. Core Timing

This subsection gives a detailed description of the HPSm pipeline. The simplified pipeline discussed in section 2.2 above followed an instruction from its entry into the machine to its exit. It did not shed enough light on the role of the smart memories. In this subsection, the pipeline is viewed from the standpoint of the smart memories rather than from that of an individual instruction. Let the *core timing*

[1] of the pipeline be defined as the minimum signal propagation timing for each microoperation. That is, the *core timing* is the ideal timing for any microoperation without events such as IU stalls, multiple cycle FU evaluation, and delays due to data dependencies.

The *core timing* is shown in Fig. 17. Each HPSm microoperation spends at least 4 cycles in the pipeline. The 4 cycles are in turn divided into 8 sub-cycles as described below:

Sub-cycle	Description
sub-cycle 1	This sub-cycle is the entire first cycle during which the FETCH_PC is determined by the IU. This involves incrementing the last FETCH_PC, adding an offset to it, and selecting 1 of the 6 potential PC's according to the conditions by the instruction decoding, external interrupts and data path execution.
sub-cycle 2	This sub-cycle is made up of phases 1, 2, and 3 of the second cycle. In this sub-cycle, the FETCH_PC is transmitted off-chip to the instruction cache and used to access the tag and data stores.
sub-cycle 3	This sub-cycle is phase 4 of the second cycle. In this sub-cycle, the first part (the source addressing modes and register numbers) of the fetched instruction is fetched from the instruction cache.
sub-cycle 4	This sub-cycle is phase 1 of the third cycle. The first source operands are fetched from the RAT if the mode is not *literal* or the register is not REGISTER 31. If the mode of a source operand is *literal*, the sign extended result of the register number is chosen to drive the TAG, the READY, and the VALUE busses. If the mode is *register* and the register is REGISTER 31, the IU drives the TAG, the READY, and the VALUE busses. The operands are latched into the NT's at the rising edge of ϕ_2.
sub-cycle 5	This sub-cycle is phase 2 of the third cycle. The second source operands are fetched from the RAT if the mode is not *literal* or the register is not REGISTER 31. If the mode of a source operand is *literal*, the sign

	extended result of the register number is chosen to drive the TAG, the READY, and the VALUE busses. If the mode is *register* and the register is REGISTER 31, the IU drives the TAG, the READY, and the VALUE busses. The operands are latched into the NT's at the rising edge of ϕ_3. The second part of the fetched instruction (opcodes and output operand specifiers) is fetched from the instruction cache in this sub-cycle. The IU STALL signal (evaluated in sub-cycles 3 and 4) is also transmitted on-chip in this sub-cycle.
sub-cycle 6	This sub-cycle is phase 3 of the third cycle. The input operands of the fetched microoperations,if they are not ready, will monitor the TAG, READY, and VALUE busses in this sub-cycle to receive the operand values.
sub-cycle 7	This sub-cycle is phase 4 of the third cycle. If both input operands of the incoming microoperations are ready, and the *pending* bit is "1", the microoperation becomes eligible for execution. If there is no older eligible microoperation in the same NT, the microoperation is fetched from the NT and submitted for execution.
sub-cycle 8	This sub-cycle consists of phase 1, 2, and 3 of cycle 4. The FU evaluates a microoperation and puts its result on the TAG, READY, and VALUE busses.

2.5.2. Extended Timing

If events like stalls and delays occur, a microoperation can spend more than 4 cycles in the HPSm pipeline. When the *core timing* is extended for some microoperations, it may not be extended for the subsequent microoperations. The events that can prevent a microoperation from immediately advancing from one stage to the next are discussed below.

2.5.2.1. Instruction Unit Stalls

The IU can stall due to instruction translation buffer misses, instruction cache misses, jump target pending, node table overflow , or double pending branch prediction.

2.5.2.1.1. Jump Target Pending Stall

The *jump target pending* stall occurs when the next fetch address cannot be determined at the moment a jump operation is executed. In the HPSm microarchitecture, control transfer to an arbitrary program location is performed by having a microoperation loading REGISTER 31 with the target address and then execute a jump microoperation. If the microoperation loading REGISTER 31 has not finished execution when the jump microoperation is fetched, all the microoperations within the fetched instruction will be prevented from being latched into the NT's and from reserving the output RAT entries. They will keep repeating their third sub-cycle of the core timing until the *jump target pending* signal goes to "0". None of the subsequent instructions will be fetched.

2.5.2.1.2. Node Table Overflow Stall

The *node table overflow* stall occurs when at least one of the NT's is full. The microoperations within the fetched instruction will be prevented from being latched into the NT's and from reserving the output RAT entries. They will keep repeating their third sub-cycle of the core timing until the *node table overflow* signal goes to "0". None of the subsequent instructions will be fetched.

2.5.2.1.3. Double Pending Branch Prediction Stall

The *double pending branch prediction* stall occurs if another conditional branch is encountered in the instruction stream before the prediction for the previous one has been verified. The second conditional branch instruction will be forced to go through the third sub-cycle of the core timing until the BRANCH PREDICTION PENDING signal goes to "0". None of the subsequent instructions will be

fetched.

2.5.2.2. Data/Control Dependency Delays

If at least one of the input operands is not ready at the time a microoperation reaches sub-cylce 6, it will be forced to go through sub-cycle 6 again until both the input operands are ready. This means that the delayed microoperation will stay in an NT and monitor the TAG, the READY, and the VALUE busses to receive the input operand(s). There can be several microoperations waiting at sub-cycle 6 in the NT's at the same time. The microengine continues issuing the subsequent instructions.

2.5.2.3. Function Unit Conflict Delays

If there is more than one microoperation eligible for execution at the same time in the same NT, only the oldest firable microoperation is fired to the attached FU for execution. The others will have to wait until they become the oldest firable entry; therefore, they are forced to go through sub-cycle 7 again. The microengine continues issuing the subsequent instructions.

2.5.2.4. Multiple-cycle FU Evaluation

If the microoperation takes more than one cycle to evaluate, it will not finish sub-cycle 8 until the FU finishes the evaluation. Since all the multiple-cycle FU's are pipelined in the HPSm microarchitecture, a multiple-cycle FU evaluation does not block any of the subsequent instructions.

2.6. Conclusions

The foregoing paragraphs reveal that HPSm is an out-of-order execution microarchitecture where several events occur in the various components in an overlapped and decentralized manner. HPSm uses this decentralized control approach to maximize the throughput and to avoid the complexity problems of controlling an out-of-order execution machine in a centralized fashion. It is clear from the above discussion that smart memories are needed to support HPSm's style of decentralized control. The roles that the smart instruction memories (NT's) and the smart data memory (RAT) play in supporting the decentralized control are borne out by the above paragraphs.

The roles of the smart memories were clarified by looking at the pipeline not only from the viewpoint of the instructions but also from the viewpoint of the memories. The above sections reveal the activities in the smart memories when an instruction can be executed with the *core timing* (i.e. minimum timing with no stalls) and when it cannot because of stalls. The NT is discussed in the next chapter while Chapter 4 covers the RAT. The paramount constraint on the design of these smart memories is that they must satisfy the requirements specified in this chapter in a single-chip environment.

References

[1] Wen-mei Hwu, "HPSm: Exploiting Concurrency to Achieve High Performance in a Single-chip Microarchitecture" *Ph.D. Dissertation, Computer Science Division, EECS Dept.*, University of California, Berkeley, CA. 94720, 1987.

[2] Wen-mei Hwu and Yale Patt, "HPSm, a High Performance Restricted Data Flow Architecture Having Minimal Functionality", in *The 13th International Symposium on Computer Architecture Conference Proceedings, Tokyo, Japan, June 1986.*

[3] Joseph A. Fisher, "The VLIW Machine: A Multiprocessor for Compiling Scientific Code" *IEEE Computer, vol. 17, July 1984, pp. 45-53.*

[4] Tomasulo, R. M., "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," *IBM Journal of Research and Development, vol. 11, 1967, pp. 25-33.* Principles and Examples, McGraw-Hill, 1982.

