University of South Wales

2053137

THE BUS STRUCTURE FOR A POLYMORPHIC COMPUTER SYSTEM

By

Andrew Kennedy Roach B.Sc.(Hons)

This Thesis is submitted in partial fulfilment for the Degree of M.Phil under the regulations of the CNAA

Department of Mathematics and Computing
The Polytechnic Of Wales

In collaboration with MITEL Corp.

<u>February 1986</u>

Statement

I hereby declare that the work embodied in this Thesis are the results of my own independant investigations unless otherwise stated. This work has not been, nor is it currently being, submitted in consideration for any other degree.

Candidate -- A. K. Roach

Director of Studies -- G. E. Quick

Acknowledgement

I wish to express my appreciation to the following, who provided me with guidance, help and support throughout my research.

Dr. Gerry Quick, Head of School of Computer Studies, West Glamorgan Institute of Higher Education, Swansea, whose initial work this Thesis is a continuation of.

All the members of the Department of Mathematics and Computing at the Polytechnic of Wales; especially Mr.D.J.Green, Dave Eyres, John Ellis, Colin and Chris Bowring; all of whom made my stay a pleasant and eventful one.

Finally my parents, Leslie and Valerie, for without their financial support this research would not have been possible.

ABSTRACT

The Bus Structure for a Polymorphic Computer System

A.K.Roach B.Sc(Hons) M.Phil Thesis

The proposed advances in fifth Generation Computing Systems aim to provide an Intelligent Image to the system user. While such images are software based, written in languages such as Prolog and LISP, much of the proposed hardware architecture has lacked innovation and vision. This Thesis addresses these two important points by providing an insight into bus interaction for various scheduling schemes and system configurations, in order that these unique system architectures may evolve.

This Thesis discusses the issues relavent to the application of cellular computer systems and their projected performance characteristics. The cellular computer system under study is the Group Processor System, which is a TRANSPUTER like computer architecture.

The Group Processor System is simulated, and important results are illustrated in graphical form. These graphs are analysed, and the conclusions drawn are of use to computer architects who wish to design and construct Group Processor Systems. The results may also be of use to those architects wishing to develop TRANSPUTER based computer systems.

As a result of the simulation, a major design fault in the original Group Processor proposal resulted in a severe 'bottle-neck' in input-output processing. This has been greatly improved by the provision of the Terminal Environment Switching System; which is also detailed in this Thesis. The result of this research has yielded a more flexible Group Processor System which may be targetted for applications in Intelligent and Knowledge Based Systems.

The relevance of current architecture is discussed in the context of proposed fifth Generation computing needs.

CONTENTS **PAGE**

Title i

Statement ii

Acknowledgement iii

Abstract iv

Table Of Figures x

CHAPTER

1 INTRODUCTION

1.0 Introduction 1-1
1.1 Computer Architecture 1-1
1.2 The Von-Neumann Architecture 1-2
1.3 Von-Neumann Architectural Implementations 1-3
1.4 Fifth Generation Computer Research 1-7

1.4.1 Japanese Activity 1-7
1.4.2 The Alvey Programme 1-8

1.5 Thesis Plan 1-10
1.6 Scope of Thesis 1-12

2 THE FOUNDATIONS OF COMPUTER ARCHITECTURE

2.0 Introduction 2-1
2.1 Why a new Generation 2-2
2.2 The Semantic Gap 2-3

2.2.1 The Operating System Semantic Gap 2-4
2.2.2 The Programming Environment Gap 2-4
2.2.3 The Storage Semantic Gap 2-5
2.2.4 Consequences of the Sematic Gap 2-5

2.3 The Von-Neumann Architecture 2-6
2.4 Parallel Systems 2-7
2.5 Multi-Processor Systems and Attributes 2-7

2.5.1 Multi-Processor Computer Systems 2-7
2.5.2 Tightly Coupled Computer Systems 2-9
2.5.3 Loosely Coupled Computer Systems 2-9

2.6 Multi Processor Classifications 2-10
2.7 Bounded Parallel Systems 2-14
2.8 Unbounded Parallel Systems 2-14
2.9 Engineered Bounded Systems 2-15

2.9.1 DEC VAX-11/782 2-15
2.9.2 B5000 2-21
2.9.3 iAPX-432 2-22

2.9.3.1 Overview of the iAPX432 2-22
2.9.3.2 Components - Configurations 2-24

2.9.4 Carnegie Mellon Machines 2-26

2.9.4.1 C.mmp 2-26
2.9.4.1 The Hardware 2-27
2.9.4.3 The Software Base 2-29
2.9.4.4 Success and Failure 2-29

2.10 Engineered Unbounded Systems 2-31

2.10.1 Cm* 2-31
2.10.2 MPP 2-35
2.10.3 The Transputer 2-37

2.11 Summary 2-41

3 FIFTH GENERATION NEEDS

3.0 Introduction 3-1
3.1 Background 3-1
3.2 Fifth Generation Systems 3-2
3.3 Application Areas 3-3
3.4 Processor Architecture Exploiting VLSI 3-3
3.5 Needs and Uses 3-4

3.5.1 Industrial Automation 3-5
3.5.2 Office Automation 3-6
3.5.3 Science and Engineering 3-7
3.5.4 Computer Hardware and Software 3-8
3.5.5 Military 3-9
3.5.6 Aerospace 3-9
3.5.7 Retail and Service Industries 3-10
3.5.8 Education 3-11
3.5.9 Health Care 3-12
3.5.10 Leisure 3-13

3.6 Involved Countries 3-13
3.7 Concerns and Goals 3-14

3.7.1 Japan and ICOT 3-14

3.8 Designing the Next Generation 3-15

3.8.1 Exploiting Parallelism 3-16
3.8.2 VLSI: The Solution? 3-16

3.9 The Problems to be Encountered 3-16

3.9.1 Physical Limitations 3-17
3.9.2 Conceptual Limitations 3-17

3.10 Future Computer Architecture 3-18

4 BUS ARBITRATION CONCEPTS

4.0 Background 4-1
4.1 Introduction 4-1
4.2 Current Computer Architecture 4-2
4.3 Bus Arbitration Objectives 4-4
4.4 Current Arbitration Techniques 4-5
4.5 Centralised Arbitration 4-6

4.5.1 Daisy Chain 4-6
4.5.2 Polling 4-8
4.5.3 Independant Requests 4-10

4.6 Distributed Arbitration 4-12

4.6.1 Distributed Daisy Chain 4-14
4.6.2 Distributed Polling 4-14
4.6.3 Distributed Independant Requests 4-17

4.7 The Universal Arbiter 4-17
4.8 Summary 4-24

5 THE GROUP PROCESSOR ARCHITECTURE

5.0 Introduction 5-1
5.1 High Level System Description 5-1
5.2 Peripheral Interface Environment 5-3
5.3 Group Processor Environment 5-6
5.4 Module External Input Output 5-6
5.5 Logical Bus Structure 5-8
5.6 Bus Structures 5-12
5.7 Inter-Module Bus Structure 5-15
5.8 Single Use Environment 5-17
5.9 Multi User Environment 5-17
5.10 The Group Processor Operating System 5-19
5.11 Distributed Control 5-20
5.12 High Level Operating System Representation 5-23
5.13 Group Processor System Summary 5-23

6 THE SIMULATION ENVIRONMENT

6.0 Introduction 6-1
6.1 The Modelling Approach 6-1

6.1.1 Synthetic Benchmarks 6-2
6.1.2 Live Benchmarks 6-2
6.1.3 Simulation 6-3
6.1.4 Mathematical Modelling 6-3

6.2 The Case For Simulation 6-3
6.3 Computer Structure, Resource Application 6-4

6.3.1 What is a resource? 6-5
6.3.2 Definitions 6-5

6.4 Simulation Architecture 6-6

6.4.1 System Architecture Components 6-6

6.4.1.1 Buses 6-7
6.4.1.2 Bus Arbiters 6-7
6.4.1.3 Cells 6-7
6.4.1.4 Modules 6-7

6.4.2 The Group Processor 6-7
6.4.3 Bus Interconnection Schemes 6-8

6.5 Bus Arbitration 6-8
6.6 Bus Requests 6-9
6.7 Actioning Bus Requests 6-9
6.8 The Simulator Program 6-9

6.8.1 System Parameters 6-10
6.8.2 Major Parameters 6-10
6.8.3 Tuning Parameters 6-10

6.9 The Simulation Environment 6-11

6.9.1 The Job Scheduler 6-11
6.9.2 The Job Server 6-13
6.9.3 System Loading 6-14

6.10 Range of Results 6-14

6.10.1 Variation of Physical Constants 6-14
6.10.2 Variation of Cell Numbers 6-15
6.10.3 Variation of Module Numbers 6-15
6.10.4 Variation of Inter-module bus 6-15
6.10.5 Variation of Intra-module bus 6-15
6.10.6 Variation of Soft Constants 6-16
6.10.7 Bus Request Rate 6-16
6.10.8 Ratio of Jobs 6-16
6.10.9 Time Required as Bus Master 6-16
6.10.10 Message Length 6-17

6.11 Simulation Goals 6-17
6.12 Limiting Factors 6-17
6.13 Conclusion 6-18

7 RESULTS AND IMPLICATIONS

7.0 Introduction 7-1
7.1 Table of Results 7-2
7.2 Basic Group Processor System 7-9
7.3 Extended Group Processor System 7-11
7.4 Operating System Constraints 7-12
7.5 Off-loading Factor 7-15
7.6 Segmemted Input/Output 7-16
7.7 Dedicated Systems 7-17
7.8 Closer Analysis 7-18

7.8.1 Effects of Bus Contention 7-18
7.8.2 Inter-Cell Communication 7-22

7.9 Software Considerations 7-28
7.10 Conclusions 7-35

8 GROUP PROCESSOR ARCHITECTURAL ENHANCEMENTS

8.0 Introduction 8-1
8.1 Group Processor System Problems 8-1
8.2 T.E.S.S. Outline 8-2
8.3 T.E.S.S. Objectives 8-4
8.4 T.E.S.S. Operations 8-8
8.5 Module/Channel Interface 8-12
8.6 Crossbar Operation 8-14
8.7 Bus Arbitration 8-15
8.8 Summary 8-18

9 CONCLUSION AND FURTHER RESEARCH

9.0 Introduction 9-1
9.1 Research Initatives 9-1
9.2 Computer Architecture 9-3
9.3 The Group Processor System 9-4
9.4 T.E.S.S. 9-6
9.5 Future Research 9-7
9.6 Summary 9-9

APPENDIX ONE: Simulator Graphical Output A1-1

APPENDIX TWO: Simulator Program Listing A2-1

Figure No.	Figure Title	Page No.
2.1a	Single User Global Memory	2-8
2.1b	Single user Global + Local Memory	2-8
2.2	Single user Loosely Coupled	2-11
2.3	Multi-user Multi-processor	2-12
2.4	VAX-11/782 Connections	2-16
2.5	VAX Architectures	2-18
2.6	Dual VAX Configuration	2-20
2.7	4 Processor iAPX-432 Configuration	2-25
2.8	C.mmp	2-28
2.9	5 Cluster Cm*	2-32
2.10	Cm* individual cluster	2-33
2.11	MPP	2-36
2.12	The Transputer	2-39
4.1	Simple Parallel System	4-3
4.2	Centralised Daisy Chain	4-7
4.3	Centralised Polling	4-9
4.4	Centralised Independant Requests	4-11
4.5	Bus Arbiter Types	4-13
4.6	Distributed Daisy Chain	4-15
4.7	Distributed Polling	4-16
4.8	Distributed Independant Requests	4-18
4.9	Hierarchical Arbitration	4-20
4.10	Linear Priority Arbitration	4-21
4.11	Fifth Generation Arbitration	4-23
5.1	Functional Composition of G.P.S	5-2
5.2	Group Processor Module	5-4
5.3	Module Component Architecture	5-5
5.4	Inter-module Bus Structure	5-7
5.5	Data Bus Architecture	5-9
5.6	Module Coupling Architecture	5-11
5.7	Single User System	5-13
5.8	Multi User System	5-14
5.9	Multi User Environment	5-18
5.10	Distributed Operating System	5-22
6.1	Simulator Schematic	6-12
7.1	Table of Results 1	7-4
7.2	Table of Results 2	7-5
7.3	Table of Results 3	7-6
7.4	Table of Results 4	7-7
7.5	Table of Results 5	7-8
8.1	Multi User System	8-3
8.2	T.E.S.S Architecture Outline	8-5
8.3	Terminal Alternatives	8-7
8.4	Ported Memory Concepts	8-9
8.5	Four Terminal/Crossbar Module Interconnection	8-11
8.6	Module/Crossbar Interconnection Network	8-13
8.7	Crossbar Addressing Mechanism	8-16
8.8	Crossbar Functional Diagram	8-12

CHAPTER ONE

CHAPTER ONE

INTRODUCTION

1.0 Introduction

There is a general consensus among the computer science fraternity that the 1990's will see the end of the traditional Von-Neumann architecture machine, and that a new generation of general purpose computing machines will evolve.

Most computer architects also see a need for a new generation of system design. Unfortunately, few truly new inititives are forthcoming, as most 'new' designs are based on a limited extension of the classical Von-Neumann computing machine. The research set out in this Thesis has set as its aim a divergence from classical views on computer architecture, to views which may seem radical. This research sees as its starting point the abandonment of classical bounded multi processor computing systems, consisting of say 64 processors. Only with seemingly unbounded systems can we say that computer architecture has developed a new Fifth Generation of computing machinery.

1.1 Computer Architecture

Defining what is meant by Computer Architecture is not a simple task. Computer Architecture is not restricted to the sole aspects of hardware. Building black-boxes from

registers, memory devices etc., is certainly part of the process, so is the interconnection of these boxes via buses, switches and controllers. A blend of hardware and software features which make the machine operate must also be included. Computer architecture may be defined as the design of the integrated system which provides a useful tool to the programmer. Computer architecture may be defined to mean:-

> The internal workings of the black-boxes which are the main components of the system and the means of interconnecting these boxes, their parallel activites and cooperation.

1.2 The Von-Neumann Architecture

The first major architecture was proposed by John Von-Neumann et al. in their 1946 paper, 'Preliminary Discussions of the Logic Design of an Electronic Computing Instrument' [5]. Even with the advent of modern machines [4] most computer architectures bear the mark of this design. Therefore before embarking on any description of the generations of computer architecture, a brief description of the Von-Neumann architectural model is essential.

The Von-Neumann architectural model was conceived for a specific purpose, that of providing a simple stored program execution mechanism to carry out the computations for the solutions of differential equations. The architecture can be said to have the following properties:-

1) A single, sequentially addressed memory. The program and its associated data are stored in a single memory, the memory being referenced with sequential (0, 1, 2, 3,...) addresses.

2) A linear memory. The memory is one dimensional, that is, it has the appearance of a vector of words.

3) No explicit distinction between instructions and data. Instructions and data are distinguished implicitly by the operations directed toward them.

4) Meaning is not part of the data. There is nothing that explicitly distinguishes a set of bits representing a floating-point number from those representing a character string. The meaning of the data is assigned by the program logic.

1.3 Von-Neumann Architectural Implementations

Computer architecture has developed much in the last 30 years, from the Manchester Mark-1 to Seymour Cray's Cray X-MP/2 [13].

The Mark 1 is of historical importance as it was the world's first stored-program computer. The machine marked the beginning of a new technological era. In todays terms the machine posessed the following hardware features:

1) A 32-bit word length.

2) Serial binary arithmetic (2's complement).

3) A main store of 32 words (expandable to 8192).

The main emphasis of the project was to prove the practicability of the Williams Tube for realising the stored program concept and as a result the logic was kept as simple as possible. The subtractor was the only arithmetic element included, as it can perform complements and additions without modification.

The next major step was the prototype construction of the Atlas machines at Manchester University in the 1950's [2]. Atlas-1 and Atlas-2 were eventually produced by the Ferranti Corp. Atlas featured multiple index registers, interrupt processing of I/O devices. Two original features of Atlas, namely a one-level storage and extracode have been copied in many other machines. The one level store is common to most time-shared or multi programmed computers.

Significant features of the Atlas system were:-

1) Provision of a virtual address space greater than the physical address space.

2) Implementation of a one level store using a mixture of core and drum memories.

3) Interrupt system and method of peripheral control.

4) Realisation at the design stage that there would be a complex operating system and provision in the hardware of specific features needed to assist such an operating system.

Computer systems have usually been designed via the 'hardware' route. Subsequent to design, these systems have been handed over to a systems programming team for the development of a package to facilitate the use of the hardware. However the Burroughs B5000 [3] was designed from top to bottom as a total hardware/software system.

The B5000 achieves a unique physical and operational modularity through the use of switches which logically function as crossbar switches. The B5000 was designed as an integrated hardware/software system which offered multiprocessing and parallel processing.

The Digital Equipment Corporation's PDP-8 is of importance as it was the first true minicomputer. The PDP-8 was a single address 8 bit computer. It was the first of a family called the 'OMNIBUS' machines. Like its predecesor, the PDP-5, the PDP-8 was a single address 12-bit [10] computer designed for 'task' environments with a minimum of arithmetic computing and and small memory requirements, i.e process control.

The early constraints placed on computer architects, created computers with what we now regard as faults or weaknesses, namely:-

1) Limited addressing capability
2) Few registers
3) No hardware stack facilities
4) Limited priority interrupt structures
5) No byte string handling

6) No ROM facilities

7) Little I/O processing

8) No simple hardware upgrade

9) High programming costs (All users use machine code)

The DEC PDP-11 was designed with the above in mind, and successfully [9] overcame these limitations. This was due mainly to the fact that semiconductor technology became available to solve the problems at low cost.

The VAX-11/780 computer system is the first implementation of the [11] VAX-11 architecture, a Virtual Address eXtension to the PDP-11 architecture. The most distinctive feature of the VAX is the extension of the virtual address from 16 bits, as on the PDP-11, to 32 bits; giving an address space of some 4.3 gigabytes. Since maximum PDP-11 compatibility was a design objective, the VAX includes a compatibility mode which provides the basic PDP-11 instruction set, without the privileged instructions.

<u>The IBM System/360 and System/370</u>

The System 360 was the first planned computer family to cover a range [1] of cost and performance. The 360 predecessor, the 7090, ran into problems later encounterd by the PDP-8, namely limited growth potential. Rather than 'fiddle' with the architecture IBM planned a family of processors with growth potential for the future. The initial family plan called for a wide range of cost and performance implementations, microcode being used to provide emulation support for prior systems.

The motivation to extend the 360 architecture came from two main sources:-

1) The experience of the 360 achitecture has identified a number of bottlenecks and limitations in the efficiency of system use has pointed out areas where additional machines were needed.

2) The lowering of the cost of technology made it economically possible to include functions that did not appear justified in the original 360 architecture.

The most interesting aspect of the 360-370 design is achieving a performance range and a primary memory size range in excess of 100:1.Thus the user is given a very large range of configuration alternatives.

1.4 Fifth Generation Computer Research

Japan's capability for producing high quality electronic products is well known. It therefore came as a shock when in 1981 the Japanese announced to the World a programme of research into Fifth Generation computing systems. This was the responsibility of The Japanese Ministry of International Trade and Industry, MITI [8].

1.4.1 Japanese Activity

In 1982 Supercomputers were an American exclusive. Today, Japanese firms are offering machines that challenge, and in some ways exceed, the performance of those American machines. This has been due mainly to the fact that MITI

has decided that Japan must learn to innovate, not just copy and improve on existing technology.

MITI's development plans for fifth Generation computers started in 1982. The budget for the years 1982-1984 being some 10 billion Yen. The Institute for New Generation Computer Technology (ICOT) has been created to spearhead Japanese efforts in the field, and has been succesful in designing and building an Inference Machine and a Database Machine.

However, NTT is the only Japanese company trying to develop a true [12] parallel processor. Two types of dataflow are under study; one is a highly parallel array processor for scientific calculations, the other is an architecture designed to apply data-flow techniques to list processing.

1.4.2 The Alvey Programme

The Japanese initative produced a number of responses around the word. The most notable was The UK's Alvey Committee, which producd a number of goals for Britan's involvements in fifth Generation computing systems. The Alvey Programme was set up as a result of the Alvey Committee report, (A Programme for Advanced Information Technology), in 1982. The Programme aims to mobilise the United Kingdom's strengths in Information Technology, (IT), in order to improve the UK's competitive position in the World's IT market.

The research programme is a collaborative effort between Government, industry, academic and commercial research units. The Programme combines projects in four main technology areas.

The four enabling areas are:-

1) VLSI

The requirement for massively parallel systems demands a VLSI approach to building systems. VLSI gives the capability of interconnecting the very large number of logic elements required for data and signal processing.

2) Software Engineering

Fifth generation computing systems will tend to be more complex than present day systems. This will result in a longer time delay in generating and maintaining proposed fifth generation programs. Therefore, Software Engineering is aimed at improving the efficiency of the specification, generation and maintenance of the program instructions for IT systems.

3) Intelligent Knowledge Based Systems

One major area of activity is the design of intelligent machine/software systems. These systems should be adaptive systems capable of learning. The object is to produce inference systems that can be incorporated in education, medicine, military, etc.

4) The Man-Machine Interface

The proposals of 'Alvey' clearly imply that computing systems will be applied to new application areas, possibly with new methods for input and output. The wider aspects of the involvement include psychological aspects of using complex systems.

Specifically, this section covers visual, speech, touch input-output devices and the better understanding of the nature of communication between the user and the machine.

Following Japan's initative, the United States and Europe have started various research programmes into fifth Generation design. Most countries are in an early stage of development and are desperately trying to catch up on Japan's early lead. One question remains unanswered; are the claimed fifth Generation systems truly advances on the older technologies, or simply refinements of proven systems?

1.5 Thesis Plan

Following a brief introduction to Computer Architecture and the area of the proposed fifth Generation computing in chapter 1, chapter 2 presents a critique of architectures from the previous generations presenting a new possible classification scheme for the architectural generations, independant of technology. The chapter also examines the reasons why the Von-Neumann architecture is in need of replacement, and examines some of the machines which have tried to overcome its limitations.

Chapter 3 looks at the new application requirements of a fifth Generation computer system, and criticises some of the so called fifth Generation machines.

Chapter 4 examines the problems of bus contention brought about by large numbers of processors and offers some solutions to this problem. This chapter presents bus interconnection patterns in the context of maximising bus traffic in fully distributed systems.

In chapter 5, an introduction to a Polymorphic, Space Sharing [6] Computer System, called the Group Processor System, is given. Details of the functions performed within the components of the Group Processor's homogenous architecture and the interaction within the organisational structure are presented. The design claims to overcome the two basic problems isolated by the Data Flow Group at U.C. Irvine [7] namely data localisation and concurrent bus access. This architecture is able to emulate a data driven machine.

Chapter 6 presents an introduction to the various methods of system performance measurement and gives a detailed simulation environment for hierarchical system interconnection. Moreover, the simulator is concerned with the dynamic environment generated by the execution Group Processor System environment.

Chapter 7 examines results from the simulator for various system configurations. The simulator provides a 'window' on the program environment within the Group

Processor System, this yields important data used to optimise the Group Processor System architecture. This chapter highlights one major weakness of Quick's original design concept for the Group Processor System, that of real time input/output under heavy loading.

The proposed architecture presented by Quick [6] provided detail into the working architecture of the basic Group Processor System. However, many areas of the bus system were left for refinement, and these are studied in this Thesis. The results of the study have enabled the next chapter, chapter 8, to propose many important system tuning features to be employed in engineering the Group Processor System. The chapter proposes a new design for the input/output system of the Group Processor System.

The final chapter, chapter 9, contains conclusions drawn from this research. Specifically, the areas of computer architecture, fifth Generation Computer Architecture, system simulation and system performance are discussed.

Each chapter is complete with references at the end of the chapter. The references are presented in the UNIX format.

1.6 Scope of the Thesis.

This research continues the abandonment of the Von-Neumann architecture, for an architecture more radical in concept. Polymorphic systems provide the necessary degree

of reliability needed for real time control systems coupled with a high degree of resource utilisation. The major advantages of the Polymorphic 'Group Processor System' is its massive parallelism. While the Group Processor System seems to offer a solution to current problems in IKBS, some potential problems arise when 'engineering' the system. These areas are addressed in this Thesis by simulating the bus interaction for various bus scheduling schemes and system configurations. The results drawn from this Thesis provide the information needed to proceed with the next stage of the Group Processor System, namely circuit and software design.

References - Chapter 1

1. IBM Corporation, "A Guide to the IBM System/370", 5th Edition 1976.

2. SUMNER, F., HALEY, G., AND CHEN, E., "The Central Control Unit of th ATLAS Computer", PROC IFIP 1962.

3. LONERGAN, W., AND KING, P., "Design of the B5000", Datamation V7 N5 1961.

4. SIEWIOREK, D., BELL, G., AND NEWELL, A., "Computer Structure: Principles and Examples", McGraw-Hill 1982

5. BURKS, A.W., GOLDSTINE, H.H., AND VON-NEUMANN,J., "Preliminary Discussion of the Logic Design of an Electronic Computing Instrument", Pt 1. No.1, Princeton N.J. 1946.

6. QUICK, G.E., "The Group Processor Approach Computer Architecture", Ph.D Thesis, UC Swansea 1982.

7. GOSTELOW, K.P. AND THOMAS, R.E., "Performance of a Dataflow Computer.," UC Irvine TR 127a.

8. MOTO-OKA, T., "Fifth Generation Systems," North Holland 1982.

9. DIGITAL, "PDP-11 Hardware Handbook", 1977.

10 BELL, G., MUDGE, J., AND McNAMARA, J., "A DEC View of Hardware System Design", Digital Press 1978.

11. DIGITAL, "VAX-11/780 Hardware Handbook", 1982-3.

12. NATO Conference on Relational Database Architecture, Les Arcs, France 1985.

13. LUBEK, O., MOOR, J. AND MENDEX, R., "A Benchmark of Three Supercomputers: Fujitsu VP-200, Hitachi S810/20 and Cray X-MP/2", IEEE Computer 1984

CHAPTER TWO

CHAPTER TWO

THE FOUNDATIONS OF COMPUTER ARCHITECTURE

2.0 Introduction

Many of the designs for computing systems have been centred on a typical system architecture comprising of a central processor (or some finite multiple up to say 16), random access memory, input-output processors, and some backup storage such as magnetic disk or tape. Highly reliable systems, such as the Tandem [17] have been an extension to these systems by incorporating some form of redundancy in one or more parts of the design.

The more "classical" designs are based on Von-Neumann's architecture, which have received criticism from some researchers [3,21,7]. A number of variations on the Von-Neumann architecture has resulted in many multi-processor and multi-computer organisations [1,4,24,14,19,30,32,33]. In developing an alternative design, researchers have proposed several alternative architectures [2,8,13,26,28]. These architectures can be best described as non sequential; they have tried to deviate from the classical Von-Neumann machine.

2.1 Why a new generation of computers?

With few exceptions, there have been no advances in computer architecture of current systems since the 1950's.

An argument against this is the introduction of such concepts as microprogramming, VLSI, pipelining and cache memories. However, these do not represent architectural concepts, but merely advances in the implementation of particular current architectures.

In comparing the architectures of the most widely used machines;

e.g. IBM S/370 S/34, DEC PDP/11, VAX etc

to the EDSAC and EDVAC, the first electronic stored program computers, all the significant differences will be found to have originated in the 1960's. Which were:-

1) Index registers
2) General purpose registers
3) Floating point data representation
4) Indirect addressing
5) Interrupts
6) Asynchronous input-output
7) Virtual storage
8) Multi-tasking

Although current systems differ significantly from their predecessors in terms of cost, speed, reliability, internal organisation and circuit technology, the computer

architecture of most current systems has not advanced beyond the concepts of the 1950's.

Given this state of affairs the following must be asked:-

1) Are the architectures of the 40's and 50's the optimal ones for today?
2) Is not todays world different, measured in cost of logic, speed, sophistication of computer application and magnitude of the software problem, and that changes in computer architecture are needed?

If the above points are valid, we must put forward some evidence why a new approach is necessary.

2.2 The Semantic Gap

Most of the shortcomings in todays machines are due to the 'Semantic Gap'. The Semantic Gap was originally defined to be a measure of the difference between concepts in current high level languages and the underlying concepts in computer architecture [20].

Todays systems have an uncomfortable semantic gap in that objects and operations reflected in their architecture are rarely closely related to the objects and operations provided by the programming languages. In broadening the definition of the semantic gap, it may be said that there is a large gap in semantics between programming environments

and their representation of the program concepts at the architectural level.

2.2.1 The Operating System Semantic Gap

The operating system is an integral part of most computing systems. In general the operating system has four purposes:-

1) Providing utility services to other programs, such as storage allocation for the execution of large programs.

2) Shielding programs from such items as interrupts, machine interfaces etc. for software portability.

3) Providing, at varying levels of sophistication, a 'virtual machine' concept.

4) Creating and enforcing system managemant policies.

A case can be built for a gap between the operating system concepts and the underlying machine. For instance, many operating systems designers recognise that the working set model is crucial to managing a storage hierarchy in a close to optimal manner. Although instrumentation required to do this has been identified it exists in no commercial architectures.

2.2.2 The Programming Environment Semantic Gap

Evidence can be found of a large gap between fundamental notions of programming and most computer

architectures. For instance, such concepts as modularity, abstract data types [22], information hiding, and monitors are important in the design of large software systems, but support for these concepts is missing from today's architectures.

2.2.3 The Storage Semantic Gap

This gap is more difficult to see as it represents a gap that does not exist. The operating system architects have obscured the gap by falling into it. The issue here is the lack of a uniform concept of storage. The programmer is faced with a visable hierarchy of registers, stacks, RAM, tapes etc. Each medium has a different addressing mechanism, allocation machanism etc. It is for the wrong reason that the gap does not exist. Rather than presenting the high level language programmer with a uniform notion of storage, one is presented with a number of inconsistent concepts which are technology dependant.

2.2.4 Consequences of the Semantic Gap

The semantic gap is a significant contributor to software unreliability in the sense that a large set of programming errors that could be theoretically prevented or detected by the computing system are not prevented or even detected in current systems, i.e array bound checks, references to undefined or unset variables.

2.3 The Von-Neumann Architecture

The basic reason for the semantic gap in current systems is that their architectures do not significantly differ from the Von-Neumann model developed in the 1940's. We may class all current machines as being of the Von-Neumann type.

Although the von Neumann architecture was a reasonable architecture for the first stored-program computer, it is alien to the execution of programs writen in high level languages.

In contrast to the four main characteristics of a Von-Neumann architecture given in Chapter-1, high level languages have the following characteristics:-

1) Storage is presented as a set of discrete named variables. There is no concept of one variable being 'next' to another. There is no reason to believe that variables in one subroutine are located in the same storage device as the variables in another. In short, the concept of a single sequential storage bears little resemblance to the concept of storage in programming languages.

2) Programming languages deal with multi-dimensional, not just linear, data types.

3) In programming languages there is a sharp distinction between data and instructions.

2.4 Parallel Systems

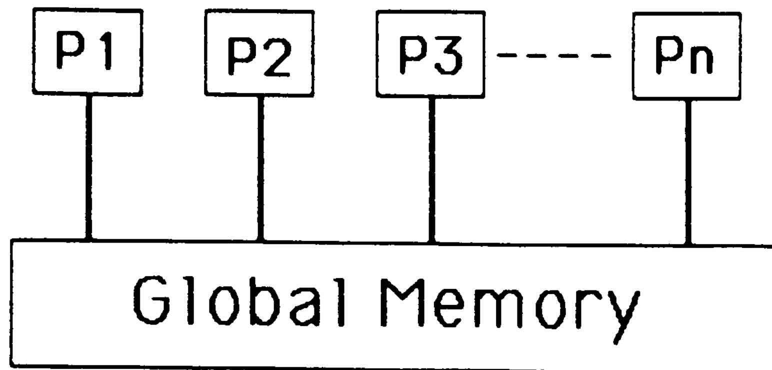
Some of the architectures which have moved away from the Von-Neumann mode are multi-processor and multi-computer systems. However, there is some confusion as to the definition of such term. Some clarification and extension of notation is first required.

2.5 Multi-Processor Systems And Attributes

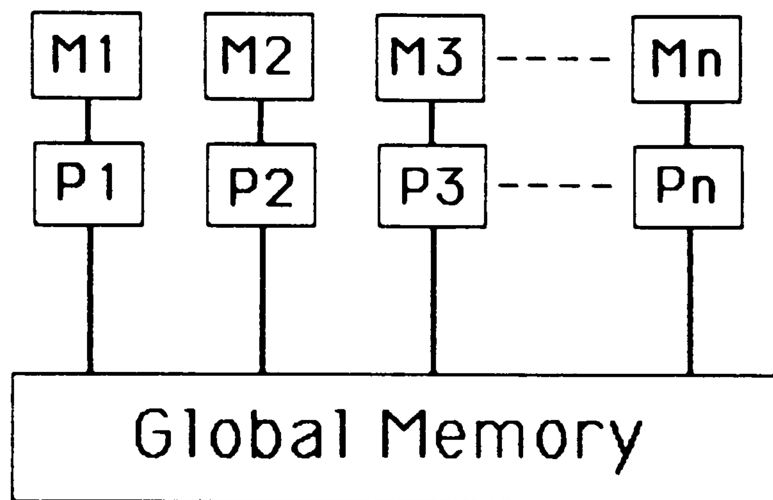
One of the fundamental problems with traditional multi-processor configurations is the interconnection of memories and processors, and also their interconection to the outside world. This section analyses the attributes of multi-processor computer systems.

2.5.1 Multi-Procssor Computer Systems

This section discusses the clasification of various multi processor computer system schemes available to the computer architect, with reference to reliability. A "black box" approach, synonymous with the integrated circuit will be used to represent the micro partitioning of systems, e.g. processor, memory, etc. Macro partitioning is used to represent a stand alone general purpose, i.e. non specialised, computer system. The figures in this section, i.e. figures 2.1. and 2.2., are single user systems which are either a host mainframe or user terminal.



(a) Totally Shared Memory



(b) Limited Shared Memory

Figure 2.1 Single User Tightly Coupled Computer System

2.5.2 Tightly Coupled Computer Systems

A tightly coupled Computer System [17], is a multiple processor, shared memory, computer system. Figure 2.1.a. shows a "totally shared memory" structure, while figure 2.1.b. shows a "limited shared memory" structure.

The major advantage of the totally shared memory is its inherent flexibility. This is best illustrated by considering a processor, say P2, as failed. From figure 2.1.a., any other processor may address each others memory space during a recovery process. By comparison, recovery is difficult in figure 2.1.b, where the recovery process has to access the local memory, i.e. M2. However, the provision of local memory does provide a closed process environment, resulting in the confinement of processing errors to the local memory. Local memory provides an additional speedup [33] in computation, because contention for shared memory access is reduced.

2.5.3 Loosely Coupled Computer Systems

Loosely Coupled Systems [10] are multiple computer systems in which there is no shared memory, and all inter processor communication takes place through input-output channels.

Loosely Coupled Systems have more structure than Tightly Coupled Systems because their inter processor communication is intelligent. This enables intelligent inter process communication between communicating processes, at

the machine level, e.g. through input-output channels. This is not possible in Tightly Coupled Computer Systems, as any processor may access any location in the shared memory, e.g. when a faulty processor writes to an output area of another processor, producing systemic process corruption.

2.6 Multi-Processor Classifications

Multi-Processor Classification has enabled a classification scheme to represent inherent qualities of two dissimilar system structures. When considering the execution of a program, as an execution of communicating processes in the single user systems of figure 2.1. and 2.2., the Tightly Coupled Computer Sytems offers a more reliable programming environment through its closed, i.e. local, memory architecture.

In a multi user interactive system, employing a multi-processor architecture, the input-output to the user terminals is performed by a fast, uni-processor based, communication processor. This type of system is shown in figure 2.3. While the execution environment of figure 2.3 is more reliable than the conventional uniprocessor systems, the reliability of the front-end communication processor is a single point of failure. With this in mind, the ideal multi-user, multi-processor machine provides a process execution environment which is loosely coupled, together with a loosely coupled input-output to the system users.

