NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Image Processing by Parallel Processors

by

François A. Gougeon

A Thesis
presented to the University of Ottawa
in partial fulfillment of the
requirements for the degree of
Master of Applied Science
in
Department of Electrical Engineering

Ottawa, Ontario, 1979

The University of Ottawa requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

ABSTRACT

In this thesis, the historical and the technological reasons for using parallel processor systems are outlined. The parallel processor systems are classified into four categories and each category is briefly analysed. Some existing systems are studied: the ILLIAC, the STARAN, the PEPE and the PMPP. The Multimicroprocessor Array Computing Structure (MACS) and a corresponding MACS simulator are described. Finaly, some image processing tasks are implemented on MACS via its simulator.

RESUME

Cette thèse comprend une revue des raisons historiques et technologiques de l'utilisation des systèmes à base de processeurs parallèles. Ces systèmes sont classifiés selon quatre catégories et chaque catégorie est brièvement analysée. Quatre systèmes contemporains sont aussi étudiés: l'ILLIAC, le STARAN, le PEPE et le PMPP. Un système de multiprocesseur (MACS) et son simulateur sont décrits. Finalement, quelques tâches relatives au traitement d'images sont réalisées grâce au simulateur.

ACKNOWLEDGMENT

The author whishes to express his gratitude to his supervisor, professor Morris Goldberg, for his encouragement and guidance throughout the course of this research.

Thanks are also due to professor Cedrick V.W. Armstrong for his efforts in coordinating the MACS project at University of Ottawa.

The author also likes to express his apreciation to Jean-François Meunier for his in depth revision of the text and his useful comments.

Thanks are due to all the members of the Department of Electrical Engineering, specially Mohammed Master, and to all the graduate students for providing a suitable atmosphere for research.

The financial support for this research was provided by NRC Grant no. A4423.

PREFACE

"Two heads are better than one."

This old adage may seem like a safe philosophical argument to uphold in any casual conversation on parallel processors or multiprocessor systems. For that matter, it could have been used in a conversation relative to the benefits of teamwork as compared to individual effort. However, in this latter case one could argue that this statement does not take into account the phychological factors involved in working with somebody else. If, for example, two cooperating engineers spend more time arguing with each other than collaborating, the net result might not be an improvement at all.

Is this philosophy applicable to computers? Probably... Two processors, or a hundred processors, should be better than a single one, provided that they are part of a well architectured system where conflicting situations are kept to a minimun. This is the hypothesis adopted as a guideline throughout this research effort.

TABLE OF CONTENTS

Chapter I

INTRODUCTION

The level of development of a society is usually judged by the amount of information and knowledge it generates. So, as society evolves, man is spending more time handling information and less time handling materials. This partially explains why the world, due to the fast development period we are going through, is now facing an information explosion. Because of this, man is stressed to handle an increasing amount of information [1].

Fortunatly, for the past thirty years, progress in the computer field has been fast encugh to relieve humans from part of this huge information processing task. Originally, computers were considered only as complex calculating machines, thus, as tools in the scientific evolution of man. However, the general computers are now more and more used as data base machines, gathering, storing, processing, and retrieving information.

In the past centuries, permanent information has been stored in the form of books. Even now, information is

mainly stored in the form of alphanumerical characters. However, since the advent of photography, cinematography and television, huge amounts of pictorial information are now stored in the form of images. When images need only to be stored and retrieved, the process is easily performed by conventional methods: photograph, film and videotape libraries. Increasingly, when images need to be processed, the computer is finding extensive use.

Since one of man's oldest occupation has always been war, it is not suprising that the incentive for picture processing arose mainly because of military applications. Surveillance satellites, orginally for military purposes only, are now playing an extensive role in civilian use. Applications using satellite pictures are seen in an increasing number of fields: geography, forestry, meteorology, agriculture, etc... The National Aeronautics and Space Administration in the United States, and the Canada Centre for Remote Sensing in Canada, have been at the forefront of extending the technology of image processing [2].

Later, other domains contributed to the advancement of image processing. Character recognition was developed to automatize the processing of mail [3]. Image compression and restoration techniques evolved out of a need to aleviate

problems encountered in communication of pictorial information [4]. Pattern recognition was stimulated by the need for providing automated machinery with some visual processing capabilities [5].

An image represents a considerable amount of data, usually more than the thousand words expressed in the famous Chinese proverb. The wider use of pictures and the demand for faster image processing, so that real time applications become feasible, already exceeds the capacity of conventional computers. Fortunatly, while image processing was evolving, ccmputer architecture was improving to handle problems such as the ones generated in aerodynamic, in radar processing, in data base systems, etc...

In this thesis, two wide fields are related via few specific cases: the processor architecture field and the image processing field. The purpose of this research is not to cover these two domains extensivly, but to demonstrate how improvements in computer architecture are beneficial to image processing. Image processing activities are by their nature very time consuming (human and computer time). Any improvement in image processing machines should bring us closer to real time pattern recognition, an important aim in cybernation (1).

---

(1) cybernation: cybernetics and automation

In general, image processing facilities usually contain some of the following:

-a general purpose computer (with memory)

-disk(s) for the operating system

-tape drive(s) for data input and backup

-big disk(s) for image data

-a display unit (color TV+refresh memory+circuitry)

-a user control unit (terminal, trackball,etc...)

-a fast slave processor for arithmetic calculations

-an image processor in-line with the display unit

-a printer-plotter

-an image recorder

-an image digitizer

The purpose of this thesis is to investigate the role of multiprocessor systems as fast slave processors or as in-line image processors. The software associated with these systems will only be examined as far as needed to implement basic operations with the hardware. The subject of general software for multiprocessor systems is a field of extensive implications, beyond the scope of this thesis.

This thesis contains six main chapters. Chapters two, three and four serve as introduction to the parallel processor field. Chapters five, six and seven contain the major contribution of this thesis, where the

Multimicroprocessor Array Computing Structure is described and simulated for image processing.

Chapter two explains the evolution of the computer field, from the sequential computers to the parallel processor systems. The historical and technological reasons for this evolution are examined in order to understand the current trend towards multiprocessor architectures.

Chapter three classifies parallel processor systems into four main classes. The advantages and disadvantages of each class are summarized. Subdivisions within these four categories clarify the matter even further by associating common generic names and their typical architectures. Also, some common characteristics of all processing systems are examined.

Chapter four investigates four different existing systems: the ILLIAC, the STARAN, the PEPE and the FMPP. It illustrates some of their similarities and some of their differences. It shows that systems belonging to the same category can be fairly different hardware-wise, and that similar problems are sometimes solved differently.

Chapter five examines a recent architecture based on bit-sliced microprocessors: the Multimicroprocessor Array

Computing Structure (MACS). The MACS is a system originally conceived for radar processing. Its architecture is described in detail, including the microinstruction fields controlling its processing elements. Part of the chapter is an introduction to bit-sliced microprocessors.

Chapter six summarizes briefly a minicomputer simulation of the MACS system and points-out its usefulness. An example demonstrates its capabilities by making use of all the monitor commands available with the simulation package. The programs forming this package are briefly outlined and their interaction with each other explainned.

Chapter seven studies MACS potential for use with image processing tasks. Four typical image processing algorithms are tentativly implemented using the MACS simulator. And finally, conclusions are drawn about the potential of MACS as part of an image processing system.

The major contribution of this thesis is found in chapters five, six and seven, where the Multimicroprocessor Array Computing Structure is described, simulated and tried for image processing applications. The MACS description comes mainly from the author first hand knowledge of the system rather than from the few publications on the subject. Such knowledge of the system was required for the

implementation of the MACS simulator. However, Macs trial as an image processor is the major assest of this thesis.

References

[1] Toffler A., "Le choc du futur", Gonthier, Paris, 1973, 637 pages

[2] Frosch R. A., "NASA Takes Stock", IEEE Spectrum, Vol. 16, June 1979, p. 44

[3] Tou J.T. and Gonzalez R.C., "Recognition of Handwritten Characters by Topological Feature Extraction and Multilevel Categorization", IEEE Trans. on Computers, Vol. C-21, No. 7, p. 776

[4] Dion J.P., "Simulation of a Digital Image Processing System", M.A.Sc. Thesis, University of Ottawa, 1974

[5] Nevatia R. and Binford T.O., "Description and Recognition of Curved Objects", Tutorial and Selected Papers in Digital Image Processing, IEEE Press, New Jersey, 1978, p597

Chapter II

REASONS FOR USING PARALLEL PROCESSORS

2.1 HISTORICAL REASONS

The demand for higher performance of processing system has always been with us, and parallel processing is not a new solution to the problem. In some of the earliest machines, predecessors of the computers as we now know them, parallelism was more a question of practicality than anything else. An example that comes to mind, is the first automatic weaving loom created by a French engineer in 1728, which was controlled by a sort of punched card. The machine was thus working completly unassisted, needing only a new set of cards to change to a different weaving pattern. Each hole of these cards controlled a specific part of the machine, allowing more than one operation at the same time[1]. Note that even in this age of CRT terminals, punched cards are still used, but the transfer of information is done serially.

Some authors [2] trace back the history of computing machines as far as 1642 when Pascal invented one of the first calculating machine. However, most authors go back a 130 years ago or so, to Babbage [3], who is considered to be one of the fathers of computer science. His differential engine, and later his analytical engine(that never saw birth) were made out of similar parallel systems of gears. They were completly parallel, the central processing unit (if we allow this metaphor), as well as the input/output(I/O).

A more vivid example of parallel processing is the analog computer. One of the main advantage of this computer is that operations are done in parallel, allowing real time feedback and contol. Two or three circuits simulating different phenomena can be working simultaneously, with or without interaction between each other, with or without feedback, and produce a result directly related to a real time process.

With the advent of faster and faster electronics, computer designers were lead away from parallelism. Designing sequential systems was simpler and sufficiently fast for their needs. However, as computer systems evolved into multi-user systems, some concurrency between I/O and processor operations was introduced. And now, concurrency or parallelism within the processor itself is a reality.

The beginning of what is now known as the era of computer started as most people would agree, with Turing machines [4] and the Von Newman organization [5]. The Turing machine is essentially sequential. It basic principles are still used because a very simple Turing machine can simulate all the functions that computers can perform. A more detailed description of Turing machines is beyond the scope of this short history. However, the Von Newman organization will be treated in more details since it is at the base of almost every computer available today and these computers are used as comparison for the parallel processors. Some of the computing devices throughout history are listed in figure 2.1.

The Von Newman computer organization is formed of 4 major blocks: the control unit, the arithmetic-logic unit(ALU), the memory unit and the input-ouput unit(as shown in fig. 2.2). The control unit interprets the instructions fetched from memory and controls the three other units. The ALU does the data transformation: the mathematical and logical operations. The memory unit stores programs and data. The I/O unit interfaces the system to the real world.

BC	Abacus
1615	Napier's Bones
1625	Slide Rule
1642	Pascal Calculator
1728	French Weaving Machine
1822	Babbage's Difference Engine
1833	Babbage's Analytic Engine
1874	Desk Calculator
1930	Bush's Mechanical Differential Analysers
1936	Turing's Machine
	Zuse's Computers(binary+floating point)
1937	Bell's Relay Computers
	Aiken's Mark 1
1938	Analog Computers
1943	ENIAC (18000 tubes)
1945	EDVAC (mercury delay lines)
1947	Whirlwind 1 (cathode ray tube memory)
1954	TRADIC (transistors)
1967	IBM 360 (integrated circuits)
1970	Cheap Portable Calculators
1971	PEPE
1972	ILLIAC IV
	STARAN
	TI ASC
1973	CDC STAR-100

Figure 2.1-Some Computing Devices throughout History[6,10,11]

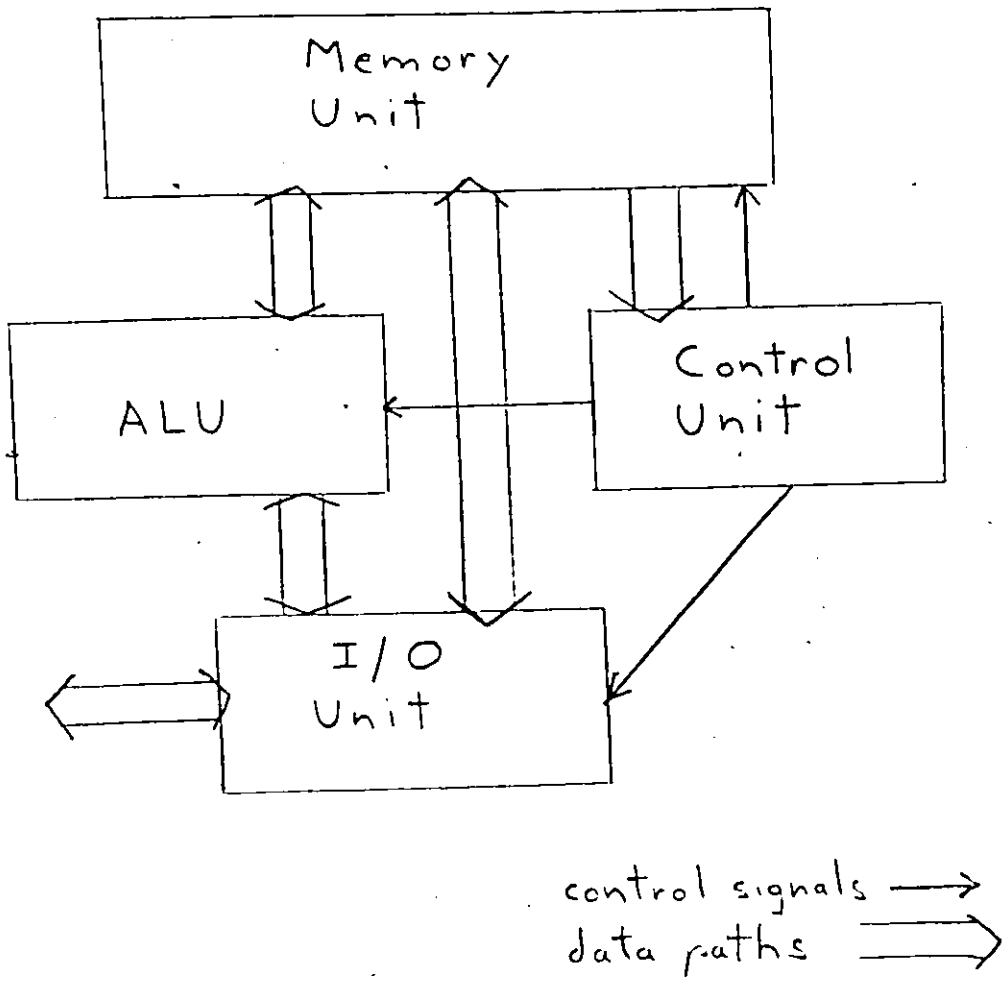


Fig. 2.2-The Von Newman Organization

The I/O operations, the bottleneck of early computers, are what prompted the first departure from the Von Newmann organization and finally led to the present day multiprocessor approach. The machines based upon Von Newman architecture persisted up to the beginning of the sixties. However, already in the mid-fifties the direct memory access (DMA) had made its appearance. The input/ouput data transfers could now be made directly to memory, relieving the ALU and maximizing the use of its time. Never-the-less, the I/O operations were not still completly independent, as they required control signals from the general control unit. This meant that the same control unit was supervising the ALU, the DMA, the I/O and the memory. There was only one central control unit and it was usually the most complicated part of the computer.

The I/O channel came later and is still with us on most major general purpose computers. It is similar in principal to DMA , but it has its own control unit. The I/O channel is almost completly independent of the ALU and the central control unit, provided there are no memory conflicts. However it functions like a slave processor, needing the central contol unit to initiate all operations. The main advantage in having an intelligent I/O processor is that as many I/O channels as needed can be used at the same time. These systems with intelligent I/O processors already

exhibit some parallel processing of the information, but they are not true multiprocessors because I/O processors are too dedicated.

After, or around the same time, various other ways of improving on the basic general purpose computers were tried. They can be classified under four different approaches. Most of these approaches are still in use today.

- Data operator improvements(ALU)
  - multiple execution units
  - pipelining

- Controller improvements
  - microprogramming
  - better instruction set design(index register)
  - direct-execution architectures
    (execute high-level language directly)

- Input/output improvements
  - intelligent I/O devices
  - timesharing

- Memory improvements
  - hierachy of memories(internal,primary,secondary)
  - cache and virtual memory
  - associative cache

At the same time that these efforts in improving computers were proceeding, the use of multicomputers (not processors) was spreading. At first they were mainly used for military applications. The main idea was to improve hardware reliability by having two or more identical computers side by side, one working while the second was idling, ready to be activated in case of failure of the first [6]. Later, the same principle was used to improve information reliability by having two or more computers performing the same calculations.

Another common kind of multiplicity, is multiplicity at bottlenecks within the central processing unit of a computer. For example, having an ALU dedicated to multiplication and division in addition to the usual ALU, or having series of dedicated ALUs in a production line fashion, forming a pipeline processor. This will be discussed in more depth in a later chapter.

The principle of multiplicity at the ALU level and by the same occasion at any other level, when generalized, is the philosophy behind parallel processor systems. The efforts in that direction started as early as 1968 with the ILLIAC computer [7], but with the technology available at the time, it lead to an enormous machine not affordable by most standards. However, with the advent of new cheap

technologies, like the microprocessors and very large scale integration, the multiprocessor architectures are becoming more cost effective.

2.2 TECHNOLOGICAL REASONS

Computer speed has been increasing tremendously in the past 30 years, mainly because of improvements in basic electronic technology. Due to physical constraints, however, this source of speed improvement will probably dry up. Engineers and scientists can not yet get accustomed to this reality, being familiar with exponential improvements in technology. Natural scientist though, know that in nature every curve levels off because of the close system we live in. Some engineers are now realizing that this phenomenon will also strike the electronic technology. Also, at any given time, the computer designer is always limited by the existing state of the art. These are some of the reasons for the turn to parallel processing for further increases in computer throughput.

So far there has not been much effort for major changes in computer architecture because whatever was done one year was always better than the year before. This is due to technological improvements in speed, density, power

consumption, etc... Using today's technology more complicated architectures can be envisaged. Reliability is improving, costs are going down, computer design is becoming automated. What is needed now are computer architectures that suit a changing "number of components vs.performance" criteria.

Today's solution for improvement on computers is through implicit parallelism. Examples of that are: overlapping some functions of the processor, leading to an internal pipeline arrangement; or duplicating parts of the computer where bottlenecks occur. These methods introduce some parallelism in the computer architecture, but this parallelism is hidden from the user. What is obtainned is basically a faster sequential computer at higher cost.

For demanding applications, like radar processing, aerodynamics, and image processing, the two previously mentionned solutions are not usually enough. Explicit parallelism is needed. Fully parallel designs, as exploited by the ILLIAC IV and the others presented in chapter 4, leads to a much larger increase in throughput. The cost increase is usually more directly proportionnal to the speed increase.

Before proceeding any further, let us define a few terms. In a broad sense, parallel processing is defined as the function performed by a system in which multiple operations are done in parallel. These operations can be executed exactly at the same time, simultaneously, or within the same time period, concurently. A multiprocessor or parallel processor is a unified system containning more than one processing element capable of concurrent operation. It could also be added that a multiprocessor system may have shared memory, shared I/O devices, and has an overall controller.

A point to keep in mind, made by B.Raphael in his very good book on artificial intelligence, is "whether computers should be parallel or sequential...are questions of speed, cost and convenience, rather than questions of inherent capabilities" [8]. In other words, the principal reason for the use of parallel processors could only be speed. When the need for a single program throughput rate exceeds that supplied by the largest serial computer, then designers have no other choice than to turn to special architectures. However, special architectures, like multiprocessors, have communication and coordination problems making them more difficult to design and work with.

Hopefully, speed is not the only factor multiprocessors are aiming to improve. There is hope to increase the cost effectiveness of building computers. In an industrial society based on mass production, costs are reduced by using similar parts. Also, reliabilty can be improved with the use of proper operating systems allowing gracefull degradation in the case of failure of one or more processors. Such failsoft systems are necessary for military and other critical industrial applications. Nevertheless, regarding image processing, speed is usually the major aspect of the problem because sequential processors are starting to be limited by the speed of light(tens of picoseconds implying distances of less than three millimeters).

One of the major arguments against the use of parallel processor systems is the accompagnying programming problems. In the context of a parallel processor system enslaved by a host general purpose computer, the programming is a two folds story. The host takes care of the general flow of information and the user interaction. This is usually programmed in high level languages. The slave processor perfprms only the mathematical compute-bound operations. So, the conversions needed to modify a specific system to make use of a parallel processor imply only the conversion of the compute-bound subroutines. These subroutines would

have been previously coded in a low level language. They would have required recoding anyway to obtain all the possible speed advantages ofa newer general purpose computer or of a slave scientific processor.

So view in this light, the software handicap of using a special processor is still big, but not as enormous as one might think at first sight. Specially, if one considers that most of the time spent in software development comes from the development of user interaction, error detection and documentation.

Other disavantages of special processors are:

- -single user mode(batch)
- -programming requires more knowledge of hardware
- -a simulator is usually needed for debugging
- -operating systems are complicated

For the moment, architectures based on parallel processors are mainly used:

1) with large data sets
2) with data suitable for parallel processors
3) for quick throughput
4) for associative addressing
5) for more reliability(graceful degradation)

Complete parallelism is certainly not the only approach to these problems. As mentionned before, implicit parallelism represents a much simpler solution from the point of view of the programmer. The machine still behave basically as a sequential processor. This reflects the philosophy behind what is done at the moment i.e.:putting some parallelism into earlier sequential computer models and keeping it invisible to the user. This is done by adding more arithmetic units or any other units where bottlenecks used to occur. This philosophy is at the base of G.M.Amdahl's success.

Amdahl's arguments in support of sequential computers are the followings [9]:

- -the data management(house keeping) takes twenty to forty percent of the execution time of a usual multiuser system and this is essensially sequential by nature

  -two computers in parallel imply a cost of 2.2 time the cost of one, because of the interface circuitry, leading only to a 1.8 improvement in performance because of conflicting situations;while by using one sequential computer and doubling the cost, the performance quadruples using Grosch's law (performance proportional to cost**2).

Replies to these arguments come easily in the form of:

-parallel processor systems and special architectures in general, are not yet aiming at multiuser system applications

-parallel processor systems usually have a sequential computer as host or within the system to take care of housekeeping

-parallel processor systems are not parallel computers

-there is probably a limit to the extent one can stretch Grosch's law

However, it is true that resource conflicts create a lot of problems with parallel architectures. That is why it is one of the most important aims of research in this domain.

To all these counter arguments, Amdahl is also quoted as saying that with fast parallel processing systems using a sequential computer to do their housekeeping, the system can only be 3 to 4 times faster than the housekeeper. Again this problem can be bypassed by making use of fast buffers or better still, intelligent interfaces built of fast medium scale integration components.

Even if now, no major computer manufacturer tackles the parallel processor business, they eventually must reach it by a direct extrapolation of the present patching of bottlenecks. So, there might as well be now, concurrently with these efforts, research done on multiprocessors that

will eventually support these sequential computer improvements later in time.

At this point in time there are a number of alternatives to true parallelism:

- -concurrent execution of several programs (multiprogramming)
- -I/O simultaneous with execution of programs
- -multiple I/O operations concurrently
- -concurrency of CPU operations

Where as concurrency in CPU operations is now achieved in two ways:

- -parallelism: where functionnal units are replicated (multiprocessing)
- -pipelining: where each instruction is broken down into elementary operations performed by different units one after the other

Nevertheless, with the demand for computer performance being what it is, any possible way of improving processors throughput must be investigated, even if the results are not immediately significant to the general computer industry. There are a number of particular applications for which parallel processor systems can give advantages, and even possibly the only solution.

Finally, at this very point in time, something is making history: the microprocessor. It has upset the balance between memory and logic. Microprocessors are so inexpensive that the logic can be extensively duplicated before adding serious cost increases to a system. Also, the microprocessor is not a naked ALU, but a full grown processor. Microprocessors are paving the way for multiprocessor architectures.

References

[1] Bergamini D., "Les mathematiques", Encyclopedie Le monde des sciences, Life, Amsterdam, Hollande, 1965, p24

[2] Adams J.M. and Haden D.H., "Computers Appreciation, Applications, and Implications", John Wiley and Sons, USA, 1973, 584 pages

[3] Randell B., "The Origins of Digital Computers", Springer-Verlag, New York, 1975, 464 pages

[4] Turing A.M., "On Computable Numbers, with an Application to the Entscheidungs Problem", Proc. London Math. Soc., No. 42, 1936, p. 230

[5] Von Neumann J., "First Draft of a Report on the EDVAC", University of Pensylvania, Philadelphia, 30 June 1945

[6] Enslow P.H., "Multiprocessors and Parallel Processing", Comtre Corporation, John Wiley and Sons Inc., USA, 1974, 340 pages,p. 11

[7] Barnes G.H., Brown R.M., Kato M., Kuck D.J., Slotnick D.L., Strokes R.A., "The ILLIAC IV Computer", IEEE Trans. on Comp., Vol C-17, No. 8, August 1968, p. 746

[8] Raphael B., "The Thinking Computer", Freeman Company, U.S.A., 1976, 322 pages, p.11

[9] Amdahl G.M., "Validity of the Single Processor Approach to Acheiving Large Scale Computing Capabilities", AFIPS, Vol.30, SJCC 1967, p.483

[10] Huskey H.D., Huskey V.R., "Chronology of Computing Devices", IEEE Trans. on Comp., Vol. C-25, No. 12, Dec. 1976, p. 1190

[11] Wu M.S., "Introduction to Computer Data Processing", Harcourt Brace Jovanovich Inc., New York, 1975, p. 8-25

Chapter III

CONCEPTS OF PARALLEL PROCESSOR SYSTEMS

3.1 INTRODUCTION

Having briefly outlined the technical evolution of computers and the historical reasons for using parallel processor systems, we now turn to the concepts on which parallel processor architectures are based. Before debating the pros and cons of different architectures, it is important to first define in a more systematic way, parallel processor architectures with respect to the rest of the computer field.

The systematic way of visualizing a wide field is usually through a classification of its components. There are many approaches to the classification of computer architectures. Most of them are either too general or too limited. Since its is not the purpose of this thesis to persue the general area of classification, a widely known classification will be used. It is also in the author eyes the one which is the most relevant, namely: Flynn's classification [1]. Since

it is a very general classification, we will also follow Higbie's example [12] and break the classes of interest to this thesis into subclasses.

Processor architectures can be divided, as proposed by Flynn, into four major categories based upon the concept of data and instruction streams:

-Single Instruction Stream Single Data Stream (SISD)

-Multiple Intruction Stream Single Data Stream (MISD)

-Single Instruction Stream Multiple Data Stream (SIMD)

-Multiple Instruction Stream Multiple Data Stream (MIMD)

Where a stream is defined as:

"A sequence of data or instructions as seen by the machine during the execution of a program." [1]

It can also be said that these four organizations are characterized by the multiplicity of the hardware provided to service the instruction or data streams.

Of these four architectures, two are of major interest to fast image processing implementations: the SIMD and the MIMD machines. These two and their associated advantages and disadvantages will be examined in more details later. Let us first examine briefly the SISD and the MISD architectures.

3.2 THE SISD ARCHITECTURE

The single instruction stream single data stream architecture is well known in the computer field. In this class are included almost all the present computers, also called serial processors. It represents the typical Von Newman machine with its four basic blocks: memory, ALU, input/output, and control unit,as shown in fig. 2.2. As mentionned before, most of these processors have some degree of parallelism embedded into them in order to make them faster. However, conceptually speaking they are basically serial computers with a few added improvements to solve the bottlenecks of previous models. The major problems are the memory and the execution bandwidths [2].

The memory bandwidth refers to the retrieval rate of operands from memory. For serial computers this depends uniquely on the speed of the memory. The CPU has to wait until the operands are obtained. The execution bandwidth is defined as the rate at which instructions are processed. In a serial machine, only a certain number of instructions can be processed per second, and only one at the time. A major impediment is the inherent sequential nature of instruction decoding. This limitation is discussed later in this chapter.

3.3 THE MISD ARCHITECTURE

The multiple instruction stream single data stream architecture is the most unconventional. It consists in multiple processing units each working on a certain independent task, with one set of data (see fig. 3.1). For example, a MISD structure could be used in a character recognition by template-matching machine [3], where the input characters are to be matched with dozens of templates.

Each processing unit has its private instruction memory, but obtain its data from a common data memory. This implies that the data memory must have a very high bandwidth. This is the usual bottleneck of these systems. To alleviate this bottleneck, some data memory could be added to each processing unit. At this point the system approaches a MIMD system.