HPSm Block Diagram

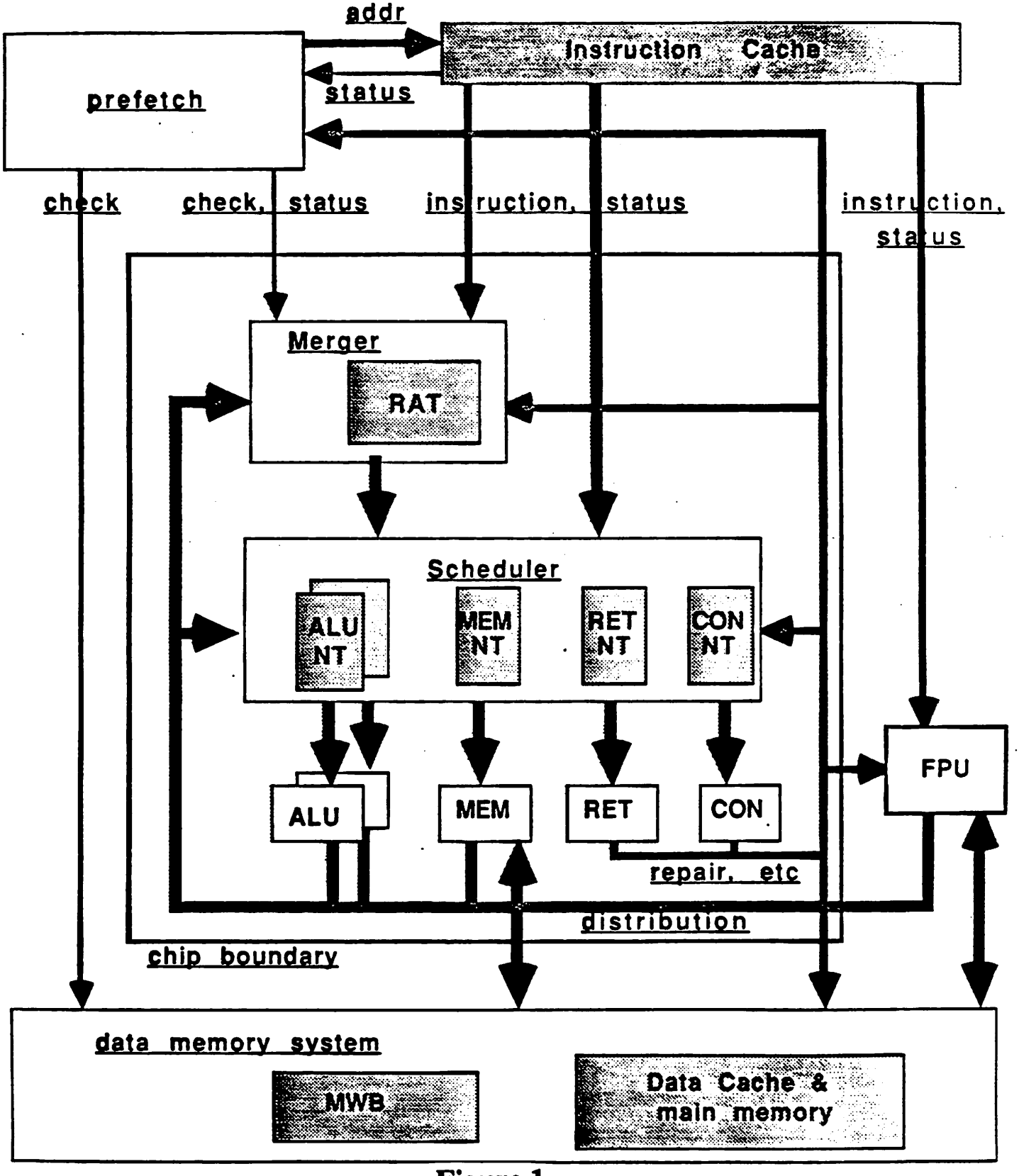


Figure 1

Data-Path Current Design

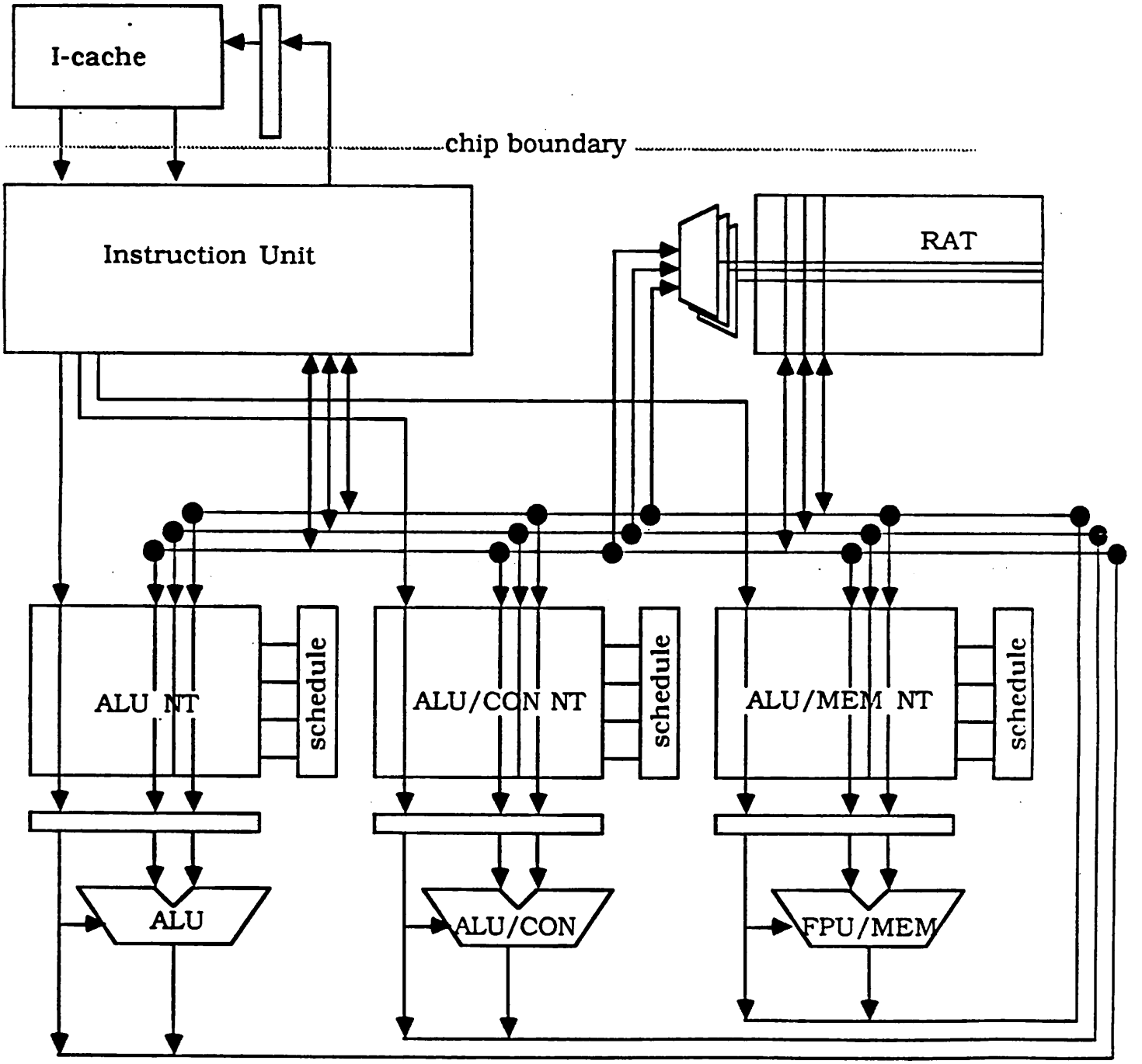


Figure 2

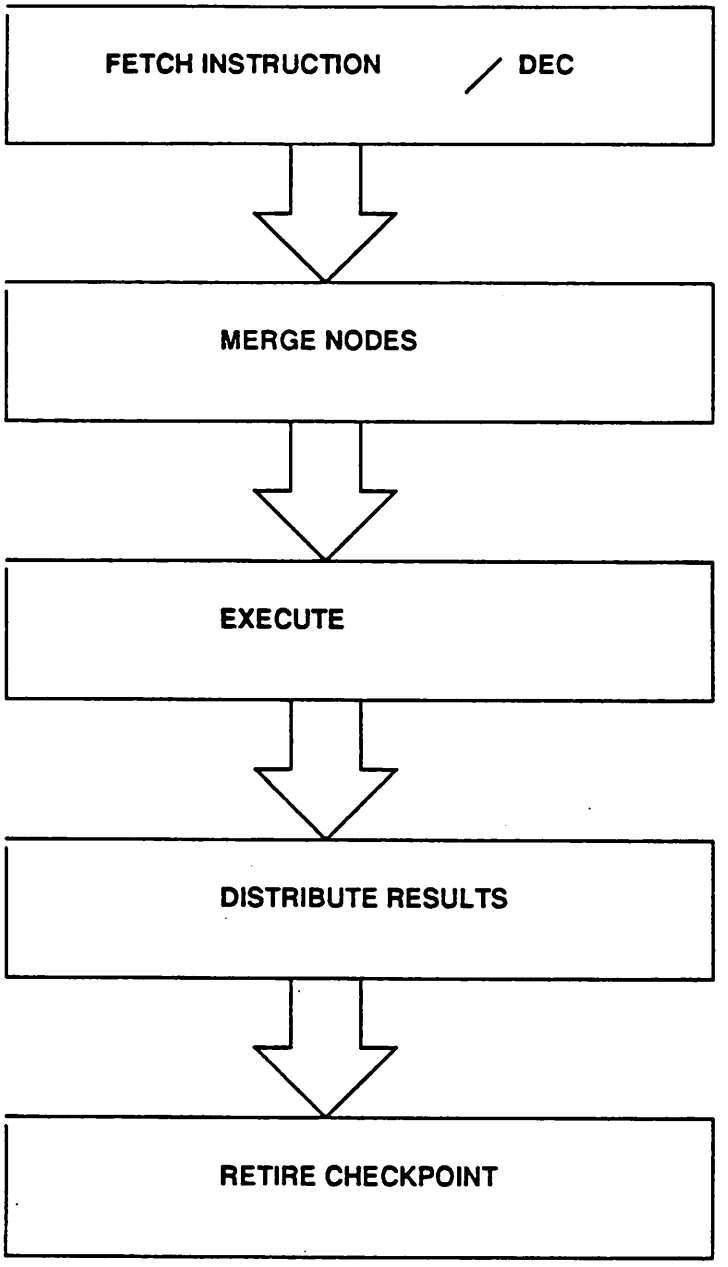


Fig. 3: HPSm Simplified Pipeline

The Register Address Bus

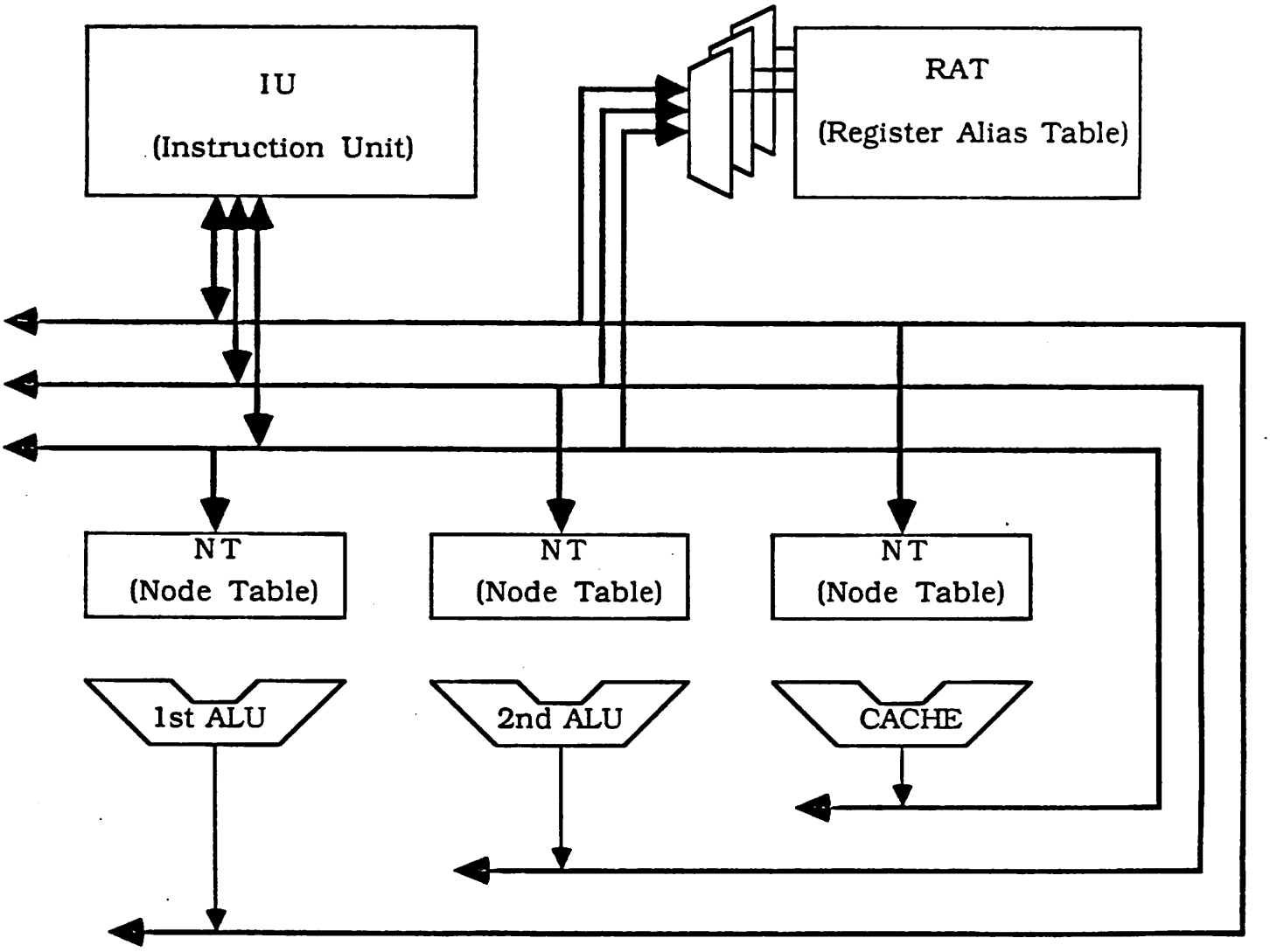


Figure 4

The Tag Bus

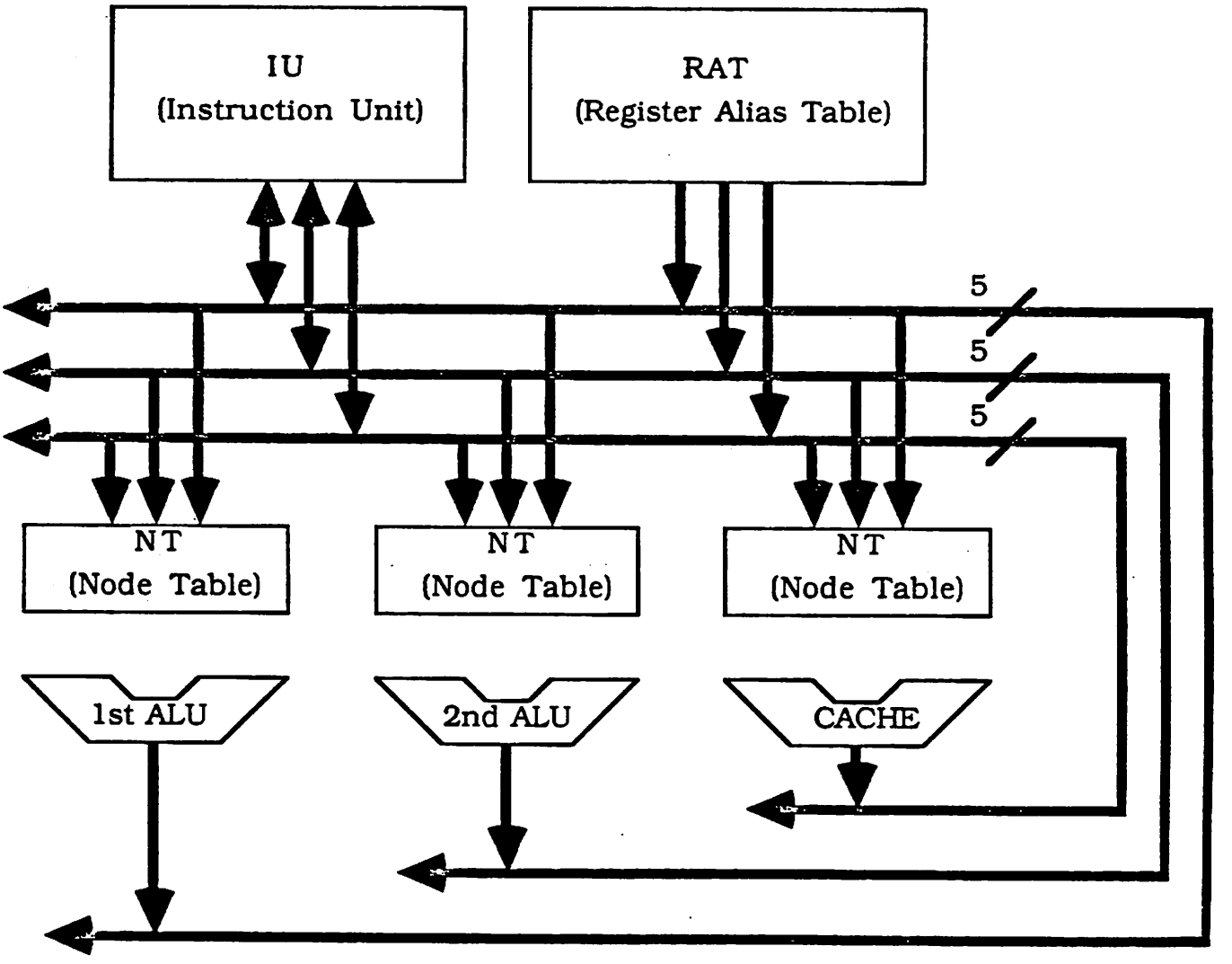


Figure 5

The ready Bus

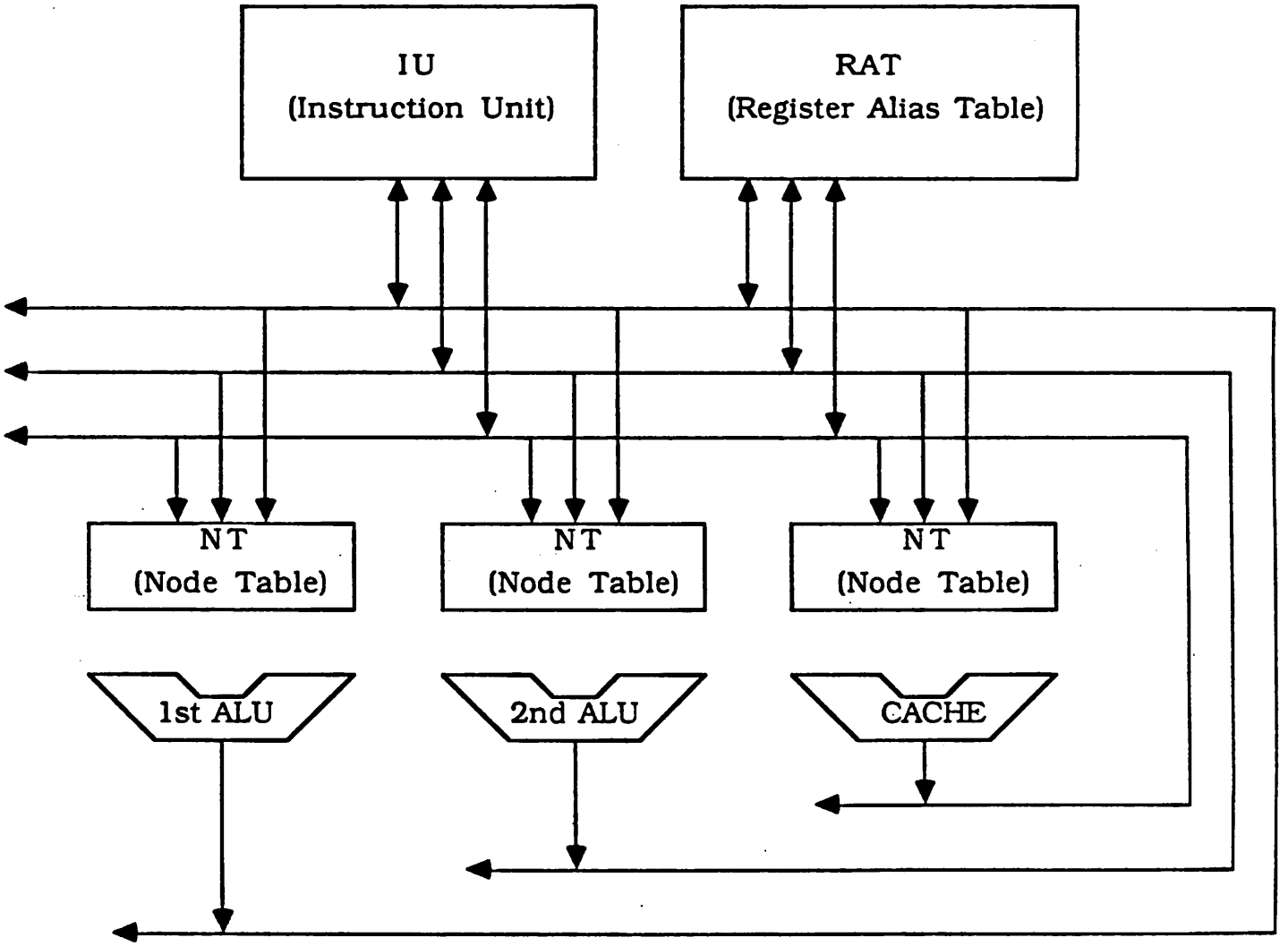


Figure 6

The Value Bus

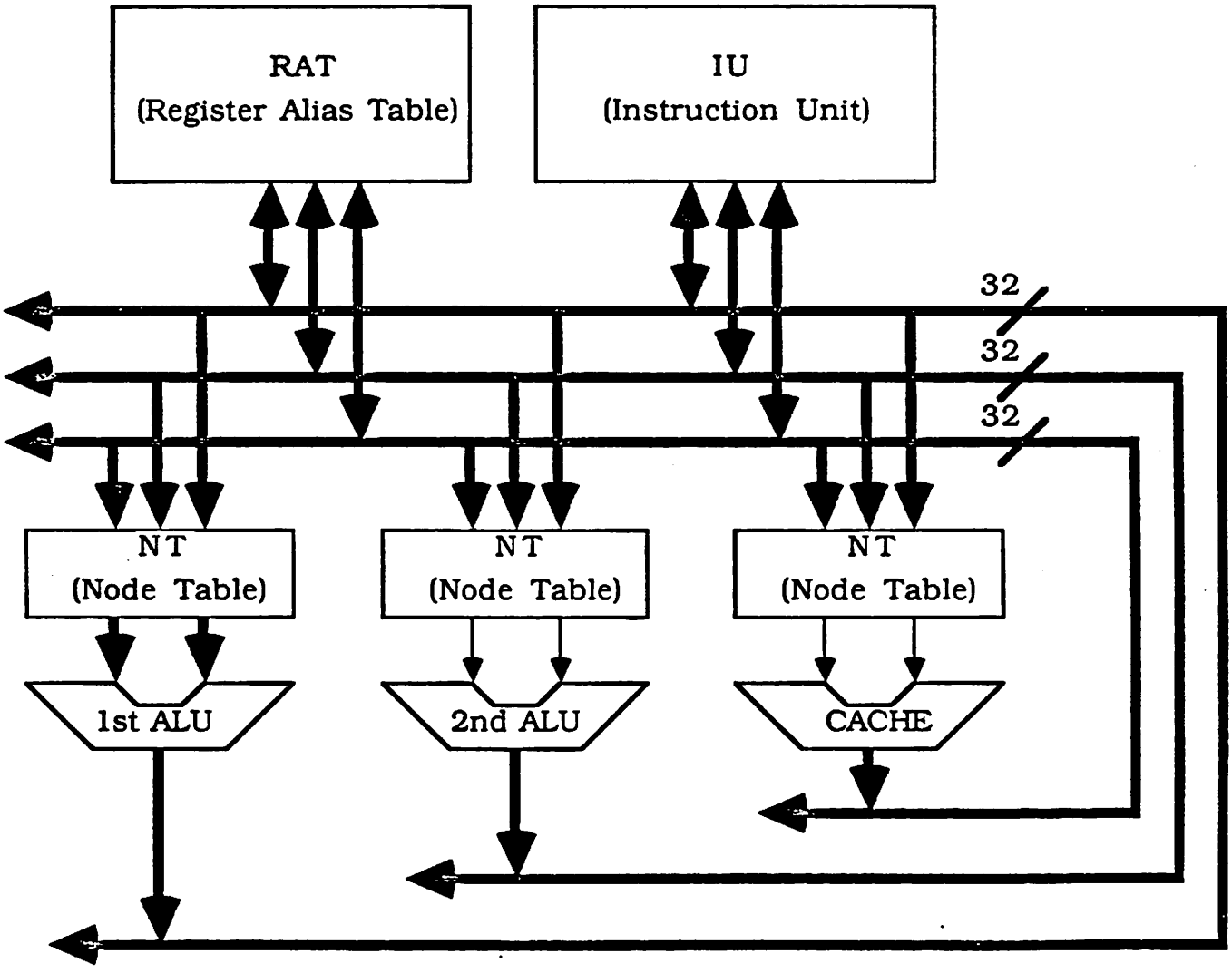


Figure 7

The Checkpoint Identification Bus

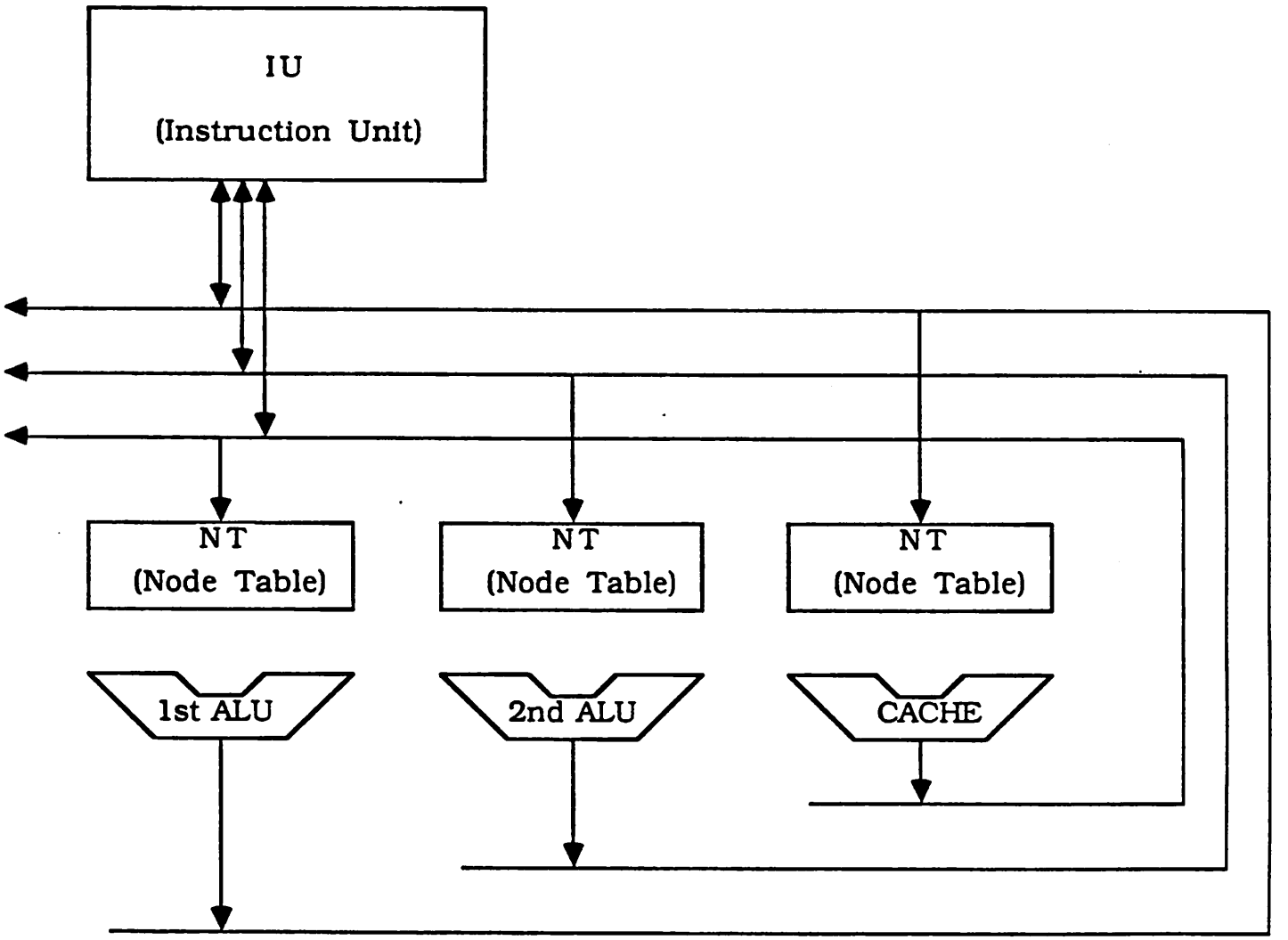


Figure 8

Global Bus and Signal Timing Table

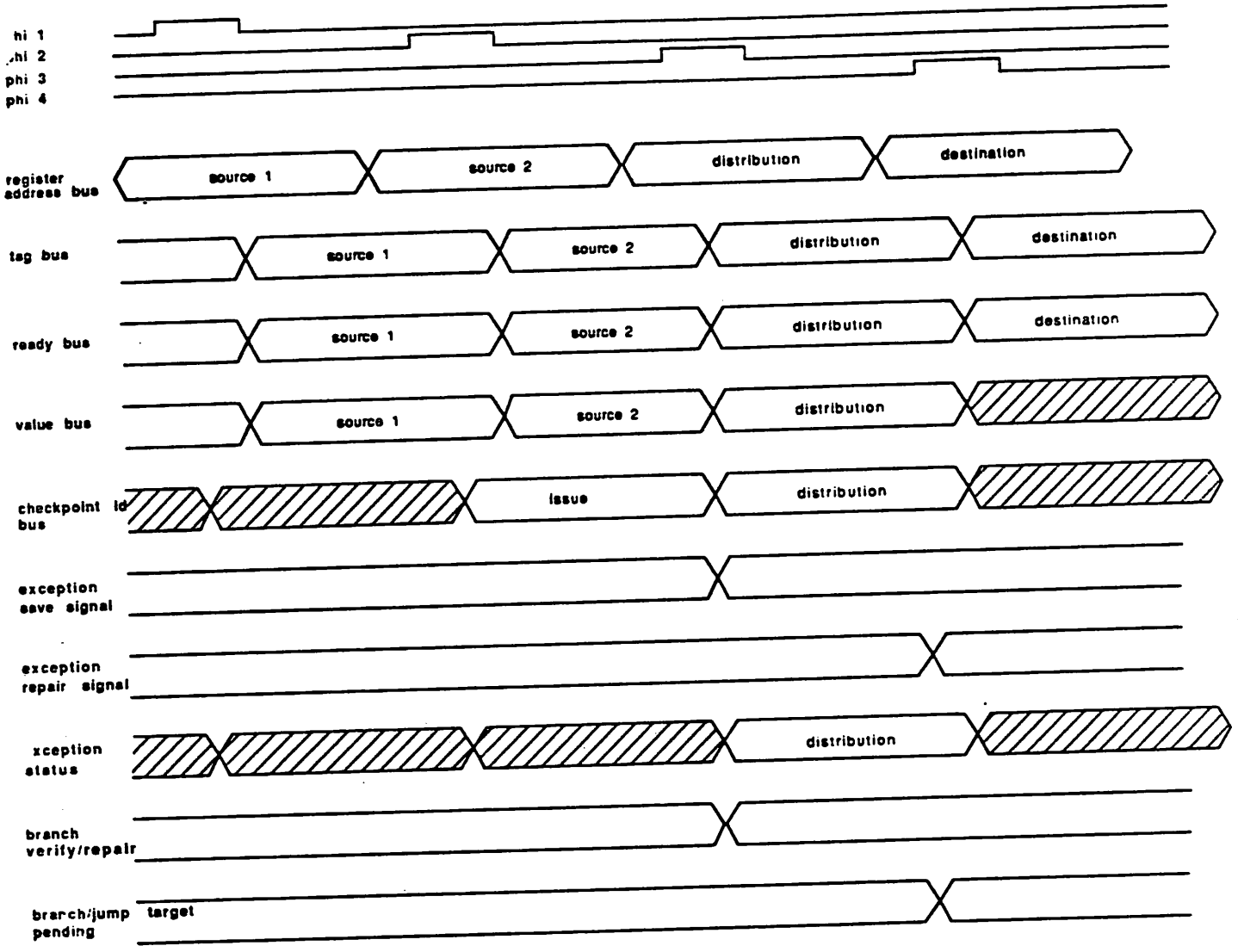


Figure 9

The E-SAVE and E-REPAIR Signals

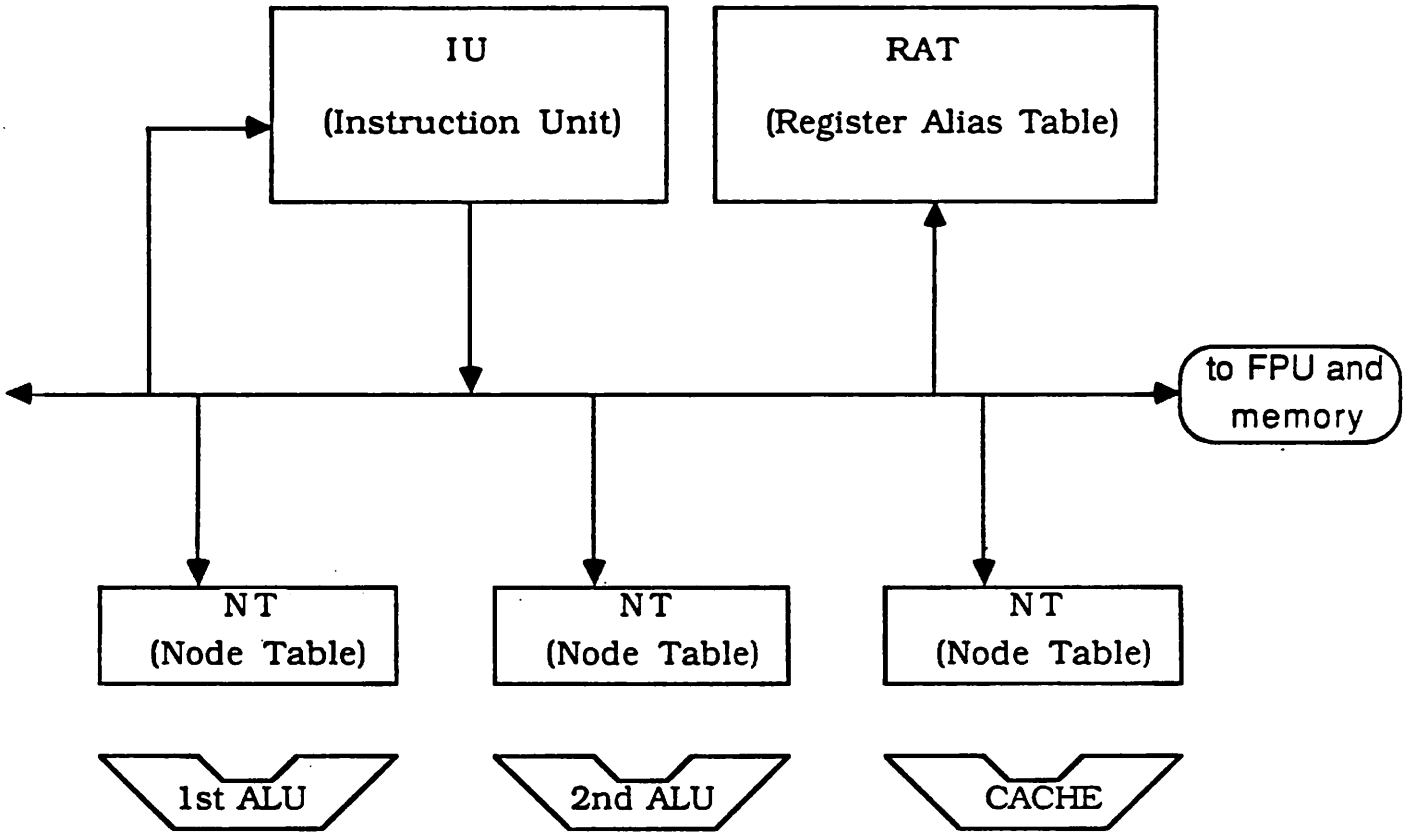


Figure 10

The Exception Status Signal

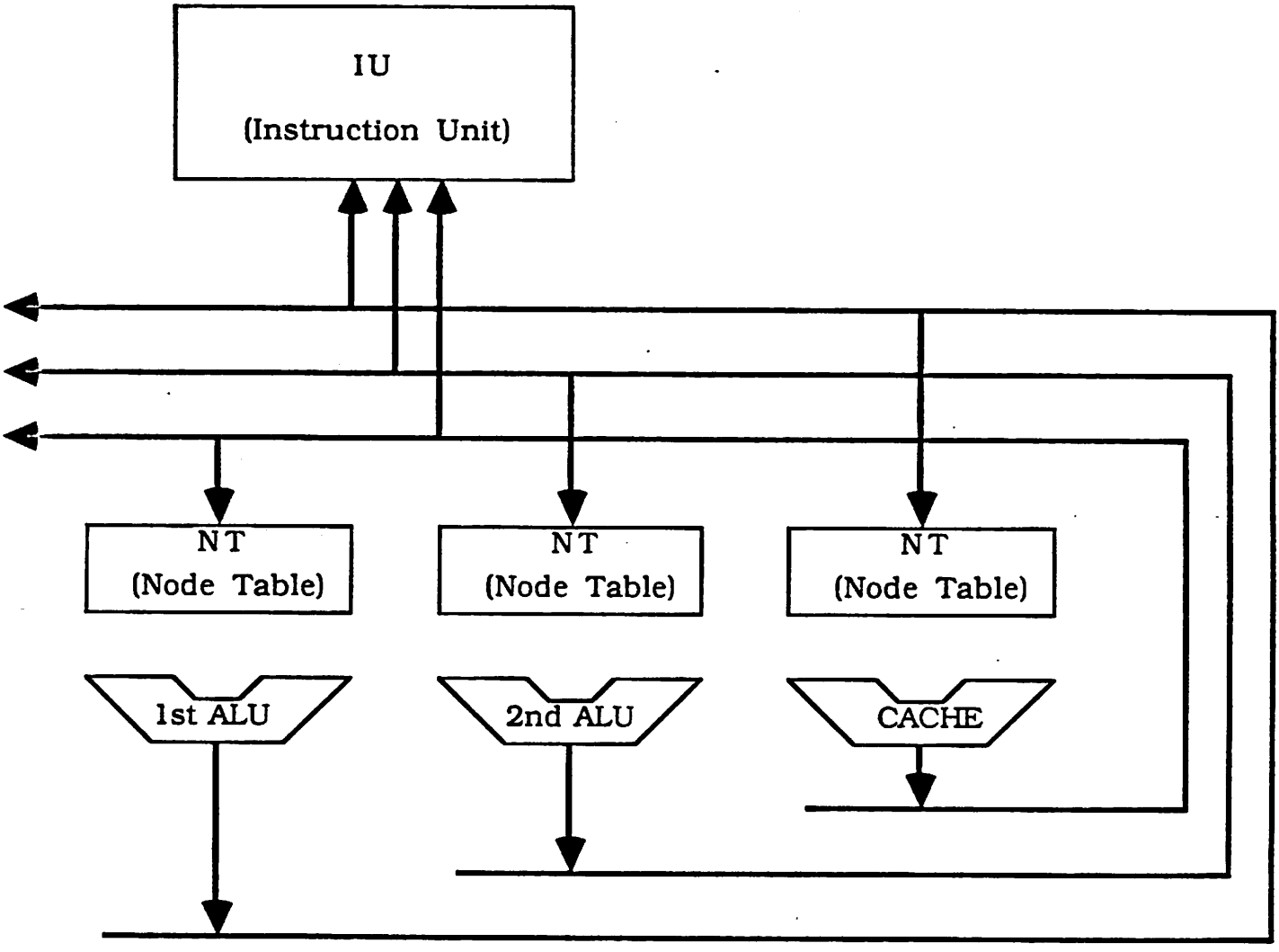


Figure 11

The B-repair and B-verify signals

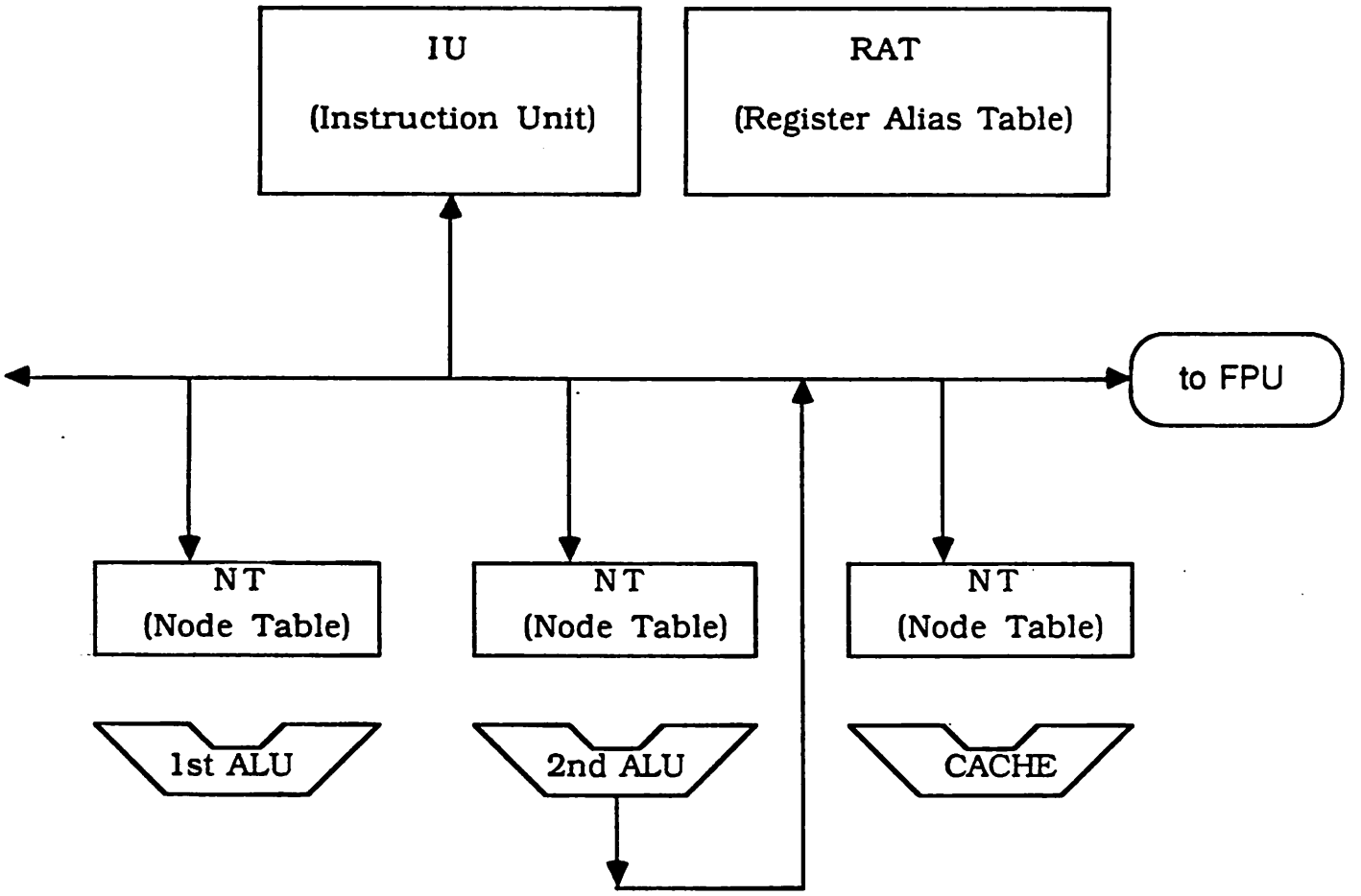


Figure 12

The Branch Pending and Jump Target Pending Signals

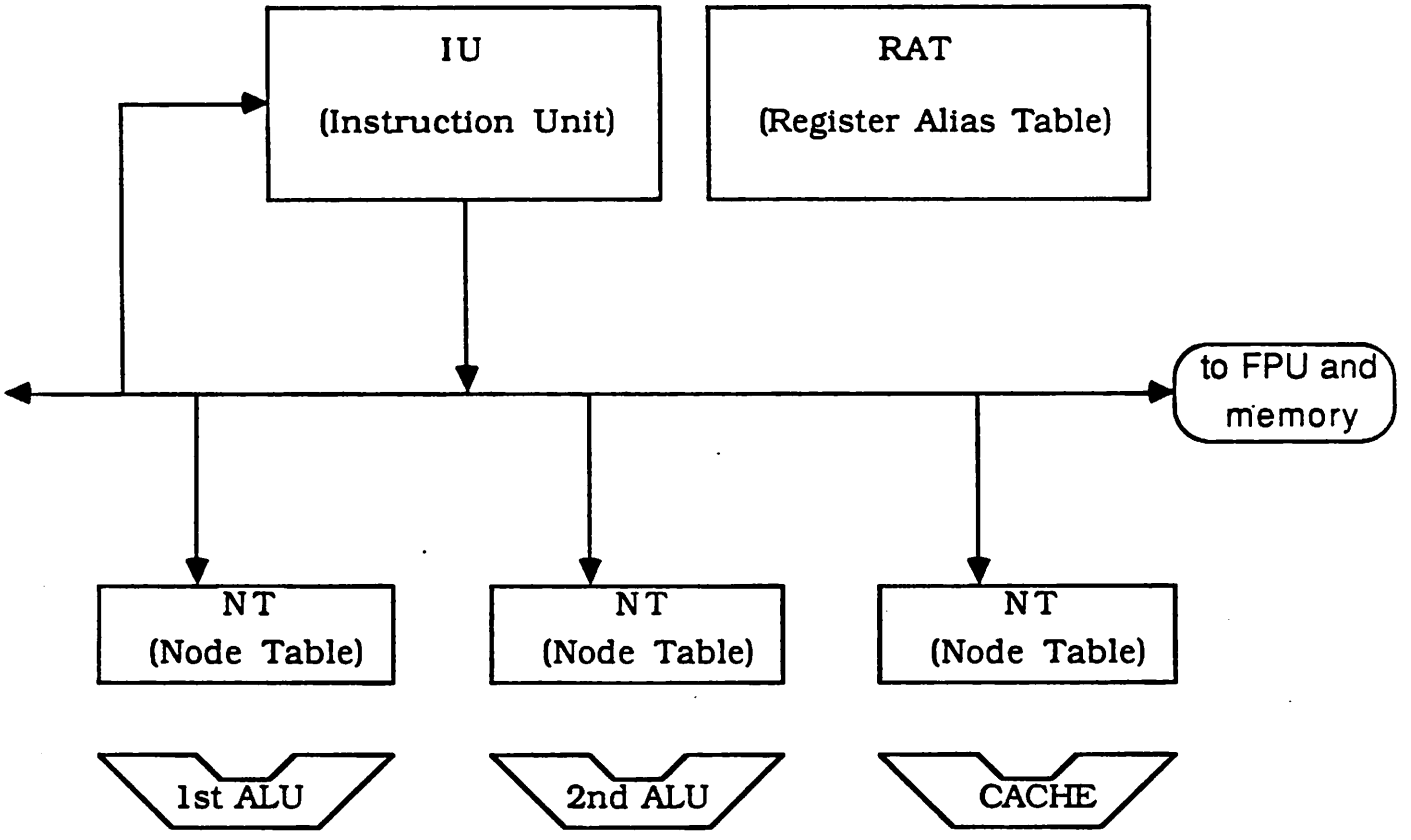


Figure 13

The Node Table Overflow Signal

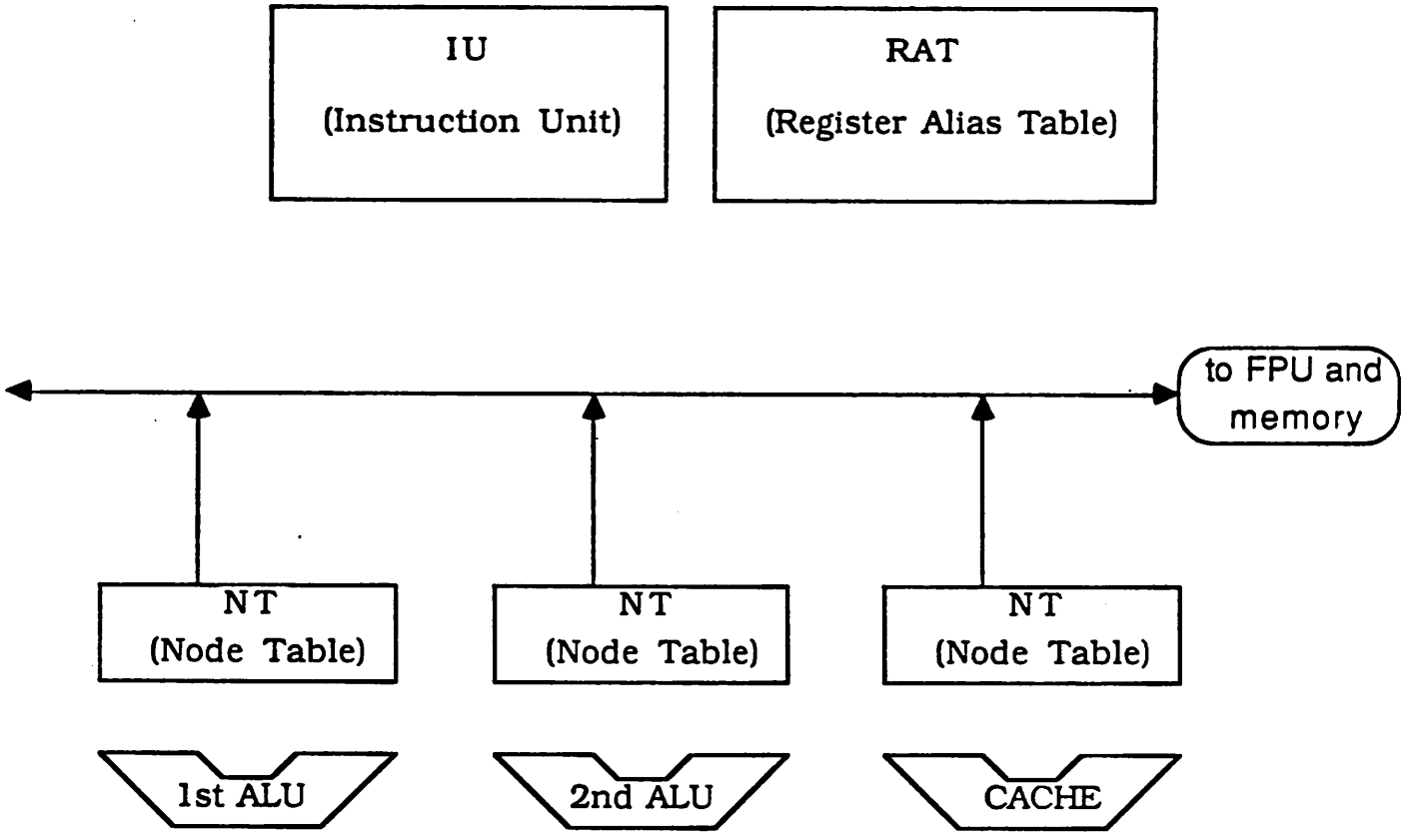


Figure 14

The Instruction Unit Stall Signal

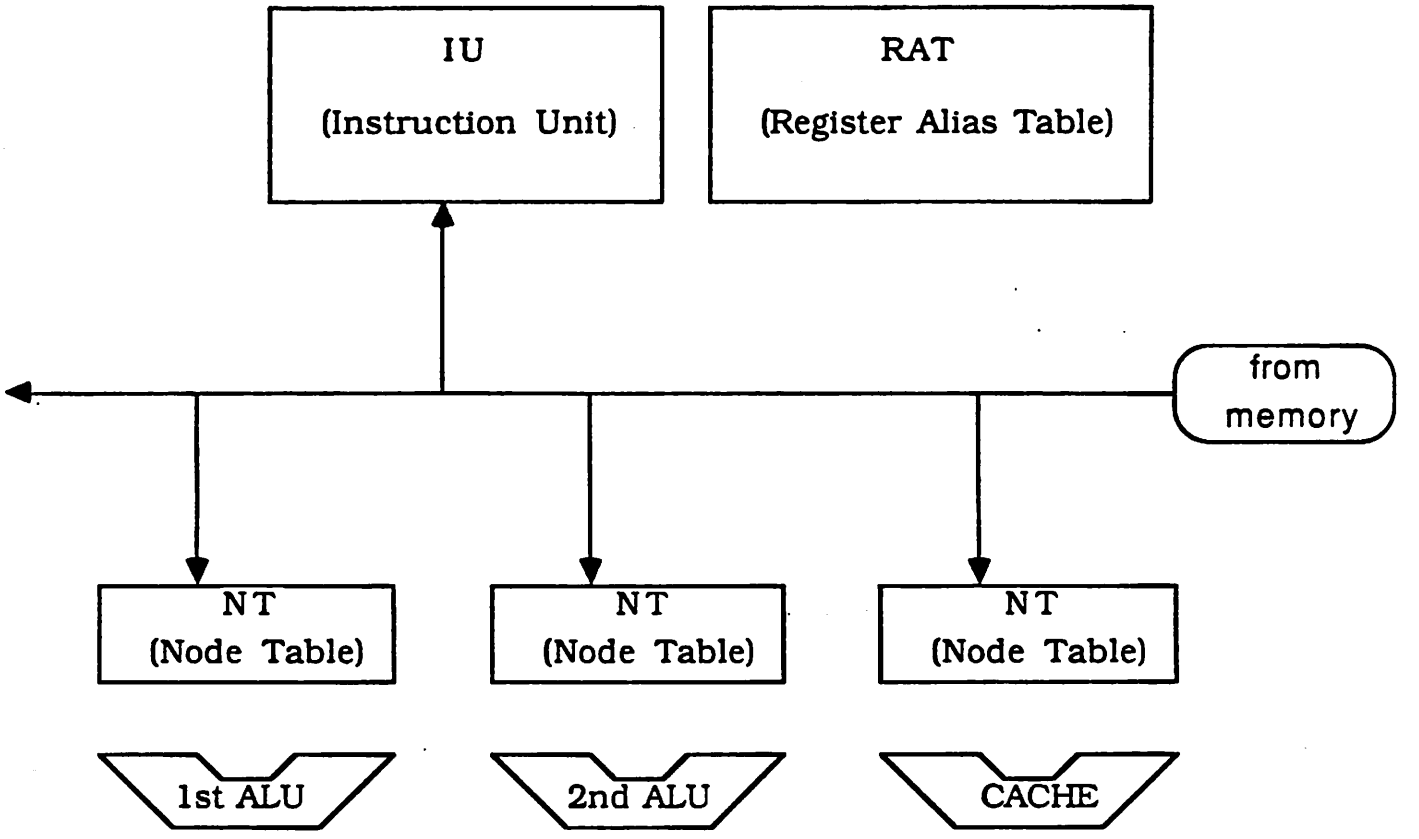


Figure 15

The New Microoperations Signal

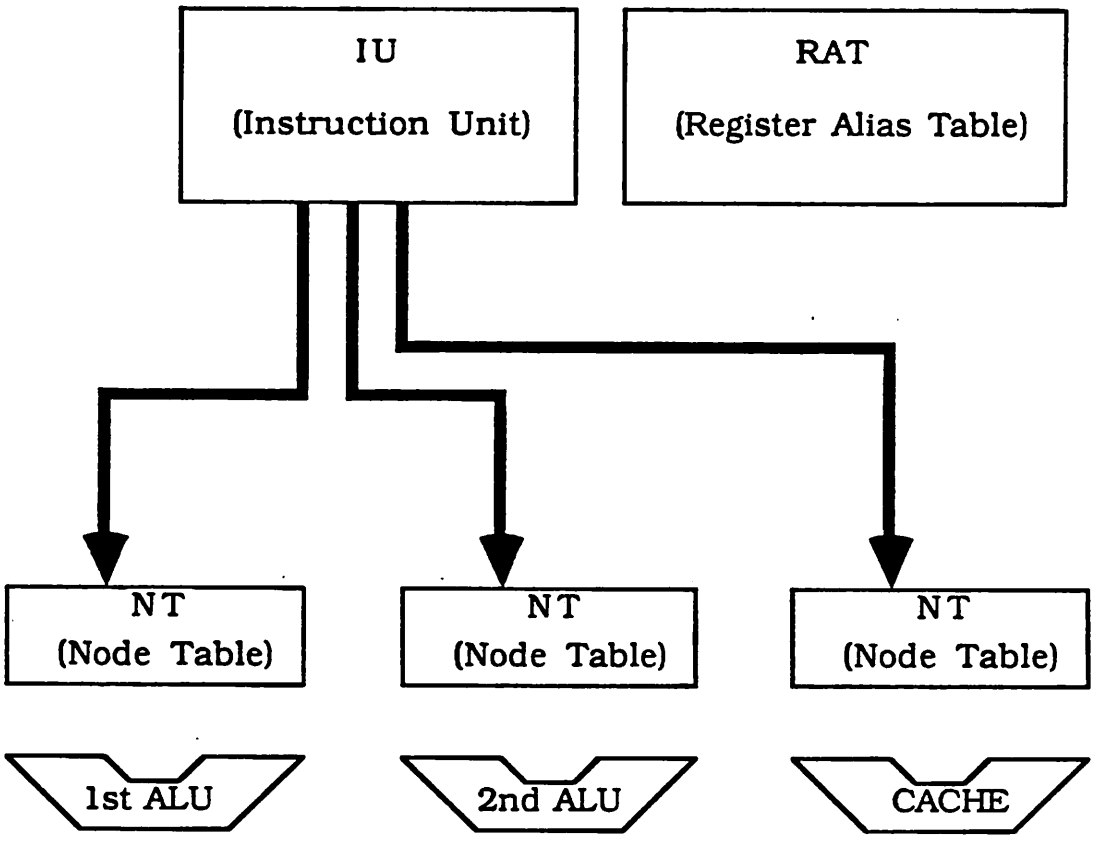


Figure 16

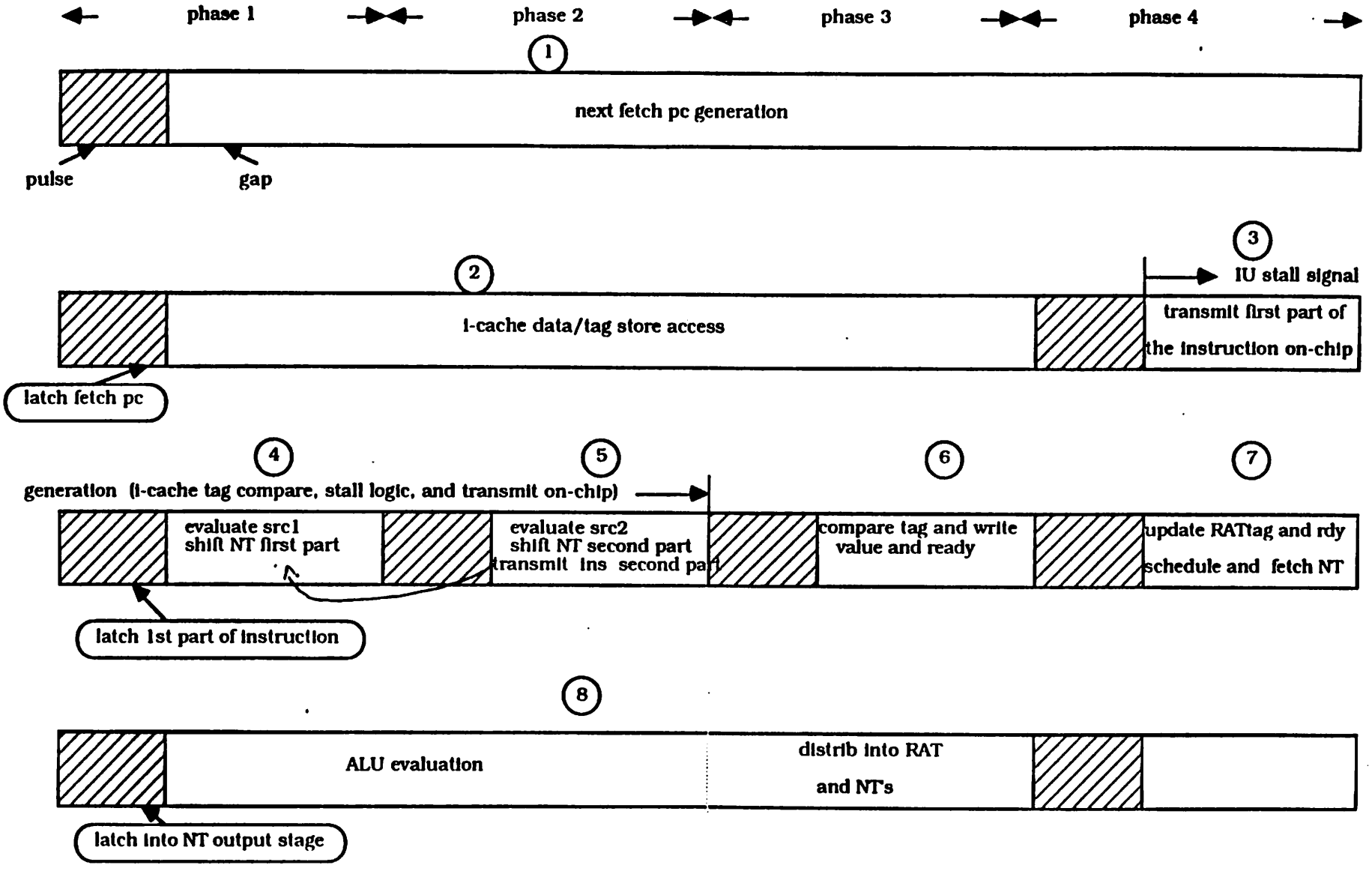

Pipeline Timing - 2nd Design
phase 1
phase 2
phase 3
phase 4
1
next fetch pc generation
pulse
gap
2
3
IU stall signal
I-cache data/tag store access
transmit first part of
the instruction on-chip
latch fetch pc
4
5
6
7
generation (I-cache tag compare, stall logic, and transmit on-chip)
evaluate src1
shift NT first part
evaluate src2
shift NT second part
transmit Ins second part
compare tag and write
value and ready
update RATtag and rdy
schedule and fetch NT
latch 1st part of instruction
8
ALU evaluation
distrib into RAT
and NTs
latch into NT output stage

Figure 17

CHAPTER 3

NT : A Smart Instruction Memory Case Study

3.

3.1. Introduction

The Node Table (NT) is a smart instruction memory that buffers microoperations awaiting data/control operands. It discards the appropriate operations during EXCEPTION and BRANCH repairs and after firing an entry for execution. Three NT's are used in the HPSm CPU chip: one for each function unit as shown in Fig. 1. The NT is a fully associative memory: all access to each entry is regulated by the contents rather than by the address of the entry. In other words, it is an "addressless" memory. Figure 2 shows the NT with its fields and its relation to the global busses and signals.

3.2. Memory Architecture

The logical format of the NT is shown in Fig. 3. Each NT entry represents a "RISC-like" microoperation (e.g. ALU ADD): a 3-bit CONTROL field, a 15-bit OUTPUT RESULT field, a 40-bit 1ST OPERAND field, and a 40-bit 2ND OPERAND field. The fields are in turn broken down as follows:

3.2.1. CONTROL Field

The CONTROL field has a VALID bit to indicate if the entry is empty, a BRANCH PENDING bit to indicate if the entry is waiting for a branch prediction to be confirmed, and a CHECKPOINT IDENTIFICATION BIT to indicate what checkpoint the operation belongs to [1].

3.2.1.1. VALID Bit

The VALID bit can be cleared (see Fig. 3) by any of the following conditions:

(1) The BRANCH REPAIR signal is "1" and the $\overline{BRANCH\_PENDING}$ signal in the same entry is "0",

(2) The EXCEPTION REPAIR signal is "1",

(3) The entry has just been fired, or

(4) The CPU is *RESET*.

3.2.1.2. BRANCH PENDING Bit

The BRANCH PENDING bit receives its value from the $\overline{BRANCH\_PENDING}$ signal in phase 2 (see Fig. 3). It is set to "1", if the BRANCH VERIFY signal is "1" and the BRANCH REPAIR signal is "0", indicating that a correct branch prediction has just been verified. It is not accessed through any bit line. The BRANCH PENDING bit participates in the scheduling operation and in clearing the VALID bit during BRANCH REPAIR.

3.2.2. OUTPUT RESULT Field

The OUTPUT RESULT field has a 5-bit OPCODE field to indicate the operation to be performed on the entry, a 5-bit OUTPUT REGISTER # field to indicate the destination RAT register for the result, and a 5-bit OUTPUT TAG field to indicate the tag for the result.

3.2.3. 1ST OPERAND and 2ND OPERAND Fields

Each operand field has a 7-bit TAG CAM field to indicate the tag of the operand, 1 RDY bit to indicate if the operand is valid (or if the entry is waiting for this operand to arrive as the result of an FU evaluation) and a 32-bit VALUE field for the value of the operand.

Since each entry has 98 bits, it infeasible to implement the NT exactly as depicted by the logical format. Rather, the 2ND OPERAND field is folded underneath the 1ST OPERAND field as shown in Fig. 4. The external SCHEDULER control uses the VALID bit and the AGE field (shows "age" of entry) to determine the oldest firable entry and the entry to be written into. The FIRE ROW DRIVERs use the control result from the SCHEDULER to drive the appropriate FIRE WL high. The DISTRIBUTION ROW DRIVERs buffer the OPERAND TAG match lines to drive the OPERAND DISTRIBUTION WLs during distribution.

3.3. Basic Operations

The timing sequence is illustrated in Fig. 5. All signals are aligned to the 4-phase clocks of the HPSm CPU chip as shown in the figure. In the sequel, *phase i* is defined as the time segment from the rising edge of ϕ_i to the rising edge of ϕ_{i+1}. In phase 1 (phase 2), the 1ST OPERAND (2ND OPERAND) is fetched from the RAT and written into the first available entry. Also, in phase 2, the OPCODE, OUTPUT RESULT TAG and OUTPUT REGISTER # is fetched from the IU and written into the first available entry. In the third phase, all entries in the NT awaiting their operands associatively receive values from the distribution bus if the stored tags match the distributed tags. Finally, in phase 4, the SCHEDULER FIREs the oldest firable entry. The VALID bit of the fired entry is then set to "0" to scratch this entry from the NT.

3.4. Priority Encoding Scheme

The NT needs a priority encoding scheme so that it can choose the oldest eligible instruction for execution in the case where there is more than one executable instruction. Designing an efficient scheme is the most intractable part of the NT design. The difficulty lies in establishing the oldest firable entry

and performing garbage collection after the oldest entry has been fired so that the fired entry is not refired.

Two broad classes of priority schemes can be employed. The first one uses "smarter" data cells by implementing some of the priority control in each cell while the other implements all the control outside the memory core. The SHIFT-REGISTER priority encoding scheme is an example of a scheme in the first class. In this case, the entries are ordered by their positions in the NT. The SHIFT operation is used to shift the latest entry into the first entry, while the other entries are shifted into the empty entry positions below them. This SHIFT operation not only establishes the oldest entry but also performs garbage collection. This SHIFT-REGISTER scheme has the typical disadvantage of shift register memories [2] : the SHIFT signal is highly capacitive since it goes to all the cells. Driving the SHIFT signal would lead to high dynamic power and peak current. These problems are compounded by the fact that 3 NT's are needed in the HPSm CPU chip. The SHIFT-REGISTER approach was rejected in favor of a POINTER-MECHANISM scheme to be discussed next. The latter scheme is an example of implementing the priority control outside the core. It therefore moves all the complexity from the core to the periphery where it can be easily handled with available CAD tools.

The POINTER-MECHANISM method reverses the process of the SHIFT-REGISTER approach. It has 2 pointers at any given time : one that points to the entry for insertion of the next instruction in phases 1 and 2, and another one that points to the entry to be fired. Instead of shifting the entire contents of the NT, it justs shifts these pointers. Any instruction can be inserted into any entry and any entry can be fired from any position. Therefore, this scheme maintains an AGE information for each entry that indicates the order in which the data was written into the NT. (The SHIFT-REGISTER approach, in contrast, does not need an explicit AGE information since the age of each entry is implied by its physical location in the memory.)

3.4.1. The SCHEDULER Finite State Machine

Figure 6 shows all the possible occupation states of the NT. The SCHEDULER is a finite state machine (FSM) as shown in Figure 7. It uses the AGE information along with the VALID BIT, and READY BITs of each entry to determine the pointer positions. AGE in Figure 7 is an UP/DOWN RESETTABLE counter. The VALID BIT indicates what entry is occupied while INSTR_RDY is the READY bit that informs the SCHEDULER if the instruction is READY (i.e. the READY BITs of the 1st and 2nd operands are SET). The FIRE_WL and INSERT_WL SCHEDULER outputs in Figure 7 are the pointers. They are qualified by the appropriate clocks to drive the NT FIRE and INSERT word lines. For each AGE counter, the SCHEDULER generates the UP, DN(DOWN), and RESET signals. The state transition diagrams that describe the behavior of the SCHEDULER FSM would be discussed next.

3.4.1.1. INSERT State Transition Diagram

In phases 1 and 2 after a new instruction has been written (inserted) into the NT, the state of the NT can change in one of several ways. Depending on the initial configuration of the NT, the state makes a transition to another configuration as shown in Figure 8a. Figure 8b focuses on one entry, ENTRY.i. Since the NT for the HPSm has 4 entries, there are only 5 possible states - 1 "EMPTY" state plus 4 possible positions for ENTRY.i.

A CPU *RESET* puts all entries at STATE 0, where they are all empty. If insertion is made into another ENTRY.j, ENTRY.i is forced to remain at STATE 0 by RESETting the Age_COUNTER for ENTRY.i. If an insertion is made into ENTRY.i, the Age_COUNTER of ENTRY.i is counted UP to increase its AGE and advance its POSITION. If ENTRY.i is at STATEs 1, 2, or 3, when an insertion is made into another ENTRY.j, the Age_COUNTER of ENTRY.i is also counted UP. Therefore, an occupied entry advances in age as long as insertion is taking place in other entries. In STATE 4, an insertion is not possible since the NODE TABLE OVERFLOW signal (see Chapter 2) generated by the NT prevents the HPSm IU from fetching any new instructions.

3.4.1.2. FIRE State Transition Diagram

Figure 9a shows the transitions between the possible NT configurations as entries are fired from the memory. Figure 9b depicts how the state of a particular entry, ENTRY.i, changes as NT entries are fired. The SCHEDULER uses the VALID BIT of "0" to make sure that ENTRY.i cannot be fired if it is in STATE 0. Whenever ENTRY.i is fired, it is automatically RESET to discard it from the NT. If another entry, ENTRY.j is fired, ENTRY.i's AGE is counted DOWN if ENTRY.i was older than ENTRY.j before it was fired; otherwise ENTRY.i's AGE is left unchanged.

3.4.1.3. Overall State Transition Diagram

The overall picture that combines both the effects of firing and insertion on a particular entry, ENTRY.i, is shown in Figure 10. For insertion, the main point is that all occupied entries increase in AGE as entries are inserted into the NT. Therefore, the ordering of the entries by the AGE is updated correctly after each insertion. All fired entries are reset no matter their AGE. The garbage collection after any entry is fired is accomplished by simply counting DOWN the AGE of each non-fired entry whose AGE is higher than that of the fired entry. The non-fired entry whose AGE is lower than that of the fired entry is left unaltered. In this way, the ordering of the occupied entries in the NT is maintained after an entry has been fired.

Updating the AGE information after insertion and firing in the way described in the last paragraph gives a clean way of doing garbage collection after firing the oldest eligible instruction. Resetting the AGE counter or just counting the AGE counter UP or DOWN avoids the complexity of moving all the pieces of data in the NT to do compaction as in the case of the SHIFT REGISTER method.

3.4.2. SCHEDULER Implementation

The shift of the priority scheme complexity from the core to the periphery increases the complexity of the SCHEDULER in a combinatorial manner. This is so because any entry can be a candidate for insertion or firing. It is clear from Figure 6 that the number of possibilities "explodes" combinatorially as N, the number of entries, increases. Fortunately, N=4 for the HPSm NT. The most recent Berkeley CAD tools (MISII, WOLFE, VULCAN, and OCTTOCIF) were employed to contain the SCHEDULER complexity. It is inefficient to implement the SCHEDULER with a PLA because of its combinatorial nature. (For instance, it is inefficient to implement an ALU with a PLA because flattening an ALU into just a 2-level logic is inefficient.) Therefore, the SCHEDULER is implemented with multi-level logic using standard cells. The CAD tools mentioned above allow a lot of flexibility, quick turnaround and extensive verification of the SCHEDULER implementation. The input description of the SCHEDULER that is fed into the CAD tools is in Appendix I.

In addition, a personal CAD routine was written to generate the vectors corresponding to all the possible configurations. These vectors were used to verify the correctness of the SCHEDULER implementation.

3.5. Circuits

3.5.1. Cells

3.5.1.1. Tag Cell

The tag cell shown in Figure 11 is a single port write-only (WOM) CAM cell. The SRAM cell at the top is used to store the data while the 3 comparators below are used for the associative operation. Tags for the 1st (2nd) operand is written into the cell in phase 1 (phase 2) with the INSERT.WL word line. In ϕ_3, the match lines are precharged while the bit lines are discharged. In phase 3, while INSERT.WL is held low, 3 tags that accompany 3 results from the HPSm FU's are broadcast to the tag cell by driving the bit lines. The match line corresponding to the external tag that matches the stored tag is left high. The XOR comparator discharges any match line associated with a tag mismatch. Because the tag is never read out, the highly capacitive (93fF) internal node (1) that connects to all the comparators does not pose a problem.

3.5.1.2. Value Cells

The value cells for the 1st and 2nd operands are shown in Figure 12. The value cell is an SRAM cell that is written into with INSERT.WL and read from with FIRE.WL. The distribution word lines (DIST.WL1, DIST.WL2, DIST.WL3) are used to associatively write results from the HPSm FU's. They are connected to the match lines of the tag cell in Figure 11 with the DISTRIBUTION ROW DRIVERs in Figure 4. The difference in the FIRE bit lines is the only difference between the 1st operand and the 2nd operand value cells. This difference is necessary since the two operands must be fetched for execution (in the ALU for instance) at the same time. BL2 and BL3 are used as FIRE bit lines to reduce the capacitive load on BL1. The function unit result bus (FU.BUS) is sent through the value cells to avoid the complexity of routing the bus around the NT.

3.5.1.3. RDYBIT Cells

The schematics for the cells for the 1st and 2nd operand READY BITs are in Figure 13. The RDYBIT cell is similar to the value cell but for two differences. First, since it is not needed for execution, it is never fired. Second, its value is sent to the SCHEDULER to indicate if the instruction is READY. To reduce the capacitive loading on the internal node, the buffer, B2, is used to isolate the internal node of the cell from the signal (RDYBIT.sigLine) that is sent to the SCHEDULER. B2 is placed on the "negative" side of the cell since it is an inverter. Another buffer, B1, is used on the "positive" side to reduce any voltage offset in the cell introduced by B2. The use of the data of the RDYBIT cell as control signal reflects the fully associative feature of the NT. It can be seen from looking at Figure 13 that the inclusion of logic in the data cells (by including B1 and B2) make smart memory data cells larger than conventional data cells.

3.5.1.4. BRANCH PENDING Cell

The BRANCH PENDING cell in Figure 14 is another example of a cell whose data is used as control information. In phase 2, the BRANCH PENDING bit is written into the cell using INSERT.WL to indicate if the corresponding instruction is waiting for a BRANCH PREDICTION to be confirmed. This information is sent to the SCHEDULER using the buffer, B1. When the BRANCH is confirmed, (i.e. BRANCH_VERIFIED is "1" and BRANCH_REPAIR is "0") the P1 PMOS and N1 NMOS transistors are used to SET the BRANCH PENDING bit.

3.5.1.5. OPCODE Cell

The OPCODE cell, in Figure 15, is a 1-port SRAM cell that is written into with INSERT.WL and read from with FIRE.WL. The CID, OUTPUT REGISTER #, and OUTPUT TAG fields employ the OPCODE cell.

3.5.2. Detailed Operations

The timing chart of the NT signals is in Figure 16. The lumped SPICE model of the NT is shown in Figs. 17a-17c. All the entries are lumped into one entry while all the columns in a field are lumped into one column.

3.5.2.1. SCHEDULER and PENDING BIT Column

Figure 17a shows the schematic for the SCHEDULER and PENDING BIT Column. The SCHEDULER is a finite state machine with the AGE counters and the NT core supplying its inputs as discussed in section 3.4. In the WL QUALIFIERs, the toFIRE.WLs and toINSERT.WLs are qualified with the EN_FIRE (ENABLE FIRE), EN_INSERT.1st (ENABLE INSERT 1st) and EN_INSERT.2nd (ENABLE INSERT 2nd) signals to drive the FIRE.WLs and INSERT.WLs at the appropriate times. Only one PENDING BIT is needed for each entry; this bit is aligned with the 2nd operand row of the NT as shown in Figure 17a. Of all the word lines only the INSERT.WL_2 is used by the PENDING BIT column. The other word lines are sent across the column to the other fields.

The buffers for writing the BRANCH PENDING signal (see Chapter 2) is implemented in the WRITE BUFFERs. The WRITE BUFFERs also contain the logic for combining the BRANCH VERIFY and BRANCH REPAIR signals (see Chapter 2). The INSTR.RDY Logic block is implemented in the empty slot above the PENDING BIT cell. It generates the INSTR.RDY signal by combining the PENDING BIT and the information about the readiness of the 1st and 2nd operands, to indicate if the instruction is executable. RDYBIT Logic is implemented besides the PENDING BIT cell. It uses a NOR gate with the 1st, 2nd, and 3rd match lines (ML1_2, ML2_2, ML3_2) and the 2nd operand RDYBIT (RDYBIT.sigLine_2) to determine the readiness of the 2nd operand. Using the match lines rapidly reflects the most recent change made to the RDYBIT (during result distribution) in the INSTR.RDY Logic block. (The RDYBIT Logic for the 1st operand is part of INSTR.RDY Logic.)

3.5.2.2. TAG FIELD and RDYBIT Column

The lumped model of the TAG FIELD, the DISTRIBUTION ROW DRIVERs and the RDYBIT column is in Figure 17b. The WRITE BUFFERs drive the bit lines during insertion and distribution. The match lines from the TAG FIELD are qualified in phase 3 with the ML2WL in the row drivers to drive the distribution word lines (DIST.WL_1 and DIST.WL_2).

3.5.2.3. VALUE, OPCODE and MISCELLANEOUS FIELDs

Figure 17c is the schematic of the VALUE, OPCODE and the MISCELLANEOUS (CID, OUTPUT REGISTER #, and OUTPUT RESULT) FIELDs. The WRITE BUFFERs of the fields drive the bit lines in phases 1, 2 and 3. The WRITE BUFFERs of the VALUE and MISCELLANEOUS fields also contain dynamic latches for storing the function unit result and for driving the distribution busses. The VALUE and OPCODE fields have READ BUFFERs that send the operands and opcode to the function unit for execution. Since the cells are SRAM cells and the bit line capacitance is not excessive (0.55pF maximum), no sense amplifiers are used.

3.6. Simulation Results

3.6.1. NT Layout

The NT layout is in Fig. 18. Its size is 1.32mm x 5.54mm, it has 9,527 transistors and it dissipates 0.25W running at 10Mhz.

3.6.2. SPICE Results

The SPICE results for the TAG and VALUE fields of the lumped model discussed above are shown in Figs. 19a-19d for two scenarios. The first scenario is when there is a tag match during the dis-

tribution of results while the second scenario is when there is a tag mismatch. The results reflect that the implemented NT satisfies the timing requirements imposed by the HPSm CPU.