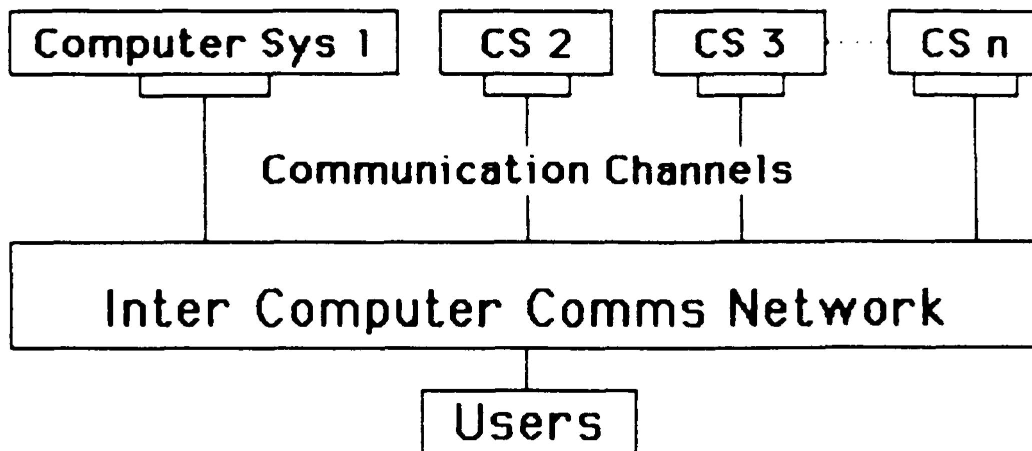


Figure 2.2 Single User
Loosely Coupled System

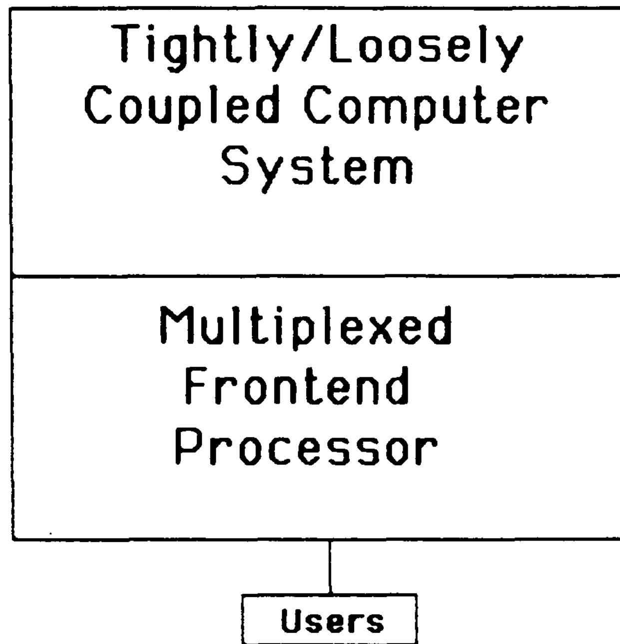


Figure 2.3 Multi-User Multi-Processor Computer System

Quick proposes a change in the system classifcation scheme which includes [12] input-output architecture to the execution environment. The Tightly/Loosely Coupled Computer Systems shown here, together with Flynn's classification [11] e.g. Multi Instruction Multi Data - MIMD, require extension, or clarification.

The extended classification is:-

1 (a) Tightly Coupled, Single I/O (i.e. Single channel I/O)

(b) Tightly Coupled, Parallel I/O (i.e. Multiple channel I/O)

2 (a) Loosely Coupled, Single I/O (i.e. Single channel I/O)

(b) Loosely Coupled, Parallel I/O (i.e. Multiple channel I/O)

Relating the above classifications to fifth generation requirement; the system structure would have the input-output equivalent of 1(b) or 2(b).

The proposals for fifth generation architectures requires a move away from the multiplexed front-end processor. The multiplexed input-output channel does not offer the speed required for the fifth generation human interface. Dedicated input-output channels seem to be the only mechanism capable of matching applications to architectural requirement.

2.7 Bounded Parallel Systems

A machine architecture is bounded if there are up to, say 64 processors where the architecture is defined in terms of maximum system configuration. That is, the maximum number of processors that can be integrated into the system is 64.

The limiting factors here are based on technological limitations such as pinout numbers on integrated circuits. Bounded systems tend to be cheaper designs than more flexible unbounded systems.

2.8 Unbounded Parallel Systems

A machine architecture is unbounded if it is designed with maximum flexibility and extensibility as a fundamental system requirement. Such systems should be capable of supporting in excess of 10,000 processors, and ideally millions of processors. In reality; it is difficult to design systems that are truly unbounded. The physical interconnection of the main components require a fixed number of hardwired connections. An unbounded interconnection scheme demands maximum flexibility in the interconnection, and hence requires soft and not hard connections. It can be said that they approach the unbounded state.

2.9 Engineered Bounded Systems

Four architectures which have moved away from the classical uni-processor architecture.

1) The VAX-11/782
2) The Burroughs B5000
3) The iAPX-432
4) C.mmp

2.9.1 Multi-processor Configurations of the VAX-11/780

The simplest multi-processor configuration of the VAX family is the [35] VAX-11/782, a tightly coupled asymmetrical multi-processor. The 782 is based on two 780 processors using the MA780 shared memory subsystem (Figure 2.4).

At the centre of all VAX multi-processors is the multiport memory. This enables up to four VAX processors to share a bank of memory. This feature allows VAX users to develop multi computer configurations for very high throughput or enhanced availability.

Applications built around multiple cooperating processes can be configured to run on multi-processor systems with no programming modification. Processes in shared memory can be moved from one procecssor to another with complete transparency to the programs involved.

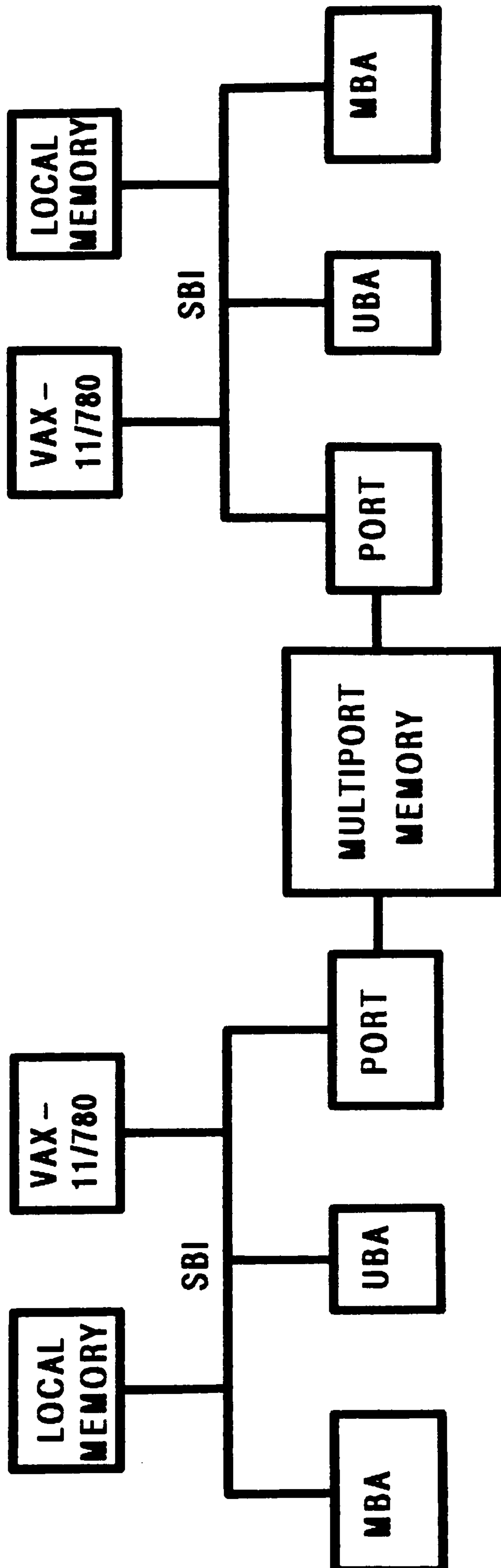


Figure 2.4

An interesting point to note is that each processor in the multiport system operates independantly using its own copy of the operating system stored in its local memory. This 'local operating system' is discussed further in chapter 5.

Port Arbitration

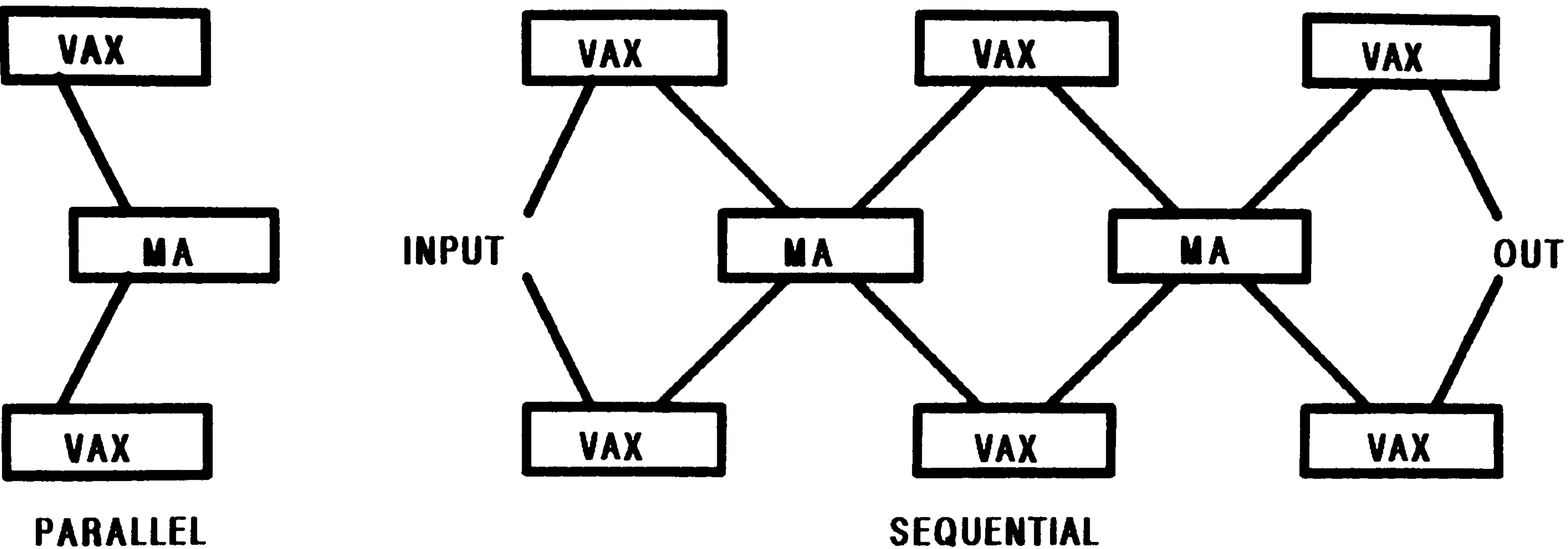
The high throughput of VAX multi-processor configurations is due to each port having a buffer for commands and data. Each port is served on a demand basis, that is, first-in first-served. No time is wasted in polling inactive ports. A serving algorithm guarantees that no port waits more than three memory cycles to gain access to shared memory.

A problem associated with multiport memory is that of one processor trying to read a location at the same time another is trying to modify it. The VAX eliminated this contention by locking out the second processor until the first has completed the transaction.

Parallel and Sequential Procesing

The multiport memory of the VAX can enhance system performance via two configurations:-

1) Parallel or 2) Pipeline processing.



PARALLEL AND SEQUENTIAL VAX
CONFIGURATIONS

Figure 2.5

In the parallel mode; two or more processors divide the task between them, allowing processors to pool resources. Pipelining can increase total system throughput by allowing instantaneous data exchange between processors that are handling the sequential parts of an application. The following figure (Fig 2.5) illustrates possible configurations.

Although the VAX can be configured to run as a multi-processor, its design is that of a uniprocessor. This leads to several major problems on throughput. In the tightly coupled system, processors execute the same copy of the operating system and share the same data structures. Asymmetric processors cannot execute the entire operating system code at the same time. In a dual processor configuration, figure 2.6, all kernal mode and interrupt code is executed by the primary processor. Also all Input/Output is conducted by the primary processor. These design features can lead to the second processor being almost inactive due to the primary being input-output bound. Therefore every time the secondary processor generates a page fault, the primary processor must halt its current activity and service the attached processor.

Due to the above problems the 782 is only 'in its element' when handling primarily compute bound jobs. This leaves the primary processor free to handle all Input-Output at a reasonable throughput.

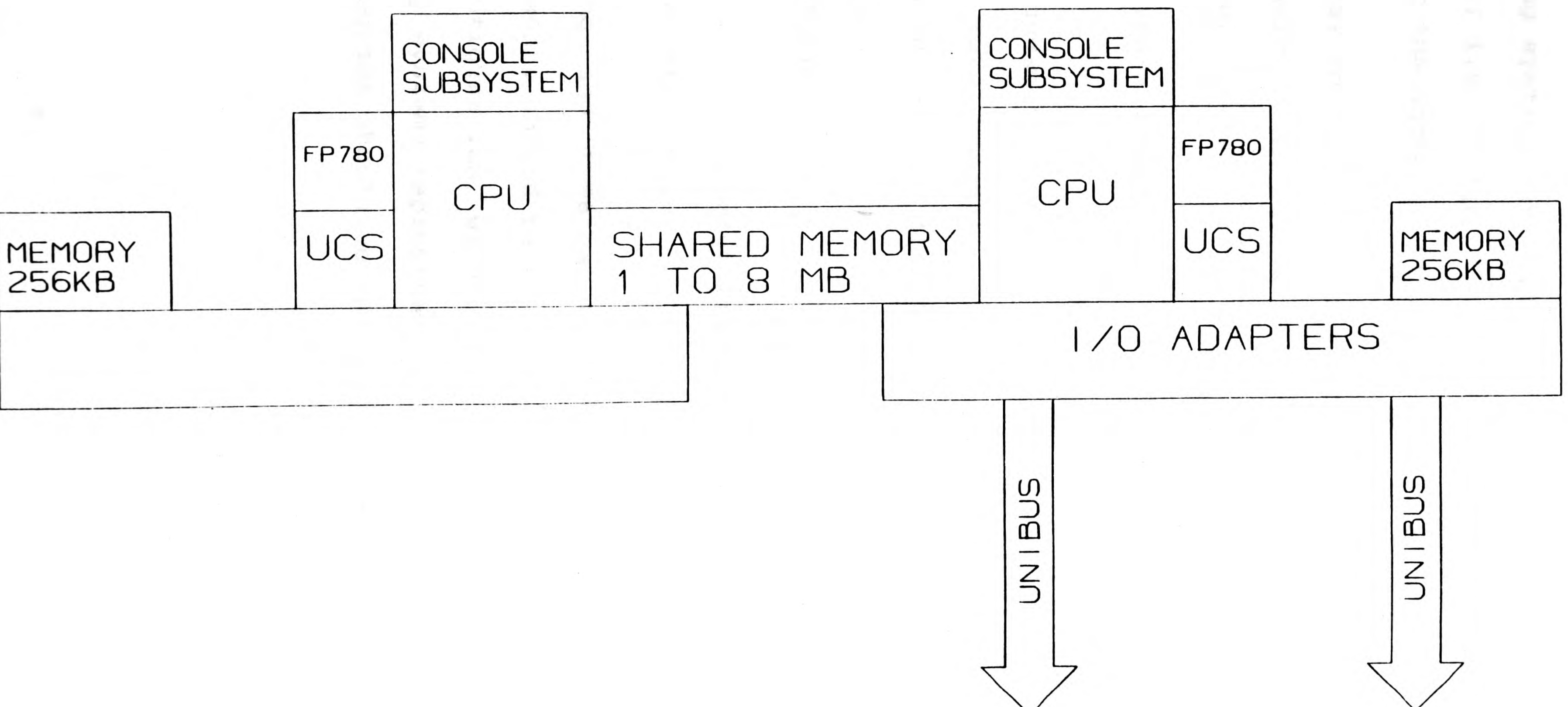


Figure 2.6

2.9.2 The Design of the Burroughs B5000

Computing systems have usually been designed via the hardware route; i.e. design the physical machine first and then implement the software at a later date.

A contrast to this was the design of the B5000. From the initial design, the system was thought of as a total hardware-software system.

Design Objectives and Criteria

The fundamental objective of the system was the reduction of total problem through-put time. A second objective, and in terms of this Thesis the more significant, was the provision of facilities to change both programs and system configuration.

Early in the design phase of the system major principles were established:

1) Multi-tasking and true parallel processing, requiring multiple processors should be provided.
2) System reconfiguration, within reasonable limits, should not require any systems reprogramming.
3) Data and programs should be independant of location.

2.9.3 The Intel iAPX-432. An Advanced Microprocessor

The iAPX-432 microprocessor was designed with the aim of reducing the software development problems created by the Semantic Gap.

With this in mind likely application areas for the 432 are:

1) Low volume applications where the programming investment is high.
2) High volume applications where programming is more than a one time occurence.
3) Those areas with a high degree of concurrency.

It is this last area which is of significance to this Thesis. Other aims of the 432 design were incremental system performance, or the ability to tune the performance of the system by adding or subtracting processors without the need to modify software. The final two aims of the system were the ability of the system to provide 'shadow' redundant processors to check system integrity and support for the ADA language at a fairly low level.

2.9.3.1 Overview of the iAPX-432

Many of the architectural attributes of the 432 are similar to that of the SWARD architecture [36], the major difference being that the 432 is not a tagged-storage machine.

The fundamental concept of the architecture is that of an 'object' [25]. An object is a collection of related information which, with a set of applicable operators, forms an abstraction.

The main features of the 432 are:

1) Capability Based Addressing

The 432 employs the addressing and protection concept of capabilities. A capability, or access descriptor, refers to an object and contains sets of access rights to that object.

2) Garbage Collection

3) Small Protection Domains

4) Automatic Subroutine Management

5) Process and Processor Objects

In the 432 concurrent processes and processors are represented by objects. This provide the system with a high degree of flexibility and regularity. Inherent in the 432 architecture is a high degree of support for concurrent processes and multiple processors. This includes interprocess and interprocessor synchronisation and communication.

These features are enabled by having a 'pool' of processors to which processes are dispatched from a central queue, that is the 432 is a polymorphic system. The 432 provides an effective and highly flexible means of controlling and dispatching of processes to processors. In

general, low level decisions are taken by the hardware but the progress of processes is controllable by the operating system by setting a variety of parameters in the process objects.

6) Send and Receive Mechanism

Inter-process communication is provided by a communication port which is itself an object.

7) Large Address Space

The 432 provides a large address space, in terms of both objects and physical store.

8) Flexible Operand addressing.

9) Extensive Floating-point Facilities

2.9.3.2 Components and Configurations

Currently the iAPX-432 system consist of three component types. The 43201 and 43202 make up a General Data Processor. The 43201 fetches and decodes instructions while the 43202 provides addressing and logic functions. The two devices are tightly coupled via a microinstruction bus. The final component is the 43203, the interface processor. This serves as an input-output channel. The following figure shows a four processor system, figure 2.7.

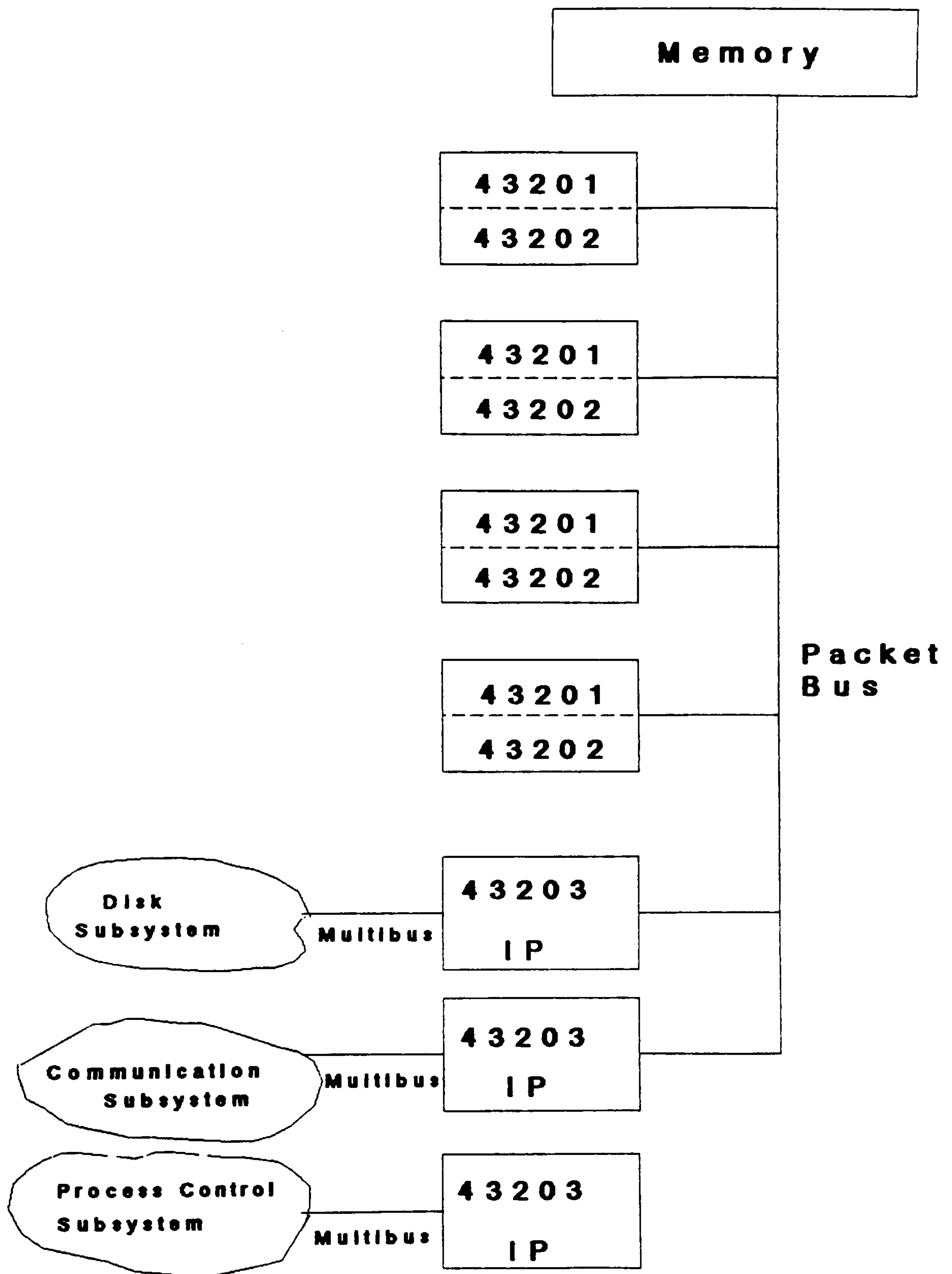


Figure 2.7

In a multi-processor configuration, memory inter-facing is a key concern. With a single memory bus the upper limit on processors seems to be five processors [23] beyond this memory interference is such that additional processors add little to system performance.

2.9.4 Multi-Processor Research At Carnegie-Mellon University

In 1971 a research project was started to examine multi-processor architectures, a main centre for the research being those architectures which share a common address space [33].

The first project, C.mmp is a relatively straight forward multi-processor. Began in 1972 in connects up to 16 prcessors to a shared memory via a crosspoint switch.

2.9.4.1 C.mmp

Four main design criteria influenced the design of the machine:

1) Minicomputers would be used as the processing elements.
2) The machine would have no 'master-slave' relationships between the processors.
3) A large address space would be provided.
4) As much commercially available hardware would be used, as was available.

2.9.4.2 The Hardware

C.mmp is an asynchronous, MIMD multi-processor [5], composed of slightly modified PDP-11/40 processors, augmented by a writable control store, figure 2.8. Up to 16 of these processors can be connected to up to 16 shared memory modules via a 16 x 16 crosspoint switch. A path through the switch is independantly established for each memory request and upto 16 paths may exist simultaneously. Control signals are carried via an independant bus called the IP-bus.

The memory modules provide a maximum physical address space of 32 megabytes. All processors are capable of accessing all memory, though the PDP-11's 16-bit architecture limits the amount of directly addressable memory at any one time to 64 kilobytes. In addition to the shared memory each processor has 8 kilobytes of private 'local' memory. This space being used for context-swaps, synchronisation etc. Input-Output devices are connected to individual processor UNI-BUSES, and are controlled by the individual processor.

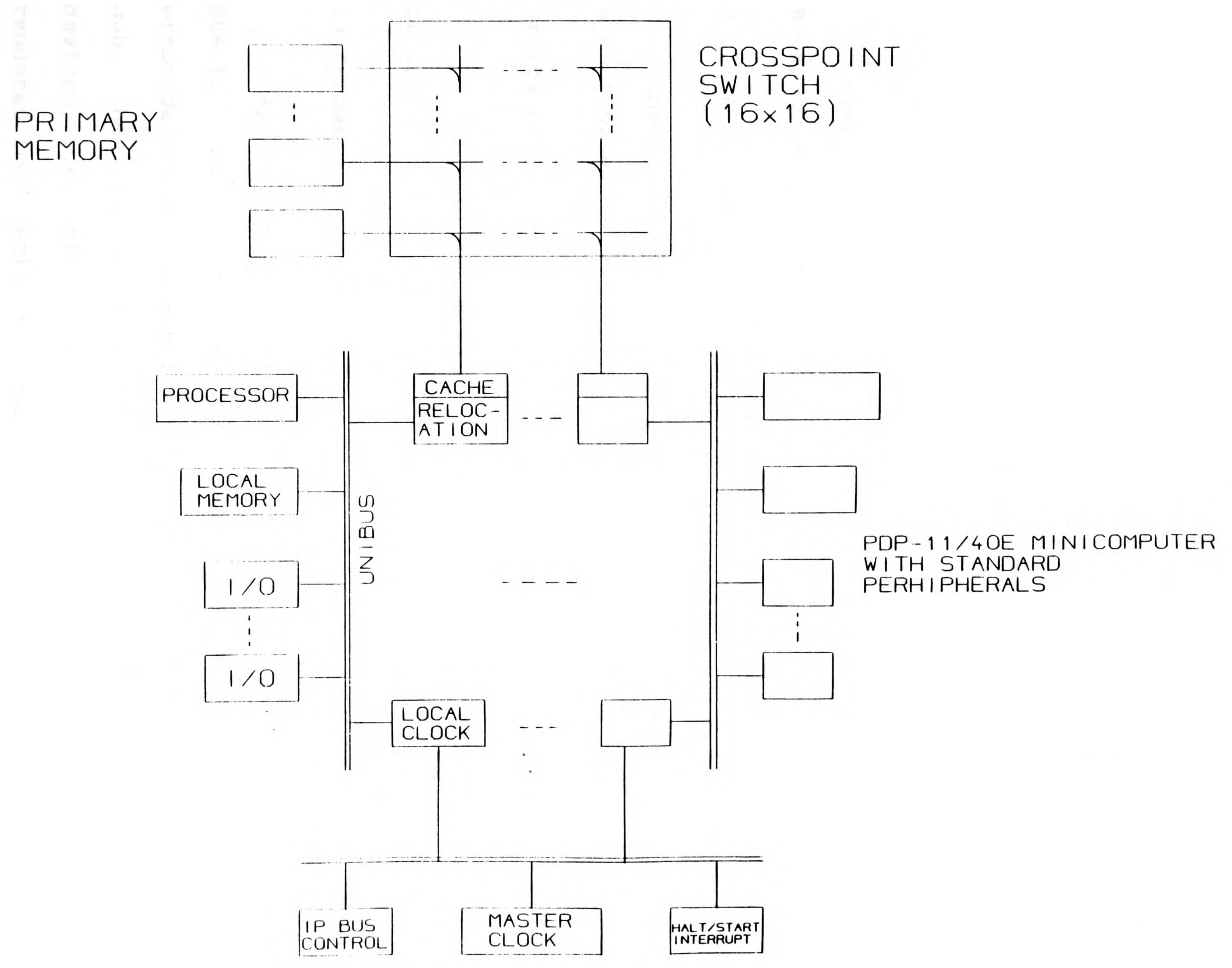


Figure 2.8

2.9.4.3 The Software Base

Hydra is the kernel operating system for C.mmp [15]. It does not provide files, command language or even a scheduler. Rather, Hydra provides an environment in which it is intended that the user should write programs that supply these facilities.

Hydra, which was a research project itself [34] , uses a capability-based protection structure.

2.9.4.4 Successes and Failures

The sucesses include then design and implementation of a cost-effective multi-processor.

Drawbacks

C.mmp had its drawbacks though, these fall into three groups:-

1) Hardware reliability.

Approximately two thirds of all system failures were due to hardware problems. This being due to insufficient error detection being built into the hardware. The systems hub, the crosspoint switch, was too reliant on other devices, i.e. processors and memory. The switch could be rendered inoperative by a malfunction in one of these units.

2) The small address space.

The PDP-11 restricts all addresses generated by user programs to 16 bits long. This address space restricts the memory size addressed to 64K. To overcome this problem C.mmp provides a facility to divide the address space into 8 pages, the addressing mechanism being similar to the 'base registers' on the IBM 360/370 style machines.

3) Partitioning.

C.mmp is able to partition processors and memory, however it is not possible to run the operating system, HYDRA, in more than one partition. C.mmp can be partitioned in such a way that some processors and memories can undergo maintenance and run stand-alone diagnostics without interfering with the larger partition running HYDRA. This means that disjoint time must be allocated for users and maintenance.

This system serves as a excellent example of a bounded parallel system, and its design must be seen as a success.

It is interesting to note the performance bottlenecks. There is a too high operating system overhead of 500 microseconds on entering and leaving the kernel. Memory contention caused by multiple processors is another problem. This is caused by several processors trying to access the same page in memory. The problem was mainly due to multiprocess applications sharing the same code amongst processors.

2.10 Engineered Unbounded Systems

In 1975 a second multi-processor project was started at CMU [6]. Cm* replaced the crosspoint switch of C.mmp with a distributed bus orientated interconnection scheme between processor-memory pairs.

2.10.1 The Structure of Cm*

One of the main features of Cm* which distinguishes it from other multi-processor architectures is that the shared memory is not disjoint from the processing elements, but a unit of memory and a processor are closely coupled in a module and a network of buses gives a processor access to non-local memory, figure 2.9.

This structure gives modular expansion without rapid interconnection costs. Memory can be shared even though there is no direct physical link between the requesting processor and memory. A Computer Module or Cm providing the processing power, primary memory and Input-Output connections for the system, figure 2.10.

The processor is a DEC LSI-11, this is a 16 bit microprocessor cluster (See below). It also provides interprocessor commnication, device interrupts, address spaces etc. The Cm's are combined into a cluster via the Map bus. This is a special purpose, packet switched bus.

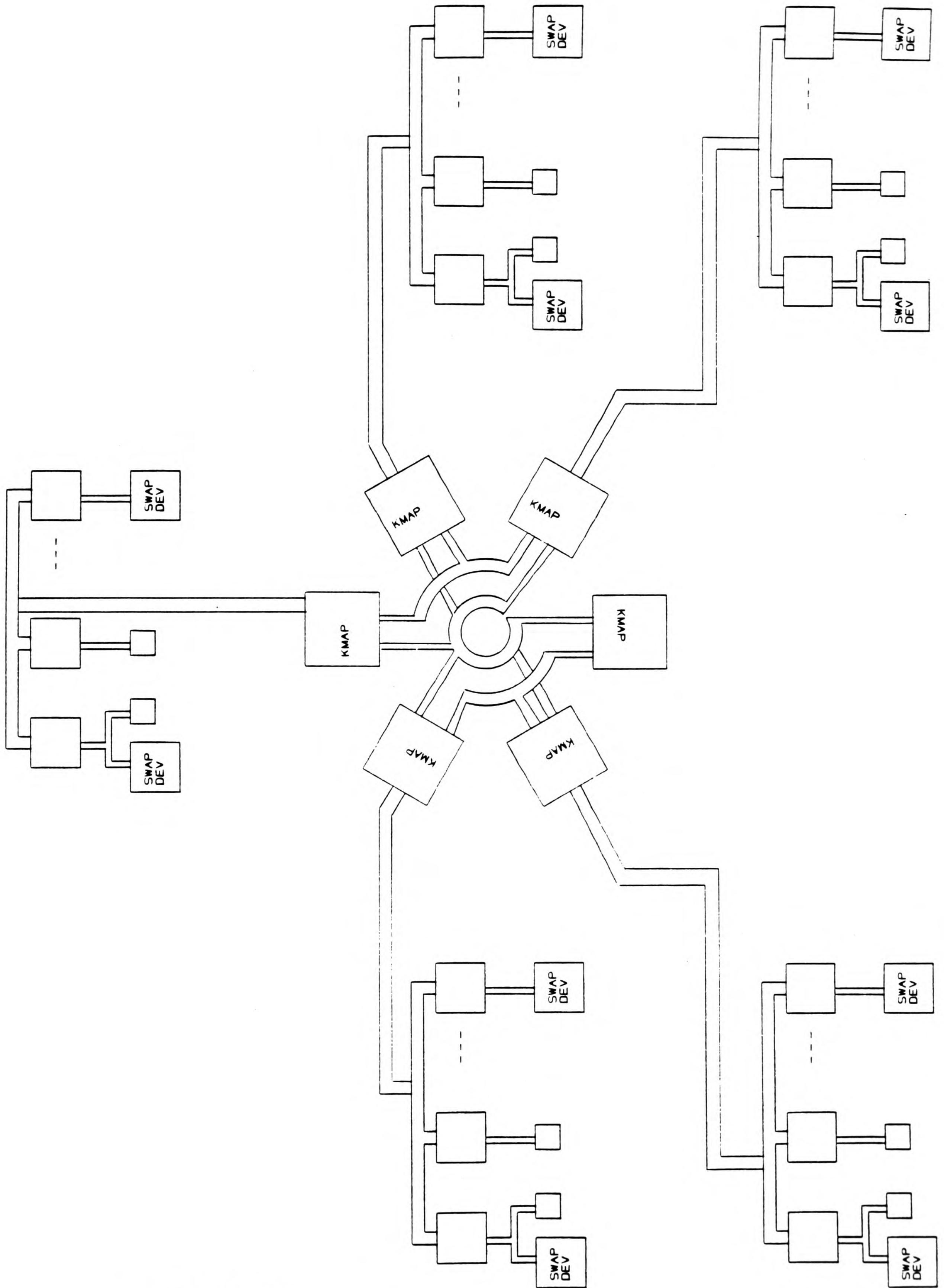


Figure 2.9

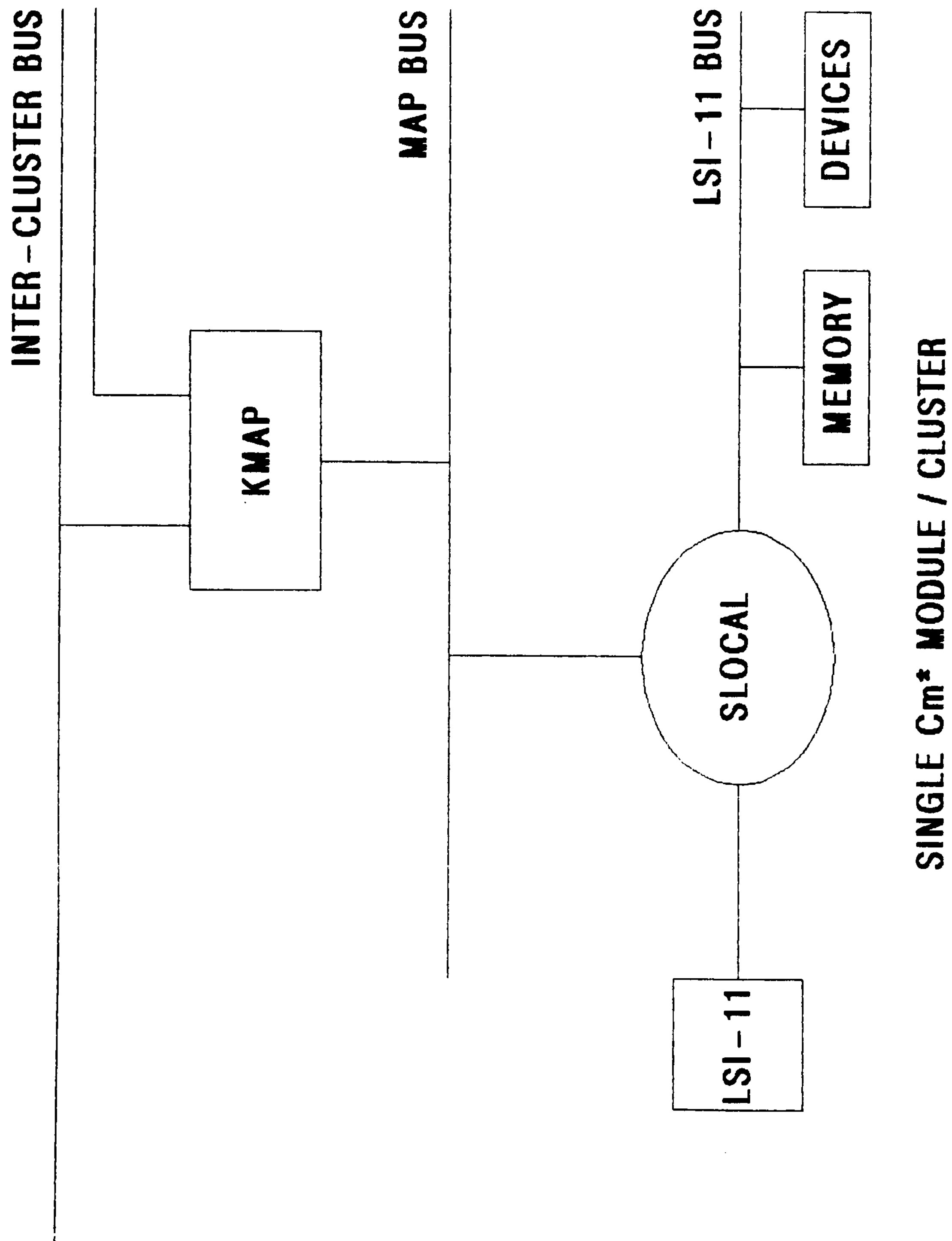


Figure 2.10

The Kmap is a special purpose 'mapping controller' which is shared by a cluster of Cm's. Clusters are connected via Inter-cluster buses. All non-local memory references in Cm* are handled by one or more Kmaps.