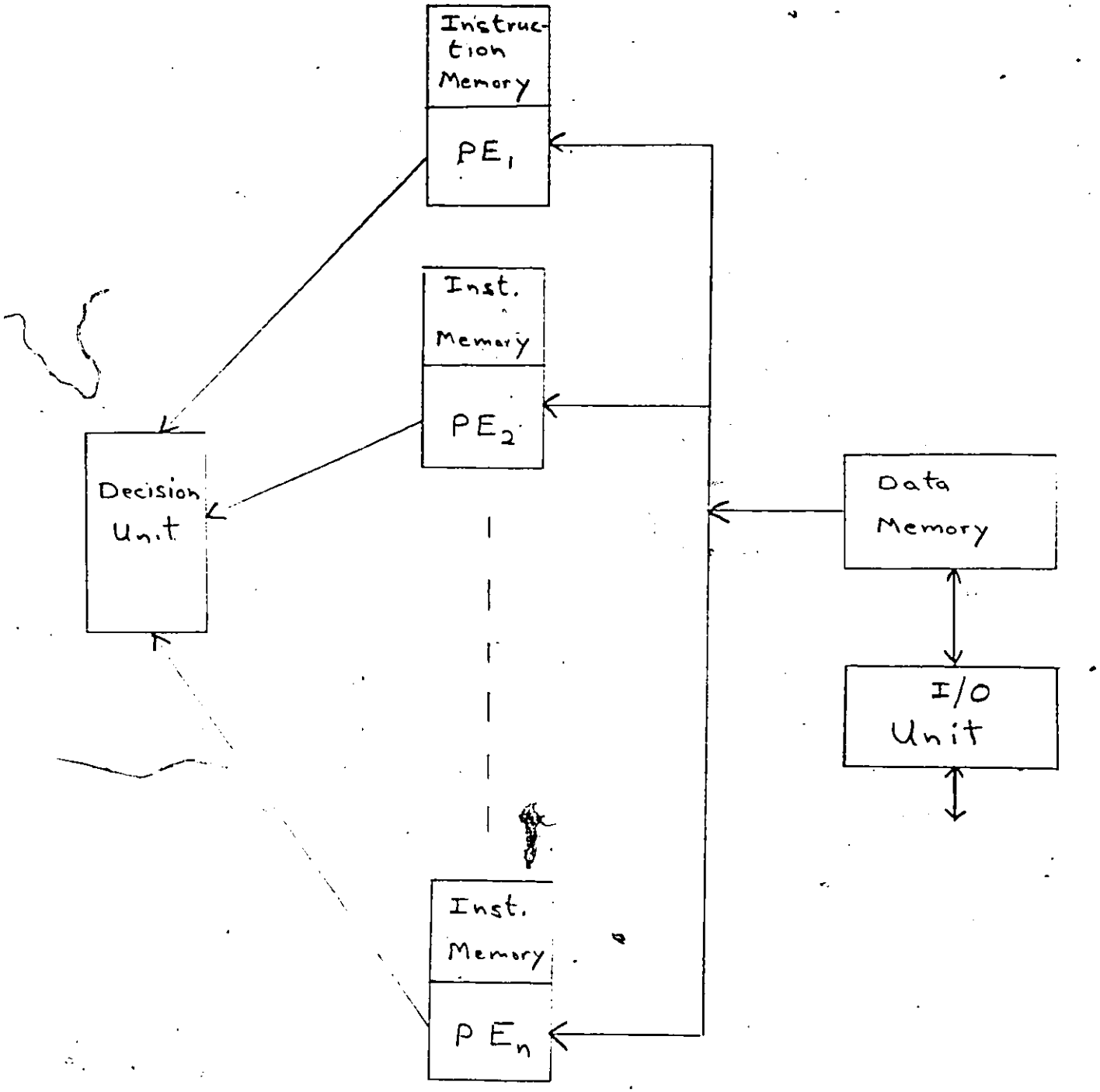


Fig. 3.1-The Multiple Instruction Single Data Stream Architecture

3.4 THE SIMD ARCHITECTURE

The single instruction multiple data stream architectures can be divided into three categories:

-array processor: one control unit and a number of directly connected independent processing elements having their own register and memory (see fig. 3.2)

-pipeline processor: a number of specialized execution units arranged in an assembly line fashion (see fig. 3.3)

-associative processor: operating like an array processor, but the processing elements are activated depending on the nature of the data (see fig. 3.4).

A major characteristic of array processors is that since the processing elements are under the control of one unit, some degradation in performance is unavoidable when branches are encountered. This is due to the fact that the PEs are not capable of independent operations. So, when the control unit is generating signals for one of the branch option, the PEs having the second option to follow must idle until their control signals are generated. A typical example of an array processor is the ILLIAC IV examined in chapter four.

Pipeline processors are often subject of controversy:

"The pipeline processors can be considered either SIMD, MISD or MIMD architectures." [4]

Some people will argue that the pipeline architecture is a MISD system, others, that it is a SIMD system. A MISD system, because every stage of the pipeline receives its own instructions (hardwired mostly) and there is only one stream of data coming in or out of the pipeline as a whole. A SIMD system, because it is in fact only one instruction which is broken into parts to activate all the pipeline's stages and that at every stage two operands are used, making it a multiple data stream system. This hard to classify processor can also be called a SISD machine for similar reasons. These are philosophical arguments which can be debated for a long time. It all depends at what depth of hardware they are looking to make their statement. In this chapter, the pipeline architecture is going to be classified as a SIMD system, since it can be analysed in similar ways and it is known to present similar problems. Examples of pipeline processors are the CDC STAR-100 and the TI-ASC [5].

In an array processor, the processing elements are disabled following the reception of a signal from the control unit. The difference for an associative processor lies in the fact that the PEs are enabled or disabled depending on tests performed by the PE on its associated data. Never-the-less, associative processors are faced with

the degradation of performance caused by branching. The STARAN system described in chapter four is a typical example of an associative processor.

The SIMD stream machines are best used with a language such as APL(A Processing Language) that processes vectors and matrices efficiently [6]. Rather than operating on each vector or matrix elements individually, the operations are performed on the whole vector. SIMD organizations are also very good for non-numeric processes, like in data management systems. This is particularly true of associative array processors [7].

Five major problems are usually encountered when dealing with single instruction multiple data stream organizations:

1) Communications between processing elements
2) Vector fitting(logical vs physical)
3) Sequential code(Amdahl effect)[8]
4) Degradation due to branching
5) Performance proportionnal to the log2 of the number of data stream(Minsky)[9]

Communication between PEs is difficult to implement because of timing considerations and data paths requirements. Fortunatly, communication is not considered to exceed forty percent of the total job time, except maybe in some special image processing tasks [10].

Vector fitting is a problem encountered when the vector needed by a program is longer than the maximun length the hardware can process immediately. Special software is used to split the vectors into shorter parts. This implies a fixed overhead.

Sequential coding or the Amdalh effect is the recognition of the lack of inherent parallelism in most programs or algorithms. This apparent sequential nature of programs could change with time as more and more parallel processors become available and motivate changes in programmer habits and programming languages.

Problems four and five are related. Minsky is responsible for the conjecture stating that the performance is proportionnal to the log2 of the number of data stream. This was obtained experimentally, but it is mainly explained by the degradation caused by branching. Under a single control unit, if a certain number of processing elements are to "branch" one way while the others "branch" another way, some will have to wait for their control signals. In other words, given equal probabilities, only half the PEs will be in operation, the other half being idle. If another branch occurs within one of the branch options, then, only a quarter of the possible PEs will be operating, etc...

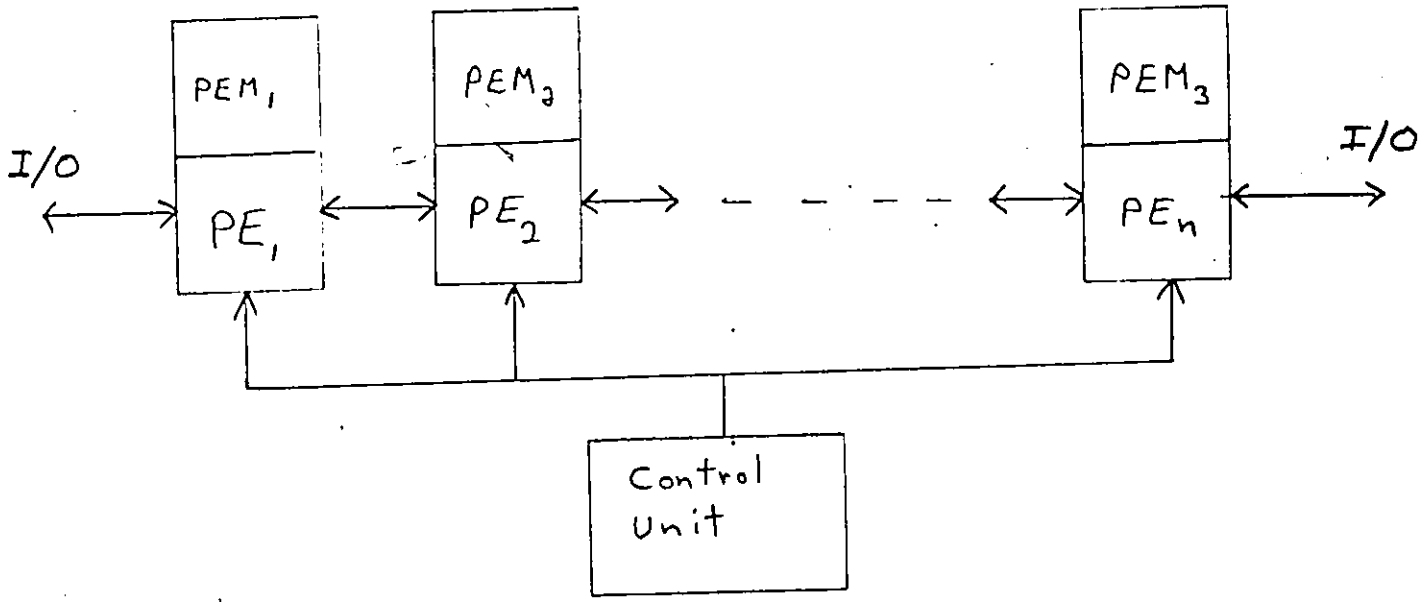


Fig. 3.2-The Array Processor (SIMD)

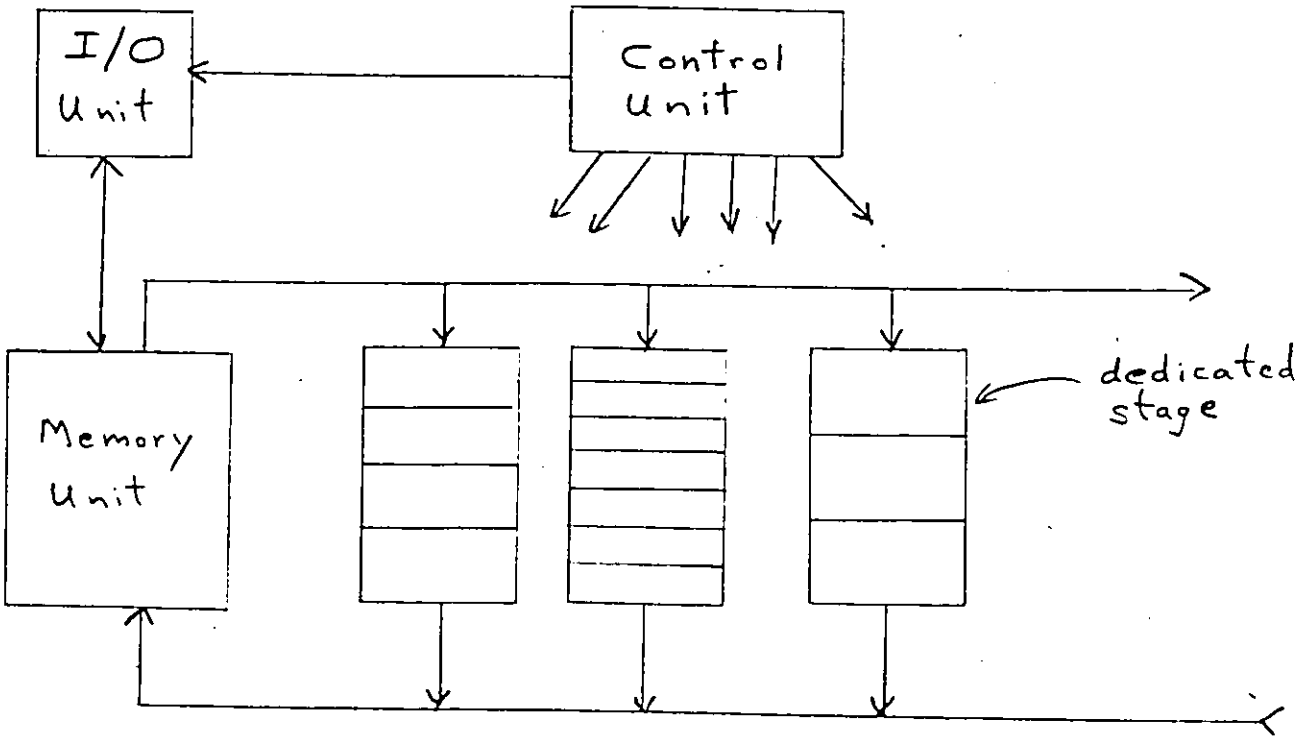


Fig. 3.3-The Pipeline Processor (SIMD)

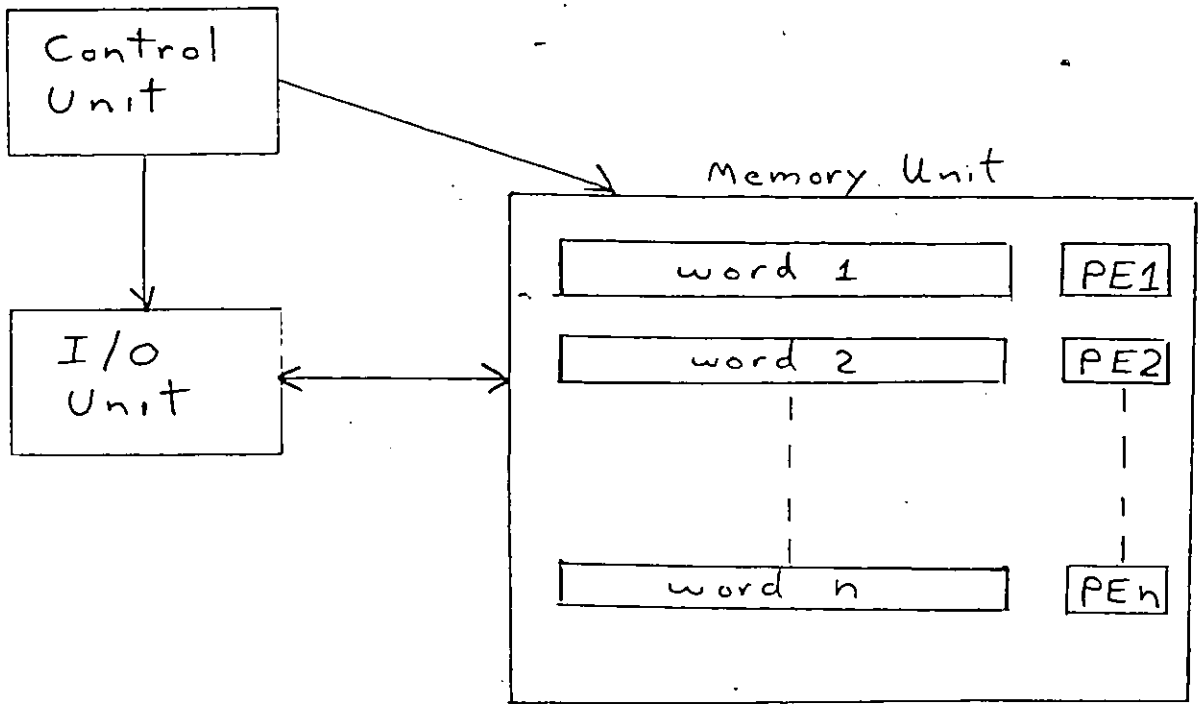


Fig. 3.4-The Associative Processor(SIMD)

3.5 THE MIMD ARCHITECTURE

The most complex and interesting architectures are usualy found in the multiple instruction multiple data stream machines. They can be divided into two principal categories:

1) Shared resources multiprocessors: skeleton processors sharing the system resources (see fig. 3.5).

2) True multiprocessors: independent processors with some shared resources and cooperative execution (see fig. 3.6)

Shared resources multiprocessor systems often encounter lockouts precisely because of the sharing of resources. In that case, time shared resources are considered to be better than shared resources using a crossbar system. In the later, extensive delays at one point have a tendency to cause more subsequent delays, in a cumulative fashion.

Because of the self-suficiency of their PEs, true MIMD multiprocessor systems will encounter mainly communication delays rather than resource delays. This is true of system where the task execution times of individual processors are vastly different, obliging some to wait for others in order to get transfered information.

MIMD systems are much more adaptive than any other multiprocessor systems, but are usually more difficult to

design and considerably harder to program. The difficulty in design arises from the need to implement the sharing of resources and the communication capabilities. This implies a certain quantity of communication paths. From a performance point of view, the number of interconnection paths has to be maximized. However, for practical design, it has to be minimized because busses use large amounts of space on printed circuit boards and are subjected to all sorts of noise problems.

The difficulty in programming is due to the parallelism of operations and the need of the PEs to communicate with each other. The parallelism of operations often implies that the programmers have to re-educate themselves to cope with this new problem. The distribution of work among the processors brings decisions not often encountered in sequential programming. The need for inter-PE communication brings all kinds of delays that are to be evaluated in advance when programming.

An advantage of MIMD over SIMD systems is that they do not encounter severe degradation of performance when branching occurs. Since each processing element follows its own program, both options of a branch can be persued simultaneously by different PEs. This is possible at the expense of more memory, since programs or parts of program may have to be replicated in each PE's memory.

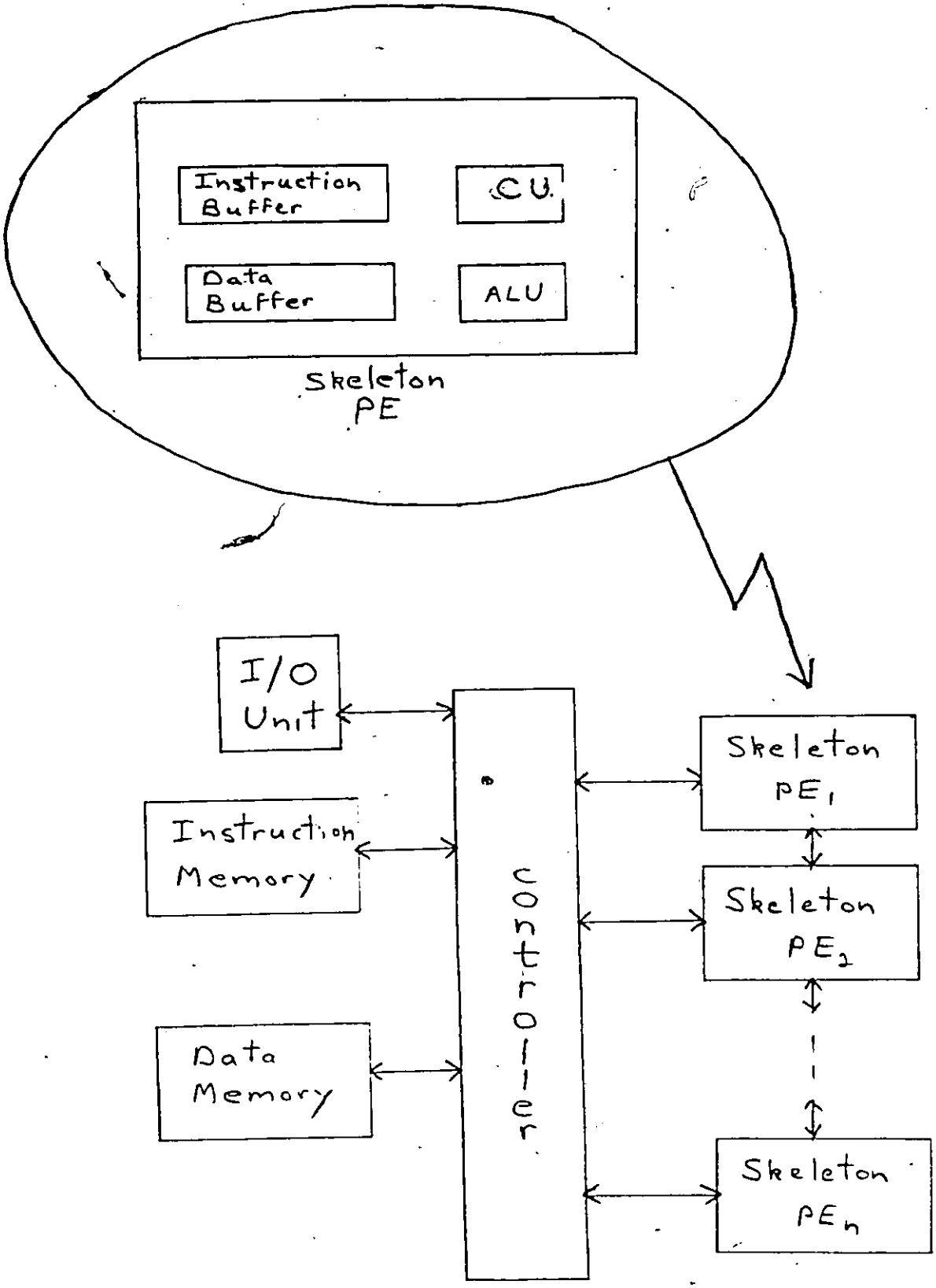


Fig. 3.5-The Shared Resources Multiprocessor (MIMD)

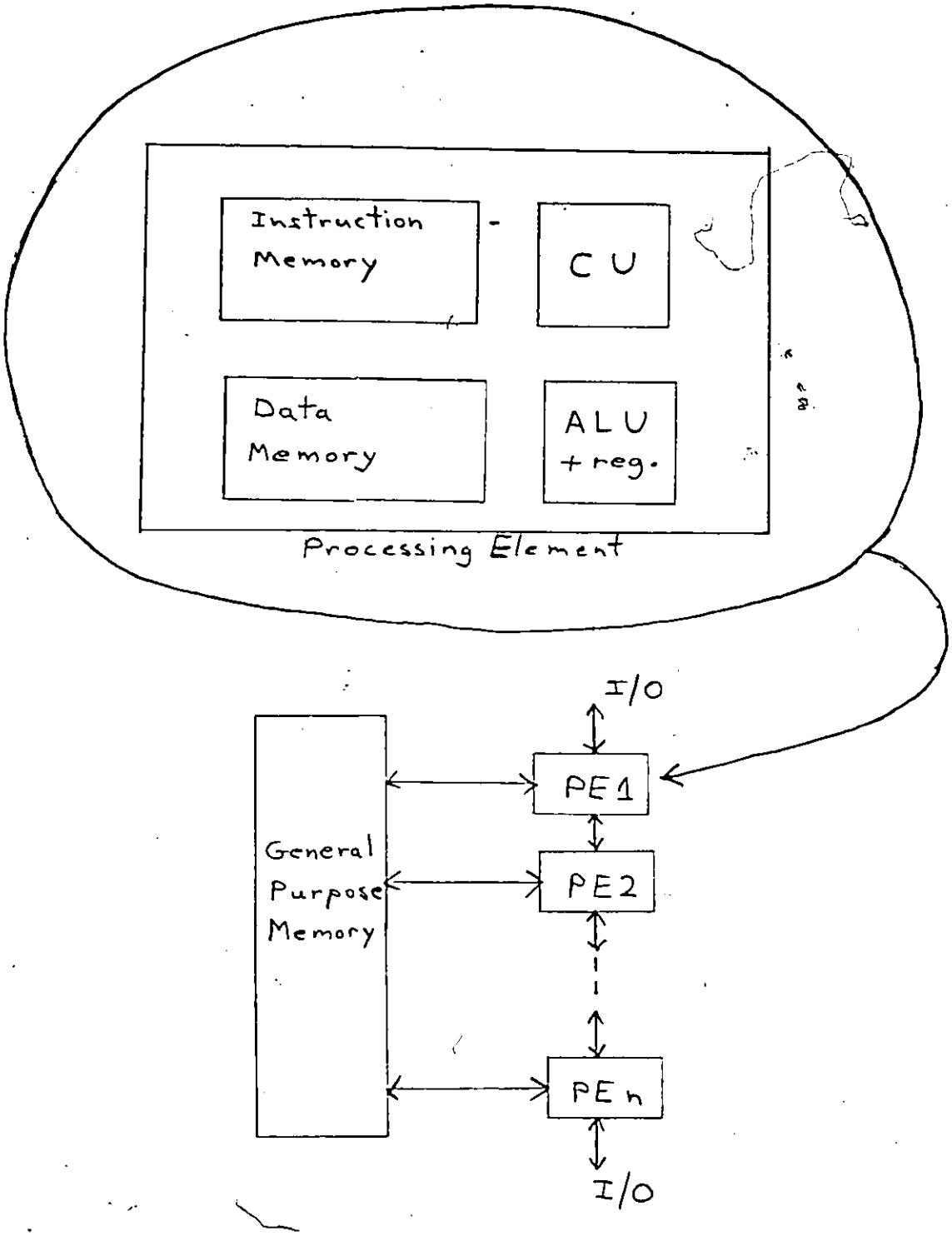


Fig. 3.6-The True Multiprocessor (MIMD)

3.6 COMMON CHARACTERISTICS OF PROCESSING SYSTEMS

The evaluation of parallel processing systems should be based on their efficiency on a particular class of problems of interest, their reliability and their ease of maintenance. Most of these systems are limited mainly by their input/output operations. Their effectiveness depends on the nature of the problem and on the availability of new algorithms oriented toward parallel processing.

Common to these four categories of parallel processors are two major limitations encountered in trying to achieve fast speeds: the execution bandwidth and the memory bandwidth (explained in sect. 3.2). These two problems are solved partially in many different ways and depend upon the architecture of interest.

A problem, related to the execution bandwidth, is the sequential nature of instruction decoding. In relation to this problem, we recall Flynn's analysis [1] of an instruction stream, in order to understand how conditionnal branching affects the execution bandwidth.

The basic time in this analysis is going to be the instruction decoding time Δt. We can define the average instruction execution time as a multiple of the decoding time, as: $L*\Delta t$. Let us define J as being the number of

instructions processed (decoded and started) during the latency time (1) of one complete instruction execution. J is called the inertia factor (see fig. 3.7). For optimal performance, we want to issue an instruction every $L*\Delta t/J$ time units. In other words, we expect M instructions within the program to take an average total time of:

$$T = M(L*\Delta t/J)$$

if no conditional branchings (turbulances) are encountered.

Let us now examine the delays caused by encountering such turbulances and their effects on these M instructions. Thinking in terms of very low level of programming, let us define an anticipation factor N as being the average number of instructions between the instruction stating the conditional branch and the instruction which tests and acts upon that condition. We can then expect a delay of :

$$\text{delay} = L*\Delta t - N(L*\Delta t/J)$$

for N smaller than J and a delay of zero for N greater or equal to J (fig. 3.8).

Note that if N=0 (that is, there is no other instruction between the two instructions related to the condition), we can expect a maximun delay of $L*\Delta t$, which is one complete instruction execution cycle.

---

(1) Latency: total time associated with the processing of a particular data unit

Now, let us examine this effect on the average total execution time of a pack of M instructions, if we suppose a probality p of encountering a turbulence.

$$T = (1-p)(M-1)(L*\Delta t/J) + p(M-1)(L-N*L/J)\Delta t + L*\Delta t/J$$

If we define the performance as being:

$$perf. = M/T$$

then

$$perf. = 1/[(1-p)(L*\Delta t/J) + p(L-NL/J)\Delta t + L*\Delta t/JM - (1-p)(L*\Delta t)/MJ - (p/M)(L-NL/J)\Delta t] \quad (2)$$

when M becomes large:

$$perf. = 1/[(1-p)(L*\Delta t/J) + p(L-NL/J)\Delta t]$$

$$perf. = J/[(L*\Delta t)(1+p(J-N-1))]$$

By plotting this equation, it is now possible to see the terrible degradation due to conditional branching when trying to speed-up a single instruction stream single data stream machine. The equation was plotted for different probabilities (p) of encountering turbulances and for L=20 time units and N=2 instructions (see fig. 3.9). For example, if J=20 and the turbulance probability is 10 percent, the performance degrades by 63 percent compared to the maximun performance obtained when there is no turbulance.

---

(2) Note that this expression is somewhat different from Flynn's because it is developed more extensivly.

Single instruction stream multiple data stream machines are also similarly affected. However, in this case it is worse since conditional branching also implies having a number of processing elements idling. In multiple instruction multiple data streams machines each particular PE is affected as in a SISD system. Typically, conditional branching represents ten to twenty percent of ordinary computer program instructions [11].