3.7. Conclusions

NT, a smart instruction memory, was described in the above sections. It combines a finite state machine with data cells for storing instructions (operands, opcode and control information). The above discussion reflects the fully associative characteristic of the NT. The use of the POINTER-MECHANISM method for implementing the priority encoding scheme trivializes the garbage collection function and allows the use of simpler data cells. RAT, a smart data memory, is discussed in the next chapter.

References

[1] Wen-mei Hwu, "HPSm: Exploiting Concurrency to Achieve High Performance in a Single-chip Microarchitecture" *Ph.D. Dissertation, Computer Science Division, EECS Dept.*, University of California, Berkeley, CA. 94720, 1987.

[2] D.A. Hodges and H.G. Jackson, "ANALYSIS AND DESIGN OF DIGITAL INTEGRATED CIRCUITS" *McGraw-Hill, 1983.*

Data-Path Current Design

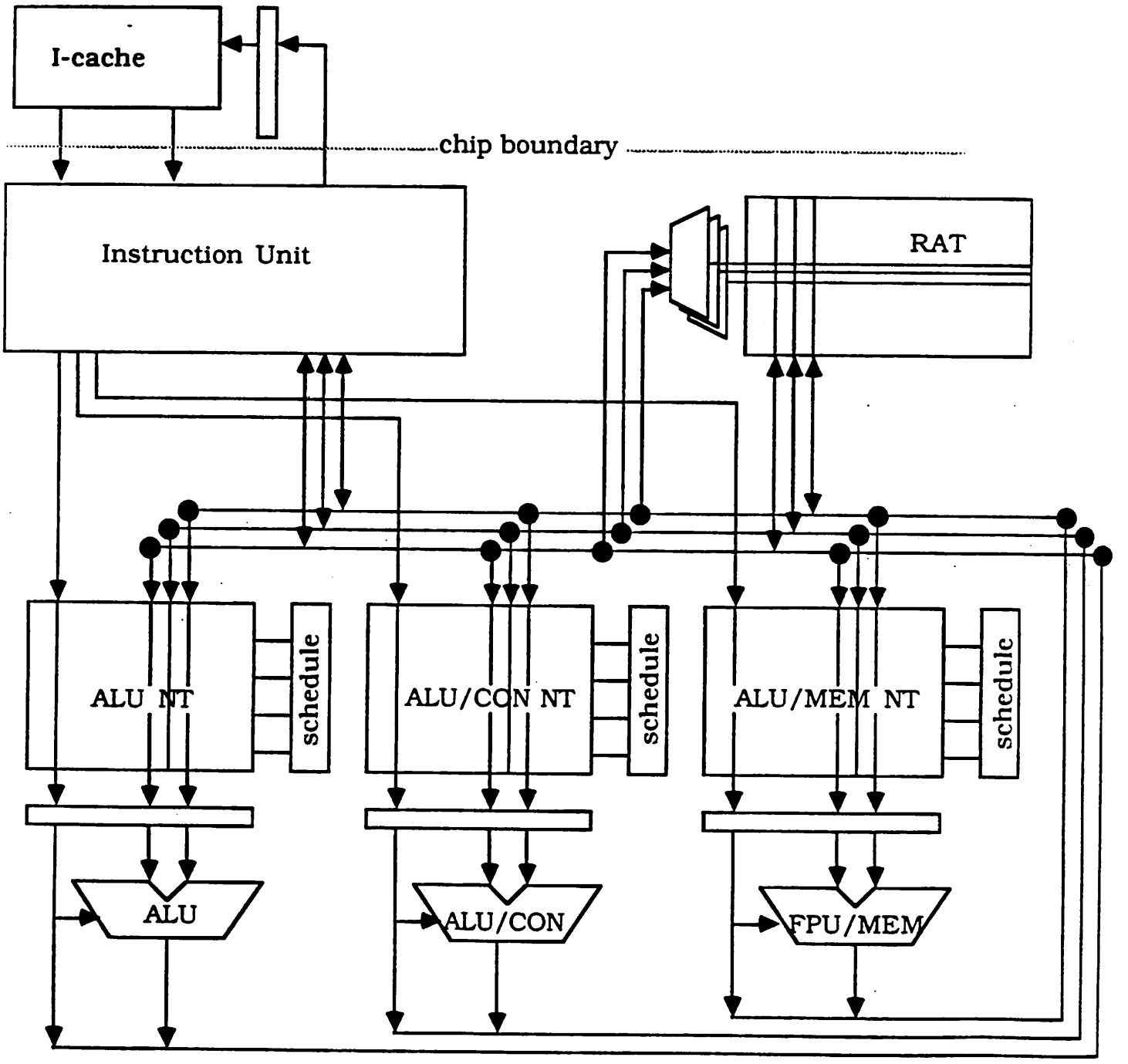
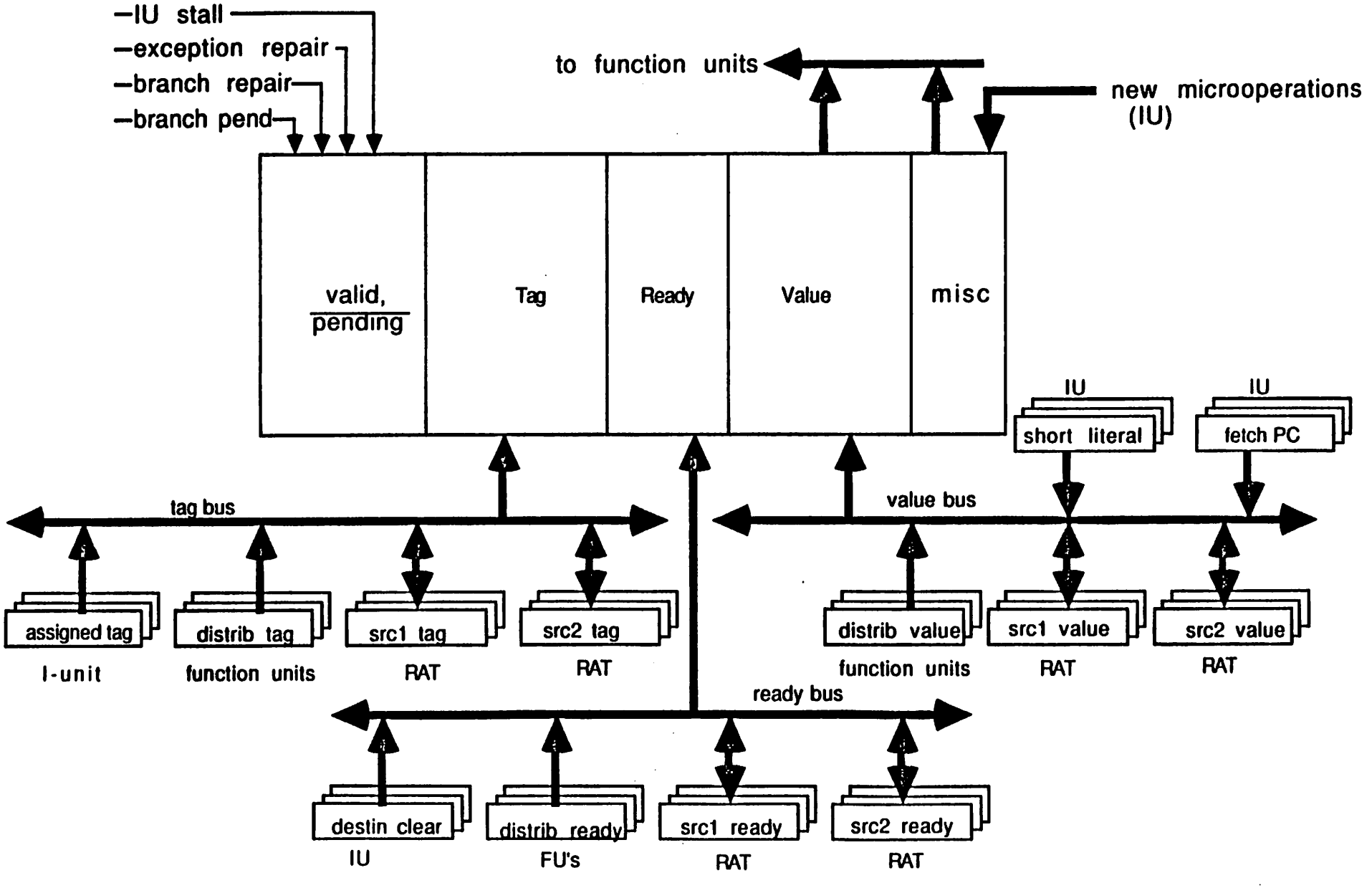


Figure 1

Fig. 2 : NT with Global Busses and Signals



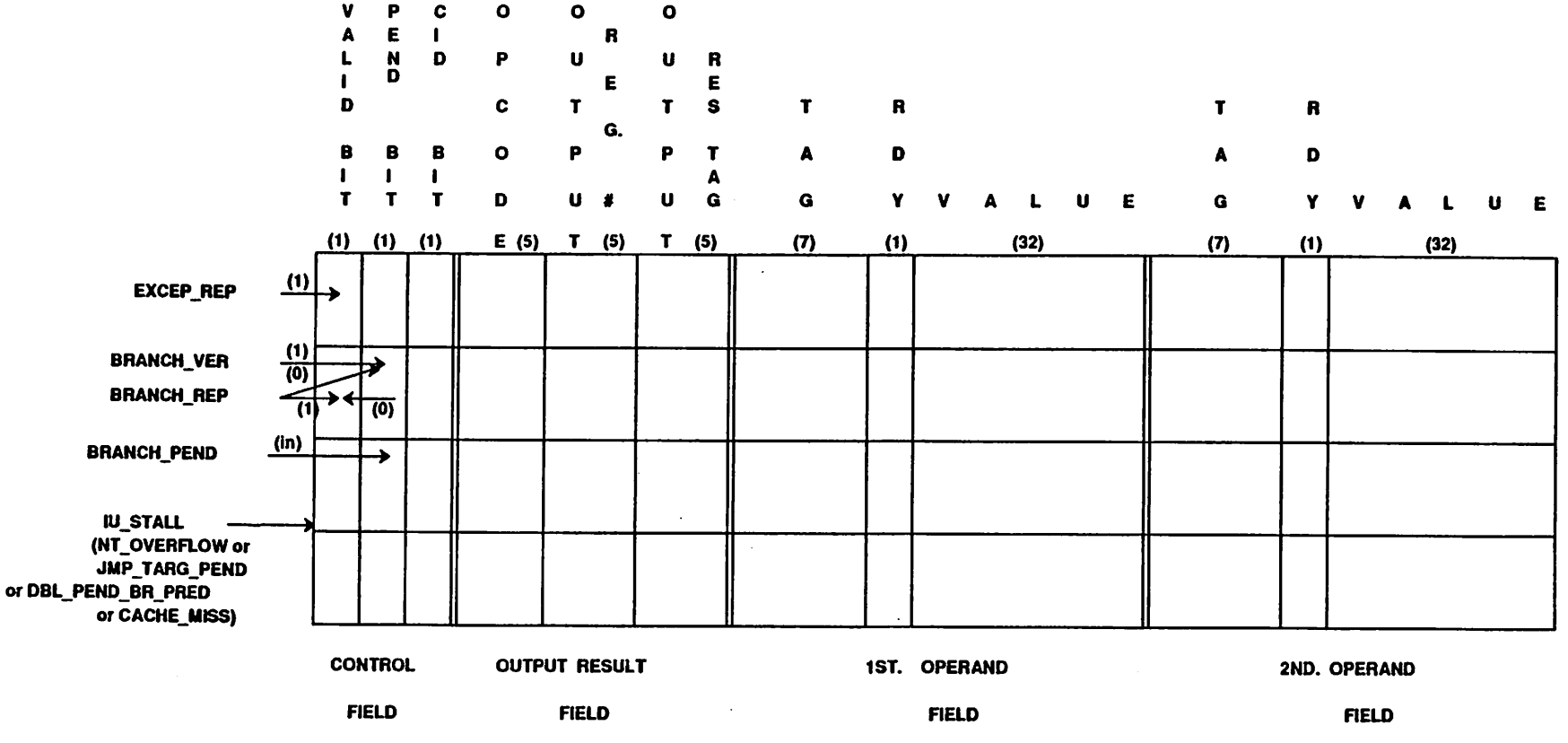


Fig. 3: NT Logical Format

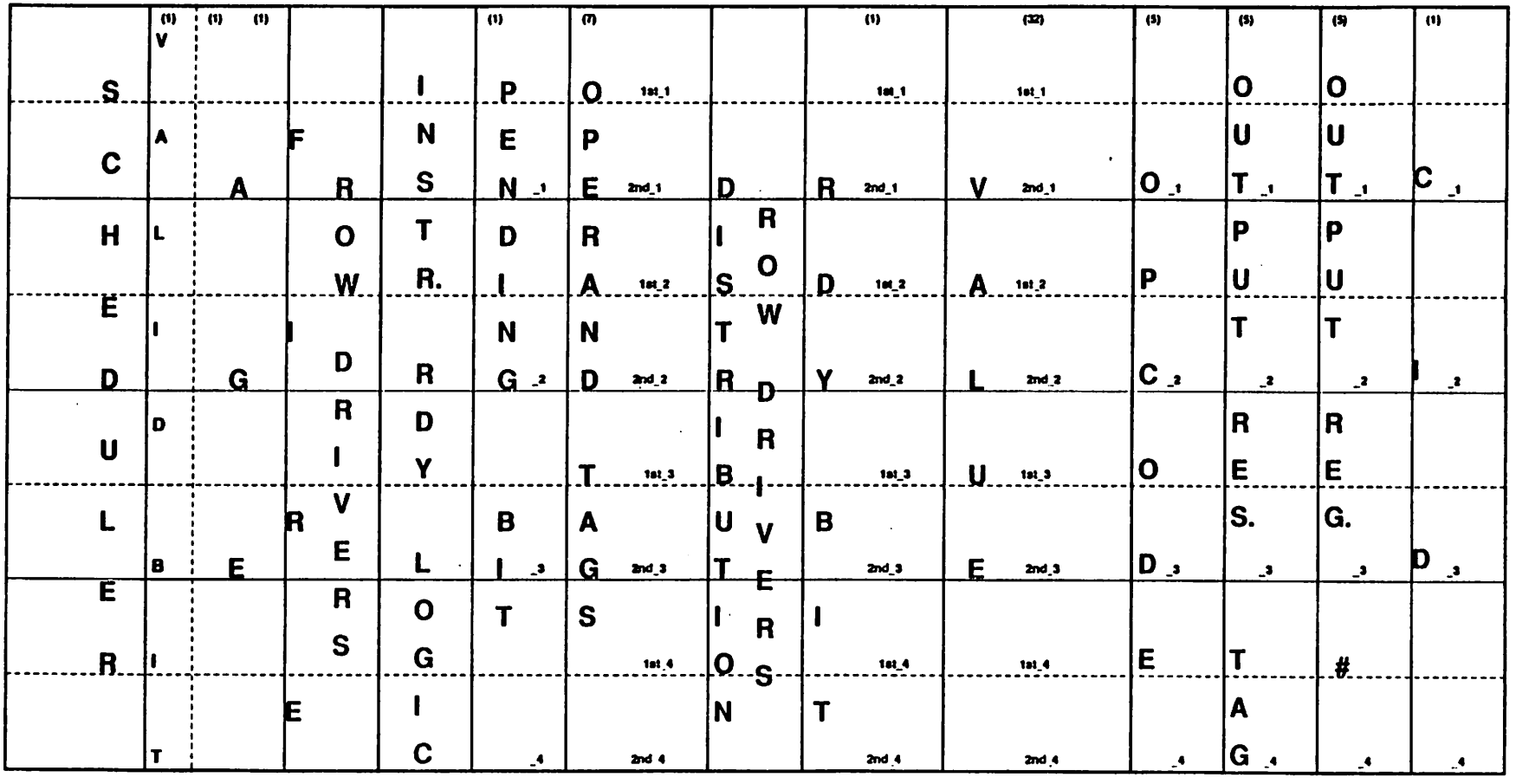


Fig. 4: NT Physical Format

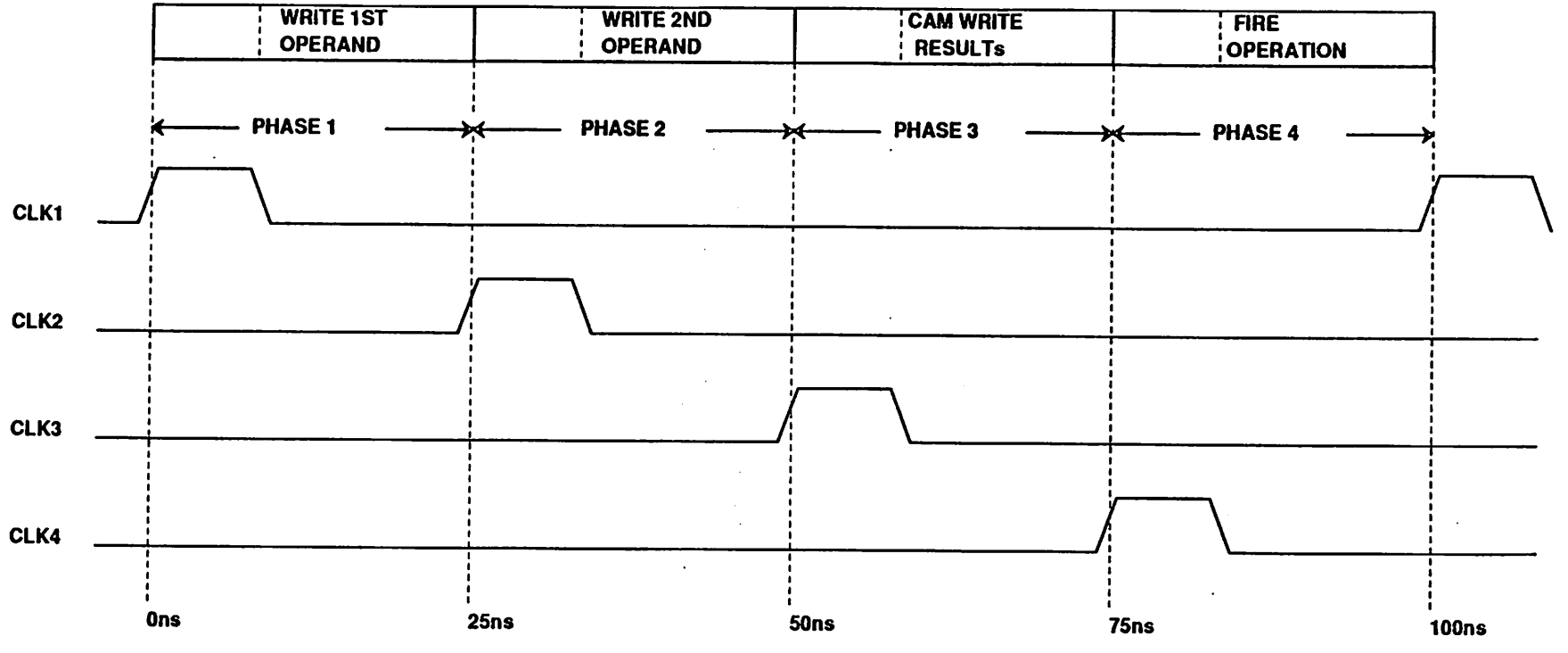


Fig. 5: NT Timing Sequence

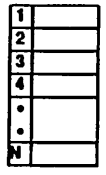
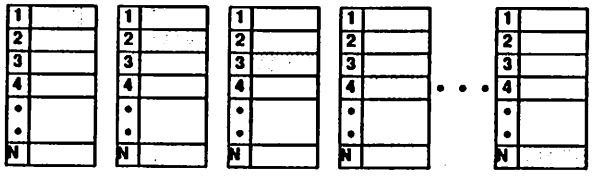
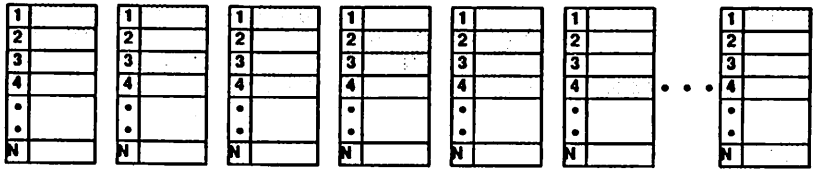
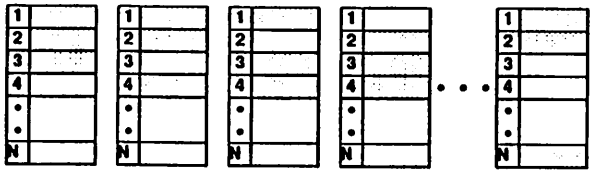
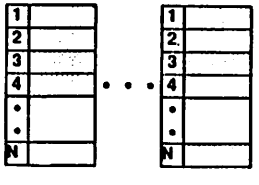
# OCCUPIED ENTRIES	FORMULA	ALL POSSIBLE CONFIGURATIONS
0	$_{N}C_{0} = 1$ (e.g. 1 for N=4)	
1	$_{N}C_{1} = N$ (e.g. 4 for N=4)	
2	$_{N}C_{2}$ (e.g. 6 for N=4)	
3	$_{N}C_{3}$ (e.g. 4 for N=4)	
4	$_{N}C_{4}$ (e.g. 1 for N=4)	