The Kmaps provide address expansion and mapping, both within a cluster and between clusters. The contents of a Kmap are:-

The Kbus, which provides an interface between the Map bus and the Pmap and controls all transactions on the Map bus.

The Pmap, a mapping processor.

The Linc, an interface between two inter-cluster buses.
The Kmaps and Slocals form the distributed switch.

The Structure of Cm*.

The way in which Cm*'s processors share primary memory was chosen as it offers a closer degree of coupling, or communication between processors, than would a multi computer or network configuration. The main feature of the switch structure is that shared memory is not separated from the processing elements, but rather a unit of memory and a processor are closely coupled in each module and a network of buses gives a processor access to non-local memory.

The Extensibility of Cm*.

Processing power can be expanded by increasing the number of Cm's per cluster of by adding additional clusters of Cm's. Memory capacity can be increased by either adding

it to an existing Cm or by adding additional Cm's. The communication bandwidth of an individual processor Cm* is limited by both its own performance and the bandwidth of the map bus and intercluster buses. Because there is no central bus or switching mechanism the machine can be virtually indefinitly extended.

2.10.2 The Massively Parallel Processor (MPP)

The MPP is a SIMD parallel computer with 16K processors, figure 2.11.

The MPP consists of 3 main elements:-

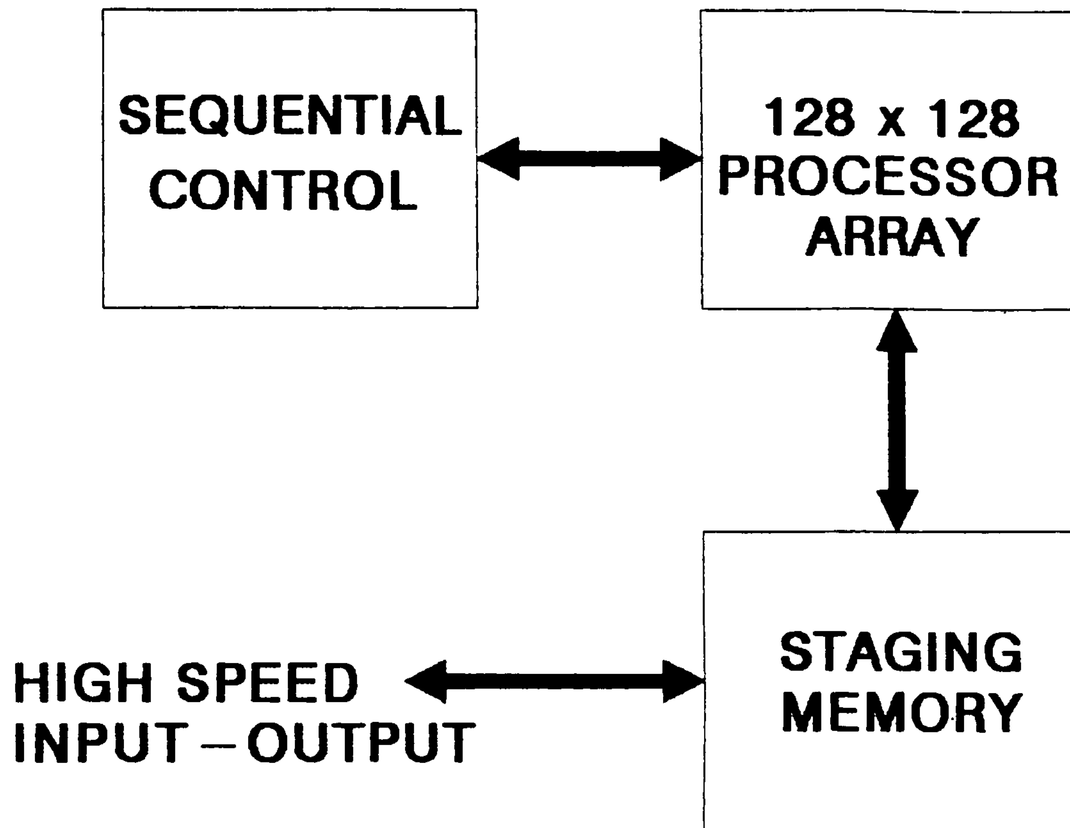
1) The sequential controller
2) The Parallel array
3) Staging memory

The controller is a high speed seqential computer with its own logic and arithmetic functions. Its primary function being to store and sequence through programs.

The controller is connected to the array via a set of interface registers.

The array consists of 16384 processors in a 128x128 configuration. Each processor acts on data in its own dedicated memory.

The array and staging memory is connected to peripherals via a high speed Input-Output bus. The staging memory acts as a data buffer between the arrays and the outside world.



Basic Organisation of MPP

Figure 2.11

2.10.3 The INMOS TRANSPUTER

The word TRANSPUTER [31] was coined to be a hybrid of 'transistor' and computer. The implication being that the device is both a component and a computer. With the TRANSPUTER, Inmos has suggested an even higher level of abstraction then the VLSI circuit.

TRANSPUTER's can themselves be used as basic cells and connected into networks, in which each node is a complete processor. Therefore the TRANSPUTER has been designed as a programmable component for building extended, parallel computing systems with a language, Occam, which allows such systems to be efficiently programmed.

The Device

The TRANSPUTER chip is a complex piece of silicon. The chip includes a high-speed, reduced instruction set, RISC, processor, 4k of static RAM, an Input-Output controller and memory controller all on a single slice of silicon. The inclusion of the Input-Output and memory controllers is similar to that of the iAPX design.

The Input-Output controller section is responsible for the four INMOS Links and an 8-bit peripheral interface bus. The net effect of this being that the TRANSPUTER can control or be controlled by existing peripheral devices.

The memory management unit allows each TRANSPUTER to address up to 4 gigabytes of off chip memory in addition to

its own RAM. One of the important decisions taken at design time was to abandon virtual memory, on the assumption that RAM is becoming cheaper more quickly than mass storage devices. As a result the address space in completely uniform thus alleviating any addressing problems.

The slightly faster on-chip memory is multi-ported so that the processor and INMOS Links can have access to it.

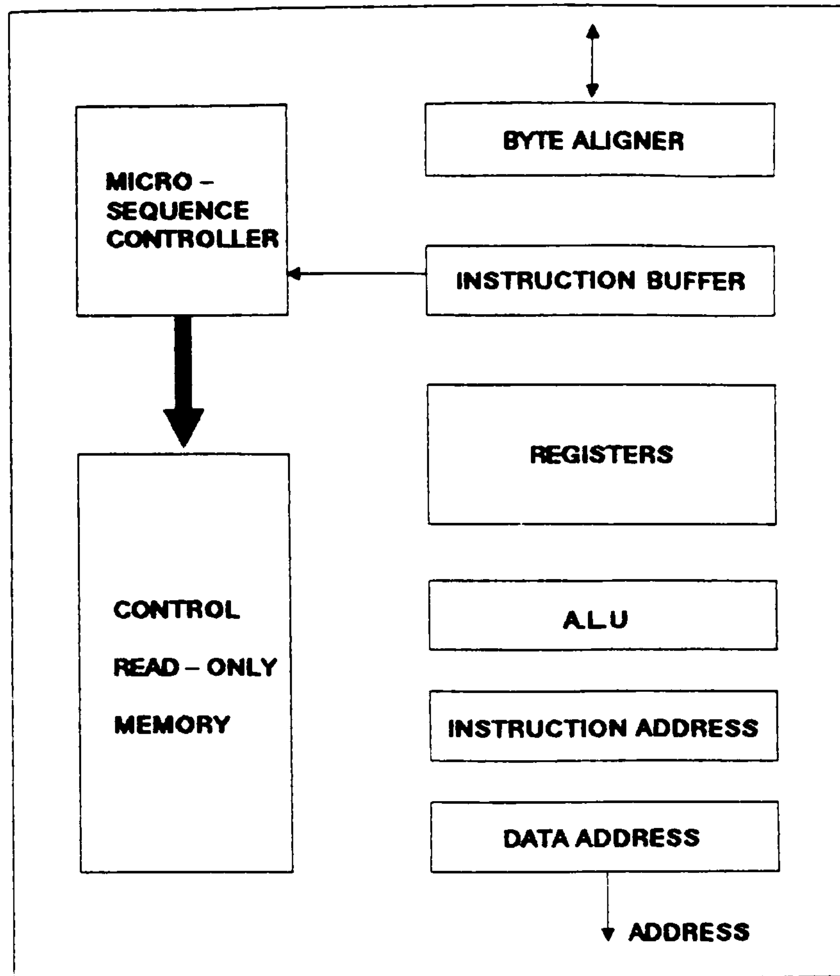
Possible Architectures

The real potential of the TRANSPUTER lies in truly parallel systems where a number of TRANSPUTERs share the workload.

The TRANSPUTER lends itself to many interconnection schemes. An obvious architecture is to connect the devices in the form of a two dimensional array, or possibly a three dimensional one, as each TRANSPUTER has 4 Links, figure 2.12.

Communication Links

The connecting of many 'conventional' microprocessors has brought several bus interconnection problems. Namely control of system interrupts; Intel's Multibus-II has five different bus structures and complex bus arbitration logic; this is mainly brought about by microprocessors being designed as uni-processors. The TRANSPUTER solves the communication problem by divorcing intercell communication from conventional memory addressing and data transfer functions, which take place via a 'conventional' bus.



TRANSPUTER PROCESSOR ORGANISATION

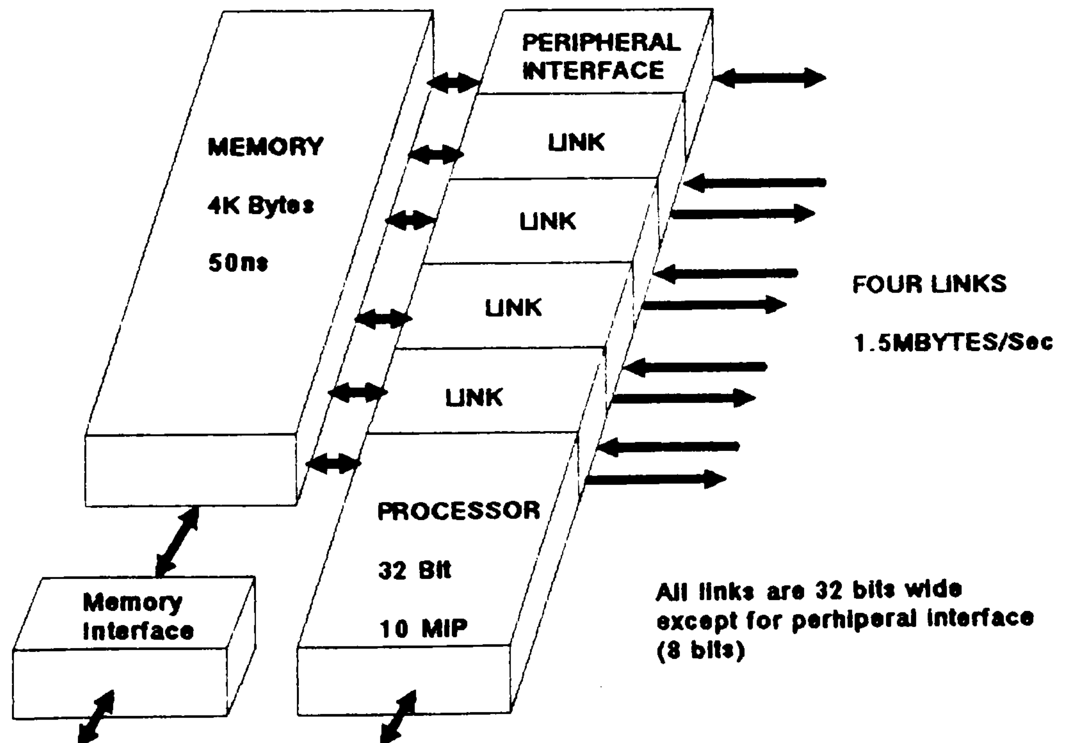


Figure 2.12

Communication between cells is accomplished via high speed serial links which operate independantly of the bus. Each TRANSPUTER has four of these links. Each is capable of operating concurrently with the others and with the processor.

The four links are a physical form of the Occam 'channel', which processes use to send data to one another [18].

The idea behind the architecture is that TRANSPUTERs will be as easy to interconnect as TTL devices.

Dataflow and Systolic Machines

The main aspect of dataflow is its elimination of the fundamental properties of conventional programming languages and machine architectures. In a data-flow architecture there is no concept of passive data storage and in a dataflow language there is no concept of variables; rather, data values move from one instruction to another as the program executes. There is no concept of flow control, counters or branching. Instead the instructions are 'Data driven'.

Dataflow Machines.

A dataflow/systolic system consists of a set of inter-connected cells, each capable of some simple operation. Because simple, regular communication and control structures have substantial advantages over complicated ones

in design and implementation. Hence, cells in a systolic system are usually interconnected too form an array or tree.

2.11 Summary

This chapter has presented an overview of existing multi-processor architectures. However all of these, with the exception of the INMOS TRANSPUTER, have built in design limitations which restrict the creation of unbounded systems. Therefore before designing new architectures it may be better to examine the applications to which these new machines will be put. This will give us a better understanding of the architectures required.

<u>References - Chapter 2</u>

1. ALEXY, G. and KATZ, B.J., "Multiprocessing Increases Power of Inexpensive Microprocessor Designs," EDN 1980.

2. AMIKURA, K., "A Logic Design for the Cell Block of a Data Flow Processor," MIT/LCS/TM 93 1977.

3. BACKUS, J., "Can Programming be liberated from the Von-Neumann Style? A Functional Style and its Algebra of Programs.," Communications of the ACM V21 N641 1978.

4. BAILEY, B., "Ceramic Chip Carriers - A New Standard in Packaging," Electronic Engineering V52 V21 N8 1978.

5. WULF, W., AND BELL, G.,"C.mmp: A Multi-Mini-Processor", AFIPS FJCC V41 1972

6. SWAN, R., "The Switching Structure and Addressing Architecture of an Extensible Multiprocessor Cm*", Ph.D. Thesis CMU 1978

7. DENNING, P.J., "Fault Tolerant Operating System," Computing Serveys V8 N4 1976.

8. DOMAN, A., "PARADOCS: A Highly Parallel Data Flow Computer and its Dataflow Language," Euromicro V7 N1 1981.

9,10.ENSLOW, P.H., "Multiprocessr Organisation - A Survey," Computing Surveys V9 N1 1977.

11 FLYNN, M.J., "Very High Speed Computing Systems", PROC. of the IEEE. Dec 1966.

12. QUICK, G.E., "The Group Processor Approach to Computer Architecture", Ph.D Thesis, UC Swansea 1982.

13. GOSTELOW, K.P, and THOMAS, R.E. "An Asynchronous Programming Language and Computing Machine," UC Irvine TR 127a.

14. I.E.E.E,, "Special Issue on Array Processor Architecture," Computer V14 N9 1981.

15. WULF, W. et al., "Hydra: The Kernel of a Multiprocessor Operating System", Comm. ACM V17 1974.

16. KUCK, D.J., "The Structure of Computers and Computation, Vol. 1," John Wiley 1978.

17. LEVY, J.V., "A Multiple Computer System for Reliable Transaction Processing," SigSmall Newsletter V4 N5 1978.

18 INMOS, "Occam Programming Manual", INMOS 1983

19. MILLS, D.L., "The Basic Operating System for the Distributed Computer Network", SigSmall Newsletter V4 N5 19778

20-23. MYERS, G., "Advances in Computer Architecture 2nd Edition," Addison Wesley 1982.

24. COMPUTER ARCHITECTURE NEWS, "Conference Proceedings 7th Annual Symposium on Computer Architecture," Sigarch News V8 N3 1980.

25. ALMES, G., "Garbage Collection in an Object-Oriented System", Ph.D Thesis CMU 1980.

26. ROSENFIELD, A., Error Recovery and Process Communication," Stanford University Ph.D 1976.

27. MYERS, G., "Advances in Computer Architecture 2nd Edition," Addison Wesley 1982.

28. SCHINDLER, M., "New Architectures Keep Pace with Throughput Needs," Electronic Design May 14 1981.

29. SHARP, J.A., "Some Thoughts on Data Flow Computer Architectures," Sigarch News V8 N4 1980.

30 SWAN, R.J., "CM* - A Modular multi-processor", AFIPS V46

31. INMOS, "The TRANSPUTER", Advance Information 1982.

32. WEITZMAN, C., "Distributed Micro/Minicomputer Systems", Prentice-Hall 1980.

33. WULF, W.A. AND HARBISON, S.P, "Reflections in a Pool of Processors," CMU-CS-78-103.

34. WULF, W.A., LEVIN, R., and PIERSON, C., "Overview of the Hydra Operating System", Proceedings of the 5th Symposium on Operating System Principles. 1975.

35. DIGITAL EQUIPMENT CORP., "VAX-11/780 Hardware Handbook, 1983

36. MYERS, G., "Advances in Computer Architecture 2nd Edition," Addison Wesley 1982.

CHAPTER THREE

CHAPTER THREE

FIFTH GENERATION NEEDS

3.0 Introduction

The proposed advanced fifth generation computer systems should incorporate all the advances made in computer technology over the last 20 years. With fifth generation computers, however, the expected changes will be more generic changes, involving not only device technology but also simultaneous changes in design philosophy and in envisioned applications. This technological change is so great that we could even call fifth generation systems new-era computers. These advances, such as high reliability, high availability, coupled with VLSI implementation impose severe difficulties for the computer architect. These difficulties are further compounded when unbounded parallelism is exploited.

This chapter discusses many aspects of fifth generation system design, and proposals for the new areas of activity.

3.1 Background

The general consensus that the computer of the 1990's will be a non Von-Neumann architecture will be substantiated by [12] application needs. Fifth generation computing

machines will be established, supporting the following application areas:

1) Knowledge based information processing systems.

2) Distributed computer systems based on wide as well as local area networks, and integrated parallel architecture.

3) Data and demand driven computers. User oriented self programming systems supporting very high level programmming languages.

4) VLSI implementation of dedicated processors exploiting miniaturisation.

3.2 Fifth Generation Computer Systems

These new machines will replace the outdated machines of the past, just as the electronic calculator replaced the Engineer's sliderule, or the wordprocessor replaced the typewriter.

In the analysis of current system architectures for fifth generation applications, a list of 'needs' may be established. The 'needs list' represents a detailed breakdown of the functional requirement of the application, and therefore provide the first stage in the top down design process.

3.3 Application Areas

Fifth generation computers are knowledge information processing systems and processors. These systems are the artifical intelligence community's view of the image presented by future computers. Three areas of research have been identified by various researchers [11, 1, 3]

1) Knowledge based expert systems.
2) Human orientated I/O.
3) Very high level languages.

Communication and computers represents Wide Area Networks, Local Area Networks and parallel computers. In the past network and parallel machines have been developed separately [6], the advancement in each being sustained by development in semiconductor technology [7]. However, it has long been advocated that the spectrum of decentralised systems should be fully integrated. Therefore to achieve this it is necessary for all components to conform to a common decentralised system architecture.

3.4 Processor Architecture Exploiting VLSI

Processor architectures to exploit VLSI define a new generation of VLSI building blocks to succeed the microprocessor. Traditional microprocessors such as the iAPX-432, which contains over 100,000 transistors, are starting to become commonplace. However, attempting to make larger scale single processors in VLSI becomes self

defeating due to communication problems and escalating cost. A solution is miniature computers which can be replicated such as memory cells and orientated as a multiprocessor architecture [8,4]. These machines are implemented by only a few different types of simple cells, and use extensive pipelining and parallel-processing to achieve high performance.

The only device to be designed with these criteria is the Inmos Transputer.

3.5 Needs and Uses

It is reasonable to assume that research into fifth generation systems using technology currently available will begin to produce results by the early 1990's. This assumption raises an important question. Where are the application areas and who will be the users of the new technology?

The Japanese see almost everybody as users, with new applications touching almost every aspect of human life. At the present time ten broad catagories of application have been identified. These are:

1) Industrial Automation.
2) Office Automation.
3) Science and Engineering.
4) Computer hardware and software.
5) Military.
6) Aerospace.

7) Retail and Service Industuries.

8) Education.

9) Health.

10) The Arts, Culture and Leisure.

All these areas have had applications demonstrated in a research setting, but none have been transferred to the commercial market. All applications shown rely in advances in the four areas mentioned at the beginning of this chapter.

3.5.1 Industrial Automation

This area is seen as one of the 'prime' application control areas for fifth generation technology. Prospective users include manufacturers of goods, designers of manufactured products, product and plant engineers.

Some aspects of industrial automation already exist, i.e automatic control systems, production line robots; however new application now become possible; i.e. automated factories, computer aided design, aomputer aided manufacturing, robotics, inventory management, product-cost estimates, control and routing of production runs, expert systems for design.

The resources needed to support these applications are common to all fifth generation requirements, namely high speed parallel data-base machines, parallel processors and high quality CAD-CAM workstations.

The last ten years have seen an explosion in the number of industrial robots and in the number of companies using CAD-CAM techniques. Significant advances have been made in sensors for robots, such as computer vision systems, and high level languages for controlling robots. The new generation of computers will integrate these components to achieve a highly automated manufacturing plant.

3.5.2 Office Automation

Of all the possible application areas office automation will probably provide the most financially rewarding area. The section heading might better be called 'Business Automation', covering all aspects of a non-manufacturing nature.

Application areas are only limited by the diversity of businesses, but management, administrators and secretaries will be the major users of such systems.

The new generation in computing will not only supply the hardware and software technology to support information management, it will also use better techniques for using these systems. The user will be able to interact with the system via natural language I/O and expert systems will give advice on how to use computing systems and will assist in recovering the relevant information.

The components for office automation will undoubtedly include wideband networks and local area networks. The use of word processors and electronic filing systems in offices

is now commonplace. Similarly the use of electronic mail, spreadsheets, appointment calendars and database management systems is becoming more popular. The technology to totally automate the office already exists, its components must now be integrated into complete systems. These components include specialised database processors, expert systems and improved I/O systems.

3.5.3 Science and Engineering

To date scientists and engineers have always been the first to employ the use of new technology [9]. There is no reason why this group of people will not do so again.

Once again the application areas are only limited by the depth of current knowledge, but suitable areas include: expert systems for fault diagnosis, capturing and applying scientific expertise.

The systems technologists will employ powerful database management systems, parallel database and expert system processors. Scientists will use new generation equipment in a variety of ways. With new hardware and improved user interfaces, fifth generation systems will be able to take advantage of expert systems to capture and apply scientific knowledge. Expert systems are already beginning to support scientific research.

3.5.4 Computer Hardware and Sofware

To enable advances in the fields described in this chapter, fifth generation computer hardware and software must first be developed.

This will enable programmers, analysts and engineers to develop the packages required by the end user.

Possible applications which effect every computer users include semi automatic design and development of programs, specalised architecture for implementing firmware and expert systems for fault diagnosis of computer hardware.

The tools necessary for future development include very high level languages and hardware specification languages, that is, languages one or two steps further removed from LISP and PROLOG. Other required components are integrated software development environments, Silicon compilers, parallel processors, dataflow languages and machines, parallel programming languages, compilers and operating systems.

One of the main aims of current research is to reduce the cost of producing software. Writing, debugging and testing current systems is a highly labour intensive process. There is much interest in automatic programming. Complete, general purpose automatic programing will probably take many years of fundamental research; however automatic programming in specalised areas may be more realistic.

A major area for fifth generation systems will be that of automated VLSI design. The very complexity of devices that VLSI permits, require that automated techniques are employed for managing this complexity. One of the most important applications of new technology will be the simulation of computer assisted design of even newer generations of machine.

3.5.5 Military

The military have always been large investors in computing technology [14], innovating many new designs and products. In the US most of financing has been via the DARPA scheme.

The new technology will be used in all existing areas of military computing as well as some new fields i.e. aspects of planning, decision support; command and control systems; supply and support logistics; intelligent autonomous weapons systems, parallel architectures for analysing RADAR and SONAR images.

Applications for military uses fall into two catagories. The first includes information management systems and expert systems that support military systems in terms of planning etc. The second comprises of the guidance and targetting of autonomous weapons.

3.5.6 Aerospace

The aerospace industry has always been a prolific user of new technology; pioneering several new designs. NASA has

been the funding body of many new architectural designs, i.e. the ILLIAC-IV [10], Pilots, air traffic control, scientists and engineers studying space and remotely sensed earth resources, will be able to employ new generation tools such as: air traffic control monitors, deep space exploration probes, earth resource monitors and semi-autonmous space ground based sensors, expert systems for fault diagnosis in space craft, self replicating machines for space manufacturing.

The components used in this field will include large scale distributed data bases for automatic data reduction from a network of ground sensors, expert systems for fault diagnosis in space craft and self replicating machines for space manufacturing.

The new generation of computers will be applied to aerospace in areas such as air traffic control, information management for pilots and control of autonomous aircraft. Even though todays most advanced aircraft are already controlled by embedded computer systems, the pilot must still cope with a large quantity of information from instruments and displays. Techniques such as voice recognition, speech output will aid the pilot to select and interpret the most vital information.

3.5.7 Retail and Service Industries

Consumers, retailers, advertisers, market development, service companies, lawyers, travel agents etc. will benefit

from systems such as computer based catalogues, remote ordering and shopping, teleconferences, accounting, billing and invoicing systems, automated information systems, expert system for a mass of uses.

Hardware will be required to support multimedia information systems combining graphical, image and textual data, natural language Input/Output systems for inquiry systems, wideband networks for communication between systems.

Consumers will be able to shop and buy goods using an online catalogue, which handles fund transfer, billing, shipping and reordering. The greatest abundance of new applications will come in the area of information service industries.

The components to handle these advances have existed for some time, but not as an integrated system. More attention must be paid to developing more natural user interfaces for both accessing and updating information.

3.5.8 Education

Students at all levels will be able to use intelligent computer-aided instruction (CAI) systems which permit the student to direct and contol the presentation of course material. Computer based training for adults and professionals, computer based assistants which explain how to use computing systems.

Software and specalised processors to support voice I/O, specalised hardware for graphics and images, and libraries available via digital networks will be needed to support such education.

CAI systems are currently in use at all levels of education and training. More advanced developments in CAI will enable intelligent CAI systems, enabling natural language to control the lessons.

3.5.9 Health Care

In the past computers in medicine have been limited to applications such as patient administration [13] and other accounting tasks. Today however, one of the first application areas for expert systems has been in the field of medical diagnosis.

The users in this field include: Doctors, hospitals, patients, the handicapped and disabled.

Other areas of use include: expert systems for diagnosis and prescripion, data base systems for medical records, management and monitoring of patients, automatic analysis of experiment, reading machines for the blind and sensory prosthesis.

Applications in medical environments will continue to be mainly in expert systems. Such systems have some of their first and most successful applications. There are, however,

a number of research and social issues to be resolved; i.e. medical liability of expert programs.

New generation technology will also support aids for the handicapped and disabled. Reading machines, sensors and computer controlled wheelchairs are some of the applications directed toward the younger generations of society. However the public at large should be involved in fifth generation applications i.e large scale storage systems, wideband networks to access cultural information, high resolution digital sound, graphical I/O devices such as electronic paint brushes and solid state cameras.

3.5.10 Leisure

A large proportion of the income that the entertainment industry has been able to gather during the last ten years has come through computer based video games. If this is indicative of the potential for new generation technology then this area may be the most financially rewarding.

3.6 Involved Countries

Within the last two to three years, Japan, the United States and Europe have initiated significant research programmes in fifth generation computing, all feeling that the first to market a commercial fifth generation system will permanently have a lead on the others.

In a way, all concerned parties are involved in a strategy of mutual catching up. Japan has the

longest-running, formal national programme. Its Fifth Generation Computer Systems Project was officially launched in 1981, but its aim is to catch up with the United States, which has been funding key research areas for some twenty years. However, it was not until the Japanese had announced their formal research project that the United States initiated its own concerted national projects.

With the advent of Fifth Generation Computing Systems [11], Japan effectively announced to the World that it would no longer be taking existing Western technology and improving on it, but that it was determined to take a lead in the research and development of unprecedented systems.

3.7 Concerns and Goals

The various countries differ in their concerns and purpose of their research, and the aims of each project are distinct.

3.7.1 Japan and ICOT

At the center of Japan's research effort is the Institute for New Generation Computer Technology (ICOT). ICOT's objective is to research and develop computer technology that can perform more humanlike intellectual functions, namely, inference, association and learning.

To achieve this objective ICOT is finding ways to supplant traditional sequential processing with parallel processing; as only the great speed and capacity of parallel

processing are sufficient for developing the new application areas. The net result of Japan's Fifth Generation Computer Systems project will be the basic technology and demonstration systems to build true fifth generation systems.

The ICOT research centre plans to approach the aim of a true fifth generation system by pursuing two intermediate hardware projects. These are a parallel inference machine and a knowledge-based machine.

The parallel inference machine is a system that follows a line of reasoning to arrive at, or infer, a conclusion on the basis of the facts presented to it. The knowledge based machine is a system that efficiently manages large amounts of data. Both machines employ forms of data flow processing.

3.8 Designing the Next Generation

Much of the technology required to achieve the aims of fifth generation computer scientists can be accomplished by advances in the current state of the art of conventional Von-Neumann computing. But certain areas, such as natural language input/output, can only be tackled with thousands or millions of times more processing power than current technology permits, and by software markedly different from todays programs.

3.8.1 Exploiting Parallelism

The only feasible answer to the above problem is massive parallelism. There are, however, problems with parallelism; matching algorithms to parallel systems is one, and getting all the processors in a parallel system to work efficiently is another.

Various parallel architectures have been discussed by computer scientists, including trees, square and cubic arrays and data-flow systems. Some researchers have suggested that a general purpose parallel processor is the best way to approach the problem, others have decided to work out parallel solutions to problems and then implement an architecture around it.

3.8.2 VLSI: The Solution?

Architectures consisting of many simple processors, each with a small amount of local memory are made feasible by VLSI technology. Once the initial circuit specification has been accomplished many processors can be placed on the same chip, and a number of chips fabricated with ease. However, designing such circuitry is not easy. Therefore, VLSI design tools, silicon compilers, are a required factor in fifth generation computing efforts.

3.9 The Problems to be Encountered

There are several major obstacles to be overcome before any fifth generation system is built. These include both conceptual limitations and physical ones.

3.9.1 Physical Limitations

Over the last decade, chip densities have been increasing at a fairly constant rate. However current VLSI techniques can produce paths two micrometers wide. At this level inpurities in the base material come into effect. One proposed solution is to move to wafer scale integration. If a number of devices are fabricated on one wafer then all non-working devices can be 'cut' out of the system electrically

3.9.2 Conceptual Limitations

Processes which execute concurrently may occasionally make simultaneous demands on shared system resources. Communication is a critial issue in concurrent machine architectures. Long delays in communication may result in performance degradation to a point where the potential speed of concurrency is negated. Therefore the design of the bus architecture is the key choice in linking thousands of processors.

Processes which execute concurrently may occasionally make simultaneous demands on shared system resources. When such contention is present, simultaneous demands are

serviced in a serial order, and so some processes must experience delayed access to the resource. Such contention introduces a coupling among processes because the activities of one process can effect the performance of other processes that share the resource with the first.

Shared resources are those for which two or more concurrently executing processes can make simultaneous demands.

For multi processor systems, interest is primarily directed at shared buses and switching devices. Contention for shared resources results in queueing delays at critical resources [2]. By their very nature, the individual components of a multiprocessor must share some of the physical resources of the machine. By definition some or all of the memory must be shared, but use of other structures, interconnect paths etc. may be critical.

For each element in the system there is a maximum demand that it can serve per unit time. In addition, any time two or more processes make simultaneous demands, performance will suffer unless the resource can serve multiple requests in parallel.

Therefore the critical area of design is not the processing cell, but the interconnection scheme of the cells and arbitration mechanism governing the cells.

3.10 Future Computer Architecture

The application of fifth generation computer systems in such are as Artifical Intelligence, Image Processing, Real-Time Language Translation etc. demand the fusion of thousands of Processing Cells. The potential advantages of cellular computer systems are in increased system throughput by the simple addition of more processing cells. With this in mind, bus arbitration must be distributed particularly when the simple multi-processor system is expanded to a fully distributed cellular processing environment consisting of thousands of processing cells. These Processing Cells are conceptually contained in a system module, currently realised as a printed circuit board. The purpose of the bus controller is to support inter module communication at one "level", and intra module communication at another "level". As many Cells share the common buses, the bus arbiter must be fast in operation. Another very imporant requirement in the bus arbiter is expandability; as a system grows in the number of modules, so must the bus arbitration grow with it.

References - Chapter 3

1. HMSO, "The Report of the Alvey Committee", (HMSO Ref. ISBN 0 11 5136533)

2. CHANSLER, R.J., "Coupling Systems with Many Processors", CMU-CS-82-131 1982.

3. FUCHI, K., "The Direction the 5GCS Project Will Take". New Generation Computing V1 N1

4. INMOS, "The Transputer Technical Reference Manual"

5. KUNG, H.T., "Why Systolic Architectures," IEEE Computer Jan 1982

6,7. PERKIN-ELMER, "PENnet R03 Network Introduction Manual"

8. QUICK, G.E., "The Group Processor Approach to Multiprocessor Architecture," Ph.D. Thesis UC Swansea 1982.

9. METROPOLIS, N., HOWLETT, J. and ROTA G., "A History of Computing in the Twentieth Century", Academic Press 1980

10. SIEWIOREK., D.P., BELL., C.G., AND NEWELL., A., "Computer Structures: Principles and Examples", McGraw-Hill

11. TRELEAVEN, P., AND LIMA, I., "Japan's Fifth Generation Computer Systems", IEEE Computer Aug 1982

12. PASEMAN, W., "Applying Dataflow in the Real World", BYTE May 1985.

13. IEEE Editorial, "THE DoD STARS Programme", IEEE Computer Nov 1983

CHAPTER FOUR

CHAPTER FOUR

BUS ARBITRATION CONCEPTS

4.0 Background

The proposed advances in fifth generation Computing Systems aim to provide an Intelligent Image to the system user. While such images are software based, written in languages such as Prolog and LISP, much of the proposed hardware architecture has lacked innovation and vision. This chapter addresses these two important points by providing an insight into modern Bus architectures for cellular systems. In order that unique system architectures may evolve, a hierarchical bus arbitration structure is proposed.

The section on bus arbitration is extraced from Quick's Thesis and is included in this Thesis so as to provide a fuller understanding of bus arbitration concepts and as background reading to the discussion on the Group Processor System Architecture.

4.1 Introduction

A cloud of uncertainty hangs over the physical image that fifth generation computing systems will adopt. Therefore in this chapter, the structure of a computer system is presented as a functional module structure, together with its operation. This presentation is oriented towards VLSI realisation of the various modules that will

eventually make up a typical fifth generation computer system. These modules will become sub-system components that will be integrated into an overall system architecture supporting hardware and software extensibility. These separate, but coupled modules, are interconnected by the common bus structure and supervised by the Bus Arbitration Mechanism.

4.2 Current Computer Architecture

In any simple parallel system, Figure-4.1, containing two or more processing cells, e.g. central-processor and I/O processor, a bus controlling mechanism is required to resolve the simultaneous requests for the use of a system resource [1,3,4,5]. A basic requirement of any controller is to allow only one system element or cell, the ability to gain access, i.e. write, to the shared bus. However, systems can be configured for multiple reads on a common data stream, on a common bus system.

The bus arbitration mechanism may be seen as the unifying factor in any multiprocessor system architecture, resolving simultaneous bus request conflicts. Alternatively, Bus Arbitration may be conceived as either the system's hub, or the achilles heel, as all communication between intelligent cells are scheduled for bus access by this mechanism.