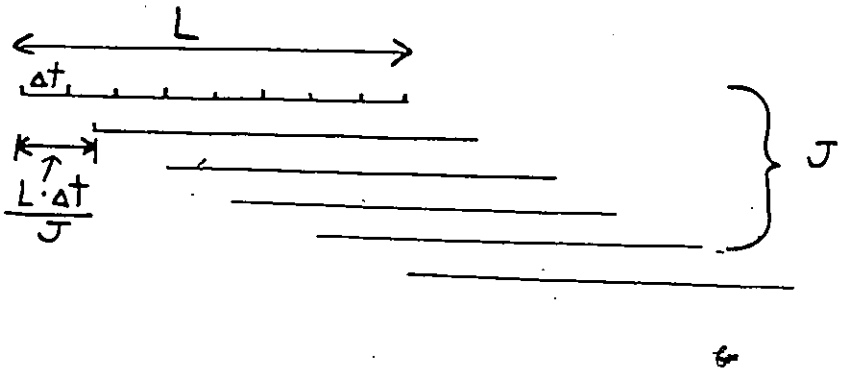


Fig. 3.7-The Inertia of Instruction Execution [1]

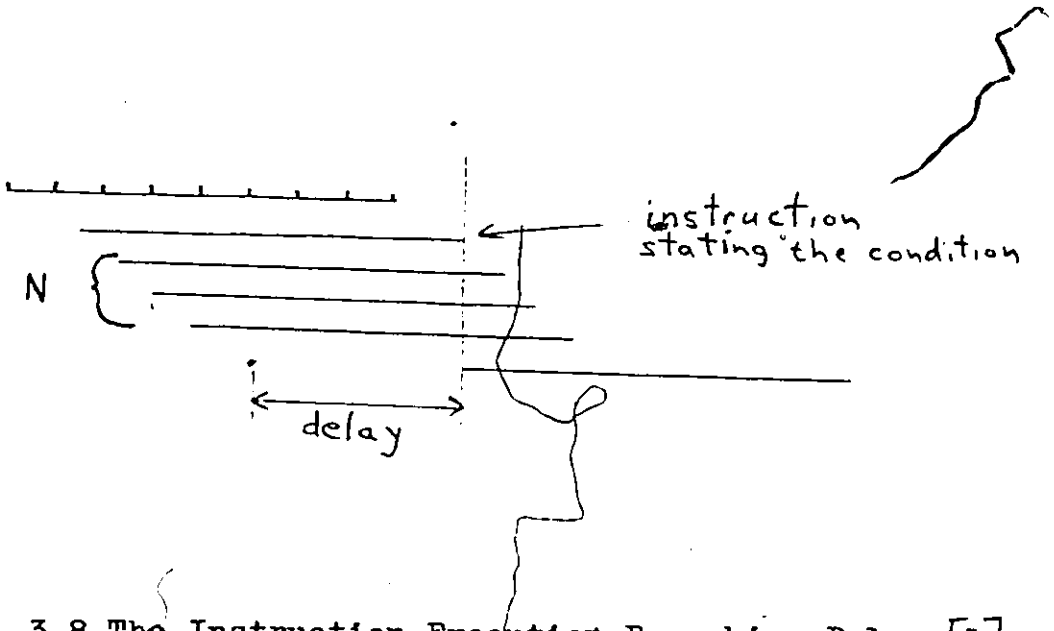


Fig. 3.8-The Instruction Execution Branching Delay [1]

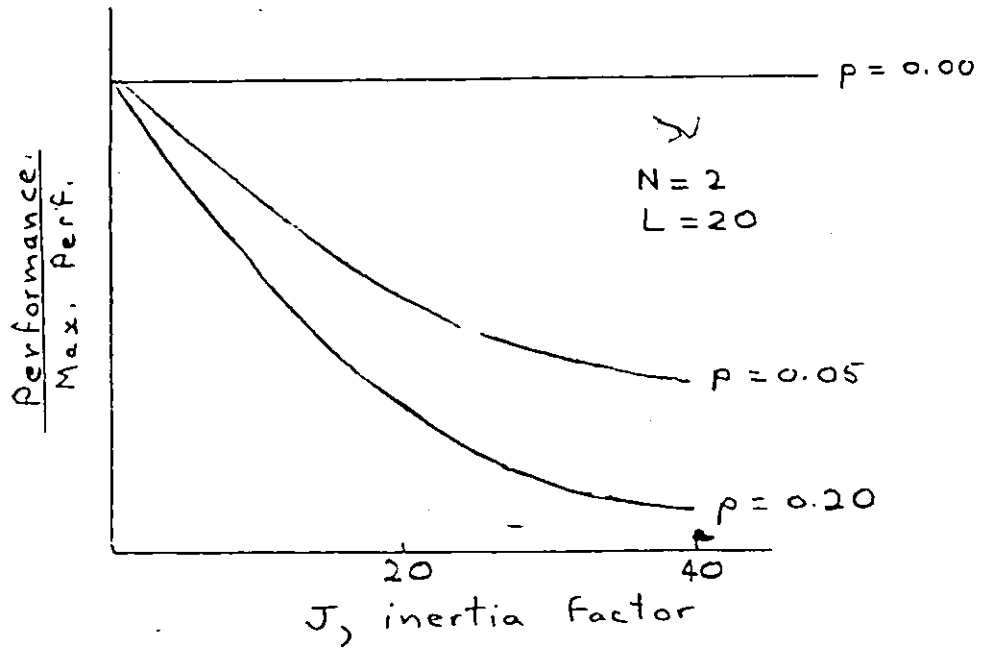


Fig. 3.9-The Stream Inertia and the Effects of Branch [1]

3.7 CONCLUSION

In this chapter, Flynn's four different processor architecture categories are examined. They are characterized by the number of streams used for the instructions and the data. This is related directly to the multiplicity of the hardware in each category.

The advantages and the disadvantages of each category are mentionned. Also, some of the common caracteristics and problems of these architectures are discussed. This leads to the conclusion that the SIMD and the MIMD architectures are both very promising, provided they are used for the right applications. Figure 3.10 presents a table of the hardware differences, the advantages, and the disadvantages of each group.

			SISD	MISD	SIMD	MIMD
Hardware	Control Unit		1	n	1	n
	ALU		1	n	n	n
	Memory	Data	1	1	n	n
		Inst.		n	1	n
Communication		Inter PE		Almost no communication	Bidirectional to neighbors	Bidiractional unlimited
		External I/O	I/O Unit	I/O Unit	Edge PEs	All PEs
Advantages			Easy to program because sequential	Very good for branching	Fast processing for large data sets	Fast for complex tasks with large data sets Most versatile
Disadvantages			Slow, repititious, because datum treated one by one	Data memory must have a high bandwidth otherwise slow	Single control unit implies branching degradation	Communication between PEs hard to realize. Hard to program. Distribution of work load difficult

Fig. 3.10 - General Architectures Comparison

References

[1] Flynn M.J., "Some Computer Organizations and their Effectiveness", IEEE Trans. on Computer, Vol. C-21, No. 9, Sept. 1972, p948

[2] Flynn M.J., "Very High Speed Computing Systems", Proc. of IEEE, Vol. 54, No. 12, Dec. 1966, p.1901

[3] Freedman M.D., "Optical Character Recognition", IEEE Spectrum, Vol. 11, March 1974, p. 44

[4] Braer J.-L., "Multiprocessing Systems", IEEE Trans. on Computers, Vol. C-25, No. 12, Dec. 76, p. 1272

[5] Strokes R.A., "An Assessment of First Generation Vector Computers", Compcon, San Francisco, Feb. 28-Mar. 3, 1977, IEEE Comp. Soc., p. 12

[6] Ung V., "An Efficient Multiprocessor Architecture", Proc. of the 1976 Int. Conf. on Parallel Processing, IEEE, p. 213

[7] Berra P.B., "The Role of Associative Array Processors in Data Base Machine Architecture", Computer, IEEE, March 1979, p. 53

[8] Amdahl G.M., "Validity of the Single Processor Approach to Acheiving Large Scale Computing Capabilities", AFIPS, Vol. 30, SJCC 1967, p. 483

[9] Minsky M., Papert S., "On Some Associative, Parallel, and Analog Computation", Associative Information Techniques, New York:Elsevier, 1971

[10] Newhauser C., "Communications in Parallel Processors", Johns Hopkins Univ., Baltimore, Dec. 1971

[11] O'Regan M.E., "A Study of the Effect of the Data Dependent Branch on High Speed Computing Systems", M.S. Thesis, Dep. Ind. Eng., Northwestern Univ., Evanston, Illinois, 1969

[12] Higbie L.C., "Supercomputer Architecture", Computer, IEEE Press, Vol. 6, No. 12,, Dec. 1973, p. 48

Chapter IV

SOME EXISTING SYSTEMS

4.1 INTRODUCTION

In this chapter, some existing parallel processor systems are examined. They are classified according to Flynn's categories, so that the general observations of chapter three still apply. However, more specific hardware observations are made. This is summerized in a table at the end of this chapter.

Four existing systems are described: the ILLIAC IV, the STARAN, the PEPE and the FMPP. Two of these are SIMD architectures, while the other two are MIMD architectures. At least three of these four architectures are not of recent origin. They were selected precisely because there is much more information available on these older systems. Also, they illustrate the diversity within each category of processors. The FMPP is introduced to illustrate a more dedicated processor, directly related to image processing.

Newer multiprocessor systems are now available. However, because of their novelty and the fact that they are commercially produced, not much information is available on their real architecture. Even the four systems of this chapter are described mainly in general terms. An effort has been made to generalize their block diagram representation in terms of only control units, processing elements, memory and input/output. This should allow easy comparisons.

4.2 THE ILLIAC IV SYSTEM

4.2.1 Introduction

The ILLIAC IV is a parallel array computer inheriting its basic design ideas from the SOLOMON system(1). It is classified as a single instruction multiple data stream machine. The ILLIAC IV is several orders of magnitude faster than a conventionnal computer for the computation of large matrices, the solution of partial differantial equations, or data correlation [2].

---

(1) The SOLOMON system is the initial design, proposed mainly by Slotnick, that eventually lead to the ILLIAC [1]

4.2.2 Hardware [3] [4]

The ILLIAC IV has 256 processing elements(PEs) organized into 4 independent arrays of 64 PEs each. Each of these arrays is under the management of a control unit and can be used individualy, or with one or more of the other three, depending on the problem's needs. The configuration used is chosen by the host general purpose computer that supervises the whole system.

In each array the control unit(CU) decodes the instructions coming from the PE's memory and generates the control pulses for the PEs. It also takes care of the distribution of constants and of the memory addresses common to all PEs. The PEs all work in parallel executing the same instruction. However, each PE decides internally whether to be active or not, depending upon its mode register. The index register in each PE modifies the common address sent by the CU. The CU also manages the serial instructions and the control loops.

The instructions are 32 bits long. They may affect only the CU or the CU may decode them for the PEs. Instructions come to the CU from the PE memory in a queue and leave the CU as control signals via another queue. Apart from these, the CU has another input buffer: the local data buffer. Both input queues are 64 words(64 bits) long. The CU also

possesses an arithmetic unit, to perform the sequential calculations needed, and four 64-bit registers. Clever programming can lead to semi-parallelism between the CU and the PEs.

The advantages of having an instruction queue is that if the queue is long enough (in this case 128 instructions), a programing loop can fit in entirely, requiring no further instruction fetching for the duration of the loop.

The individual processing elements (PEs) have their own index register, a memory adder circuit, four 64 bits general purpose registers, adder and multiplier circuits, logic unit and mode register. Each PE is itself divided into subprocessors used alone or in groups to form any integer combination of 8 bits, up to 64 bits. Also, every PE has its own almost private memory consisting of 2048 words of thin film memory. Here, "almost private memory" means that the memory can be accessed by the PE, the CU, or the I/O connections, but not directly by other PEs. Each PE can add 64-bit operands in 240 nsec and multiply them in 400 nsec.

These 64 PEs in an array are logically arranged to function as an 8*8 square or matrix with each PE having direct connections with the preceeding PE, the following PE, the one over and the one under it. In other words, looking

at it serially, a PE "i" is connected with PEs i-1, i+1,i-8, and i+8, in the same way as the SOLOMON computer. So matrices are stored in a 8*8 format and if higher dimensions are needed, they are stored in the PEs memory(PEM). Big matrices of 2 dimensions are stored, one dimension as a vector, the other in the PEM. Ordinarely, they are stored in a skewed format to help processing. Matrices bigger then 256*256 are partitionned into severals smaller submatrices and stored in the PEM.

Input/output operations are performed using the 16 edge PEs and a 1024-lines bus switchable among the arrays. The data are stored in a 16 k-bits buffer (256*64 bits) to match the ILLIAC speed with the host. In general the host machine takes care of the interfacing with the outside world, but the arrays can be connected directly using a fast real time link to any device fast enough to receive and transmit at the proper rate. The arrays also have their own backup memory facility, a large parallel-access disk system.

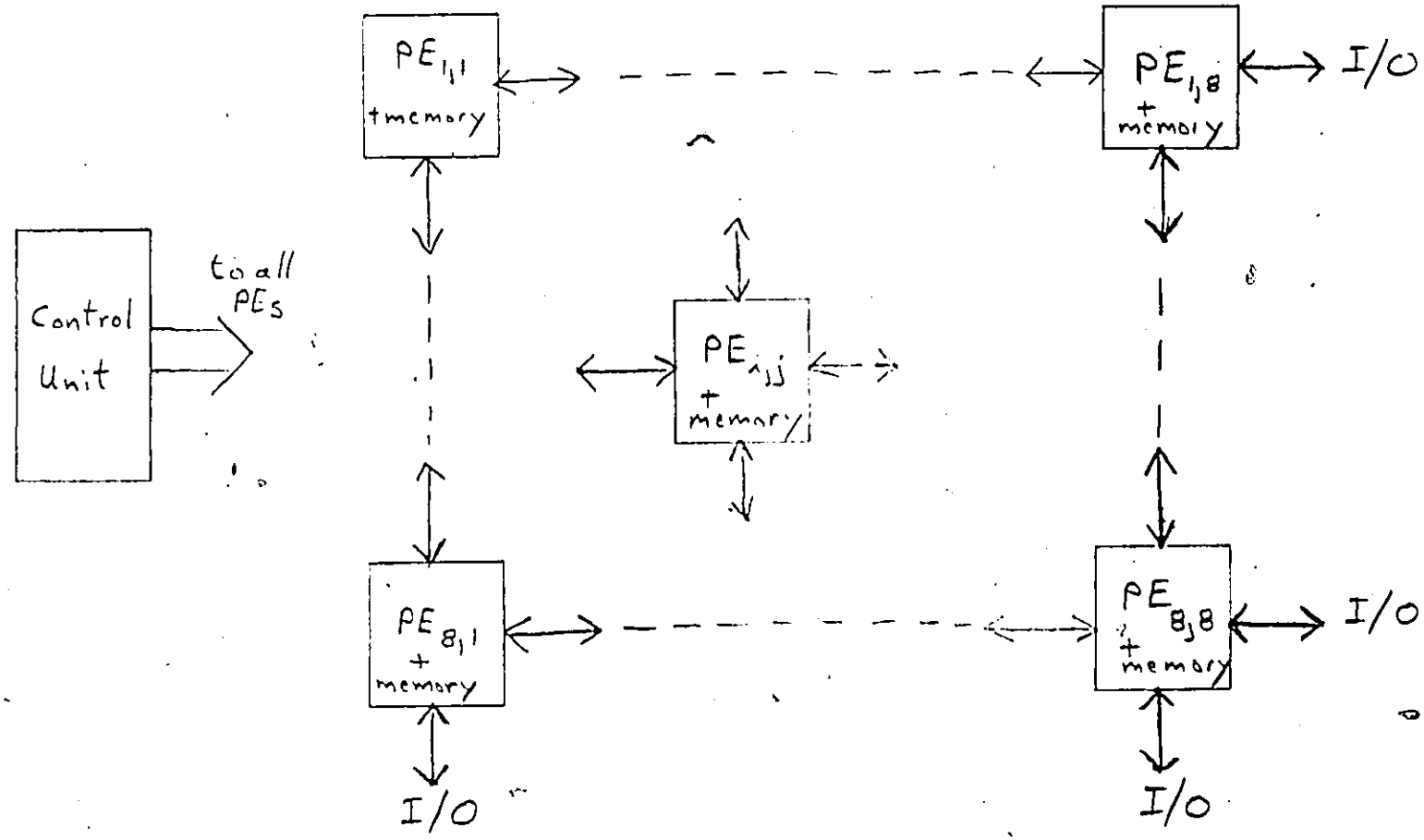


Fig. 4.1-The ILLIAC General Block Diagram

4.2.3 Remarks

The ILLIAC IV is available to every user either in batch mode or using timesharing terminals. In fact the ILLIAC IV is just another timeshared device except that it has the highest priority. Programs are translated by the host and sent to ILLIAC's disk. At run time, the ILLIAC fetches the program from disk, executes it, and stores the results as a data set back on the disk.

For operations on matrices with sizes that are compatible with ILLIAC (multiple of 256), speedup of two or three decimal orders of magnitude over the CDC6600 (c.c.: 110 nsec) have been reported [2]. In these cases, matrix data were stored in a skewed format to improve efficiency by having as much PEs as possible working at the same time. However, as reported by Bailey[5], who worked with very big matrices, if the data is more than 131,072 words per hardware array, the system has to get data from disk. This slows down the process terribly. Nevertheless, Bailey reported an improvement of a factor of six over a CDC7600 (c.c.: 27.5 nsec) for two dimensional problems and of nine for three dimensions.

The ILLIAC IV is easily programmable using a language called TRANQUIL which is just a subset of instructions added to ALGOL or FORTRAN. Storage is allocated by dimension

declaration statements. Data can be skewed easily by one instruction, SKEW. The disposition of data within an array is the responsability of the compiler.

One major drawback in systems like the ILLIAC IV is that all the PEs receive their control signals from a single source, the control unit. This implies that if a branching condition occurs, the CU has to follow all the paths one after the other, in order to provide the PEs with the proper signals. This situation was described in chapter 3 and considered one of the major factors for lost of efficiency in SIMD systems.

4.3 THE STARAN SPECIAL PROCESSOR

4.3.1 Introduction

The Goodyear Aerospace Corporation's associative array of parallel processors, STARAN, is one of the few non-conventional processors ever to be built in more than one copy. STARAN is known mainly as part of the RADCAP system in Rome Air Development Center (RADC) because of the number of papers published on its performance there [6] [7]. At RADC it works in conjunction with a Honeywell HIS645

sequential computer which handles the users'terminals and interfaces them with STARAN via one of its input/output channels or via a custom input/output unit. But STARAN can also be used in a stand alone mode since it has a PDP-11 to take care of its mass storage system and some I/O devices.

STARAN is also used by the LACIE(Large Area Crop Inventory Experiment) project at NASA where it relieves five IBM360/75 from a large part of their workload [8] [9]. In this context, the special processor is almost exclusively used for image processing and is fast enough for interactive image processing.

4.3.2 Hardware

Used in conjunction with a host, STARAN looks like an associative memory. However, an associative memory would only help in the search for specific bit patterns and signal them to the CPU for processing, while STARAN, once the data has been found, has the capability of performing immediately the required operations on the data. It brings the processing power to the data instead of bringing the data to the processor. This is what is called associative processing. It can perform search, arithmetic operations or logical operations, on all or selected words(256-bits) of its memory.

There are basically two modes of access to STARAN's memory: by bit-slice or by word. Bit-slice access allows parallel vector arithmetic and content addressable search, while conventional word access is used for input/output and scalar arithmetic. So, to access any particular bit, the programmer uses a word stencil or mask and a bit-slice stencil.

The STARAN is classified as a single instruction multiple data stream machine. It is a high speed asynchronous processor which can perform a search at 150nsec/bit or an addition at 800nsec/bit, but on 256 words at the same time. It is a typical computer of the sixties in the sense that it is built of dual in-line integrated circuits(ICs) on multilayer printed circuit boards. A STARAN with one array requires roughly 9,000 ICs, with two arrays, 11,500; and the semi-intelligent interface with the HIS645 at RADC, about 8,000 ICs.

STARAN consists of up to 32 associative arrays, each of which has 256 processing elements and 65,536 (256*256) bits of storage (see fig. 4.2). These arrays are under the control of the array processor(AP) controller which gets its instructions(microprogram) from the AP control memory. The AP control memory itself, which holds the 32 bits instructions, receives these from the PDP-11 sequential

controller or the big computer which assembles STARAN programs.

Using STARAN interfaced with another processor, as in the RADCAP system, can be done easily since STARAN's AP controller or sequential controller can address directly the processors's memory via a 32 bits direct memory access bus. The processor can do the same via another 32 bits bus called buffered input/output. This usually requires more than an ordinary I/O channel to the host, rather an intelligent interface, almost a processor. Exchange of control signals is done via the external function logic.

A STARAN array module contains 256 processing elements with each PE taking care of 1 bit. However, it is simpler to visualize the system as one big 256-bit vector processor. In this case, the processor would have three 256 bits long registers, M, X, and Y, and a 256 bits long flip network. The M register is just used as a mask for some WRITE operations into the memory while the X and Y registers are the working registers. To accomodate both word by word input/output and bit-slice operations every STARAN array module is equipped with a 256*256 bits multi-dimensional access memory.

The flip network(scramble/unscramble network), which is an inherent part of every array (as shown in fig. 4.2), receives control signals and influences directly the operation on the X or the Y registers [10]. It also allows inter-PE communications. With the help of the flip network a PE can access any two words or parts of words in memory, add them, and store the result elsewhere. This is particularly useful in fast Fourier transforms where the network can scramble and unscramble data while transferring from memory to PE, PE to memory, or between PEs. Also, shifts are done by the flip network, requiring no PE time. The flip network permutes data according to a flip-permutation control word and shift data as specified by the shift control word, from source(memory,PE) to destination (memory,PE). Almost any permutation and/or shift can be performed in one or at most a few passes through the flip network.

The scramble/unscramble network is used in every store or fetch operations, effectively replacing the skewing of data in memory as in the ILLIAC IV [7]. In fact, the scrambling in STARAN requires only N exclusive-or circuits while the skewing would require 2**N adders for local address generation. The basic idea behind scrambled or skewed data is to allow multi-dimensional access with ordinary random access memory.

The AP controller which is the brain of the system and supervises everything at a more concrete level, has:

-a 32 bits instruction register

-a program counter

-a 32 bits operand register

-an array select register

-4 field pointers for MDA(Multi-dimensionnal Access) memory addresses

-3 counters for loops

-a data pointer

-2 access mode registers

to help the assembly language programmers in their task of keeping track of everything.

Of extreme importance in any parallel processor is the time wasted in transferring the data from the mass storage system. With the STARAN computer four major ways are provided for fast transfers:

-I/O using the sequential controller(1 Mbytes/sec)

-parallel I/O (PIO) (80 Mbytes/sec/array)

-direct memory access (DMA) (2 Mbytes/sec)

-buffered I/O (BIO) (2 Mbytes/sec)

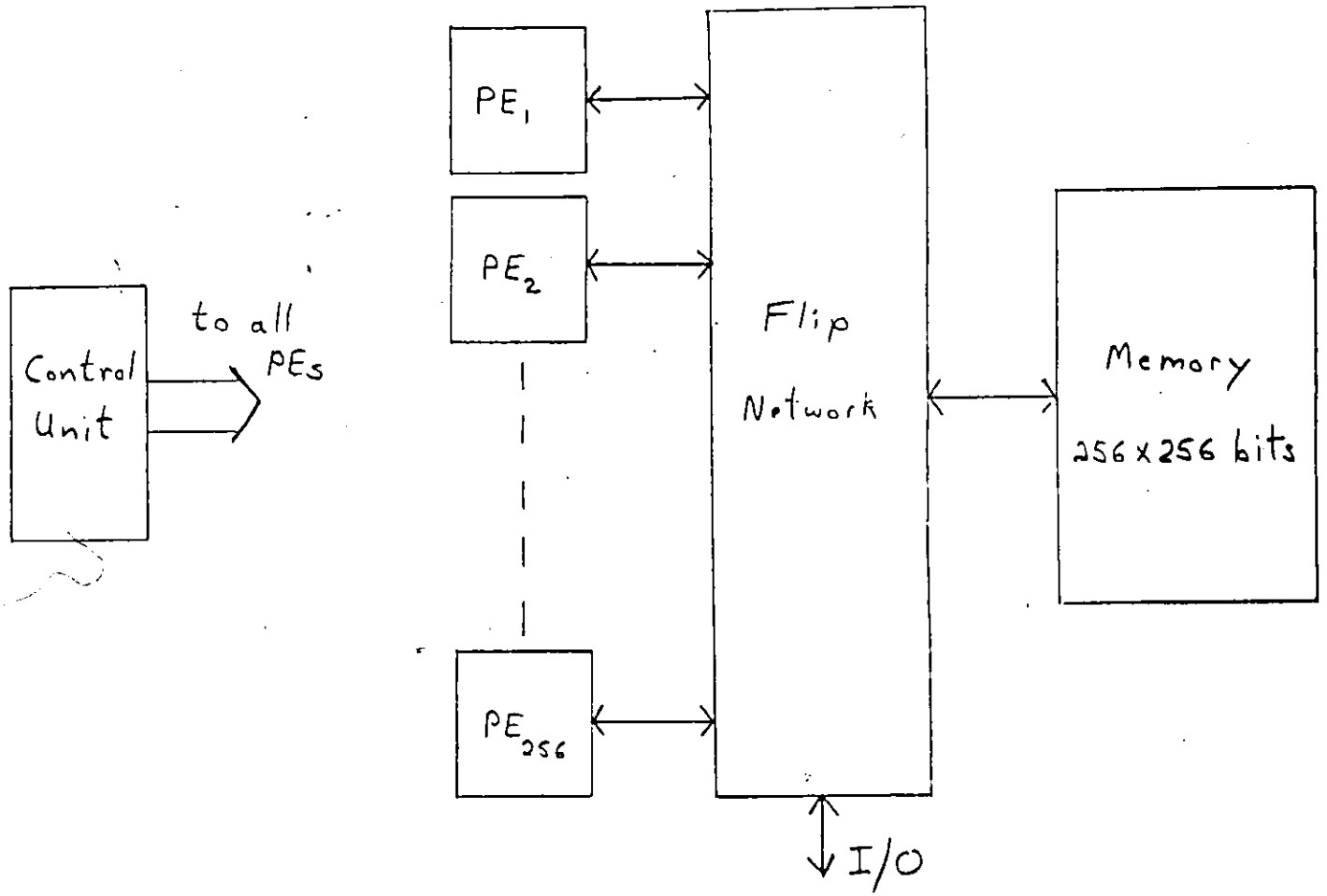


Fig. 4.2-The STARAN General Block Diagram

4.3.3 Remarks

The STARAN, like any other SIMD machine, is very useful when the same operations needs to be performed on a large number of data items. For example, a 250*ten-bit vector addition with a vector of the same length requires only 30 memory cycles, 20 cycles for bit-slice fetches of the two vectors, and 10 bit-slice stores of the result[11]. In addition, the STARAN system is not handicapped by serial operations, performing these at the same rate as contemporary serial computers.

Unfortunately, STARAN's arrays do not have any memory other than their memory dedicated to special processing, so part of the valuable memory has to be used as a scratch pad to store intermediate calculations. At NASA, the 256-bit words were organized into 20*8-bit words of data plus 96 bits of scratch memory [3], with the drawback that part of the scratch memory was wasted to keep information (line,column) about the data positions.

A drawback of STARAN lies in the fact that it lacks floating-point hardware for arithmetic operations. Multiplications and divisions have to be performed via software, wasting most of the computer time. Another drawback is the fact that STARAN does not support unsigned integer arithmetic so that for example, values from 0 to 256 have to be translated into levels from -128 to +127.

The concept of bit-slice and word memory accesses offers a lot of versatility in any case where data items are operated upon in two dimensions (horizontally,vertically). However, having to compute on a bit by bit basis leads to tedious programming. A byte format within 256-bytes(or 512) words plus scratch memory would provide a better working environment.

The STARAN is crippled by the same deficiency as the ILLIAC. Due to the central origin of the PEs control signals, when a branching point is encountered, only part of the PEs will be working, satisfying one particular condition, while the rest of the PEs are idled, waiting for their control signals.

4.4 THE PEPE SYSTEM

4.4.1 Introduction

The original work on PEPE (Parallel Element Processing Ensemble) was performed at Bell Telephone Laboratories in the early 1960's [12]. Its purpose was to help a host computer employed in a radar system by updating the data on all the objects the host was tracking. The Bell Lab's

design has an IBM360/65 as host and 16 processing elements(PEs). Each PE has an arithmetic unit, a correlation unit and a random access memory, all under the supervision of an arithmetic control unit and a correlation control unit. The PEPE is a multiple instruction multiple data streams machine.

In this section, the system of interest is the PEPE used at the Army Advanced Ballistic Missile Defense Agency Research Center [13]. This PEPE consists of 288 Emitter Coupled Logic PEs arranged in 8 bays of 36 under the control of a CDC7600 computer.

4.4.2 Hardware

To give a general idea of the complexity of PEPE, it is interesting to note that every bay needs 275 dual in-line packages(DIPs), consuming 30,000 watts; while the control console needs 150 DIPs and 6,000 watts. The whole unit is cooled by air as well as water.

Each PE is formed by 3 independent units: an arithmetic unit, an associative output unit, and a correlation unit. These units are independent, adding more parallelism to the system. Each unit can be activated or deactivated from the

current execution depending on the status of its activity register, following in this some design ideas of the SOLOMON computer.

The arithmetic unit in each PE is capable of integer, logical, or floating point operations using a conventionnal accumulator, an operand register, and a quotient register. The AU also has an activity stack with an activity register and a tag register to analyse data from the control unit. The associative output unit is very similar to the AU unit and so is the correlation unit which has 16 correlation registers in surplus for register to register correlation operations. Every PE has 1K of memory accessable on a priority basis by any one of these 3 units.

Every one of these 3 subunits of the PEs are controlled by a corresponding control unit (see fig. 4.3):

- -the arithmetic control unit
- -the associative output control unit
- -the correlation control unit

These control units are of common design in spite of differences in memory capacity. Together, they constitute what is called the control console. To these control units belong a data memory, a program memory, a sequential controller, a parallel controller and some intercommunication logic.

In this system, the PEs hardware blocks can be added or retrieved from the system without causing any software problems [14], since the PEs do not have direct permanent intercommunications links. This makes the system very reliable. A PE cannot execute an instruction by itself, its receives all its timing and control signals from the control console.

Each control unit has two input/output units (IOU) to transfer blocks of data back and forth between the CDC7600 and the control unit memory. This leads to almost no I/O overhead since the input and output operations are done simultaneously. This feature is particularly important since fast processors are usually very I/O bounded.

The CDC7600 does the sequential calculations, takes care of the peripheral devices, and compiles PEPE's programs. The PEPE's instructions words are 32 bits long.