Fig. 6. NT Possible Configurations

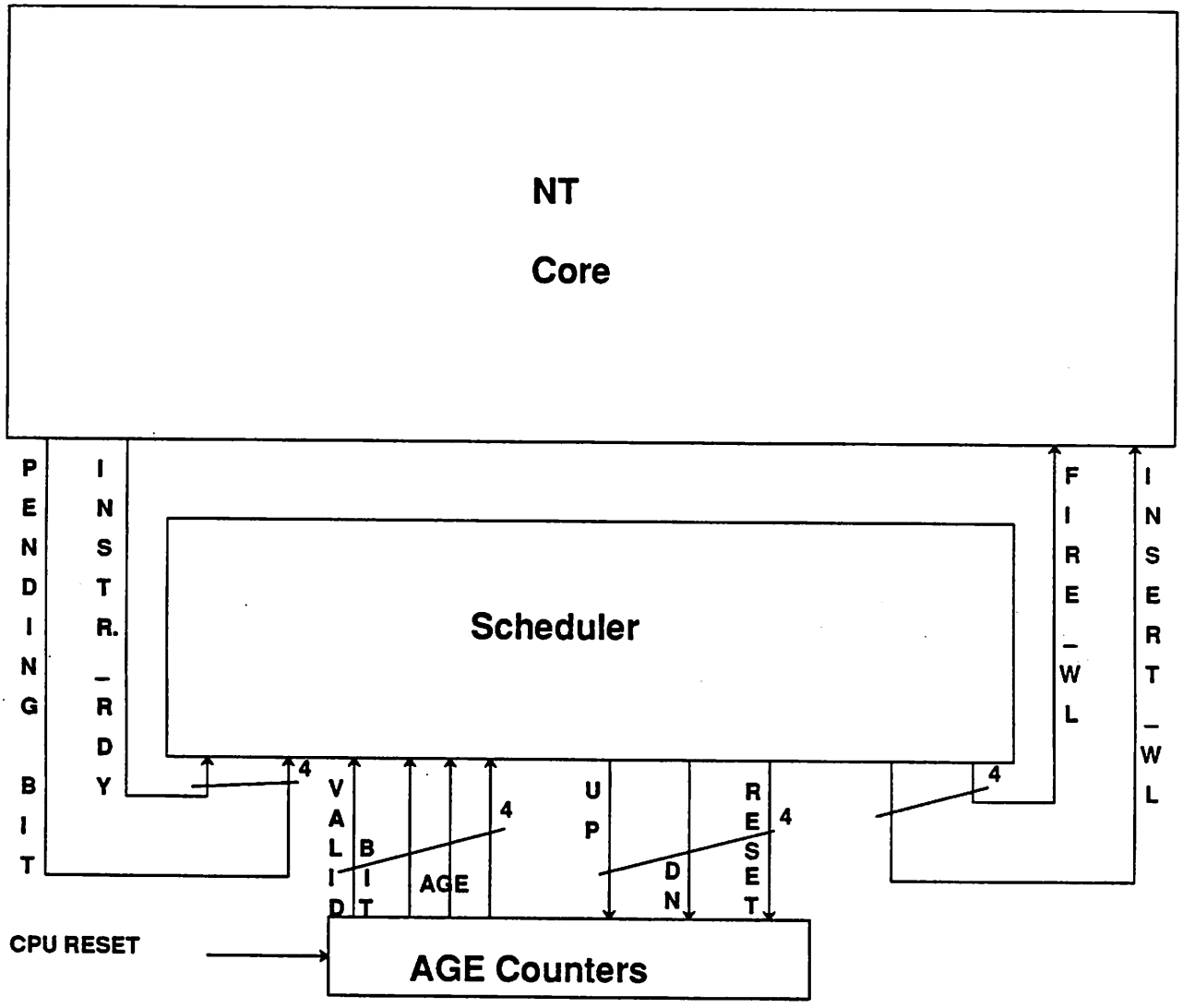


Fig. 7. SCHEDULER Finite State Machine

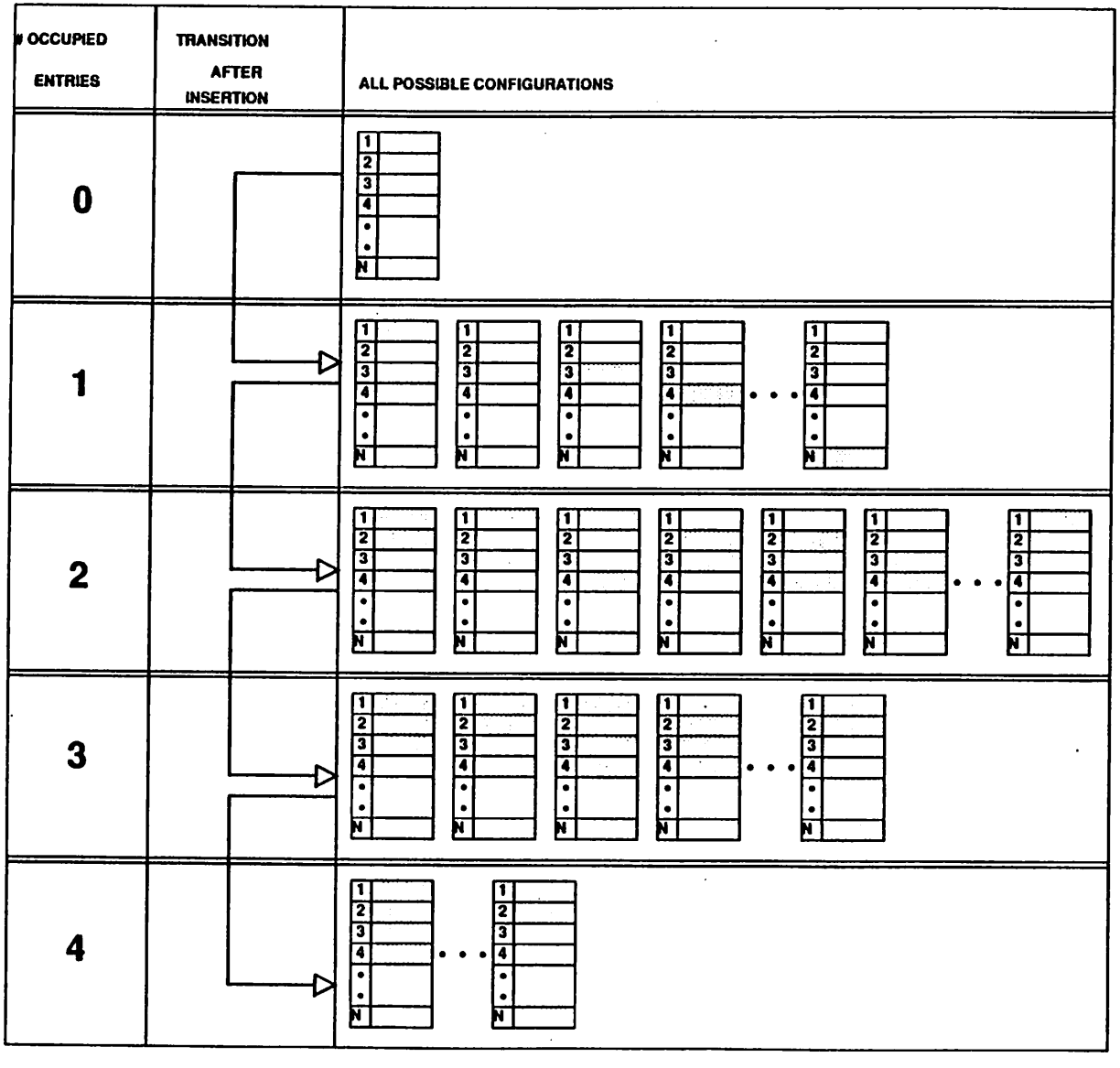


Fig. 8a. NT Configurations After INSERTion

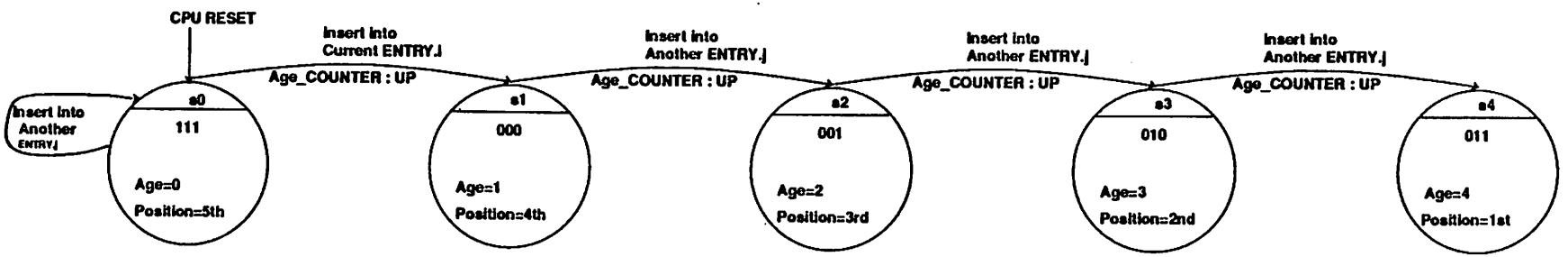


Fig. 8b. ENTRY.i AGE State Transition Diagram for INSERTion

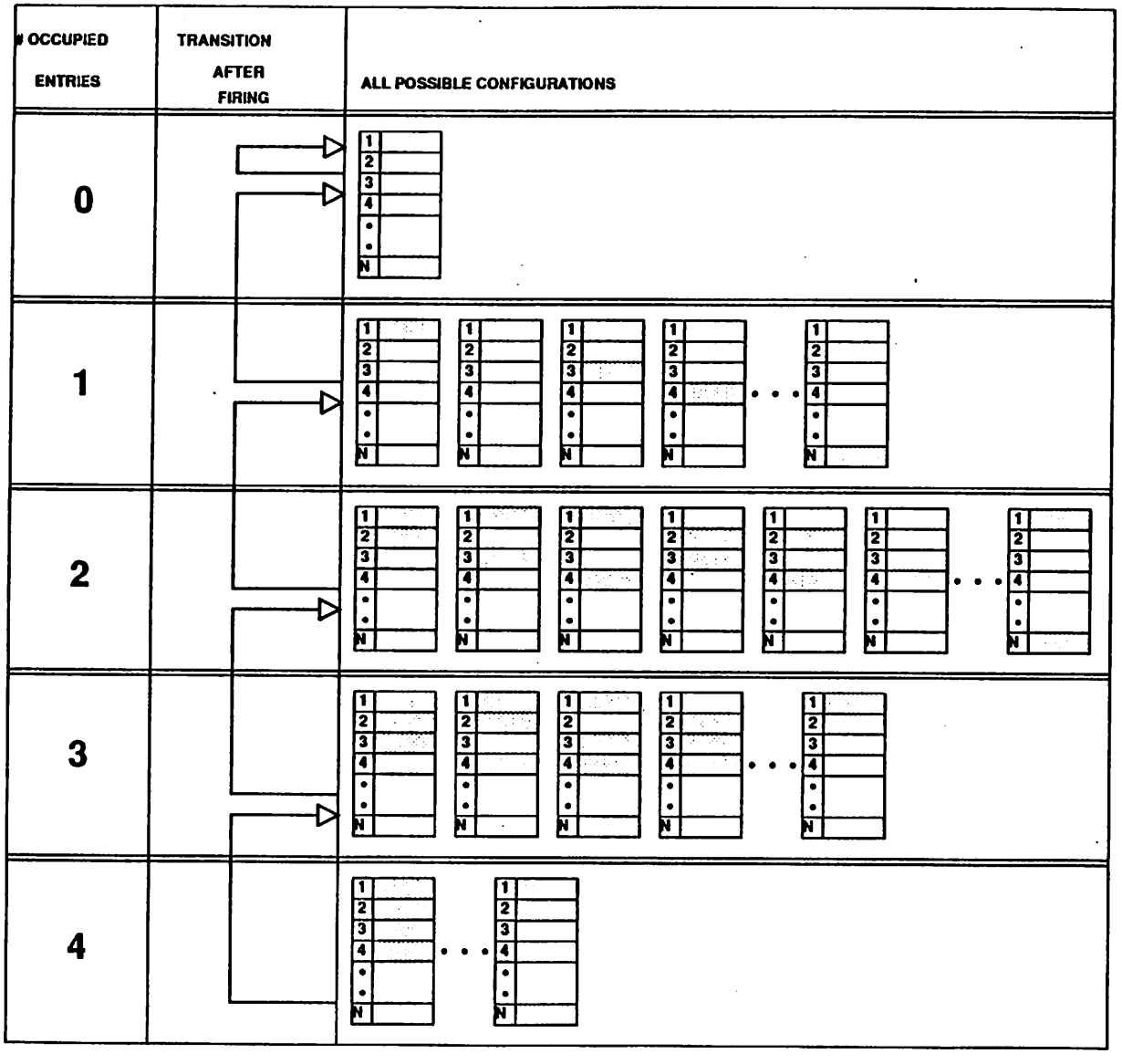


Fig. 9a. NT Configurations After FIRing

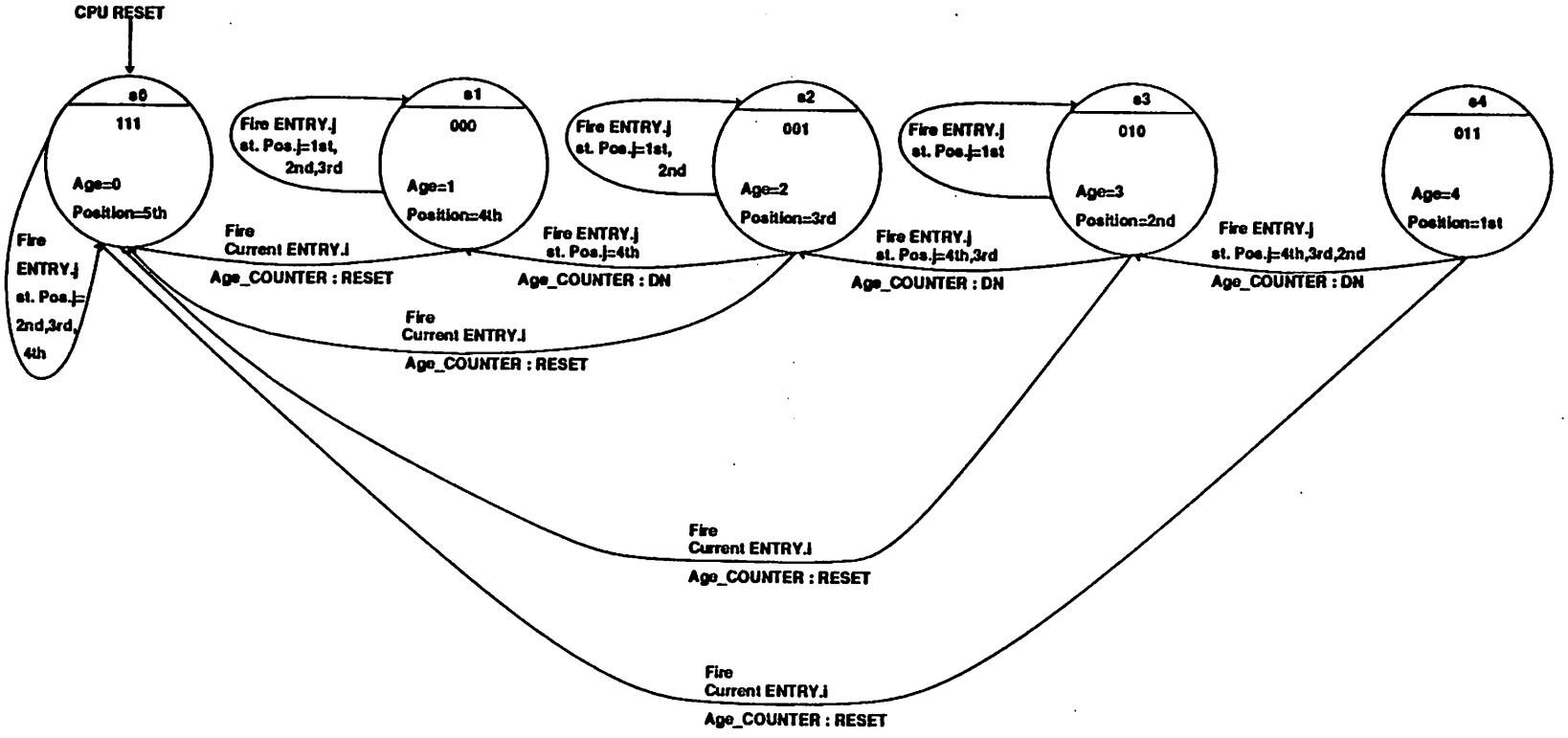


Fig. 9b. ENTRY.i AGE State Transition Diagram for FIRing

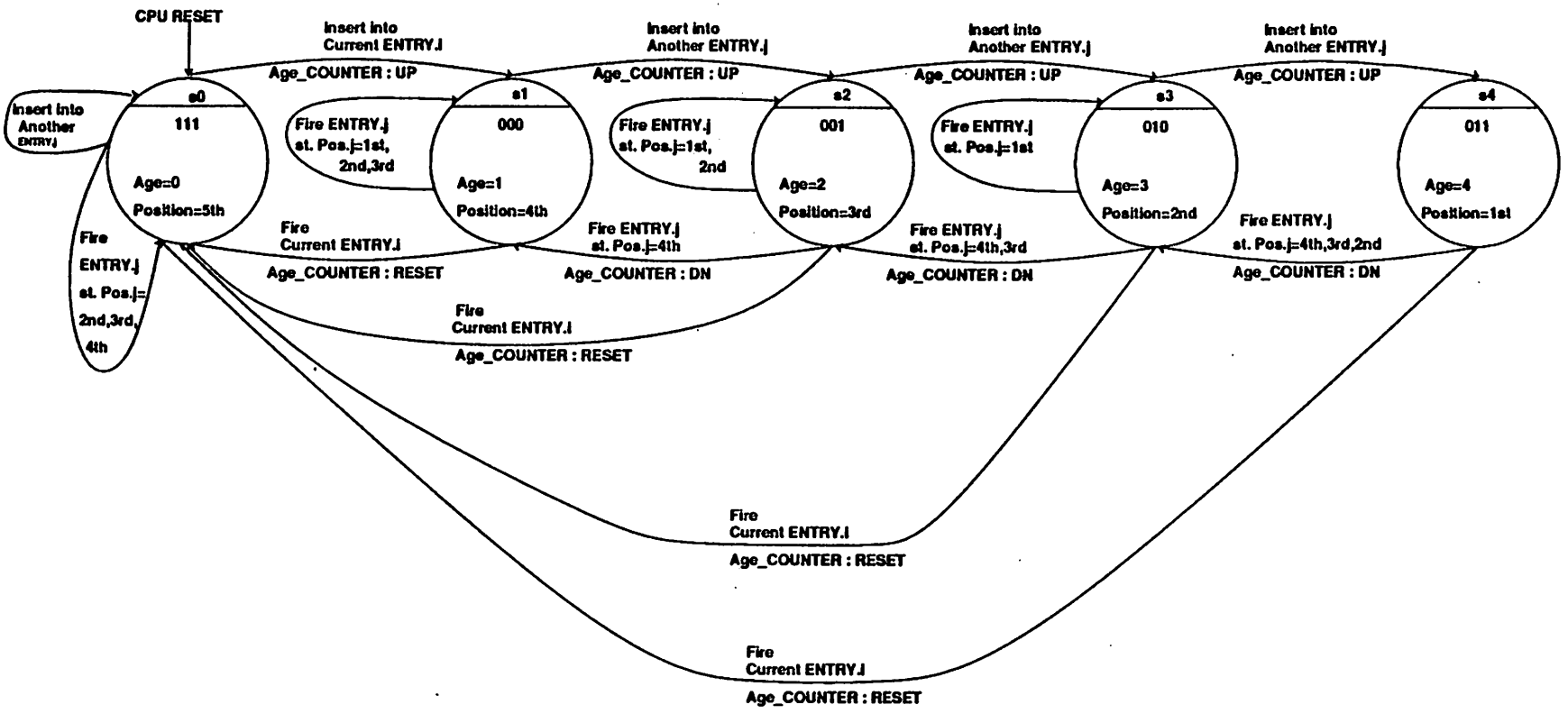


Fig. 10. ENTRY.i AGE State Transition Diagram

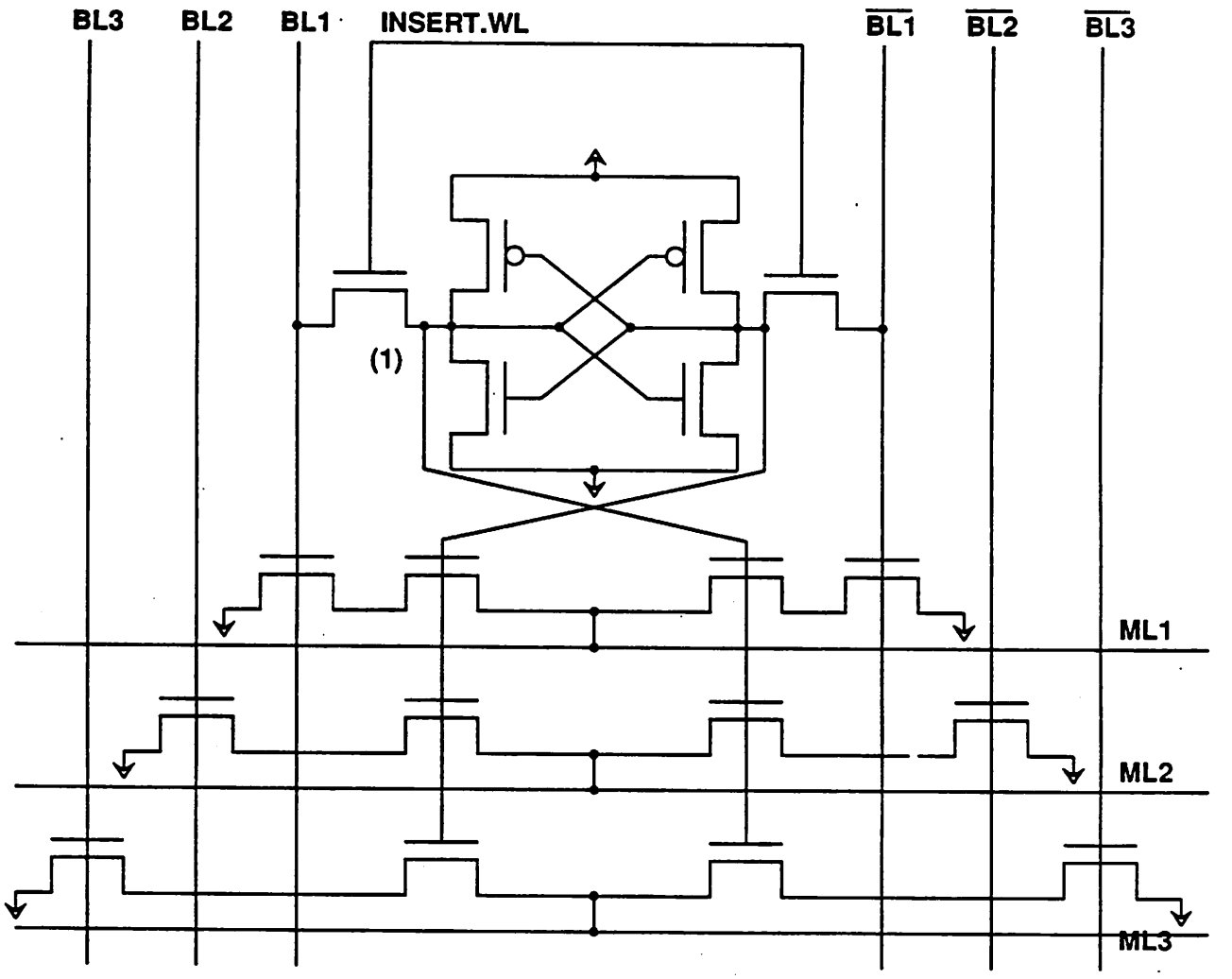


Fig. 11: NT TAG Cell

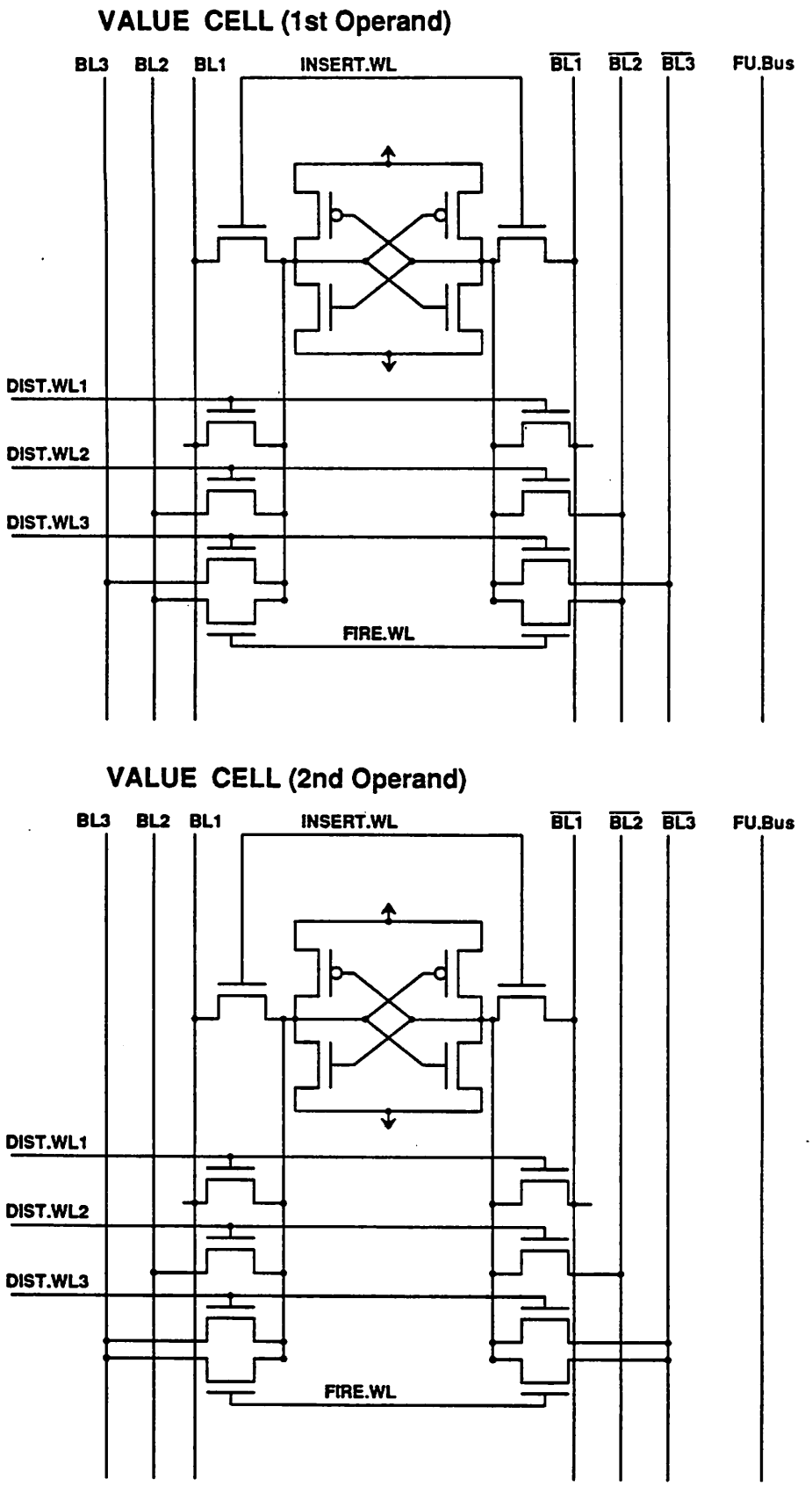


Fig. 12: NT VALUE Cells

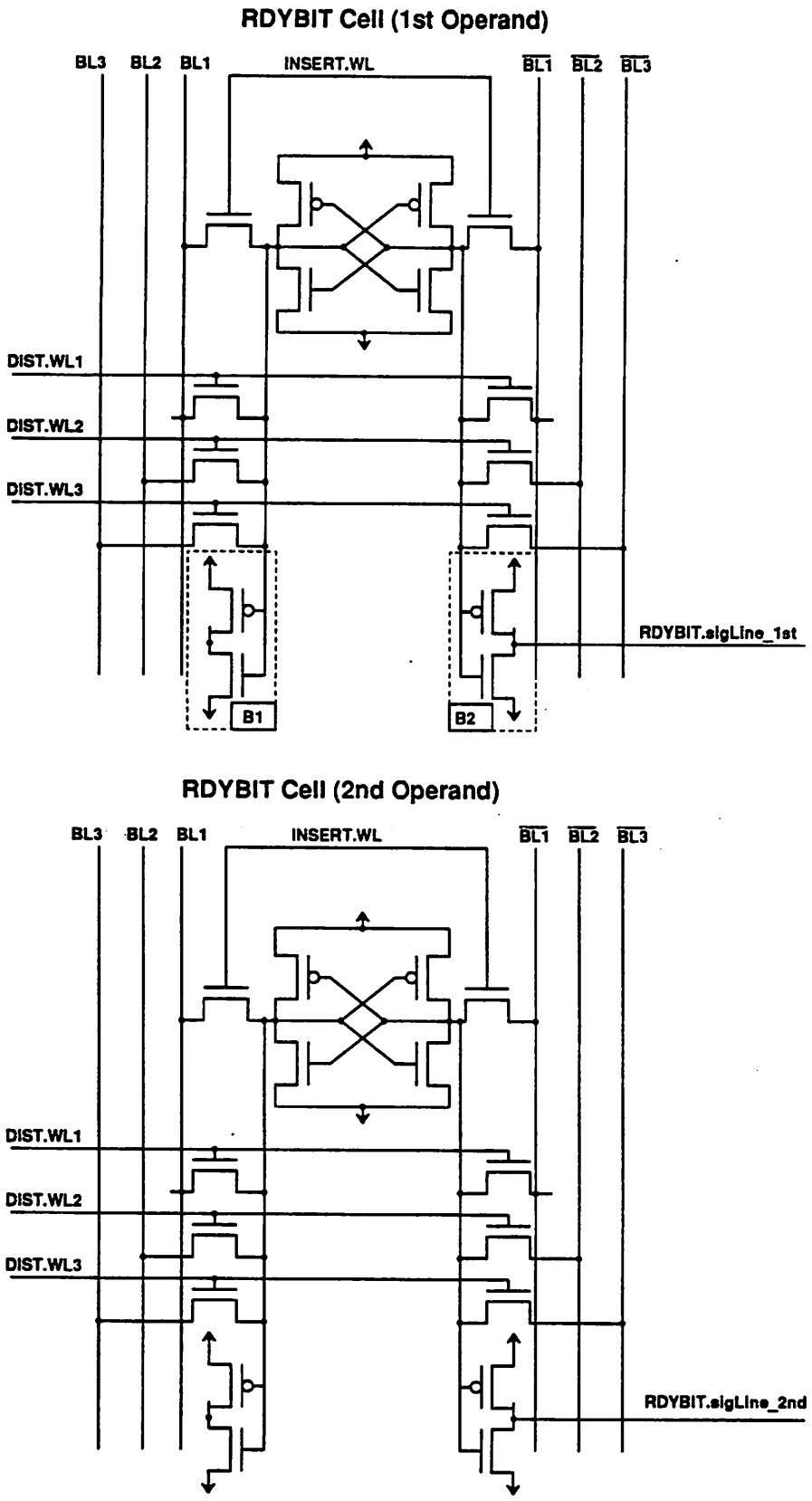


Fig. 13: NT RDYBIT Cells

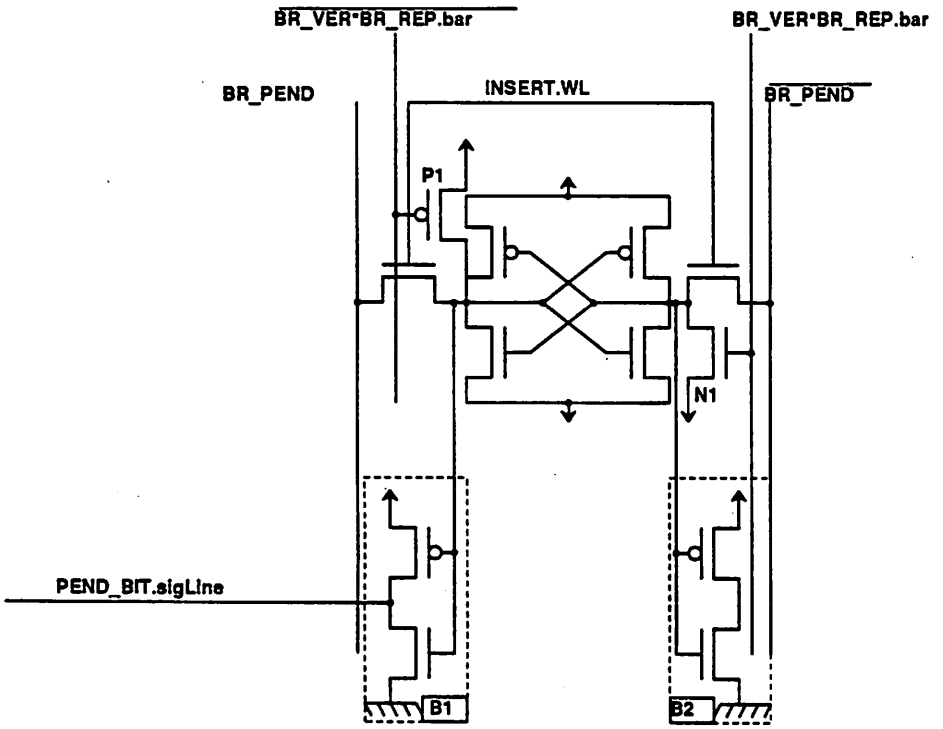


Fig. 14: NT BRANCH PENDING Cell

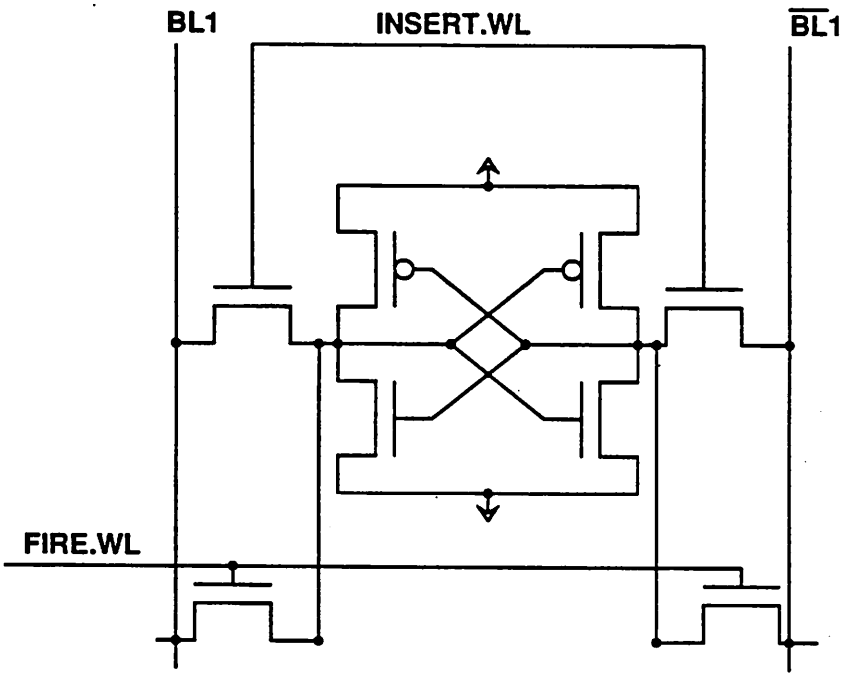


Fig. 15: NT OPCODE Cell

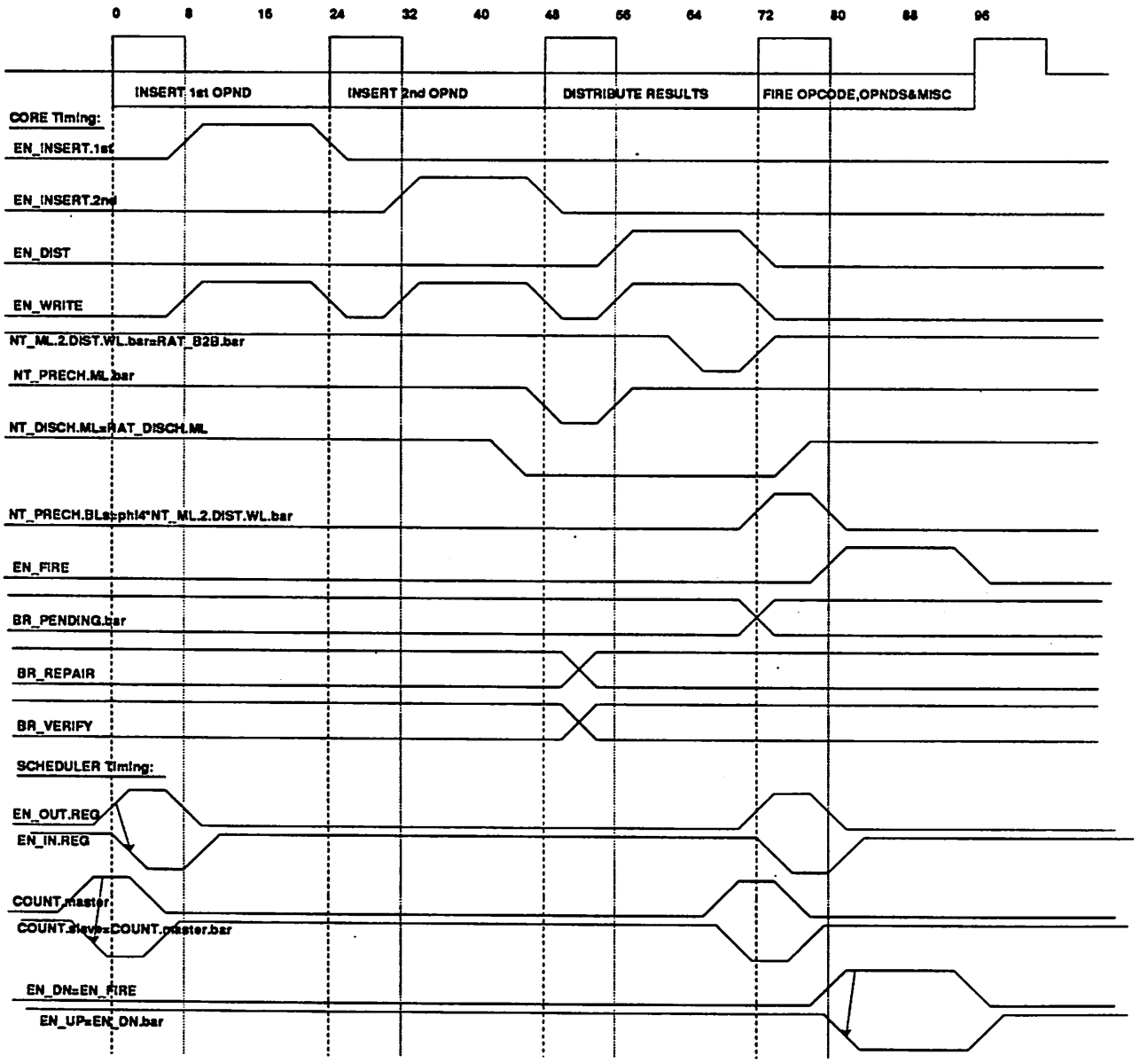


Fig. 16: NT Timing Chart

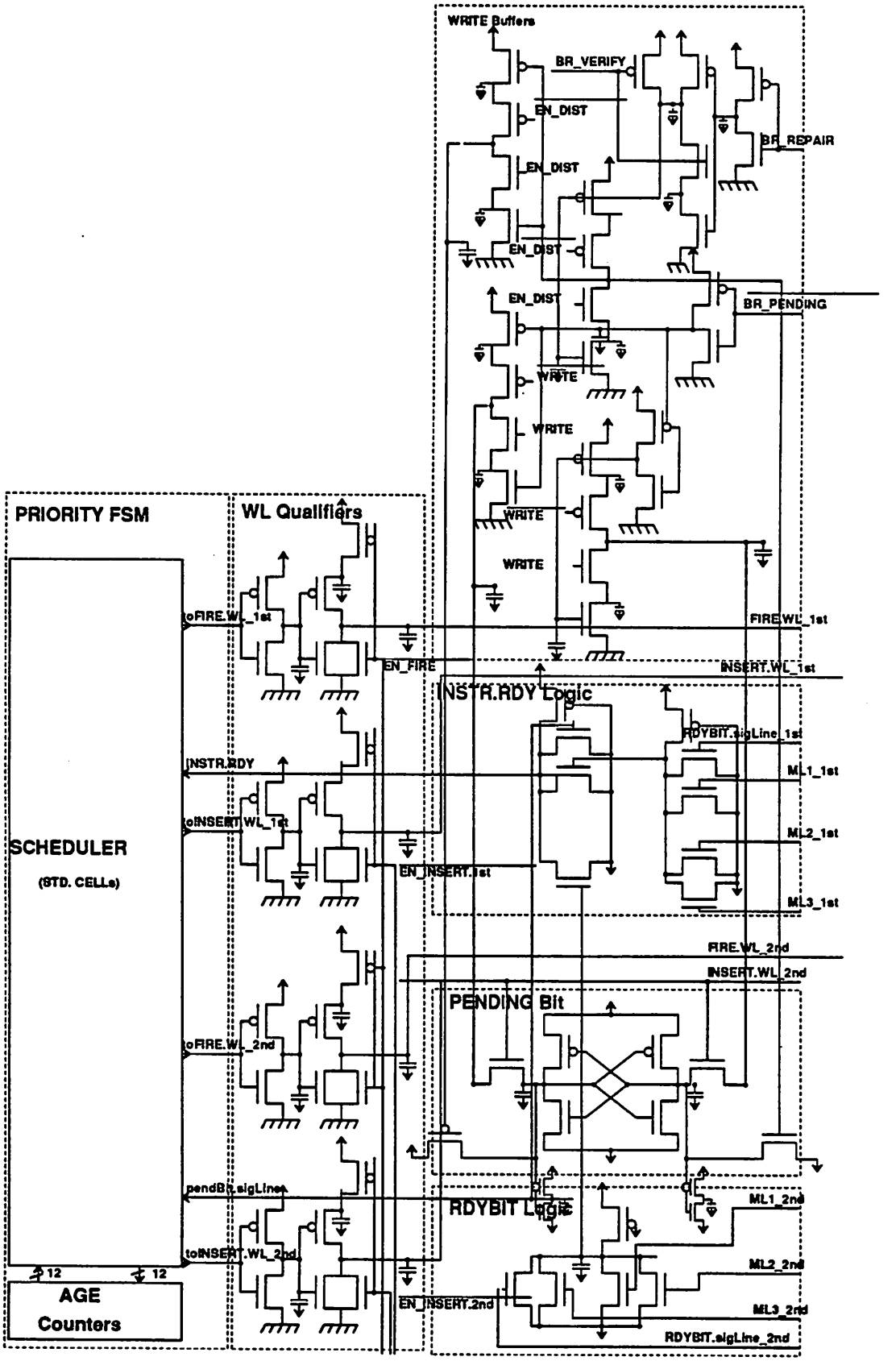


Fig. 17a: NT Lumped Model (Part 1)

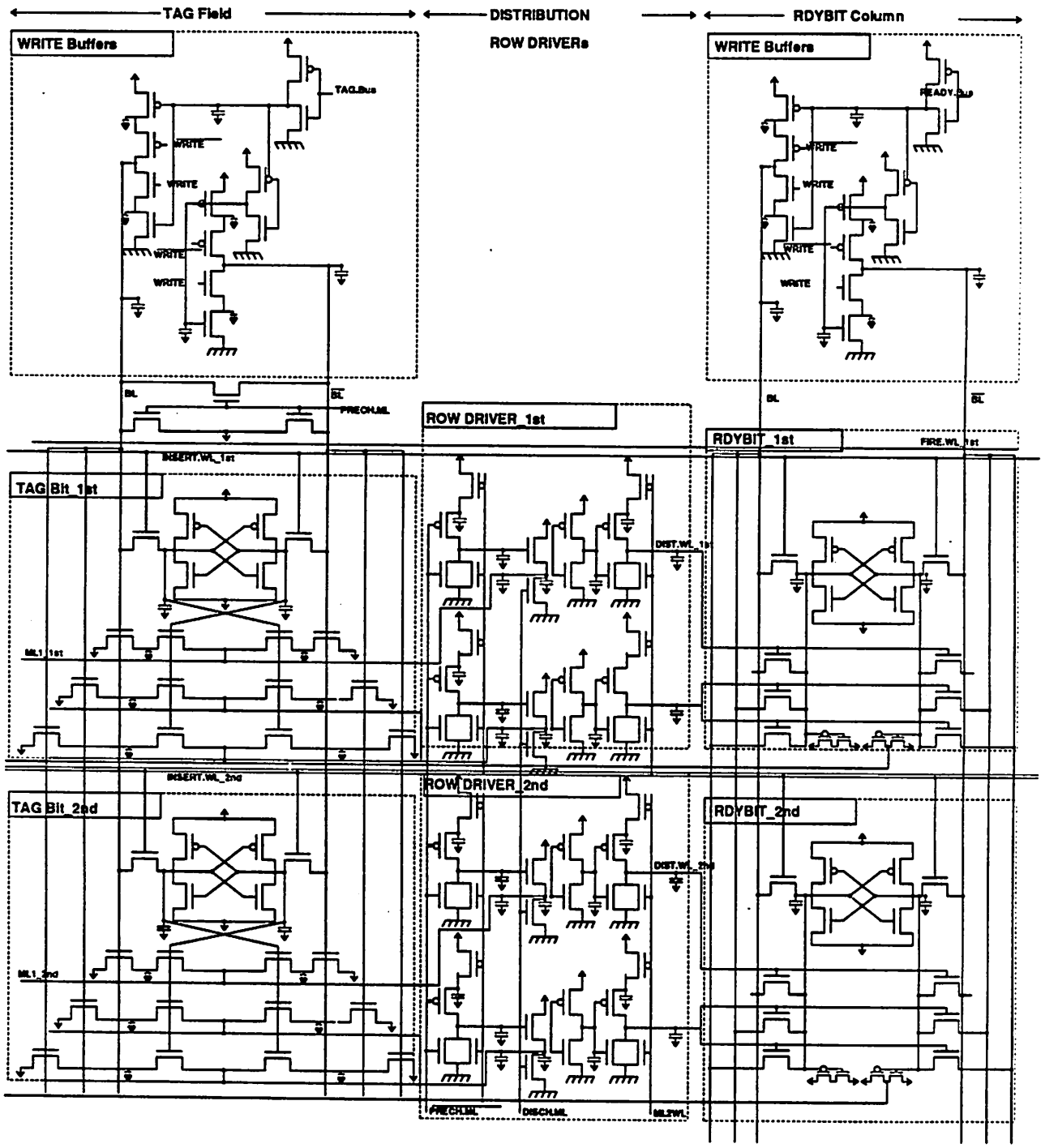


Fig. 17b: NT Lumped Model (Part 2)

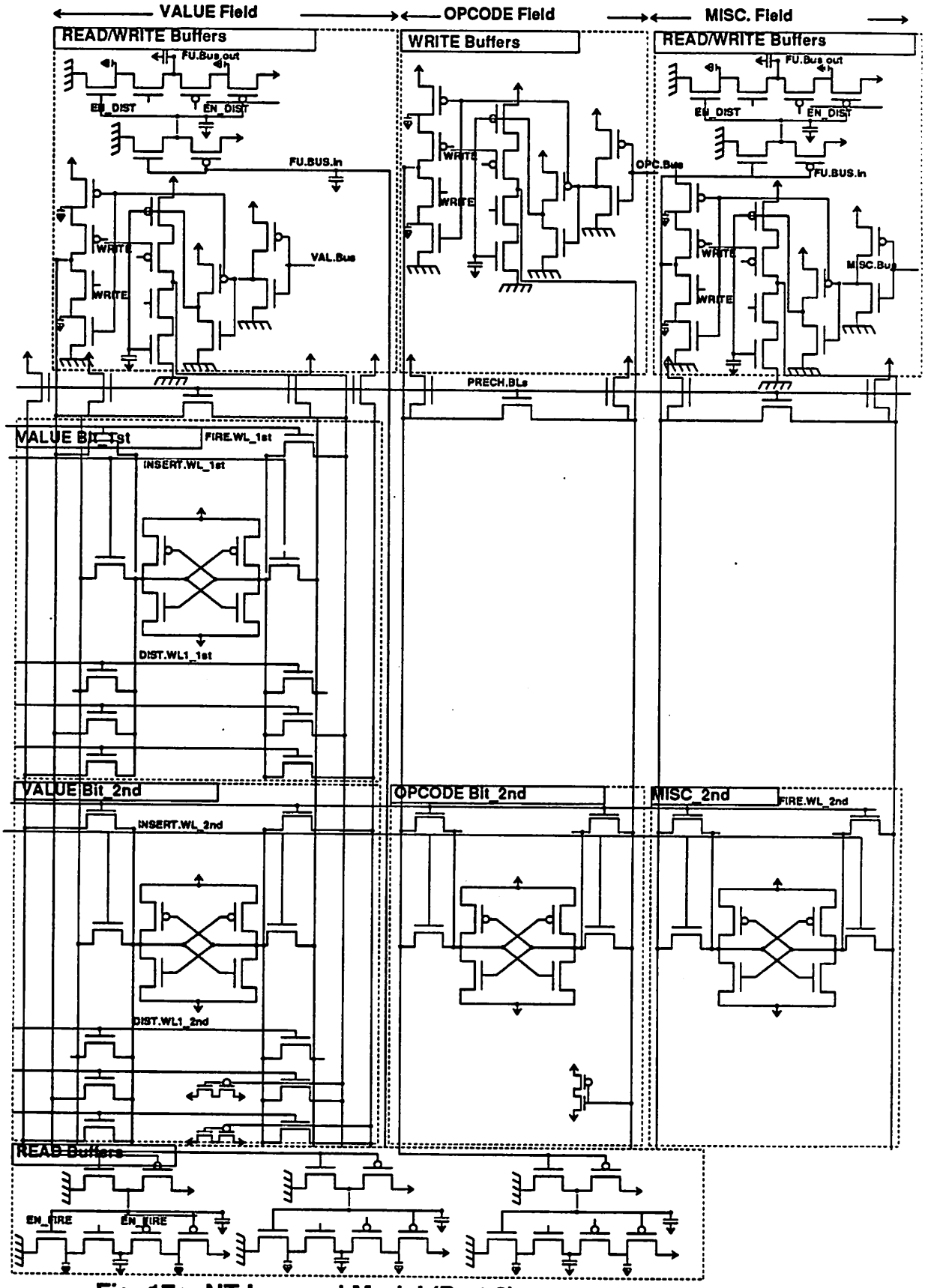


Fig. 17c: NT Lumped Model (Part 3)

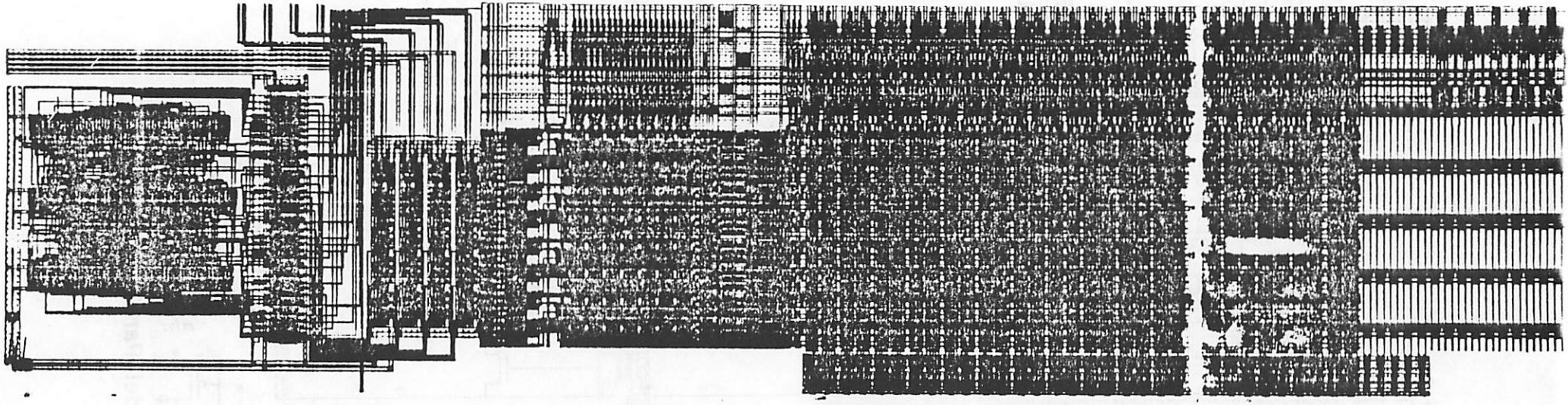


Fig. 18: NT Layout

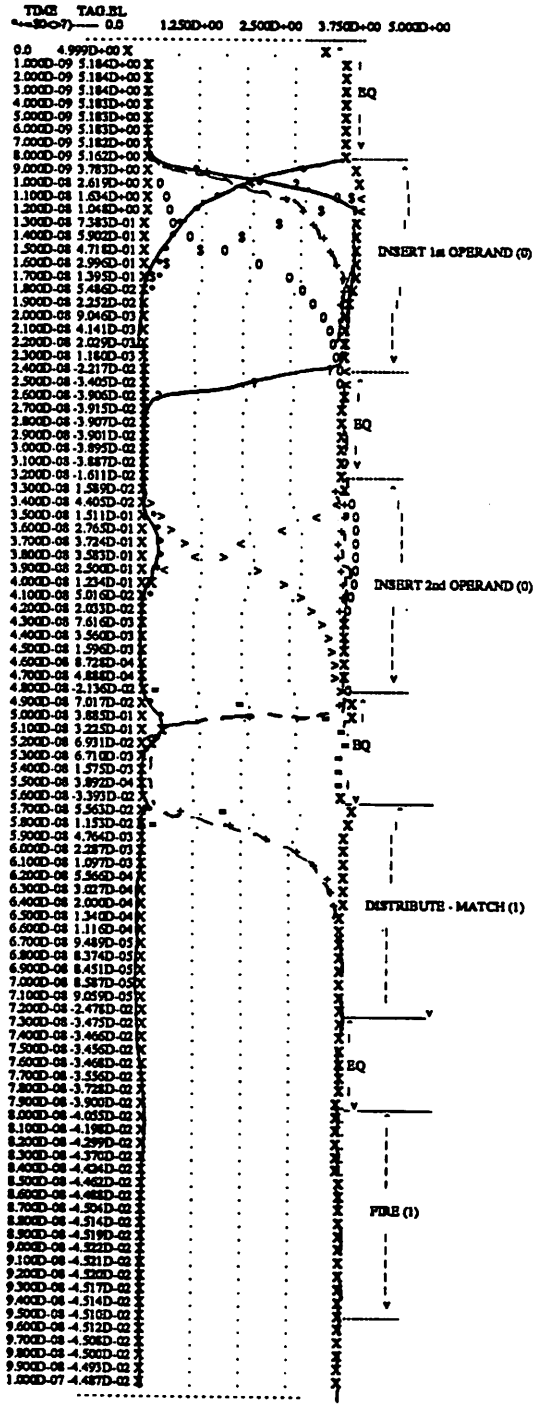
****

Fig. 19a: TAG FIELD - INSERT.1st OPERAND (0); INSERT.2nd OPERAND (0); DISTRIBUTE (1) - MATCH; FIRE (1).

****

*: TAG.BL
+: TAG.BL_bar
=: DISCH.BLs (=PRECH.ML)
$: TAG.DATA.node_1st
0: TAG.DATA.node_bar_1st
<: TAG.DATA.node_2nd
>: TAG.DATA.node_bar_2nd

?: INSERT.WL_1st



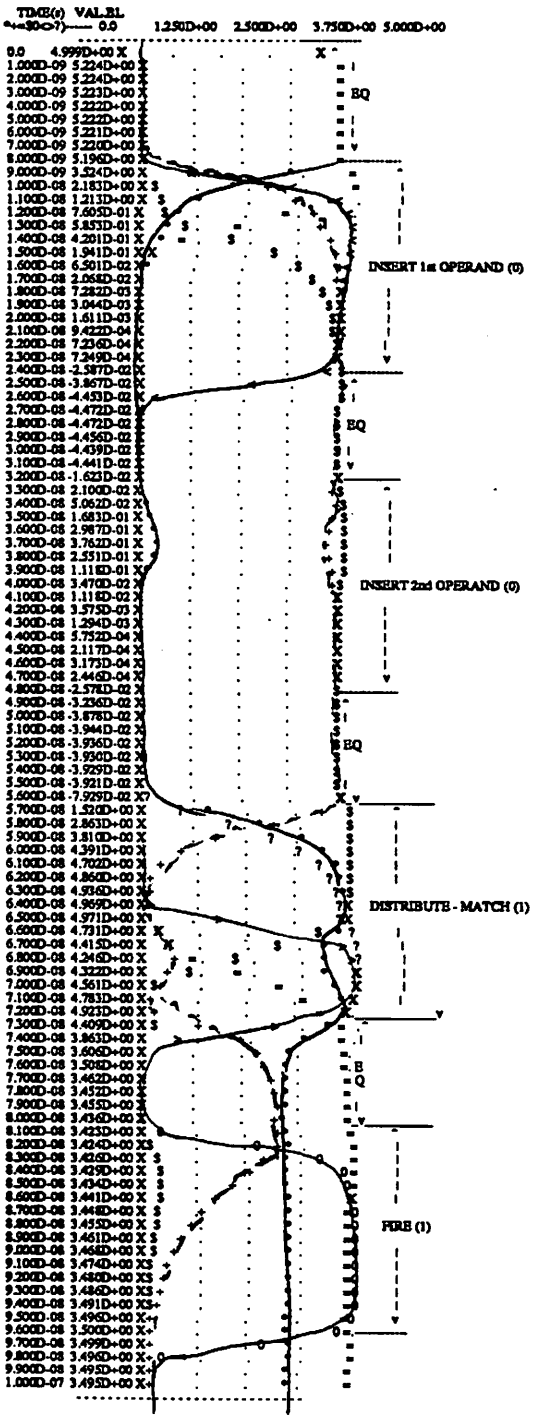
****

Fig. 19b: VALUE FIELD - INSERT.1st OPERAND (0); INSERT.2nd OPERAND (0); DISTRIBUTE (1) - MATCH; FIRE (1).

****

*: VAL.BL
+: VAL.BL.bar
=: VAL.DATA.node
$: VAL.DATA.node.bar
0: FIRE.WL_1st
<: INSERT.WL_1st
>: DIST.WL1_1st

?: VAL_FU.Bus.out



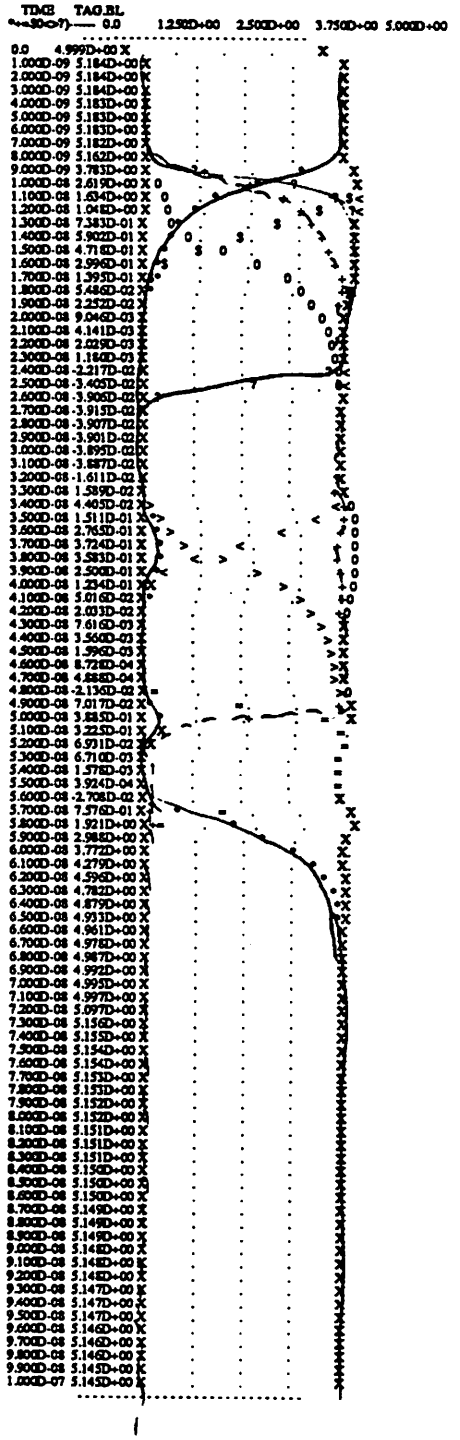
****

Fig. 19c: TAG FIELD - INSERT.1st OPERAND (0); INSERT.2nd OPERAND (0);
DISTRIBUTE (1) - MISMATCH; FIRE (0).

****

*: TAG.BL
+: TAG.BL.bar
=: DISCH.BLs (=PRECH.ML)
$: TAG.DATA.node_1st
0: TAG.DATA.node.bar_1st
<: TAG.DATA.node_2nd
>: TAG.DATA.node.bar_2nd

?: INSERT.WL_1st



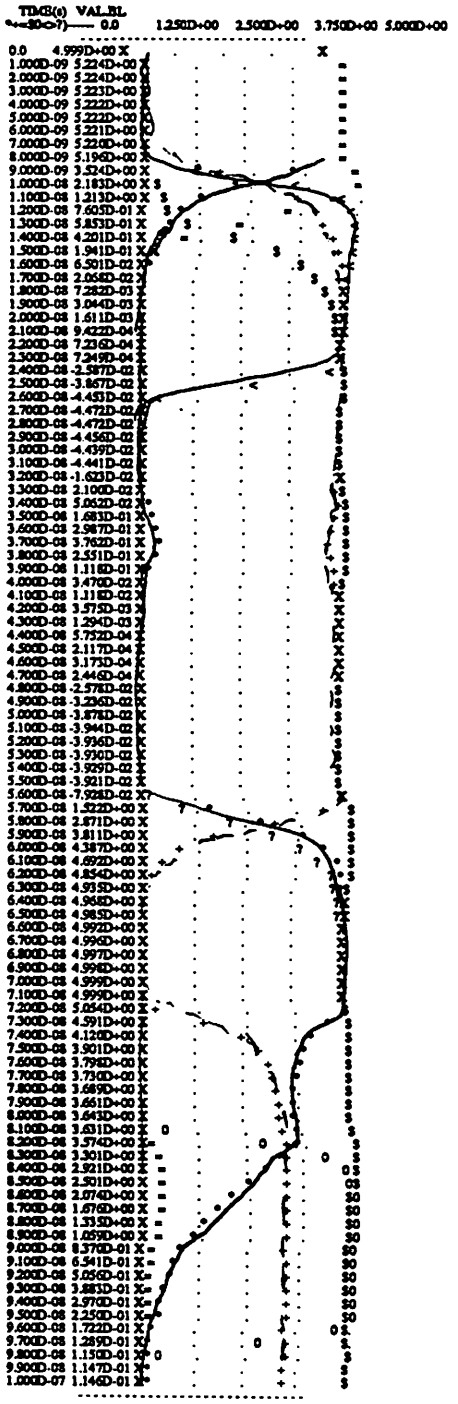
****

Fig. 19d: VALUE FIELD - INSERT.1st OPERAND (0); INSERT.2nd OPERAND (0);
DISTRIBUTE (1) - MISMATCH; FIRE (0).