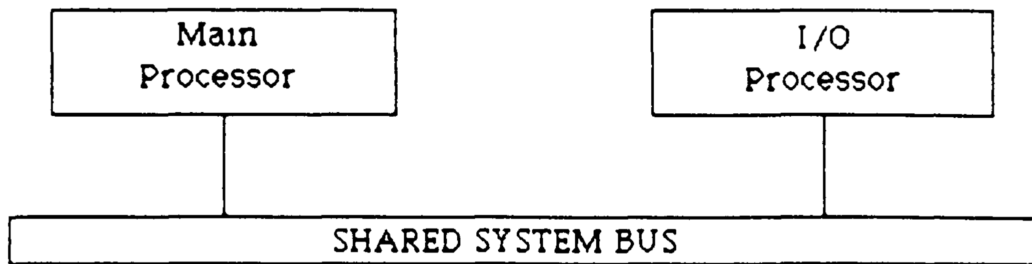


Figure 4.1 A Simple Parallel System

4.3 Bus Arbitration Objectives

According to Plummer [6] the design of arbiters is somewhat harder than most logic circuits because traditional design approaches are vastly too cumbersome. The usual design assumptions are that inputs are allowed to change only if the circuit is in a stable state and only one input at a time will change. Arbiters violate both of these assumptions.

The basic functional requirement of any integrated bus arbiter must satisfy five basic design operations which are:-

(1) The operation of mapping one, and only one, output to the corresponding input request must be executed in finite time. That is, the delay in allocation must be seen as transparent to the requesting resource.

(2) The arbiter must be independent of the communication between the communicating cells, during all communication activities. That is, the communicating data should not control the allocation, it should be directed by the system control structure.

(3) The interconnection of arbiters should provide for both equal and priority based arbitration [2]. This allows the operating system to gain control of the system hardware when required.

(4) Mechanisms must be available for the dynamic locking out of an arbiter. This enables a degree of added reliability when arbiter cells become unserviceable.

(5) The maximum number of cells, and system architectures, should be able to share a common arbiter design. That is, the design should not be tailored to a unique architecture enabling the replication of cells to a high degree.

4.4 Current Arbitration Techniques

Several methods have been implemented to realise the resolving of bus conflicts. The different control schemes can be roughly classified as being either "Centralised" or "Distributed". If the hardware for passing bus control from one cell to another is largely concentrated in one location, it is called "centralised", while in a distributed system the control logic is spread throughout the cells on the bus. Most arbiters use combinations or modifications of the following three schemes:-

1. Daisy Chaining.
2. Polling.
3. Independant Requests.

4.5 Centralised Arbitration

With centralised arbitration, a single hardware unit is used to recognise and grant requests for use of the bus. While this system has many advantages, such as much simpler hardware design, it clearly can impose severe limitations when the number of processors expands dramatically.

4.5.1 Daisy Chaining

Each cell on the bus can generate a request via a common Bus Request line, Figure 4.2. Whenever the Bus Controller receives a request on the Bus Request line, it returns a signal on the Bus Available line, which is daisy chained through each cell. If a cell receives the Bus Available signal and does not require it, then it is passed on to the next cell in the line. If a cell does require control of the bus then the bus available signal is not propagated to the next in line. The requesting cell raises the bus busy line and drops its bus request line. The cell then starts to write to the bus. The Busy line keeps the Bus Available line up until the end of transmission when the Bus Busy and Bus Available lines are lowered.

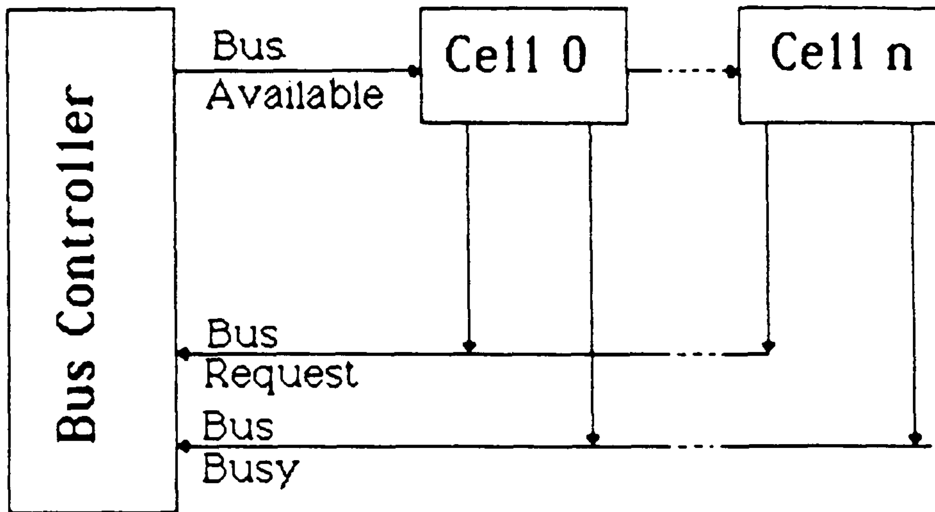


Figure 4.2 Centralised Bus Control
Daisy Chain

The advantages of this system are in its simplicity as very few control lines are needed, and additional cells can be added by simply connecting them into the bus. A disadvantage of the system is its susceptability to failure. If a failure occurs in the Bus Available circuitry, it could prevent succeeding cells from ever obtaining control of the bus. Another problem is the fixed priority structure. Cells which are "closer" to the bus controller always receive control of the bus in preference to those "further" away. Clearly, mapping important software physically near to the arbitration reduces the machines desirability, as most software would have to be written so as to be position dependant. Clearly this is a non-goal of fifth generation needs, although it may be felt desirable to place the operating system in this high priority position.

4.5.2 Polling

As in the previous system; each cell on the bus, Figure 4.3, can place a signal on the Bus Request line. When the Bus Controller receives a request it starts polling the cells to determine which one made the request. The polling is done by issuing addresses on the polling lines. When the address corresponds to a requesting cell, that cell raises the Bus Busy line. The controller stops polling until the cell has completed its transmission and removed the Busy signal. If there is another request, the addressing may start from zero or continue from where it stopped.

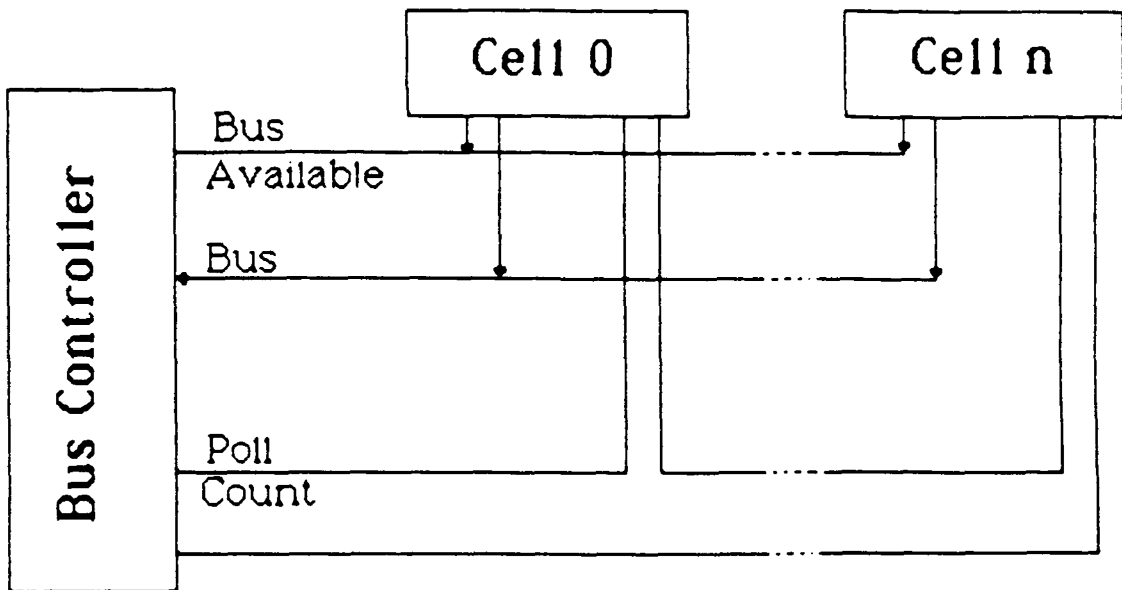


Figure 4.3 Centralised Bus Control
Polling With Global Counters

Starting from zero fixes the priority of the system, so that the most important software must be located near the low addresses. Continuing from the stopping point gives each cell an equal chance. Placing intelligence within the arbiter enables greater flexibility in scheduling the physical addresses of high priority software, but adds greatly to the cost of the arbiter.

The advantages of this mechanism are that polling does not suffer from reliability or physical displacement problems, but the number of cells is limited by the number of polling lines. The use of 100's of thousands of processing elements would imply that a corresponding number of lines must be available for each cell. This is clearly not acceptable, as the number of lines at some point must be finite, if only to constrain the location of such lines to within a single cabinet of acceptable size.

4.5.3 Independant Requests

In this method each cell has a pair of Bus Request and Bus Grant lines, which it uses for communication with the Bus Controller. When a cell requires use of the bus, it sends its Bus Request to the controller. The controller selects the cell to be serviced and sends a Bus Grant to it. The selected cell lowers its Bus Request and raises the Bus Assigned line, indicating to all other users that the bus is busy. After transmission, the cell lowers the Bus Assigned and the Bus Controller removes the Bus Grant signal and selects the next requesting cell. (See figure 4.4)

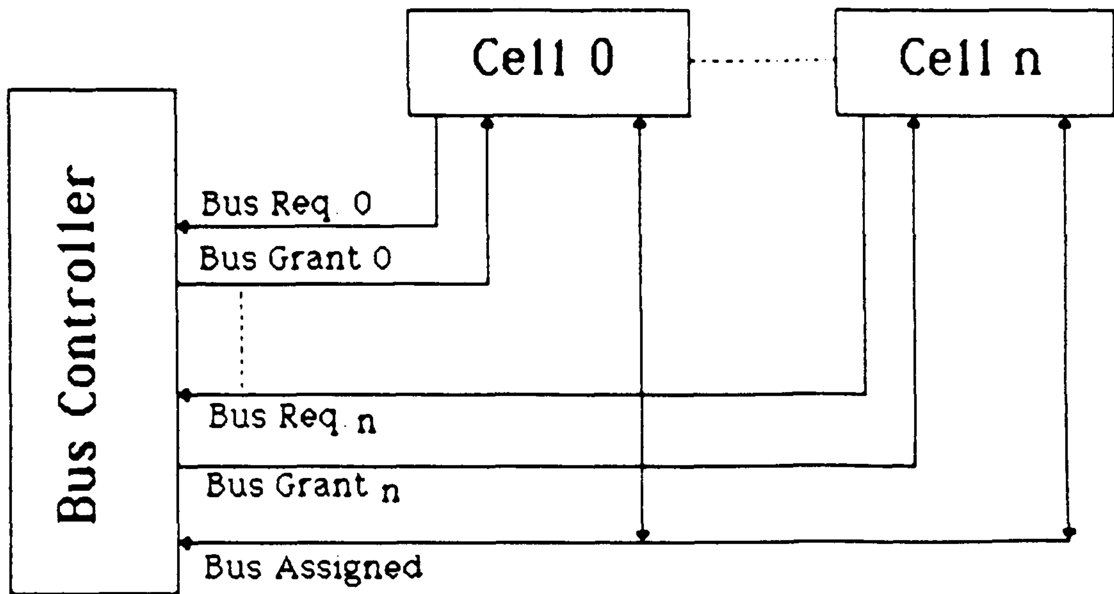


Figure 4.4 Centralised Bus Control
Independant Requests

This method has the advantage of lower overheads when allocating the bus, since all bus requests are presented simultaneously to the Bus Controller. In addition, there is complete flexibility for selecting the next requesting cell as the system is performing a true first in first out sequence.

The major disadvantage of this system is the number of lines and connectors needed for contol. There must be a pair of Bus request/Bus grant lines for each cell, although if bus grant speed were not important they could be multiplexed onto one line. The complexity of the allocation algorithm will also be reflected in the amount of Bus Controller hardware available to the cell. As an indication; for application areas that require redundancy, three communication buses may be required, with corresponding three Bus request/Bus grant lines. Clearly, in a system with only 1000 processing cells this would result in 3000 request/grant lines, a figure too large so as to be feasible.

4.6 Distributed Arbitration

The block diagram of a typical arbiter network is illustrated, showing the complexity of the interconnection of the various module interface/inter module bus cells, figure 4.5.

Within a distributed system; the bus control logic is primarily spread throughout all the cells on the bus.

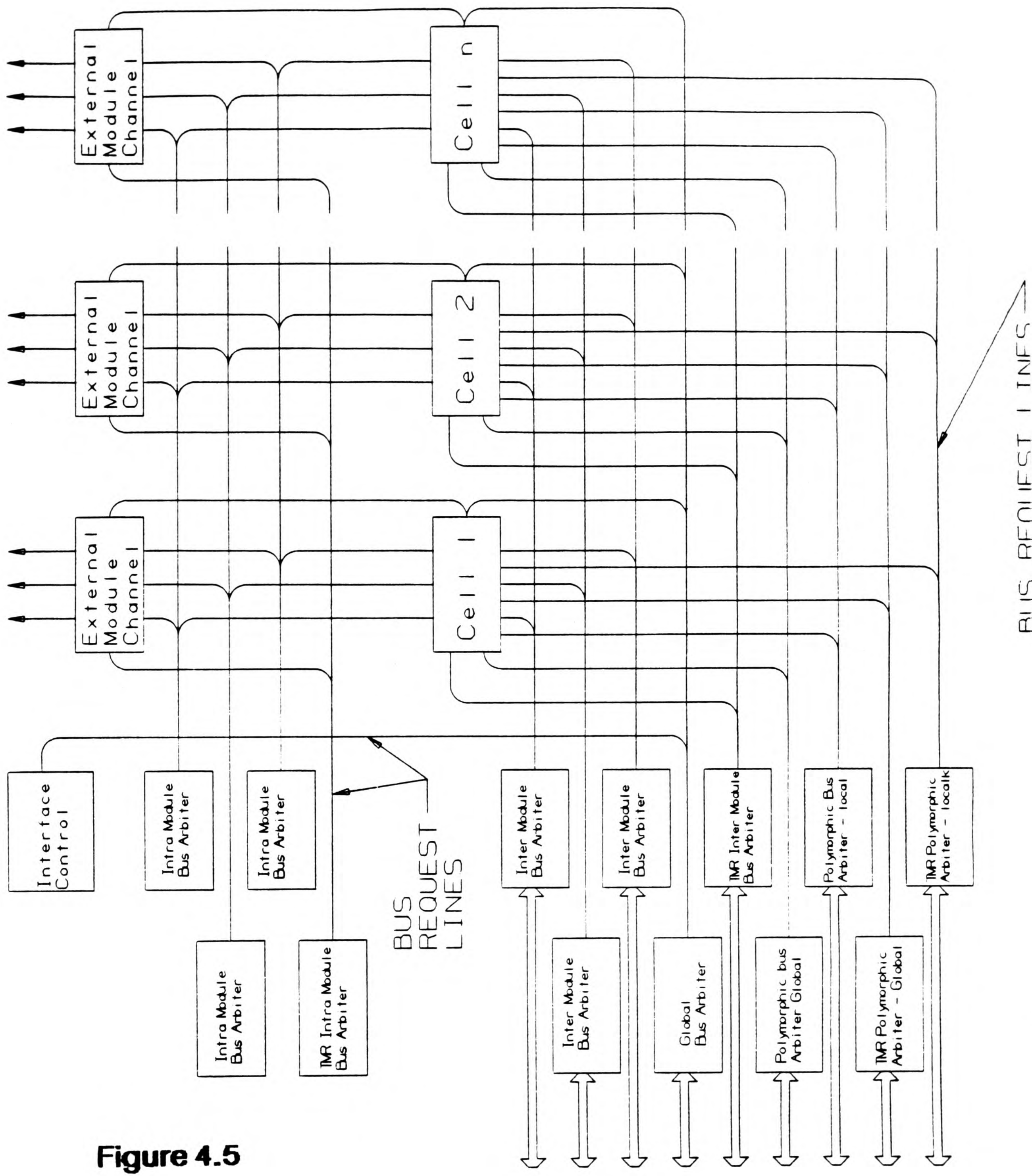


Figure 4.5

4.6.1 Distributed Daisy Chain

A distributed Daisy Chain may be constructed from a centralised one by omitting the Bus Busy line and connecting the common Bus Request line to the "first" Bus Avalable. A cell requests service by raising its Bus Request line if the incoming Bus Available is low. When a Bus Available signal is received, a cell that is not requesting the bus passes the signal on. The first cell which is requesting does not propagate the Bus Available, and keep its Bus Request up until it has finished with the bus. Lowering the Bus Request lowers the Bus Available if no successive cells also have high Bus Requests, in which case the "first" cell that wants the bus gets it. However, if some cell "beyond" this has a Bus Request, control propagates to it. Therefore allocation is on a round-robin basis, figure 4.6.

4.6.2 Distributed Polling

When a cell is willing to release the bus, it puts an address on the polling lines and raises Bus Available. If the address corresponds to that of another cell which requires the bus, that cell responds with Bus Accept. The former cell drops the polling and Bus Available lines and the latter cell lowers the Bus Accept and begins using the bus. If the polling cell does not receive a Bus Accept, it changes the address according to some allocation algorithm and tries again. This method requires that exactly one cell be granted bus control when the system is initialised.

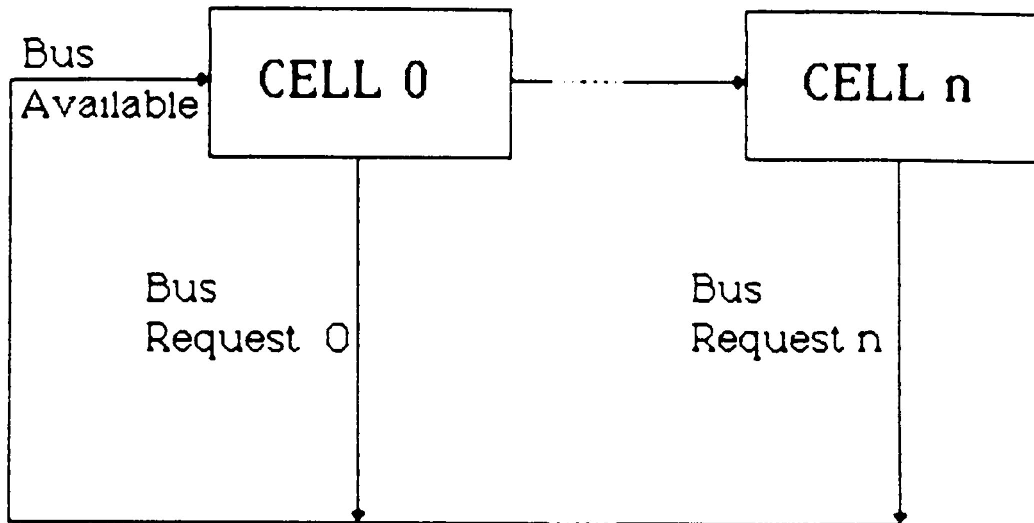


Figure 4.6 Decentralised Daisy Chain

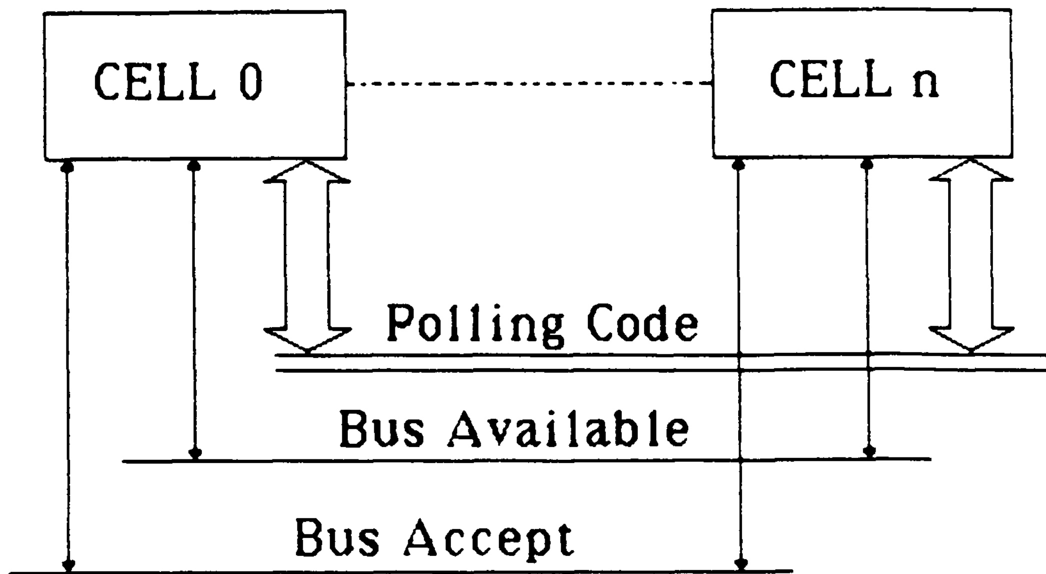


Figure 4.7 Decentralised Polling

The system uses more hardware due to every cell having the same allocation hardware as the centralised system. However, this improves reliability as the failure of a cell does not usually degrade bus operation, figure 4.7.

4.6.3 Distributed Independant Requests

Any cell needing the bus raises its Bus Request line, which corresponds to its priority. When the current bus master releases the bus by dropping the Bus Assigned, all requesting cells examine all active requests. The cell which recognises itself as the highest priority requestor obtains control of the bus by raising the Bus Assigned. This causes all other requesting cells to lower their Bus Requests. The logic in the distributed system is simpler than that of its centralised counterpart, but the number of lines and connectors is high, figure 4.8.

4.7 Universal Arbiter

The uncertainty that exists in the physical image of fifth generation machines, requires generality in the topology of a 'universal arbiter'. Clearly system bus architectures of various types have to be considered in order that the arbiter be integrated into a single integrated circuit.

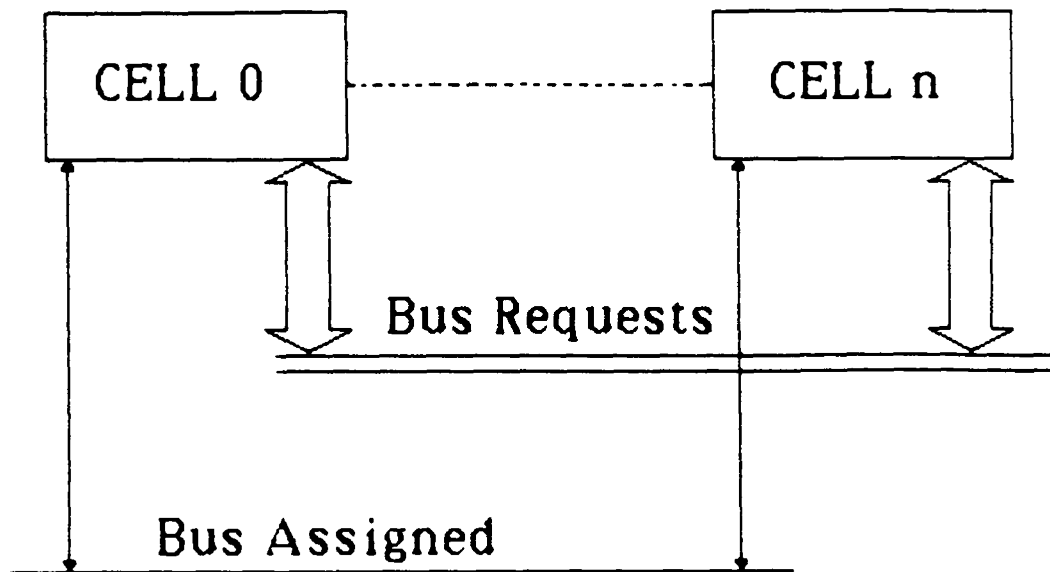


Figure 4.8 Decentralised Independant Requests

By providing a distributed but universal arbiter design, the overall bus arbiter design becomes more complex, as part of the design must cater for a priority based architecture. As an example of this consider an equal priority general design. In reality no such design exists, as a simultaneous demand, e.g. two simultaneous bus requests, are conflicting and hence require arbitration.

The design topology illustrates a hierarchical arbitration structure, Figure-4.9. Conceptually, each bus request has equal priority, within bus arbiter 1 for example. Similarly, bus arbiter 2 and bus arbiter "n" have equal priority in the centralised, or root arbiter. The centralised arbiter is in effect an overall arbiter to the other arbiters below it in the hierarchy. Although only 24 inputs are shown; the hierarchy is infinitely extendable with seemingly equal priority, by organising the interconnections as a hierarchical tree structure.

By comparison the daisy chain, i.e semi-linear priority scheme of figure 4.10, makes line 8 of bus arbiter 1, the lowest priority, and line 1 of bus arbiter "n" the highest priority. It is conceiveable from this topology that in a long daisy chain; the lowest priority may take a long time to get served, due to repeated requests by higher priority requests.

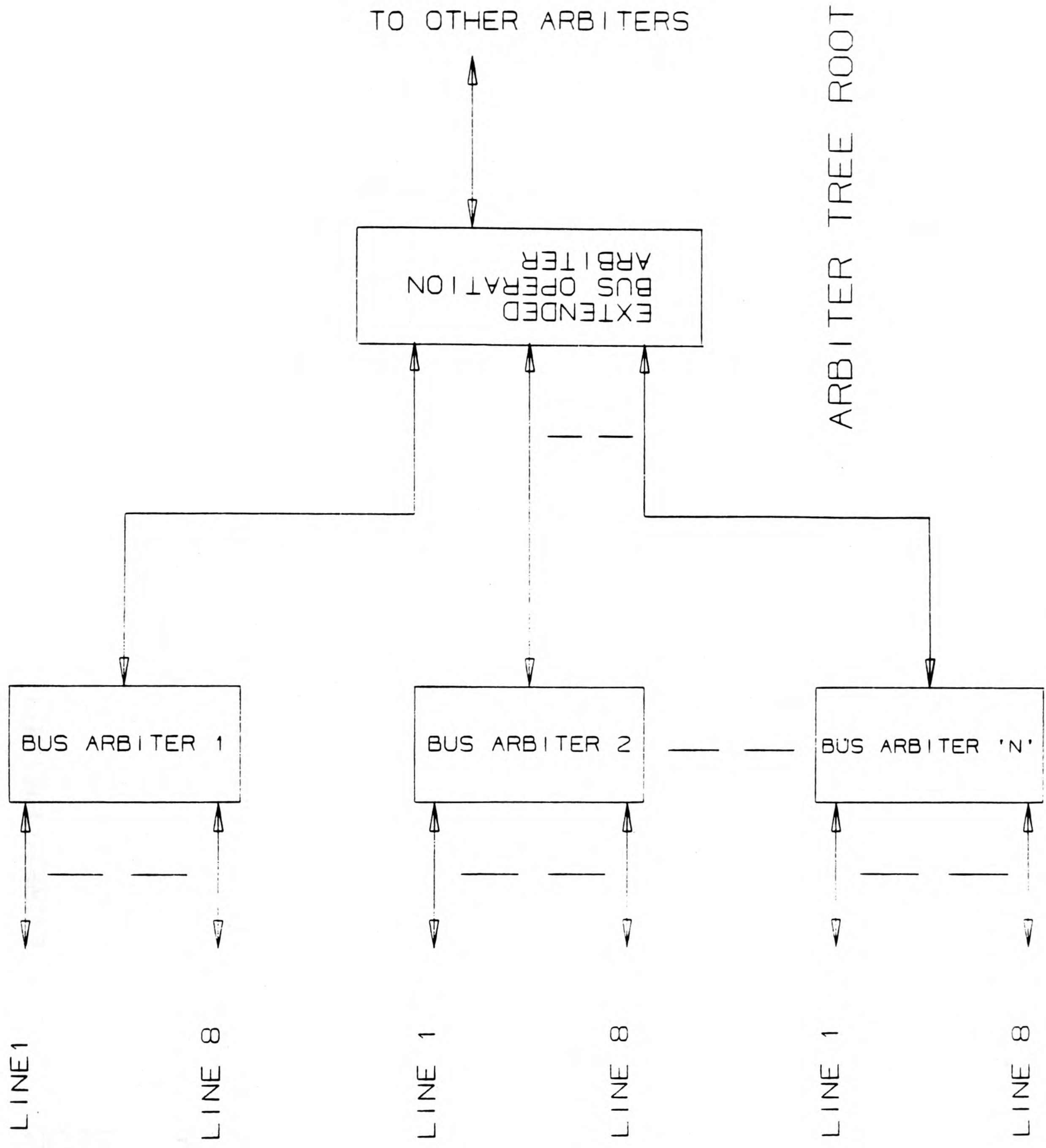


Figure 4.9

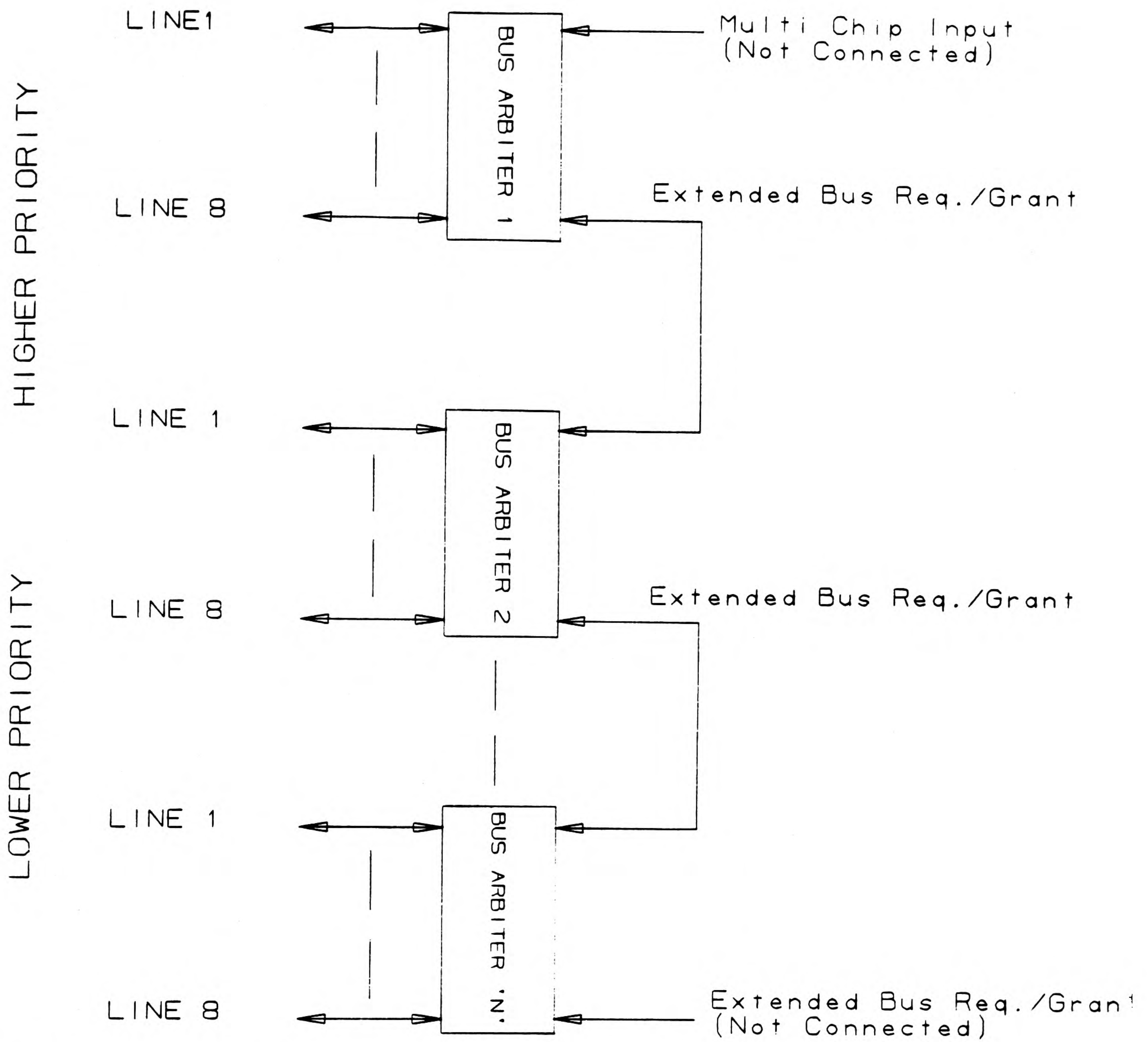


Figure 4.10

In terms of fifth generation multi-processor systems; the daisy chain has certain advantages over an equal priority design. The most important of these is the ability to give

a higher priority to the user and the operating system. In reality a totally daisy chained system is impractical as all user oriented modules should have equal priority. In this case a mixture of daisy chaining and equal priority topology may be accepted as providing the degree of operation required, for rapid intervention to the operating system, and equal priority for the user modules. It is also conceivable, that in fifth generation operating systems, that bus arbitration logic is visualised as shown in figure 4.11., where, in conflict conditions, line 1 has a higher priority over line 2 in the system arbiter. Similarly, the nearer to the lower numbered lines a module is connected, the higher the conflict priority.

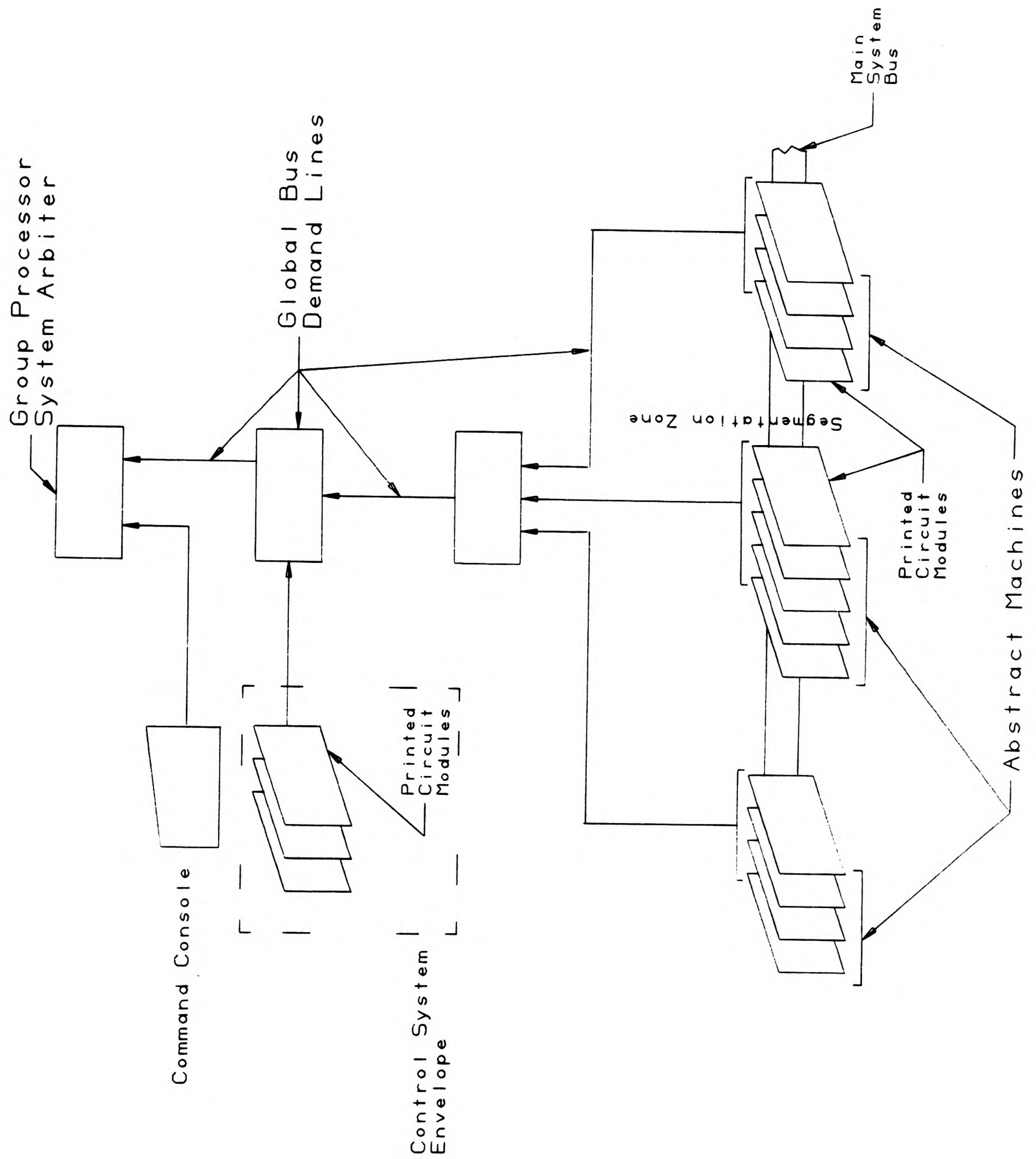


Figure 4.11

4.8 Summary

In this chapter, the architecture of a computer system has been presented as a functional module structure that represents a hardware/software environment for the execution of user programs. This presentation has been orientated toward a true fifth generation machine architecture, through the consideration that machines will be constructed from 100's of thousands of processing cells. These separate, but coupled cells, need a flexible and extensible bus arbitration network. Such a network has been outlined as a first stage in understanding the complexities that exist when the burden of design and implementaion of a fifth generation machine is placed on the computer architect.