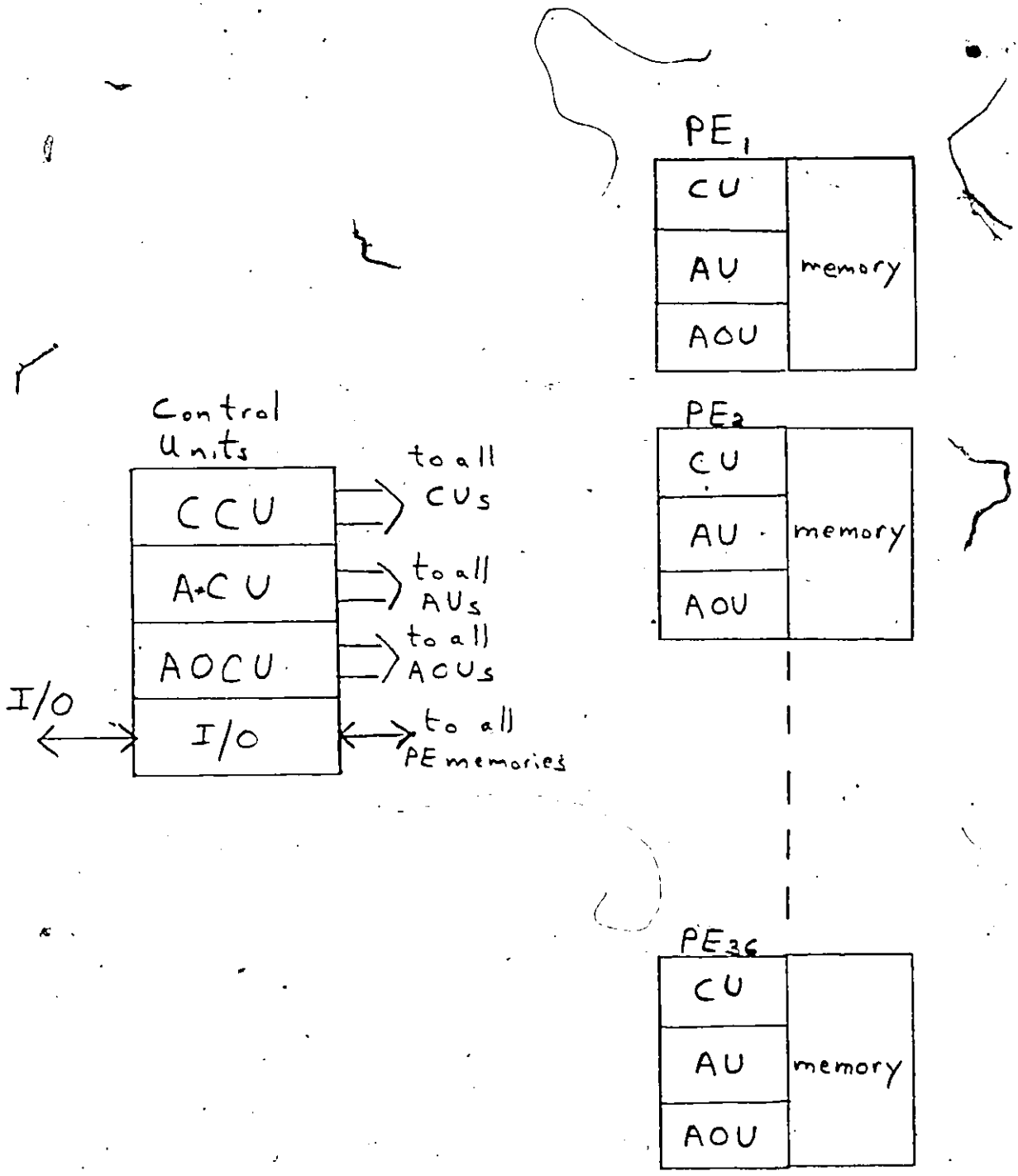


Fig. 4.3-The PEPE General Block Diagram

4.4.3 Remarks

Alexander and Parker [15] noticed a lack of fixed point multiply operation in the associative output unit and in the control unit. For image processing as well as weather modelling, the PEPE is not considered very good because of its lack of easy communication between PEs. This ease of communication was not introduced because it is not needed for radar processing and not desirable from the point of view of reliability (as is the system allows PE failure).

4.5 THE FLEXIBLE MULTI-PIPELINE-PROCESSOR SYSTEM (FMPP)

4.5.1 Introduction

The Flexible Multi-Pipeline-Processor is a design idea advanced by K. Vorgrimler [16] as a solution to slow picture processing by windowing techniques (1). For window operations, the calculations are constantly the same going through the whole image, thus leading easily to a pipeline implementation. Also, many of these calculations on a given window are only comparaisons or simple arithmetic functions

---

(1) windowing techniques are explained in chapter seven
(2) pixel -> picture element

between two pixels(2) and are usually repeated a number of times within the window. So the parallelism involved can be used to speed-up the processing. This led Vorgrimler to design a multiple instruction multiple data streams machine.

4.5.2 Hardware

The FMPP, as designed at the Forschungsinstitut fur Informationsverarbeitung und Mustererkennung(FIM) uses 16 individual processors(IPs) out of a possible maximun of 62. These IPs are serviced by two data busses (IBA,IBB), one instructions bus (IB) and one processor status bus(SB). Also, each IP has it own output bus(Bn), as shown in figure 4.4.

The instruction bus is used to transfer instructions to the IPs at initialization time. The instructions are then stored locally. They are four bytes long: the op-code byte, two operand-source addresses and one operand-destination address. Every IP has two input-control units(IUA,IUB) which are programmed by the operand-source addresses. These units then know to which output busses (B0,B1,B2,,B61) to hook-up in order to get their input data. The SB transmits status information about the processors. All busses are 8-bits wide and transfers only take place at

the request of the receiving processor. IBA and IBB are used to send the data from the outside world to the FMPP input processors(which can be any processor).

The input units of the individual processors (IUA,IUB) can be connected to any of the output busses of the other IPs or to IBA or IBB for external inputs. The input data of each side can be stored in an internal register(RSA,RSB) for further use or sent directly to the arithmetic logic unit(ALU) for processing. Similarly the output data of the ALU can be stored in an internal register for reusage or output on the output bus(Bn). Instructions come initially from IB, via the instruction input unit(IIU), to the instruction memory(IM) where they are stored. During processing these instructions are fetched and decoded by the decode and control unit(DECC) and the ALU can perform its task.

At FIM the FMPP is tested using a PDP11/45. To interface the FMPP to the unibus and to transfer the image pixels from main memory to the FMPP, three more units are needed. One input/output unit to send the instructions to the FMPP's processors at initialization time, to send input data, and to read status information while they are in operation. A second unit to get the output data from the special

processor and put them into main memory. A third programmable input buffer(PIB) to format the input data in a window fashion for more direct IBA and IBB accesses.

The programmable input buffer is formed of three consecutive 1024 pixels registers and transfers data to IBA and IBB at their request. IBA and IBB are programmed to know when to pick up the data. So data can be supplied to the PIB at a much slower rate then it would if it had to be supplied directly to the FMPP, which suits the PDP11 better. The PDP11 takes care of transferring blocks of picture data from the main memory to the disk where the images are usually stored.

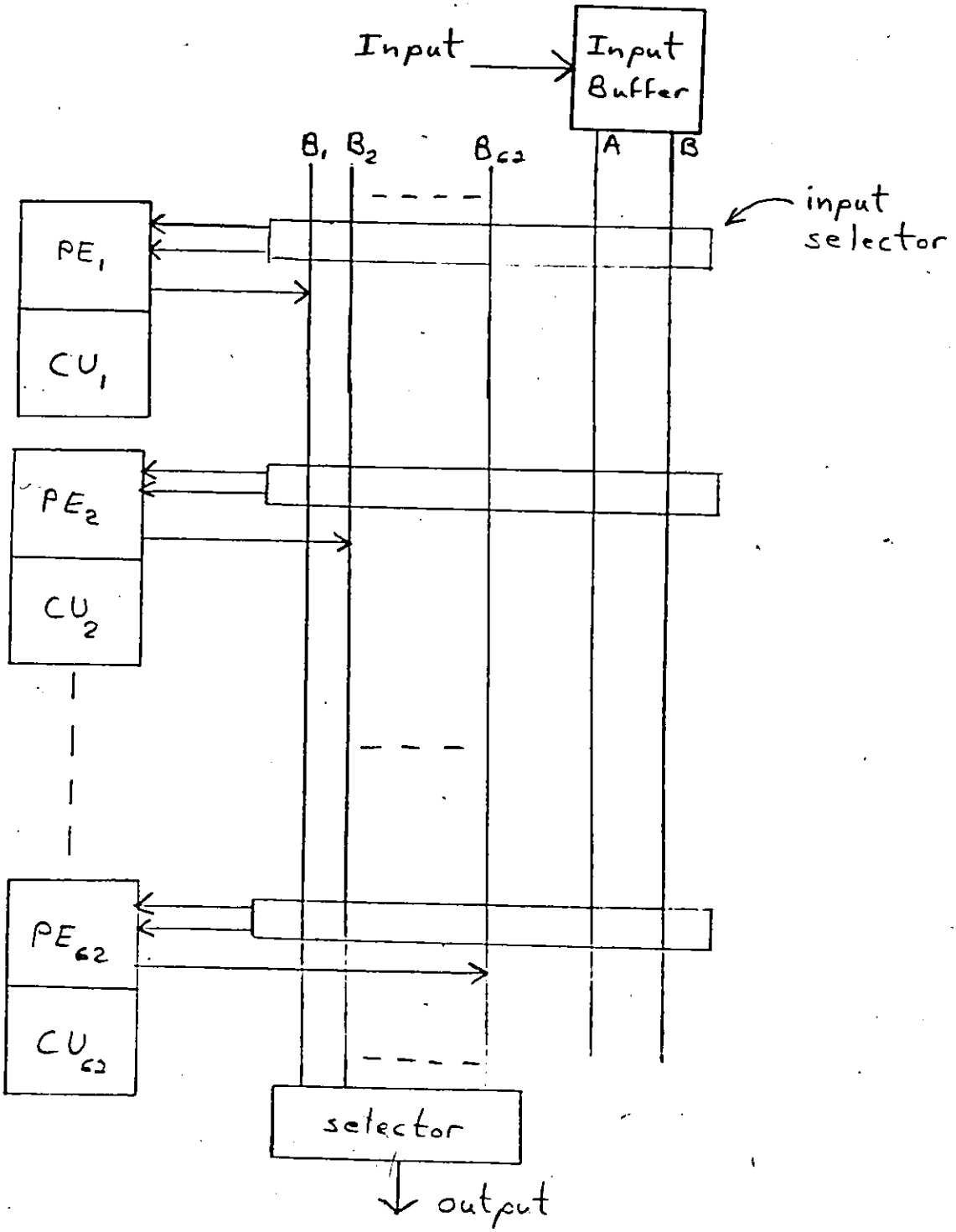


Fig. 4.4-The FMPP General Block Diagram

4.5.3 Remarks

Windowing image processing tasks on 1024*1024 pixels images can now be performed by this system in roughly 10 sec. The improvement, as reported by Vorgrimler, is of the order of 7 to 300 times faster than when the same algorithms are performed on a CDC3300(c.t.=1.25 microsec.).

The major drawback of this PMPP special system is the tremendously big number of bus lines used(greater than 500 for a complete 62 processors system) which is, from a hardware point of view, totally unacceptable.

The idea of combining parallelism and pipelining is surely an interesting way to try to obtain faster image processing. At first sight it looks like it is worth while to go further in the investigation of this method. A primary step would be to develop a more reasonnable communication system between the processors. Another important feature of this system is the software control capabilities for restructuring the system by rearranging all the possible communications path between the processing elements. This is a must for any system that is to be versatile.

4.6 CONCLUSION

This chapter illustrates some of the similarities and differences between four processor architectures. It shows that even within the two most interesting Flynn's categories of processors, SIMD and MIMD, major variations exist. Even the categories themselves are not so distinctive. It is also shown how similar problems can be solved differently by various groups of researchers. The table of figure 4.5 summerizes all the important characteristics of the four architectures.

Having analysed processor architecture in general and four specific systems, we are now in a good position to analyse the Multiprocessor Array Computing Structure (MACS). A simulator of MACS has been implemented and is used to test some image processing algorithms. This is going to be examined in the next three chapters.

ORGANIZATION		ILLIAC	STARAN	PEPE	FMPP
		SIMD	SIMD	MIMD	MIMD
Control Units	ALU	1	1	3	
	Instruction	32 bits	32 bits	32 bits	
	Registers	4 x 64 bits	12	yes	
	# of C.U.	1/array	1	3	ø
	Data Memory Inst.	yes	yes	2 to 3 K	
	Operand	64 bits	32 bits	2 to 32K	
	Other	Queues		-	
				CU does sequential inst.	
Processing Elements	Max. # of PEs	4 arrays of 64 PEs	32 arrays of 256 PEs	8 bays of 36 PEs	62 PEs
	Word length	64 bits	1 bit	-	8 bits
	Registers	7	3	34	3
	Data Memory Inst.	2K x 64 bits	256 bits	1K	none
	ALU	none	none	none	yes
	Instruction	1	1	3	1
	Special Hardware	none	none	none	32 bits
		Barrel switch	Flip Network	associative output unit correlation unit arithmetic unit	control unit and instruction decoding 3 buffers

Fig. 4.5 - Comparison of Four Architectures

	ILLIAC	STARAN	PEPE	FMPP
Communication between PEs	North, east, west and south PE	through Flip network	via control units	62 x 8 bit busses
Inupt/Output	16 edge PEs/array with a 1024 bit I/O bus 256 x 64 bit buffers	32 bit DMA bus 32 bit buffed I/O bus parallel I/O bus control/signal bus through PDP11	each control unit has 2 I/O units	1 I/O part for inst. data input and status output 1 I/O part data output 1 programmable input buffer 1024 x 8 bit
Data width	64 bits	256 bits	--	8 bits
Host	Burroughs B6500	any host plus/or a PDP11	CDC 7600 or IBM 360/65	PDP11/45
Mass Storage	Parallel access disk 16 K bit buffer	256 x 256 bits bit-slices or words access memory Disk	Data memory	Memory Disk
Comments	- good for vector and matrix operations - degradation with branching	- good for vector and matrix operations - not enough PE memory - no floating point op. - branching degradation	- almost no I/O overhead - comm. between PE not easy - restructurable	- too many busses - restructurable - parallelism and pipelining

Fig. 4.5 - Comparison of Four Architectures (cont.)

References

[1] Slotnick D.L., Borch W.C., Mc Reynolds R.C., "The SOLOMON Computer", AFIPS, Vol. 22, FJCC, 1962, p. 97

[2] Kuck D.J.,"ILLIAC IV Software and Application Programming", IEEE Trans. on Computer,vol. C-17, no. 8, August 1968, p758

[3] Barnes G.H., Brown R.M., Kato M., Kuck D.J., Slotnick D.L., Stokes R.A., "The ILLIAC Computer", IEEE Trans. on Comp., Vol C-17, No 8, August 1968, p746

[4] Enslow, P. H.Jr., Comtre Corporation, "Multiprocessors and Parallel Processing", John Wiley and Sons, New York, 1974

[5] Bailey F.R.,"Computational Aerodynamics-ILLIAC IV and Beyond", Compcon, San Francisco, Feb. 28-Mar. 3, 1977, IEEE Comp. Soc., p10

[6] Feldman J.D. and Reimann O.A., "RADCAP: An Operationnal Parallel Processing Facility", 1973 Sagamore Comp. Conf. on Parallel Processing, Sagamore Lake, August 22-24, 1973, IEEE Press, p. 140

[7] Batcher K.E., "STARAN/RADCAP Hardware Architecture", 1973 Sagamore Comp. Conf. on Parallel Processing, Sagamore Lake, August 22-24, 1973, IEEE Press, p. 147

[8] Rohrbacker D. and Potter J.L.,"Image Processing with the STARAN Parallel Computer", Computer, Vol. 10, No.8, IEEE Press, August 1977, p54

[9] Rubben S.,Faiss R.,Lyon J.,Quinn M.,"Application of a Parallel Processing Computer in LACIE", Proc. of 1976 Int. Conf. on Parallel Processing, August 24-27, 1976, IEEE Press, USA, p. 24

[10] Batcher K.E.,"The Flip Network in STARAN", Proc. of 1976 Int. Conf. on Parallel Processing, IEEE Press, USA, p65

[11] Batcher K.E.,"The Multi-Dimensional Access Memory in STARAN", IEEE Trans. on Comp., Vol. C-28, Feb. 1977, p174

[12] Crane B.A., Gilmartin M.J., Huttenhoff J.H., Rux P.T., Shively R.R., "PEPE Computer Architecture", Compcon 72, 6th Annual IEEE Comp. Soc. Int. Conf., San Francisco, Sep. 12-14, IEEE Press, 1972, p. 57

[13] Evensen A.J., Troy J.L., "Introduction to the Architecture of a 288-Elements PEPE", 1973 Sagamore Comp. Conf. on Parallel Processing, Sagamore Lake, August 22-24, 1973, IEEE Press, p. 162

[14] Diggeldine J.R., Martin H.G., Patterson W.M., "Operating System and Support Software for PEPE", 1973 Sagamore Comp. Conf. on Parallel Processing, Sagamore Lake, August 22-24, 1973, IEEE Press, p. 170

[15] Alexander P.T. and Parker R.O., "A Comparaison of a Parallel and Serial Implementation of a Large Real Time Problem", 1973 Sagamore Comp. Conf. on Parallel Processing, Sagamore Lake, August 22-24, 1973, IEEE Press, p. 180

[16] Vorgrimler K.,"Enhancement of Computing Power in Multiprocessor Systems for Processing of Digitized Pictures",Proc. 1976 Int. Conf. on Para. Proc. , IEEE Press, USA, August 24-27, 1976, p. 11

Chapter V

THE MULTIMICROPROCESSOR ARRAY COMPUTING STRUCTURE

5.1 INTRODUCTION

The Multimicroprocessor Array Computing Structure (MACS) is a dedicated special processor to be used as a digital system for radar processing. MACS was originally designed by C.V.W. Armstrong for the Communication Research Center [1]. It enhances the evaluation of radar signals by preprocessing the raw radar data. It filters out the noise, keeps track of the planes, and can even warn the operator of possible collisions.

Conventional systems for performing similar functions are usually made of discrete medium scale integration components. They have only few of the capabilities of the MACS system and their cost is much higher. The algorithmic processing of data was chosen because the system has to be flexible enough to perform different functions using the same hardware configuration. Also, it has to be capable of modifying these functions or their parameters in real time

in order to adapt the system to changes in the environment. Furthermore, the data rate is to be of the order of one microsecond per data item, beyond the reach of sequential computers of comparative cost, justifying the need for a special processor.

5.2 GENERAL ARCHITECTURE [1] [2]

The MACS is a multiple instruction multiple data streams system. It basically consists of a column of 32 processing elements (PEs), each made up of three AM2901 bit-sliced microprocessors. The column functions as a pipeline, but a pipeline where the data can go up as well as down. Every PE can communicate directly with its immediate neighbours, above or below, without having to go through the system bus. The system bus is only used for communication over longer distances (see fig. 5.1). Every PE has it own microprogram memory.

Also noticeable in fig 5.1 are the top and bottom buffers which are the input/output ports of the MACS. These programmable buffers receive and transmit data between the host computer (or in a real application the radar system) and the PE column. Their program informs them at which clock cycle the next data will be needed by the closest PE

or by the system bus, or when to fetch the data from these two sources.

Every processing element has 17 internal registers and 16 external registers. If more memory space is needed, for example, to implement lookup tables, every processing element has access to the working memory via the working memory bus. The working memory is assumed to be programmable and thus "intelligent" enough to know what and when to send to the PE. It is also assumed to be capable of picking-up data from the bus and use it as a pointer to locate the next data to be put on the bus.

The MACS operates using a clock cycle between 60 and 100 nanoseconds. The data are represented by 12-bit numbers and the busses are 12-bit wide.

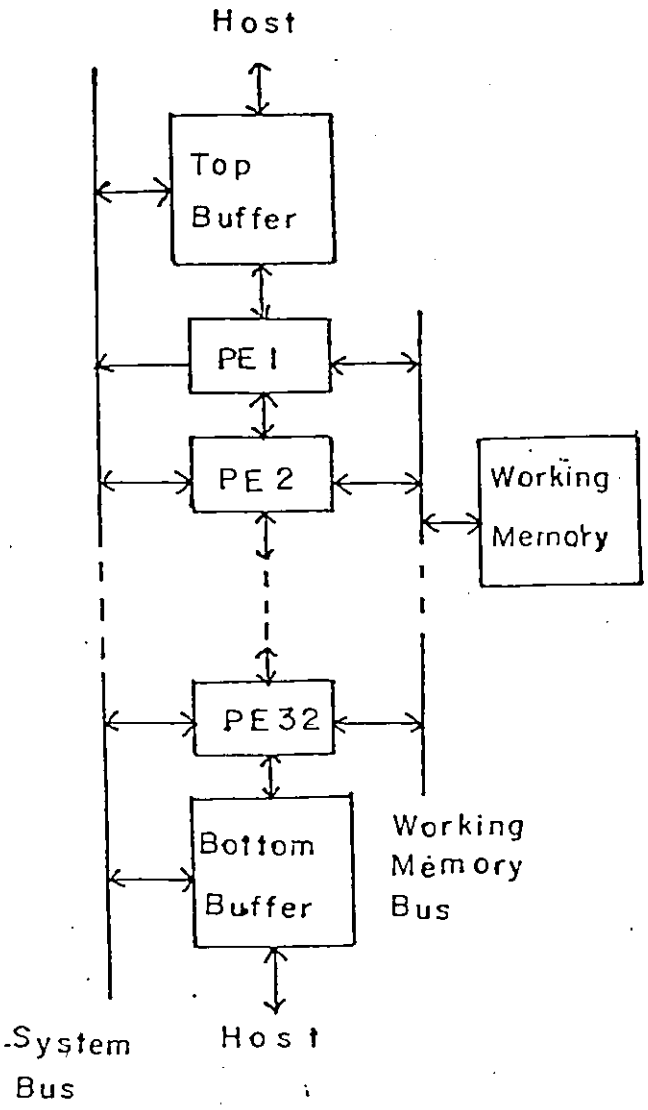


Fig. 5.1-The MACS Architecture

5.3 BIT-SLICED MICROPROCESSORS [3] [4] [5]

5.3.1 Introduction

Microprocessors allow much more flexibility in the design of digital systems than anything previously available. These Metal-Oxide-Semiconductor processors are well suited for a vast number of applications. They are used as hardware for systems ranging from simple programmable instruments to complete multiprocessor systems, including the hobbyist small computing center. However, for some applications, they are just not fast enough. This is where the bit-sliced microprocessor comes in.

5.3.2 Hardware

The bit-sliced microprocessor is usualy manufactured with bipolar technology. It is also fabricated using much simpler basic elements than the general microprocessor. These two features provide the bit-sliced microprocessor with an improvement in performance by a factor of ten over conventional microprocessors [3]. However, the second feature also implies that programming has to be done at a much lower level, making it more flexible, but demanding

more of the programmer's time. A bit-sliced microprocessor usually needs several supporting chips, for instance: a microprogram sequencer, some external general pupose registers, and if necessary a look-ahead carry generator. This is due to the lower packing density of bipolar technology. Several bit-sliced microprocessors are usually concatinated to provide a respectable word size.

To program, hence fully understand, a bit-sliced microprocessor, a good background in basic computer architecture and design is required. It is to be realized that the hardware is directly controlled by the microprograms. It is therefore impossible to program such processor while ignoring the architecture of the device or its internal timing of events. A basic bit-sliced microcomputer is shown in fig. 5.2 .

In this figure some bit-sliced microprocessors are shown as the heart (CPU) of a general purpose sequential machine. This is a very common practice. This system could simulate any ordinary sequential processor with proper microprogramming. In the processing section, as many bit-sliced microprocessor chips as needed are used to obtain the desired word size. They receive their control signals from the control section and return their status to it. In concert, they perform operations on the data sent to them.

The control section contains the microprogram memory and a microprogram sequencer which keeps track of the instructions to be sent to the processing section.

5.3.3 Microprogram Instructions

Microprogram instructions are generally very long and very complex. They are decomposed into several different fields. The op-code field, which directly drives the processing section, is usualy sent to the ALU via a register or a pipeline of registers in order to save fetching time.

Microprogram instructions also have a sequencer control field that serves several purposes. The sequencer control field is fedback to the microprogram sequencer to indicate the next program address. This address can be modified or changed by the selection logic. The selection logic receives it informations from the processing section(e.g.: for conditionnal branching) or the outside world. This option does not exist with the MACS system because the data rate involved and the necessary synchronisation between the processing elements makes conditional branching almost impossible. The only conditionnal branching instruction available is a form of SKIP if negative instruction.

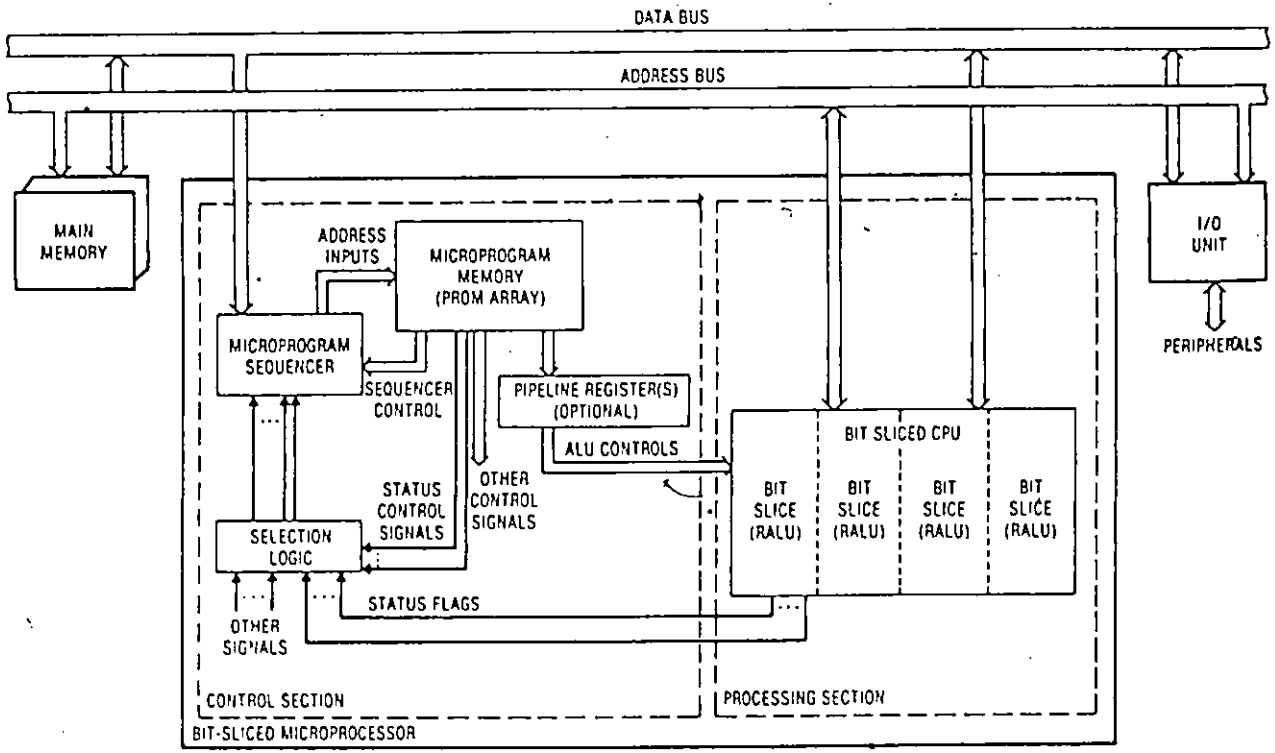


Fig. 5.2-A Microprogrammable Bit-Sliced Microcomputer [3]

5.4 ARCHITECTURE OF THE 2901 [6]

The AM2901-A bit-sliced microprocessor is the heart of the processing elements of MACS. Most of MACS characteristics and limitations are due directly to the 2901. Therefore it is important to fully understand this microprocessor before proceeding any further in the study the MACS system.

As seen in fig. 5.3 the three basic blocks forming a 2901 are: the arithmetic-logic unit (ALU), the Q register, and a 16 by 4-bit 2-port RAM. The RAM can be considered, and will be sometimes referred to, as 16 internal registers. The addresses of these internal registers are provided externally. Two registers might be accessed at the same time and their content used as input to the ALU. Also, the result of the ALU operation can be stored in the register pointed by the B address within the same microinstruction cycle.

Note that there is no dedicated register allocated to the ALU. However, it can take its inputs, via multiplexers, from any internal register (including Q) or from the D input, the outside world. It can also assume a null operand (zero). This is easily seen from the 2901 block diagram and the operands are referred to "ALU source" in the opcode fields shown in fig. 5.4.

Still following the block diagram, it can be seen that the result generated by of the ALU can be routed to three different places: the Q register, the internal registers (as specified by the B address), and an output port (Y). Looking carefully one might also notice that the input multiplexers for the Q register and the RAM can be used as data shifters. Naturally in order that significant shifts be possible, the least significant bits (LSB) and the most significant bits (MSB) must be available to the neighboring 2901's. This is illustrated by the "ALU destination control" fields table in fig. 5.4.

The most important item which yields some insight on the capabilities of the 2901, is the "ALU function" field. The ALU is capable of performing 8 basic operations. This is a set of the most typical basic computer instructions, and needless to say, a serious appreciation of these functions must be made before attempting any microprogramming on the MACS system.

In the case of MACS, three AM2901-A are used to form a processing unit capable of handling 12-bit data. Other features of interest in evaluating the computational power of this processing unit are due to some of the external circuitry:

-two AM25LS253 multiplexers

-some three states gates

The two AM25LS253 are used as illustrated in fig. 5.5 to introduce more practicality in the shifting operations. By taking care of the LSB and the MSB out of the processing unit, these ICs allow forcing in of zeros and ones while shifting, or even rotations in both directions. It also allows double precision shifting using the Q register in conjunction with an internal register.

The additional three-states gates are used to simplify the multiplication process, as shown in fig. 5.4. This is needed since the 2901s do not have any testing capabilities of their own. The addition of these gates allow a conventional multiplication algorithm to be implemented in the 2901 (as described in [6]). However, since this process usually takes more than a microsecond, a hardware multiplier was added to the design for radar processing (as explained in the next section).