****

*: VAL_BL
+: VAL_BL.bar
=: VAL_DATA.node
$: VAL_DATA.node.bar
0: FIRE.WL_1st
<: INSERT.WL_1st
>: DIST.WL1_1st

?: VAL_FU.Bus.out



APPENDIX I

```
*************************************************************************
  APPENDIX I : SCHEDULER Description as CAD Tools Input.
*************************************************************************
NAME = SMART.PLA ;
INORDER  = BUS_VALID.BIT
        AGE_1[3] AGE_1[2] AGE_1[1] INST_RDY_1
        AGE_2[3] AGE_2[2] AGE_2[1] INST_RDY_2
        AGE_3[3] AGE_3[2] AGE_3[1] INST_RDY_3
        AGE_4[3] AGE_4[2] AGE_4[1] INST_RDY_4;
OUTORDER = toINSERT.WL_1 toFIRE.WL_1 toUP_1 toDN_1
        toINSERT.WL_2 toFIRE.WL_2 toUP_2 toDN_2
        toINSERT.WL_3 toFIRE.WL_3 toUP_3 toDN_3
        toINSERT.WL_4 toFIRE.WL_4 toUP_4 toDN_4
        QFULL;
POS_1.1 =  AGE_1[2]& AGE_1[1];
POS_1.2 =  AGE_1[2]& (!AGE_1[1]);
POS_1.3 = (!AGE_1[2]) & AGE_1[1];
POS_1.4 = (!AGE_1[2]) & (!AGE_1[1]);
POS_1.5 = AGE_1[3];
POS_2.1 =  AGE_2[2]& AGE_2[1];
POS_2.2 =  AGE_2[2]& (!AGE_2[1]);
POS_2.3 = (!AGE_2[2]) & AGE_2[1];
POS_2.4 = (!AGE_2[2]) & (!AGE_2[1]);
POS_2.5 = AGE_2[3];
POS_3.1 =  AGE_3[2]& AGE_3[1];
POS_3.2 =  AGE_3[2]& (!AGE_3[1]);
POS_3.3 = (!AGE_3[2]) & AGE_3[1];
POS_3.4 = (!AGE_3[2]) & (!AGE_3[1]);
POS_3.5 = AGE_3[3];
POS_4.1 =  AGE_4[2]& AGE_4[1];
POS_4.2 =  AGE_4[2]& (!AGE_4[1]);
POS_4.3 = (!AGE_4[2]) & AGE_4[1];
POS_4.4 = (!AGE_4[2]) & (!AGE_4[1]);
POS_4.5 = AGE_4[3];

EMPTY_1 = POS_1.5;
EMPTY_2 = POS_2.5;
EMPTY_3 = POS_3.5;
EMPTY_4 = POS_4.5;
QFULL = !(EMPTY_1 + EMPTY_2 + EMPTY_3 + EMPTY_4);
NES.IS.ENTRY_1 = EMPTY_1;
NES.IS.ENTRY_2 = (!EMPTY_1)&EMPTY_2;
NES.IS.ENTRY_3 = (!EMPTY_1)&(!EMPTY_2)&EMPTY_3;
NES.IS.ENTRY_4 = (!EMPTY_1)&(!EMPTY_2)&(!EMPTY_3)&EMPTY_4;

FIRE_1.1st = POS_1.1 & INST_RDY_1;
FIRE_1.2nd = POS_1.2 & INST_RDY_1;
FIRE_1.3rd = POS_1.3 & INST_RDY_1;
FIRE_1.4th = POS_1.4 & INST_RDY_1;
OTHER.FIRE.1st.WL_1 = FIRE_2.1st + FIRE_3.1st + FIRE_4.1st;
OTHER.FIRE.2nd.WL_1 = FIRE_2.2nd + FIRE_3.2nd + FIRE_4.2nd;
OTHER.FIRE.3rd.WL_1 = FIRE_2.3rd + FIRE_3.3rd + FIRE_4.3rd;
FIRE.1st.WL_1 = FIRE_1.1st;
```

```
FIRE.2nd.WL_1 = FIRE_1.2nd & (!OTHER.FIRE.1st.WL_1);
FIRE.3rd.WL_1 = FIRE_1.3rd & (!OTHER.FIRE.1st.WL_1) &
                (!OTHER.FIRE.2nd.WL_1);
FIRE.4th.WL_1 = FIRE_1.4th & (!OTHER.FIRE.1st.WL_1) &
        (!OTHER.FIRE.2nd.WL_1) & (!OTHER.FIRE.3rd.WL_1);
toFIRE.WL_1 = FIRE.1st.WL_1 + FIRE.2nd.WL_1 + FIRE.3rd.WL_1
                + FIRE.4th.WL_1;
FIRE_2.1st = POS_2.1 & INST_RDY_2;
FIRE_2.2nd = POS_2.2 & INST_RDY_2;
FIRE_2.3rd = POS_2.3 & INST_RDY_2;
FIRE_2.4th = POS_2.4 & INST_RDY_2;
OTHER.FIRE.1st.WL_2 = FIRE_1.1st + FIRE_3.1st + FIRE_4.1st;
OTHER.FIRE.2nd.WL_2 = FIRE_1.2nd + FIRE_3.2nd + FIRE_4.2nd;
OTHER.FIRE.3rd.WL_2 = FIRE_1.3rd + FIRE_3.3rd + FIRE_4.3rd;
FIRE.1st.WL_2 = FIRE_2.1st;
FIRE.2nd.WL_2 = FIRE_2.2nd & (!OTHER.FIRE.1st.WL_2);
FIRE.3rd.WL_2 = FIRE_2.3rd & (!OTHER.FIRE.1st.WL_2) &
                (!OTHER.FIRE.2nd.WL_2);
FIRE.4th.WL_2 = FIRE_2.4th & (!OTHER.FIRE.1st.WL_2) &
        (!OTHER.FIRE.2nd.WL_2)  & (!OTHER.FIRE.3rd.WL_2);
toFIRE.WL_2 = FIRE.1st.WL_2 + FIRE.2nd.WL_2 + FIRE.3rd.WL_2
                + FIRE.4th.WL_2;
FIRE_3.1st = POS_3.1 & INST_RDY_3;
FIRE_3.2nd = POS_3.2 & INST_RDY_3;
FIRE_3.3rd = POS_3.3 & INST_RDY_3;
FIRE_3.4th = POS_3.4 & INST_RDY_3;
OTHER.FIRE.1st.WL_3 = FIRE_1.1st + FIRE_2.1st + FIRE_4.1st;
OTHER.FIRE.2nd.WL_3 = FIRE_1.2nd + FIRE_2.2nd + FIRE_4.2nd;
OTHER.FIRE.3rd.WL_3 = FIRE_1.3rd + FIRE_2.3rd + FIRE_4.3rd;
FIRE.1st.WL_3 = FIRE_3.1st;
FIRE.2nd.WL_3 = FIRE_3.2nd & (!OTHER.FIRE.1st.WL_3);
FIRE.3rd.WL_3 =FIRE_3.3rd&(!OTHER.FIRE.1st.WL_3)& (!OTHER.FIRE.2nd.WL_3);
FIRE.4th.WL_3 = FIRE_3.4th & (!OTHER.FIRE.1st.WL_3)
        &(!OTHER.FIRE.2nd.WL_3)&(!OTHER.FIRE.3rd.WL_3);
toFIRE.WL_3 = FIRE.1st.WL_3 + FIRE.2nd.WL_3+FIRE.3rd.WL_3+FIRE.4th.WL_3;
FIRE_4.1st = POS_4.1 & INST_RDY_4;
FIRE_4.2nd = POS_4.2 & INST_RDY_4;
FIRE_4.3rd = POS_4.3 & INST_RDY_4;
FIRE_4.4th = POS_4.4 & INST_RDY_4;
OTHER.FIRE.1st.WL_4 = FIRE_1.1st + FIRE_2.1st + FIRE_3.1st;
OTHER.FIRE.2nd.WL_4 = FIRE_1.2nd + FIRE_2.2nd + FIRE_3.2nd;
OTHER.FIRE.3rd.WL_4 = FIRE_1.3rd + FIRE_2.3rd + FIRE_3.3rd;
FIRE.1st.WL_4 = FIRE_4.1st;
FIRE.2nd.WL_4 = FIRE_4.2nd & (!OTHER.FIRE.1st.WL_4);
FIRE.3rd.WL_4 =FIRE_4.3rd &(!OTHER.FIRE.1st.WL_4)&(!OTHER.FIRE.2nd.WL_4);
FIRE.4th.WL_4 = FIRE_4.4th & (!OTHER.FIRE.1st.WL_4)
        & (!OTHER.FIRE.2nd.WL_4) & (!OTHER.FIRE.3rd.WL_4);
toFIRE.WL_4 = FIRE.1st.WL_4 + FIRE.2nd.WL_4+FIRE.3rd.WL_4+FIRE.4th.WL_4;

toINSERT.WL_1 = BUS_VALID.BIT & NES.IS.ENTRY_1;
toINSERT.WL_2 = BUS_VALID.BIT & NES.IS.ENTRY_2;
toINSERT.WL_3 = BUS_VALID.BIT & NES.IS.ENTRY_3;
toINSERT.WL_4 = BUS_VALID.BIT & NES.IS.ENTRY_4;
```

```
INSERT.ANOTHER_1 = BUS_VALID.BIT & (!QFULL);
INSERT.ANOTHER_2 = BUS_VALID.BIT & (!QFULL);
INSERT.ANOTHER_3 = BUS_VALID.BIT & (!QFULL);
INSERT.ANOTHER_4 = BUS_VALID.BIT & (!QFULL);
toUP_1 = toINSERT.WL_1 + (INSERT.ANOTHER_1 &  (!EMPTY_1));
toUP_2 = toINSERT.WL_2 + (INSERT.ANOTHER_2 &  (!EMPTY_2));
toUP_3 = toINSERT.WL_3 + (INSERT.ANOTHER_3  & (!EMPTY_3));
toUP_4 = toINSERT.WL_4 + (INSERT.ANOTHER_4  & (!EMPTY_4));

FIRED_1.AT.2OR3OR4 = FIRE.2nd.WL_1 + FIRE.3rd.WL_1 + FIRE.4th.WL_1;
FIRED_2.AT.2OR3OR4 = FIRE.2nd.WL_2 + FIRE.3rd.WL_2 + FIRE.4th.WL_2;
FIRED_3.AT.2OR3OR4 = FIRE.2nd.WL_3 + FIRE.3rd.WL_3 + FIRE.4th.WL_3;
FIRED_4.AT.2OR3OR4 = FIRE.2nd.WL_4 + FIRE.3rd.WL_4 + FIRE.4th.WL_4;
FIRED_1.AT.3OR4 = FIRE.3rd.WL_1 + FIRE.4th.WL_1;
FIRED_2.AT.3OR4 = FIRE.3rd.WL_2 + FIRE.4th.WL_2;
FIRED_3.AT.3OR4 = FIRE.3rd.WL_3 + FIRE.4th.WL_3;
FIRED_4.AT.3OR4 = FIRE.3rd.WL_4 + FIRE.4th.WL_4;
FIRED_1.AT.4 = FIRE.4th.WL_1;
FIRED_2.AT.4 = FIRE.4th.WL_2;
FIRED_3.AT.4 = FIRE.4th.WL_3;
FIRED_4.AT.4 = FIRE.4th.WL_4;

POS1.AND.FIRED.LOWER_1 = POS_1.1&(FIRED_2.AT.2OR3OR4
                + FIRED_3.AT.2OR3OR4 + FIRED_4.AT.2OR3OR4);
POS2.AND.FIRED.LOWER_1 = POS_1.2&(FIRED_2.AT.3OR4 + FIRED_3.AT.3OR4
                        + FIRED_4.AT.3OR4);
POS3.AND.FIRED.LOWER_1 =POS_1.3&(FIRED_2.AT.4+FIRED_3.AT.4+FIRED_4.AT.4);
toDN_1 = (POS1.AND.FIRED.LOWER_1 + POS2.AND.FIRED.LOWER_1
                + POS3.AND.FIRED.LOWER_1);
POS1.AND.FIRED.LOWER_2 = POS_2.1&(FIRED_1.AT.2OR3OR4
                + FIRED_3.AT.2OR3OR4 + FIRED_4.AT.2OR3OR4);
POS2.AND.FIRED.LOWER_2 = POS_2.2&(FIRED_1.AT.3OR4 + FIRED_3.AT.3OR4
                        + FIRED_4.AT.3OR4);
POS3.AND.FIRED.LOWER_2 =POS_2.3&(FIRED_1.AT.4+FIRED_3.AT.4+FIRED_4.AT.4);
toDN_2 = (POS1.AND.FIRED.LOWER_2 + POS2.AND.FIRED.LOWER_2
                + POS3.AND.FIRED.LOWER_2);
POS1.AND.FIRED.LOWER_3 = POS_3.1&(FIRED_1.AT.2OR3OR4
                + FIRED_2.AT.2OR3OR4 + FIRED_4.AT.2OR3OR4);
POS2.AND.FIRED.LOWER_3 = POS_3.2&(FIRED_1.AT.3OR4 + FIRED_2.AT.3OR4
                        + FIRED_4.AT.3OR4);
POS3.AND.FIRED.LOWER_3 =POS_3.3&(FIRED_1.AT.4+FIRED_2.AT.4+FIRED_4.AT.4);
toDN_3 = (POS1.AND.FIRED.LOWER_3 + POS2.AND.FIRED.LOWER_3
                + POS3.AND.FIRED.LOWER_3);
POS1.AND.FIRED.LOWER_4 = POS_4.1&(FIRED_1.AT.2OR3OR4
                + FIRED_2.AT.2OR3OR4 + FIRED_3.AT.2OR3OR4);
POS2.AND.FIRED.LOWER_4 = POS_4.2&(FIRED_1.AT.3OR4 + FIRED_2.AT.3OR4
                        + FIRED_3.AT.3OR4);
POS3.AND.FIRED.LOWER_4 =POS_4.3&(FIRED_1.AT.4+FIRED_2.AT.4+FIRED_3.AT.4);
toDN_4 = (POS1.AND.FIRED.LOWER_4 + POS2.AND.FIRED.LOWER_4
                + POS3.AND.FIRED.LOWER_4);
```

CHAPTER 4

RAT : A Smart Data Memory Case Study

4.

4.1. Introduction

RAT [1] is the main smart memory on the HPSm CPU chip and plays the major role of manipulating the data flow graph - the central data structure in a data flow CPU. It is analogous to a register file in a conventional von Neumann CPU but with extra capabilities for enhanced data manipulation and support for out-of-order execution control. In Figure 1, RAT communicates with 3 other smart instruction memories (MEMORY NODE TABLE, ALU NODE TABLE, and CONTROL NODE TABLE) through 3 ports. These smart instruction memories in turn support 3 CPU functional memories (MEMORY, ALU and CONTROL). RAT has a content-addressable tag field to support associative operations, and has 2 backup copies per data element to support branch prediction and exception handling. Figure 2 shows the RAT with its fields and its relation to the global busses and signals.

4.2. Memory Architecture

The memory architecture is shown in Figure 3. Information is stored in a 31-word format - 40 bits/word - forming a 1.24K array that is accessible from the external world. The core is partitioned into 3 fields - a 7-bit tag field (a 2-bit non-content addressable tag field and a 5-bit content addressable tag field), a ready bit field (1 bit) and a value field (32 bits). Each field is divided into 2 halves by a set of sense amplifiers. Each of the 3 smart instruction memories in Figure 1 logically sees one I/O port looking into the RAT which it uses repeatedly for all its interaction with the RAT. By multiplexing each

port for 2 READs, 1 WRITE and 2 associative WRITEs per cycle, the RAT provides the function of a 15-port memory while paying the price of a 3-port memory in terms of area and power. However, the time-sharing of the ports by different operations certainly has an impact on the cycle time.

Six multiplexers and six decoders are used - 3 for conventional READ/WRITE operations and 3 for assisting the associative WRITE operations. The READ_ENABLE circuitry divide the decoders into an upper half and a lower half for the upper and lower halves of the core respectively. This circuitry is used to conditionally inhibit some ports during READing. The row drivers buffer and multiplex the tag word line (for conventional operations) and the tag match line (for content addressable operations) to drive the ready bit/ value field word lines.

The tag and ready bit fields have 1 backup copy while the value field has 2 backup copies to support branch prediction and exception handling. So indeed, although only 31 x 40 bits are directly accessible from the external world, the core really stores 3.47K bits. The programmer-visible cells are called the *current* cells (*C.cell*) while the first and second backup cells are called the *transit* (*T.cell*) and *settled* (*S.cell*) cells respectively. Conceptually, the RAT can be viewed as a "3-dimensional" memory where the backup copies (the boxes in dotted lines in Figure 3) represent the depth.

4.3. Basic Operations

The timing sequence is illustrated in Figure 4. The RAT is designed to run at the 10MHz clock frequency required for the HPSm CPU chip. All signals are aligned to the 4-phase clocks of the data flow CPU as shown in the figure. In the sequel, *phase i* is defined as the time segment from the rising edge of ϕ_i to the rising edge of ϕ_{i+1}. In each of the first two phases, 3 words from 3 (possibly different) locations are read out into the three different ports. (Actually, 0 to 3 words are read out depending on the VALID bits of the ADDRESS busses fed into the READ_ENABLE circuitry.) Then in phase 3,

computed results from the 3 function units are written into the *current* and *transit* cells of the ready/value fields if the interrogative tags match the stored tags. Phase 4 is used to update the data flow graph by writing new values into the tag and ready fields (the MERGE operation mentioned in Chapter 2).

The RAT supports branch prediction and exception handling by allowing SAVE and REPAIR operations. These operations are defined as follows. If a conditional branch is encountered in the instruction stream, the data flow CPU predicts the branch to be taken and informs the RAT to save (the SAVE operation) its *current* values into the *transit* cells by raising the C2T signal in ϕ_2. If the CPU later on discovers that the prediction was incorrect, it informs the RAT to recover (the REPAIR operation) the last saved values by raising the T2C signal in phase 4. This has the effect of copying the *transit* values into the *current* cells. Exception handling needs all the SAVE/REPAIR functions of branch prediction. It also requires that the *transit* values be copied into the *settled* cells (in ϕ_1 with T2S) and that the *settled* values be copied into the *current* cells (in phase 4 with S2C). The architecture design is such that the REPAIR operations from the *transit* and the *settled* cells into the *current* cells in phase 4 do not occur together; therefore the state of the *current* cells is always well-defined.

4.4. Circuits

4.4.1. Cells

4.4.1.1. Value Triple-Cell

The schematic for the value cell is shown in Figure 5. The layout has an area of 135μm x 58μm. The value cell has 3 data storage elements:

(1) 1 pseudo-static cell for the *C.cell*,

(2) 1 dynamic cell for the *T.cell* (for branch-prediction and exception handling support) and,

(3) 1 dynamic cell for the *S.cell* (for exception handling support).

The READ/WRITE operations are:

(1) a 1-port (one of the 3 available ports) conventional WRITE to the *C.cell*,

(2) a 3-port conventional READ from the *C.cell*, and

(3) a 1-port (one of the 3 available ports) associative WRITE to the *C.cell* and the *T.cell*.

The SAVE/REPAIR operations are:

(1) a *current-to-transit* SAVE,

(2) a *transit-to-settled* SAVE,

(3) a *transit-to-current* REPAIR, and

(4) a *settled-to-current* REPAIR.

The *C.cell* is implemented as a pseudo-static cell for two reasons. The first reason has to do with the high bandwidth requirement of the RAT. The high bandwidth requirement can be met by having several ports (several bit lines and word lines), several phases or a combination of the two. With several ports, several operations occur in parallel; each operation using its own port. When several phases are used to accomplish the task, different operations time-share the use of 1 port. If a combination of the 2 strategies is used, the right balance has to be struck. More bit lines means a shorter cycle but more area and reduced noise margins while more phases means less area and increased noise margins but more control complexity and a longer cycle. In the case of the RAT, since 3 FU's have to be supported, it is

natural to have 3 bit lines, 3 word lines for the current cells and 3 word lines for the backup cells.

This multiport structure of the RAT is necessary to prevent the cycle from being prohibitively long from excessive time-sharing of a single port. The multiport structure has certain implications for the I/O buffer, sense amplifier and data cells. The I/O buffer and sense amplifier have to satisfy more and more stringent pitch requirements as the number of ports increase. Beyond a certain number of ports, certain traditional design styles for the data cells have to be abandoned to ensure that the cells are compact. First, bit lines cannot be dedicated for READ and WRITE as in RISC I [2]. Second, for the same reason, the traditional "bit line pair" approach shown in Figure 6a cannot be used. This then leaves us only with the option of employing some single-ended technique with bit lines shared by both READ and WRITE accesses. Single-ended cell designs are more critical than those of conventional differentially accessed cells. A conventional cell like that in Figure 6a preserves data during a READ disturb because of the well-known restoring action of differential access. The single-ended cell in Figure 6b by Stewart and Dingwall [3] is more compact than a differentially accessed cell but it requires boosted word lines to WRITE a "1" into the cell and half-V_{DD} bit line precharging along with PMOS or depletion loads to protect the cell data during READs [4] . Providing single-ended access using boosted word lines was rejected for reasons of complexity and process-dependence.

For the proposed pseudo-static cell in Figure 5, READ and WRITE operations are performed on the *C.cell* by directly controlling the feedback path in the cell. The REFRESH signal is kept high during a READ operation while it is taken low for WRITE operations. Keeping the REFRESH signal high protects cell data during READing while taking it low allows a contention-free WRITE. Therefore, the *C.cell* behaves like a static RAM cell for READing and like a dynamic RAM cell for WRITing. Since the bit lines are shared by the *C.cell* and the *T.cell* to save area, there is a grave danger of a sneak path between nodes (1) and (2). The sneak path is blocked by ensuring that the word lines of the *T.cell* are low throughout the READ operation. In conclusion, the pseudo-static technique permits the implemen-

tation of the RAT as a 3-port memory with 3 single bit lines rather than with 3 bit line pairs with substantial savings in area while avoiding the price of using boosted word lines.

The second reason for using the pseudo-static cell has to do with the REPAIR operation. If a static *C.cell* is used, there would be a lot of contention during a REPAIR operation from the *T.cell* or from the *S.cell*. Firstly, this contention means that the *T.cell*, the *S.cell* and the REPAIR pass transistors (M1 and M2) have to be large for the REPAIR operation to be successful. Secondly, the contention leads to high power dissipation and high peak currents with the attendant inductive noise problems. The need to carry out the REPAIR operation on all the cells of the core (992 cells) at once, compounds the above 2 contention-related problems. In the RAT value cell, the REPAIR operation is carried out by keeping the REFRESH signal low while T2C or S2C is raised high. In this way, the contention between the current and backup parts of the cell is avoided.

Only one of the backup cells (the *T.cell*) interacts with the external world. Dynamic cells are used in the backup section for area and simplicity reasons. (A pseudo-static version had to be used in the current part to avoid the complexity of providing *REFRESH*.) The use of dynamic cells for the backup section also prevents contention during the SAVE operations. The dynamic *S.cell* does not need *REFRESH* since it is guaranteed by the CPU design that its value is used within 1ms. The *REFRESH* for the *T.cell* is initiated by software.

4.4.1.2. Tag Double-CAM-Cell

The schematic for the tag cell is shown in Figure 7. The layout has an area of 135μm x 88μm. The tag cell has 2 data storage elements:

(1) 1 pseudo-static cell with 3 tag comparators for the *C.cell*,

and

(2) 1 dynamic cell with 3 comparators for the *T.cell*

(for branch-prediction support).

The READ/WRITE operations are:

(1) a 1-port (one of the 3 available ports) conventional WRITE
to the *C.cell*, and

(3) a 3-port conventional READ from the *C.cell*.

The associative operations are:

(1) a 3-port tag comparison with the *C.cell*
and the *T.cell*.

The SAVE/REPAIR operations are:

(1) a *current-to-transit* SAVE, and

(2) a *transit-to-current* REPAIR.

The pseudo-static cell was used to implement the tag *C.cell* for the same reasons that justified its use for the value *C.cell*. The tag cell has to be implemented with bit line pairs to allow a complete logic comparison with an external tag. Fortunately, the total area cost is bearable since only 5 CAM tags are needed. The tag CAM cell needs 6 tag comparators so that both the *C.cell* and the *T.cell* can be compared with 3 external interrogative tags at once.

This need for 6 comparators requires a careful choice of the tag comparator to minimize the size of the tag cell and the capacitance of the bit lines. The 3 CAM cell comparators in Figure 8 were considered. The most serious flaw of the Mundy [5] CAM cell comparator is the extra capacitive loading on the BL due to the internal node (1). The extra capacitive loading is non-deterministic, being dependent on the stored data. BL "sees" node (1) only when M1 is ON while BL.bar is loaded by node (1) if M2 is ON. The worst case for BL is when all the M1's in all the comparators in the tag column are ON. This pattern-dependent extra capacitive loading on the BL, which is worse when there is a backup CAM cell, slows down all the operations. Another flaw with the Mundy comparator is the area consumed by the

diode in each comparator.

The Kadota [6] comparator in Figure 8b does not have the area and capacitance problems of the Mundy comparator. In addition, its use of the gate (rather than the drain of the Mundy comparator) for broadcasting the external tag to the cell gives the bit line in Figure 8b less capacitance than the one in Figure 8a. However, the charge-sharing between the match line and the internal nodes (2) and (3) in Figure 8b during a tag match, is the reason for rejecting the Kadota comparator. The CAM cell comparator in Figure 8c is the one used for implementing the tag CAM cell. It does not have the problems of the Mundy and the Kadota comparators

4.4.2. Peripheral Circuits

Figure 9 shows the peripheral circuits.

4.4.2.1. Decoders and MUX's

The MUX's sample the ADDRESS busses and drive the decoder inputs with the address and its complement. The decoders for the READ/WRITE and CAM functions use 2 3-input NAND gates that feed a 2-input NOR gate. Compactness (due to the NAND) is the main reason for choosing this decoder since 186-word (2 X 3 X 31) decoders are needed. The fact that only one control signal is needed is another advantage. This signal also leads to the discharge of the word line whenever the decoder is being precharged. The READ/WRITE decoder is clocked with the signal (SEL.MUX.bar) that is used to sample the ADDRESS bus in the MUX's. Because of the complexity of the CAM timing, the signal (EVAL.CAM.DEC) for evaluating the CAM decoder is different from that (SEL.DIST) for sampling the CAM ADDRESS bus.

The decoder is slower than other more complex versions because of the 4 NMOS transistors in series followed by a pull-up that uses 2 PMOS transistors in series. However, in the case of the

READ/WRITE decoder, the decoder only drives the tag word line and not the entire word line. In the case of the CAM decoder, the decoder output is not highly capacitive since all it does is to precharge the match lines. Consequently, the speed penalty of using this decoder is not severe while the decoder is compact enough to fit 186 decoders to the RAT pitch.

By a very careful layout, the internal node capacitances are made small enough compared to the precharge node capacitance so that the charge-sharing problem is not serious [7].

4.4.2.2. READ-ENABLE Circuitry

The READ-ENABLE circuitry lies between the upper and lower halves of the decoder array. There is one for each port. It is used to qualify the READ signal (READ.signal in the figure). The qualified READ signal is sent to the I/O buffers to conditionally inhibit certain ports during the READ operations. This inhibition may be necessary if an HPSm instruction, for instance, needs to use a literal rather an RAT register entry as an operand (see Chapter 2). The HPSm Instruction Unit uses the VALID bit of the register address to inform the RAT that the READ operation should not take place. A VALID bit of "0" sent into the READ-ENABLE logic ensures that the port does not drive the external bus, while the "0" VALID bit sent into the READ/WRITE decoder ensures that the word line does not rise for the inhibited port.

4.4.2.3. I/O Buffers

Since high bandwidth communication in the HPSm means a lot of busses, it is desirable to make the busses as short as possible to minimize the bus areas and delays. Having each smart memory interact with the busses from only one side is one way to keep the busses short. Therefore, the RAT I/O buffers are designed to interact with the busses in a bidirectional manner while minimizing the pitch-matching problems of fitting 120 (3 X 40) buffers to the RAT pitch. In Figure 9, the VALUE FIELD I/O drives only the upper bit line while the TAG FIELD I/O drives both the tag bit line and its compliment. The

SA (Sense Amplifier Evaluation) signal is used to dis-enable the output buffer during bit line equalization to prevent penetration current from flowing.

4.4.2.4. Sense Amplifier

The need to support multiport access, branch prediction and exception handling, makes the bit lines very long and therefore highly capacitive (at least 2.62pF/bit line), since they have to traverse the backup cells and 6 word lines/register. A sense amplifier is therefore essential for fast access. The sense amplifier must be compact since 120 ports have to be supported. The conventional split-bit line, half-V_{DD} cross-coupled latch sense amplifier in Figure 9 satisfies the sensing requirements. A charge-sharing sense amplifier was considered but rejected. (It is a sense amplifier that has been successfully used in EPROMs and PLAs. This is possible because EPROMs and PLAs only READ and never WRITE.) It was rejected for the RAT because of the inefficiency of doing reset after writing a "1".

The proposed sense amplifier has the following features:

i) a complimentary sense amplifier consisting of NMOS and PMOS cross-coupled devices,

ii) it is placed in the middle of the RAT and divides the memory array into 2 halves: an upper half that is of the same polarity as that of that data from the busses and a lower half that has the opposite polarity, and

iii) a shorting transistor to equalize the precharge potential of the bit lines.

The operation of the sense amplifier is as follows. At the end of the previous cycle, one bit-line half is at V_{DD} and the other half is at GND. The bit line precharge before sensing is initiated by taking the equalizing signal, EQ, high to turn ON the equalizing transistor which shorts the 2 bit line halves together. The charge sharing between the 2 bit line halves results in a precharge that is nearly half V_{DD}. (The extra capacitance on the upper bit line half due to the I/O buffer prevents the precharge potential from

being exactly equal to half V_{DD}. More registers are placed under the sense amplifiers than above as a first order minimization of this capacitance difference.)

The sensing starts by establishing a difference in voltage between the 2 bit line halves. In the case of a READ, this difference is established by raising one of the word lines connected to either the upper half or the lower half. The cell connected to the bit line and the raised word line pulls the bit line low or high depending on the stored data. In the case of a WRITE, the difference is established by the WRITE buffer that drives the upper half bit line either low or high. (The WRITE buffer of the TAG FIELD also drives the complimentary bit line.) After establishing enough difference to account for any noise and voltage offset, the sense amplifier is activated by strobing the SA and SA.bar clocks.

The folded bit line version of this sensing scheme has the advantage of locally canceling the coupled noises due to bit line and word line swings. This version could not be used to implement the RAT sense amplifier because of the backup word lines. SPICE simulations predict and experimental measurements confirm that the noise problem of the open bit line sense amplifier is not severe.

4.4.2.5. SKEW Logic

The SKEW logic in Figure 9 is used to prevent any skew-related hazards by qualifying the EQUALIZATION signal for the TAG and VALUE fields. The signals that clear the TAG and VALUE word lines are inputs to the logic so that no word line is high while the equalization is taking place. The skew-related hazards are discussed in detail in a subsection below.

4.4.3. Clock Generation

The increased functionality of the RAT implies that many signals are needed for its control. At the same time, these signals must be generated reliably, and with minimum skew. Fortunately, most of the signals are non-state dependent signals since most of the events in any cycle are predictable. Hence,

most of the signals have a well-defined pattern each cycle. A 12-stage version of the PLL-based clock generator used by D.K. Jeong [8] is used to generate all the signals for the RAT Test chip. The clock generator block diagram is in Figure 10a. The timing chart showing all the RAT signals are in Figure 10b.

4.4.4. Detailed Operations

For the sake of clarity, a 1-port version of the main circuits are shown in Figure 11. Please refer to the memory architecture in Figure 3, the peripheral circuits in Figure 9, the timing chart in Fig. 10b, and the circuits in Figure 11 for the following discussion.

4.4.4.1. READ Operation

The conventional READ operation is as follows. In ϕ_1 for example, the word lines are discharged while the REFRESH (REF. in the figure) signal is kept high, and the bit lines are equalized. After ϕ_1 goes low, the READ/WRITE decoder conditionally keeps the tag_C.WL1 (tag current word line 1) low or raises it high. The row driver transfers the state of tag_C.WL1 to val_CWL1. The information in the cells are read out to the bit lines, if the word lines are high. The sense amplifiers amplify the bit line signals and the I/O circuitry drives the busses.

4.4.4.2. WRITE Operation

Conventional WRITE is done in the fourth phase. The feedback paths in the C.cells are broken by taking REF. low for the WRITE. After the WRITE, the feedback path is closed again and regeneration through the first and clocked inverters keeps the internal and output nodes consistent.

4.4.4.3. CAM Operation

The CAM decoder in Figure 9 is used to avoid severe power and inductance problems by permitting the precharging of a maximum of 6 instead of 186 match lines. In ϕ_3, the tag bit lines (BL1 and BL1.bar) and current word lines (tag_C.WL1) are discharged while the CAM decoders selectively precharge the current and backup match lines (C.ML1 and B.ML1) of the tag field. After ϕ_3 goes low, the interrogative tag is broadcast to the RAT by writing it on the bit lines. The match lines are conditionally pulled low or left high depending on whether the stored tag mismatches or matches the distributed tag. After enough time has elapsed for the match line to be completely discharged if there is a mismatch, the TAG.ML.to.VAL.WL and TAG.BML.to.VAL.BWL.bar signals (Figure 10b) are raised to transfer the result of the CAM operation to the value field word lines (val_C.WL1 and val_B.WL1).

4.4.4.4. SAVE/REPAIR Operations

Communication between the C.cells and the T.cells of the tag double-CAM-cell and the value triple-cell is done by raising the C2T signal (for SAVE) while the REF. signal is kept high or by raising the T2C signal (for REPAIR) after the REF. signal has been lowered. In the value triple-cell, the T.cell data is saved into the S.cell with the T2S signal and the S.cell data is restored into the C.cell with the S2C signal. By completely overlapping the normal operations, the SAVE/REPAIR functions do not cause any loss of cycles.

4.4.5. SKEW-related Hazards

The RAT needs many signals to control its many functions. Using many signals increases the potential of having skew-related hazards. Only the hazards that cannot be avoided by stretching the clock will be discussed. In Table 1, 5 hazards caused by skews are listed. The first and third hazards where the EQ (equalization) signal arriving too early causes data corruption, are the most severe.

One solution to the skew problem is to use the Mead and Conway [9] approach of putting non-overlapping time between any two signals whose skew can cause a hazard. This approach is the safest but the costliest in terms of speed. In the case of the RAT, it will make the cycle longer by at least 16ns (16% increase). An alternative way is to use the SKEW logic discussed above. The logic does not permit the equalization to take place until all the word lines are low, and the sense amplifier and I/O buffers are turned OFF. Using this approach, the skew problem was adequately solved without an appreciable increase in the cycle time.

4.5. Experimental Results

4.5.1. RAT Test Chip

The microphotograph of the RAT Test Chip is in Figure 12. The RAT memory is in the left while the clock generator is in the right. The chip size is 7.45mm x 7.61mm with 42,068 transistors. It dissipates 1.5W running at 10MHz compared to the estimated power dissipation of at least 0.97W for the chip plus 0.5W for the pads. The detailed RAT microphotograph is in Figure 13. It has a size of 5.37mm x 3.97mm with 38,468 transistors. Its estimated power is 0.51W. The clock generator has a size of 4.78mm x 1.89mm with 1,463 transistors. Its estimated power is 0.46W. The estimated power of the RAT and the clock generator are within the measured power dissipation of the entire chip. The process summary is shown in Table 2.

The chip is mounted in a 108-pin PGA package and has the following pads:

(1) pads for all the RAT addresses,

(2) pads for all the tag ports,

(3) pads for the 3 ports of the READY BIT,

(4) pads for the last bit of the VALUE FIELD, and

(5) pads for the REGISTER 30 *Tag.Current.WL*,

Tag.Current.ML, *Value.Current.WL*,

and *Value.Backup.WL*,

The Clock Generator pads are:

(1) pads for the clock generator PLL and filter,

(2) pads for all the generated clocks, and

(3) pads for the SAVE/REPAIR control signals.

The internal probe pads are:

(1) probe pads for viewing the bit lines of the last VALUE FIELD bit,

(2) probe pads for viewing the internal nodes of all the

clock signals.

4.5.2. Clock Signals

The clock signals generated by the clock generator as seen at the external pads are shown in Figure 14. The waveforms in Figure 14 correspond to those in the timing chart of Figure 11. The observed signals agree with the SPICE simulations and satisfy all the timing requirements for the required 100ns clock cycle.

4.5.3. Basic Operations

Figure 15a depicts the basic operations discussed in section 4.3 for a single port. It shows the experimental results for a 1-port READ-1st-operand (1), READ-2nd--operand (1), Associative-WRITE (1), and WRITE (1) operations as seen at the external pads. The corresponding basic sense amplifier operations with the sense amplifier signals are shown in Figure 15b.

4.5.4. Access Time

The delay from the address input to the last bit by accessing the last register is shown in Figs. 16. The experimental and simulated results agree.

4.5.5. CAM Operation for Current Cells

The detailed CAM operation for the *current* part of the RAT from the CAM address input to the VALUE CURRENT_WORD_LINE (val_C.WL) of the last register are depicted in Figs. 17. These measurements were made at the external pads. The experimental results agree with the simulated results.

Figure 17a shows the general impact of the SELECT_CAM_ADDRESS (SEL.DIST.MUX) and the DISCHARGE_MATCH_LINE (DISCH.ML) signals on the TAG CURRENT_MATCH_LINE (C.ML). Figure 17b is Figure 17a in detail and shows the delay from the CAM decoder input through the CAM decoder to the C.ML.

Figure 17c shows the general impact of the EVALUATE_CAM_DECODER (EVAL.CAM.DEC) and the DISCH.ML signals on the C.ML. Figure 17d is Figure 17c in detail and depicts the delay from the EVAL.CAM.DEC signal to the C.ML.

Figure 17e shows the delay from the DISCH.ML signal to the C.ML.

Figure 17f shows the general impact of the TAG_CURRENT_MATCH_LINE_to_VALUE_CURRENT_WORD_LINE (TAG.ML.to.VAL.WL) signal on the val_C.WL for a tag MATCH scenario. Figure 17g shows Figure 17f in detail. It depicts the delay from the TAG.ML.to.VAL.WL signal to the val_C.WL.

Figure 17h and Figure 17i portray the impact of the interrogative tag for MATCH and MISMATCH scenarios respectively. The stored tag contains a "1". In both cases, the match line is

precharged but discharged only in the case of Figure 17i when the tag bit line goes to "0" - a mismatch since "1" is the stored tag. (The observed dip in the tag bit line waveform is due to the equalization operation as seen at the external pad.) Finally, the detail of Figure 17i is shown in Figure 17j.

4.5.6. CAM Operation for Backup Cells

The detailed CAM operation for the *backup* part of the RAT from the CAM DECODER output (PRECH.ML) to the VALUE_BACKUP_WORD_LINE (val_B.WL) of the last register are depicted in Figs. 18. Since the CAM operations of the *current* and *backup* sections of the RAT are similar, the similarities are not duplicated in Figs. 18. All measurements were made at the external pads. The experimental results agree closely with the simulated results.

Figure 18a shows the general impact of the TAG_BACKUP_MATCH_LINE_to_VALUE_BACKUP_WORD_LINE.bar (TAG.BML.to.VAL.BWL.bar) signal on the val_B.WL. It is clear from the figure that the val_B.WL goes high only during phase 3 when the CAM operation takes place, as it should. The val_B.WL must be kept low at all other times to prevent any corruption of the data in the *T.cells* of the VALUE FIELD through the sneak path in Figure 5 mentioned in subsection 4.4.1.1. above. Figure 18b and Figure 18c show the rise and fall delays of the val_B.WL relative to the TAG.BML.to.VAL.BWL.bar signal.

4.5.7. SAVE/REPAIR Interaction with CAM Operation

The SAVE/REPAIR operations for the VALUE FIELD are difficult to demonstrate on an oscilloscope screen. The TEKTRONIX DAS equipment was used to verify these operations. The SAVE/REPAIR operations for the TAG FIELD are depicted by showing an interesting interaction of these operations with the associative operation.

4.5.7.1. SAVE: C2T

Figs. 19 show the impact of the *current_to_transit* (C2T) SAVE on the *backup* MATCH operation. The tag BL is toggled cycle by cycle so that the data written into the *current* tag cell in one cycle is MATCHed by the distributed tag during that cycle. Therefore, the experiment is such that the *current* tag always MATCHes the interrogative tag as shown in Figure 19a and Figure 19b.

In Figure 19a, the *backup* tag always MATCHes the external tag because the C2T signal goes high in every cycle to transfer the updated state of the *current* tag into the *backup* cell. On the other hand, Figure 19b shows the case where the *backup* tag MATCHes the external tag in one cycle but misMATCHes it in the next cycle. The MATCH only takes place when the C2T signal goes high in that cycle.

In the context of the HPSm microarchitecture (see Chapter 2), this scenario arises when a different tag is written in phase 4 just after a branch prediction (that leads to the C2T SAVE) has been made. Then, in the next cycle a CAM operation is performed where the distributed tag MATCHes the *current* tag but misMATCHes the *backup* tag because no C2T SAVE is done in the cycle. The tag alteration in phase 4 has the effect of making the state of the machine before the BRANCH point different from that after the BRANCH point. This interaction between the SAVE and CAM functions illustrates a very interesting feature of the RAT : it maintains 2 independent states of the machine in the TAG FIELD and distribution of results can be made to update the 2 states of the machine in parallel.

4.5.7.2. REPAIR: T2C

Figs. 20 show the opposite of the above subsection. The narrower pulse of the CURRENT_MATCH_LINE (CML) show a tag misMATCH (since the CML is high for just a while before being discharged by the CAM operation). The CWL never rises if there is a tag misMATCH. The data in the *backup* cell is left undisturbed throughout the experiment and it always MATCHes the exter-

nal tag. The tag BL is toggled every cycle so that the data written into the *current* tag cell in a given cycle does not MATCH the distributed tag during the next cycle unless the *transit_to_current* (T2C) signal goes high before the CAM operation.

Figure 20a shows the case where the T2C REPAIR takes place in every cycle; both the *current* and *backup* tags always MATCH the distributed tag. The *backup* tag MATCHes the external tag because they both contain "0", while the *current* tag MATCHes because a REPAIR is done every cycle. In Figure 20b, the *current* tag only MATCHes after a REPAIR operation has taken place.

In the context of the HPSm microarchitecture (see Chapter 2), this scenario arises when the distributed tag MATCHes the *backup* tag because it belongs to the state of the machine before the BRANCH point. The *current* tag would not MATCH the interrogative tag because it belongs to a different state; therefore, it would only MATCH after the REPAIR operation (that restores the former state of the machine) is performed.

4.5.8. GROUND Bounce

The RAT design requires 6 signals (REFRESH and REFRESH.bar, C2T, T2C, T2S, and S2C) that go to every data cell. These signals are therefore highly capacitive because their capacitance increases in proportion to N^2 where N is the dimension of the memory as compared to the usual peripheral signals whose capacitances follow a linear law. The capacitances of these signals and their associated peak currents are shown in Table 3.

The fact that the capacitances follow a square law is one of the unusual features of the RAT and illustrates one of the costs of smart memories. Because driving these signals would need high peak currents, it is important to know the impact of the high capacitances on the stability of the chip ground.

A 3.5Ω resistor was connected between the chip ground and the oscilloscope ground for the experiment in Figure 21a. Figure 21a shows that the chip ground potential varies over a 1.18V range. It also

shows that the greatest swing occurs when 4 of these high-capacitance signals (S2C, T2C, REFRESH, and REFRESH.bar) are switching at the same time. (However, any skew between the signals will reduce the peak current problems since the total peak current won't be the sum of all the peak currents caused by driving the various signals.) It must be pointed out that Figure 21a is the worst case scenario since the HPSm microarchitecture does not allow 2 REPAIRs (T2C and S2C) at once. Yet, it is shocking to observe that the GROUND bounces over about a 1.2V range compared to about 0.5V that will be considered desirable. The peak current of 232mA in Figure 21 agrees closely with the calculated values in Table 3.

4.5.9. Skew-Related Hazards

Figs. 22a and 22b illustrate one of the hazards discussed in an earlier section. They show that the unqualified EQ signal partially overlaps the VALUE FIELD and the TAG FIELD WORD LINEs (VAL.WL and TAG.WL). This skew would have caused a hazard (the equalized bit line corrupting the stored data while the WLs are high) without the correction mentioned above. To show that the correction is successful, consider the case in Figure 22c. Here, the qualified EQ signal rises only after the VAL.WL has gone low because the unqualified EQ signal that is used to CLEAR the VAL.WL is also fed into the logic that produces the qualified EQ signal.

4.6. Conclusions

RAT, a smart data memory that is intended for aiding the HPSm data flow CPU in controlling its out-of-order execution was described in the foregoing paragraphs. The circuit design problems that result from its "smartness" were clearly identified and solutions were developed. Experimental results from a test chip confirm that the employed circuit design techniques were adequate.

References

[1] Gregory A. Uvieghara, Y. Nakagome, D. K. Jeong, D. A. Hodges, "An On-Chip Smart Memory for A Data Flow CPU," in *1989 Symposium of VLSI Circuits Digest of Symposium Papers*, May 1989.

[2] R.W. Sherburne et. al., "A 32 bit NMOS microprocessor with a large register file," *IEEE J. Solid-State Circuits, vol. SC-19, no. 5, pp. 682-689.*

[3] R.G. Stewart and A.G.F. Dingwall, "16K CMOS/SOS asynchronous static RAM," in *ISSCC Dig. Tech. Papers*, Feb 1985, pp. 44-45.

[4] K.J. O'Connor, "The Twin-Port Memory Cell," *IEEE J. Solid-State Circuits, vol. SC-22, no. 5, pp. 712-720,* October 1987.

[5] J.L Mundy et. al., "Low-Cost Associative Memory", *IEEE J. Solid State Circuits*, vol. SC-7, no.5, pp. 364-369, October 1972.

[6] H. Kadota et. al., "An 8-kbit Content-Addressable and Reentrant Memory", *IEEE J. Solid State Circuits*, vol. SC-20, no.5, pp. 951-956, October 1985.

[7] N.H. Weste and K. Eshragrian, *PRINCIPLES OF CMOS VLSI DESIGN - A Systems Perspective*, Addison Wesley, 1985.

[8] D.K. Jeong et. al., "Design of PLL-Based Clock Generation Circuits," *IEEE J. Solid State Circuits*, vol. SC-22, no. 2, pp. 255-261, April 1987.

[9] Mead, C. and Conway, L., *Introduction to VLSI Systems*, Reading, MA:Addison-Wesley, 1980.