When complex bus structures are studied in depth; there is a realisation that long and involved research effort is needed into the wider aspects of bus structures. The interaction of operating systems and the cell design will result in a much closer working attitude between the computer architect and software engineer. Additionally, the practicality of these systems can only be gauged through real design efforts resulting in VLSI cells being produced.

References - Chapter 4

1. FARBER, G., "A Decentralised Fair Bus Arbiter", EuroMicro V7 N1 1981

2. KOVALESKI, A.B., "High Speed Bus Arbiter for MultiProcessors," I.E.E.E. Proc. V130 N2 1983

3. NADIR, J. AND MCCORMICK, B., "Bus Arbiter Streamlines Multi-Processor Design," Computer Design V19 N6 1980.

4. PETRIU, E., "N-Channel Asynchronous Arbiters Resolves Resource Allocation Conflicts," Computer Design V19 N8 1972.

5,6. PLUMMER, W.W., "Asynchronous Arbiters," I.E.E.E. Transactions on Computers V21 N1 1972.

CHAPTER FIVE

CHAPTER FIVE

THE GROUP PROCESSOR CONCEPT

5.0 Introduction

This chapter introduces the Group Processor System [9] as viewed from a single-user, multi-user and operating system viewpoint. It also describes the cell, module and bus design in detail. The work described in this chapter draws heavily on the work undertaken by Quick [10] and is a summary of the work undertaken on the Group Processor System. The terminology used by Quick is maintained in this chapter.

5.1 High Level System Description

The Group Processor concept is not a total system design, but an environment for process execution. The realisation of the architecture for a computing system is based on systems principles. That is, the complete system is built up of sub-systems of common elements, which are cells and modules. Figure 5.1. shows the functional components of the Group Processor System.

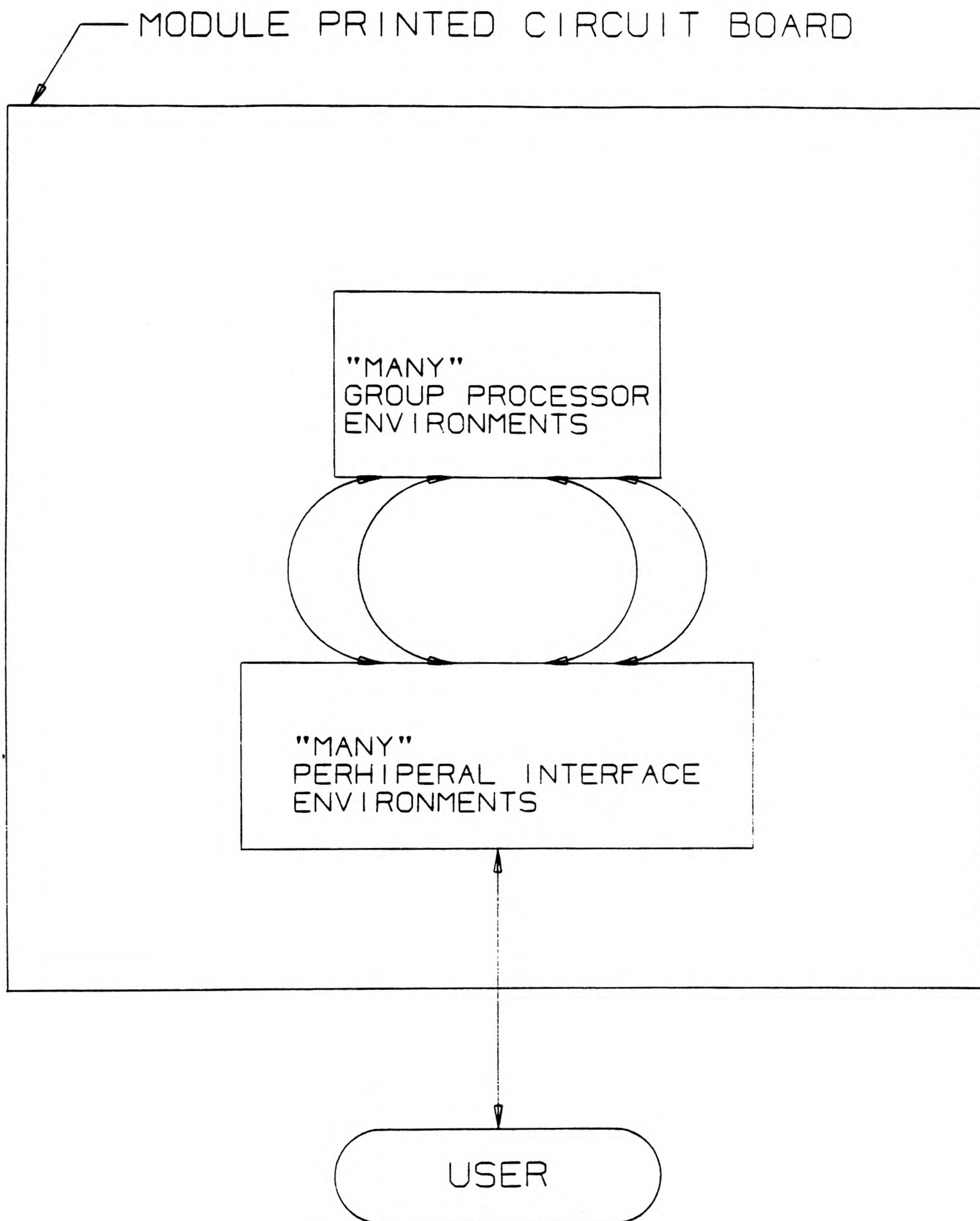


Figure 5.1

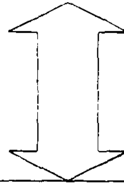
The user interfaces to the Group Processor through a dedicated software based frontend processor. The frontend processor performs many operations on objects (programs and data); e.g. editing and language translation, during the development of user programs. The main software features supported within the software based frontend processor, termed the 'peripheral interface environment', is the ability to schedule and transmit the communication between user terminals and Group Processor Modules that make up the Group Processor System.

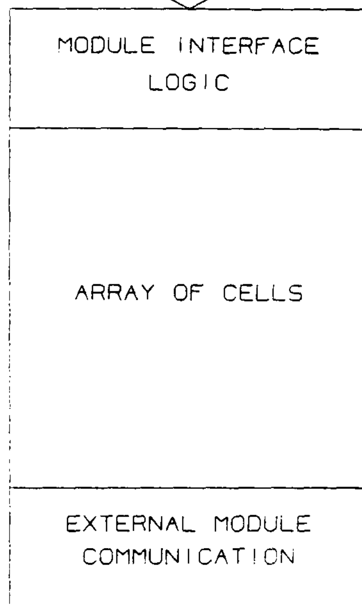
5.2 Peripheral Interface Environment

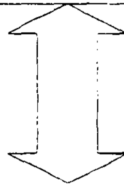
The peripheral interface environment is constructed from the same module architecture, shown diagrammatically in figure 5.2, as the Group Processor Environment.

Such a uniform hardware design is a feature of the Group Processor System and minimises the variations of printed circuit board design, which results in lower production costs. The cellular structure of the module, illustrated in figure 5.3., executes it's processes in the cells in the Group Processor environment. The executing process within the cells of each module is the only difference between the Group Processor environment and the peripheral interface environment, and as a result both environment types are interchangeable.

COMMUNICATION TO OTHER MODULES







COMMUNICATION TO OTHER USERS

Figure 5.2

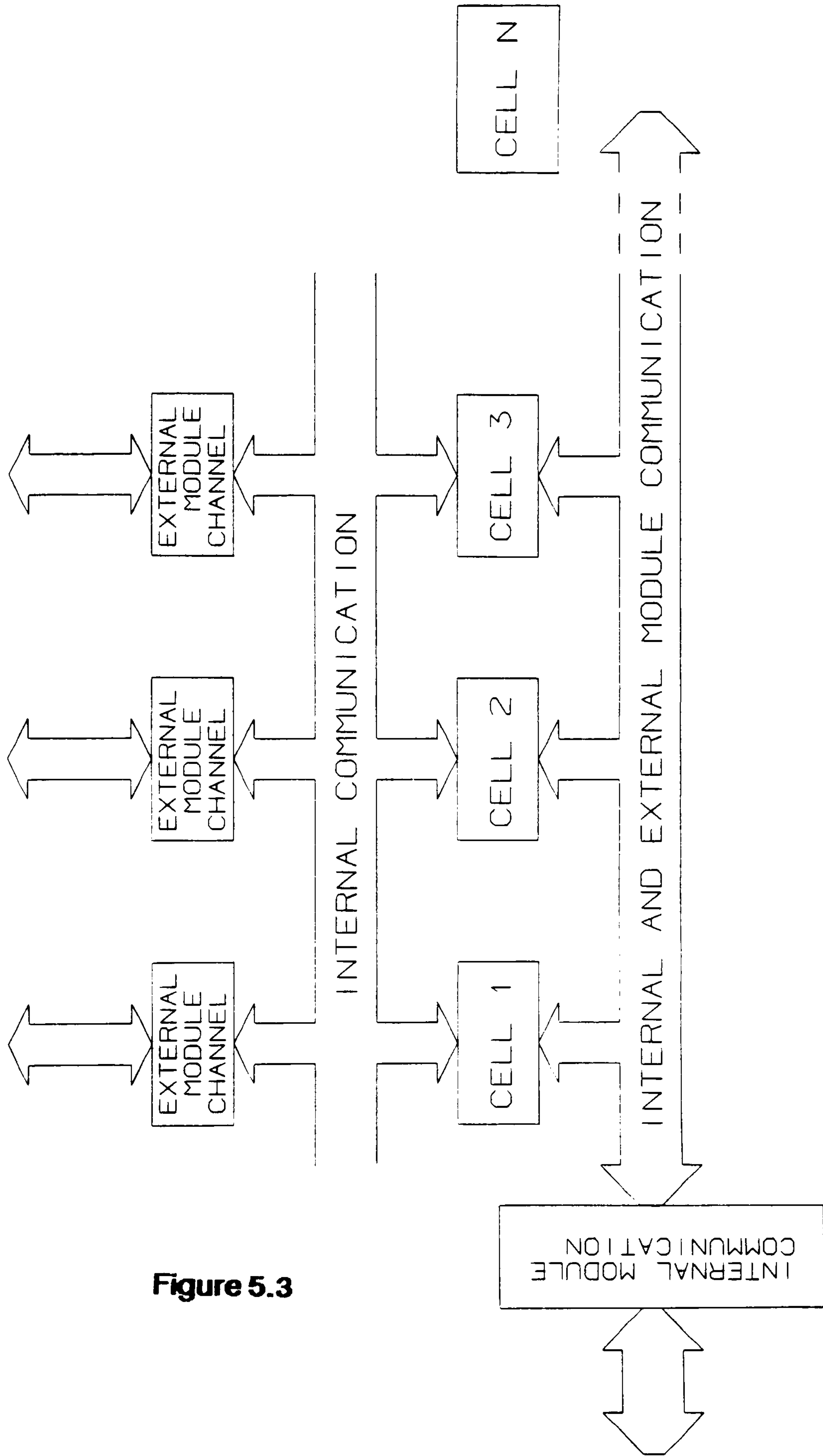


Figure 5.3

The architecture of the Group Processor environment is identical to that of the peripheral interface environment, and consists of identical parts programmed to accept tasks which perform operations on objects.

5.3 Group Processor Environment

The operations performed within the peripheral interface environment and group processor environment, which correspond to either a frontend or backend system requirement, are executed by groups of cells configured to work groups, are achieved by allowing each member to communicate with other members via the common bus structure. This structure consists of a number of functionally dedicated buses that are available for use by any cell.

5.4 Module External Input Output

The internal data routing of the inter-module bus does not perform any input-output to the system user, as can be seen from figure 5.4., although the system may be configured so as to provide such a function using dummy modules, which are then directly linked to the user. The input-output described in this section concentrates on the interface between the Peripheral Interface Environment and the user terminal.

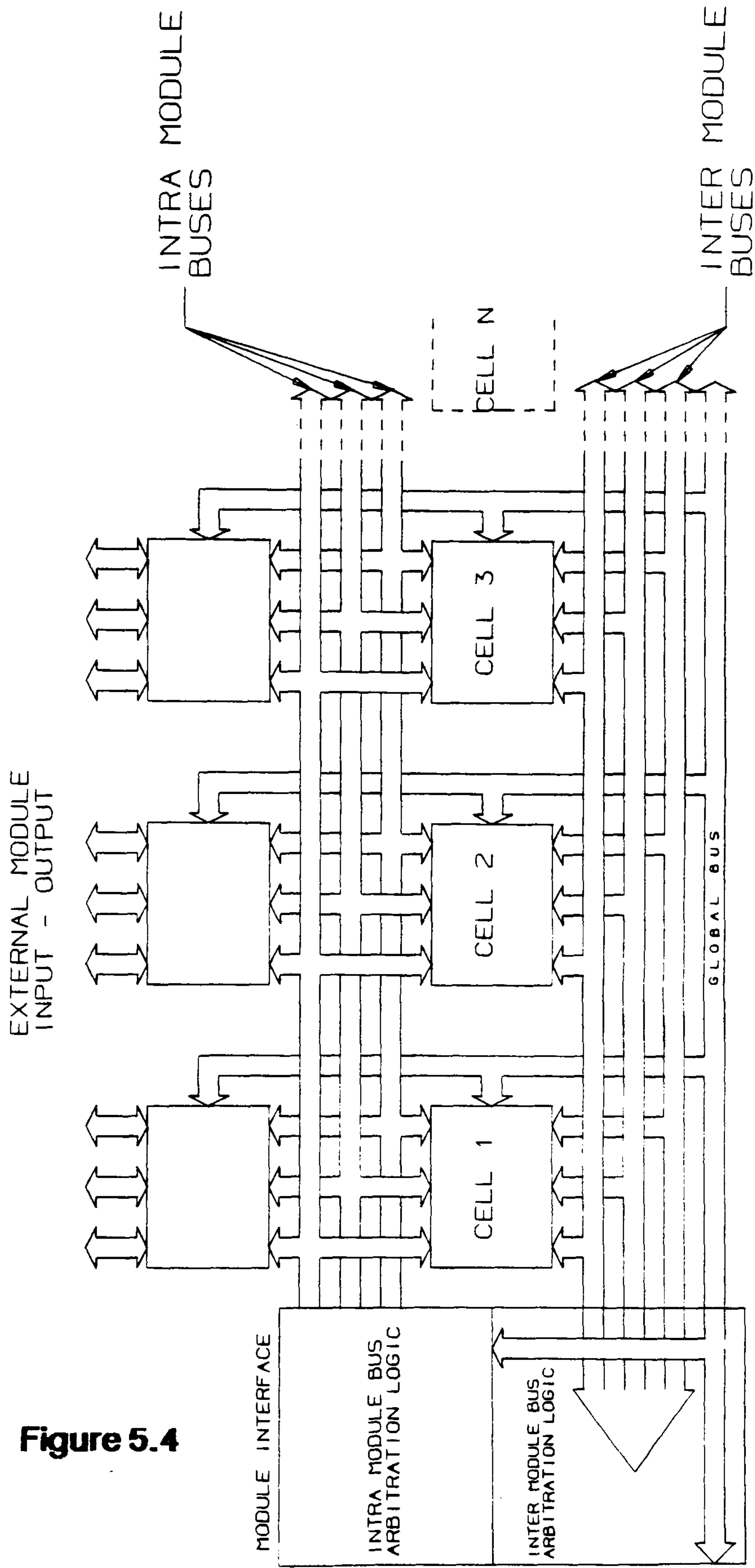


Figure 5.4

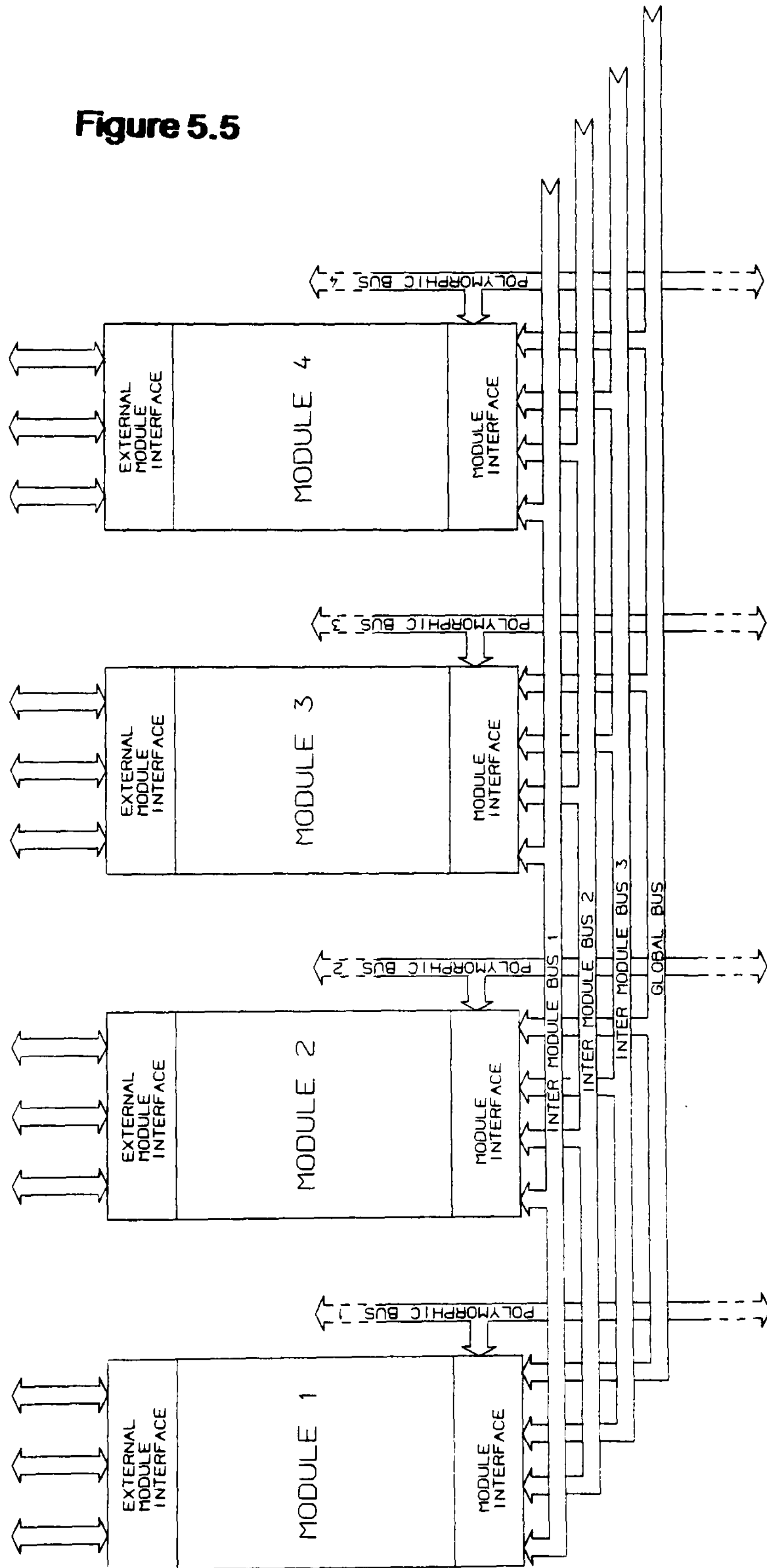
The architecture of the cell and the module interface enables the various internal systems to communicate via the numerous system buses. The communication to the outside world takes place through an external access port, mounted on the module. This access port is a cell, identical to the other cells on the module, but the three inter module bus ports are dedicated to external module communication.

The external cell architecture is illustrated in figure 5.5., where the cell's three intra module bus ports are physically connected to the intra module bus, the inter module bus ports are connected to the segmented inter module bus. The global bus is connected to all cells. A personality pin indicates what function the device is dedicated to, i.e. cell or external module I/O processor.

5.5 Logical Bus Structure

The logical coupling of the various software work groups relates to the input and output to each process. This is reflected in the actual communication on the various system buses. For example; where one work group resides in one module and communicates with another work group in another module, a bus functionally dedicated to inter module transmission provides the logical bus communication structure.

Figure 5.5



Where both work groups reside within the same module boundary, the logical bus structure does not require the services of the inter module transmission system. In this module, a localised bus system conveys the logical information. The operation of the communication system for updating the various work groups that use the system information, e.g resource schedulers, utilize a dedicated bus for global transmission of data objects. This bus may, if required, be used for the global query of the data stored in each cell.

The buses supported here are therefore related to logical communication between processes, but influenced by the physical location of the communicating processes. The three inter module buses are capable of being scheduled as to pass-on, distribute, or block, terminate, communication, i.e to provide partitioning zones of bus segmentation.

Consider the system bus structure of figure 5.6, this illustrates that all modules are connected to the common inter module communication system. If however, modules 1 and 2 were segmented from cells 3 and 4, then the inter module communication bus system would be physically broken between modules 2 and 3. This enables parallel access of the inter module bus for write operations by any one cell, in each segmented zone.

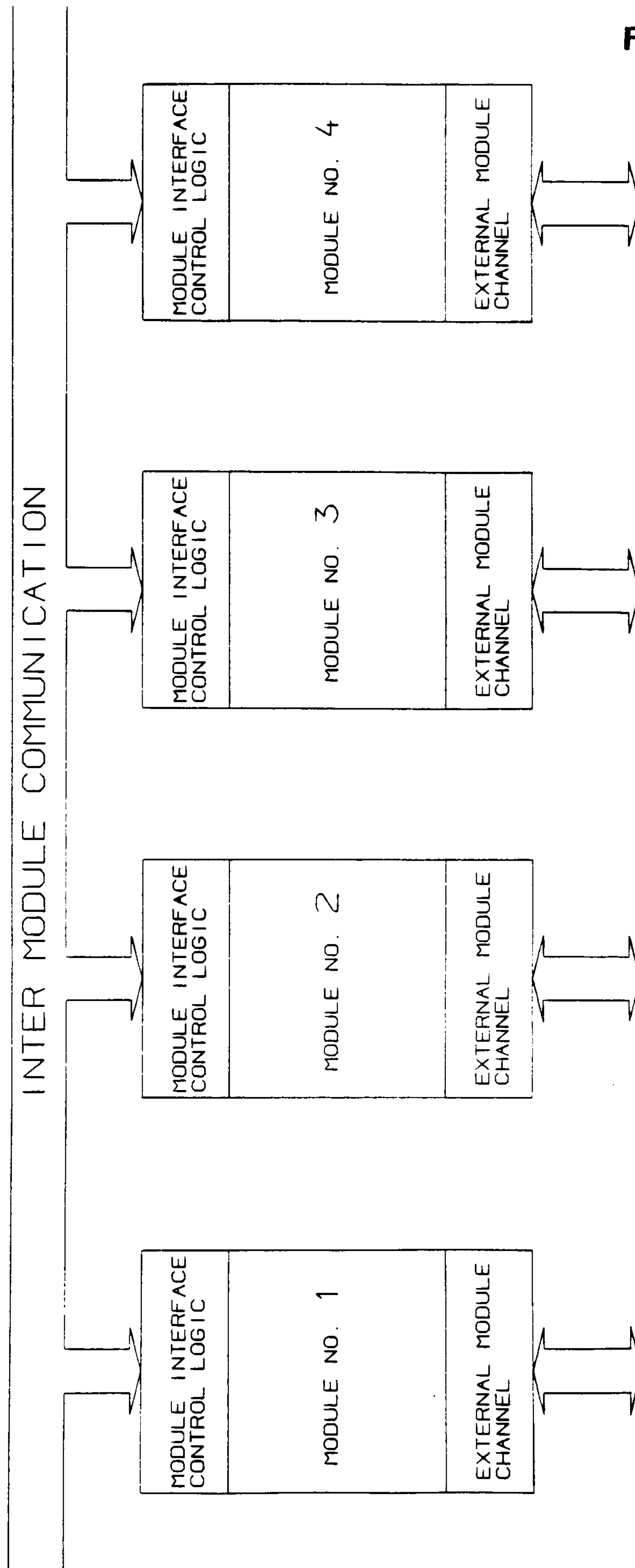


Figure 5.6

The scheduling of the three segmented buses is controlled by the operating system and performed by the module interface. Each bus has its own control intelligence which enables three levels of communication to take place simultaneously, corresponding to each of the three buses.

Entry to a module interface is by single entry to each controller, thereby providing parallel access to the module interface, in inter cell communication.

5.6 Bus Structures

In most of the multiprocessor configurations developed in recent years [2,14] bus contention has been a major system overhead. In this area, the proposed modular design enables parallel access of the system bus through the use of inter module bus segmentation, as seen in figures 5.3 and 5.6. The proposed bus structure overcomes the two basic problems isolated by the Data Flow Group at U.C. Irvine [3], namely data localisation and concurrent bus access.

Communication buses are used to transfer control, timing and data signals between the module sub-systems. These buses are designed to provide physical and electrical isolation and provide for the simple addition of module units.

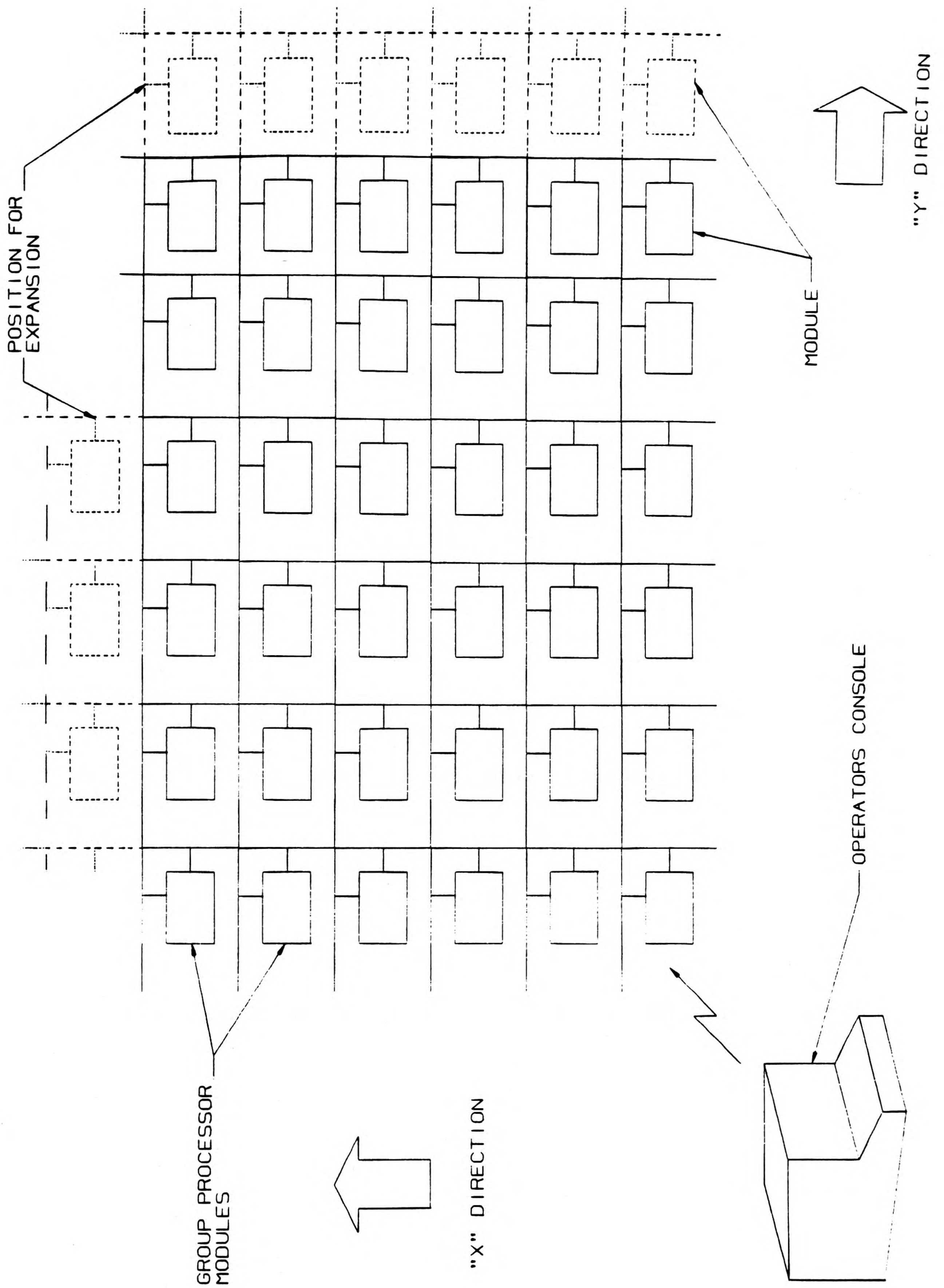


Figure 5.7

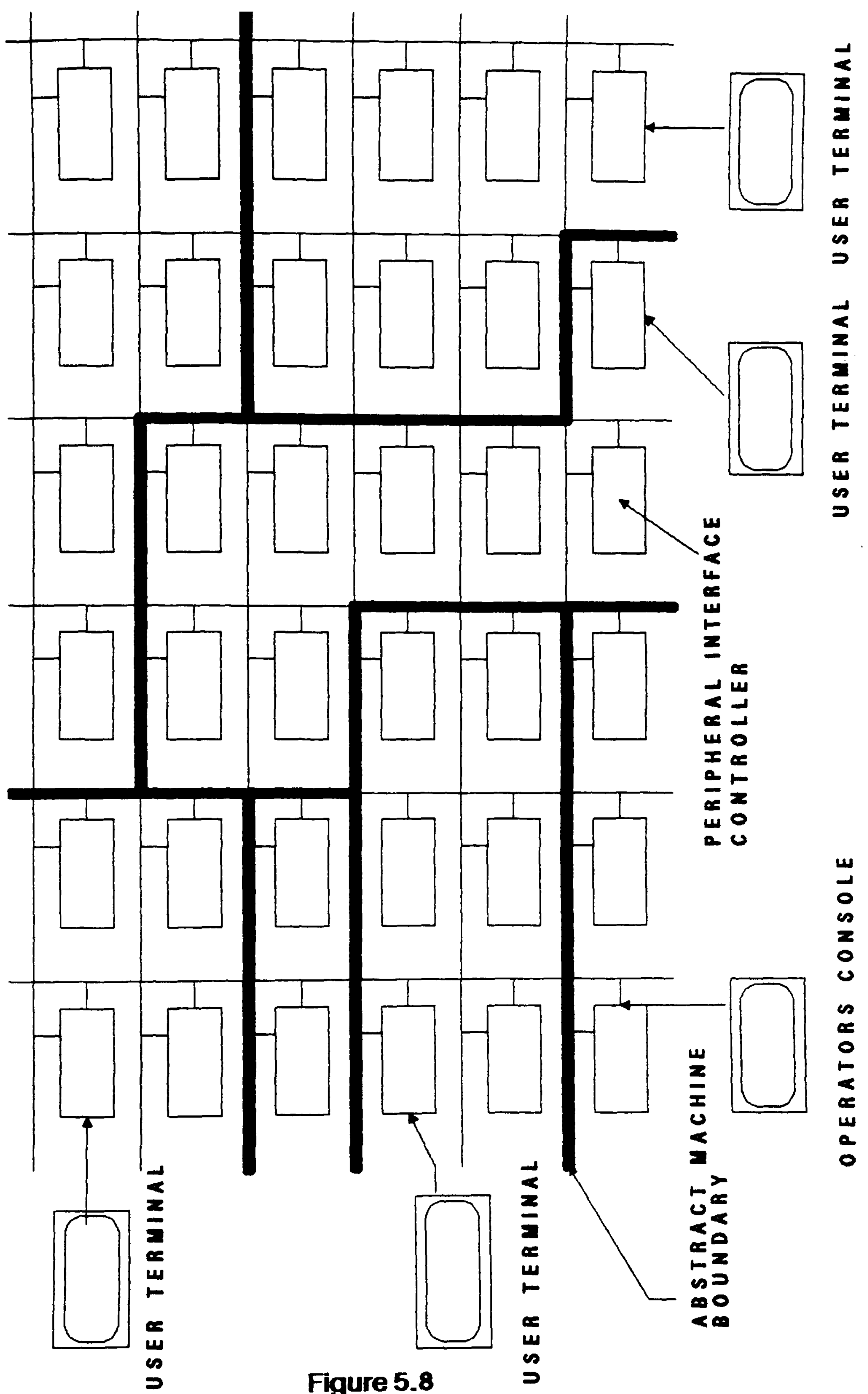


Figure 5.8

The transmission of data between modules is the function of the inter module transmission system, which can connect modules in a number of unique ways. Dummy modules may be incorporated, thereby enabling peripherals to be interfaced directly to the system bus, and not attached to a module input-output channel.

5.7 Inter-Module Bus Structure

Cooperation between work groups is achieved by allowing each member to communicate to others via the common bus structure. This structure consists of a number of functionally dedicated buses that are available for use by any cell.

These buses are:

(a) Intra-Module Bus

(b) Segmented Inter-Module Buses
(in "X" & "Y" directions)

(c) Non-segmented Global Bus

(d) Control Bus/Lines

For a minimal system structured on reliability concepts; seven segmented buses are incorporated in this Thesis, although only one is required as a minimum configuration. The control bus is transparent to the user, as its function is to aid the electronic operation of the system [6]. The actions on the control bus are detailed in the next two sections.

The inter module bus structure is "set up" by addressing the module interfaces using the global bus. There are three basic hardware set up structures:-

1) Distribute
2) Block
3) Unique

The basic distribute and block structures enable the hardware to pass data along the bus, distribute, or create separate, segregated, buses.

When the system is initialised, by a reset signal, the bus segmentation switches are set in a high impedance mode, i.e the bus segmentation switches are in an off state. No inter module dialogue is permitted during the reset phase, as all message requests are inhibited until the module interface sucessfully performs an error checking routine.

When a module requests the inter-module bus, a request is sent to the bus controller unit, which assesses whether the path from the source to the destination is available. If not available the request is queued, otherwise the bus is allocated.

A typical section, e.g. modules 1 to 5 in figure 5.5, illustrates the inter module bus structure. The broken arrows in this figure serve to indicate that a segmentation zone exists at that point, i.e. between module 1 and 0, and module 5 and 6.

5.8 Single User Environment

The single user environment consists of the hardware, as illustrated by boxes in figure 5.7, and the software support structures. This environment supports the user's work space in which the programmer executes the program.

The basic integration of the general system of figure 5.7 into an extensible computer system, is shown in figure 5.8 as a single user hardware architecture. This architecture enables segmentation schemes to be set-up in both X and Y directions.

5.9 Multi-User Environment

The single user environment of figure 5.7 may be extended to provide a multi-user system. In a conventional multi-user time share system, processor time is shared (multiplexed) between users, whereas the multi-user Group Processor System does not time share, but shares the processor space among many users in the same time interval. This results in a faster user response time compared to conventional time multiplexed system design as each user has his own machine, and not the illusion of being the only user on the machine. Although the Group Processor is a space share design, the system may support a time share mode of operation.