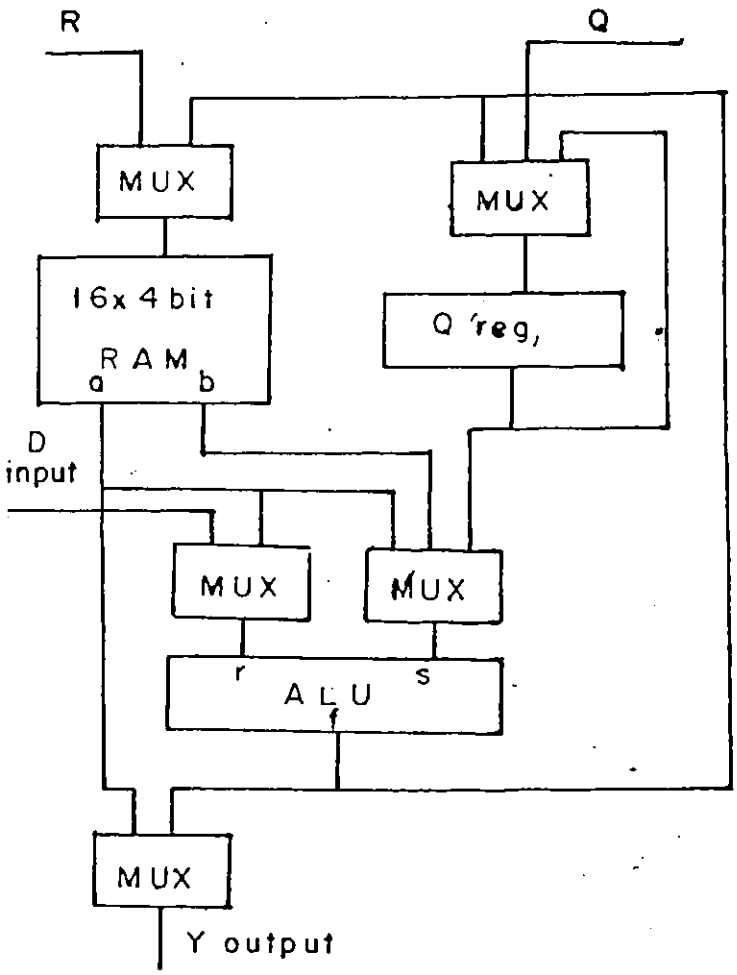


Fig. 5.3-The 2901 Block Diagram

MICRO CODE				ALU SOURCE OPERANDS	
Pin 14 I_2	Pin 13 I_1	Pin 12 I_0	Octal Code	R	S
L	L	L	0	A	Q
L	L	H	1	A	B
L	H	L	2	0	Q
L	H	H	3	0	B
H	L	L	4	0	A
H	L	H	5	D	A
H	H	L	6	D	Q
H	H	H	7	D	0

ALU Source Operand Control.

MICRO CODE				ALU Function	Symbol
Pin 27 I_5	Pin 28 I_4	Pin 29 I_3	Octal Code		
L	L	L	0	R Plus S	R+S
L	L	H	1	S Minus R	S−R
L	H	L	2	R Minus S	R−S
L	H	H	3	R OR S	R∨S
H	L	L	4	R AND S	R∧S
H	L	H	5	$\bar{R}$ AND S	$\bar{R}$∧S
H	H	L	6	R EX-OR S	R∨S
H	H	H	7	R EX-NOR S	$\overline{R \vee S}$

ALU Function Control.

MICRO CODE				RAM FUNCTION		Q-REG. FUNCTION		Y OUTPUT	RAM SHIFTER		Q SHIFTER	
Pin 9 I_8	Pin 7 I_7	Pin 6 I_6	Octal Code	Shift	Load	Shift	Load		RAM_0	RAM_3	Q_0	Q_3
L	L	L	0	X	NONE	NONE	F→Q	F	X	X	X	X
L	L	H	1	X	NONE	X	NONE	F	X	X	X	X
L	H	L	2	NONE	F→B	X	NONE	A	X	X	X	X
L	H	H	3	NONE	F→B	X	NONE	F	X	X	X	X
H	L	L	4	DOWN	F/2→B	DOWN	Q/2→Q	F	F_0	IN_3	Q_0	IN_3
H	L	H	5	DOWN	F/2→B	X	NONE	F	F_0	IN_3	Q_0	X
H	H	L	6	UP	2F→B	UP	2Q→Q	F	IN_0	F_3	IN_0	Q_3
H	H	H	7	UP	2F→B	X	NONE	F	IN_0	F_3	X	Q_3

X = Don't care. Electrically, the shift pin is a TTL input internally connected to a three-state output which is in the high-impedance state.
B = Register Addressed by B inputs.
Up is toward MSB, Down is toward LSB.

ALU Destination Control.

I_{210} OCTAL		0	1	2	3	4	5	6	7
OCTAL I_{543}	ALU Source / ALU Function	A, Q	A, B	0, Q	0, B	0, A	D, A	D, Q	D, 0
0	C_n = L, R Plus S	A+Q	A+B	Q	B	A	D+A	D+Q	D
	C_n = H	A+Q+1	A+B+1	Q+1	B+1	A+1	D+A+1	D+Q+1	D+1
1	C_n = L, S Minus R	Q−A−1	B−A−1	Q−1	B−1	A−1	A−D−1	Q−D−1	−D−1
	C_n = H	Q−A	B−A	Q	B	A	A−D	Q−D	−D
2	C_n = L, R Minus S	A−Q−1	A−B−1	−Q−1	−B−1	−A−1	D−A−1	D−Q−1	D−1
	C_n = H	A−Q	A−B	−Q	−B	−A	D−A	D−Q	D
3	R OR S	A∨Q	A∨B	Q	B	A	D∨A	D∨Q	D
4	R AND S	A∧Q	A∧B	0	0	0	D∧A	D∧Q	0
5	$\bar{R}$ AND S	$\bar{A}$∧Q	$\bar{A}$∧B	0	B	A	$\bar{D}$∧A	$\bar{D}$∧Q	0
6	R EX-OR S	A∨Q	A∨B	Q	B	A	D∨A	D∨Q	D
7	R EX-NOR S	$\overline{A \vee Q}$	$\overline{A \vee B}$	$\bar{Q}$	$\bar{B}$	$\bar{A}$	$\overline{D \vee A}$	$\overline{D \vee Q}$	$\bar{D}$

+ = Plus, − = Minus, ∨ = OR, ∧ = AND, ∨ = EX-OR

Source Operand and ALU Function Matrix.

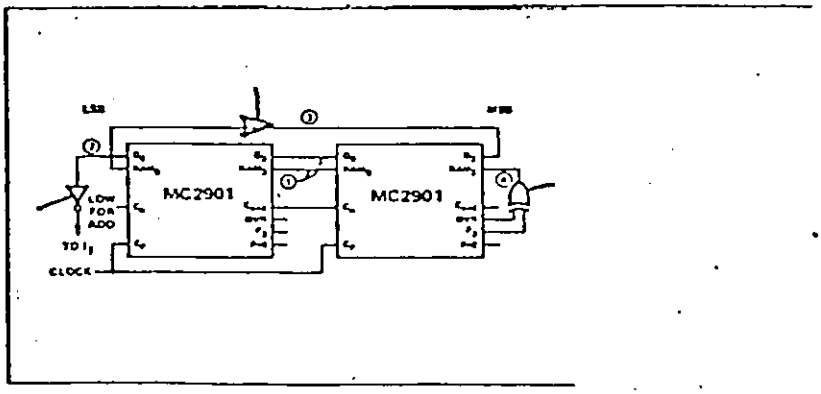
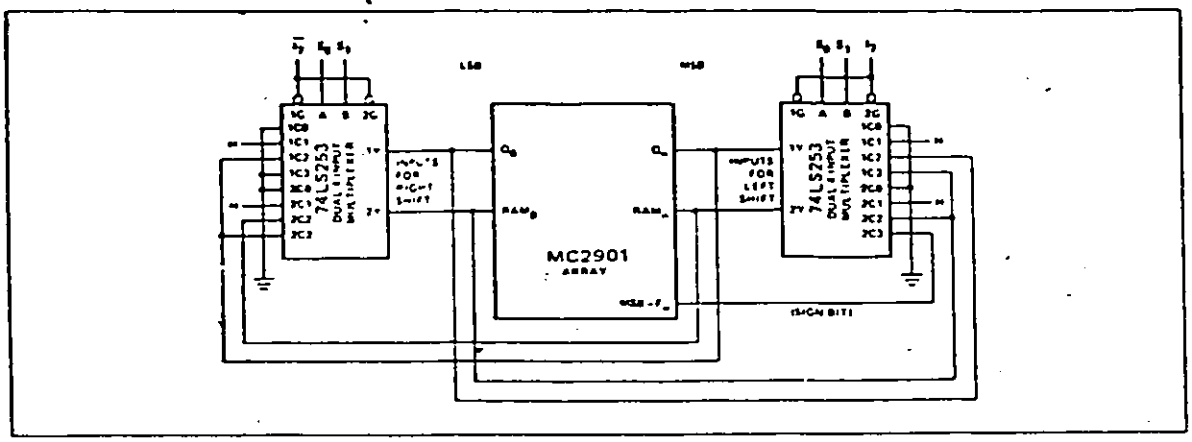


Fig. 5.4–Opcode Fields and Multiplication [6]



Code			Source of New Data				Shift	Type
I_7	S_1	S_0	Q_0	Q_n	RAM_0	RAM_n		
H	L	L	0	Q_{n-1}	0	F_{n-1}	Up (Right)	Zero
H	L	H	1	Q_{n-1}	1	F_{n-1}		One
H	H	L	Q_n	Q_{n-1}	F_n	F_{n-1}		Rotate
H	H	H	0	Q_{n-1}	Q_n	F_{n-1}		Arithmetic
L	L	L	Q_1	0	F_1	0	Down (Left)	Zero
L	L	H	Q_1	1	F_1	1		One
L	H	L	Q_1	Q_0	F_1	F_0		Rotate
L	H	H	Q_1	F_0	F_1	$RAM_n = RAM_{n-1} = F_n$		Arithmetic

Fig 5,5 Shifting Hardware [G]

5.5 ARCHITECTURE OF THE PROCESSING ELEMENTS [1][2][7]

The building block of the MACS is the processing element. The most important part of the processing element is naturally the block formed by the three AM2901 bit-sliced microprocessors. This 12-bit processing unit has an arithmetic logic unit, a Q register, and 16 internal registers as described in the previous section. It receives the control signals from the opcode RAM and the addresses of the internal registers from the address RAM. The data is received from the input multiplexer and the output result is directed to a latch, as shown in fig. 5.6.

The opcode RAM contains the microinstructions. It is controlled by the clock and the memory address register (MARO). MARO points to the next microinstruction of the opcode RAM to be executed. Each microinstruction has a field (NOCRA) containning the address of the next microinstruction. This address is loaded into MARO at every clock cycle and used in the next clock cycle to address the next microinstruction. This scheme, as implemented, preclude any conditional jump. This was felt necessary because of the timing constraints of the radar processing application.

The address RAM has two memory address registers (MARA1 and MARA2). Both contain an address pointing to the next

possible set of register addresses. MARA1 gets its address from the opcode RAM, while MARA2 gets its from the address RAM. The selection of MARA1 or MARA2 as next address is made through a multiplexer (MUX) controlled by some microinstruction bits. This scheme was implemented in order to have more flexibility in addressing the internal registers. It allows certain microinstructions using MARA1 to have internal register permanently assigned to them, while some other microinstructions using MARA2 can change cycliquely their associated internal registers.

Both the address RAM and the opcode RAM are only sixteen microinstructions deep. This limitation was introduced in MACS design because of the fact that for radar processing operations, no more than sixteen cycles could be used. Indeed, for a minimun clock cycle of 60 nanoseconds only sixteen cycles are possible in one microsecond.

The processing unit also has access to 16 external registers located in the external RAM. The external memory is similar to the internal memory described in the previous section. It is a dual ports memory. Its addresses are provided by two microinstruction fields (AADDE and BADDE). From the processing unit point of view, the data from memory is obtained indirectly via the input multiplexer. This is done using the A address, fast enough to be used in the same

microinstruction cycle. The writing of data to memory is done during the next cycle, using the B address of that cycle. The B address can also be used to obtain data for the output multiplexer and the hardware multiplier.

Also available as part of the processing element is a hardware multiplier. The multiplier takes its inputs from the output latch of the processing unit and from a latch at the external RAM output. It takes at least two clock cycles (175-200 nsec.) for a multiplier like the TRW MPY-12AJ to produce the multiplication results. Therefore, the more significant part of the result will be available at the input multiplexer on the third cycle. The less significant part of the result is delayed in order to be available at the fourth clock cycle, since the processing unit can only input one datum at the time.

To communicate with the rest of the system, the processing element uses an input and an output multiplexer. The input multiplexer selects between inputs either from the PE above, the PE below, the external RAM, the system bus, both halves of the hardware multiplier's results or the memory bus. The input selected is available to the processing unit in the current clock cycle. The output multiplexer outputs either from the external RAM latch or from the processing unit, to either the system bus or the

memory bus. The output is available at the end of the current clock cycle and also at the beginning of the next one. Inter-PE communication is established when one PE loads the system bus or its inter-PE bus with a data item. The loading of the bus is done at the end of a clock cycle whent the corresponding output latch receives the datum and is switched from high impedance mode to an output mode. This output mode is maintained through the initial nanoseconds of the next clock cycle, allowing another PE (or more) to pickup the datum from the bus. The same method is used by the top and bottom buffers and the programmable working memory.

This communication scheme relies on software control to prevent two PEs from outputing on the same bus at the same time. It implies an additional burden for the programmer. Fortunatly, the MACS simulator (chapter six) is capable of detecting such mistakes.

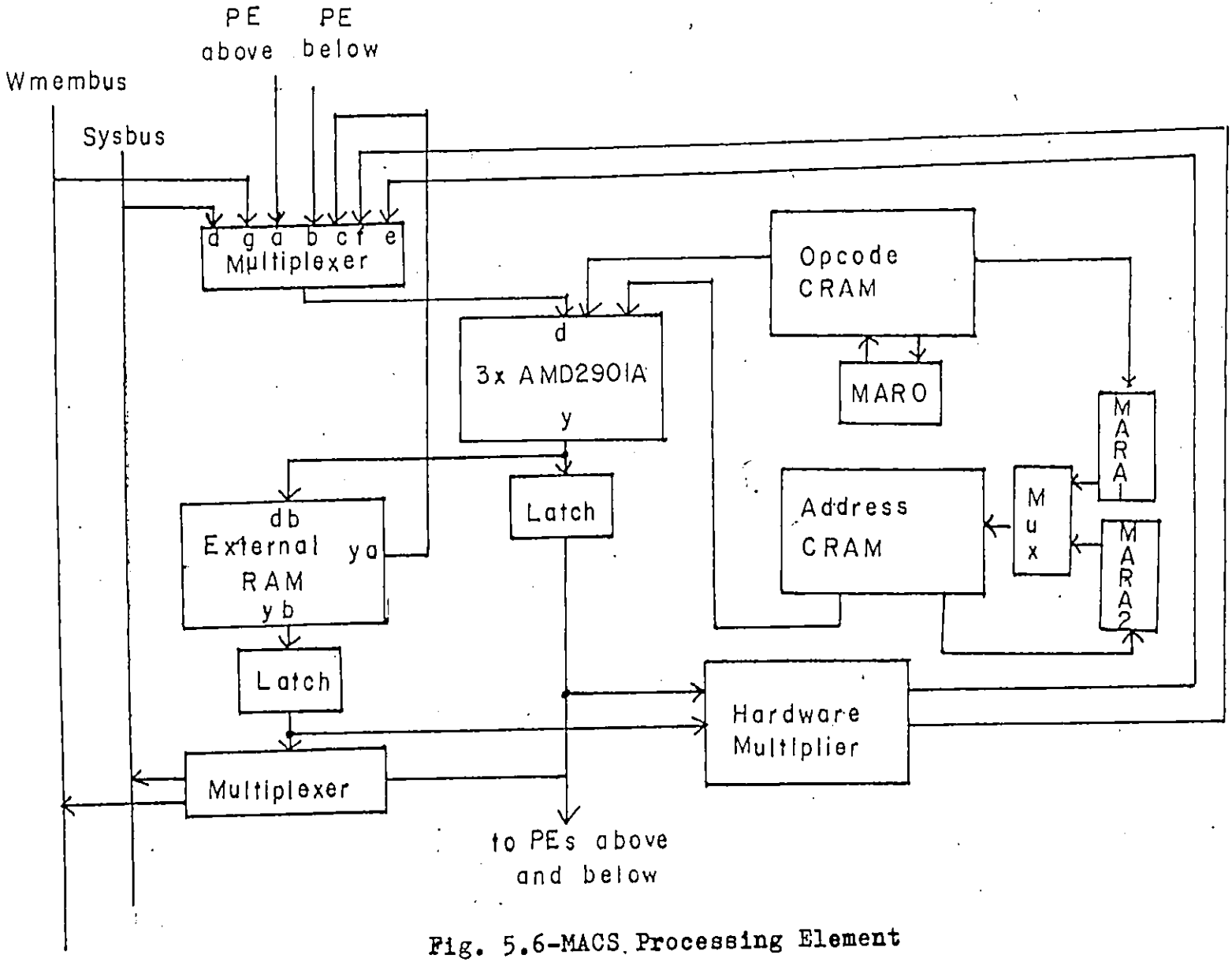


Fig. 5.6-MACS Processing Element

5.6 THE MICROINSTRUCTION FIELDS [2][7][8]

After having examined all the hardware constituents of the MACS system, let us now study the construction of the microinstructions which are to be executed by the processing elements of MACS. The microinstruction fields are tabulated in fig. 5.7. This figure, with 5.4 and 5.6, should always be at hand while creating microprograms.

Let us examine the fields one by one:

-->the opcode field <OPCODE>

The opcode field was explained in detail in section 5.4. Note that the subfields composing this field are in the following order: source, function, and destination. This arrangement is reverse to the one used by the 2901, but is of a more logical approach from a programmer point of view.

-->the carry field <CARRY>

The carry field is used to force a carry into the processing unit. It is mainly used with the substraction <1-->carry-->no borrow>.

-->the disable field <DISPE>

A "one" in the disable field puts the processing element in a disable mode at the end of the clock cycle if the result from the operation is negative. However, a "one" in the enable field of any of the following clock cycles would override this disable mode and terminate it.

-->the enable field <ENPE>

A "one" in that field enables the processing element to perform the operations as specified by the microinstruction, by putting it into enable mode at the beginning of the clock cycle. A "zero" leaves it in its previous mode.

-->the bus transmitter select field <BXMS>

This field determines which data is to be output to the busses if an output operation is done.

-->the bus transmitter enable field <BXME>

This field is a 2-bit field enabling or disabling transmission of data to the busses, and selecting which of the busses is to receive the output (see fig. 5.7).

-->the input multiplexer select field <IMIS>

This is a 3-bit field selecting the data to be used as input to the processing unit. Two of the inputs are the hardware multiplier outputs: the more significant half and the less significant half.

-->the next opcode RAM address field <NOCRA>

This field is used to point to the next microinstruction to be executed.

-->the next random address RAM address <NACRA>

This field points to the addresses to be used by the processing unit for the selection of its internal registers. This field may or may not be used depending on the <ACRMC> and the <LMUXC> fields described later.

-->the external RAM A address field <AADDE>

This field is used to designate the external register in external RAM to be read as input data to the processing unit, providing the input multiplexer controlled by <IMIS> allows it.

-->the external RAM B address field <BADDE>

This field designates the external register to be read or written. Data from a read or write operation is available at the output multiplexer. Data to be written into a register is coming from the processing unit. In either case, the data is always available as an input to the hardware multiplier.

-->the address CRAM control field <ACRMC>

This field enables either or both MARA1 or MARA2 to be loaded with respectivly the <NACRA> field or the <NACRAI> field.

-->the latch multiplexer control field <LMUXC>

This field enables either the data contained in MARA1 or in MARA2 to be passed on to the address RAM in order to designate the next set of addresses to be used in the selection of the internal registers.

-->the external RAM write enable field <ERWE>

A "zero" in this field enables the writting of data into the external register as pointed by the B external RAM address.

-->the shift control field <SHIFTC>

This field controls the two AM25LS253 as mentioned in section 5.4.

-->the multiplication field <HMUL>

A "one" in this field enables the three-states gates that are used to facilitate software multiplication as described in the M2900 manual, pages 15 and 16 [2].

-->the A address for internal registers <AADDI>

This field resides in the address RAM. It points to an internal register designated as A by the processing unit.

-->the B address for internal registers <BADDI>

This field also resides in the address RAM and points to an internal register known as B by the processing unit.

-->the next sequential address RAM address <NACRAI>

This field is also contained in the address RAM. It points to one of the next addresses that can be used by the processing unit to select its internal registers, depending on the <ACRMC> and the <LMUXC> fields.

Fields	# of bits	Description
OPCODE	9	ALU source, function and detination
CARRY	1	Carry into ALU
DISPE	1	Disable PE at end of clock cycle if MSB of ALU negative
ENPE	1	Enable PE at beginning of clock cycle
BXMS	1	Bus transmitter select 0->ext.mem.data B 1->current PE output
BXMT	2	Bus transmitter enable 0->both busses 1->working mem. bus 2->system bus 3->no transmission
IMIS	3	Input multiplexer select 0->PE above 1->PE below 2->current PE 3->sysbus input 4->hard. mult. H 5->hard. mult. L 6->working mem. input
NOCRA	4	Next opcode RAM address
NACRA	4	Next random address RAM address
AADDR	4	External RAM A address
BADDR	4	External RAM B address
ACRMC	2	Address CRAM MARA control 0->both latches 1->random latch (MARA1) 2->sequential l. (MARA2) 3->disable
LMUXC	1	Latch MUX control 0->ramdom latch 1->sequential latch
ERWE	1	External RAM write enable 0->write
SHIFTC	2	Shift control 0->zero 1->one 2->rotate 3->arithmetic
HMUL	1	Hybrid multiplier enable
AADDI	4	Internal A address
BADDI	4	Internal B address
NACRAI	4	Next sequential address RAM address

Figure 5.7-Microinstruction Fields

	No. of PEs	Latency (microsec.)	Data Rate (nsec.)
Running Mean	3	7	300
Two-Pole Filter	4	.825	600
Threshold Detection	12	.975	300
Mean Deviate	33	19	750
Rank Detector	9	13	750
Mean Detector	5	.600	450

Fig. 5.8-Radar Processing Operations [1]

5.7 OPERATION IN RADAR PROCESSING

The MACS system has been programmed for various radar processing functions. In [1], MACS has been shown to be theoretically capable of handling basic and often used operations (see fig. 5.8). In that paper, the algorithms for these operations are explained and compared with the same operations implemented in hardware. It is shown how semi-parallelism between some processing elements is acheived within some pipeline stages. The performance in the examples of fig. 5.8 was evaluated using a 75 nanoseconds cycle time. It should be pointed out that a lengthy multiplication procedure (software multiplication) is used in these implementations., the hardware multiplier having been integrated in the system at a later time.

More recent papers [9] [10], show how MACS is used for more complicated radar processing applications. For example, the plot-to-track correlation implementation is done in two passes using a maximun of 5 processing elements: one pass where gate association is obtained using only one PE, and a second pass where the maximun likelihood function is performed using 5 PEs. The later involves quite difficult calculations for a simple microprocessor like the 2901. Few multiplications are done using the hardware multiplier and, logarithmic and exponential results are obtained using the working memory as lookup tables.

All those implementations have been able to yield a data rate less than one microsecond.

5.8 CONCLUSION

The Multimicroprocessor Array Computing Structure is a new multiple instruction multiple data streams machine. The architecture is constructed of 32 independent processing elements based on the AM2901A bit-sliced microprocessor. Each PE can communicate directly with the PE above or the PE below, and indirectly, with any other PE.

MACS is a versatile pipeline processor where every stage can be organized at will: the stages can make use of more than one PE and therefore some degree of parallelism can be implemented in every stages. MACS can also be used as if it was formed of a number of parallel pipelines.

A serious drawback is the fact that the MACS processing elements are not capable of branching in the conventional sense (conditional jump changing MARO). This capability was not implemented because in the original radar processing application each processing element must terminate its basic operation in only sixteen microcycles. The only conditional processing available with MACS derives from the DISPE

(disable PE) field. This can be treated by the programmer as a conditional skip of a number of microinstructions, allowing conditional execution of parts of the microprogram. Using the same principle the programmer can disable a complete processing element from performing a specific task, provided that the test is done in the first microinstruction. This crude capability makes conditional execution very hard to handle, but future model can easily be modified.

Another drawback is the fact that the programmer is responsible for keeping track of the bus utilization, preventing more than one PE from loading the same bus at the same time (as explained in section 5.5). This makes programming difficult, as PEs might have to wait few clock cycles for the busses to be available. It also makes the detection of such mistake nearly impossible at first sight, unless spevial circuitry is used or the microprograms are developed on a simulator such as the one described in chapter six.

Over the past year, MACS has certainly demonstrated its capability in handling radar data. The question remains: is MACS suitable for image processing. As a parallel processor, it lacks direct communication between all the PEs. Since most image processing tasks are conducted in two

dimensions, this can be a drawback. As a pipeline processor, which seems to be a trend in image processing nowadays, it lacks the calculating power. In spite of these facts, much can be learned from this particular organization, since it has some of the advantages of parallel systems and some of the pipeline processors. MACS seems to be a worthwhile organization to investigate for applications in image processing.

References

[1] Armstrong C.V.W., "A Multimicroprocessor System for Radar Signal Processing", I.P.Sharp Associates Ltd. Internal report, IPS-CRC-001

[2] Armstrong C.V.W., Brans N.A., Ahmed H.M., Fathi E., "An Adaptive Multimicroprocessor Array Computing Structure for Radar Signal Processing Applications", Proc. of the 6th Annual Symposium on Computer Architecture, April 23-25, 1979, Philadelphia

[3] Alexandridis N.A.,"Bit-Sliced Microprocessor Architecture", Computer, IEEE Press,, Vol. 11, No. 6, June 1978, p. 56

[4] Adams W.T. and Smith S.M., "How Bit-Slice Families Compare: Part 1, Evaluating Processor Elements", Electronics, Vol. 51, August 3, 1978, p. 91

[5] Adams W.T. and Smith S.M., "How Bit-Slice Families Compare: Part 2, Sizing Up the Microcontrollers", Electronics, Vol. 51, August 17, 1978, p. 96

[6] Motorola Semiconductor Product Inc.,"M2900 TTL Processor Family",Motorola Inc., 1977, USA

[7] Gougeon P.A., "MACS Simulator User Manual", Research Report No. 4, MACS Project, March 31, 1979, Supply and Services Canada, Contract No. OSU-78-00099

[8] Armstrong C.V.W., Ahmed A.M., Brans N.A., "The Control Aspects of Multiprocessor Array Computing Structure for Radar Signal Processing Applications", The Second Rocky Mountain Symposium on Microcomputers: System, Software, Architecture, August 27-30, 1978, Pingrec Park, Colorado

[9] Armstrong C.V.W., "The Use of a Multimicroprocessor System for Automatic Plot-to-Track Correlation", Proc. of Seventh Int. Symposium on Mini-and-Microcomputers, Jan. 15-18, 1979, Anaheim, California

[10] Armstrong C.V.W., "Research on Microprocessor Arrays for Signal Processing at High Data Rates", Research Report No. 3, MACS Project, March 31, 1979, Supply and Services Canada, Contract No. OSU-78-00099

Chapter VI

THE MACS SIMULATOR

6.1 INTRODUCTION

Simulators have been recognized as a valuable tool by scientists and engineers. Analog and hybrid computers have been used for a number of years to simulate various physical systems. Mechanical and aerodynamic engineers especially, have used simulation extensively. A simulator was also developed for the Multimicroprocessor Array Computing Structure.

Normally, a simulator is used as a design tool in the early stages of conception in order to provide an economical way of verifying the system potential [1]. The practicality and performance of future hardware improvements can be tested, preventing hardware bugs, and major flaws before it is too late. The simulator can also be used to test abnormal conditions that can not readely be verified on a real system, for example a processing element failure. With regards to computer simulation, simulators facilitate

software development by increasing the debugging capabilities [2]. In general, it can be said that the benefits obtained by developing a simulator are so important that a large and complex design project should never be undertaken without one.

The MACS simulator can be used to implement and debug microprograms dedicated to run on the MACS system. There are twelve programs, one for each physical device. This allows the programmer to easily modify the architecture of the simulated MACS by simply creating the appropriate programs. The interaction between the programs is minimal. In order that the simulator be easily transportable from one computer to an another, the subprograms were implemented in FORTRAN when possible and in MACRO-11 assembly language when inevitable.

6.2 THE SIMULATOR CAPABILITIES

In this section the simulator capabilities are infered via a description of the simulation process and an example of the kind of results it produces.

Once the architecture is fixed, the MACS simulator is only used to implement, test, and debug microprograms. To acheive these goals, three major steps are followed:

-creation of the microprograms

-creation of the interface programs

-running the simulator

6.2.1 Creation of the Microprograms

Creating microprograms is done with the help of the INMPRG program. This program uses the terminal to prompt the user for information. The information is entered for one processing element at a time, one microinstruction at a time. The microinstructions are typed in, either directly as complete microinstructions represented in octal, or field by field, letting the computer compose the microinstructions.

Entering microprograms by microinstructions in their octal format consists of typing lines representing microinstructions and their associated comments. Each line is formed of 20 octal numbers and 50 comment characters, separated by a semi-colon. The 20 octal numbers represent the 53 bits of a microinstruction, plus some leading zeros left for future expansion.

Entering microinstructions field by field is easier. The user is prompted field after field for an octal value

(following the sequence of fig. 5.7). The number of bits expected for each field is displayed beside the field name. It is sufficient to only know roughly (with experience) what each microinstruction is supposed to do, taking extra care entering the important field properly and entering default values in the fields that are not of current interest.

After the microprogram has been entered, it is saved on disk to be used directly by the simulator or for future corrections. A program called HCMPRG is also available to help the user get prepared for a terminal session with INMPRG. This program prints a list of the field prompts in a questionnaire form, reducing the composition of a microprogram to filling the questionnaire.

6.2.2 Creation of Interface Programs

The interface programs are the programs simulating the behavior of the top buffer, the bottom buffer and the working memory. These elements of the MACS system are programmable, but their hardware is not defined yet. Therefore, their functions, rather than the devices themselves, are simulated in FORTRAN subroutines.

These subroutines must then be rewritten for each different application. The process is simple and easily done using previous versions of the subroutine as guidelines. The programs consist mainly of declaration and IF statements. The subroutines are named: TOPBUF, BOTBUF, WORKMI and WORKMO. They receive the clock cycle number as input and produce an output accordingly.