Data-Path Current Design

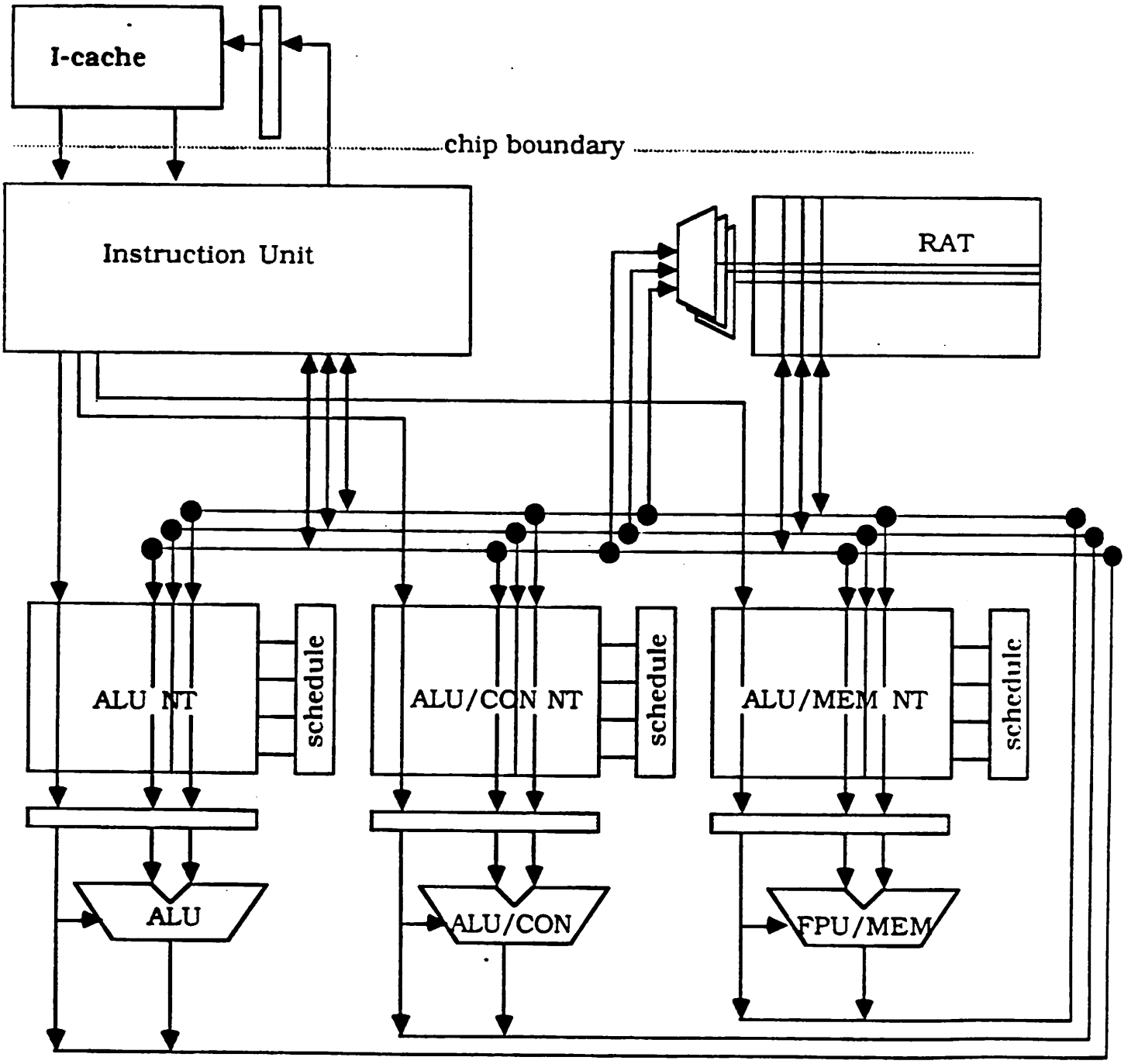


Figure 1

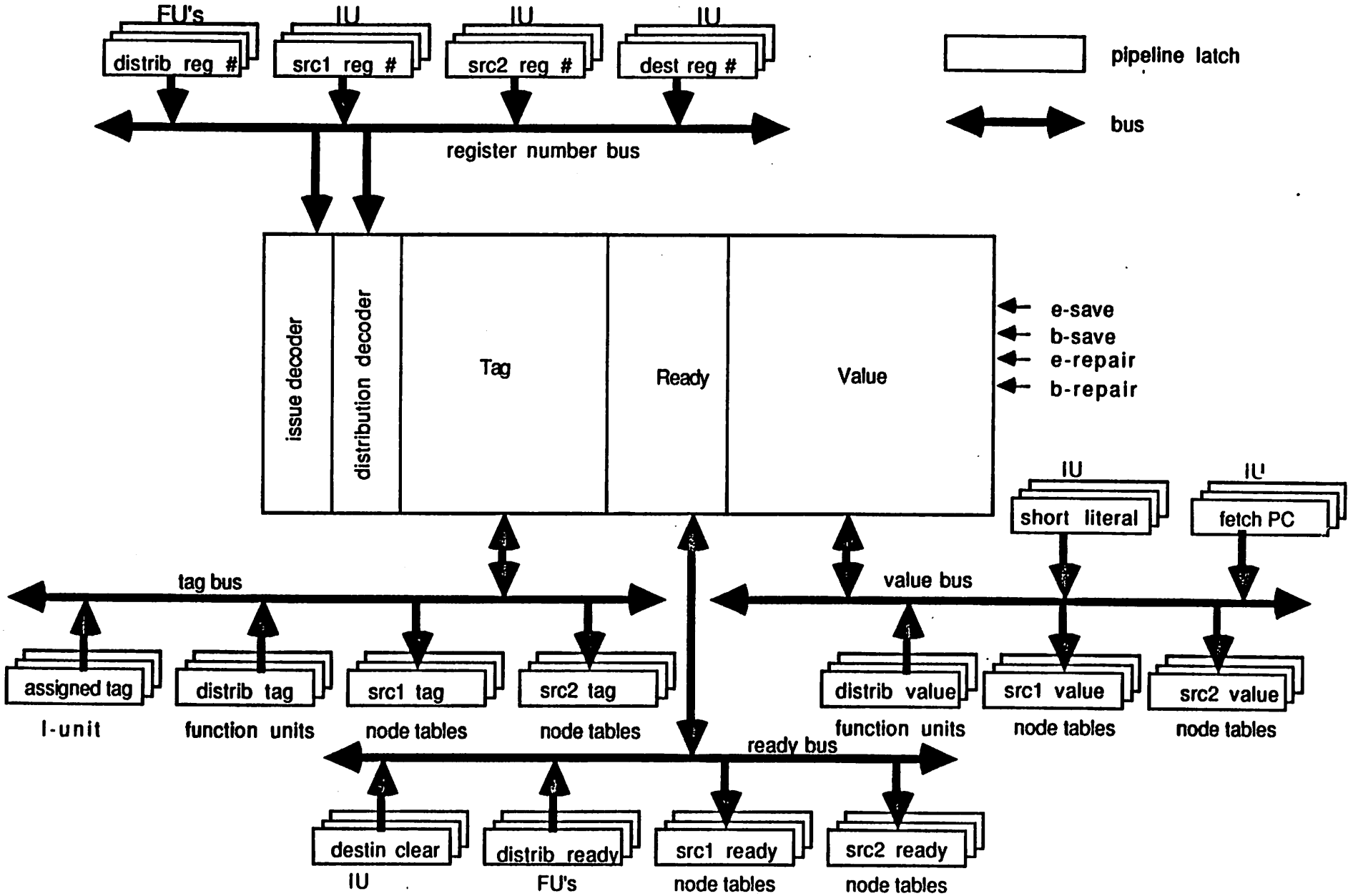


Fig. 2 : RAT with Global Busses and Signals

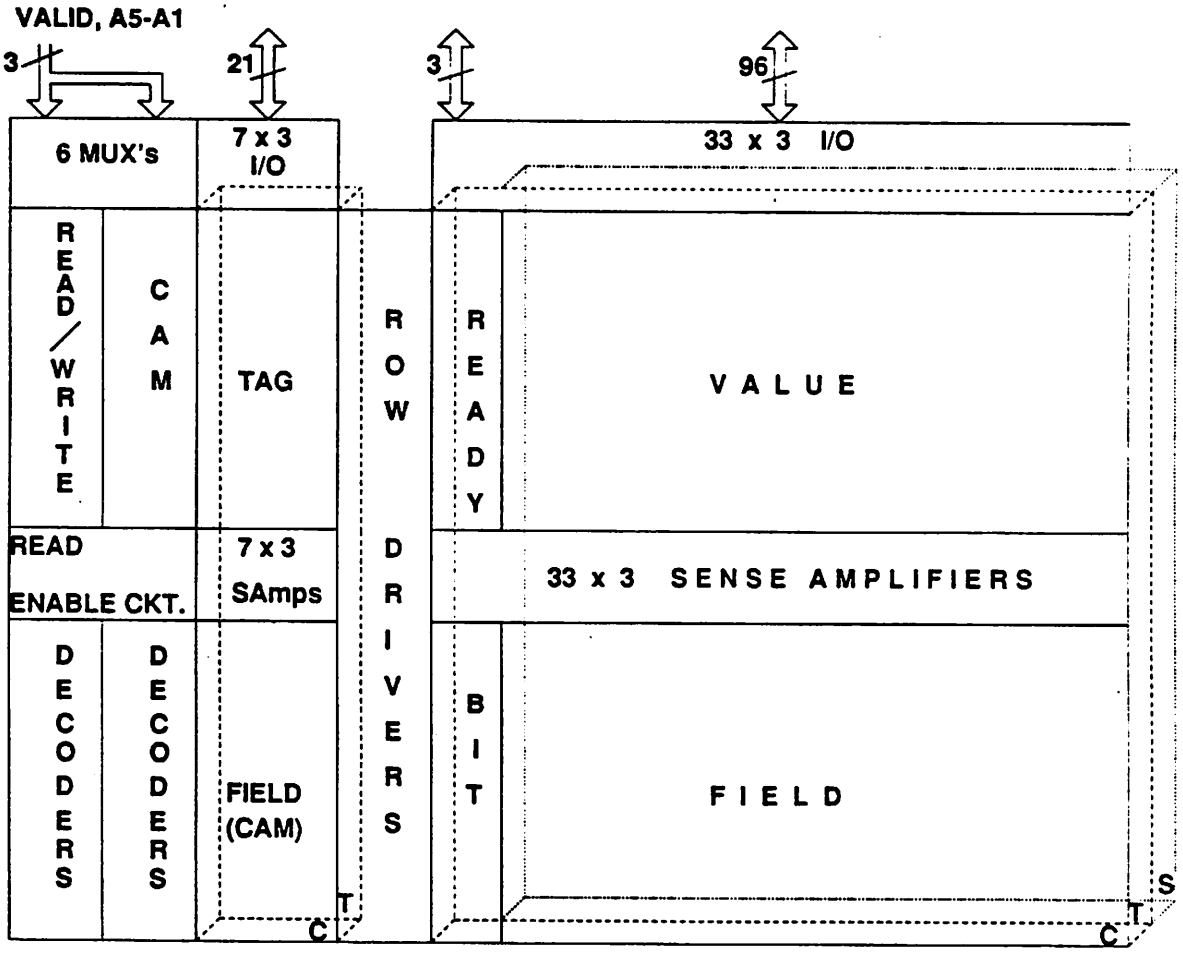


Fig. 3: RAT Memory Architecture

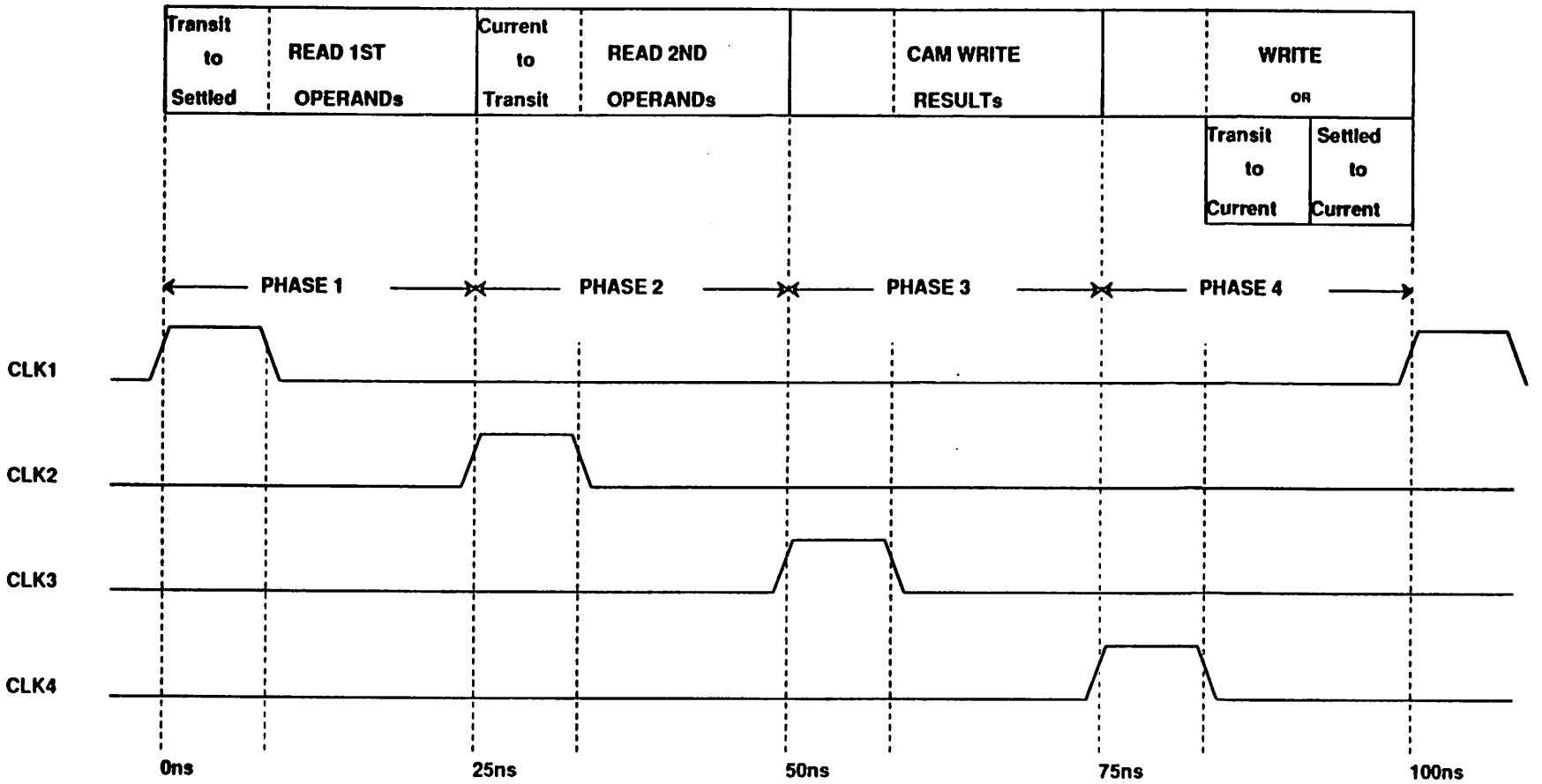


Fig. 4: RAT Timing Sequence

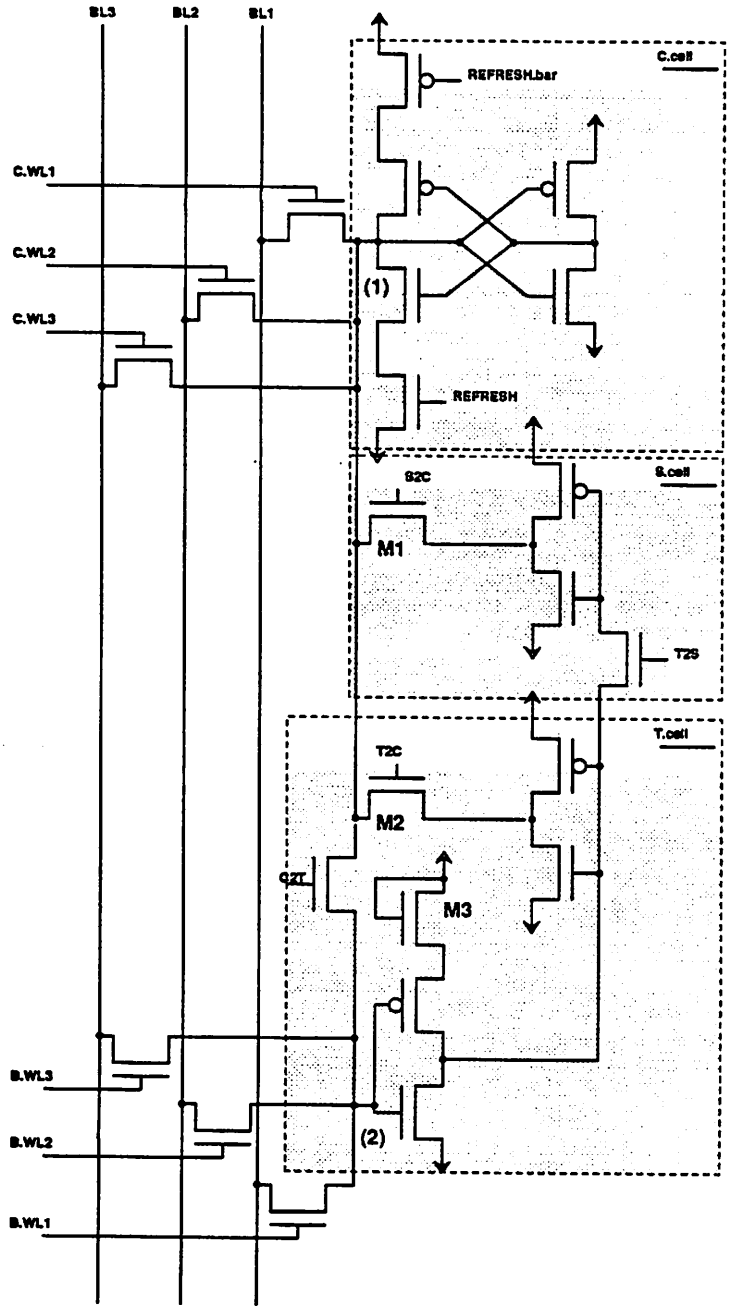


Fig. 5: Value "Triple-Cell" Schematic

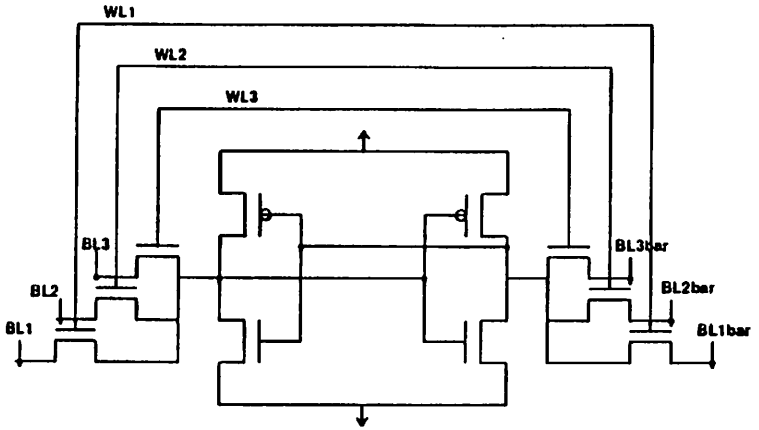


Fig. 6a: A Conventional Triple-port Differentially Accessed Cell

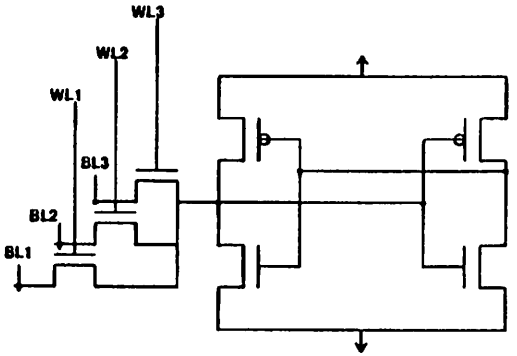


Fig. 6b: A Triple-port Single-ended Access Cell

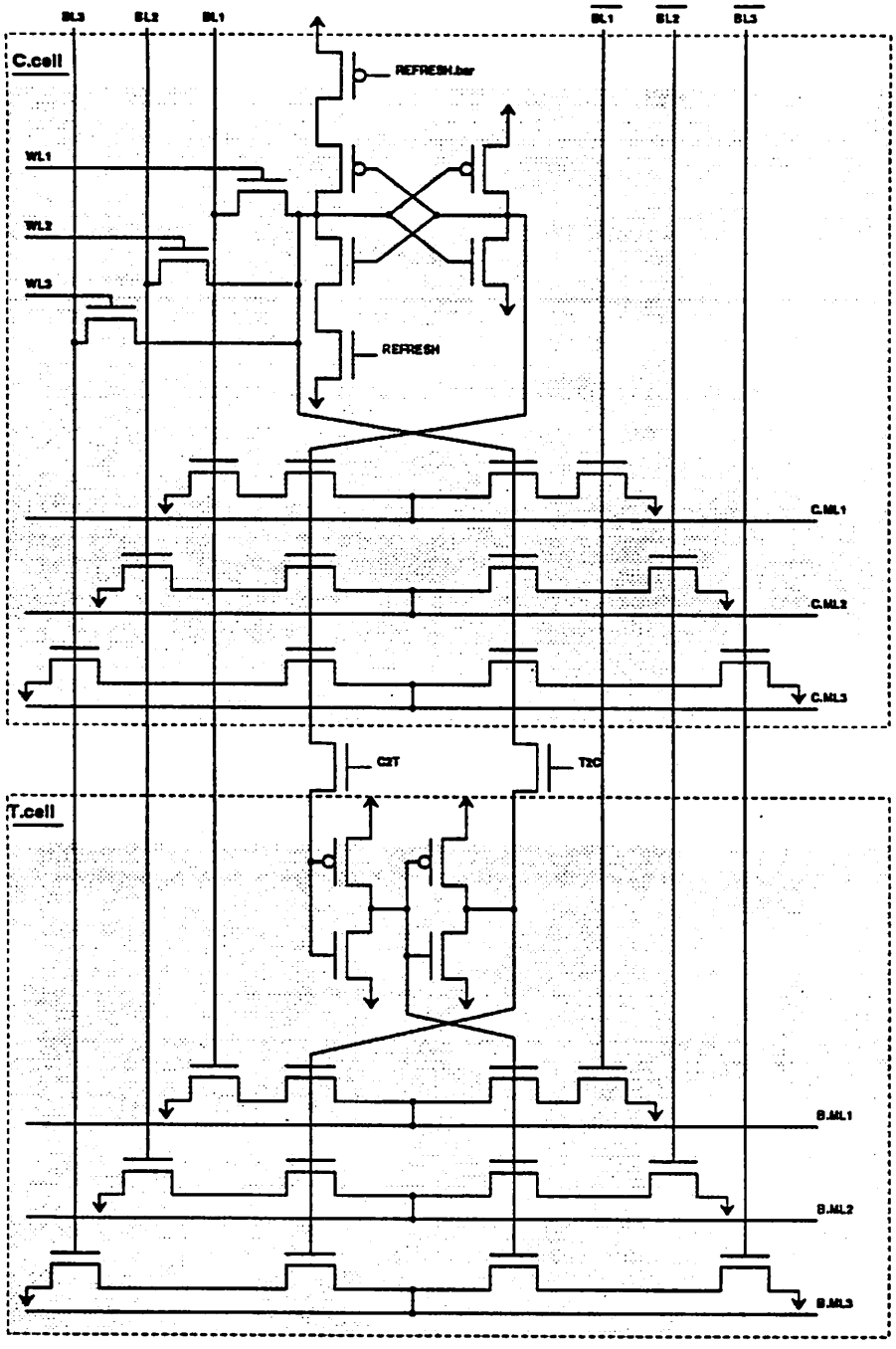


Fig. 7: Tag "Double-CAM" Cell Schematic

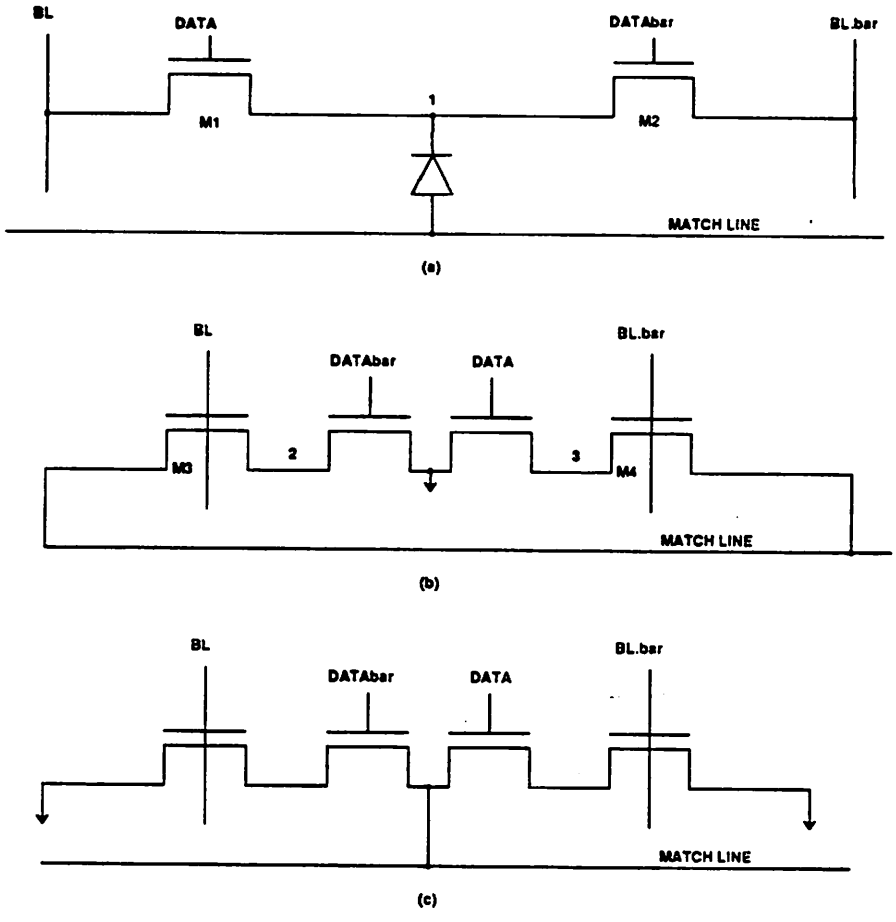


Fig. 8: CAM Cell Comparators:

(a) MUNDY CAM Cell Comparator.

(b) KADOTA CAM Cell Comparator.

(c) PROPOSED CAM Cell Comparator.

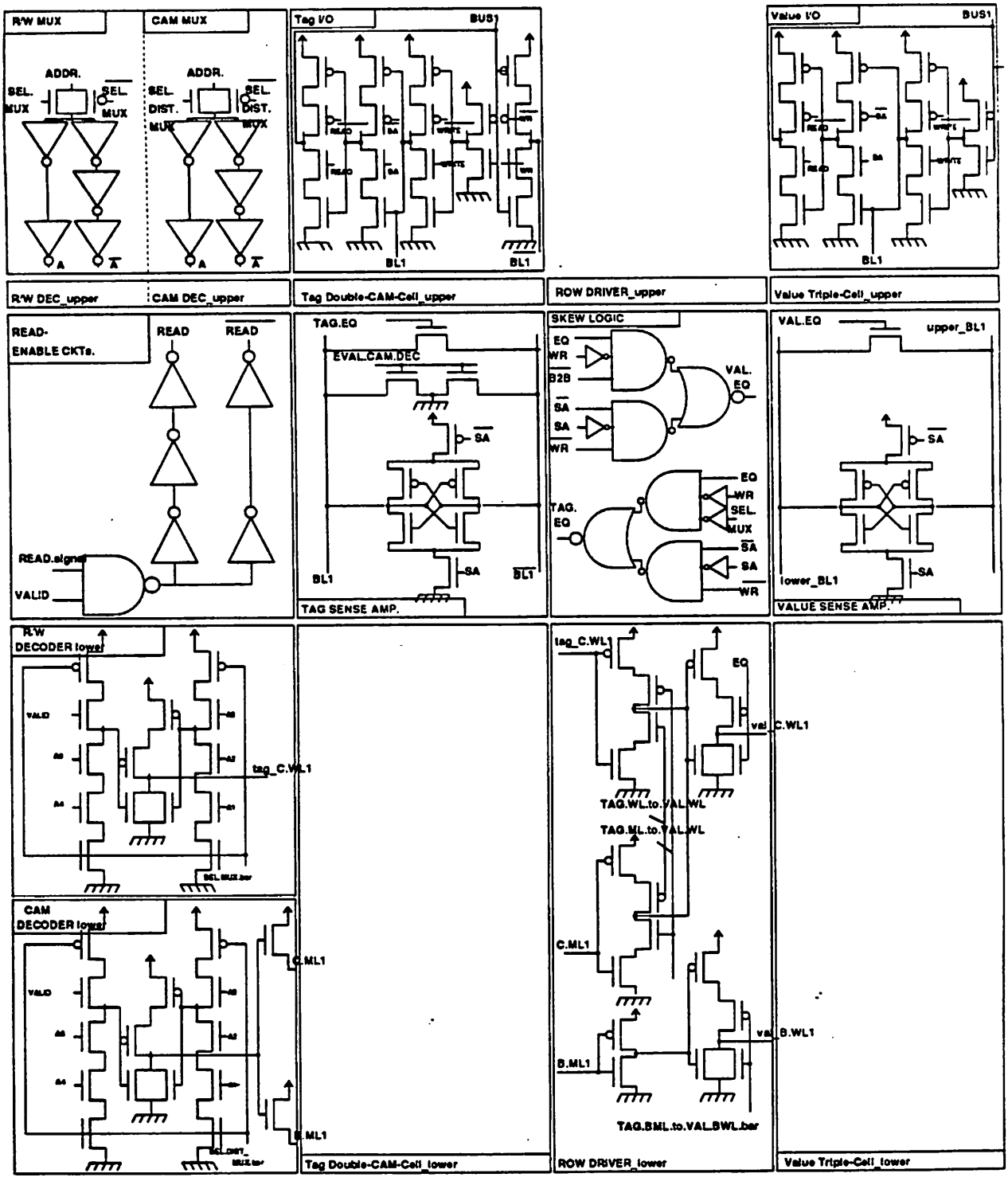


Fig. 9. Peripheral Circuits

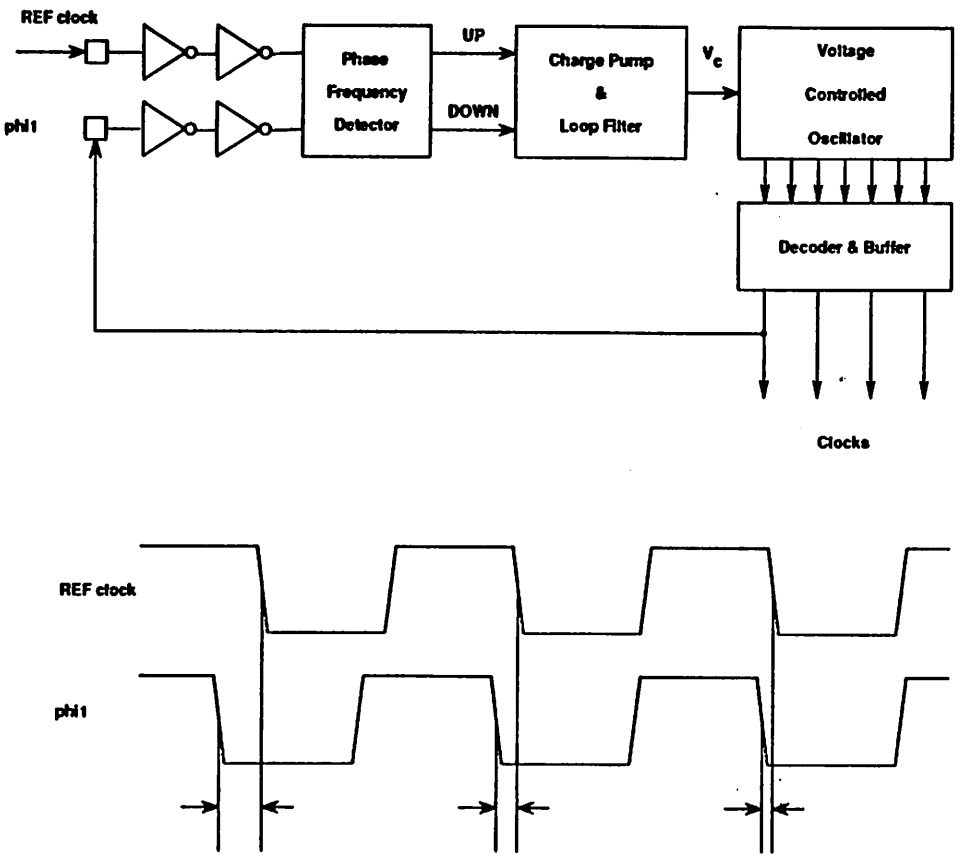


Fig. 10a: PLL-Based Clock Generator Block Diagram

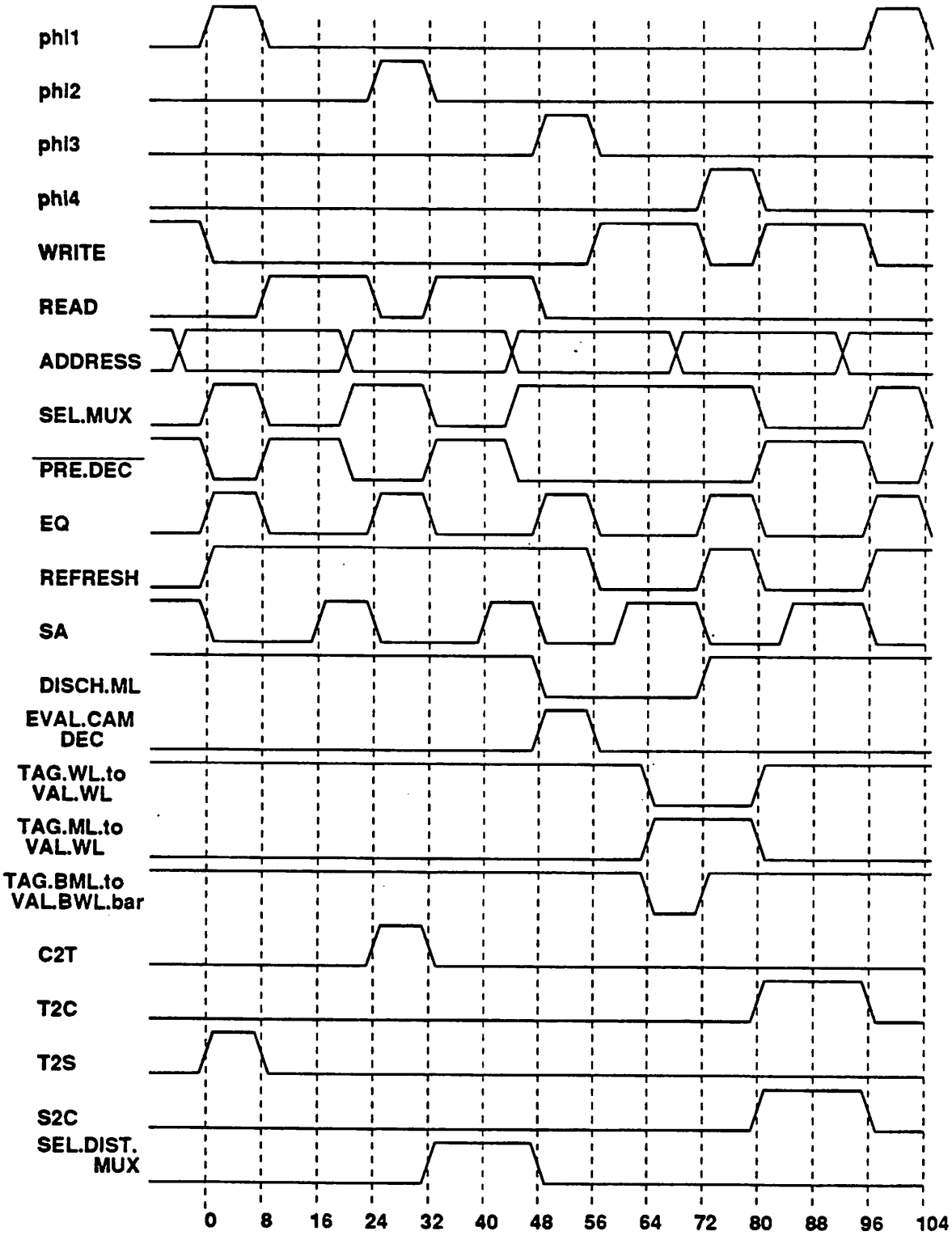


Fig. 10b: RAT Timing Chart

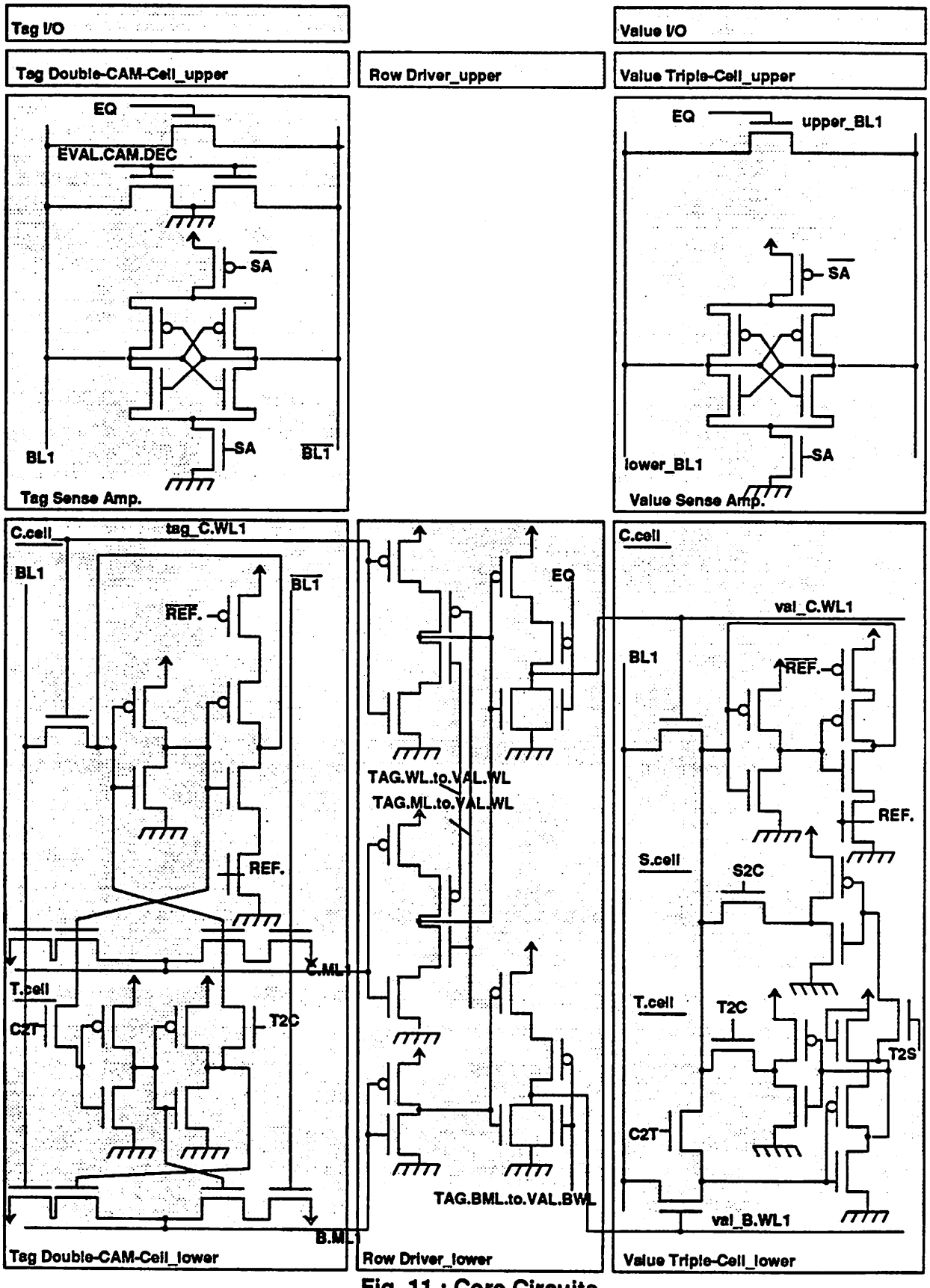


Fig. 11 : Core Circuits

Table 1 : Skew-Related Hazards

Skew	Scenario	Hazard
1	SEL.MUX.bar falls too late. EQ rises too early.	Not enough time for driving the word lines low. Data is corrupted.
2	REFRESH rises too early. SEL.MUX.bar falls too late. EQ rises too early.	BL and BL.bar equalized away from half-VDD. Read noise margin reduced.
3	EQ rises too early. DISCH.ML rises too late. TAG.BML.to.VAL.BWL.bar rises too late.	Not enough time for driving the backup word lines low. VALUE FIELD backup data corrupted.
4	WRITE falls too late. EQ rises too early.	BL and BL.bar equalized away from half-VDD. Read noise margin reduced.
5	EQ rises too early. SA falls too late. SA.bar rises too early.	BL and BL.bar equalized away from half-VDD. Read noise margin reduced.