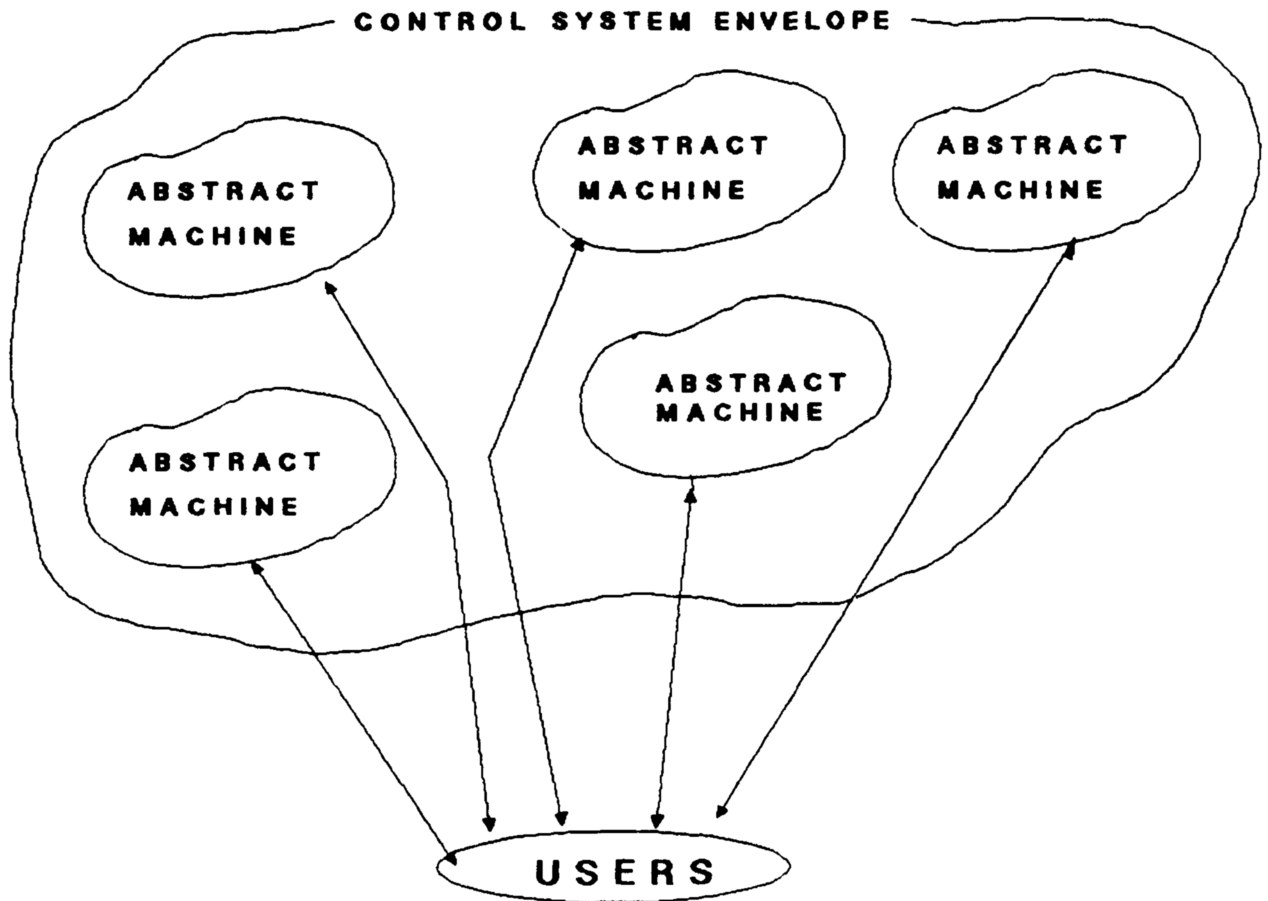


Figure 5.9

In figure 5.9 the various users' work space environments are shown as an enclosed spaces. These are not physical boundaries in the general sense of the word, only conceptual, as each user's abstract machine may grow into anothers user's free work space, simply by extending one user's segmented zone, and contracting adjacent zones. However, each user should always maintain the very minimum of resources, called the first level processor.

As with the single user concept, the multi user system may be extended in both X and Y directions. Also, when a dynamic Group Processor system is visualised, the user terminal may be either the top-most level as the Group Processsor or peripheral interface environment. The program execution space may then expand into lower levels, but should occupy contiguous dynamic address spaces, in order to simplify storage management.

5.10 The Group Processor Operating System

The operating system of any uniprocessor machine, and a very bounded multiprocessor machine for that matter, is inherently sequential in nature. Bounded multiprocessor systems have tended to be a "rehash" of the uniprocessor machine [15], with the corresponding rehash of the original operating system. The main reason for this has been greatly reduced development costs for the operating system.

The ideal environment requirement for the support of the Group Processor operating system image is one that is

truly dynamic. That is to say, user's abstract machines are given the power to allocate, or de-allocate, resources at will. A dynamic system can be viewed as a decentralised environment in which there is a control component, and a centralised arbitration mechanism, but no central control. The only known related system is Medusa [7,4]. STAROS and the now dismantled distributed computing system at the Universty of California, Irvine [12] are too far removed from utilising the architecture of the Group Processor System.

The Group Processor is a loosely coupled logical structure and, because of its looseness, to maintain system reliability the operating system must maintain 'tight' control. The interaction of processes may be strictly controlled through the generous use of interface, or context changing processes, possibly with a corresponding reduction in execution time. When context is changed, say from user to operating system process, it is due to the action of the 'context changer', as processes should not communicate implicitly.

5.11 Distributed Control

In this Thesis "a system is distributed if it consists of many small physically independent entities, that cooperate closely by the receiving of objects, which in turn may create further objects, which offer a single service image to other systems, including itself". By treating the execution of the user's program as the execution of disjoint

processes (or tasks), a non-hierarchical structure of currently active application, control and service routines, may be realised. The use of context changers allows for the dynamic activation of related, as well as unrelated, logical processes. This is achieved by threading processes together by the passing of objects through the logical routine templates, thus building up the required application and operating system context states.

A conceptual relationship between the common control software to the user's system is shown in figure 5.9. The main executive program, i.e. the control system envelope, dispenses the control of resources following system reset procedures, to software created abstract machines. The abstract machines, in turn, dispense control to their many abstract first level processors, thereby creating a fully distributed environment. It should be remembered, that every active process is an abstract processor whether dedicated to user or operating system functions.

The single user system maintains a one to one mapping between the user's work space and the system control terminal as the user terminal. A many to one relationship is supported in the multi-user group processor system machine, because each user's work space is an abstract machine, which maintains its own independent link to each user's terminal image.

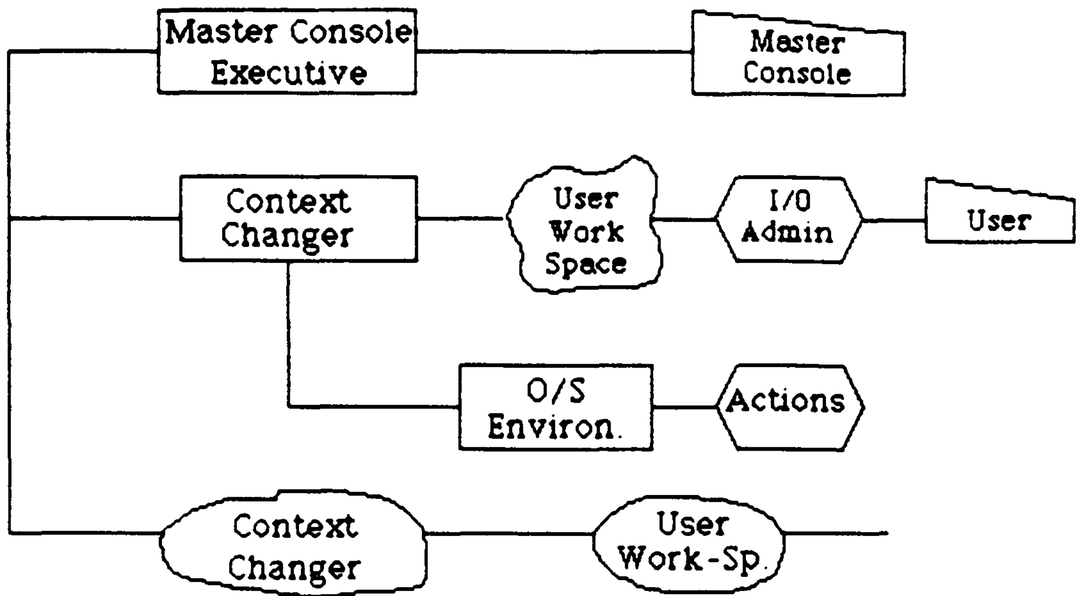


Figure 5.10
Distributed Operating System

There also exists a many to one relationship between the abstract machine and the control system envelope in the multi-user mode. The multi-user machine consists of many single user machines residing side by side (see figure 5.8). As there are no physical boundaries between these machines, only segmentation switches, the user's work space may freely expand into adjacent free modules during processing.

5.12 High Level Operating System Representation

Due to the Group Processors inherent flexibility and dynamic nature, the operating system is difficult to represent diagrammatically. Figure 5.10 gives a possible snapshot of the system configuration, where the various activities are being executed in parallel, within their own hardware environment. This is significantly different to the multi-programmed environment of the hierarchical uniprocessor operating system.

5.13 Group Processor System Summary

This chapter has been dedicated to the functional hardware description of a flexible general purpose cell, module and communication architecture. This communication structure is simpler than that found in Cm* [16] yet being extensible. In a commercial machine, Quick suggests certain factors must be considered relative to their functional application requirement, and integrated into the module, or cell architecture, and hence become implementation

considerations. It is the intention of this Thesis to investigate these application architectures so that a clearer understanding of system performance may be established.

References - Chapter 5

1. MCMILLEN, R. AND SIEGEL, H. J., "Performance and Fault Tolerance Improvments in the Inverse Augmented Data Manipulator Network" IEEE 1982.

2. ENSLOW, P.H., "Multiprocessor Organisation - A Survey", Computing Serveys V9 N1 1977.

3. GOSTELOW, K.P., AND THOMAS, R.E., "Performance of a Dataflow Computer", UC Irvine TR 127a.

4. JONES, A. AND GEHINGER, E.F., "The Cm* Multiprocessor Project: A Research Review", CMU-CS-80-131.

5. OPPER, E., MALEK, M., AND LIPOVSKI, G, "Resource Allocation in Rectanglar CC-Banyans", ACM 1983.

6. OSBORNE, A., "Introduction to Microcomputers Vol.2 Some Real Products (8 & 16 bitdevices)", Osborne/McGraw-Hill 1981.

7. OUSTERHOUT, K.J., "Partitioning and Cooperation in a Distributed Multiprocessor Operating System: MEDUSA", CMU-CS-80-112.

8. PARKER, D.S. AND RAGHARENDRA, C.S., "The Gamma Network: A Multiprocessor Interconnection Network with redundant Paths.", IEEE 1982.

9, 10, 11. QUICK, G.E., "The Group Processor Approach to Multiprocessor Arcitecture", Ph.D. Thesis UC Swansea 1982.

12. ROWE, L.A., "The Distributed Computing Operating System", UC Irvine TR 66.

13. SHEN, J. P. AND HAYES, P. J., "Fault Tolerance of a Class of Connecting Networks", IEEE 1982.

14, 15. SWAN, R.J., "Cm* - A Modular Multiprocessor", AFIPS V46.

16. SWAN, R.J, "The Switching Structure and Addressing Architecture of an extensible Multiprocesor Cm*", CMU-CS-78-138.

17. AL., A. GOTTLIEB ET, "The NYU Ultracompuer--Designing a MIMD, Shared Memory Parallel Machine.", IEEE 1982.

CHAPTER SIX

CHAPTER SIX

THE SIMULATION ENVIRONMENT

6.0 Introduction

The performance evaluation of a computer system plays an important part in determining the success of the system. This chapter introduces the various methods of estimating system performance and explains in depth the 'Simulation Environment' for the Group Processor System.

The output from the simulator is presented in graphical form. This output quickly allows potential system architects an appreciation of the performance limiting factors associated with various bus interconnection and scheduling schemes. The graphs are a general representation of bus performance and should be used by system architects as a guide to system performance.

6.1 The Modelling Approach

Performance measurement techniques can be grouped under the characterisation of models. These models, or benchmarks, can be studied instead of the entire workload, thus reducing the cost of evaluating a wide range of architectures. The models can be divided into four areas:-

1) Synthetic benchmarks

2) Live benchmarks

3) Simulation

4) Mathematical modelling

6.1.1 Synthetic Benchmarks

A synthetic benchmark is a program which within it has the ability to use the resources of the system under test, according to specific parameters, i.e read 1000 lines of text from a specified disk file.

The advantage of such a benchmark is that it is easy to construct, and relatively easy to convert from machine to machine. However, it has the major disadvantage in that it is very difficult to demonstrate that a synthetic benchmark is truly representative of the entire system workload.

6.1.2 Live Benchmarks

A live benchmark is a set of programs drawn from the current user's workload, and is taken as to represent the entire workload. The advantages of such benchmarking is that is uses 'real' programs. The disadvantages include the cost of porting the code to other machines and ensuring that the benchmark represents a future typical workload.

6.1.3 Simulation

In the simulation approach, a program simulates the hardware and operating system of the target system. The simulator is fed data, which are taken to be the workload being used for the evaluation.

The main advantage of simulation is also its major disadvantage, namely its inherent flexibility. Simulation is particularly useful in the prototyping of new machines, or the evaluation of systems that have yet to reach the detailed design stage. The main disadvantage is that the system which will run the users workload is not being directly tested, only the projected capacity of the machine.

6.1.4 Mathematical Modelling

Some attempts have been made to model system performance by the techniques of mathematical programming. The prospects for the general use of such methods do not appear good. Many dynamic interactions in systems, e.g user interacion and speed of software execution, are difficult to represent mathematically, Sharpe [3].

6.2 The Case for Simulation

There is some controversy in respect to the relative merits of simulation and benchmarks. Howard, [4], draws the following conclusions:-

"The fact of the matter is that there is a place for both approches. Benchmarks have their place in situations involving upgrades and replacement. Benchmarks are also important where performance of the workload is of primary consideration. Simulation may be preferred where new applications or processing approaches are involved. Simulation is also preferred where the proposed hardware is not available for benchmarking".

Therefore in this Thesis, as the proposed Group Processor System hardware does not exist, simulation is the only feasible solution. Several sucessful simulations of proposed hardware have been carried out. Two notable examples being the Data General MV8000 and the simulation of Cm* on the C.mmp machine at Carnegie Mellon University [2,5].

6.3 Computer Structure, Resources and Application

The Group Processor System consists of a number of units each of which is capable of operation at specified rates. These units represent the resources of the system. The Group Processor is examined from the point of view of these resources rather than the view point of the functions performed by the component units. In the Group Processor System, resources are categorised into types and the relations between resources have been qualitatively examined.

6.3.1 What is a Resource?

The various units which comprise the Group Processor System represent the resources; these resources are measured by one or more parameters, such as execution rate or capacity. Thus memory has two such resource parameters: access time and memory capacity.

In this Thesis the term 'resource' is taken to be the one defined by Beizer [1]:-

"The term 'resource' will be used to mean `resource parameter', where the specific parameter is obvious from the context".

A resource may be created by software. These are called 'synthetic' resources; i.e Processing time can be subdivided into microseconds or other convenient time slices, creating a processing resource which is available at a rate of one million per second. Ultimately all synthetic resources must be expressed in terms of physical resources.

Physical resources have the additional property that they are available only in a finite quantity. Resource parameters can only have positive quantities.

6.3.2 Some Definitions.

Utilisation is the percentage of time that a resource is in use divided by the total observation time.

Usage is the quantity of a resource required for a specific task.

Demand Rate is the rate at which requests for the resource are being made.

Relative Demand is the ratio of demand to the available resource.

Saturation occurs when the utilisation equals one.

Efficiency is the ratio of relative demand to the utilisation.

6.4.0 Simulation Architecture

In this section we present the architecture of the simulator model. The simulator is viewed from the architectural viewpoint, not the PASCAL program view.

The architecture of the simulation model is identical to that given in Chapter 5. In addition, a number of alternative bus interconnection patterns are available which enable the simulation of various Group Processor configurations.

6.4.1 System Architecture Components

The Group Processor System consists of six main components:-

6.4.1.1 Buses

The simulator has no representation of any physical bus in the system; rather, a bus is inferred to exist by the introduction of transmission delays built into transfer rates of various subsystem components.

6.4.1.2 Bus Arbiters

For every bus in the system there is a bus arbiter dedicated to the resolution of bus requests for that bus.

All arbiters in the system are identical in structure and function.

6.4.1.3 Cells

A cell is the fundamental processing element in the simulation. In this simulation the cell is not implemented as a real device, rather it is an object capable of controlling its own bus activity and performing local housekeeping.

6.4.1.4 Modules

A module is the circuit board level of the simulator. A module may contain an arbitrary number of cells.

6.4.2 The Group Processor

The Group Processor comprises a number of modules, buses and arbiters.

6.4.3 Bus Interconnection Schemes

The bus interconnection scheme determines the physical form the Group Processor takes at a given time. It is an interconnection pattern for Cells, Modules, Buses and Arbiters.

The interconnection scheme is totally flexible and provides for an arbitrary number of:-

a) Cells per module

b) Modules per system

c) Number and type of bus

6.4.4 Simulator Mechanics

This section deals with the mechanisms by which the simulator executes the hardware model being studied. Its major sections are:-

6.5 Bus Arbitration

All the bus arbiters in the system have the same function and behave according to the following rules:-

1) For distinct bus requests, the arbiters act as FIFO's, queueing requests for bus access in the order in which they arrive.

2) The arbiters provide a method for resolving simultaneous bus demands. When two or more cells, at the same clock tick, request bus access via a common arbiter, then the bus

arbiter actions the cell with the lowest physical address. This provides a mechanism for ultimate arbitration.

6.6 Bus Requsts by Cells

When a cell requests a bus of a given type, it queues a bus request on all such buses in the system. It maintains this state until one of the bus arbiters grants the cell bus access. At this time, the cell drops all of its outstanding requests and becomes bus master.

6.7 Actioning Bus Requests

On gaining control of the bus, the cell starts execution of its current task. After completion, control is returned to the bus arbitration logic for the next request to be processed.

6.8 The Simulator Program

The simulator detailed so far has the basic abilities to perform low level functions associated with the hardware.

The remainder of this chapter details the software which drives these low level features, and provides a complete view of the system simulator.

The Group Processor System simulator environment is constructed in such a way that changes in architecture can be achieved without the need for reprogramming. Architectural modification is accomplished via sets of system parameters which define the layout of the simulator.

6.8.1 System Parameters

The parameters provided by the simulation environment enable minor tuning of the architecture under investigation or major architectural changes to be made. Available parameters fall into these two groups.

6.8.2 Major Parameters

1) The number of cells per module (MAXCELLS)
2) The number of modules in the Group Processor (MAXMODULES)
3) The number of global buses (MAXGLOBALBUS)
4) The number of intra-module buses (MAXINTRABUS)
5) The number of inter-module buses(MAXINTERBUS)
6) The size of global memory (MXMEMORY)
7) The percentage of actual requests to the system (REQUESTCONST)

6.8.3 Tuning Parameters

1) The duration of the simulation (MAXSIMTIME)
2) A loading factor for the simulator (LOADFACTOR)
3) A time range for memory requests (MEMCONT)
4) A time range for I/O requests (IORCONST)
5) A time range for CPU requests (CPUCONST)
6) A time range for operating system calls (OSCONST)
7) A time range for cell activity (PROCESSINGCONST)
8) The probability of a cell failing (CELLFAILCONST)
9) The probability of a module failing (MODULEFAILCONST)

6.9 Main Sections of the Simulation Environment

At the heart of the simulation environment are a set of procedures which generate jobs for the system and subsequently handle the execution of these events.

6.9.1 The Job Scheduler (SETUPQUEUES)

The job scheduler has the task of asking every cell in the system if it requires access to any of the system's buses. The procedure decides which type of request a cell wishes to make and queues the request on the appropriate bus arbiter(s).

The possible requests on the system are:

1) Do nothing, continue present activity
2) Generate a compute bound job requiring no bus access.
3) Generate a request for an additional Cell. Either for more processing power or for more memory. This places calls on all inter and intra module bus arbiters.
4) Request intervention from the global operating system. This places calls on the global bus arbiter.
5) Request intervention from the local operating system. This places a call on all intra module bus arbiters.
6) Generate a request for a slice of global memory. This places a call on the global bus arbiter.
7) Generate an input-output bound job. This places a call on all bus arbiters in the system.

The job scheduler calls a routine, JOBSELECTION, to determine which of the above options are to be used. Only

REQUESTCONST percent of all possible bus requests are ever actioned as a request gan be generated by every cell at every clock tick.

The following is a schematic layout of the jobscheduler, figure 6.1.

System Setup	System Initalisation
JOB CREATION MECHANISM	
CELL MANAGER	
JOB SELECTION PROCEDURES	
BUS QUEUE MANAGEMENT SYSTEM	
BUS ARBITER MECHANISM	

Figure 6.1 Group Processor Simulator Schematic Diagram

6.9.2 The Job Server

The second major section of the simulator is the job server. This has the function of monitoring all the bus arbiters and ensuring the correct bus request/grant/relinquish sequence.

Once the Server has given a cell bus control it inquires of the cell to what purpose it is required for and acts accordingly. Possible actions available to the server are:

1) If the bus is idle then execute the request
2) If the cell has finished with the bus then queue next request
3) If a cell is actively using the bus then do nothing

The Job server calls several routines during its execution these are:

1) A process which starts bus activity (STARTQUEUE)
2) A process which removes unwanted arbiter entries (REMOVEENTRYS)
3) A process which initiates a new bus master (SERVECELL)

On system boot each cell, module and bus is given a percentage chance of failure. Any defective components are marked as defective and take no further part in the simulation.

6.9.3 System Loading

After system initialisation the simulator queues a large quantity of bus requests upon the arbiters. This takes place before the bus system becomes operational. This initial loading represents the system examining its startup configuration.

6.10 Range of Results

The Group Processor simulator has been designed to give a range of results, specified by specific parameters, for a single program run. This is achieved by varying the specified parameter as the program executes and placing the resultant data in one of ten specific directories.

6.10.1 Variation of 'Physical Constants'

In the simulator a 'Physical Constant' is taken specifically to be one of the following:

1) The number of Global Buses
2) The number of Inter-module Buses
3) The number of Intra-module Buses
4) The number of Cells per Module
5) The number of Modules per Machine
6) The number of Input/Output buses

6.10.2 Variation of Cell Numbers

For a given number of Global, Intra and Inter-module buses and modules, the number of cells per module were varied in ten steps from 2 to 1024.

6.10.3 Variation of Module Numbers

For a given number of Global, Intra, Inter and I/O buses and cells per module, the number of modules per configuration was varied from 1 to 16.

The single module environment represented the smallest machine configuration. The 16 module environment constituted a machine with some 67,108,864 bytes of memory. These configurations represented the single user system. A multi-user mode was also simulated with varying number of modules in each users work-space, as well as a random number of modules per user.

6.10.4 Variation of Inter-module bus Numbers

For a given number of Modules, Cells, Global and Intra-bus numbers, the number of Inter-module bus numbers was varied from 1 to the number of Modules present.

6.10.5 Variation of Module numbers and Inter-module buses

For a given number of Cells per module, Global and Intra-module buses, the number of Inter-module buses and Modules were increased from 1 to 64 in equal amounts.

6.10.6 Variation of 'Soft Constants'

A soft constant is defined to be one of the following:

1) Rate of production of bus requests
2) Ratio of I/O, CPU, Memory, Cell requests and memory requests
3) Time required by a cell as bus master
4) Message transmission time (message length)

6.10.7 Bus Request Rate

The rate at which the simulator (CELLS) demand the bus was varied from 1 per clock cycle to 1 per 100 clock cycles.

6.10.8 Ratio of Jobs

The simulator is able to vary the job load of the system. Available jobs include:

1) Cpu bound jobs
2) Input-output bound jobs
3) A mix of (1) and (2)
4) Memory request
5) Cell request

6.10.9 Time Required as Bus Master

This is the time for which a cell will have exclusive control of a particular bus.

6.10.10 Message Length

This will be the amount of time it takes a cell to broadcast its message over the bus system.

6.11 Simulation Goals

The original aims of the simulation were:-

1) To show the Group Processor System works.
2) Given the throughput and resources, determine system performance.
3) Given the throughput and target performance, determine the system configuration to achieve that performance.
4) Given a system configuration, determine the throughput of the system.

6.12 Limiting Factors

The following chapter presents the results of the simulation. But what are the limiting factors of the simulator presented in this chapter?

The simulator used random numbers to gain an understanding of typical system workload and operation. In real life, a user's needs may be more predictable over a given length of time. This would result in a much more highly tuned machine than simulated. However, the main objective of the simulation is to identify typical bus interconnection patterns for particular application areas. Chapter 7 examines these interconnections in detail. In this area the simulation provides useful information.

6.13 Conclusion

The simplicity, yet complex nature of the simulator make it impossible to try all possible permutations of system parameter. Therefore only those factors influencing major architectural features have been modified:-

1) The number and type of buses
2) Message Length
3) Off loading factor
4) Numbers of modules and cells
5) Single vs. Multi-user Systems

While the aims of the research were very important, a number of technical and time related problems existed. The original aims were too demanding for the research time allowed. However, careful analysis proved that most of the information needed for further development of the Group Processor System could be obtained with the eventual simulation environment. A further move to the current simulator resulted in the time required to execute just one run of the simulator dropping to acceptable scales. The fully developed simulator would exhaust the available processing equipment available for the research.

References - Chapter 6

1. BEIZER, B., "Micro-Analysis of Computer System Performance", Van Nostrand Reinhold 1978.

2. SWAN, R., "The Switching Structure and Addressing Architecture of an Extensible Multiprocessor - Cm*", CMU-CS-138

3. SHARPE, W., "The Economics of Computers", Columbia University Press 1969

4. HOWARD, P., "Measuring System Performance Using Benchmarks", EDP Performance Review 1973

5. KIDDER. T., "The Soul of a New Machine", Penguin 1982.

CHAPTER SEVEN

CHAPTER SEVEN

ANALYSIS OF GROUP PROCESSOR SYSTEM SIMULATION

7.0 Introduction

In this chapter the output from the simulation of the Group Processor System is analysed. The conclusions drawn from this output will aid computer architects design more efficient cellular systems of the Group Processor type, as the simulator provides a window on the executing Group Processor System.

The simulation produces an abundance of results, as a result only those results of significant value are presented in this chapter. A fuller listing of results is available in appendix 2, which provides the reader with a much more wide spread comparision of results. A much fuller set of output listings is available for viewing, the sheer volume of output listings makes it impractical to include them in this Thesis.

This chapter is organised into two sections. Section one provides a general analysis of a summary of the simulation results. These results are provided in tabular form and give the reader an overview of the simulation results taken from each architecture under investigation.

Section one is mainly concerned with the number, and type, of buses connected to each cell. Section two looks much closer at key output graphs from the simulation. The relative performance of each architecture is compared throughout the simulation time, and major defects and features of the overall architecture are identified.

7.1 Table of Results

A summary of results in tabular form enables the reader to quickly compare the the various systems and strategies that have been tested. These tables have been broken down under five main headings, some further sub-divided where different variables have been used to tune the basic architecture under examination.

These architectures are grouped as tables 1-5, and describe:-

1. The basic Group Processor System, as described by Quick, with little or no variation in operating system variables. This basic architecture is a fundamental building block for further system variations.

2. This architecture represents the first variation on the basic Group Processor System. While this is a table with only one entry, it is most significant with respect to the architectural design of the future Group Processor System. In this configuration, the basic cell has an extra bus added in the form of a dedicated Input-Output bus.

3. This architecture outlines the variations in bus scheduling, and its corresponding effects it has on system performance on the architecture found in 2 above. The architecture of that found in this system is complex, as it shows how the change in operating system variables can have a profound influence on the efficient running of the finished design.

4. One of the basic Group Processor's features is that of system segmentation. The output in table 7.4 shows how this feature can affect the performance of the Input-Output bus.

5. Lastly, this architecture tries to evaluate the performance of a typical multi-user Group Processor System.

BASIC GROUP PROCESSOR ARCHITECTURE

	BUSES				CONST					SERVICE RATE %				CELL STATUS		
GLOBAL	INTRA	INTER	!/O	O/S	MESSG	C.REG	IO	TIME	I/O	SEND-MSG	OS CALL	FREE	ACTI	WAIT	COMM
1	3	3	0	1	5	10	20	200	86.3	80.4	99.8	26	559	432	6
1	3	3	0	1	5	10	20	5	69.2	68.0	100.0	29	18	971	6
1	3	3	0	1	10	10	100	5	---	---	---	2	1	1014	7

TABLE 7.1

BASIC GROUP PROCESSOR & GLOBAL I/O BUS

BUSES				CONST				SERVICE RATE %				CELL STATUS			
GLOBAL	INTRA	INTER	I/O	O/S	MESSG	C.REG	IO	TIME	I/O	SEND-MSG	OS CALL	FREE	ACTI	WAIT	COMM
1	3	3	1	1	5	10	20	200	27.0	100.0	100.0	29	644	345	6

TABLE 7.2

BASIC GROUP PROCESSOR & SHARING I/O BUS

BUSES						SERVICE RATE				CELLS			
Global	Intra	Inter	I/O	IO	IO-Q	I/O	Send-Msg	OS Call	New Cell	Free	Acti	Wait	Comm
1	3	3	1	20	70	81.8	90.4	100.0	90.1	31	641	345	7
1	3	3	1	20	30	84.6	87.9	100.0	91.2	31	649	336	7
1	3	3	1	20	50	82.7	89.3	99.8	89.7	32	666	319	7
1	3	3	1	5	50	94.7	99.2	100.0	98.8	41	861	115	7
1	3	2	1	5	50	93.5	95.2	99.7	96.5	36	763	219	6
1	3	2	1	20	50	71.2	85.1	100.0	93.9	28	577	413	6
1	2	3	1	20	50	81.4	83.0	99.8	82.3	27	563	427	6
1	2	3	1	5	50	93.4	96.8	100.0	95.7	37	768	213	6
1	2	2	1	5	60	94.1	90.3	100.0	89.8	32	688	299	5
1	2	2	1	20	60	75.9	77.5	100.0	80.3	25	512	481	5
1	3	1	1	5	50	95.9	86.5	100.0	88.8	31	660	328	5
1	3	1	1	20	50	53.5	81.6	100.0	99.0	24	529	466	5
1	1	3	1	20	50	76.9	73.2	99.8	74.2	22	479	518	5
1	1	3	1	5	50	96.7	75.8	100.0	99.8	32	661	326	5
1	1	1	1	5	70	99.6	59.4	100.0	60.2	22	454	545	3

TABLE 7.3

SEGMENTED GROUP PROCESSOR INPUT/OUTPUT BUS

BUSES				CONST						SERVICE RATE				CELL STATUS			
Global	Intra	Inter	I/O	O/S	Mess	C.reg	IO	Time	IO-Q	I/O	Send-msg	OS Call	New cell	Free	Acti	Wait	Comm
1	3	3	16	1	5	10	20	200	50	99.8	99.8	99.9	99.9	44	917	52	11
1	3	2	16	1	5	10	20	200	50	99.7	96.1	99.8	95.9	40	807	166	10
1	2	3	16	1	5	10	20	200	50	99.5	94.8	99.8	96.1	39	807	168	10
1	2	2	16	1	5	10	20	200	50	100.0	85.9	100.0	88.5	34	679	286	8
1	1	1	16	1	5	10	20	200	50	100.0	59.7	100.0	64.0	23	464	532	5
1	3	2	16	1	5	10	100	200	50	75.2	99.6	100.0	99.8	36	771	196	21
1	2	2	16	1	5	10	50	200	50	99.6	88.9	100.0	88.7	34	704	272	14

TABLE 7.4

SEGMENTED GROUP PROCESSOR INPUT/OUTPUT BUS

BUSES					CONST					SERVICE RATE				CELLS			
Global	Intra	Inter	I/O	OS	Messg	C.reg	IO	Time	IO-Q	I/O	Send-Msg	OS Call	New Cell	Free	Acti	Wait	Comm
1	3	3	16	1	5	10	20	200	50	99.1	98.9	100.0	99.3	44	906	63	11

TABLE 7.5

7.2 Basic Group Processor System

The main problem with the projected Group Processor System is the potential bottleneck on the system buses. Any problems at this level will manifest itself as waiting time for any cell wishing to gain access to the bus, and will result in increased processing time for any program. With this in mind, a number of key tables are coompared which will serve to identify the architectural features most likely to slow down processing at the system level.

The three entries in table 7.1 identify a number of key variables which are pursued throughout the analysis of the simulation results. The first variable relates to the length of message allowed by each cell at any one time. The variables used here are 5 and 10. These are in effect lengths of time slots for any one cell to transmit on a bus, after which it must relinquish the bus. If a cell wishes to send a very long message, then the cell must make repeated calls on the system buses.

The second variable relates to the amount of input-output processing that the system is performing, the higher the Input-Output constant the more Input-Output bound the system is. In the context of the tables shown here, the figure of 100 proved in reality to be the equivalent of infinity. Similarly, the Time heading in table 7.1, relates to the amount of processing done within a cell. Ideally, the higher the processing time constant the more memory a cell has. In reality, this may not be true as it may be an

indication of whether the cell makes repeated calls to the internal memory within the cell, or whether the cell persistantly makes calls to the system buses. The underlying theory behind this is, the less memory a cell has, the more calls it will make to other cells for services, e.g. to store data.

In analysing the results from the simulation of the basic Group Processor System; the third entry in table 7.1 can be disregard. This entry was so bad that by the end of the simulation period, nearly every cell within the system was waiting for a system bus. Clearly, the high Input-Output constant of 100 made this an architecture targetted for application in IKBS. From the poor Input-Output performance result, the basic Group Processor System is not suitable for application in IKBS.

Even with the Input-Output constant greatly reduced, the system has a poor service rate. The second entry shows the performance for a system configuration of small memory size and low Input-Output processing. This configuration still produces very high bus waiting times, with an added degrade in the average number of buses being used, i.e. only six out of seven being utilised at any one time.

An additional consideration is the use of the global operating system bus for Input-Output calls. From Table 7.1 there is a very serious degrade in Input-Output performance if all Input-Output calls are mapped to the global bus. This configuration also degrades the operating system calls from

a service rate of 100% down to 33%. Clearly this configuration is not desirable.

7.3 Extended Group Processor System

The major problem associated with the basic Group Processor System is the relatively poor bus service rate. One important requirement in any cellular system is that of immediate communication with other cells within the execution environment. Another requirement is to provide a very fast Input-Output system both to the user and to the program execution environment.

The need for fast turn-around in Input-Output traffic cannot be met with the current cell design. With this in mind the provision of an dedicated Input-Output bus offers a "potential" speedup in Input-Output communication. It is important to note here that an additional bus has been provided to each cell within the system. No other system changes have been made.

From table 7.2; there is a marked increase in service rates on all the basic system configuration's buses. However, there is a marked degrade in the processing of Input-Output calls. On face value this would appear to be contradictory to providing a dedicated bus for Input-Output operation. The Input-Output degrade is quite simple; whereas the basic Group Processor System queued all Input-Output bus requests on all system buses, excluding the global bus, the new configuration reduced the number of channels available

for Input-Output from the 6 in the basic system down to only one in the new configuration, i.e. the dedicated Input-Output bus.

If we are to compare the original Group Processor System proposed by Quick, and the expanded system proposed here, then it has to be admitted that Quick's original design averages out quite well for bus service requests.

7.4 Operating System Constraints

Up to this stage the variations that can be imposed by the operating system have been ignored. We have concerned ourselves with only adding physical resources, i.e. the addition of a single Input-Output bus. In this section the extended view of the system, which as a result includes in the picture the role of the operating system is taken into account.

A number of operating system variables will be examined, and what influences these play on the simulation output will be examined. From table 7.3 a number of hardware architectures, e.g. 1,3,3,1, which represent the number of buses that are available to each cell. In addition to these hardware resources a new operating system variable is introduced, this being an "Off-Loading Factor".

The Off-loading Factor is a recognition of the value that the basic Group Processor System had in providing a relatively high bus service rate, it is the cut off point at which bus requests are redirected to 'other' buses, and also

the importance of providing a dedicated Input-Output bus system. Together the basic Group Processor System provides a good beginning, while the introduction of the dedicated Input-Output bus provided a high degree of tuning on the main system buses. What would be ideal would be the maintenance of the 100% service rates on the main system buses, coupled with a great improvement in the capability of the dedicated Input-Output bus.

In table 7.3 are the results from the above scenario. The results from the additional Input-Output bus coupled with Quick's original Input-Output sharing scheme provides the system with a greatly improved system performance. This table provides most of the architectural information needed for designing and configuring a Group Processor based on the original concepts outlineb by Quick. From this table five important entriescan be identified, which are:-

Bus Configuration	I/O Constant	Off-loading
1, 3, 3, 1	20	50
1, 3, 3, 1,	5	50
1, 2, 3, 1,	5	50
1, 2, 2, 1	5	60
1, 2, 2, 1	20	60

The above entries represent the main cell inter-connection patterns available to the system architect. There is one other which is the 1,1,1,1, but this system we

shall ignore for now because it represents a "bare bones" Group Processor System. The first two inter-connections are based on the 1,3,3,1 pattern, which have proved so far to be the best inter-connection pattern. The service times for these patterns are very high, many being well in excess of 80%. The most important entry in this area of the table is number 4, where the service rates are very high with at least 95% service rates on all buses. However, this entry represents a small Input-Output loading comparable to a processor bound job. For a corresponding high Input-Output traffic bound job there is a 10% drop off in bus service rates.

The mapping of the Input-Output calls to the global bus in Table 7.1 produced disappointing results. In Table 7.3 the same concept is extended by adding the Input-Output bus to the basic system configuration of 1,3,3. In this configuration the Input-Output calls are mapped to the Input-Output bus until they reach 50 entries, after which they are mapped onto the global operating system bus. As with the 1,3,3 inter-connection scheme, this Input-Output queuing pattern proved to be undesirable.

The other important consideration is in table 7.3; consider the average number of cells Waiting and Active during the simulation run. The first entry above, averaged 336 cells waiting for the system buses. This represented a high Input-Output bound job compared to the lower Input-Output traffic which yielded only 115 waiting cells. It is interesting to note these values as percentages, the first being 48% and the second being 13% illustrate a different way

in which Input-Output traffic affects bus waiting time in this system configuration.