6.2.3 The Simulation Process

After the microprograms and the interfaces programs have been created, the simulator is ready to be run. The program MACSIM offers a monitor in order to control the execution of the simulator and thus debug the microprograms. MACSIM is a user oriented program which prompts the user for information in a self explanatory way.

As MACSIM is executed (see fig. 6.1), the user is first prompted for the desired level of information. The simulator can be initially run at any of the six information levels available. The user has the possibility of changing the information level at any point during the simulation process. The most often used levels are: zero, one and four. Level one is used to get a general appreciation of what is happening in the PE column, namely, which PEs are

enabled, what data are getting in or out of the PEs. Level four is used for extensive debugging, giving the maximun amount of information available. Finaly, level zero is used to get as fast as possible to a breakpoint of interest.

The next step in initializing a simulation run is to specify the number of processing elements to be used. Then, the user is prompted for the names of the disk files containing the microprograms, the external RAM data, the working memory data, the top buffer data and the bottom buffer data. It is only after these requirements are satisfied that the simulator goes into the monitor mode and displays the available monitor commands.

Figure 6.1 shows a typical simulation session and demonstrates the use of the monitor commands. As an example, we follow what has been done in that session.

After the initialization process and at the first monitor command prompt, a BREAK command is issued. In this case, the command is used to set a breakpoint within the second clock cycle, just before the sequential simulation reaches the execution of the microinstruction of the third processing element.

Then the GO command is issued and the simulator performs the first microinstruction of each PE. An example of the display obtained with the level of information one is shown. Before proceeding to the next clock cycle, the final values of the busses for the current clock cycle are displayed. Stars are synonymous with high impedance mode.

While going through the second clock cycle, the simulator reaches the previously set breakpoint. The user then decides to change to information level four using the INFO command and to set the simulator in single step mode. In the single-stepping mode, the simulator only executes one microinstruction at the time, the monitor prompting the user between each of them.

A CONTINUE command being issued, the simulator executes the microinstruction associated with the third PE for clock cycle two. The information level four is now in use. The microinstruction and its field by field expansion are displayed. The outputs at major ports of the processing element are shown, as well as the internal and external registers.

Before executing the next microinstruction, the user wants to see the previous condition of the fourth processing element. Thus a DISPLAY command is issued. The DISPLAY

command gives information about the last microinstruction executed by the PE and the present status of its registers. Using the MODIFY command, the user can change the values stored in these registers. The change made to internal register four is observed later, when a CONTINUE command is issued.

Finaly, the monitor also has a command named REMOVE. This command is used to remove breakpoints or exit from the single step mode. For the removal of breakpoints, three options are available. It is possible to remove either a specific breakpoint, the next breakpoint, or all the breakpoints.

```
>RUN MACSIM
*****************************************
*                                       *
*             MACS  SIMULATOR           *
*                                       *
*         BY: FRANCOIS A. GOUGEON       *
*             UNIVERSITY OF OTTAWA      *
*****************************************

LEVELS OF INFORMATION AVAILABLE:
LEVEL 1-->CRT DISPLAY
LEVEL 2-->LIMITED PRINTOUT
LEVEL 3-->AVERAGE PRINTOUT
LEVEL 4-->EXTENSIVE PRINTOUT
LEVEL 5-->REGISTER&MEMORY INFO. ONLY
LEVEL 0-->NO INFORMATION
WHAT IS THE LEVEL OF INFORMATION NEEDED?
1
WHAT IS THE NUMBER OF PE NEEDED?
5
WHAT IS THE BIG MPROG. FILE NAME?LIKALL.MIC
WHAT IS THE EXTRAM DATA FILE NAME?ERLIK.BUF
WHAT IS THE WORKING MEMORY DATA FILENAME?WMLIK.BUF
WHAT IS THE TOPBUFFER DATA FILENEME?TBLIK.BUF
WHAT IS THE BOTTOM BUFFER DATA FILENAME?BBLIK.BUF
YOU ARE NOW IN MONITOR MODE
THE MONITOR COMMANDS ARE:
        GO->TO START THE EXECUTION
        BREAK->TO INSERT BREAKPOINTS
        CONTINUE->TO CONTINUE AFTER A BREAK OR A STEP
        REMOVE->TO REMOVE BREAKPOINTS OR STEP PROCESS
        MODIFY->TO MAKE SOME MODIFICATIONS TO THE
                DATA WHILE AT A BREAKPOINT
        DISPLAY->TO DISPLAY INFORMATION AT BP
        INFO->TO CHANGE THE INFORMATION LEVEL
        STEP->TO PROCEED STEP BY STEP
THE FIRST LETTER IS SUFFICIENT(E.G.:G->GO)
MONITOR COMMAND?B
AT WHAT CLOCK CYCLE?2
FOR WHICH PE?3
MONITOR COMMAND?G
PE 1  E  IMXOUT:    0  YOUT:    0
PE 2  E  IMXOUT:    0  YOUT:    0
PE 3  E  IMXOUT: 1661  YOUT: 1661
PE 4  E  IMXOUT:    0  YOUT:    0
PE 5  E  IMXOUT:    0  YOUT:    0
CLOCK CYCLE:  1.          SYSBUS:*****          WMEMBUS:*****
H
PE 1  E  IMXOUT:  400  YOUT:  400
PE 2  E  IMXOUT:    0  YOUT:    0
MONITOR COMMAND?I
WHAT IS THE NEW LEVEL OF INFORMATION?4
MONITOR COMMAND?S
MONITOR COMMAND?
```

Fig. 6.1-Monitor Command Examples

```
MONITOR COMMAND?C

CLOCK CYCLE:     2.      PENUM:  3.
MICROINSTRUCTION: 610515440  130  ADDRESS CRAM:     2
OPCODE: 610    CARRY: 1    DISPE: 0    ENPE: 1    BXMS: 0    BXME: 3
IMIS: 3    NOCRA:  2    NACRA:  0    AADDE:  0    BADDE:  0
ACRMC:  2    LMUXC:  1    ERWE:  1    SHIFTC: 0    HMUL: 0
AADDI:  0    BADDI:  0    NACRAI:  2    MUX:  1
ERAMIN: 1661    ERAOUT:     0    ERBOUT:     0
IMXOUT: 2210
YOUT: 7451
XMOUTA:*****  XMOUTB:*****
THE INTERNAL Q REGISTER: 7451
THE INTERNAL MEMORY FOR THAT PE:
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
THE EXTERNAL MEMORY FOR THAT PE:
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
MONITOR COMMAND?D

CLOCK CYCLE:     2.      PENUM:  4.
****INFORMATION FROM PAST CYCLE****
OPCODE: 610    CARRY: 1    DISPE: 0    ENPE: 1    BXMS: 0    BXME: 3
IMIS: 3    NOCRA:  2    NACRA:  0    AADDE:  0    BADDE:  0
ACRMC:  2    LMUXC:  1    ERWE:  1    SHIFTC: 0    HMUL: 0
THE INTERNAL Q REGISTER:     0
THE INTERNAL MEMORY FOR THAT PE:
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
THE EXTERNAL MEMORY FOR THAT PE:
    0  400    0    0    0    0    0    0    0    0    0    0    0    0
MONITOR COMMAND?M
DO YOU WANT TO MODIFY THE Q REGISTER?N
DO YOU WANT TO MODIFY INTERNAL MEMORY?Y
WHAT LOCATION?4
NEWVALUE TO PUT IN INTERNAL MEMORY  4 ?231
DO YOU WANT TO MODIFY INTERNAL MEMORY?N
DO YOU WANT TO MODIFY EXTERNAL MEMORY?N
MONITOR COMMAND?R
REMOVE BREAKPOINTS OR STEP PROCESS(B,S)?B
WHICH BREAKPOINTS DO YOU WANT REMOVED(#,N,A)?A
MONITOR COMMAND?C

CLOCK CYCLE:     2.      PENUM:  4.
MICROINSTRUCTION: 241115040  130  ADDRESS CRAM:     2
OPCODE: 241    CARRY: 0    DISPE: 0    ENPE: 1    BXMS: 0    BXME: 3
IMIS: 2    NOCRA:  2    NACRA:  0    AADDE:  0    BADDE:  0
ACRMC:  2    LMUXC:  1    ERWE:  1    SHIFTC: 0    HMUL: 0
AADDI:  0    BADDI:  0    NACRAI:  2    MUX:  1
ERAMIN:     0    ERAOUT:     0    ERBOUT:     0
IMXOUT:     0
YOUT:     0
XMOUTA:*****  XMOUTB:*****
THE INTERNAL Q REGISTER:     0
THE INTERNAL MEMORY FOR THAT PE:
    0    0    0  231    0    0    0    0    0    0    0    0    0    0
THE EXTERNAL MEMORY FOR THAT PE:
    0  400    0    0    0    0    0    0    0    0    0    0    0    0
MONITOR COMMAND?
```

Fig. 6.1-Monitor Command Examples

6.3 THE SIMULATOR TECHNICAL DESCRIPTION

The MACS simulator is formed of sixteen programs. Twelve programs simulate each a physical device. Four programs establish the communication links between these devices and control the simulation. The package also contains 5 other programs that are of immediate utility or that could be helpful. Programs of immediate utility are associated with the INMPRG program described in the preceeding section. Programs of certain usefulness are programs like HCMPPG and SIMRUN(1).

Seventeen are in FORTRAN while four are in MACRO-11 assembly language. The programs in assembler can easily be translated for uses on any other minicomputer. The package, counting all the usual files (source, list, object, running), can take up to one megabyte of disk space. The simulator presently runs in 44 kilobytes of memory on a PDP11/34.

Before going through how these programs interact with each other, a very brief description of the purpose of each program is needed:

---

(1) see description on page 95

Device Programs:

M29014-->simulates the AMD2901A integrated circuit

S2900 -->simulates three AMD2901A together, plus the
AM25LS253 multiplexers and the three-states
for multiplication (using M29014)

PRGRAM-->simulates the opcode RAM

ADDRAM-->simulates the address RAM

EXTRAM-->simulates the external memory (registers)

RCIM -->simulates the input multiplexer

XM -->simulates the output multiplexer

HARMUL-->simulates the hardware multiplier

TOPBUF-->simulates the top buffer

BOTBUF-->simulates the bottom buffer

WOPKMI-->simulate the working memory

WORKMO

Communication and Control Programs:

PEPFM -->simulates a complete processing element

MACRUN-->simulates the complete MACS

MACSIM-->runs the simulator with the monitor

DISP -->displays information on the terminal

Help and Utility Programs:

INMPRG-->create the microprograms

INMIBF

LSTBF

HCMPRG-->helps to compose microprograms

SIMRUN-->runs the simulator in a simple way

The interaction between these program is minimal. All the programs simulating physical devices within the processing element are subroutines called by PEPEM (see fig. 6.2). The data normally passed from one device to another are transfered via parameters in the calling sequence. The variables representing the control signals are obtained from common blocks. Common blocks are also used for all variables representing internally stored data, in order for this data to be available to the monitor and the display program (DISP).

The timings of the operations within the processing elements are not difficult to replicate, being essentially sequentials. However, at the multiprocessor level, operations are done in parallel. But, since the processors are all independent of each other except for a few transferts of information on the busses, the task is simple. The PE are simulated one after the other within the same clock cycle, updating the bus values only after all the PEs have been treated. This is done by the MACRUN program (see fig. 6.3).

MACRUN is initially called by MACSIM after a GO monitor command (see fig. 6.4). MACRUN then starts the simulating process and, for every clock cycle:

-gets data from the top buffer (TOPBUF)

-gets data from the working memory (WORKMI)

-executes a microinstruction corresponding to the clock cycle in progress for every PE, taking care of inter-PE communications (PEPEM)

-displays some information (DISP)

-takes care of all the busses

-sends data to the bottom buffer (BOTBUF)

-sends data to the working memory (WORKMO)

However, for every PE of each clock cycle, MACRUN checks for breakpoints or single step mode, and returns the execution to MACSIM if these are detected. MACSIM, which takes care of the monitor, obeys the user's commands and returns to MACRUN at the point of interruption, on a CONTINUE command.

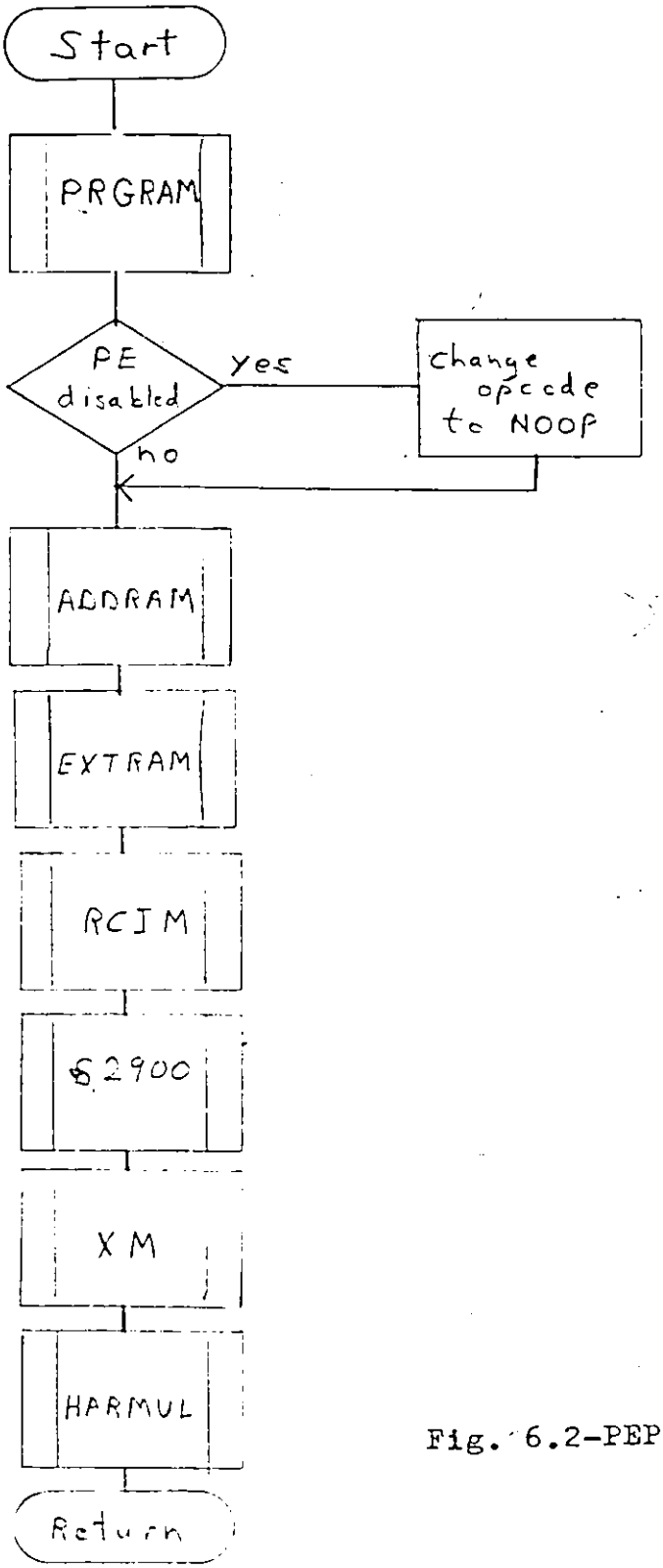


Fig. 6.2-PEPEM Flowchart

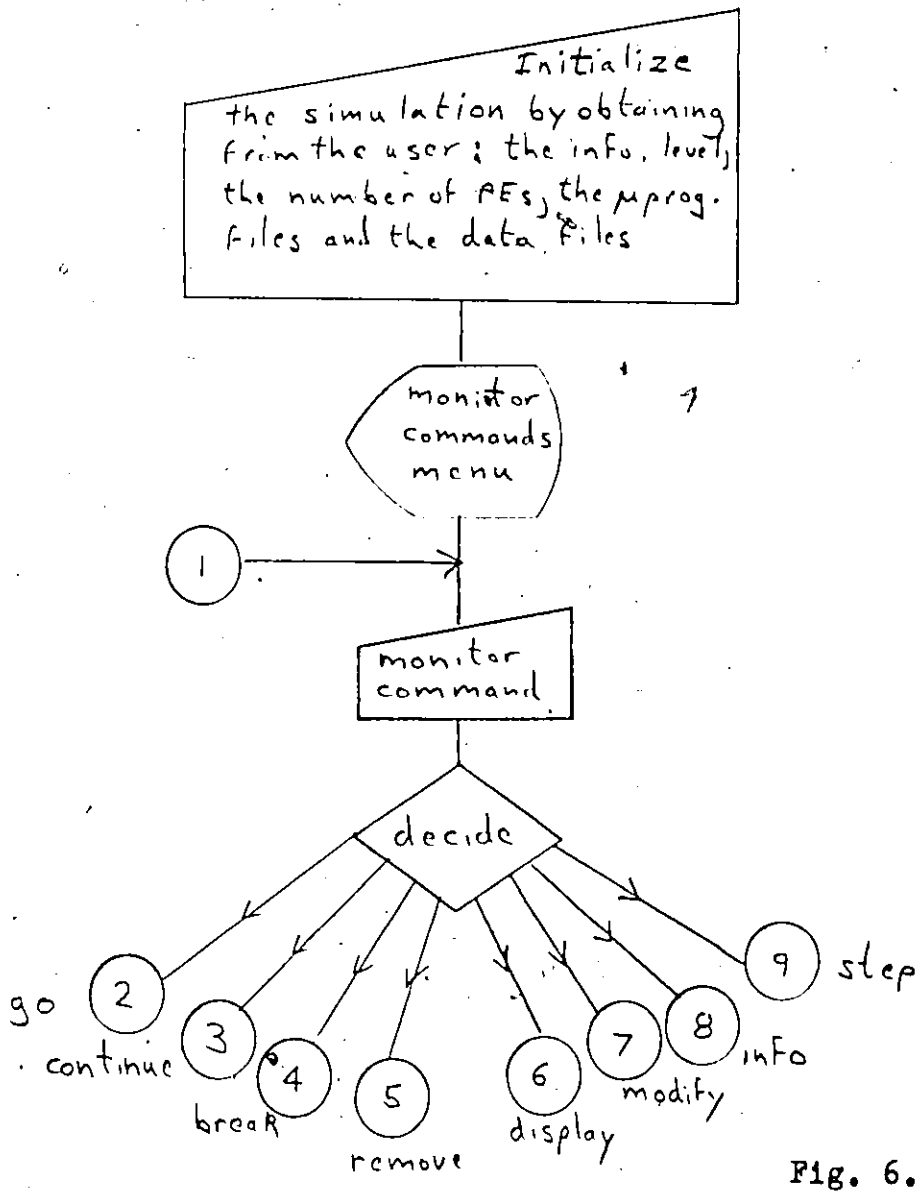


Fig. 6.3-MACSIM Flowchart

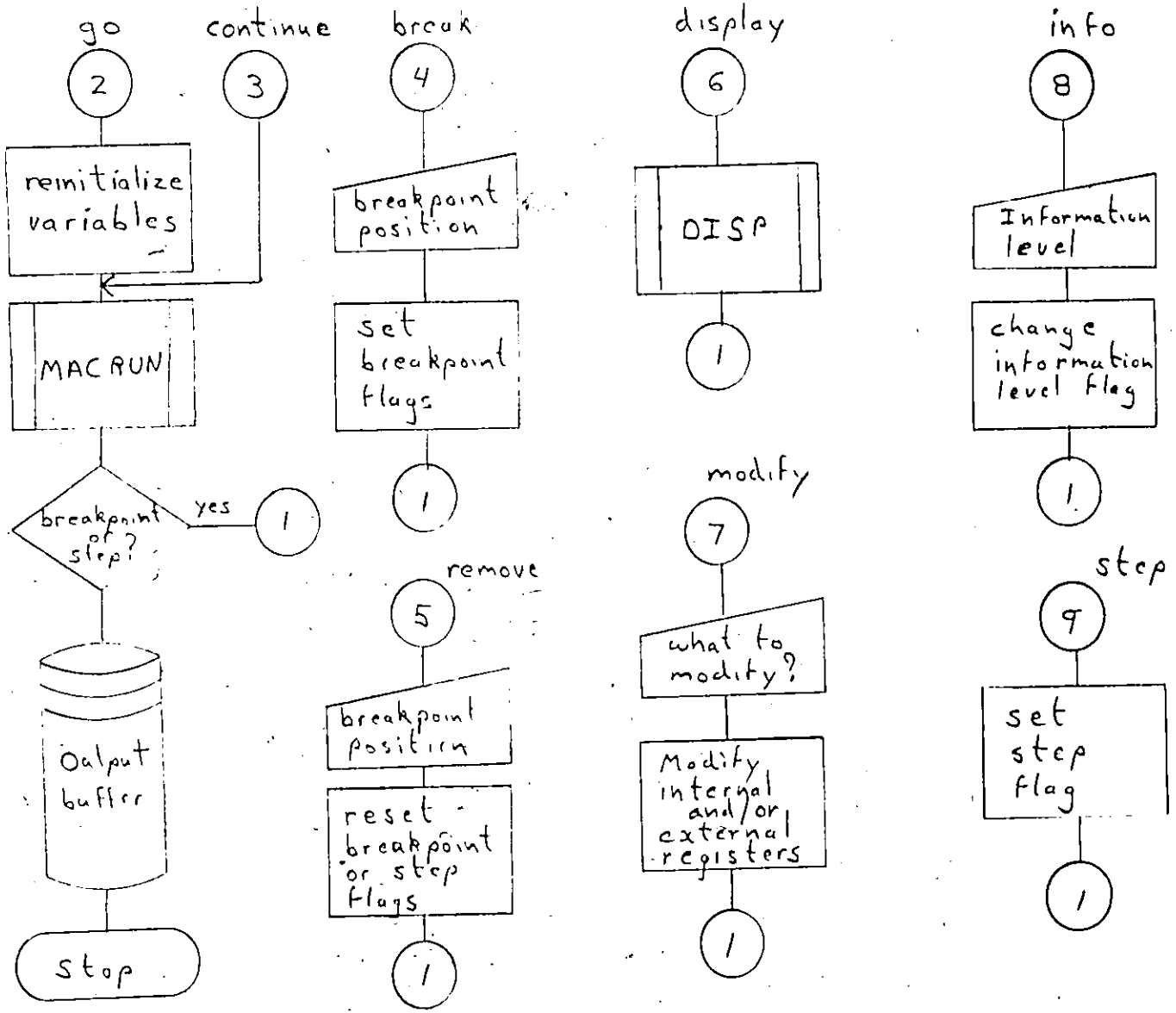


Fig. 6.3-MACSIM Flowchart (cont.)

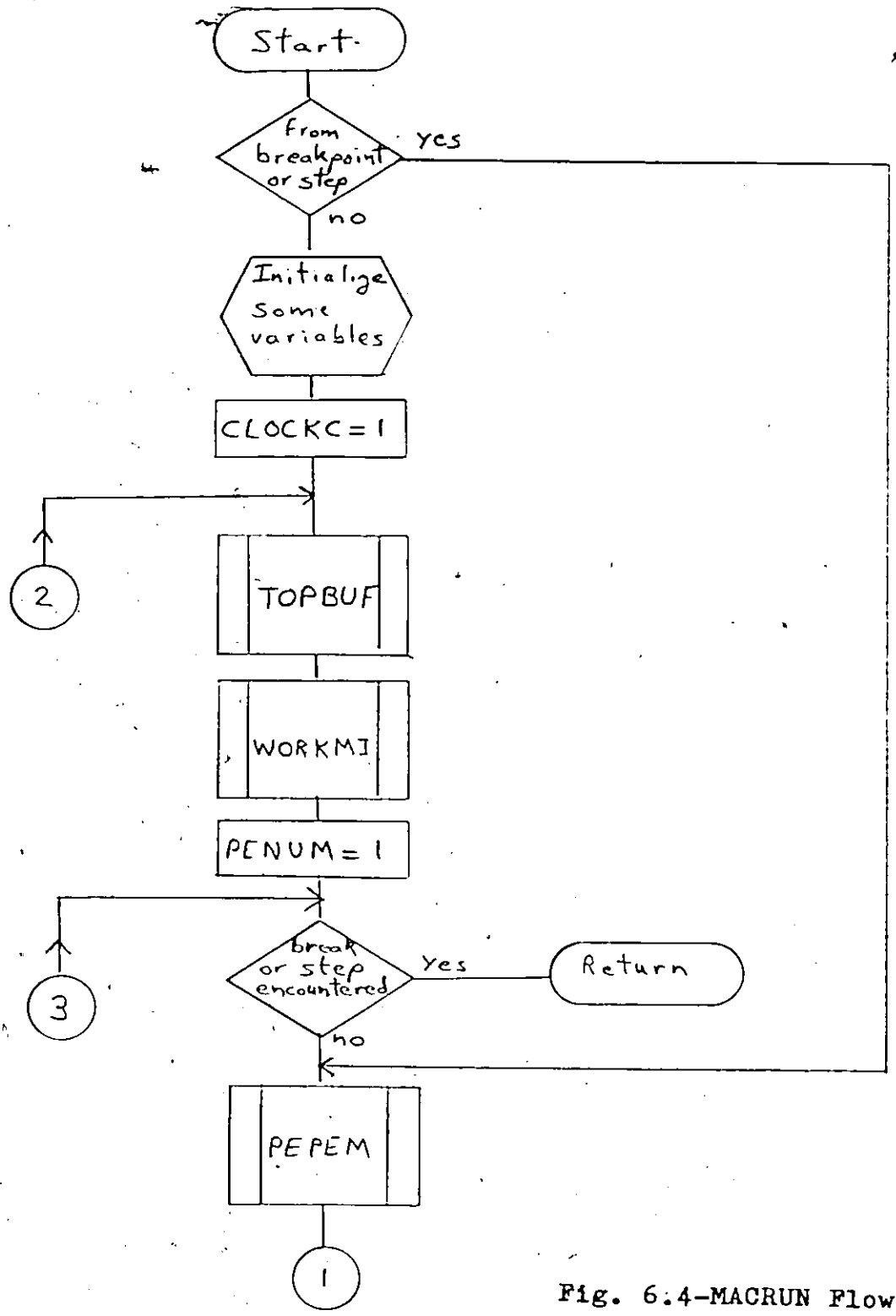


Fig. 6.4-MACRUN Flowchart

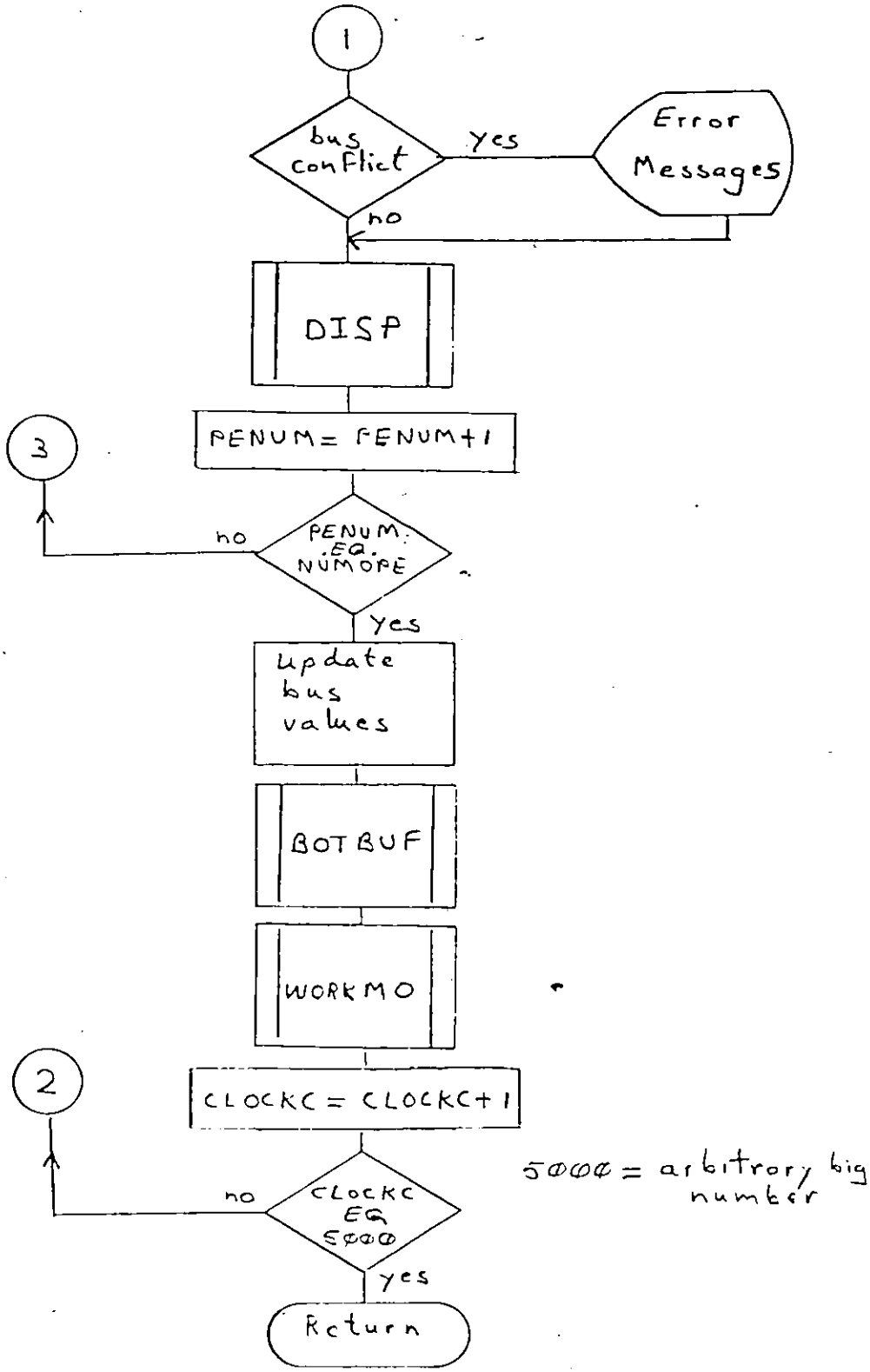


Fig. 6.4-MACRUN Flowchart (cont.)