Fig. 12 : RAT Test Chip Microphotograph

Chip Size = 7.45mm x 7.61mm
RAT Size = 5.37mm x 3.97mm
Clock Generator Size = 4.78mm x 1.89mm

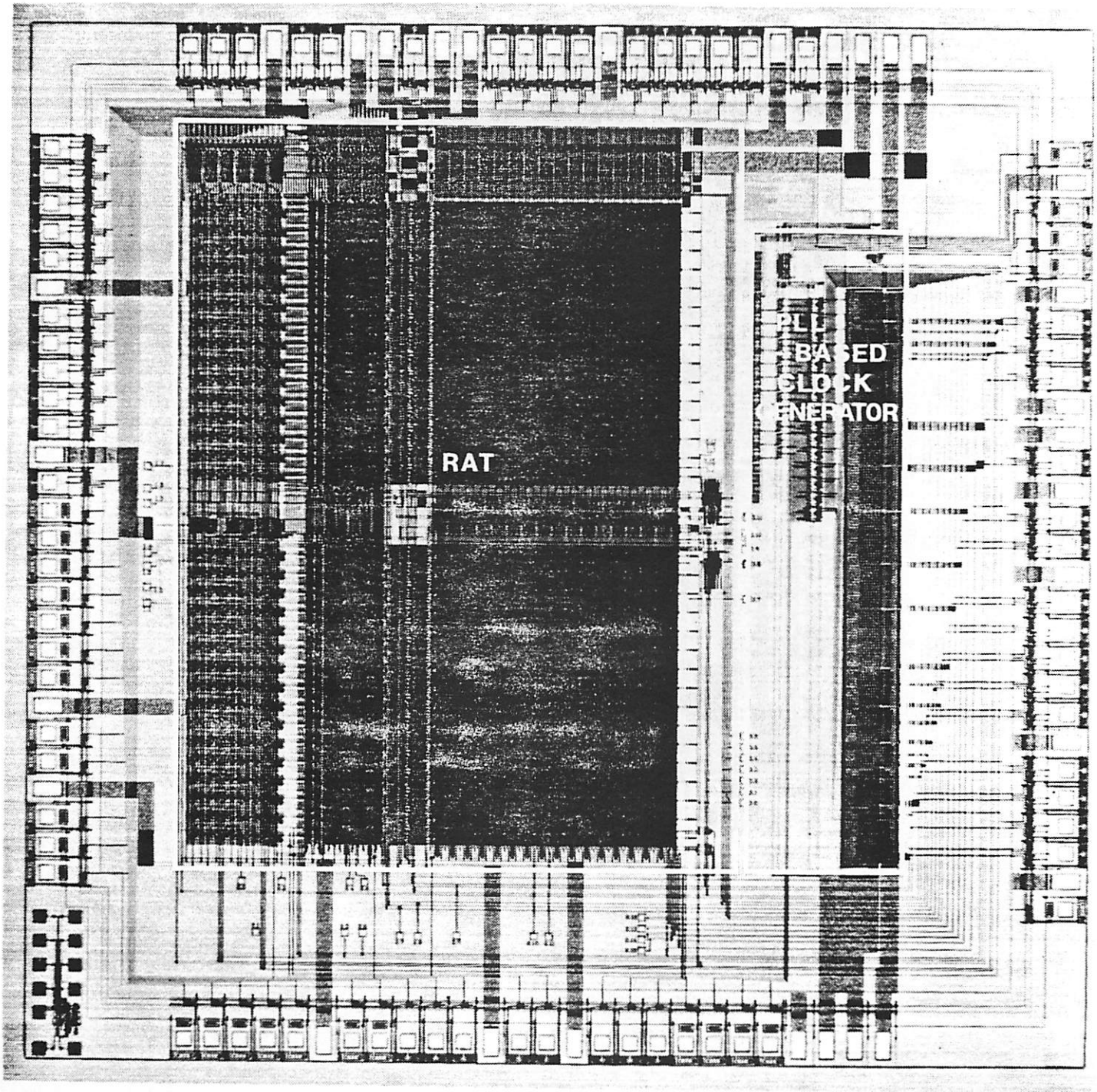


Fig. 13 : RAT Microphotograph

Die Size = 5.37mm x 3.97mm

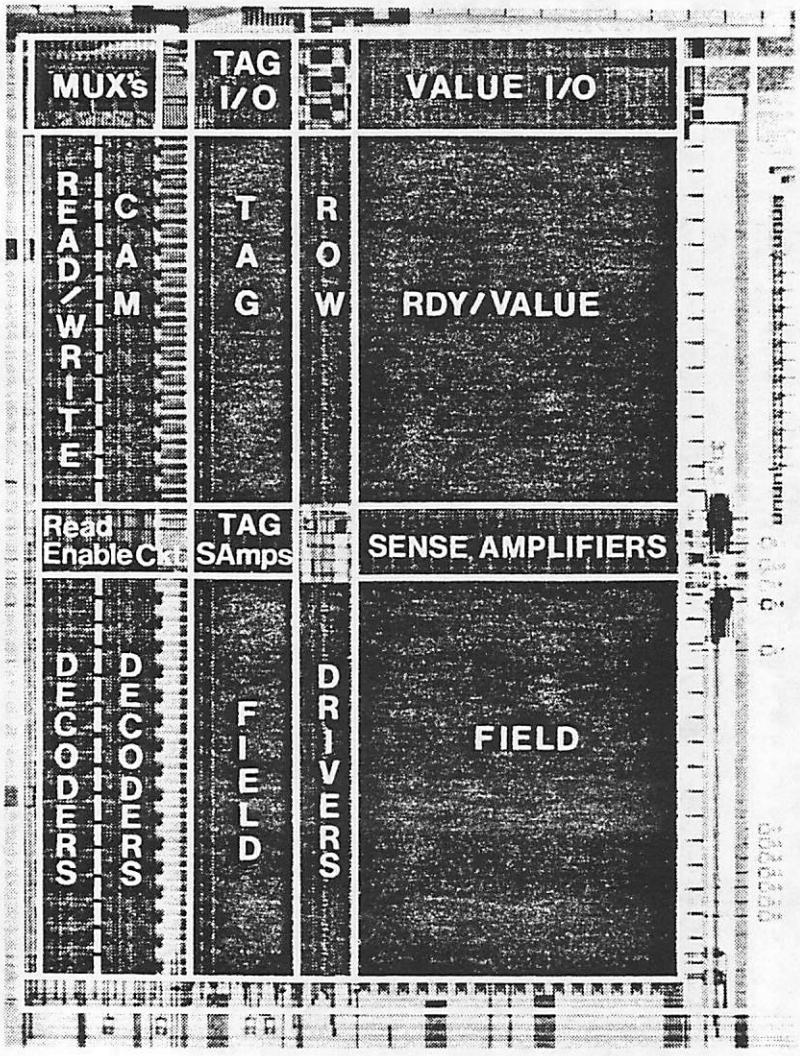


Table 2 : Process Summary

Parameter	CMOS	NMOS	PMOS
Technology	Scalable CMOS, N-Well, Double Metal, Single Polysilicon		
Lambda	0.8um		
Gate Length(Drawn)	1.6um	1.6um	1.6um
Gate Length(Effective)		1.154um	1.513um
Gate Oxide Thickness		24.1nm	24.1nm
VT		0.774V	-0.873V

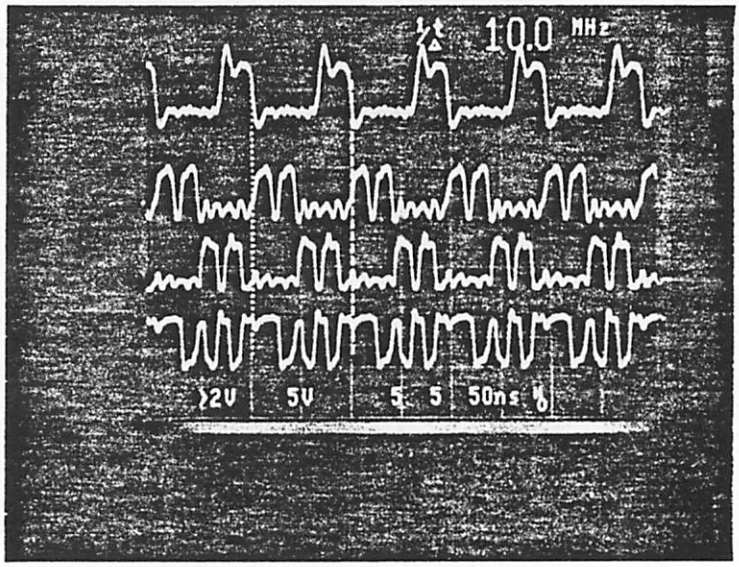
Fig. 14 : Generated Clock Signals

REFERENCE_CLOCK

WRITE

READ

SEL.MUX



EQ

REFRESH

SA

DISCH.ML

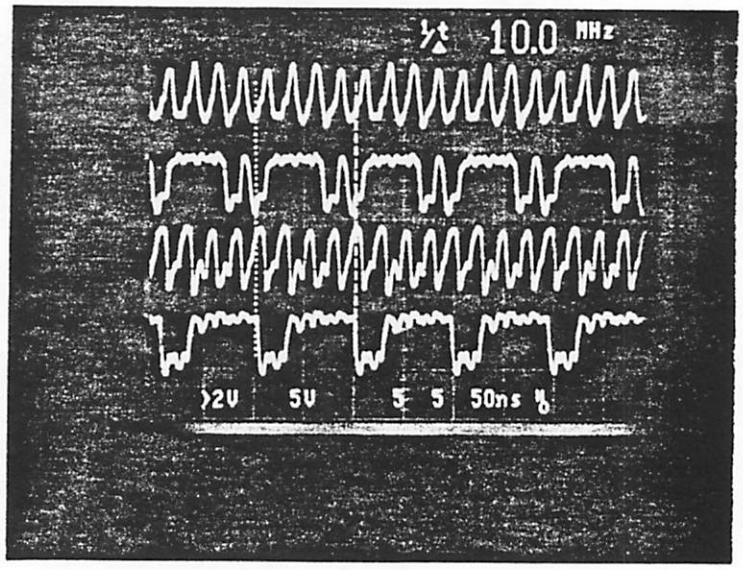


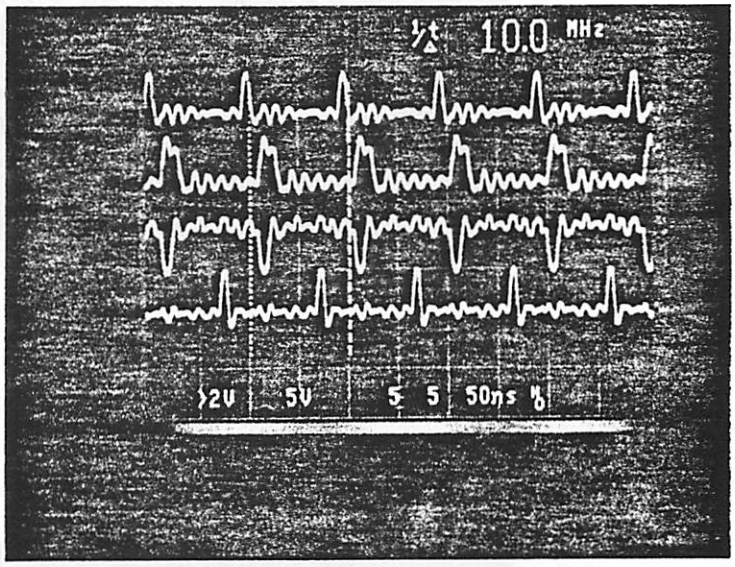
Fig. 14 : Generated Clock Signals

EVAL.CAM.DEC

TAG.ML.to.VAL.WL

TAG.BML.to.VAL.BWL

C2T

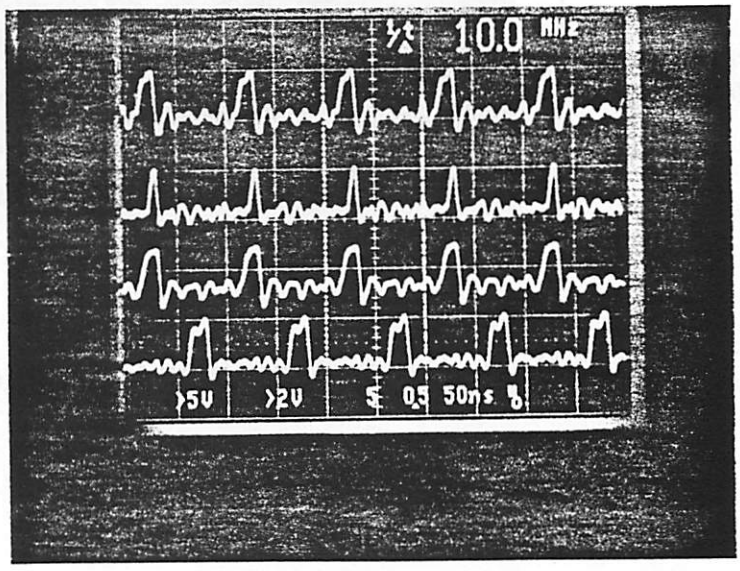


T2C

T2S

S2C

SEL.DIST.MUX



SA

WRITE

READ

BL(external pad)

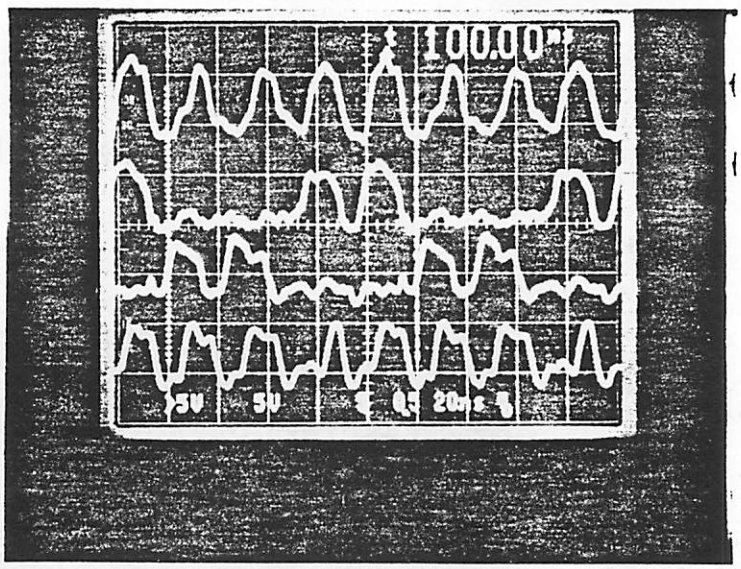


Fig. 15a: Basic Operations

WRITE

READ

EQ

SA

BL (internal pad)
BL.bar (internal pad)

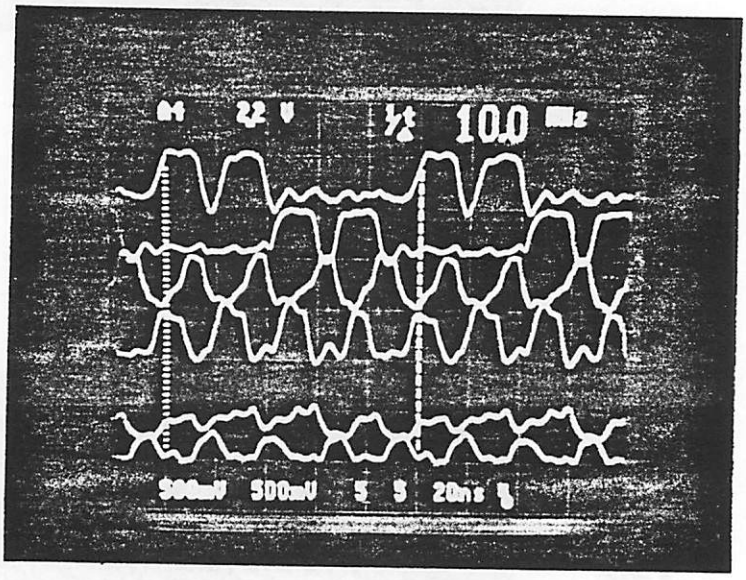


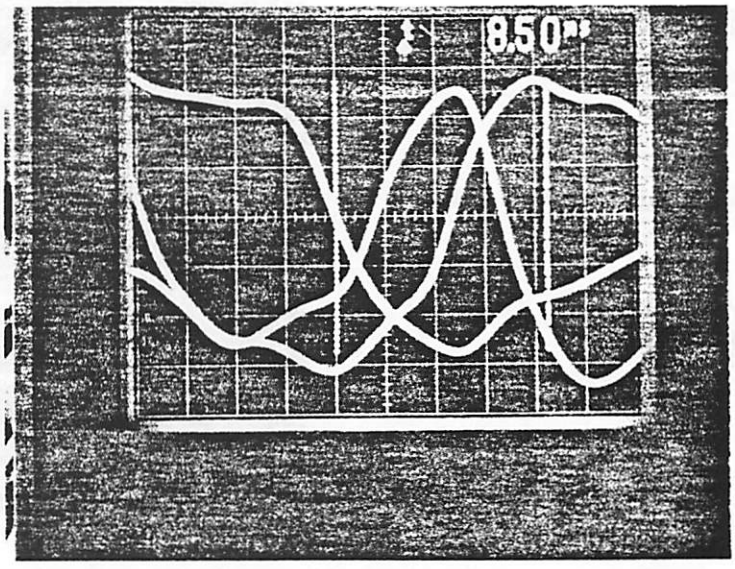
Fig. 15b: Basic Sense Amplifier Operations

SEL.MUX

LAST.WL(external pad)

LAST.BL.bar(external pad)

(1V/Vert. Div.)
(2ns/Hor. Div.)



Path	SEL.MUX to LAST.WL	LAST.WL to LAST.BL.bar	SEL.MUX to LAST.BL.bar
Simulated	4ns	5ns	9ns
Measured	4.5ns	4.0ns	8.5ns

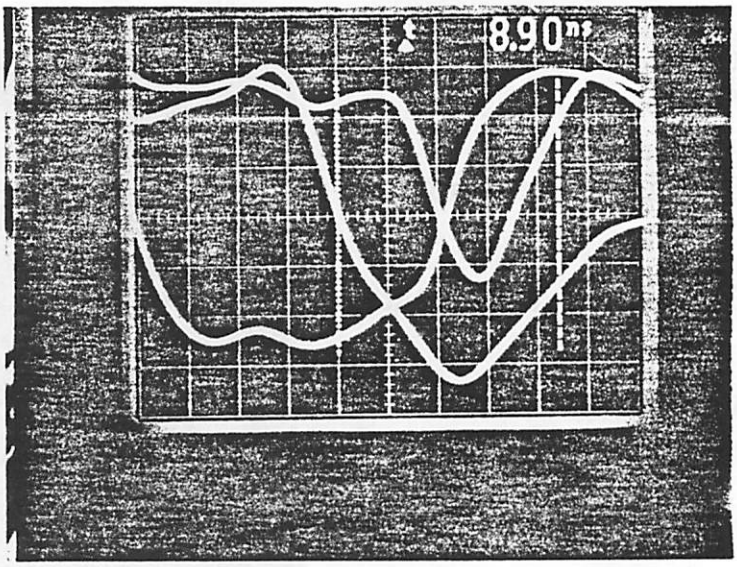
Fig. 16a: ADDRESS_to_LAST.BL Delay (READ "1")

LAST.BL.bar(external pad)

SEL.MUX

LAST.WL(external pad)

(1V/Vert. Div.)
(2ns/Hor. Div.)



Path	SEL.MUX to LAST.WL	LAST.WL to LAST.BL.bar	SEL.MUX to LAST.BL.bar
Simulated	4ns	5ns	9ns
Measured	4.2ns	4.7ns	8.9ns

Fig. 16b: ADDRESS_to_LAST.BL Delay (READ "0")

SEL.DIST.MUX

PRECH.ML(external pad)
(CAM Decoder Output)

C.ML(external pad)

DISCH.ML

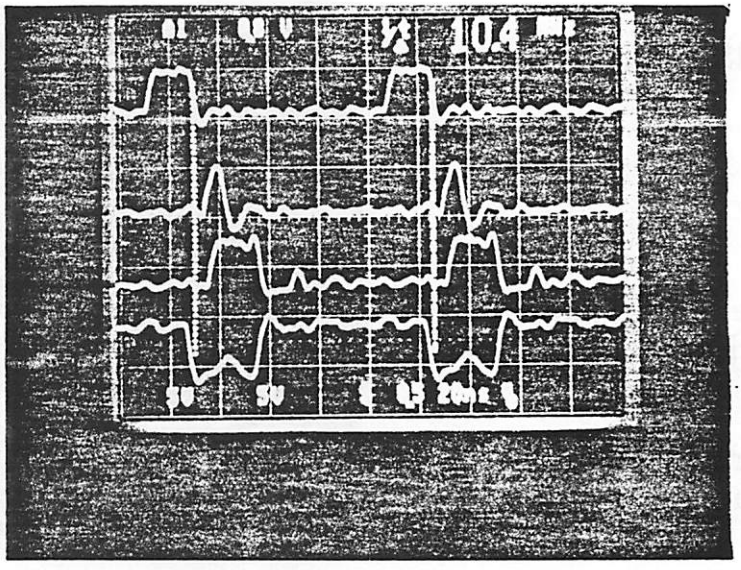
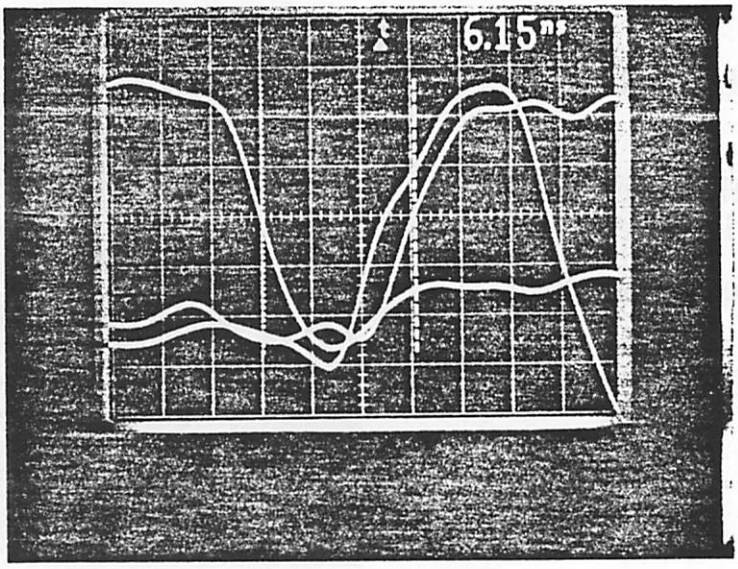


**Fig. 17a: CAM ADDRESS_to_CURRENT MATCH LINE Delay (MATCH)
(Basic Operation @ 10.4Mhz)**

SEL.DIST.MUX

C.ML(external pad)
PRECH.ML(external pad)

(1V/Vert. Div.)
(2ns/Hor. Div.)



Path	SEL.DIST.MUX to C.ML
Simulated	5.5ns
Measured	6.15ns

Fig. 17b: CAM ADDRESS_to_CURRENT MATCH LINE Delay (MATCH) (Detailed Operation @ 10.4Mhz)

EVAL.CAM.DEC

PRECH.ML(external pad)
(CAM Decoder Output)

C.ML(external pad)

DISCH.ML

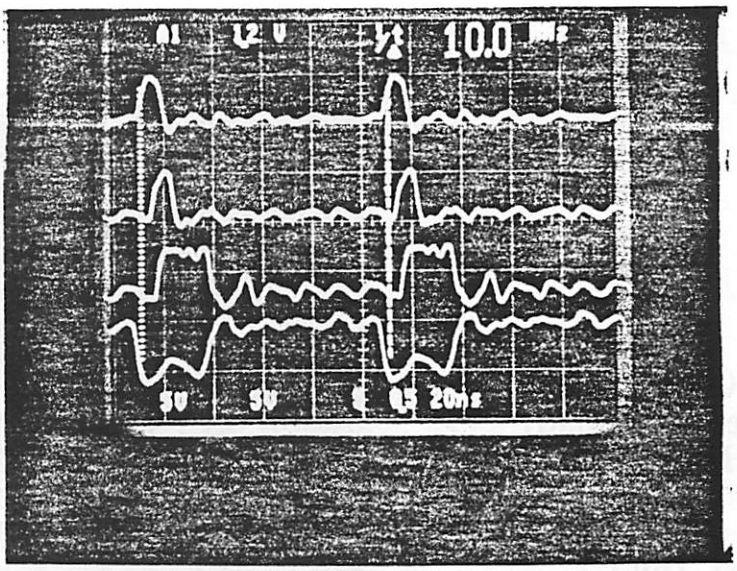
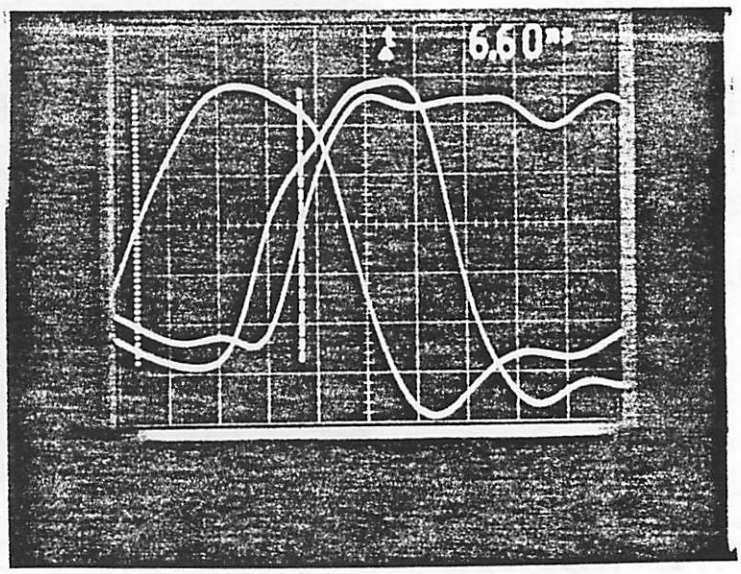


Fig. 17c: EVALUATE CAM DECODER_to_CURRENT MATCH LINE Delay (MATCH)

(Basic Operation)

EVAL.CAM.DEC
C.ML(external pad)
PRECH.ML(external pad)

(1V/Vert. Div.)
(2ns/Hor. Div.)



Path	EVAL.CAM.DEC to C.ML
Simulated	5.2ns
Measured	6.6ns

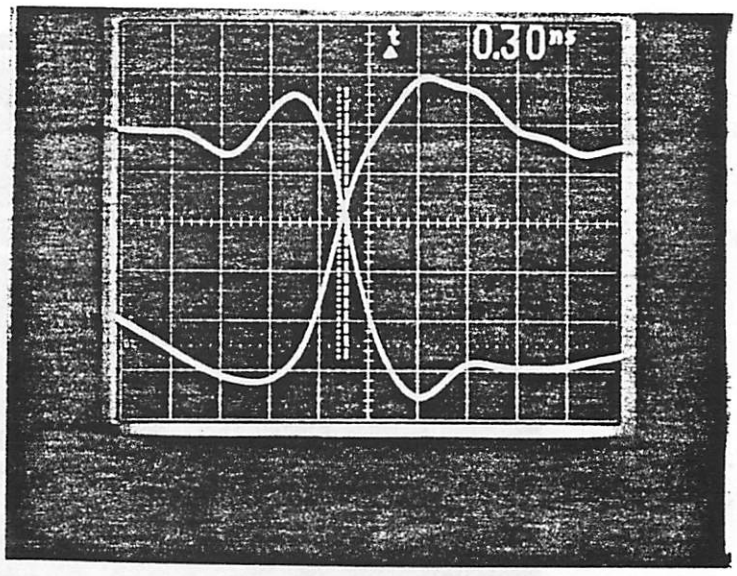
Fig. 17d: EVALUATE CAM DECODER_to_CURRENT MATCH LINE Delay (MATCH)

(Detailed Operation)

C.ML(external pad)

DISCH.ML

(1V/Vert. Div.)
(2ns/Hor. Div.)



Path	DISC.ML to C.ML
Simulated	0.5ns
Measured	0.3ns

Fig. 17e: DISCHARGE MATCH LINE_to_CURRENT MATCH LINE Delay (MATCH)

(Detailed Operation)

C.ML(external pad)

TAG.ML.to.VAL.WL

val_C.WL

(5V/Vert. Div.)
(20ns/Hor. Div.)

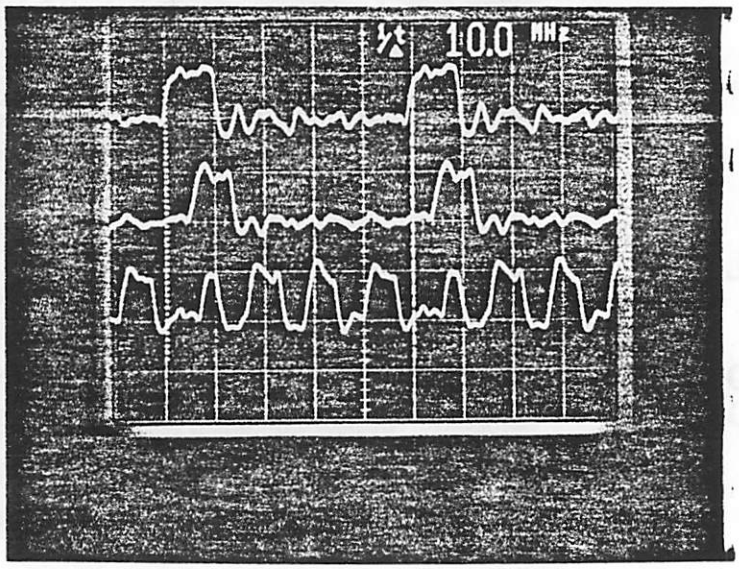
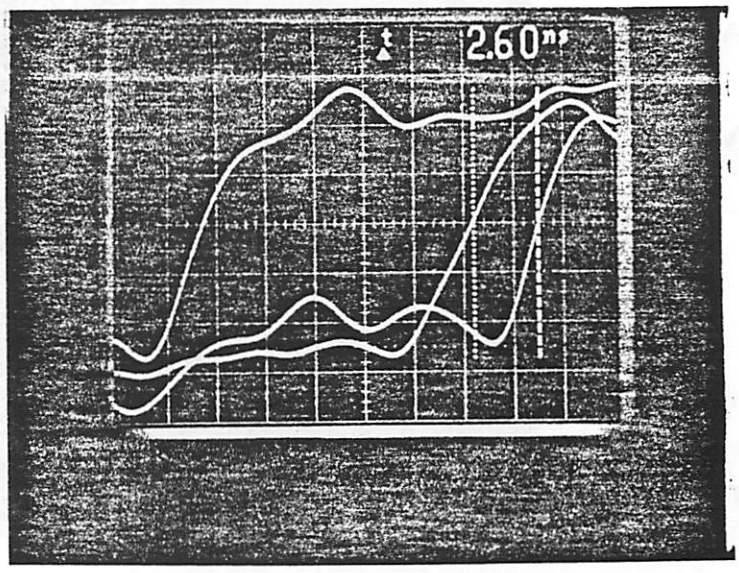


Fig. 17f: CURRENT MATCH LINE_to_VAL.CURRENT WORD LINE Delay (MATCH)

(Basic Operation)

C.ML(external pad)
TAG.ML.to.VAL.WL
val_C.WL(external pad)

(1V/Vert. Div.)
(2ns/Hor. Div.)



Path	TAG.ML.to.VAL.WL to val_C.WL
Simulated	2.25ns
Measured	2.6ns

Fig. 17g: CURRENT MATCH LINE_to_VAL.CURRENT WORD LINE Delay (MATCH)

(Detailed Operation)

EVAL.CAM.DEC

PRECH.ML

C.ML

tag_BL

(5V/Vert. Div.)
(20ns/Hor. Div.)

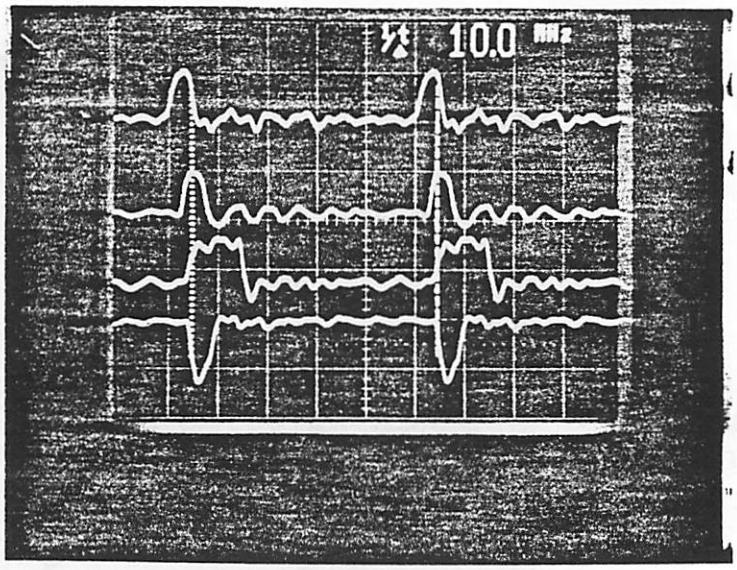


Fig. 17h: TAG BIT LINE_to_CURRENT MATCH LINE Delay (MATCH) (Basic Operation)

EVAL.CAM.DEC

PRECH.ML(external pad)

C.ML(external pad)

tag_BL(external pad)

(5V/Vert. Div.)
(20ns/Hor. Div.)

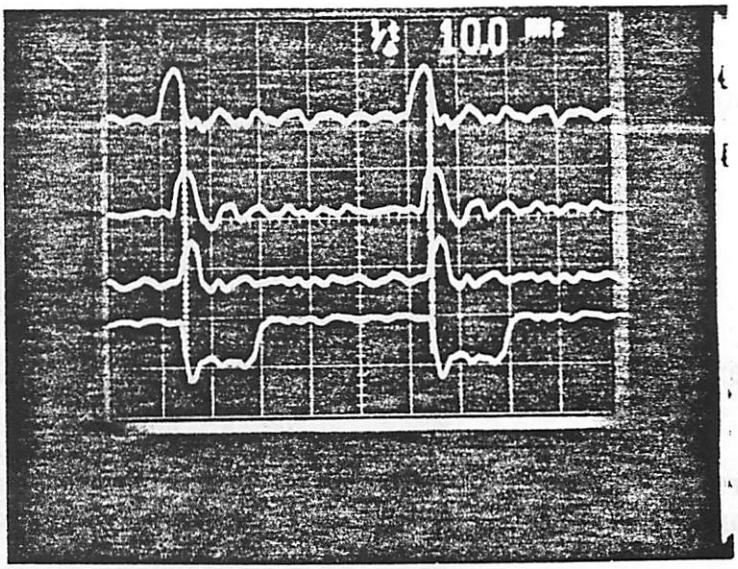


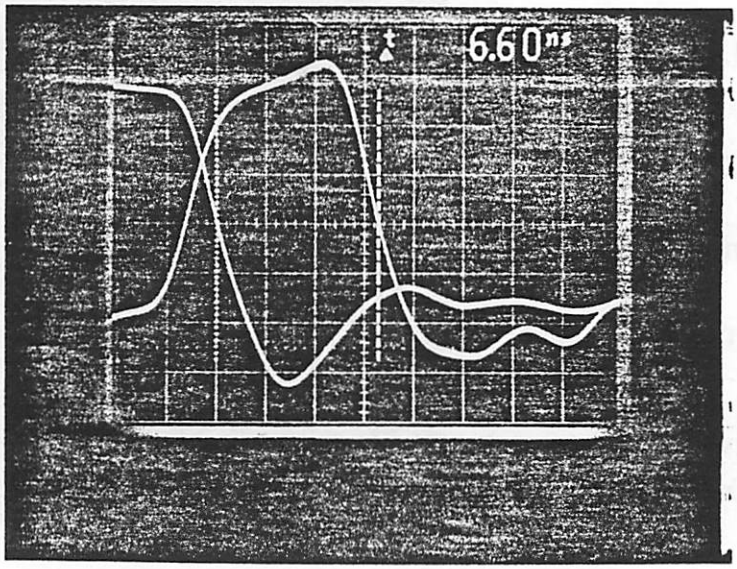
Fig. 17i: TAG BIT LINE_to_CURRENT MATCH LINE Delay (MISMATCH)

(Basic Operation)

tag_BL(external pad)

C.ML(external pad)

(1V/Vert. Div.)
(2ns/Hor. Div.)



Path	tag_BL to C.ML
Simulated	6ns
Measured	6.6ns

Fig. 17j: TAG BIT LINE_to_CURRENT MATCH LINE Delay (MISMATCH)

(Detailed Operation)

PRECH.ML(external pad)
(CAM Decoder Output)

TAG.BML.to.VAL.BWL.bar

val_B.WL(external pad)

val_C.WL

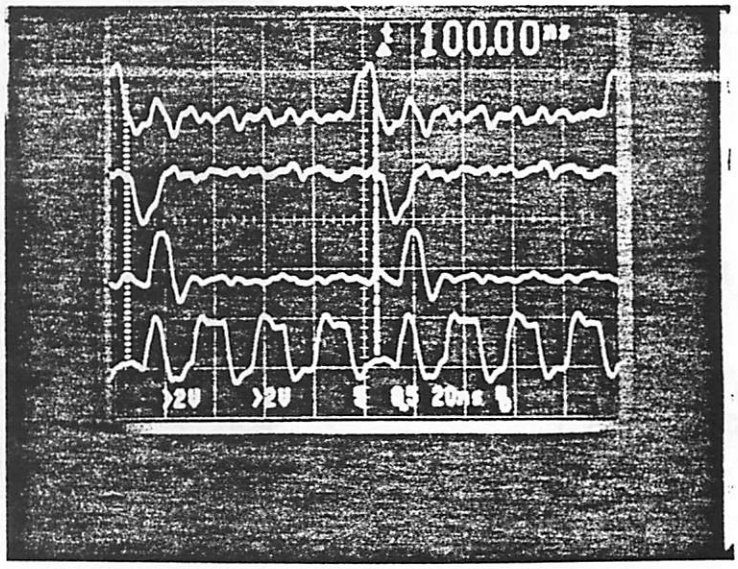


Fig. 18a: BACKUP MATCH LINE_to_BACKUP WORD LINE Delay (MATCH)

(Basic Operation)

TAG.BML.to.VAL.BWL.bar

val_B.WL(external pad)

(1V/Vert. Div.)
(2ns/Hor. Div.)

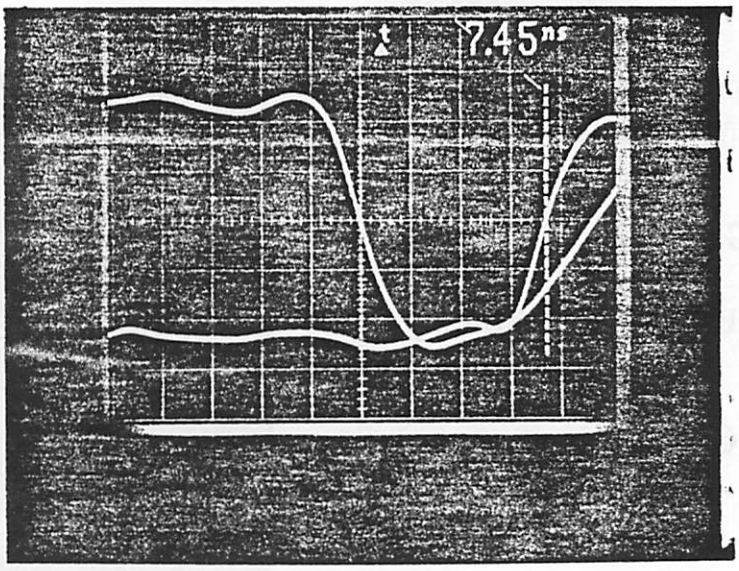


Fig. 18b: TAG.BML.to.VAL.BWL.bar_to_BACKUP WORD LINE Delay (MATCH)

(Detailed Operation)

TAG.BML.to.VAL.BWL.bar

val_B.WL(external pad)

(1V/Vert. Div.)
(2ns/Hor. Div.)

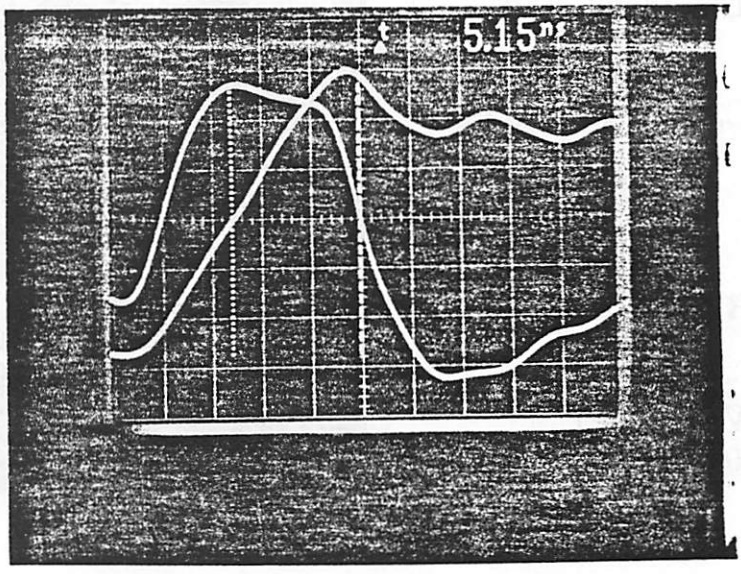


Fig. 18c: TAG.BML.to.VAL.BWL.bar_to_BACKUP WORD LINE Delay (WL DISCHARGE)

(Detailed Operation)

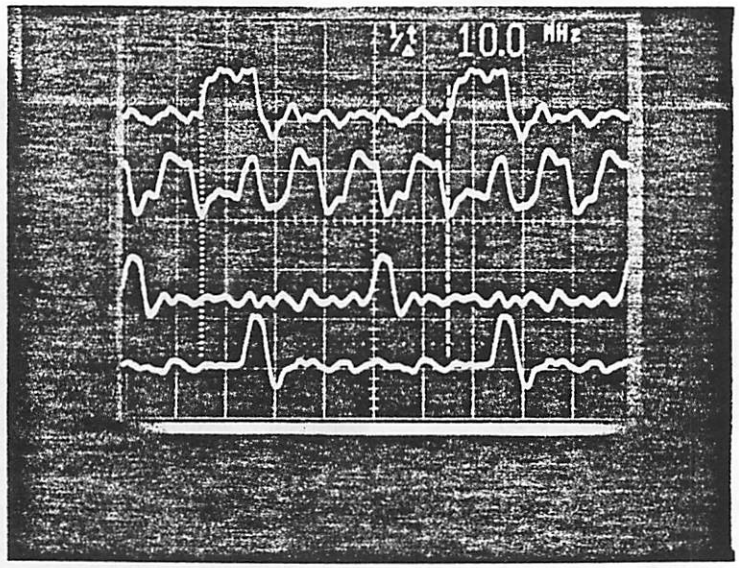
C.ML(external pad)

val_C.WL(external pad)

C2T

val_BWL(external pad)

(5V/Vert. Div.)
(20ns/Hor. Div.)



CYCLE	1st Cycle	Next Cycle
Observation	Currrent and Backup MATCH Because of SAVE	Currrent and Backup MATCH Because of SAVE

Fig. 19a: SAVE (C2T) INTERACTION WITH CAM OPNs. (SAVE EVERY CYCLE)

(Basic Operation)

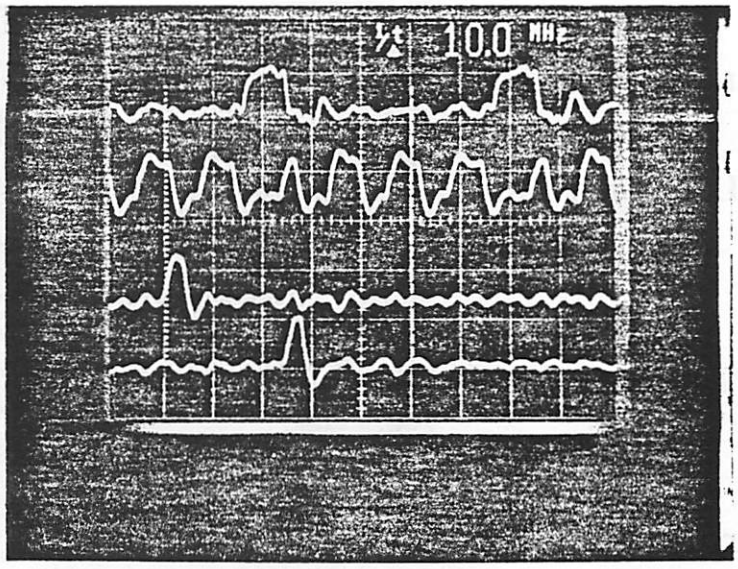
C.ML(external pad)

val_C.WL(external pad)

C2T

val_BWL(external pad)

(5V/Vert. Div.)
(20ns/Hor. Div.)



CYCLE	1st Cycle	Next Cycle
Observation	Currrent and Backup MATCH Because of SAVE	Only Currrent MATCHes Because NO SAVE

Fig. 19b:SAVE (C2T) INTERACTION WITH CAM OPNs. (SAVE EVERY OTHER CYCLE)

(Basic Operation)

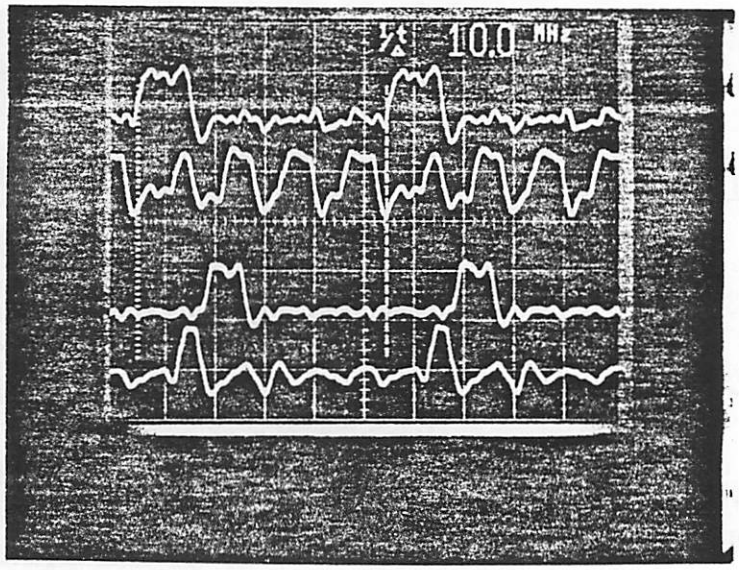
C.ML(external pad)

val_C.WL(external pad)

T2C

val_BWL(external pad)

(5V/Vert. Div.)
(20ns/Hor. Div.)



CYCLE	1st Cycle	Next Cycle
Observation	Currrent and Backup MATCH Because of REPAIR	Currrent and Backup MATCH Because of REPAIR

Fig. 20a: REPAIR (T2C) INTERACTION WITH CAM OPNs. (REPAIR EVERY CYCLE)

(Basic Operation)

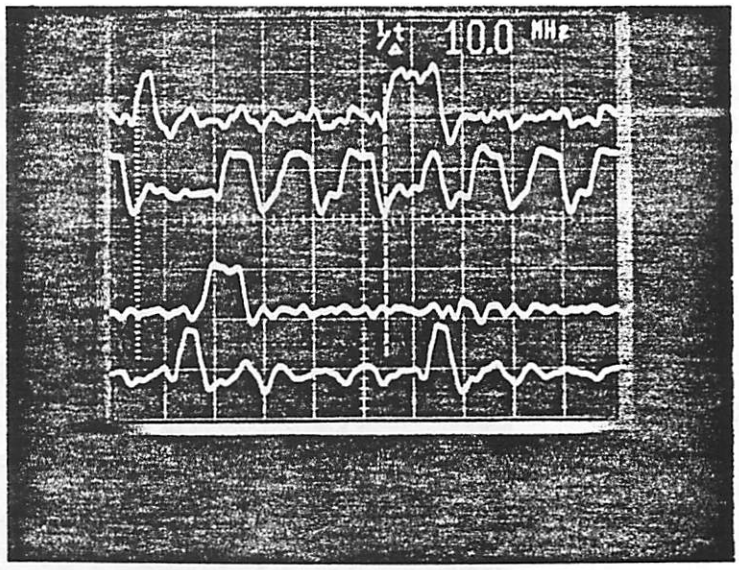
C.ML(external pad)

val_C.WL(external pad)

T2C

val_BWL(external pad)

(5V/Vert. Div.)
(20ns/Hor. Div.)