The last two entries above denote a Group Processor System with less buses. The overall performance of these inter-connection and bus scheduling strategies provides a useful system configuration if technological constraints impose a limitation on the number of available buses. If a prototype Group Processor System were to be constructed, with the corresponding limitations in costs and complexity, then the 1,2,2,1 system configuration would be a good starting point for system development. However, this system would not be seen as a high availability, high integrity system design as there would not be any form of triple modular redundancy at the bus level. With this in mind the 1,2,3,1 configuration is suitable, with a corresponding increase in system performance. The three Inter Module Buses would allow Triple Modular Redundant communication anywhere within the system, whereas three Intra Module Buses in the 1,3,2,1 configuration would only allow triple modular redundant communication within each module.

7.5 Off-loading Factor

The Off-loading factor initially seemed to be a critical section for improving system performance. This turned out to be not the case. Within the Input-Output queue length range of 30 to 70 there was very little variation in system performance. The lower the Off-loading Factor the better the Input-Output performance, but at the cost of a decrease in

other bus performance. For systems that need higher Input-Output throughput an Off-loading Factor of 30 is recommended.

7.6 Segmented Input-Output

The most interesting results relate to the segmentation of the Input-Output bus. The segmentation of the Input-Output bus enables each module to have its own dedicated Input-Output system. This Input-Output system can be coupled together to provide either general purpose or dedicated Group Processor Systems.

From table 7.4 a number of interesting observations can be made, the most important of these is the Input-Output constant. This constant is high at 20, and extremely high at 50. The readers attention is also drawn to the 100 constant, which is seen as infinity. The overall performance of this inter-connection scheme is a positive indicator of a typical engineered system architecture.

Taking the 1,3,3 scheme first; the addition of 16 Input-Output buses produces Bus Service Rates approaching 100%. Clearly, this system design could be used for real time IKBS as there is little or no cell waiting time. If we take the 1,2,3 inter-connection scheme, this scheme offers very good performance figures in excess of 95%. In fact, the cell Input-Output has to go as low as the 1,1,1 inter-connection scheme before we see a marked degrade in system performance.

The ratio of cells active to those waiting for a bus at any one time during the simulation is also very interesting in this system configuration. The 1,3,3 scheme has a ratio of only 0.06, compared to the previous 1,3,3 in table 7.3 of 0.54.

7.7 Dedicated Systems

The inter-connection schemes that have been presented so far have been based on regular module inter-connection schemes. In real life applications there may be one or more modules dedicated to supporting a number of execution environments. In table 7.5 the high performance inter-connection scheme, i.e. the 1,3,3. inter-connection are analysed.

The important consideration here is that the system supports segmented Input-Output buses. As a result of this, coupled with the fact that Off-loading is also used, the system performance is very good indeed. Service Rates of 99% clearly indicate that irregular system inter-connection is acceptable.

This system can also be considered in the same light as a general purpose system. From time to time it is perfectly feasible that a bus inter-connection scheme in a multi user Group Processor System would take on the appearance of figure 8.3. Therefore, the highest performance figures obtainable in a multi user system would be 1,3,3 in table 7.4 (service rates

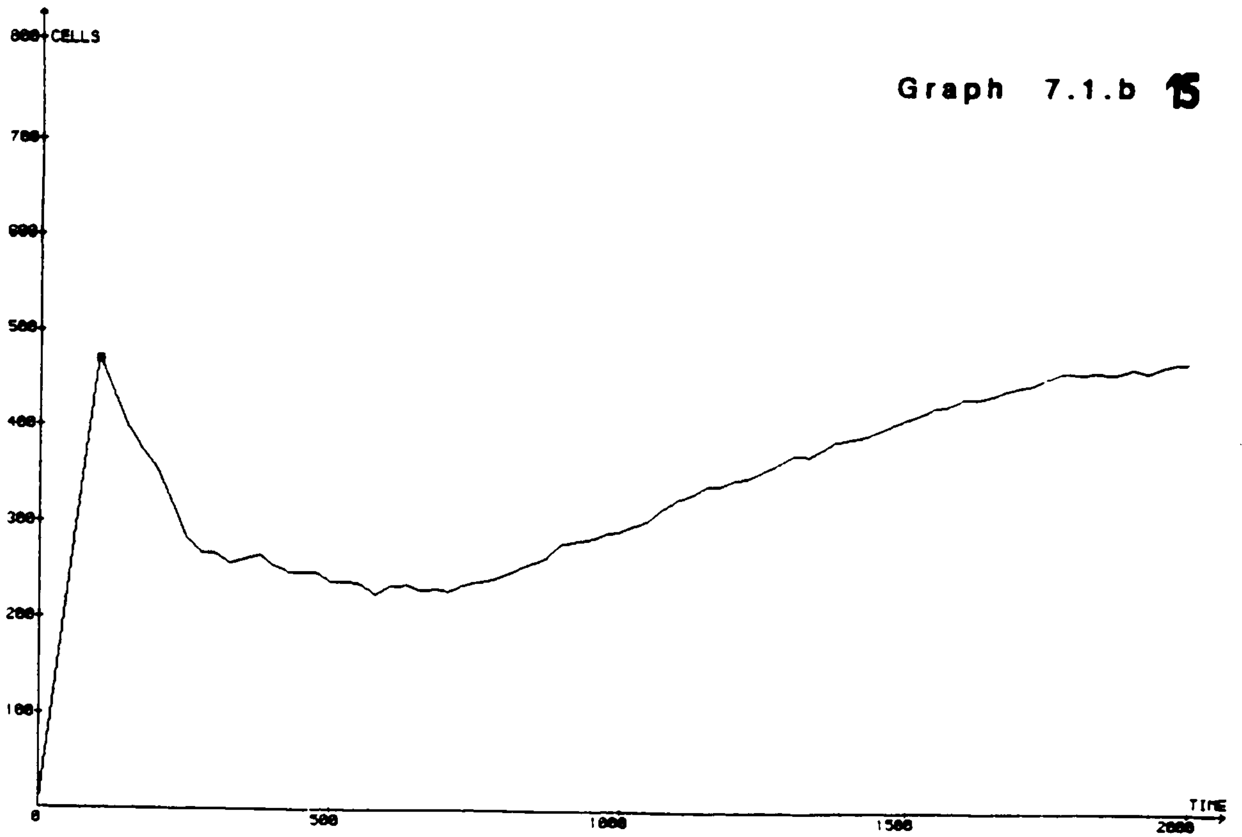
of 100%) and the worst case of 1,3,3 in table 7.2 (with average service rates of 90%).

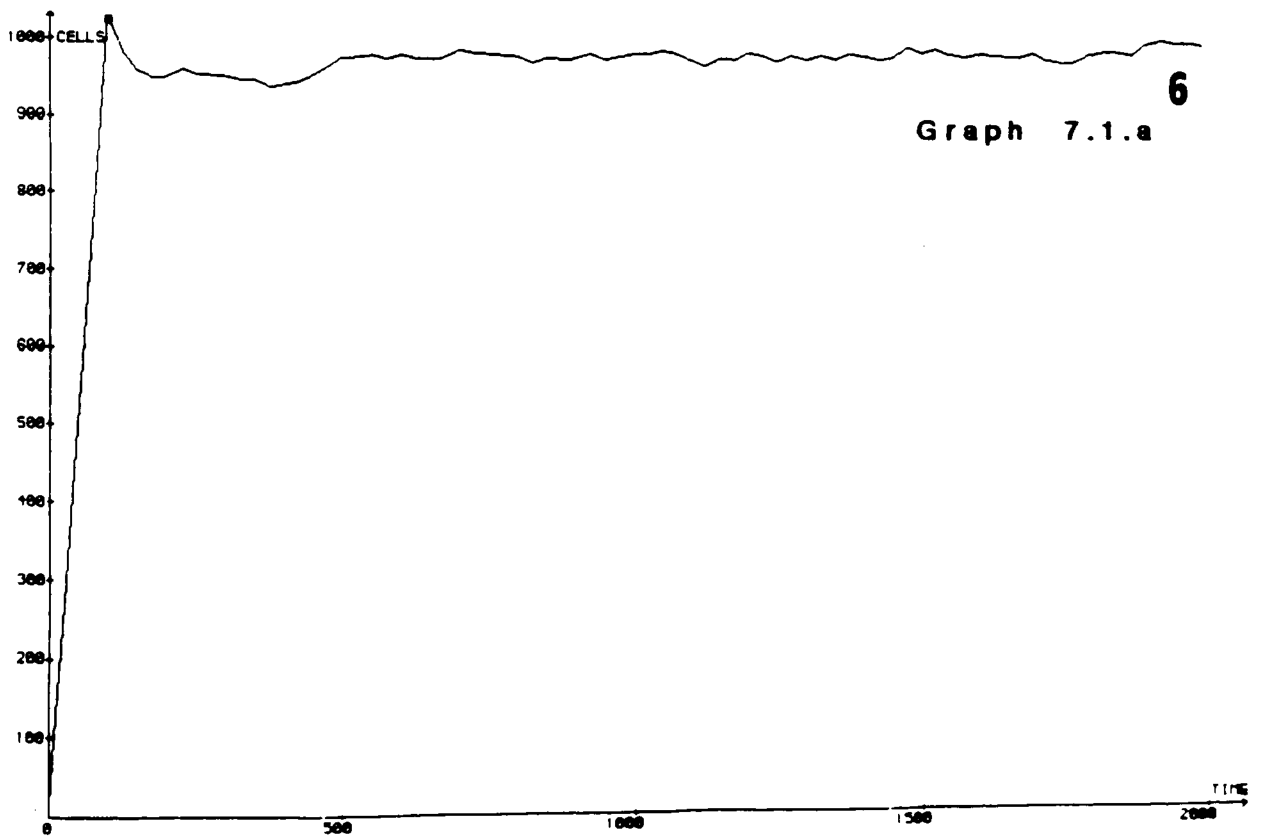
7.8. Closer Analysis

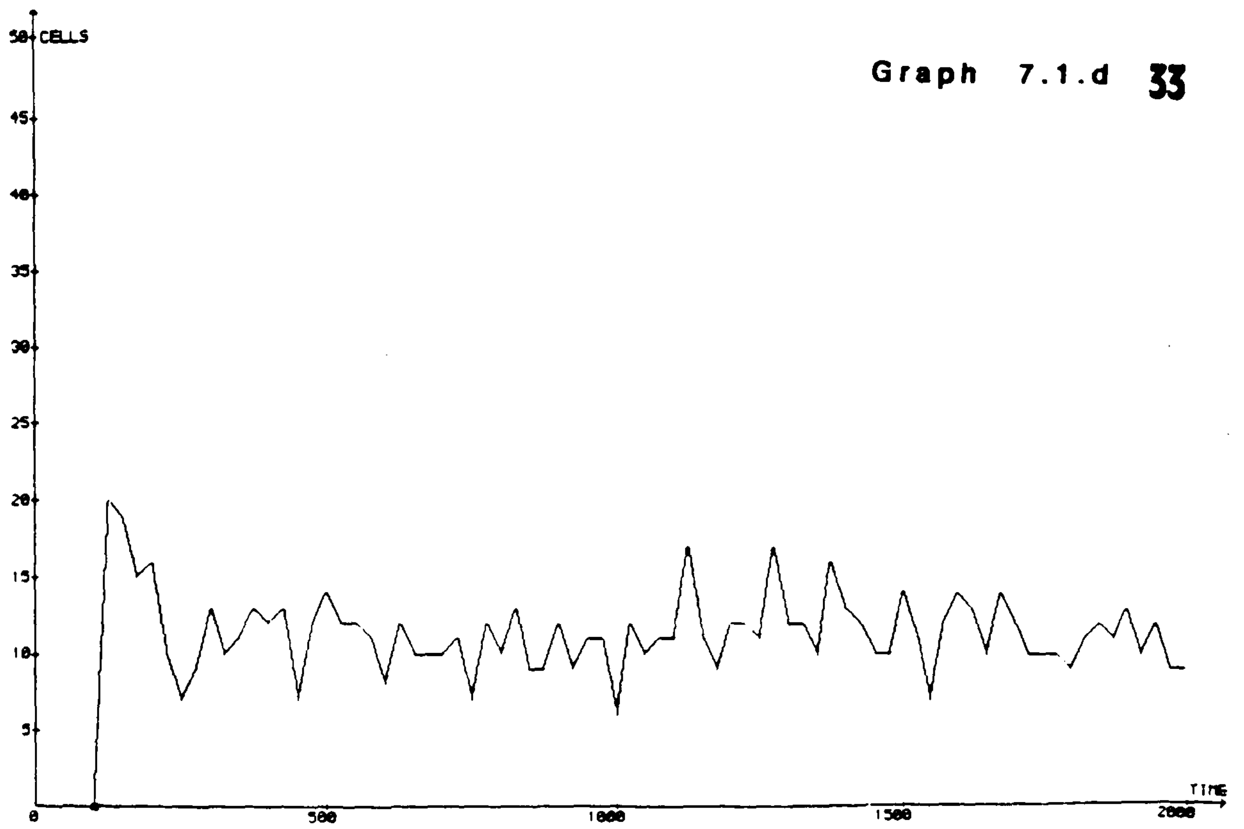
In this section we take a closer look at the graphs generated during the simulation. A number of observations concerning interesting points can be identified within each of the graphs presented. The architecture under consideration consists of 16 modules, each module containing 64 cells.

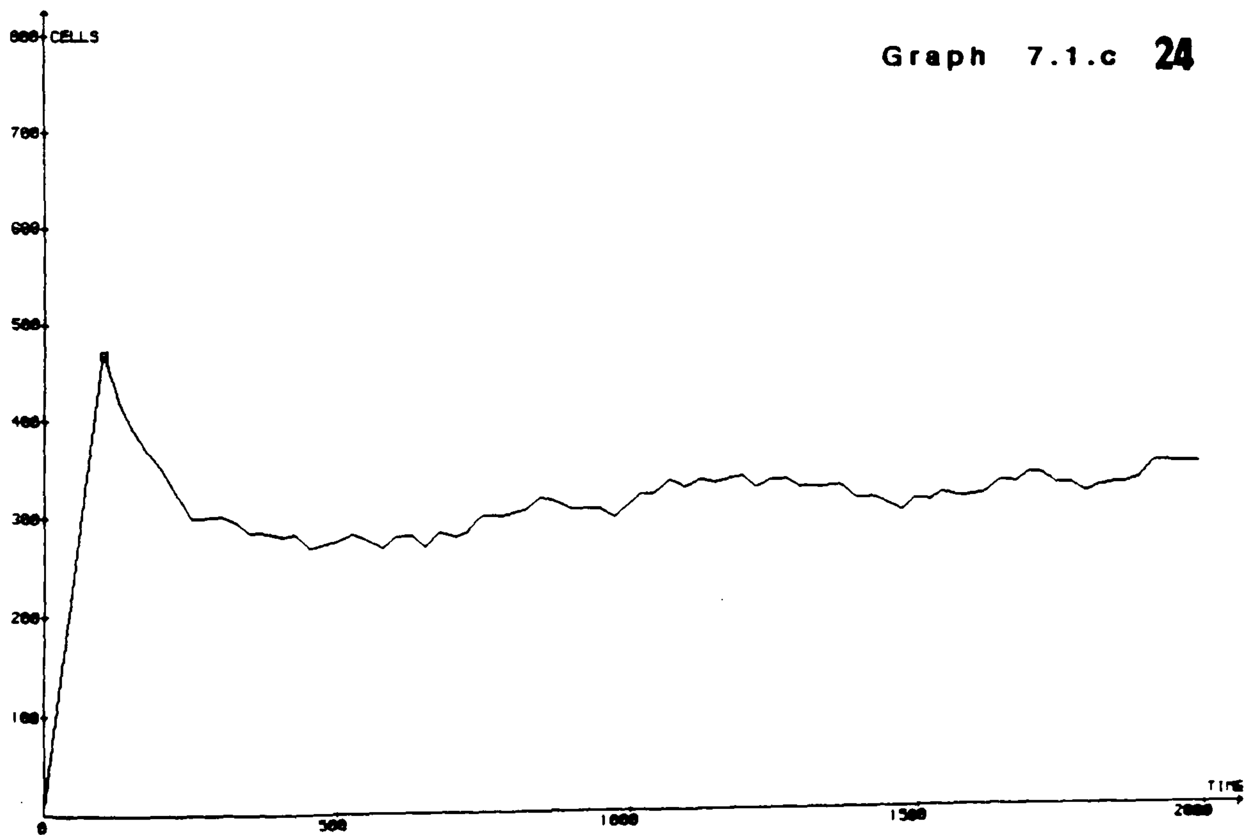
7.8.1. Effects of Bus Contention

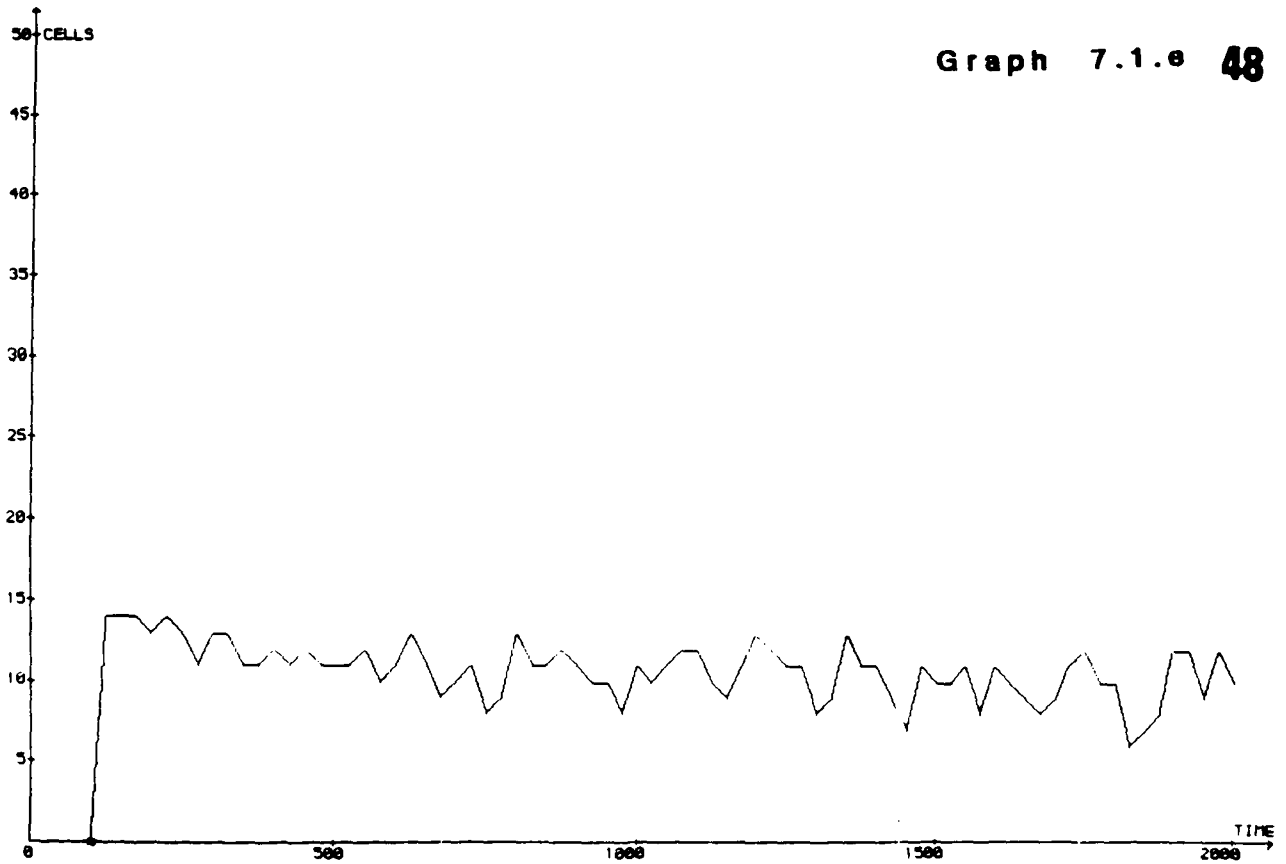
Graphs 7.1.a. to 7.1.e. compare the average number of cells waiting for a bus. Graph 7.1.a. represents the basic Group Processor System compared to graph 7.1.b. which represents the extended Group Processor with only one dedicated Input-Output bus. There is always a very high demand for a bus in the basic system and clearly points to a major system design flaw. The extended system design produces a graph which falls off in bus demand early on in the simulation, indicating the beneficial nature of the dedicated Input-Output bus. The bus demands increase as time progresses through the simulation. This is because of the bottleneck of placing all Input-Output operations on the single Input-Output bus.


CELLS
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
TIME
Graph 7.1.b
15


CELLS
1000
900
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
TIME
6
Graph 7.1.a


Graph 7.1.d 33
CELLS
50
45
40
35
30
25
20
15
10
5
0
500
1000
1500
2000
TIME


Graph 7.1.c 24
CELLS
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
TIME


50 CELLS
45
40
35
30
25
20
15
10
5
0
500
1000
1500
2000
TIME
Graph 7.1.e 48

Graph 7.1.c. shows how the Operating System can play an important role in the efficient running of the Group Processor System. This graph reduces the cells waiting for a bus to one third that of the basic Group Processor System. Compared to the extended Group Processor System, there is a reduction in the number of cells waiting for a bus after 600 time slots. This graph is predictable over a wide spread of the simulation, which is a useful feature when calculating system response times.

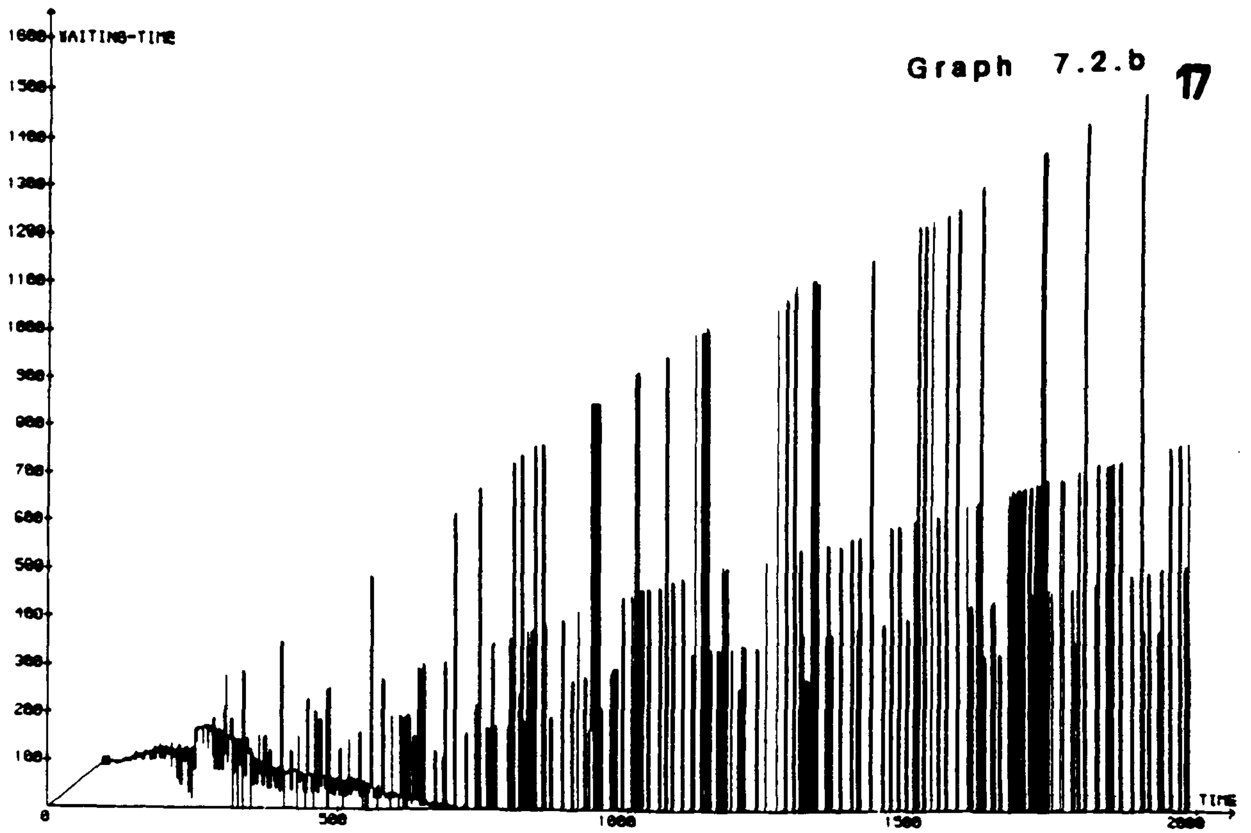
Graphs 7.1.d. and 7.1.e. represent a segmented Input-Output bus structure. From the graphs there is a very low number of cells waiting for a bus. On average only ten cells are waiting for a bus, compared to the basic Group Processor System of some 950 cells waiting at any one time. The spread of the graph in 7.1.d. is between 6 and 18 cell waiting for a bus. However, graph 7.1.e. has a spread from 7 to 13. Both of these graphs illustrate architectures which have highly responsive bus systems.

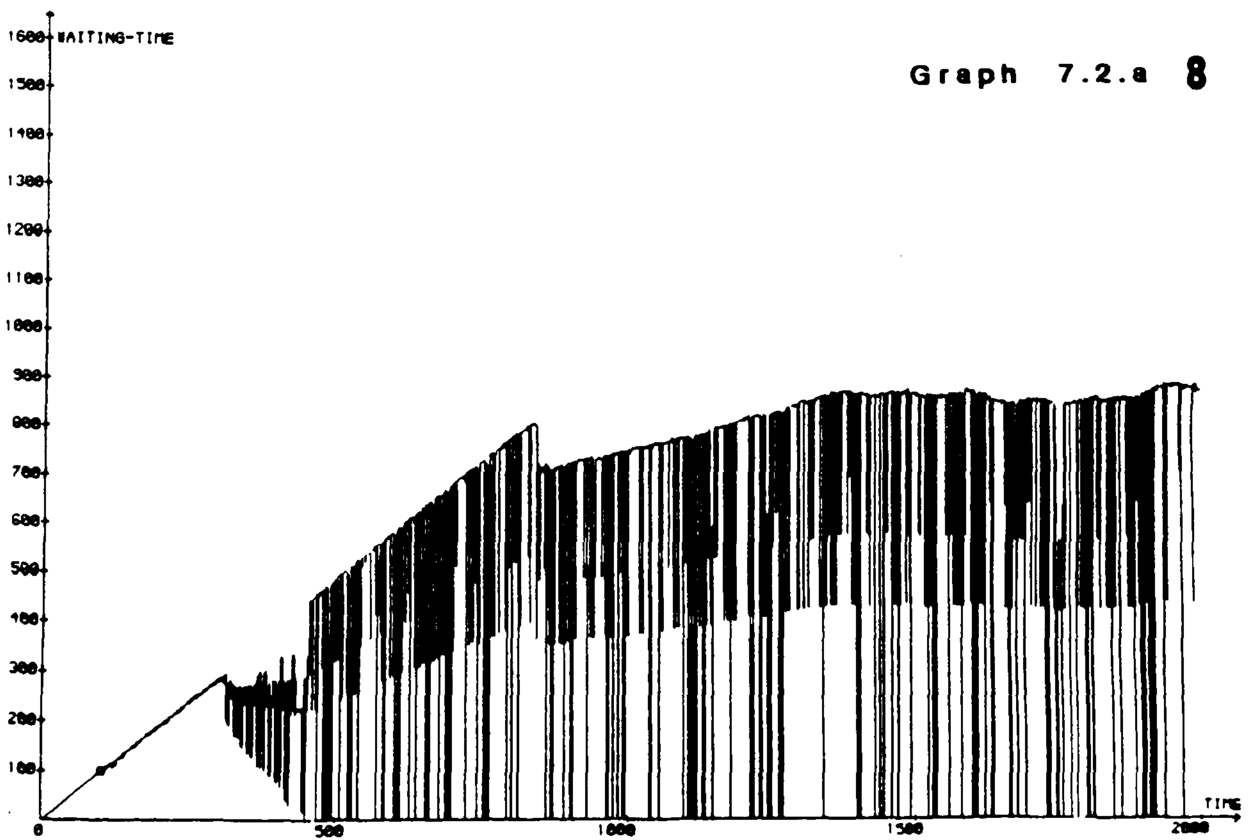
7.8.2 Inter-cell Communication

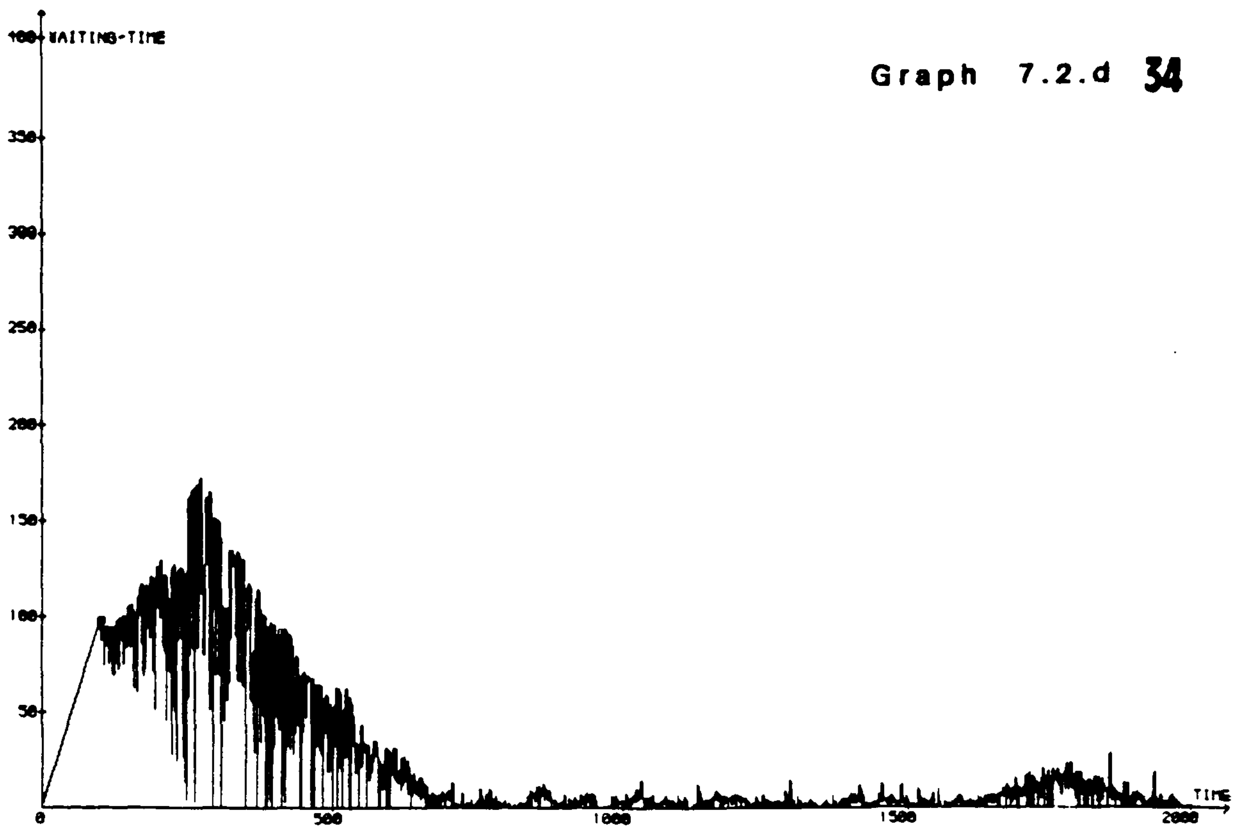
The performance of any Group Processor System ultimately depends on the communication delays incurred during message passing. While this is more often than not communication delays between executing functions within cells, there can also be delays incurred in communication between cells and the Input-Output channels.

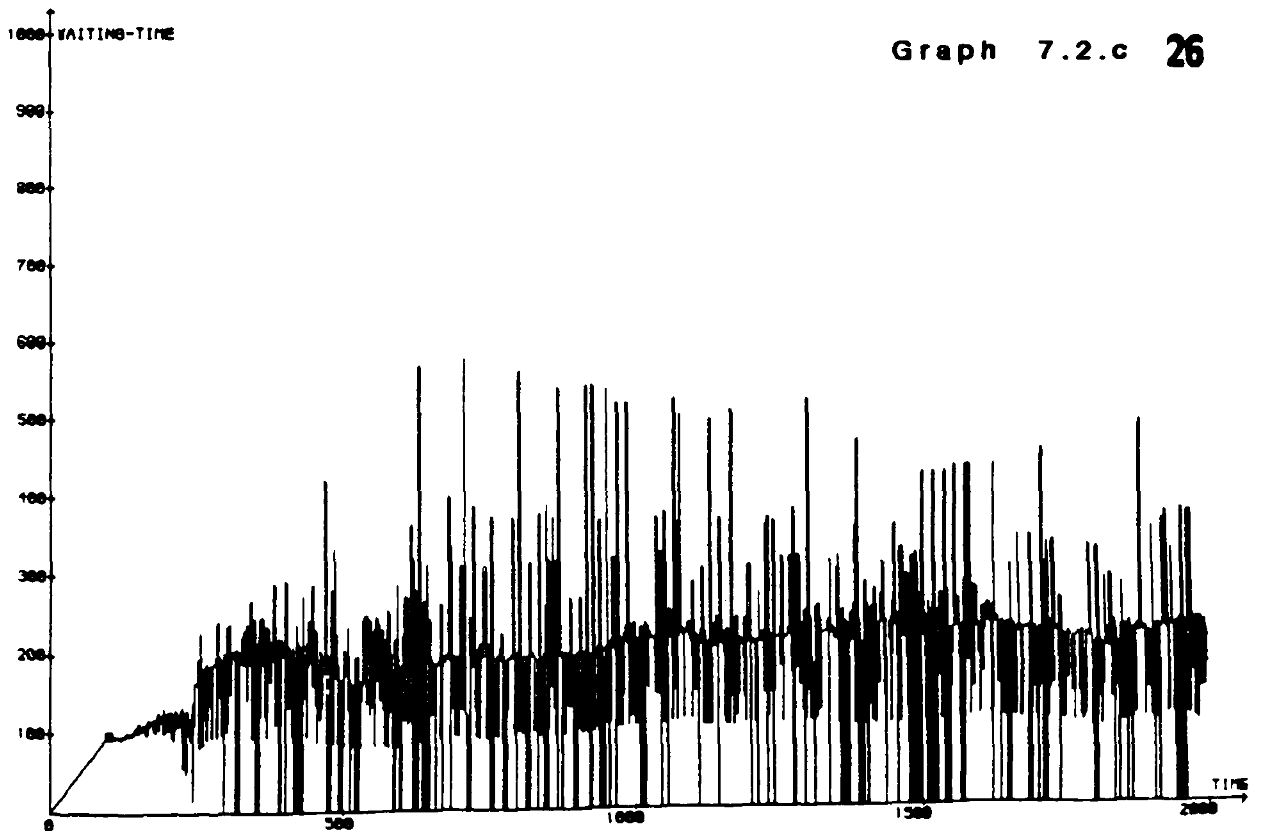
The graphs of 7.2. show the merits of the architectures for minimising communication delays. The basic architecture of 7.2.a. shows that this system architecture is very bad in minimising the transmission of data/functions between cells. The graph climbs steadily upwards, until about one third of the way into the simulation. Unfortunately the graph does not decline, but continues horizontally for the remaining life of the simulation, and averages out at approximately 400 time units for a cell to gain bus access.