6.4 CONCLUSION

This chapter describes the MACS simulator. In the second section, the creation of the microprograms and of the interface programs are outlined. Later, the running of the simulator is explained and its capabilities demonstrated using an example. The example has been selected to illustrate all the monitor commands.

In the third section, a descriptioon of the MACS's simulator package is given in terms of: program count, programs utility, programming languages and computer space. The individual programs are named and briefly described. Finaly, the interaction between these different programs are explained.

In the next chapter, this simulator is used to implement some image processing tasks on the Multimicroprocessor Array Computing Structure.

References

[1] Breuer M.A.,"Design Automation of Digital Systems",Prentice Hall, New Jersey,1972

[2] Schneidewind N.F.,"The Use of Simulation in the Evaluation of Software", Computer, IEEE Press, Vol. 10, No. 4, April 77,p. 47

Chapter VII

SOME IMAGE PROCESSING ALGORITHMS FOR MACS

7.1 INTRODUCTION

The Multimicroprocessor Array Computing Structure (MACS) architecture and its associated simulator are described in the two previous chapters. In this chapter, the potential of MACS is investigated for implementation of image processing tasks. A data rate of one pixel/microsecond is set as an objective for these implementations, since it is a suitable rate for the MACS system without modifying its present architecture (16 microinst. per PE).

Others have investigated systems for potential usefulness in the image processing domain. Figure 7.1 gives a list of these systems and classifies them in three categories: non-specialized, specialized, and specialized microprocessor based systems. Specialized is used to indicate that the primary goal of these systems is to perform image processing tasks. Other dedicated systems are commercially available, for example, Comtal Vision One/20 [16],but they are

hardwired machines and are not within the scope of this thesis.

Three typical image processing tasks are tested on the MACS simulator: filtering, resampling, and fast Fourier transform. These techniques are frequently applied in image processing. The filtering operation, performed by windowing techniques (1), is mainly an I/O bound operation. This means that the I/O operations take more time than the actual computation. This occurs because two spatial dimensions are considered at the same time. The fast Fourier transform on the other hand, is mainly computation bound, where the calculations involved for every pixel are time consuming. Also, the two-dimensional FFT need only be applied to one dimension at the time (it is seperable) [17]. The requirements of resampling methods lie between these two extremes.

This chapter is divided into four major parts:

- -salt and pepper filter by windowing technique [18]
- -resampling using the MSINC function [19]
- -resampling by WEIMAN's method [20]
- -fast Fourier transform [17]

---

(1) to be explained in section 7.2.1

One of the resampling methods, the MSINC function, is often used as reference for evaluating other resampling methods. The other one, Weiman's resampling method, is said to be more compatible with parallel processing.

Each of these sections are subdivided into three parts: the theory, the implementation, and some remarks on the performance of the method on the MACS processor.

Non-Specialized

ILLIAC (Burroughs) [8]

ASC (TI) [1]

STAR-100 (CDC) [6]

PEPE (Bell) [5]

STARAN [3][4]

HARPP (Honeywell) [2]

CYPHM 60 [7]

Specialized

CLIP [9]

PPM [10]

FMPP [13]

Specialized Microprocessors Based

GLOPP [11]

MPIP [12]

G-471 [14]

IAP [15]

Figure 7.1 Some Image Processing Systems

7.2 THE SALT AND PEPPER FILTER

7.2.1 The theory

Pictures invariably suffer some degradation due to sensors, recording or transmitting instruments, or other external factors such as haze in remote sensing applications. When the degradation process is well known and its effects easily described mathematically, a specific corrective process can be applied to the picture. This is called image restoration [21]. However, in most cases, either because the degradation process is not known well enough or because simpler methods of improving the picture are sufficient, more heuristic methods of image enhancement are often used. These methods are used to increase the contrast, to deblurr, to smooth, to remove noise or to geometricaly adjust the pictures [31].

One of these enhancement method is the salt-and-pepper filter. It is used when the image noise consists of isolated points that contrast with their neighbours, the so called, salt-and-pepper noise. The computer has to recognize a noise point and eliminate it. The detection of noise points is done by comparing each point's gray level with the level of its neighbours. Only the points within a limited distance from the point under consideration (within

a window) are checked. If the pixel of interest is substantially different from all, or nearly all, its neighbours, it is considered to be noise. The noise point is then replaced by some function of the neighbouring pixels, for example the average, the median, or the mode [18]. Here we consider the average value.

A window is an area containing the pixel of interest and its immediate neighbours. Standard sizes for rectangular windows are of the order of three to seven pixels wide, forming windows of 3*3, 5*5, or 7*7. Odd numbers are used in order for the pixel of interest to be at the center of these areas. The operations using the so called windowing techniques consider only the area defined by the window as their operating area. This limits the amount of calculation involved since the amount of input parameters is limited. The window usually moves across the image as the point of interest changes.

The salt-and-pepper filter uses the window technique approach. Several parameters can be adjusted within this filter. The window size can be modified to increase or decrease the neighbourhood size. The threshold value by which the central pixel has to differ from another pixel to be considered substantially different can also be made adjustable. The number of pixels from which the pixel of interest is substantially different can also be modified.

Some complications are associated with this method. If the window is large and all the contained pixels are to be used in evaluating the average value, more weight should be given to nearest neighbours. This increases the computing time required for each pixel. Another problem is encountered for the border points of the image. Two methods can be used to alleviate this problem. One consists in assuming that a mirror image of the picture is present outside the border line. A simpler method involves giving outside pixels a null value and simply disregarding the results near the borders. In this implementation we ignore these problems.

7.2.2 The Implementation

A salt-and-pepper filter based on a 3*3 window is implemented on MACS. On such a window, eight comparisons are necessary between the pixel of interest and its surrounding neighbours. For each neighbour, the absolute value of the difference with the central pixel is calculated. This difference is then compared with a predetermined threshold value. If the neighbouring pixels are substantially different from the pixel of interest, a flag is set.

In the implemention, eight processing elements are used to compare the eight surrounding pixels with the main pixel. Each PE first gets the pixel of interest, then after a certain amount of wait instructions, the pixel to be compared. The PEs are programmed according to the desciption in the preceding paragraph. The operations are done concurrently, each PE being one instruction out of phase from the previous PE. A typical microprogram is shown in figure 7.2. Note that the flag set or reset by each comparison is forwarded to the next processing element. This implies that after PE8, a decision or weight factor is obtained, corresponding to a count of the number of divergences encountered.

Two other processing elements are needed to complete the operation. PE9 computes the average value of the eight surrounding pixels. This value is substituted for the value of the pixel of interest if it is found to be noise. It forwards to the next PE the computed average value, the center pixel value, and the weight value. This is shown in figure 7.3. PE10 makes the decision, using the weight received from above and comparing it with a preset value. This processor decides if the center pixel is different from all, or nearly all, its neighbours. If it is substantially different, it is considered to be noise and the average value is output. If not, the center pixel is output, unchanged. This is shown in figure 7.4.

In this implementation, the top buffer is assumed to be capable of storing four lines of the input picture. This gives time to the host computer, or any system, to feed in the fourth line while the three other lines are being processed. The programmable top buffer is in charge of outputting all the pixels within a window to the system bus, starting with the center pixel. The pixels are picked up from the bus by the processing elements as specified by their own microprogram.

Each time a window is processed, the bottom buffer receives only one pixel. This pixel is either the pixel from the original image, or the computed average of the window. Therefore the bottom buffer must have the capacity of storing two lines. One is to be created while the processing is done, the other one, is the previously created line to be read by the host computer.

7.2.3 Remarks

The preceding section has demonstrated that a salt-and-pepper filter is realizable on MACS. The filter is applied to an input picture using a 3*3 window. The interface with the host computer is minimal, requiring only the top buffer to receive the picture lines one after the other, and the bottom buffer to return the output lines in the same fashion. The filter parameters can be easily modified since

they reside in external memory, directly accessable by the outside world. The complete operation requires only ten processing elements. A data rate of one pixel per microsecond is easily maintained. Extrapolation to bigger windows, 5*5, 7*7, is easily done if we do not limit the MACS'PEs to 16 microinstructions per microprogram. The speed would go down to two or three microseconds per pixel.

Typical images are 512 pixels in length by 512 pixels in width (262,144 pixels). Image processing systems using conventional raster scanned television equipment need a frame refreshing rate of 30 frames/second. MACS, with a data rate of one pixel per microsecond, can acheive a frame rate of 4 frames/second. This is sufficient for images where nc movement is involved, e.g. satellite images. To use the MACS in a real time environment where movement is encountered, 256 by 256 pixels images would have to be used. This would imply an acceptable frame rate of 15 frame/second.

Many other enhancement techniques using windows could be implemented in a similar way: averaging, thresholding, edge detection, etc...[18]

```
PIP TI:=SPFIL1.MIC
703115420  130   21;GET CENTER PIXEL FROM SYSBUS IN R1
513715440  130  442;GET PIXEL OF INTEREST,R1-INPUT->R2,DISABLE IF NEG.
323415060  130   43;COMPLEMENT R2,IF NOT DISABLE
323515100  130   44;COMPLEMENT R2,R2->ABS(DIFFERENCE)
513715120 4130 1065;DIFFERENCE(R2)-THRESHOLD(ER1)->R3,DISABLE IF NEG.
200415140  130   6;ADD 1 TO Q IF NOT DISABLE
200115160  130    7;OUTPUT WEIGHT(Q) BELOW,Q=1 ->OUT OF RANGE
240115200  130   10;WAIT
240115220  130   11;WAIT
240115240  130   12;WAIT
240115260  130   13;WAIT
240115300  130   14;WAIT
240115320  130   15;WAIT
240115340  130   16;WAIT
240115360  130   17;WAIT
240115000  130    0;WAIT,BACK TO BEGINNING
```

Fig. 7.2-The SPFIL1.MIC Microprogram

```
703115420  130   21;GET CENTER PIXEL FROM SYSBUS IN R1
703115440  130 1042;GET PIXEL FROM SYSBUS IN R2
503115460  130 1043;GET NEXT PIXEL FROM SYSBUS,ADD TO R2
503115500  130 1044;GET NEXT PIXEL FROM SYSBUS,ADD TO R2
503115520  130 1045;GET NEXT PIXEL FROM SYSBUS,ADD TO R2
503115540  130 1046;GET NEXT PIXEL FROM SYSBUS,ADD TO R2
503115560  130 1047;GET NEXT PIXEL FROM SYSBUS,ADD TO R2
503115600  130 1050;GET NEXT PIXEL FROM SYSBUS,ADD TO R2
503115620  130 1051;GET LAST PIXEL FROM SYSBUS,ADD TO R2
305115240  130   52;DIVIDE R2 BY TWO
305115260  130   53;DIVIDE R2 BY TWO
305115300  130   54;DIVIDE R2 BY TWO,AVERAGE DOWN
301115320  130   35;CENTER PIXEL(R1) DOWN
240115340  130   16;WAIT
701114360  130   17;WEIGHT FROM ABOVE,DOWN
240115000  130    0;WAIT,BACK TO BEGIN
```

Fig. 7.3-The SPFIL9.MIC Microprogram

```
PIF TI:=SPFI10.MIC
403015020   130   441;MOVE R1 TO R2 IF NOT DISABLED
301115040   130    42;OUT R2 DOWN, NEW PIXEL VALUE
241115060   130     3;WAIT
241115100   130     4;WAIT
241115120   130     5;WAIT
241115140   130     6;WAIT
241115160   130     7;WAIT
241115200   130    10;WAIT
241115220   130    11;WAIT
241115240   130    12;WAIT
241115260   130    13;WAIT
703119300  4130    74;ER1-[R3;# OF PIXELS TO DIFFER FROM TO BE NOISE
703114320   130    35;GET AVERAGE FROM ABOVE IN R1
703114340   130    56;GET CENTER PIXEL FROM ABOVE IN R2
241115360   130    17;WAIT
521714000   130  1400;GET WEIGHT FROM ABOVE-R3,DISABLE IF NEG.
```

Fig. 7.4-The SPFI10.MIC Microprogram

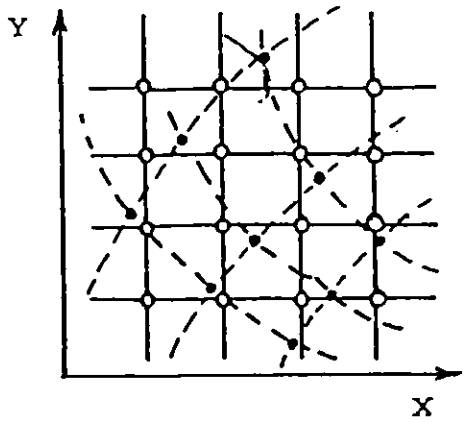


Fig. 7.5-Changing an Image to New Coordinates

7.3 RESAMPLING BY THE MSINC FUNCTION

7.3.1 Introduction

A discrete image, as used by a computer, can be considered as a grid made of samples of a continuous image. It may be necessary to change the image to a new grid format, for example, in going to a new coordinate system (see fig. 7.5). The optimal method would be to resample the continous image according to the new grid. However, in most cases, this is impractical or impossible. Interpolation of the new samples values between the existing samples is more convenient. This is what is usually called resampling.

Resampling is often used in image processing. One of the most important use of resampling is with satellite images. The samples of image picked-up from a satellite are usually distributed in time. For many applications, such as comparisons with existing maps, a uniform distribution in space is much more desirable [19]. Other important uses of resampling are for zooming capabilities or rotation of images.

It can be shown [22], that if a sampled image is transformed to the frequency domain, the spectrum of the original image (continous) is repeated every 2PI intervals.

To reconstruct the image in the space domain, the inverse Fourier transform needs only to be applied to one such interval. This, transposed in the space domain is equivalent to a convolution with sin(PIx)/PIx , the SINC function.

Unfortunatly the SINC function is usually impractical since it decays to zero very slowly. This implies that to use the SINC function to its full potential, a large number of input pixels are needed to evaluate the value of one output pixel. This theoretically best method though, can give some insights as to what to look for in an interpolator. More practical interpolator are: nearest neighbour, linear, MSINC, cubic convolution, cubic spline, quartic spline, etc...[19]

A good interpolator is created by multiplying SIN(PIx)/PIx by the function (1-x**2/n**2) and truncating it at "n" points. "n" is the number of points used on both side of the point of interest to evaluate the resampled pixel (see fig. 7.6). This forms a resampling function that has its derivatives equal to zero at -n and +n, removing the first order discontinuity encountered with SINC when truncated at these points. This function is often used as a standard when comparing the performance of resampling methods. This is why it was chosen for implementation on MACS. It will be called the modified SINC function (MSINC).

7.3.2 The implementation

The modified SINC resampling function can be implemented on MACS. The MSINC function truncated at 8 points is described by:

$$y = f(x) = SIN(PIx)/PIx * (1-x**2/4**2)$$

As example, two typical resampled pixels would look like:

$$y5 = x1*f(4) + x2*f(3) + x3*f(2) + x4*f(1) + x5 + x6*f(1) + x7*f(2) + x8*f(3) + x9*f(4)$$

$$y6 = x2*f(4) + x3*f(3) + x4*f(2) + x5*f(1) + x6 + x7*f(1) + x8*f(2) + x9*f(3) + x10*f(4)$$

Note that in this case, nine pixels of the original image are needed to produce one resampled pixel and, the MSINC function needs to be evaluated for four different values because of its symmetric property. Also, each input pixel needs at one time or an other to be multiplied by each of the function's four values.

The resampling process can be implemented using four processing elements, one for each of the four values taken by the function. These values can be stored in advance in the external memory of the PEs. A fifth PE is used to add the factors and output the resampled pixels.

The process performed by the first four PEs is simple. They get an input pixel from the top buffer or the PE above and multiply it with their MSINC function value stored in external RAM. As soon as the result is obtained it is put on the system bus. Also, after a certain amount of time, the result is put a second time on the system bus. This is done because these results are always needed for the calculation of two output pixels. This process is shown in figure 7.7.

The fifth PE collects these values in order to evaluate the output pixel. Every clock cycle the PE receives a number from the system bus or from the PE above, and adds it to a different but consecutive internal register. After the nine required additions to a specific register have been performed, the resampled pixel is output. This is shown in figure 7.8. The first valid output pixel is made available nine machine cycles (1 machine cycle= 16 clock cycles) after the first input pixel has been taken in.

7.3.3 Remarks

The modified SINC resampling function for 8 points has been realized using five processing elements. This implementation is capable of handling a data rate of one

pixel per microsecond. However, there is a five microseconds delay between an input and it corresponding output.

The technique by which the function is implemented in this example can be used as guideline for implementations using up to 14 input points, still limiting the microprograms tc 16 microinstructions. A 14 points modified SINC function is quite resonnable. It could keep-up with a data rate of one microsecond per pixel but would introduce an eight micrcseconds delay. Extension to more points necessitates an increase of the microprogram memory.

This is an excellent performance compared to the same algorithm implemented on a minicomputer. Assuming that, as with the MACS, the function has been precalculated for the values of interest, the program would still need to frequently use statements of the form:

$$A = A + (x*B)$$

This statement, coded in machine language, takes up to eleven microseconds to execute on a PDP11/34 minicomputer. For the 14 points resampling function, this statement needs to be repeated 15 times. This implies a total execution time of 165 microseconds per pixel. Therefore, in this case, the MACS is a 165/1 improvement over a PDP11/34. To resample one dimension of a 1000*1000 pixels image, the MACS

would take one second, where as a PDP11/34 would take 2 minutes and 45 seconds.

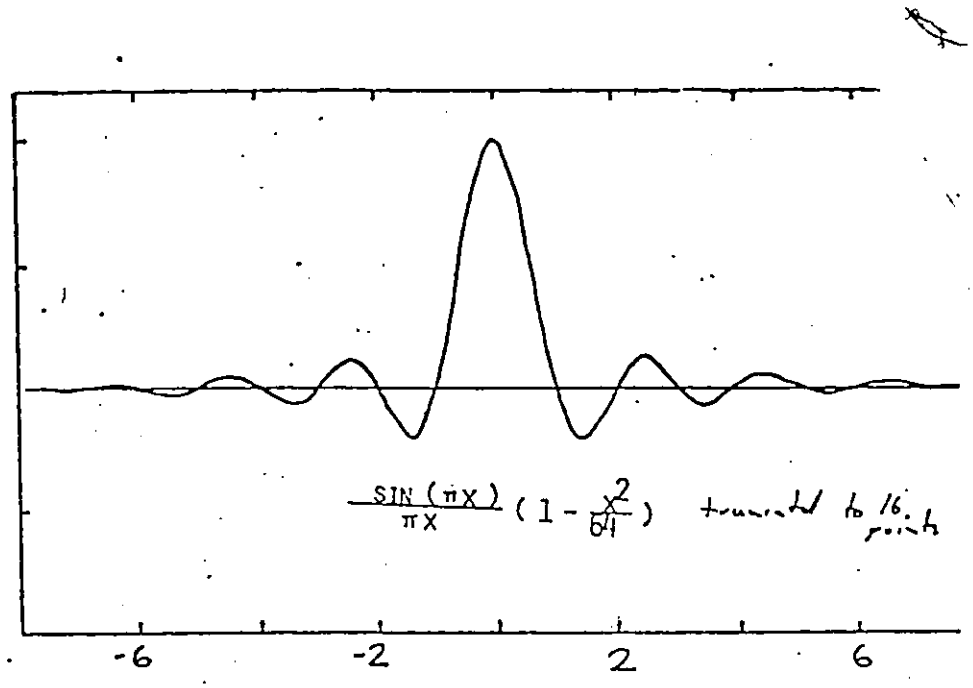


Fig. 7.6-The MSINC Function [19]

```
701114020   130     1;GET X(N) ABOVE,DOWN,X(N)*F(4),F(4) IN ERO
241115040   130     2;WAIT FOR MULTIPLICATION RESULT
241115060   130     3;WAIT
703132100   130     4;GET RESULT IN RO AND ON SYSBUS
241115120   130     5;WAIT
241115140   130     6;WAIT
241115160   130     7;WAIT
241115200   130    10;WAIT
241115220   130    11;WAIT
241115240   130    12;WAIT
241115260   130    13;WAIT
401131300   130    14;GET RO ON SYSBUS
241115320   130    15;WAIT
241115340   130    16;WAIT
241115360   130    17;WAIT
241115000   130     0;WAIT,BACK TO BEGINNING
```

Fig. 7.7-The MSINK1.MIC Microprogram

machine cycle \ clock cycle	5	6	7	8	9	10	11	12	13
1	get $x_1 \cdot f(4)$ in RØ	get $x_1 \cdot f(3)$	get $x_1 \cdot f(2)$	---				— — —	get $x_1 \cdot f(4)$
2	get $x_2 \cdot f(4)$ in R1	get $x_0 \cdot f(3)$ add to RØ	-------						
3	get $x_3 \cdot f(4)$	get $x_3 \cdot f(3)$ add to R1	get $x_3 \cdot f(2)$ add to RØ	— —					
4	get $x_4 \cdot f(4)$ in R3	--		get $x_4 \cdot f(1)$ add to RØ					
5					get x_5 add to RØ				
6						get $x_6 \cdot f(1)$ add to RØ			
7							get $x_7 \cdot f(2)$ add to RØ		
8			Data rate=1μsec.					get $x_8 \cdot f(3)$	
9	get $x_9 \cdot f(4)$ in R8	--	Delay = 9 μsec.						get $x_9 \cdot f(4)$ add to RØ output down
10	get $x_{10} \cdot f(4)$ in R9	--							

Fig. 7.8 - The MSINC Function PE5 Timing Table

7.4 RESAMPLING BY THE WEIMAN

7.4.1 The Theory

In this section, a relativly new and simple way to resample pictures is described. The technique was developped by C. Weiman [20], based on an earlier development by J. Rothstein of a code for digitized straight lines [23]. The resampling method requires only simple calculations, making it very fast. It is said to be naturally oriented toward implementation on parallel processor systems.

Rothstein's code for straight lines was originally conceived as a tool for pattern recognition [24]. A simple short binary code can describe a line with relation to a quadratic lattice (grid). Figure 7.9 pictures a line (I) starting at the origin of a cartesian plane (grid) and going in the first quadrant with a slope smaller than one, but greater than zero.

Each square unit of the grid which is crossed by the line will have its boundaries cut at two points. These two points are either on opposite (parallel) sides or on adjacent (perpendicular) sides. They lie on the adjacent sides whenever the line crosses a vertical division line and

on the opposite sides when the line stays within the same vertical division. When square units of the grid are crossed on the opposite sides, the code is issued a zero for that horizontal division, and when it is on adjacent sides, it gets a one. The horizontal division containing the origin is always set to zero. The final horizontal division, the division just before the line code gets repeated cyclicly (where the line crosses a corner), gets a one.

The relation between Rothstein's code and the application we have in mind, namely resampling, is only one of homogeneity. When crudely shrinking an image, there is a problem in deciding which lines or columns of the original image are to be kept. A similar problem occurs when stretching: which original lines and columns are to be placed in the new wider image. The idea is to get as the result the most homogeneous picture possible. In the WEIMAN's method, the code for a line of slope 5/8 for example, is used as a guideline for shrinking a picture by 5/8 or stretching one by 8/5. The end result is a very homogeneous picture, because, as WEIMAN wrote:

"The code comprises the most homogeneous possible distribution of p 1's among q digits."[20]

In this example, it means an homogeneous distribution of five ones among eight digits.

Continuing with the same example, if an image is to be shrinked by 5/8 in one direction, the image is scanned line by line (column by column), retaining only five lines out of each eight. A circular shifter holding the code for a straight line of slope 5/8 points out the lines to be kept. Only the lines associated with a code value of one are kept. The process can be repeated in the other dimension to shrink a complete two-dimensional image.

The same principle is used for stretching pictures, the code giving the position to be occupied by the old image's lines in the new image. What is more striking when stretching a picture is the empty lines created. The counterpart of this in shrinking, is the fact that some lines are deleted, thus providing no information at all in the new image. These discontinuities could be smoothed out by an averaging process, but this would lead to a considerably degraded picture.

Another solution is available. It was already pointed out that the straight line code represents the most homogeneous distribution of 1's in the total number of code elements. If the elements of this cyclic code are rotated, the homogeneity is conserved. The newly obtained codes can be used in transformations to obtain slightly different pictures. If all the pictures resulting from all the

possible combinations of the code are averaged, a much better result is obtained than by smoothing only one of them. An example of this method is shown in figure 7.10, where a picture is stretched by 4/3 using the code for a line of slope 3/4 (0111).

This method of avoiding discontinuities is fairly simple and leads to very good results. The whole resampling procedure is computationaly simple needing only very basic operations. And finaly, as the next section demonstrates, the WEIMAN's method is easy to implement on multiprocessor systems.

7.4.2 The Implementation

In order to demonstrate how the WEIMAN's method is implemented on MACS, the example of figure 7.10, where an image is stretched by 4/3, will be used. Four processing elements (PEs) are needed, each corresponding to one of the different code patterns. These PEs receive the input pixels from the top buffer, store them temporarely, and output them according to their own version of the code. A fifth PE receives these outputs and averages them to produce the final output pixels. These pixels are then sent to the outside world by the bottom buffer.

The top buffer receives the input pixels from the host computer or any image processing system. These pixels are then output at the rate of the most demanding PE (WEIMAN's code->1110). In our example, this implies that the pixels are output at the rate of one pixel per machine cycle for three cycles, every fourth cycle being without output. The first four PEs are receiving the pixels at the same rate, according to the same code (receiving code->1110). The values are then stored in the PE's internal registers and output according to each PE's WEIMAN's code. For example, PE3 (code->1101) will output a value in the first two machine cycles and in the fourth one.

We examine in more detail the typical microprogram of the first four PEs to obtain a better understanding. First, the processing element receives a pixel according to the value of the receiving code. If at that time the code is a "one", it receives the pixel, if it is a "zero", it disables the receiving operation. If a pixel has been received, it is put into an internal register as designated by pointer 1. Secondly, a temporary register (ER3), assigned to contain the output pixel value, is set to zero and the WEIMAN's code is checked. Depending on the logical value of the code, the next operation is performed or disabled. If the code shows a "one", an internal register value indicated by pointer 2, will be moved to ER3. If the code shows a "zero", the ER3

value will remain at zero. Thirdly, the external register three value will be output.

Each time the receiving or the WEIMAN code is checked, it is also shifted. Rotations would have been better, but since the length of these codes is not necessarely a factor of twelve bits, it becomes impractical. This is why every fourth machine cycle these codes are replaced by fresh ones. The complete microprogram is shown in figure 7.11.

Basically, the first four PEs are acting as delay lines for the incoming pixels, some pixels being delayed more than others, according to WEIMAN's code. Every machine cycle, the fifth PE receives a value out of each preceeding processing element. Note that one of these four values is always a zero. PE5 then, makes the sum and divides the result by four to obtain the output pixel (see fig. 7.12). This pixel is collected by the bottom buffer and stored until the host computer or the host system is ready to receive it.

7.4.3 Remarks

In the preceeding section, a possible implementation of the WEIMAN's method was demonstrated. It takes five

processing elements to realize the stretching of an image by 4/3 and a speed of 3/4 of pixel per microsecond could be acheived. However, as simple as this implementation may seem, it is not possible with the present MACS architecture.

The problem is that pointer 2, as used in the microprogram of fig. 7.11, is not realizable with the present architecture. Pointer 1 is realized using the fact that the address RAM has a seperate memory address register (MARA2). This allows the internal registers to be addressed sequentially to store the input data. However, even if there is some facility (MARA1) to address the internal registers randomly as required by pointer 2, this access can not be disabled when the PE is disabled.

From a designer point of view, the solution is simple. It consist in adding one more memory address register (MARA3) to the address RAM. This MAR would be a counter rather than a register. One bit in the microinstruction would increment it, one bit would reset it to zero, and one bit would be used to disable the increment mode when the PE is disabled.

This simple modification would allow the WEIMAN's method to be implemented on the MACS. At this point though, with three memory address registers for the address RAM, a better

solution would probably be to revise the whole system of addressing of the internal registers.