CYCLE	1st Cycle	Next Cycle
Observation	Currrent and Backup MATCH Because of REPAIR	Only Backup MATCHes Because of NO REPAIR

Fig. 20b: REPAIR (T2C) INTERACTION WITH CAM OPNs. (REPAIR EVERY OTHER CYCLE)

(Basic Operation)

CHIP GROUND
(3.5ohm Resistor
in Series with Chip)

S2C

T2C

REFRESH

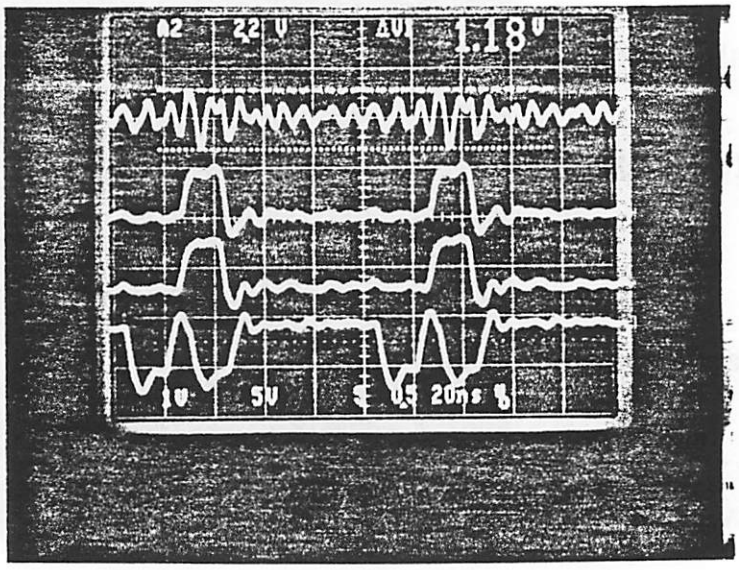
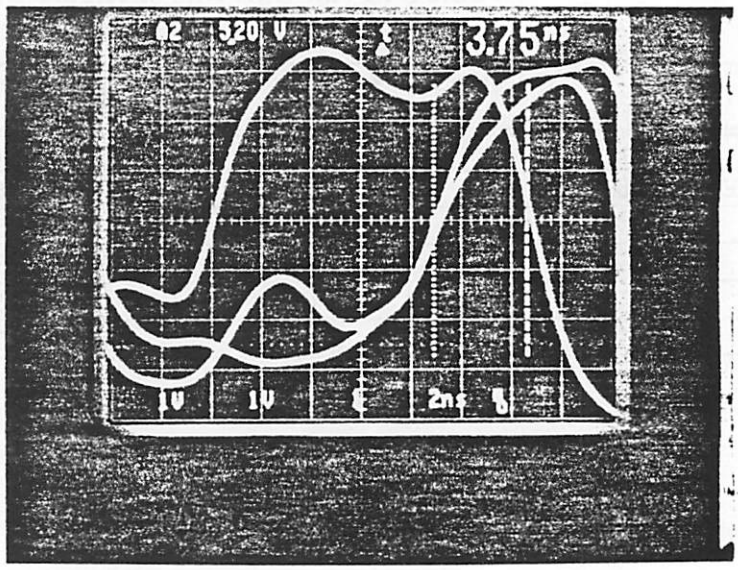


Fig. 21a: GROUND BOUNCE
(Basic Operation)

Table 3 : Peak Currents of Highly Capacitive Signals

Signal	Calculated Capacitance (pF)	Calculated Peak Current (mA)
REFRESH	112.92	245.1
REFRESH.bar	110.73	223.6
C2T	100.74	216.3
T2C	100.74	216.3
T2S	87.45	186.6
S2C	79.20	181.7

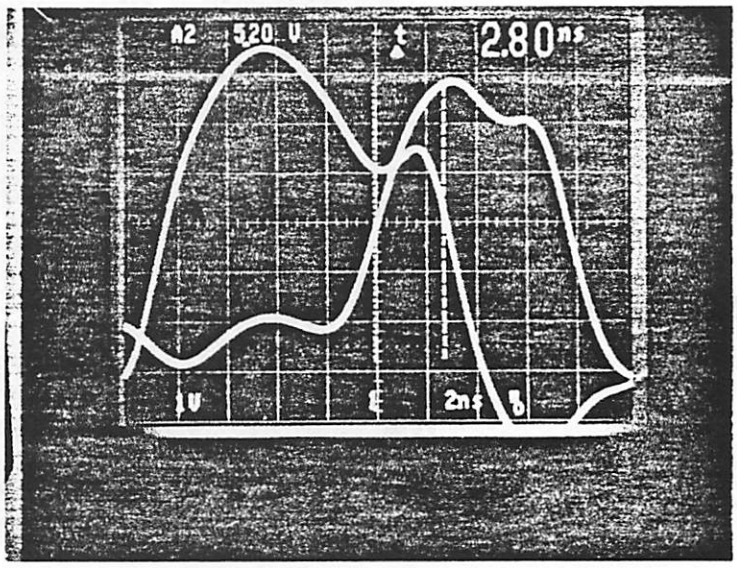
tag_C.WL(external pad)
TAG WL CLEAR (SEL.MUX)
EQ



Path	SEL.MUX to tag_C.WL
Simulated	2.0ns
Measured	3.75ns

Fig. 22a: HAZARD 1 - Early EQ Corrupts TAG Data (Detailed Operation)

val_C.WL(external pad)
VAL WL CLEAR (EQ)



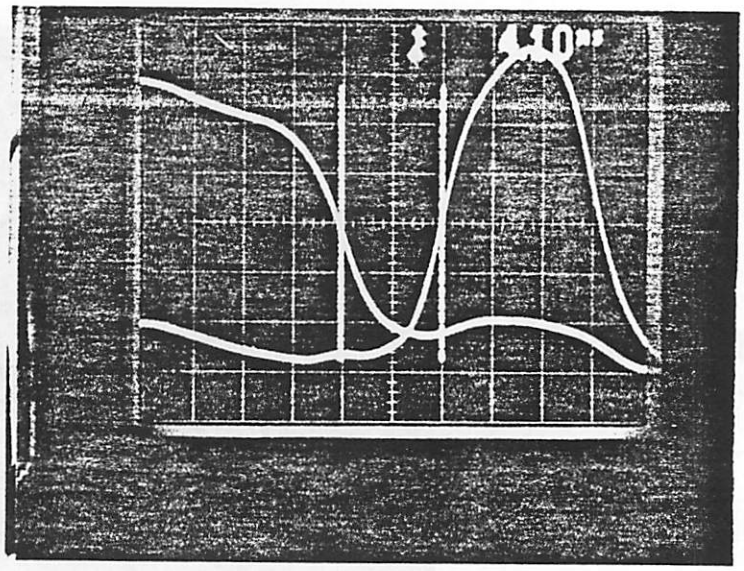
Path	EQ to val_C.WL
Simulated	1.0ns
Measured	2.8ns

Fig. 22b: HAZARD 1 - Early EQ Corrupts VAL Data (Detailed Operation)

val_C.WL(internal pad)

Qualified EQ(VAL.EQ)
(internal pad)

(1V/Vert. Div.)
(2ns/Hor. Div.)



Path	val_C.WL to VAL.EQ
Simulated	3.5ns
Measured	4.1ns

Fig. 22c: HAZARD 1 CORRECTION
(Early EQ Does Not Corrupt VAL Data)

(Detailed Operation)

CHAPTER 5

Smart Memory Design Issues

5.

5.1. Introduction

Two smart memories (one for instructions, and one for data) were described in the previous two chapters. Hopefully, their treatment has shed some light on the main characteristics of smart memories for data flow CPU's. This chapter discusses smart memories in a more abstract way than the last two chapters. First, broad issues and solutions will be dealt with one by one in the following paragraphs. Next, a cost/functionality study will be made on the RAT to show the cost of the RAT's extra functionality.

5.2. Issues and Proposed Solutions

5.2.1. Issue : High Bandwidth

In order to fully exploit the benefits of out-of-order execution, several function units that run in parallel have to be used. This implies that multiple busses and multiport memories must be used to achieve high bandwidth accesses of operands and distribution of results. Each function unit should be able to fetch 2 input operands and distribute 1 output result every cycle. This translates to $3n$ data accesses per cycle for n function units.

5.2.1.1. Sub-issue : Multiple Busses

To make the busses as short as possible to minimize the bus areas and delays, each smart memory interacts with the busses from only one side. This one-sided approach is employed in the RAT and the NT. In the case of the NT, the FU.BUS (the output of the function units) is made to pass through the NT core to accomplish this.

5.2.1.2. Sub-issue : Multi-Port Memories

Direct control of the feedback path in each data cell is the technique used to achieve single-ended access without excessive area penalty in the case where multi-port memories are needed. For instance, for the RAT, a REFRESH signal is used to control the feedback path. The RAT value cell is shown in Fig. 1. The feedback path is broken for the WRITE operation by taking the REFRESH signal low. The REFRESH signal is kept high for the READ operation to prevent a destructive READ. The chosen single-ended technique also gives a clean way for allowing communication between the current and backup sections of the cell for memories that have backup copies to support branch prediction and exception handling. By breaking the feedback path, save or repair can take place unhindered by any contention.

This approach has certain shortcomings. Firstly, it has to be controlled very carefully since it is what determines if a WRITE or a READ is taking place. Secondly, it shifts complexity to the clock generator which generates the feedback control signal (called REFRESH). REFRESH is a highly capacitive (113pF !) signal since it goes to all the cells in the memory!

5.2.2. Issue : Longer Cycle

A longer cycle is a consequence of the high bandwidth communication mentioned above. The word lines are longer and therefore more capacitive with more bit lines. When branch prediction and

exception handling are supported, the bit lines are made longer still since the bit lines have to traverse the backup copies in addition to the current ones. Again, the cycle is made longer still by the many operations that have to be performed per cycle. For example, the RAT has provision in each cycle for 4 SAVE/REPAIR, 2 READ and 2 WRITE phases.

For memories that must do SAVE/REPAIR operations, the impact on the cycle is minimized by overlapping these operations with the normal READ/WRITE operations. The use of sense amplifiers is a solution suggested for shortening the cycle for the big memories like the RAT. The single-sidedness and stringent pitch requirements imposed by the need for high bandwidth communication, automatically limits the possible number of sense amplifier techniques that can be considered. The first sense amp that was considered is the charge-sharing sense amp. This type of sense amp has been successfully used in EPROMs and PLAs. This is possible because EPROMs and PLAs only READ and never WRITE. The need to WRITE a "1" in the HPSm memories makes the use of such a sense amp inefficient.

The proposed sense amplifier is the split-bit line, half-V_{DD} sense amplifier used for the RAT. The main advantage of this sense amplifier is: it is one of the fastest way to sense a signal in a small area because it uses positive feedback. In the case of the RAT, the need to have 120 sense amplifiers for 120 ports makes this advantage very important.

5.2.3. Issue: Multiport Associative Operations

The distribution of results into the memories is done only when the interrogative tags accompanying the results match the stored tags. For memories like the RAT which have backup copies, the tag comparison must be made with the backup tags as well. The 2 choices considered are:

i) Fetch tags and compare in the periphery before distributing. This option was rejected since one extra phase is needed for fetching the tags.

ii) Use a multiport CAM cell in the tag field. This CAM cell has to contain a backup copy if the memory supports branch prediction. Hence, the worst case task is to design a multiport CAM cell that has 2 back-to-back parts which is more efficient than coupling several single CAM cells. A more compact cell is achieved by sharing bit lines and data storage nodes and using a compact tag comparator as shown in Fig. 2 for the RAT tag cell. Only the upper part of Fig. 2 is needed in the memories that don't need to support branch prediction like the NT tag cell shown in Fig. 3. Because the number of comparators is proportional to the number of ports (2 are needed per port in the dual-cell case), most of the efficiency of the cell depends on the efficiency of the comparator used. A modified Kadota [1] tag comparator is the proposed comparator. It neither has the capacitive-loading and area problems of the Mundy tag comparator [2] nor the charge-sharing problem of the Kadota tag comparator.

5.2.4. Issue : Dynamic Power Consumption and High Peak Currents.

The multiport nature of the memories implies a multiplicity of bit lines, match lines, word lines and busses. To prevent excessive dynamic power dissipation and high peak currents (with associated inductance effects), precharging the many match lines, bit lines and busses is avoided. In the case where match lines have to be precharged, a hashing scheme is employed by using some information about the memory (e.g. the REGISTER # or the READY BIT) to select just a few match lines for precharging. In the case of the RAT, the REGISTER # is used to select only 6 out of a total of 186 match lines for precharging. In the case of the NT, the READY BIT is used to selectively precharge the match lines. By avoiding precharging, the electromigration problem, IR drop, voltage bouncing noise due to wiring inductance and ac power for charging and discharging are considerably reduced. The chip reliability is also increased.

The half-V_{DD} sensing scheme mentioned above has the following advantages as regards the present issue:

i) Due to the half-V_{DD} swing, it cuts down by almost a factor of 2 the peak currents both during sensing and bit line precharge.

ii) It reduces the dI/dt by a factor of 2 during bit line precharge and discharge if the time is fixed, which decreases the inductance-induced voltage noise. If the voltage bumping noise is not the limit, the precharge and discharge times can be reduced by about a factor of 2.

5.2.5. Issue: Low Bit Density

The smart memories suffer from low bit density because of the extra functionality. For instance, to support exception handling and branch prediction, 2 backup copies are required for every programmer-visible cell. To minimize the area cost, single bit lines rather than bit line pairs are used as described above. The CAM field that needs complimentary bit lines to achieve complete logic comparison, is the only field that uses bit line pairs. But since the CAM field columns are few, the area penalty of using bit line pairs for the CAM field is minimal.

Dynamic and static latches are judiciously combined to give compact but robust data cells. Dynamic latches rather than static latches are used when compact cells are needed and the *REFRESH* problem can be adequately handled. These cells are used in the backup sections of the RAT as shown in Fig. 1. Static latches are used in the NT since only single-port access is needed in the NT. A dynamic-static "hybrid" latch (pseudo-static) is used to exploit the advantages of both dynamic and static cells in one cell. Pseudo-static latches are used to avoid providing *REFRESH*. They are also used to allow contention-free communication between backup and current cells like in the case of the RAT. To represent 3 states in the HPSm RAT, a pseudo-static cell is used as the current (programmer-visible) cell while dynamic latches are used in the transit and settled cells as shown in Fig. 1.

Because the density is inversely proportional to the "smartness" of the cell, the issue of low bit density is related to the issue of how much "smartness" or logic to embed in the data cells. Pushing the logic to the periphery is the alternative to embedding the logic in the cell. The decision of where to put the logic (the periphery or the memory core) was made depending on the cost of the implementation in terms of the area, speed, power and noise. In the case of the RAT, the decision was made to implement the associative operation by using "smart" CAM cells. The alternative of implementing the same operation by fetching and comparing the tags in the periphery was rejected because of the impact on the cycle time as mentioned above. At the same time, the cost of using the CAM was bearable since the tag field is only 5 bits wide. On the other hand, in the case of the NT, the SHIFT-REGISTER priority encoding scheme that embeds the scheduling logic in the data cells was rejected in favor of the POINTER-MECHANISM because of the increased memory core area and the higher dynamic power and peak currents.

5.2.6. Issue: Branch Prediction and Exception Handling Support

5.2.6.1. Sub-issue: Branch Prediction Support

To achieve much higher performance, an out-of-order execution machine may take the out-of-order execution model one step further by performing branch predictions. That is, instead of just executing out-of-order the instructions within a block, two blocks can be executed out-of-order. HPSm performs branch predictions on the instruction stream. Therefore the capability of recovering cleanly from an incorrect prediction is very essential.

To support branch prediction, a subset of the machine state has to be saved when a conditional branch instruction is encountered in the instruction stream. Only a subset of the entire machine state is saved since it will be too costly to save the entire state. Also, it will be too costly to support the branch

prediction in a *context switch* manner of writing and reading all the smart memory contents to main memory. Therefore, some of the on-chip smart memories must have the capability of storing the subset of the machine state that must be saved.

The smart memories that must store the essential subset must contain at least 2 back-to-back parts : current entry and branch prediction backup entry. The contents in the backup must be restored back to the current entry when it is later on discovered that the branch prediction was incorrect. The backup entry must be accessible by the distribution bus since operations are executed out-of-order and therefore one cannot be sure that all instructions before the branch have been executed. It is therefore necessary to be able to write both into the current and backup entries. In addition, the save into and repair from the backup cell must be efficient and doable in 1 cycle since branch instructions occur quite frequently (1 branch instruction per 5 instructions is typical). The memories that are not in the essential subset that save contents, must have a means to invalidate the necessary entries. Finally, to avoid undue performance degradation, the save/repair and invalidation operations must be overlapped with the normal operations. In the case of HPSm, the RAT is in the essential subset while the three NT's are not. Both kinds of memories have the desirable features, listed above, for supporting branch prediction.

5.2.6.2. Sub-issue: Exception Handling Support

The out-of-order nature of execution makes exception handling for out-of-order execution machines very difficult. Some form of hardware support for exception handling is needed. An exception repair (E-repair) involves cleanly suspending the process to a point preceding the violating instruction, switching context, handling the exception, and resuming execution from that point. HPSm uses the concept of checkpoints [3] to handle exceptions. At every checkpoint, the current contents are saved into "transit" cells so that repair can be made using these saved contents. After a given number of instructions, the "transit" information is saved into "settled" cells. Three cells are used for correctness reasons (see Chapter 4 of Reference [3].

5.2.6.3. Sub-issue: Proposed Solution

The triple-cell shown in Fig. 1. is used to efficiently support branch prediction and exception handling. It uses 1 pseudo-static latch for the "current" cell, and 2 dynamic latches for the "transit" and "settled" cells. Using dynamic latches in the backup section gives a compact cell and makes it possible to do save and repair without any contention. (The contention reflects a disturbing feature of smart memories - the cells not only behave like logic elements but sometimes have to behave like peripheral circuits. The REPAIR operation is essentially a WRITE operation. But a WRITE operation is easier to do with a large peripheral WRITE buffer that is shared by the cells in each column rather than with cell circuitry if the cell area must not be excessively large.)

Four SAVE/REPAIR signals are used for communication between the states. The timing of the SAVE/REPAIR signals ensures that the save/repair operations are overlapped with the normal operations.

5.2.7. Issue: REFRESH

If dynamic cells have to be used to minimize the area, the issue of REFRESH has to be addressed. The information stored on the dynamic nodes must have leaked away after about 2ms. Hence, REFRESH is necessary to ensure the integrity of the stored information. REFRESH cannot be provided in a CPU the way it is done for conventional DRAMs : the constraints are much more stringent in the former case. Since, REFRESH is needed only once every 2ms, speed will be degraded if a phase is allocated in every cycle just for REFRESH. On the other hand, a more complex control will be needed to do REFRESH if REFRESH is done in time slots allocated 2ms apart. The challenge, therefore, is to find a way to overlap the REFRESH operation with the normal CPU operations with minimum control complexity.

The proposed solutions are:

i) Ignore REFRESH if the cell does not need to retain the information for up to 2ms.

ii) If (i) cannot be done, avoid REFRESH by using static or pseudo-static cells if possible. This approach is used for the current cell of the RAT triple-cell in Fig. 1 and for the NT cells.

iii) If (i) and (ii) cannot be done, push the complexity to the software level by having software-generated REFRESH. This approach is used for the backup cells in Fig. 1. The REFRESH is initiated by software with no hardware cost. Normally, after an average of every 5 instructions, a branch prediction is made and the transit cell is written into from the current cell. If a branch prediction has not taken place in a reasonable time (2ms, say), the compiler inserts an artificial branch in the code to force a current-to-transit save. Because, as was discussed in a section above, the save/repair operations are overlapped with the normal operations, the software-generated REFRESH does not need a special timing provision.

5.3. Cost/Functionality Study

In this section, a comparison is made between the register files of a conventional von Neumann CPU and that of an out-of-order execution CPU that have been implemented in silicon. The HPSm and SPUR (UC Berkeley's Symbolic Processing Using RISCs) [4] register files are compared. Since both the HPSm and the SPUR register files were implemented with the same technology, this comparison factors out the impact of technology. Strictly speaking, comparing with the SPUR register file (SPUR-RF) is not ideal since the latter is not a conventional register file. The SPUR register file also has some additional functionality like handling register windows, and aiding the pipeline implementation. Nevertheless, it is safe to say that the SPUR register file belongs to an earlier generation of CPU register

files for the following reason : it is meant for a sequential von Neumann machine. The most important consideration is that a comparison can be made to highlight the impact of the "smartness" of the HPSm RAT on its area and complexity.

Fig. 4a and Fig. 4b show a comparison of the HPSm RAT and SPUR-RF functions while Fig. 5a and Fig. 5b show a comparison of their costs. While their speeds are comparable, the RAT performs terribly as regards area and power. The low bit density of the RAT can clearly be seen by looking at the following ratios : number of transistors per $micron^2$ and number of stored bits per $micron^2$. The power penalty would have been worse if the half-V_{DD} sense amp discussed above was not employed. The speed performance of the RAT is due mainly to the shorter word line the decoder has to drive (just the tag word line), and the sense amplifiers. The SPUR-RF will certainly be faster than the RAT if it is designed for 31 registers and with sense amps. The first shocking revelation is that the RAT (which has 31 programmer-visible registers) is bigger than the SPUR register file (which has 138 programmer-visible registers)! Now, if one argues that the RAT's effective number of registers is really 93 and not 31 (due to the 3 states represented in the RAT), one observes that with 33% less registers, the RAT is 76% bigger! This imbalance can only be explained by the extra functionality the RAT has over the SPUR-RF as revealed in Figs. 4.

5.4. Conclusions

In this chapter, the smart memories were discussed from the viewpoint of the pertinent design issues. Hopefully, looking at the smart memories from the concrete standpoint of individual designs (in Chapters 3 &4) and from the abstract perspective of the issues have highlighted the circuit challenges one must face in designing smart memories.

References

[1] H. Kadota et. al., "An 8-kbit Content-Addressable and Reentrant Memory", *IEEE J. Solid State Circuits*, vol. SC-20, no.5, pp. 951-956, October 1985.

[2] J.L Mundy et. al., "Low-Cost Associative Memory", *IEEE J. Solid State Circuits*, vol. SC-7, no.5, pp. 364-369, October 1972.

[3] Wen-mei Hwu, "HPSm: Exploiting Concurrency to Achieve High Performance in a Single-chip Microarchitecture *Ph.D. Dissertation, Computer Science Division, EECS Dept.*, University of California, Berkeley, CA. 94720, 1987.

[4] M.D. Hill et al., "Design Decisions in SPUR", *IEEE Computer, vol. 19, no. 10, pp. 8-24, Nov. 1986.*

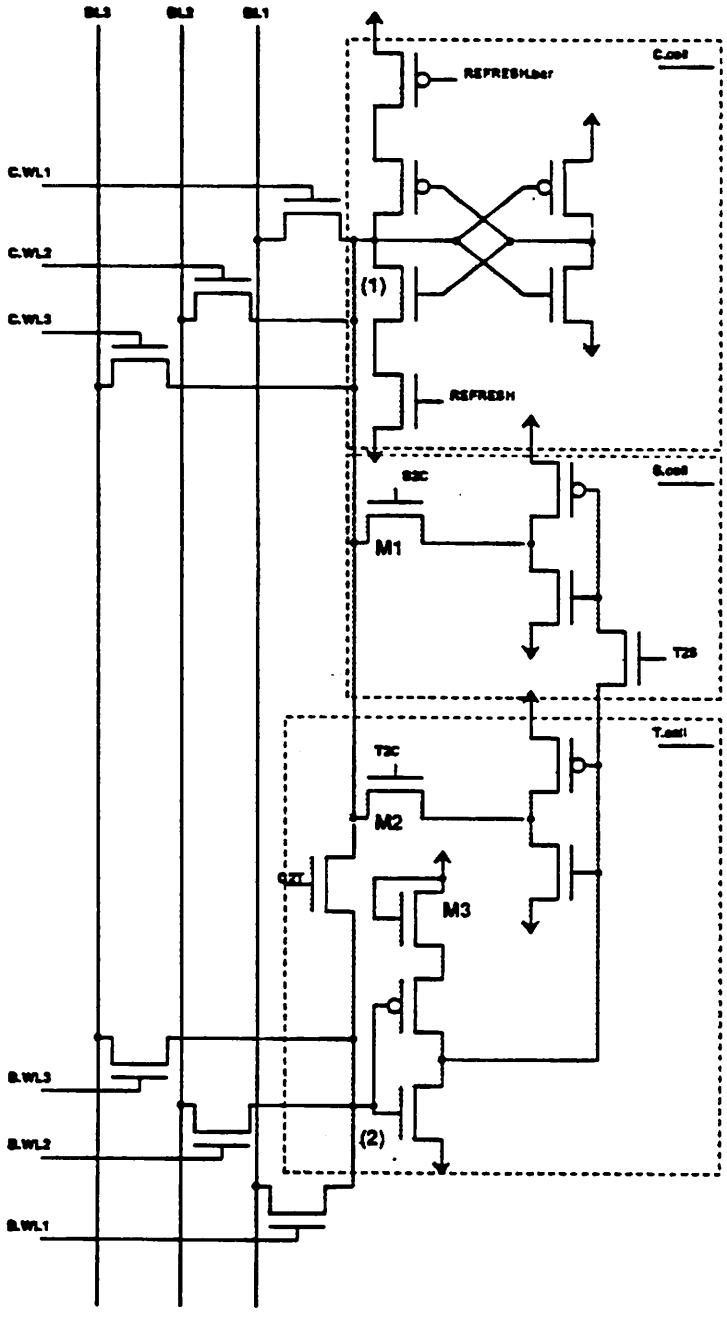


Fig. 1: RAT Value Cell

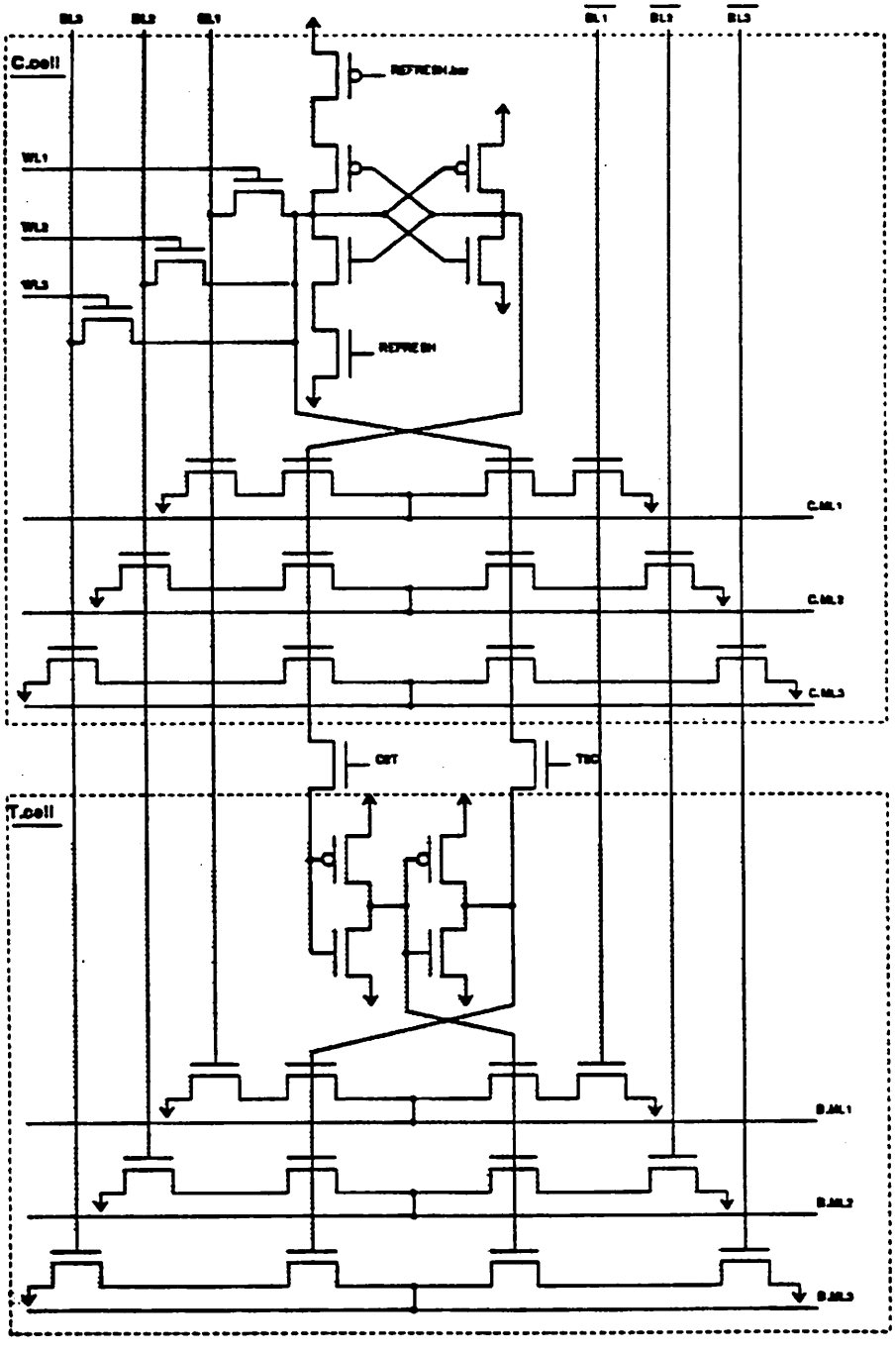


Fig. 2: RAT Tag "Double-CAM" Cell

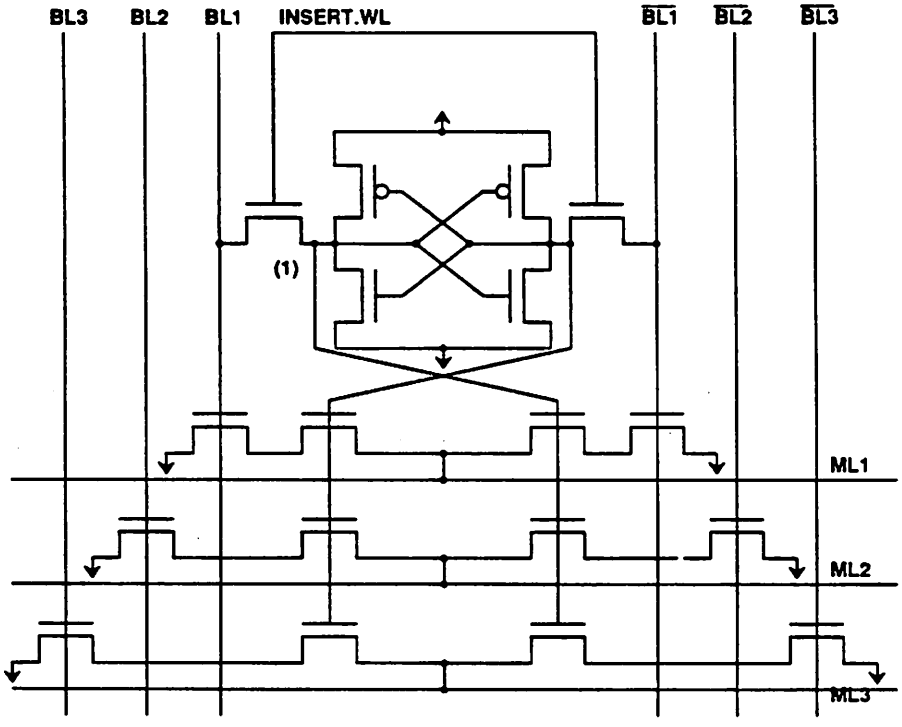
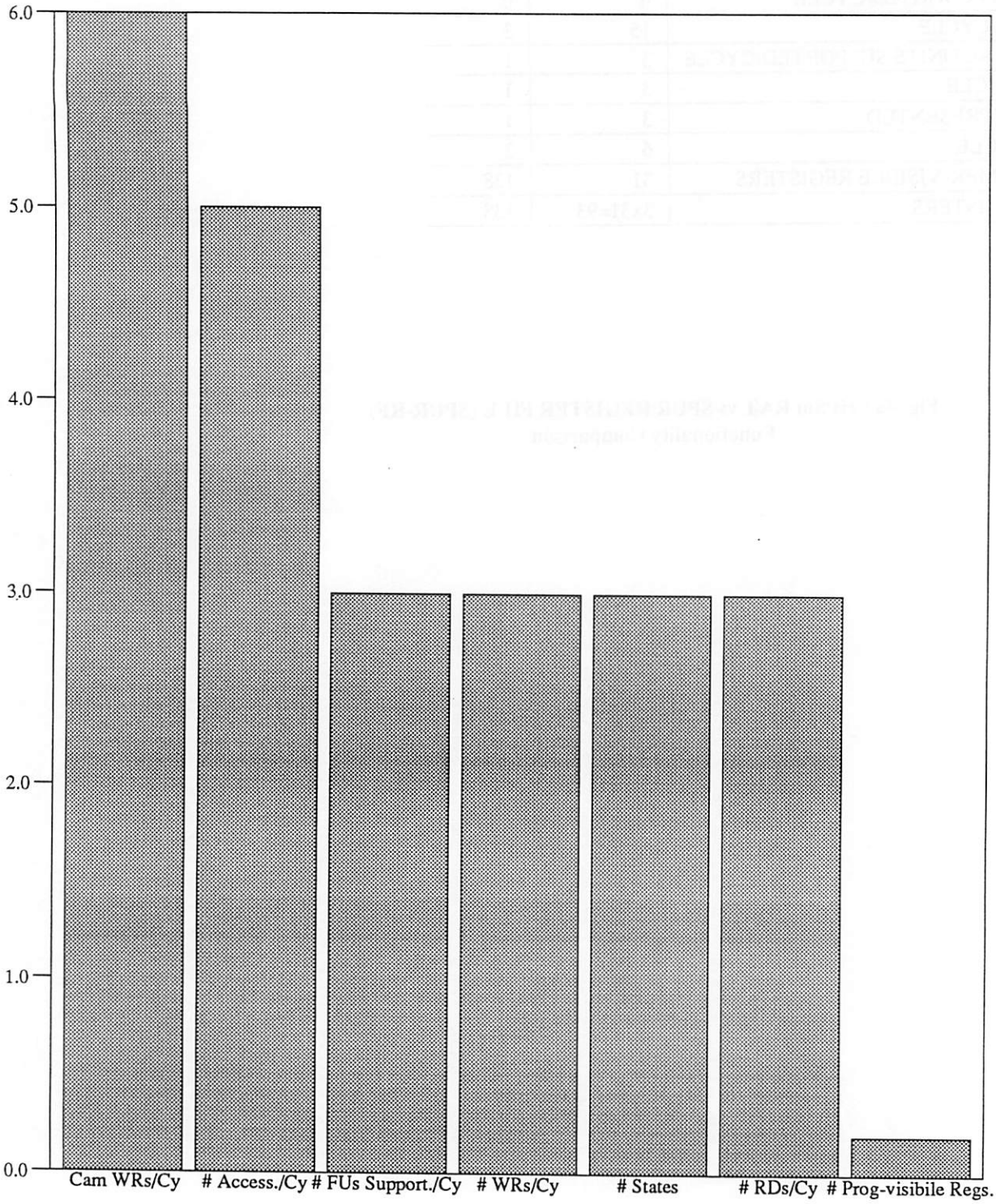


Fig. 3: NT Tag "Single-CAM" Cell

FUNCTIONALITY MEASURE	HPSm RAT	SPUR-RF
# ASSOCIATIVE WRITES/CYCLE	6	0
# ACCESSES/CYCLE	15	3
# FUNCTIONAL UNITS SUPPORTED/CYCLE	3	1
# WRITES/CYCLE	3	1
# STATES REPRESENTED	3	1
# READS/CYCLE	6	2
# PROGRAMMER-VISIBLE REGISTERS	31	138
# TOTAL REGISTERS	3x31=93	138

Fig. 4a : HPSm RAT vs SPUR REGISTER FILE (SPUR-RF) Functionality Comparison

Fig. 4b: HPSm RAT vs SPUR-RF Functionality Comparison

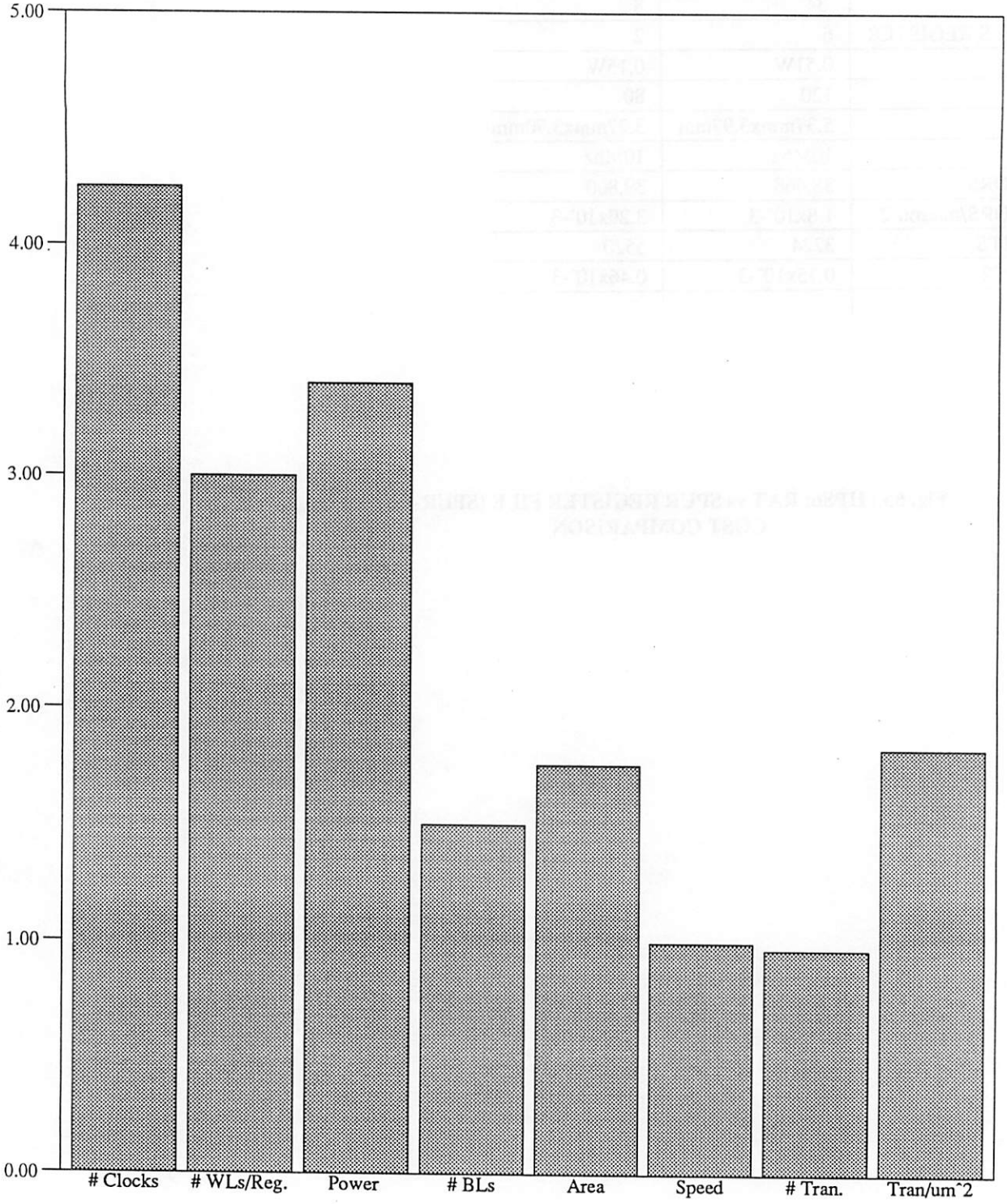


VERTICAL AXIS SHOWS THE RATIO OF
THE HPSm RAT FUNCTIONALITY MEASURE TO
THE SPUR-RF FUNCTIONALITY MEASURE.

COST MEASURE	HPSm RAT	SPUR-RF
# CLOCKS	34	8
# WORD LINES/REGISTER	6	2
POWER	0.51W	0.15W
# BIT LINES	120	80
AREA	5.37mmx3.97mm	3.27mmx3.70mm
SPEED	10Mhz	10Mhz
# TRANSISTORS	38,468	39,800
# TRANSISTORS/micron^2	1.8x10^-3	3.29x10^-3
# STORED BITS	3224	5520
# BITS/micron^2	0.15x10^-3	0.46x10^-3

Fig. 5a : HPSm RAT vs SPUR REGISTER FILE (SPUR-RF) COST COMPARISON

Fig. 5b: HPSm RAT vs SPUR-RF Cost Comparison



VERTICAL AXIS SHOWS THE RATIO OF
THE HPSm RAT COST MEASURE TO
THE SPUR-RF COST MEASURE.

CHAPTER 6

Conclusions

6.

6.1. Future Work

Reliability is an unresolved research issue. The use of dynamic latches was proposed to minimize the real estate requirement. But, this automatically raises efficiency and reliability questions. Having V_{DD} - V_T at the gate of any dynamic latch may lead to static power dissipation due to the uncorrelation of the V_T's of the PMOS and NMOS devices. Soft errors due to alpha particle events are a reliability issue that plagues dynamic memories. The need for backup word lines prevents one from using a folded bit line sense amplifier scheme. The open bit line scheme that is employed does not enjoy the noise immunity that folded bit line schemes have due to local cancellation. Future research has to grapple with these issues.

The impact of technological trends on the design implications of smart memories is another research issue. In particular, the effect of employing BICMOS technology and scaling the device sizes and the power supply needs to be studied.

6.2. Final Conclusions

This dissertation has discussed the circuit challenges one must address in designing **Smart Memories for Out-of-Order Execution VLSI Architectures.** Certain alternatives were compared for each design issue. The proposed solutions were used to implement a design of a smart instruction memory (the NT) on paper and a design of a smart data memory on silicon (the RAT). The NT and the

HPSm CPU design could not be advanced to the silicon stage because of lack of time. The proposed solutions for the RAT were demonstrated to be adequate by the successful testing of the RAT test chip.

However, it must be pointed out that the test chip is an artificial environment since the RAT was not meant to be a stand-alone smart memory. Therefore, it will never be known if the RAT satisfies the requirements for a smart data memory for an out-of-order execution machine until it is fabricated along with the other smart memories (the NT's) in a single-chip CPU. It is also interesting to know how the ideas presented in this thesis can be extended to other smart memories especially smart memories for other data flow CPUs, HPS Prolog CPU, and HPSm's FPU (Floating Point Unit), MMU (Memory Management Unit) and CC (Cache Controller).

It is clear from the discussion in this dissertation that the smart memory circuits designer needs a solid background in logic design since the memory-logic boundary is blurred in smart memories as discussed in Chapter 1. The discussion in Chapter 3 bears this out where the recognition that the priority encoding scheme can be implemented as a finite state machine outside the NT core trivialized the garbage collection function and allowed the use of simpler data cells. It is also clear that, for him to design efficient memories, the designer needs a fair understanding of the system into which his memories will fit since the designs are application-specific. Above all, since these memories are costly, a lot of interaction between the circuit design and computer architecture research efforts is imperative to reduce the costs. In the case of the HPSm microarchitecture design, the feedback to the architecture research from the circuit design research led to a reduction of the types of needed smart memories from 5 to 2 and the number from 7 to 4.

It is obvious from the comparison of the HPSm and SPUR register files in Chapter 5, that the increased functionality of smart memories carries a heavy price tag. It is also obvious that by expending more design effort, employing the appropriate circuit techniques and by shifting some complexity to the software level (e.g. software-generated REFRESH), this price tag can be made less threatening.

Nevertheless, the main feedback to the computer architects is that these memories are very costly. Therefore, they must be employed with few entries and only if the advantages outweigh the disadvantages. It does not make more sense, for instance, to have an 8-entry NT instead of a 4-entry NT since the former reflects a machine where there are a lot of data dependencies and one may be better off implementing the architecture as a sequential execution machine at a lower cost.