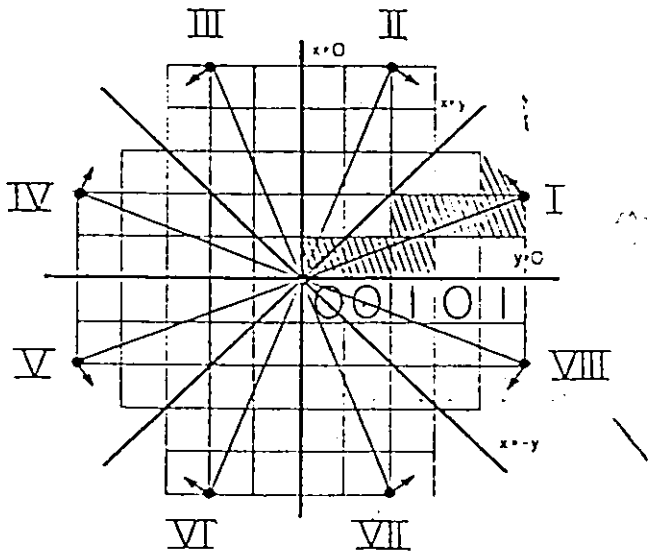
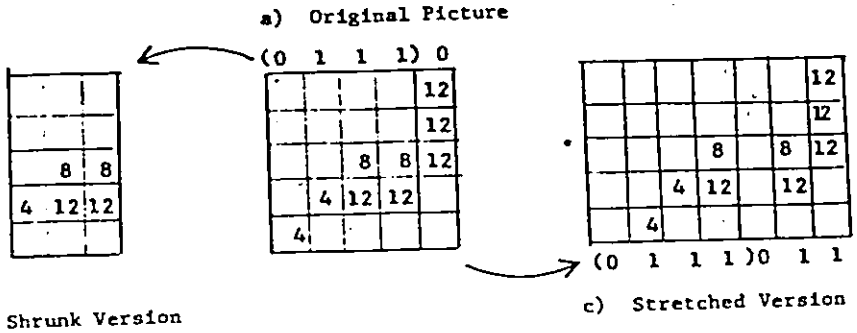
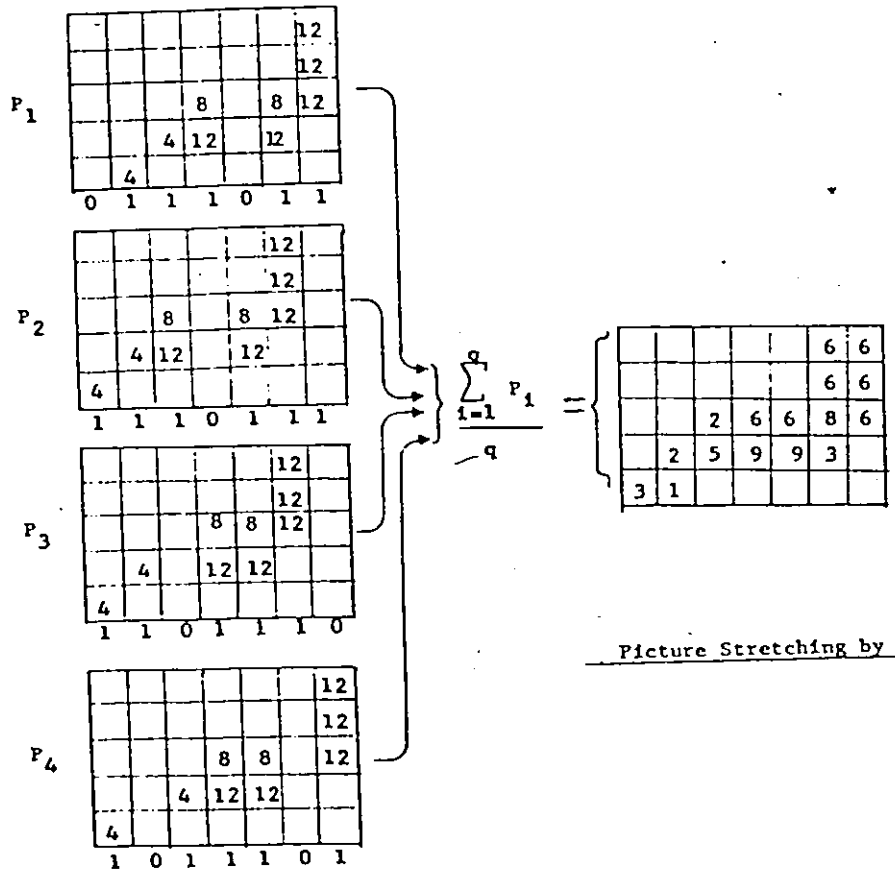


Fig. 7.9-Rothstein's Code Definition [23]



Digitized Stretching and Shrinking



Picture Stretching by Averaging

Fig. 7.10-Weiman's Method Example of Picture Streching [20]

PE 1,2,3,4

ER0 inverted receiving code
ER1 counter
ER2 inverted Weiman's code
ER3 output pixel
ER4 inverted receiving code
ER5 inverted Weiman's code
ER6 initial count value

```
cycle
  1  enable,check receiving code,disable if neg.,shift code
  2  get pixel from sysbus, put in int. reg. (pointer 1)
                increment pointer 1
  3  enable, move zero to ER3
  4  check Weiman's code, disable if neg., shift code
  5  move int. reg.(pointer 2) to ER3, inc. pointer 2
  -  enable, wait
  -  wait
  -  output ER3 to sysbus, enable
  -  increment counter ER1, disable if neg.
  -  replace receiving code ER4 to ER0
  -  replace Weiman's code ER5 to ER2
  -  reinitialize counter ER6 to ER1, go to beginning
```

Fig. 7.11-Weiman's Method PE1,2,3,4 Pseudoinstructions

PE5

```
cycle
  1  clear IR1
 2-6 wait
  7  get value from PE1 in IR1
  8  add value from PE2 to IR1
  9  add value from PE3 to IR1
  10 add value from PE4 to IR1
  11 divide IR1 by 2
  12 divide IR1 by 2
  13 output IR1 down
```

Fig. 7.12-Weiman's Method PE5 Pseudoinstructions

7.5 THE FAST FOURIER TRANSFORM

7.5.1 The Theory

The theory behind the Fourier transform [17] is well known to engineers. It will be briefly outlined here as an introduction to fast Fourier transform and methods of implementing it. The Fourier transform is used to transpose a function from the time domain to the frequency domain. As such, it is very useful for some operations that are more easily implemented in the frequency domain, e.g.: the convolutions are transposed to simple multiplications.

The transform can be stated as:

$$F(w) = \int_{-\infty}^{\infty} f(t) * \exp(jwt)\ dt$$

for continuous time [25]. For discrete time, as in the case of digital processing, the discrete Fourier tranform (DFT) can be stated as:

$$Ap = \sum_{n=1}^{N-1} Xn(WN)^{**}np$$

where Wn=exp(-2j*PI/N).

Ap is a set of N complex numbers and the DFT of the sequence Xn. This is a finite transform as it used N points

to evaluate every DFT element. It involves an extensive amount of computing time, requiring N**2 complex multiplications.

The fast Fourier transform (FFT) brings a net computer time gain over the DFT. In order to use FFT, the length N has to be an integral power of 2, e.g.: N=2**L (L=1,2,3,...). The main idea behind FFT is that the sequence Ap of length N can be obtained by a suitable combination, using N complex multiplications, of the results of two transforms of length N/2. This decomposition is repeated L times (as shown in fig. 7.13) to end up with transforms of length two, where W=exp(-2j*PI/2)= -1. This method reduces the number of complex multiplications from N**2 to NL or Nlog N.

For every L stages, butterflies are computed:

$$A_p = A_p^e + W_N^p * A_p^o$$

$$A_{p+N/2} = A_p^e - W_N^p * A_p^o$$

Since the multiplication with the twiddle factor is the same for both parts of the butterfly, it is not repeated. This brings down the computation load to N/2 log N complex multiplications and N log N complex additions.

The procedure just outlined is called "decimation-in-time FFT" [25]. It has the disadvantage of not addressing the input sequence in its natural order. The inputs must be shuffled prior to performing the FFT. Another version of the FFT algorithm is called "decimationin-frequency FFT". In this case, the output data of the frequency domain is shuffled. The flowgraph for a FFT of length sixteen is shown in fig. 7.14. As predicted it needs 3 stages, 12 complex multiplications, and 24 complex additions. This example is used to illustrate the implementation of FFT on MACS.

7.5.2 The Implementation

In order to implement the fast Fourier transform, it is necessary to simulate the operations of a butterfly. Looking at fig. 7.14 it can be noted that the input data are needed in their natural order. Also, the first butterfly operation can be started as soon as half of the total data is available. So, the first stage must have the capacity of storing N/2 data for a FFT of length N (here 4 for length 8). After the operation has started, stage one is expected to keep-up with the data coming in, passing its results to stage two. Stage two needs to store N/4 complex data before computing some results to be passed on to stage three,etc...

The principle of the pipeline processor developed by Groginsky and Works [26] can be used as guideline in this implementation. Their machine uses one module made of an arithmetic unit, a delay line, and some swithching circuitry, for each FFT stage. Each module receives its rotation vector (twiddle factor(W)) from a common storage area. In MACS case, the work performed by a module could be done by a processing element, and the working memory could take care of the rotation vectors.

The pipeline operation is simple. The FFT of length eight will be used to explain it. The first stage has a four data wide delay line and receives the input data. For the first half of the data (4 data item) the arithmetic unit is disconnected and the data gets stored in the delay line. After the fourth data item, the switches are changed, connecting the arithmetic unit as shown in fig. 7.15. The fifth datum (X4) is then used as input to the arithmetic unit as well as the datum getting out of the delay line (X0). A butterfly operation is performed and one of the result is put at the beginning of the delay line while the other is sent to the next stage.

At this point the second stage can start to operate and shift the data into its delay line. The second stage delay line is only capable of storing two data item. The first

stage performs three more similar butterflies before the switches are reset to their original position. Then as the next data are input, the data stored in the delay line are sent to the second stage. The process is similar for every stage.

A simplified version of this process has been implemented on the MACS simulator. The modules used for every stage are easily simulated by a MACS processing element. For the moment, the delay line is realized using the PE's internal registers and the data are assumed to be integer numbers (not complex). These assumptions are taken in order to simplify the problem. Extrapolation from these to the real situation is straigthfoward as demonstrated in the next section.

The algorithm used is the same for every stage (as shown in fig. 7.16). The only differences are the length of the delay line and the reinitialization values of some counters. Three counters are used: the twiddle counter, in order for the PE to know when to get its rotation vector from the memory bus; the data counter, to count the number of data item tc be put in the delay line before starting any calculation; and finally, the butterfly counter, to know how many butterfly operations are needed. The previous stage corresponds to the PE above and the next stage to the one

below. The butterfly operation, when not disabled, is easily done in the form of:

$$Xout = Xin + Yin * W^{**}Z$$

$$Yout = Xin - Yin * W^{**}Z$$

7.5.3 Remarks

With integer rather than complex numbers, every processing element goes through a cycle of 17 microinstructions. With a microinstruction cycle taking between 60 and 80 nanoseconds, this implies being able to keep-up with a data rate of 1 to 0.5 datum per microsecond. Extrapolation to complex numbers implies that all the data input/output operations as well as all the additions, substractions, and delay line accesses, are to take twice the time used for integer numbers. The multiplication is transposed into four multiplications and two additions, $(a+bj)*(c+dj) = (ac-bd)+(bc+ad)j$, taking 16 cycles rather than three. This implies the execution of 38 microinstructions for a complete data cycle, short of three microseconds.

Under these conditions, a 1024-point complex FFT would take less than four milliseconds. As comparision, the

"Fourier Processing Element" for PDP11 made by Gen Rad [27] takes roughly 100 milliseconds. A better machine, the "Advanced Scientific Array Processor", takes 8.8 milliseconds [28]. The famous AP-102B does the complex FFT in 9.2 milliseconds [29]. The closest in performance is the MAP-300 with 4.5 msec. [30].

Unfortunatly, to implement the complex FFT, some modifications are needed to MACS. The internal registers can not be used anymore to simulate the delay lines. A true hardware delay line or some general purpose random access memory has to be utilized. The simplest solution is probably to increase the capacity of the existing external RAM, assuming that some scheme similar to the address RAM addressing is used.

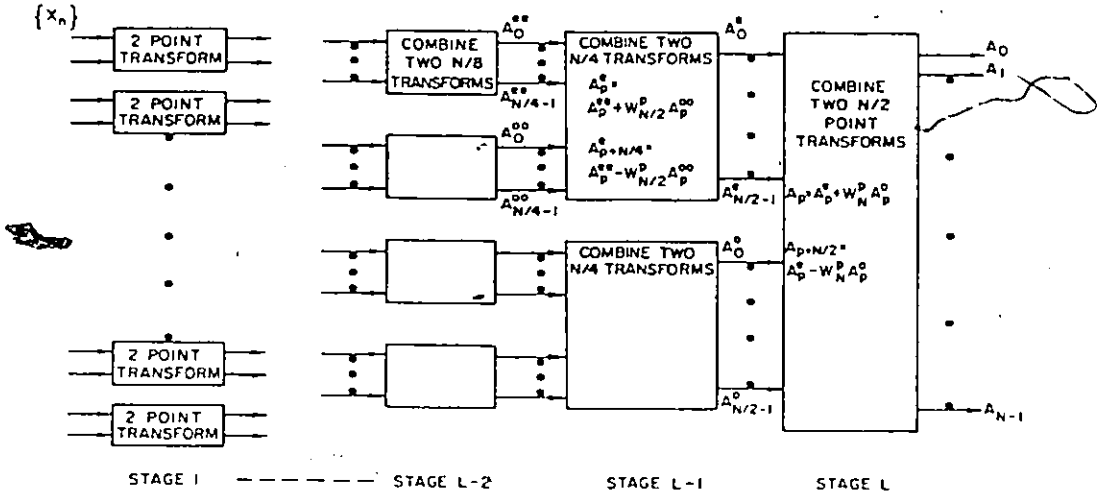


Fig. 7.13-The FFT Decomposition [25]

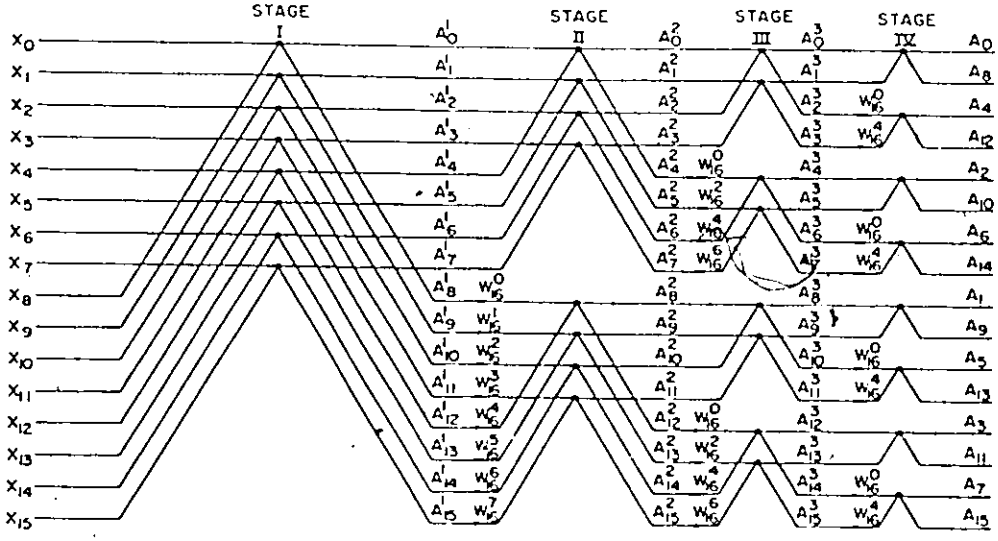


Fig. 7.14-Flowgraph of a FFT of Length Sixteen [25]

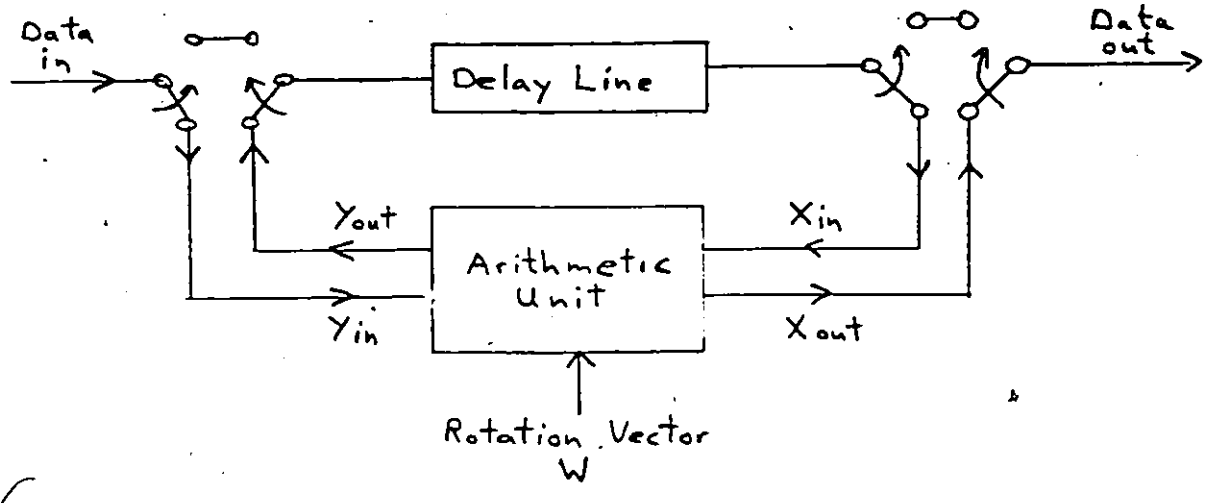


Fig. 7.15-A Pipeline FFT Module [26]

```
70171502035720    1;INCREMENT TWIDDLE COUNT(ER7),DISABLE IF NEG.
701017040 2160  422;INPUT TWIDDLE(W) TO ER10 FROM MEM. BUS
70101506031760 1043;REINITIALIZEE TWIDDLE COUNTER(ER7) WITH ER6
700114100  170 1460;GET DATA FROM ABOVE(YIN) IN Q REG.
401115120  120    0;LEEAST SIGNIFICANT REGISTER(XIN) TO ER0
70171514010520    0;INCREMENT DATA COUNTER(ER2),DISABLE IF NEG.
200015160 2130    0;MULTIPLY Q BY ER10->YIN * W
241015200  170    0;WAIT
700015220  170    0;GET ER0(XIN) INTO Q(WORKING REG.)
701016240  760    0;GET MULTIPLICATION RESULT IN ER3
60101526014160    0;Q(XIN)+ER3->ER0(XOUT)
61041530014170    0;Q(XIN)-ER3->Q(YOUT)
70161532021160    0;INCREMEMENT BUTTERFLY COUNTER(ER4),DISABLE IF NEG.
70101534024560    0;REINITIALIZE DATA COUNTER(ER2) WITH ER5
70101536025160    0;REINITIALIZE BUTTERFLY COUNTER(ER4) WITH ER5
203115000  130    0;ENABLE,PUT Q(YOUT) IN MOST SIGNIFICANT REG.
```

Fig. 7.16-The FFT4.MIC Microprogram

7.6 CONCLUSION

In this chapter four image processing tasks are tried on the MACS simulator. Two of them, the salt-and-pepper filter and the resampling using the MSINC function, are realized using the existing hardware. However, both tasks could have benefited from a simple hardware modification of the architecture: an increase in the microprogram memory. This would allow the filter to be applied using windows bigger than 3*3 and the resampling method to use more than 14 points.

The other two tasks need some hardware modifications to be realizable. The complex fast Fourier transform needs the previouly mentionned change ,plus more private access memory for every PE. The WEIMAN 's method of resampling can only be implemented with a new system of internal register addressing.

This implies that the MACS architecture is not necessarely well suited for image processing applications. The machine is too dedicated. However, most of the general principles used in designing the MACS are good. A somewhat similar machine, slighly more versatile, would make an important part of an image processing system. For example, a processor like MACS could be inserted between an host computer and a display unit.

References

[1] Strokes R.A.,"An Assessment of First Generation Vector Computers", Compcon, San Francisco, Feb. 28-Mar. 3, 1977, IEEE Comp. Soc.,p.12

[2] Marvell O.E., "HAPPE-Honeywell Associative Parallel Processing Ensemble", Proc. Symposium on Computer Architecture, Univ. of Florida, 1973, p.261

[3] Rudolph J.A., "A Production Implementation of an Associative Array Processor-STARAN", AFIPS Conf. Proc., Fall Joint Comp. Conf., Vol. 41, Dec. 5-7, 1972, Anaheim, California, AFIPS Press, New Jersey, p. 229

[4] Batcher K.E., "STARAN/RADCAP Hardware Architecture", Proc. 1973 Sagamore Conf. on Parallel Processing, IEEE Press

[5] Evensen A.J., Troy J.L., "Introduction to the Architecture of a 288-Element PEPE", Proc. 1973 Sagamore Conf. on Parallel Processing, Springer-Verlay, N.Y., 1973, p. 162

[6] Hintz R.G., Tate D.P., "Control Data STAR-100 Processor Design", Architecture", Compcon 72, 6th Annual IEEE

Comp. Soc. Int. Conf., San Francisco, Sep. 12-14, IEEE Press, 1972

[7] Higbie L.C., "The Omen Computer: Associative Array Prooessor", Architecture", Compcon 72, 6th Annual IEEE Comp. Soc. Int. Conf., San Francisco, Sep. 12-14, IEEE Press, 1972, p. 287

[8] Barnes G.H., Brown R.M., Kato M., Kuck D.J., Slotnick D.L., Stokes R.A., "The ILLIAC Computer", IEEE Trans. on Comp., Vol. C-17, No. 8, August 1968, p. 746

[9] Duff M.J.B., Watson D. M., Fountain T.J., Shaw G.K., "A Cellular Logic Array for Image Processing", Pattern Recognition 5,1973, p. 229

[10] Kruse B., "A Parallel Picture Processing Machine", IEEE Trans. on Computers, Vol. C-22, No. 12, Dec. 1973, p.1075

[11] Sellner H.R., "Microprocessor Arrays for Parallel Pattern Recognition", SPIE, Vol. 119, p. 206

[12] Roesser R.P., "Two-Dimentional Microprocessor Pipelines for Image Processing",Computer, IEEE, Computer Society, R.77-55, 1977

[13] Vorgrimler K., "Enhancement of Computing Power in Multiprocessor Systems for Processing of Digitized Pictures", Proc. 1976 Int. Conf. on Parallel Processing, IEEE Press, USA, August 24-27, 1976, p. 11

[14] Gaertner W.W., "Architecture for a Highly Reliable Parallel Computer System", Proc. 1975 Sagamore Conf. on Parallel Processing, Springer-Verlag, 1975

[15] Brown R.M., Neely P.L., "An Image Array Processor for the Investigation of Architectures and Algorithms", Sagamore Comp. Conf. on Parallel Processing, Springer-Verlag, 1975, p. 204

[16] Comtal Company, "Comtal...The Prime Choice. Vision One/20", P.O. Box 5087, Pasadena, California 91107

[17] Oppenheim A.V. and Schafer R.W., "Digital Signal Processing", Prentice-Hall, New Jersey, 1975, 585 pages

[18] Rosenfeld A. and Kak A.C., "Digital Picture Processing", Academic Press, New York, 1976, 457 pages, p.195

[19] Shlien S., "Geometric Correction, Registration, and Resampling of LANDSAT Imagery", Canadian Journal of Remote Sensing (to appear)

[20] Weiman C.F.R., "Highly Parallel Digitized Geometric Transformations without Matrix Multiplication", Proc. of 1976 Int. Conf. on Parallel Processing, IEEE Press, USA, August 24-27, 1976, p.1

[21] Andrews H.C. and Hunt B.R., "Digital Image Restoration", Prentice-Hall, New Jersey, 1977, 238 pages

[22] Hsu H., "Fourier Analysis", Simon and Schuster, New York, 1967

[23] Rothstein J. and Weiman C.F.R., "Parallel and Sequential Specification of a Contex Sensitive Language for Straight Lines on Grids", Computer Graphics and Image Processing, Vol. 5, March 1976, p.106-124

[24] Rothstein J. and Weiman C.F.R., "Pattern Recognition by Retina -Like Devicess", Computer and Information Science Dept., Ohio University, OSU-CISRC-TR-72-8

[25] Peled A. and Liu B., "Digital Signal Processing", John Wiley and Sons, New York, 1976, 304 pages

[26] Groginsky H.L. and Works G.A., "A Pipeline Fast Fourier Transform", IEEE Trans. on Computer, Vol. C-19, 1970, p. 1015

[27] Gen Rad Compagny, "FPE Series Fourier Processing Element", Time/Data Division, 2855 Bowers ave., Santa Clara, CA95051, USA

[28] ESL Incorporated, "Advanced Scientific Array Processor", 495 Java dr., Sunnyvale, CA94086, USA

[29] Floating Point System Inc.,"The age of array processing is here", Portland, OR97223, USA

[30] CSP Inc.,"An Introduction to the MAP Series", 209 Middlessex Turnpike, Burlington, Mas. 01803, USA

[31] Nathan R., "Picture Enhancement for the Moon, Mars and Man", Symposium on Automatic Photointerpretation: Pictorial Pattern Recognition, Thompson Book Comp., USA, 1968, p. 239

Chapter VIII

CONCLUSION

In this thesis, the parallel processor architecture field is examined. In particular, the Multimicroprocessor Array Computing Structure is investigated for image processing applications.

In chapter two, the reasons for using parallel processing are explained, both, from an historical point of view and from a technological point of view. Historically, it is shown that parallelism at the processor level has evolved from a logical continuation of parallelism in the input/output and at the bottlenecks of computers. The motivations behind this evolution are demands for higher performances (throughputs) in domains at the fore-front of progress and, for reliability in industrial and military applications. Technologically, parallel processor architectures have progressed from the possibilities offered by integrated circuits. Now, the use of microprocessors as building blocks makes multiprocessor architectures the next logical step. It simplifies their desing and increases their cost effectiveness. This chapter also outlines the

advantages and disadvantages of such architectures, for example, the programming problems.

In chapter three, the processor architecture domain is divided into four main categories. The advantages and disadvantages of each category are examined and some common problems are analysed. The multiple instruction multiple data streams machine has the most interesting architecture because of its versatility, its adaptability and its processing speed. The major drawbacks of this class are the inter-PE communication problems, the programming difficulties and the cost involved. Inter-PE communication and programming of MIMD machines are still at an early stage of development. However, the advent of microprocessors is producing a major decrease in cost of systems employing multiple processors.

In chapter four, four major existing systems are summarized from a hardware point of view. It is shown how similar problems can be solved differently by various design team of researchers. Three very sophisticated, almost "general purpose", architectures are investigated. Unfortunatly, the cost of such systems is prohibitive. A fourth system, the Flexible Multi-Pipeline-Processor, is presented as a possible solution for image processing tasks, but the number of busses used for inter-PE communication makes the system almost impractical.

In chapter five, the Multi Microprocessor Array Computing Structure based on bit-sliced microprocessors is examined in detail. It is noted that MACS is really a versatile pipeline. All the pipeline stages can be reorganized to suit different applications, being formed by a dynamically modified number of processing elements. However, the MACS lacks calculating power and easy direct communication between all the PEs. Nevertheless, it is found to be a worthwhile architecture for possible application to image processing.

In chapter six, the MACS simulator is described. The necessary steps to create microprograms and interface programs are outlined. All the monitor commands available are examined in an example to illustrate the simulator debugging facilities. Finally, a summary of the simulator package is given, including a description of the individual programs and their interaction with each other.

Chapter seven investigates the potential of MACS for image processing by simulating four typical applications. It is found that the basic architecture of MACS is of interest for image processing, but that the MACS in its present form requires some hardware modifications. Two tasks, the salt-and-pepper filter and the resampling by the MSINC function, are implemented with the present

architecture. However, for the fast Fourier transform and the resampling by the Weiman's method, MACS needs some modifications to its architecture. The modifications required are:

-an increase in microprogram memory

-an increase in external (private) memory

-better methods of addressing the internal registers

Apart from the specific problems encountered in chapter seven, the Multimicrprocessor Array Computing Structure has some general weaknesses. Because of the requirements of the initial radar processing application, the processing elements do not have any conditional branching capabilities. This creates programming difficulties and wastes execution time. The construction of the system bus also causes some programming problems. Only one processing element can use it at the time, and more severly, if more PEs were to use it during the same clock cycle, the errors might not get detected. This is one of the major reasons for the need of a simulation as described in chapter six.

As it presently stands, given these drawbacks, the MACS is not suitable for image processing. However, with extensive modifications, this architecture might ispire some design along similar general principles.

The incapability of the PEs to perform conventional conditional branching drastically limits the possible applications of the MACS. Further investigations regarding this problem are needed if a system similar to the MACS is to be used. Conditional branching is relatively easy to implement hardware-wise, but makes the programmer's task of synchronising the PEs critical. Another area of further study is the problem of busses contention. Two possible solutions are available: the use of a partionned system bus or of a more sophisticated method of inter-PE communication.

In this research, the MACS is assumed to be used as a slave processor. In this context, it would have been interresting to compare it with a standard scientific slave processor (like the AP-120B). Also, more research is certainly needed with regards to multimicroprocessor systems as stand alone systems for image processing.

In general, this thesis establishes a bridge between parallel processor architectures and image processing. It demonstrates how improvements in computer architecture can help a demanding domain like image processing, and conversely how demanding domains stimulate the computer architecture field. It reveals some architectural concepts and classifies possible architectures. It infers that simulation is a valuable tool in processing system design.

More specifically, a recent system based on bit-sliced microprocessor, MACS, is analysed. Finaly, this thesis suggests that modifications to the MACS architecture could lead to the design of a fast image processing system.

As electronics progresses, computer architectures improve, and image processing comes closer and closer to real time applications.

BIBLIOGRAPHY

-Armstrong C.V.W., "PE Column Memory Modules", I.P.Sharp Associates Inc. internal report, IPS-CVWA-CRC-005, Feb. 78

-Durniak A., "VLSI Shakes the Foundations of Computer Architecture", Electronics, May 24, 1979, p. 111

-Enslow P.H., "Multiprocessors and Parallel Processing", Comtre Corporation, John Wiley and Sons Inc., USA, 1974, 340 pages

-Flynn M.J.,"Some Remarks on High Speed Computers", Compcon, San Francisco, Feb. 28-Mar. 3, 1977, IEEE Comp. Soc., p. 98

-Hobbs L.C., "Parallel Processor Systems, Technologies, and Applications", Spartan Books, New York, 1970, 438 pages

-Kochenburger R.J.,"Computer Simulation of Dynamic Systems", Prentice Hall,New Jersey,1972

-Korn G.A., Vichnevetsky R., "Analog/Hybrid Computation and Digital Simulation", Trans. on Computers, Vol. C-25, No. 12, Dec 76, p. 1312

-Krygiel A.J.,"An Implementation of the Hadamard Transform on the STARAN Associative Array Processor", Proc. of 1976 Int. Conf. on Parallel Processing, IEEE Press, USA, August 24-27, 1976,p. 34

-Lipovski G.J., Doty K.L., "Developments and Directions in Computer Architecture", Computer, August 1978, Vol. 11, No.8, p.54

-Margulies A.S., "Array Processing Eases High-Spedd Vector Computation", Digital Design, June 1978, p. 52

-Schumaker D.L., "Small is Beautiful", (french version)Contretemps/ Le Seuil, France, 1978, 316 pages

-Slotnick D.L., "Unconventional Systems", AFIPS, Vol.30, SJCC 1967, p.12

-Strokes R.A., "An Assessment of First Generation Vector Computers", Compcon, San Francisco, Feb. 28-Mar. 3, 1977, IEEE Comp. Soc., p.12