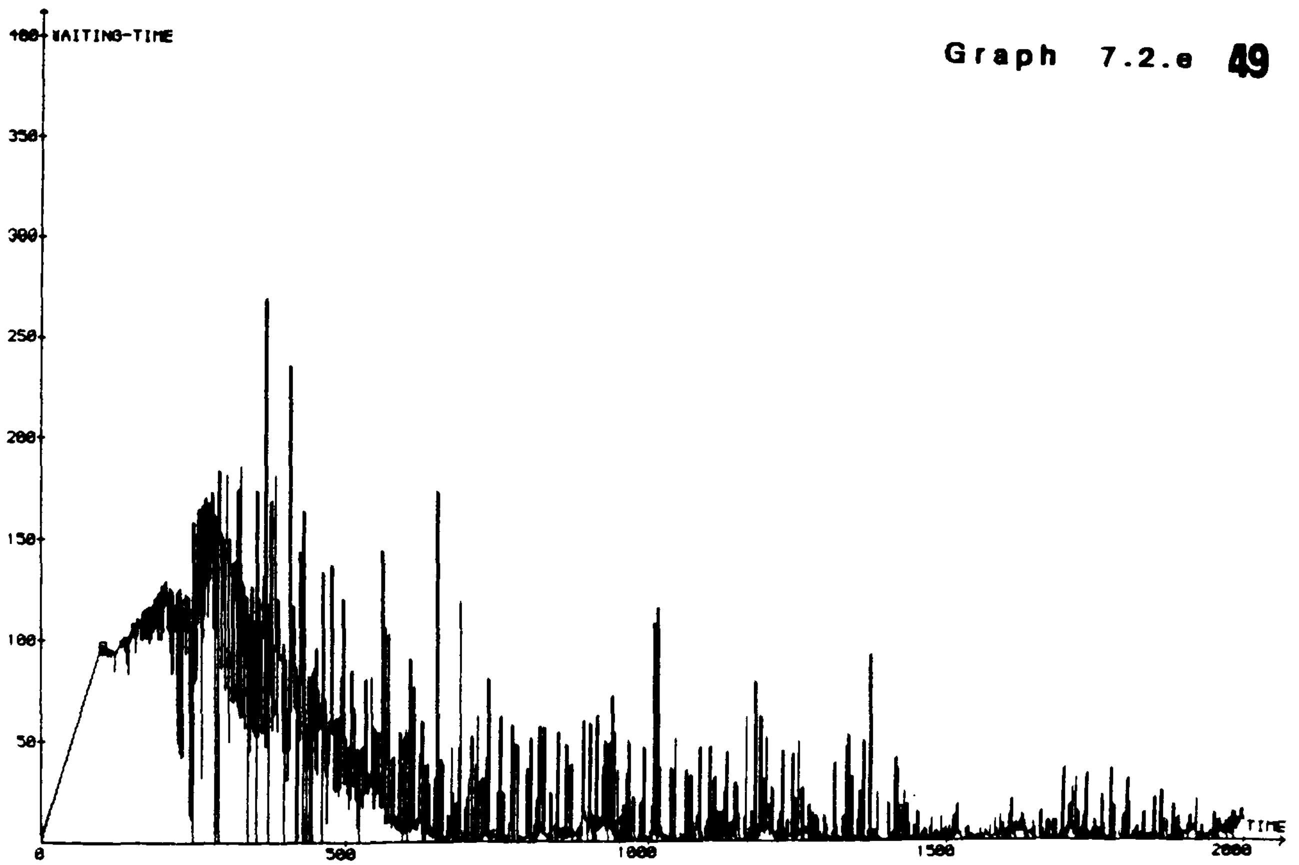
The provision of an Input-Output bus within the architecture does have a dramatic effect in the early stages of the simulation. However, whereas graph 7.2.a. had a spread of about 900 time units at the end of the simulation, graph 7.2.b. has a spread of some 1500 at the end. The density of the "ink" on these two graphs gives an indication of the relative merits of these two architectures. Graph 7.2.a. has much more density of waiting time towards the top of the graph. Graph 7.2.b. has the ink density towards the bottom of the graph, and as result show the usefulness of the Input-Output bus on system performance.


WAITING-TIME
Graph 7.2.b
17
1600
1500
1400
1300
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
TIME


WAITING-TIME
Graph 7.2.a
8
1600
1500
1400
1300
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
TIME


WAITING-TIME
Graph 7.2.d 34
350
300
250
200
150
100
50
0
500
1000
1500
2000
TIME


1000
WAITING-TIME
Graph 7.2.c 26
900
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
TIME


400
350
300
250
200
150
100
50
WAITING-TIME
0
500
1000
1500
2000
TIME
Graph 7.2.e 49

The most remarkable graph is 7.2.c. in terms of the general system configuration, together with the additional one Input-Output bus. The overall performance of this system architecture in minimising communication delay is it's relatively predictable performance. The average time a cell has to wait for system bus is some 200 time units. Throughout the simulation, the architecture presents a very horizontal averaging, with some very occasional wide spread fluctuation. The average spread is 400. Clearly, this architecture is very predictable and suitable for a wide range of applications which require fast inter-cell communication.

While the general Group Processor System with a dedicated Input-Output system gives reasonable performance for fast inter-cell communication, it is the Group Processor System with a segmented Input-Output system which provides the most dramatic inter-cell communication capability. Graph 7.2.d. shows the architecture of such as system with an almost idealised graphical output. The linear form of this graph, together with it's minimal spread of just a few time units, provides a clear conclusion that where possible, Group Processor Systems with segmented bus systems should be used.

Graph 7.2.e. is a compromise between the architecture of the Group Processor System with a dedicated Input-Output bus, and the Group Processor System with a segmented Input-Output bus system. Between these two inter-connection schemes graph 7.2.e. gives an insight into a randomly generated time probability for inter-cell communication. While this

architecture is not generating the idealised graph of 7.2.d., it certainly does approach 7.2.d.'s shape or form.

7.9. Software Considerations

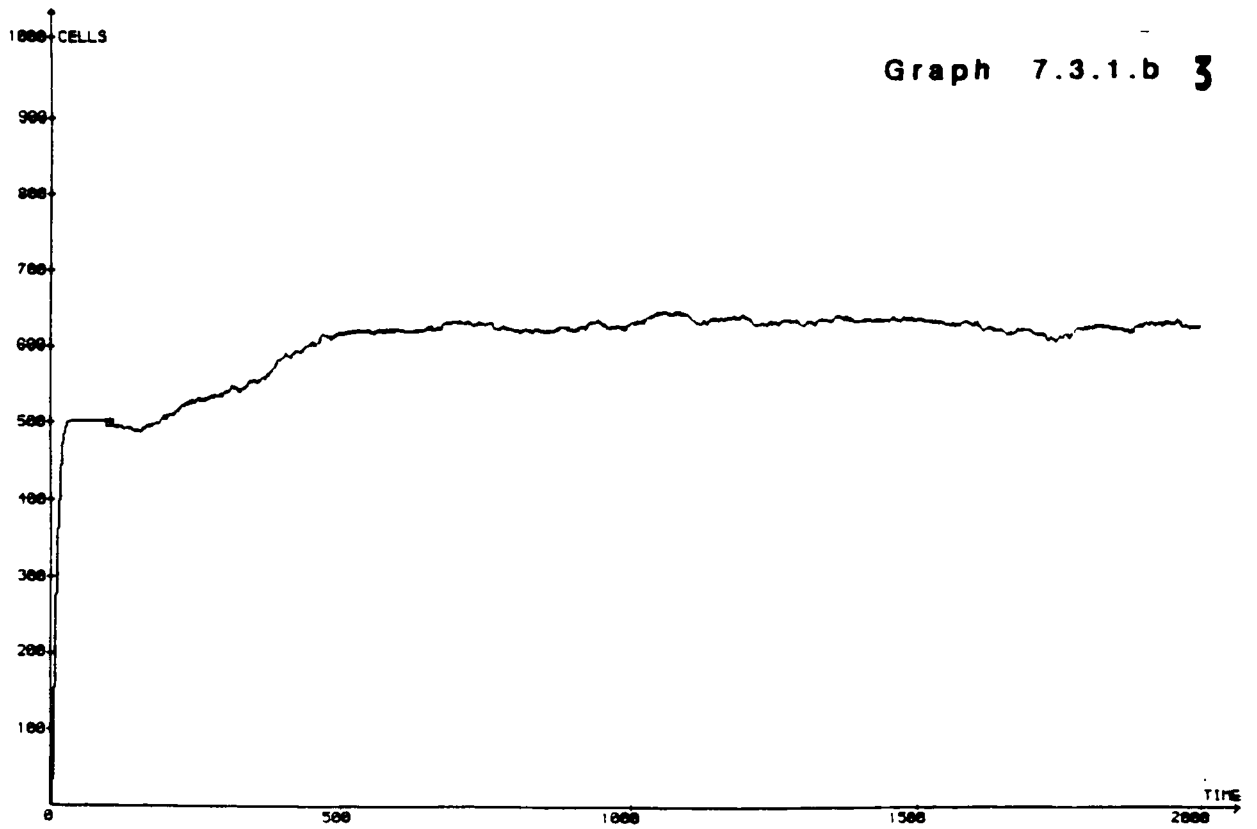
In this section of the chapter, the overall policies of the Operating System are analysed. This analysis looks at each architecture during the life of the simulation and tries to draw important conclusions with respect to bus scheduling and inter-cell communication. The problem of maintaining a high Input-Output bandwidth is also addressed.

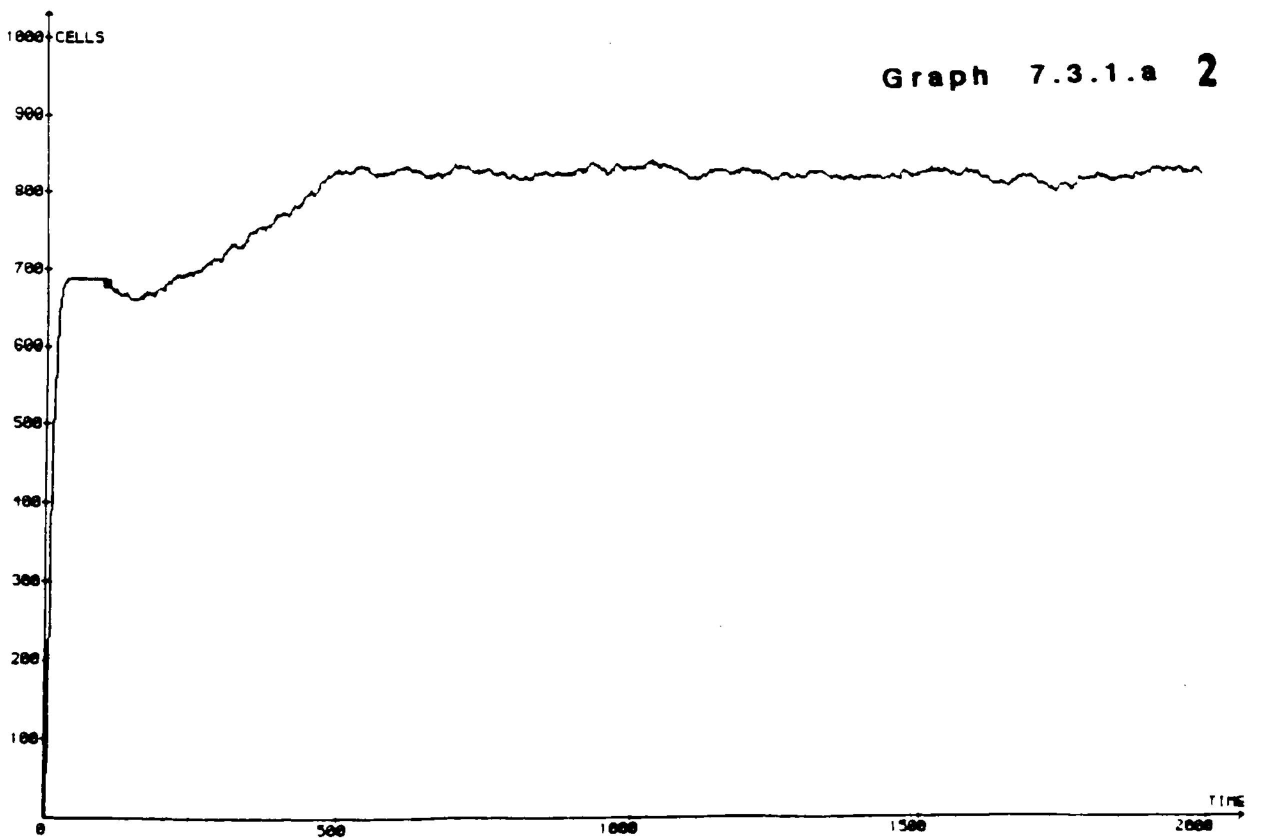
The basic Group Processor System makes very high demands on system buses. The Inter Module Bus has a low bus queue compared with the Intra Module Bus because, a cell that wishes to communicate with another cell within a module can queue on both bus types, as both Inter and Intra Module Buses connect all cells within a module. A cell that wishes to communicate with cells on other modules can only queue on the Inter Module Bus. Graphs 7.3.1.a. and 7.3.1.b. represent the performance of the basic Group Processor System.

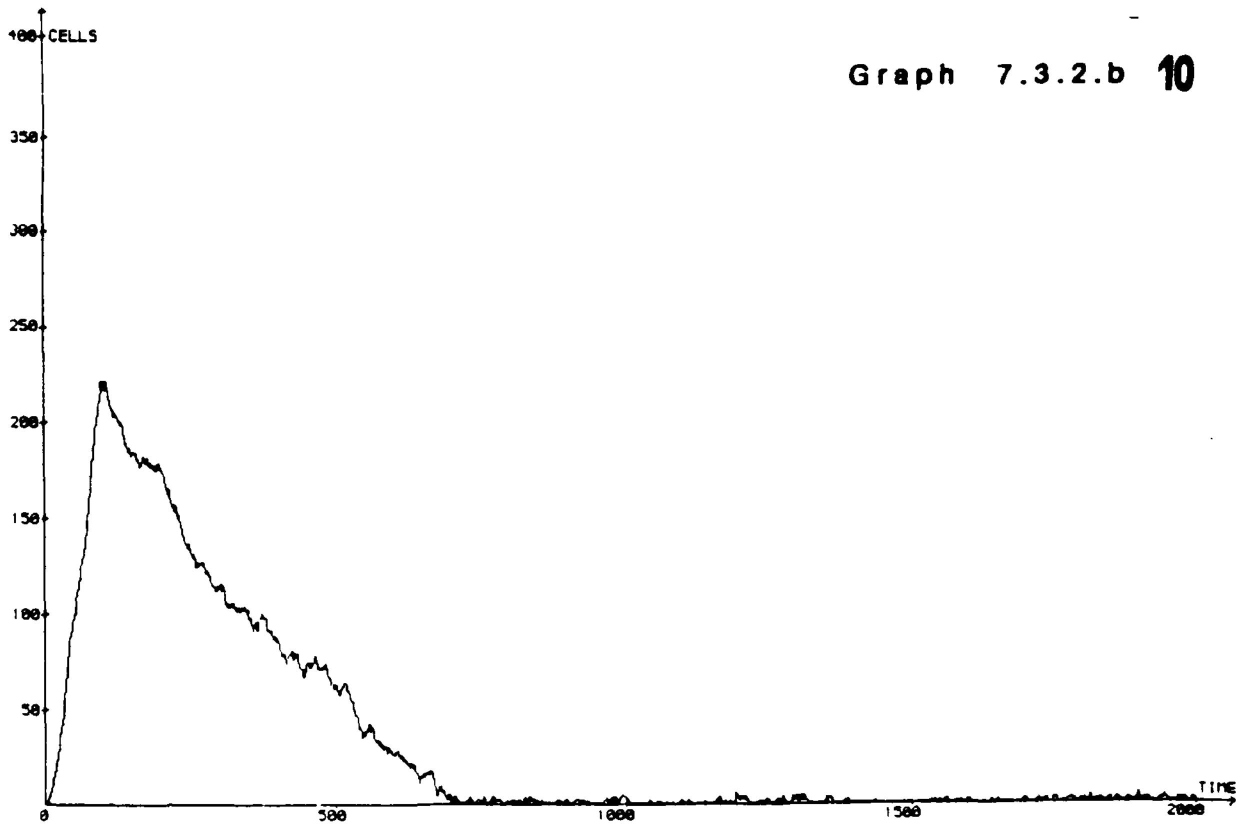
The basic Group Processor System has little opportunity for varying system parameters. The extended Group Processor System has more opportunity for fine tuning system performance outside that of the simple provision of more system buses. Graphs 7.3.2. and 7.3.3. show the effects of changing the scheduling parameters. The low Inter and Intra Module bus queues are very desirable features within any Group Processor System. On the other hand, the upward spiral of the

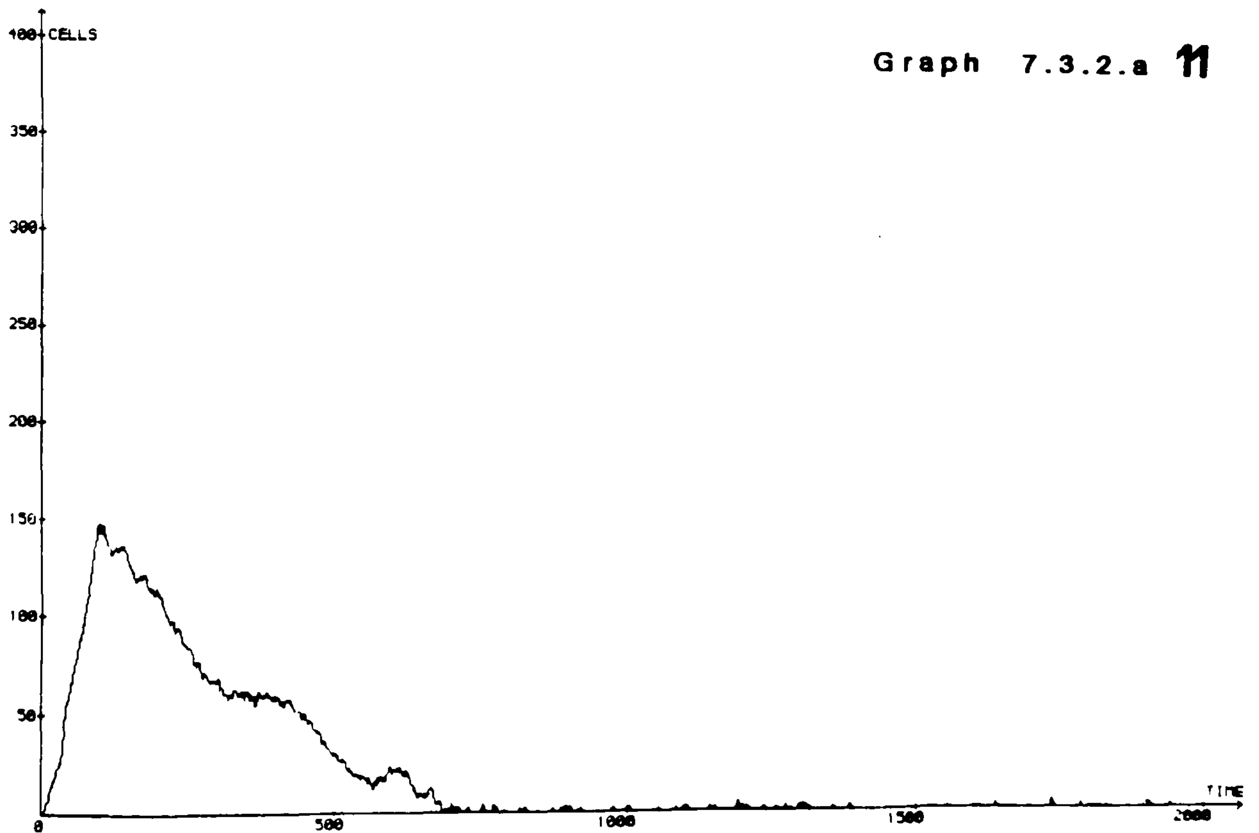
Input-Output queue is not a desirable feature. Graphs 7.3.3. provide a different picture. The low bus queues of the previous graphs give way to seemingly undesirable Inter and Intra Module graphs in 7.3.3.

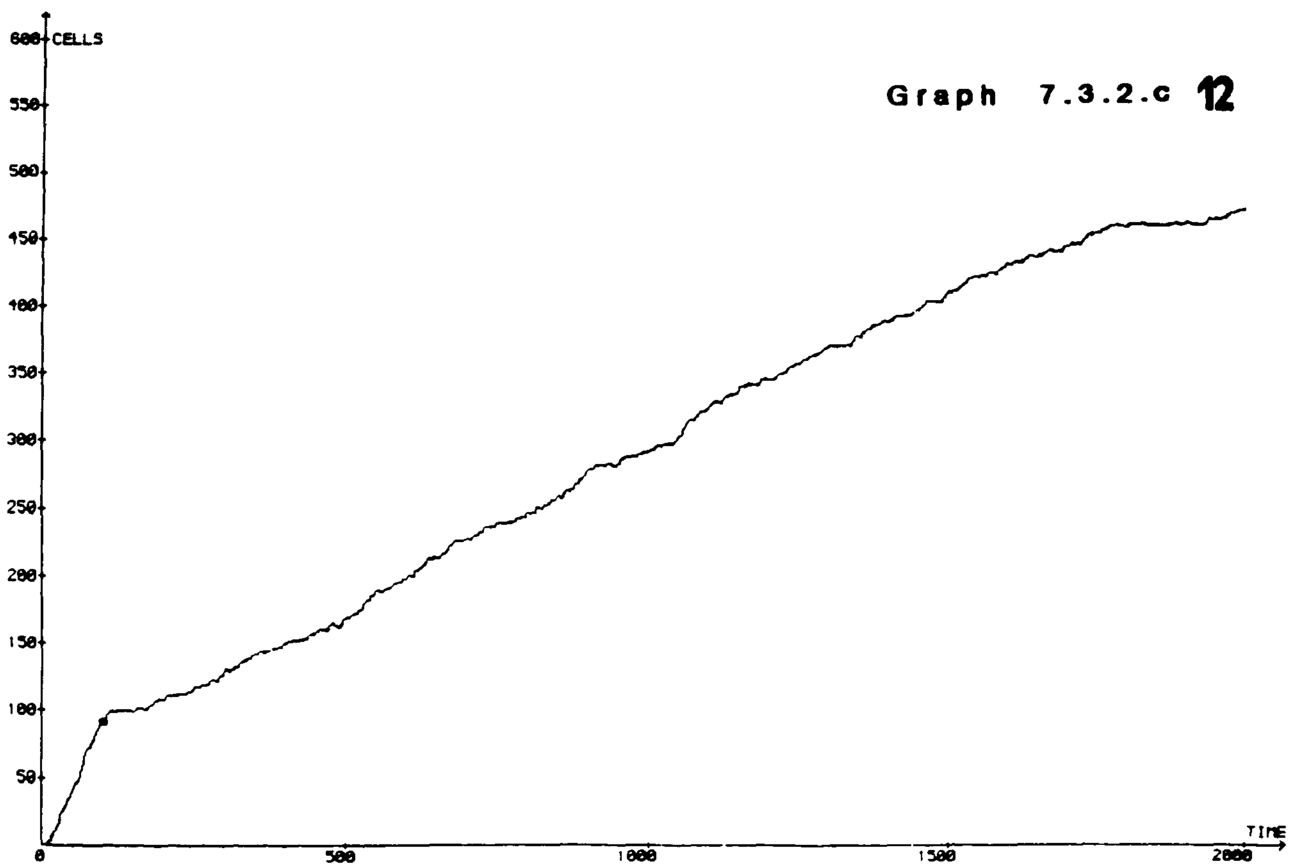
The differences in the graphs above show the effect of off-loading the demand for the Input-Output bus onto the Inter Module bus, and the subsequent knock-on effects that this has on the queue lengths for Intra Module bus queues. Graph 7.3.3.c. shows that throughout the simulation, the Input-Output bus was used for a high percentage of the time. This high Input-Output bus utilisation is a desired feature, with this graph giving possible optimum performance that could be expected from a non-segmented Input-Output bus system.

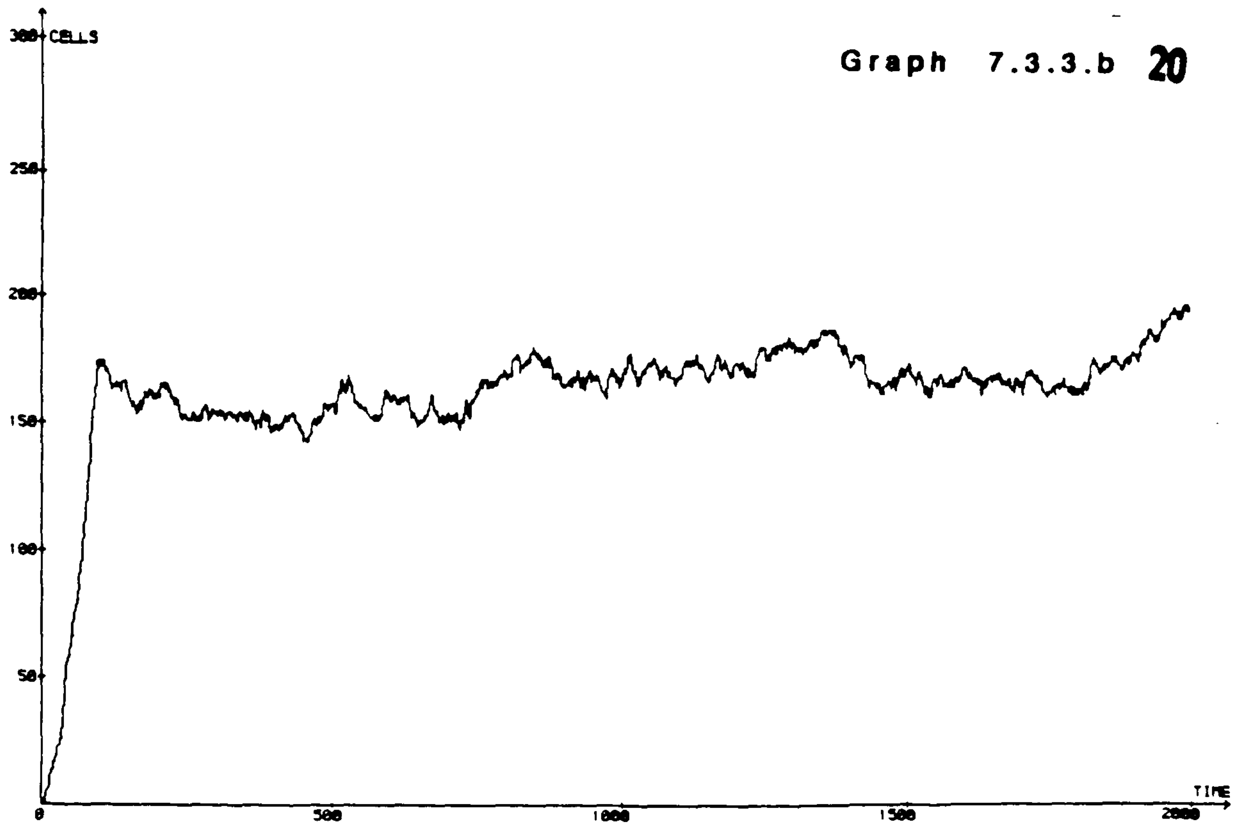

CELLS
1000
900
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
TIME
Graph 7.3.1.b 3

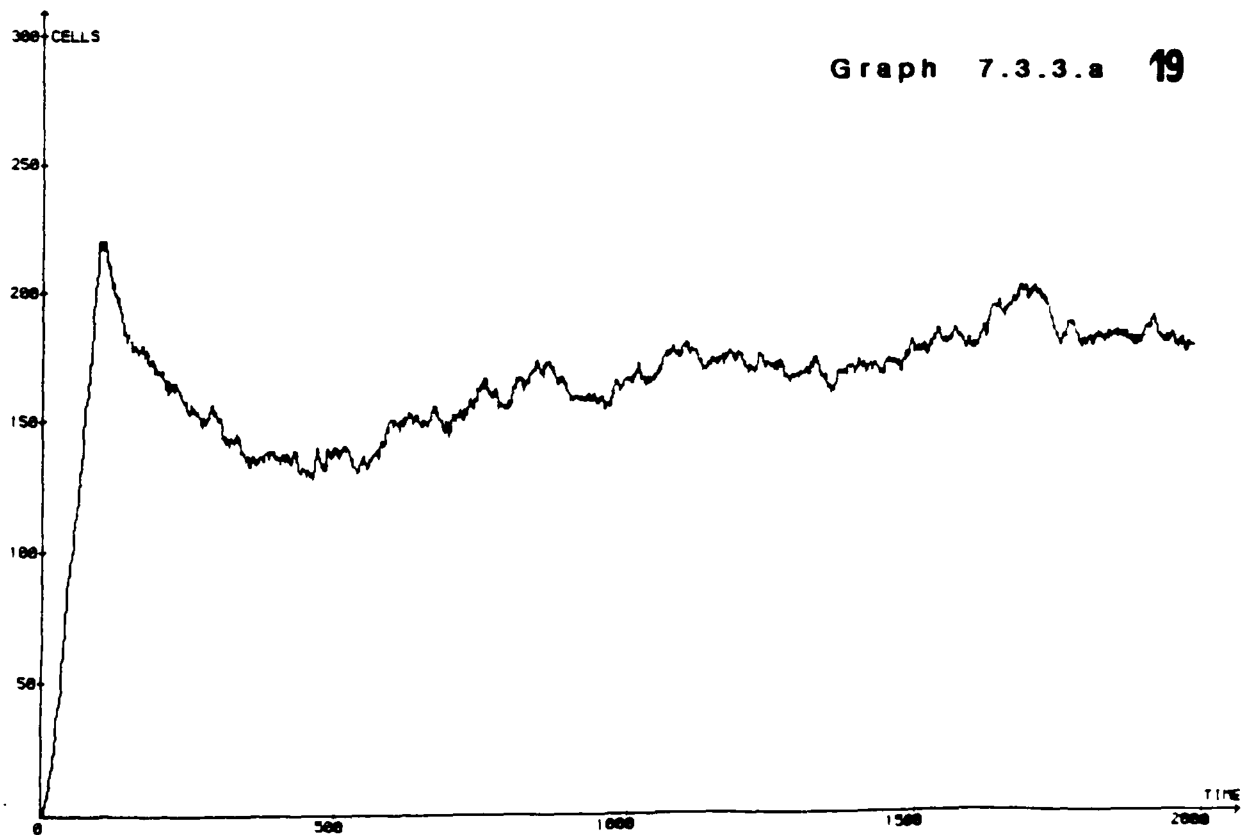

CELLS
1000
900
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
TIME
Graph 7.3.1.a 2

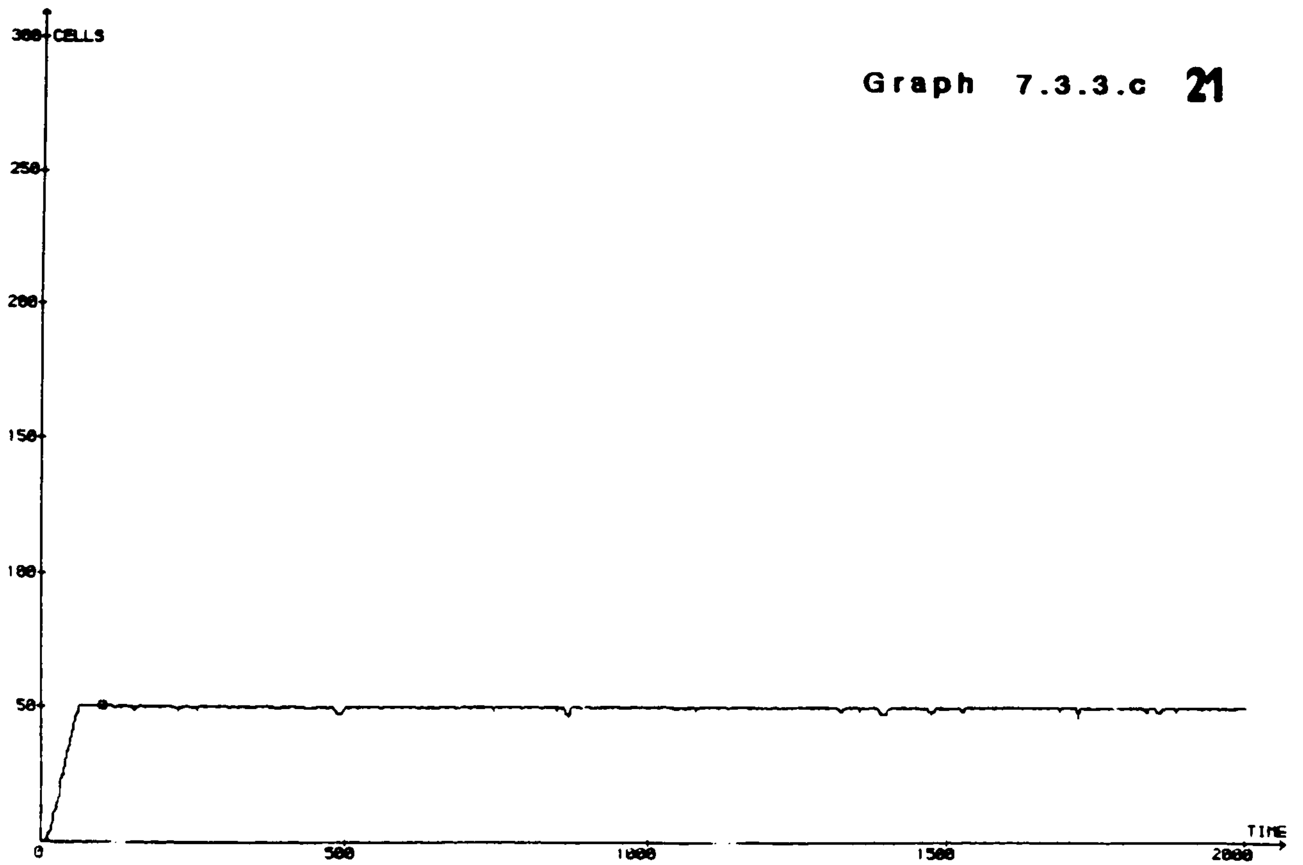

CELLS
Graph 7.3.2.b 10
400
350
300
250
200
150
100
50
0
500
1000
1500
2000
TIME


CELLS
Graph 7.3.2.a 11
400
350
300
250
200
150
100
50
0
500
1000
1500
2000
TIME


CELLS
600
550
500
450
400
350
300
250
200
150
100
50
0
500
1000
1500
2000
TIME
Graph 7.3.2.c 12


CELLS
Graph 7.3.3.b 20
300
250
200
150
100
50
0
500
1000
1500
2000
TIME


CELLS
Graph 7.3.3.a 19
300
250
200
150
100
50
0
500
1000
1500
2000
TIME


CELLS
300
250
200
150
100
50
0
500
1000
1500
2000
TIME
Graph 7.3.3.c 21

7.10. Conclusions

To understand the conclusions that lay behind the Group Processor System typical applications of the system must be considered. A Group Processor System which is being used as a database machine will require very fast searches of wide variety of databases. These systems do not require Input-Output operations as found in conventional system architectures. While on the other hand, real time process control systems demand a very high Input-Output bandwidth. The conclusion drawn from this chapter is that the Group Processor System can cater for both types of applications.

The concepts behind the Input-Output architecture of the Group Processor System is very similar to more conventional Input-Output systems. The main dissimilar feature is the distribution of terminals, as found in a multi-user system, to each of the many Modules that would make up a typical system design. The extension of the Input-Output system provided much needed flexibility for the whole Group Processor System.

The most important consideration when designing cellular systems is the need for at least one dedicated Input-Output bus. From the results, the Input-Output bus provision increased the capacity considerably. More importantly, Input-Output bus segmentation is needed to further maximise Input-Output system throughput.

In section 7.4; the 1,1,1,1 inter-connection interpretation was postponed until now. One conclusion which

must be drawn from the simulation is that this scheme has no value for IKBS. As this system is identical to the TRANSPUTER, it must be stated that as a result of these findings, the TRANSPUTER is not suitable as a cell in the Group Processor System. It is doubtful whether the TRANSPUTER does have a wide range of applications with which to be configured.

But what about the limitations of the simulation? Within the limitations of the equipment executing the simulation program; the simulation provided a wide range of results. The main problem that was encountered was the capacity of the computing equipment to process the simulation in real time. Another limitation was the Languages available on the machines. All in all, three mainframe/supermini computer systems were used to process the simulation. These systems proved to be lacking in many areas, the biggest problem was that of space limitations within the machines.

CHAPTER EIGHT

CHAPTER EIGHT

GROUP PROCESSOR
ARCHITECTURAL ENHANCEMENTS

8.0 Introduction

The Group Processor System has one major design problem. This problem relates to the functional attributes of the system when the user is interacting with many abstract processors. In this chapter this problem is analysed and a solution is proposed which overcomes most of the foreseen problems associated with the bus system.

8.1 Group Processor System Problems

To analyse the functional problems associated with interacting with many abstract processors the problems associated with user inter-action should first be understood. In this section typical terminal usage will be described.

When a user first interacts with the Group Processor System a number of abstract processors are made available. These abstract processors contain programs which may be invoked by sending the equivalent of a "start signal" to initiate program execution. Where there is a database located in the abstract processor the user may send a query to interrogate the database. These messages are sent along the system buses from the first level processor to the

abstract processor which stores the file. The main problem with the Group Processor System is that where files are very remote from the first level processor, complex inter-connection paths have to be set up on the Inter Module Bus.

These communication paths are often made more complex when they cross the communication paths of other users or other executing abstract processors. All too often there are considerable time delays between invoking the request to send a message to these remote abstract processors, and the time they are actually executed. The setting and re-setting of bus segmentation switches is very dynamic, and hence difficult to optimise for efficient bus performance.

Figure 8.1 illustrates the above problem when mapped onto a typical multi user machine. The many users within this machine architecture conflict for the two database areas. The main problems associated with user inter-action is when parallel access to these often remote areas results in bus contention. In the basic Group Processor System, bus contention will occur on the Inter-Module bus systems, although the bus system is triplicated. What is required in the bus system is a hierarchy of bus inter-connections which allows a high degree of system flexibility and structure.

8.2 T.E.S.S. Outline

The bus structure that is presented in this Thesis does not change the basic Group Processor System module

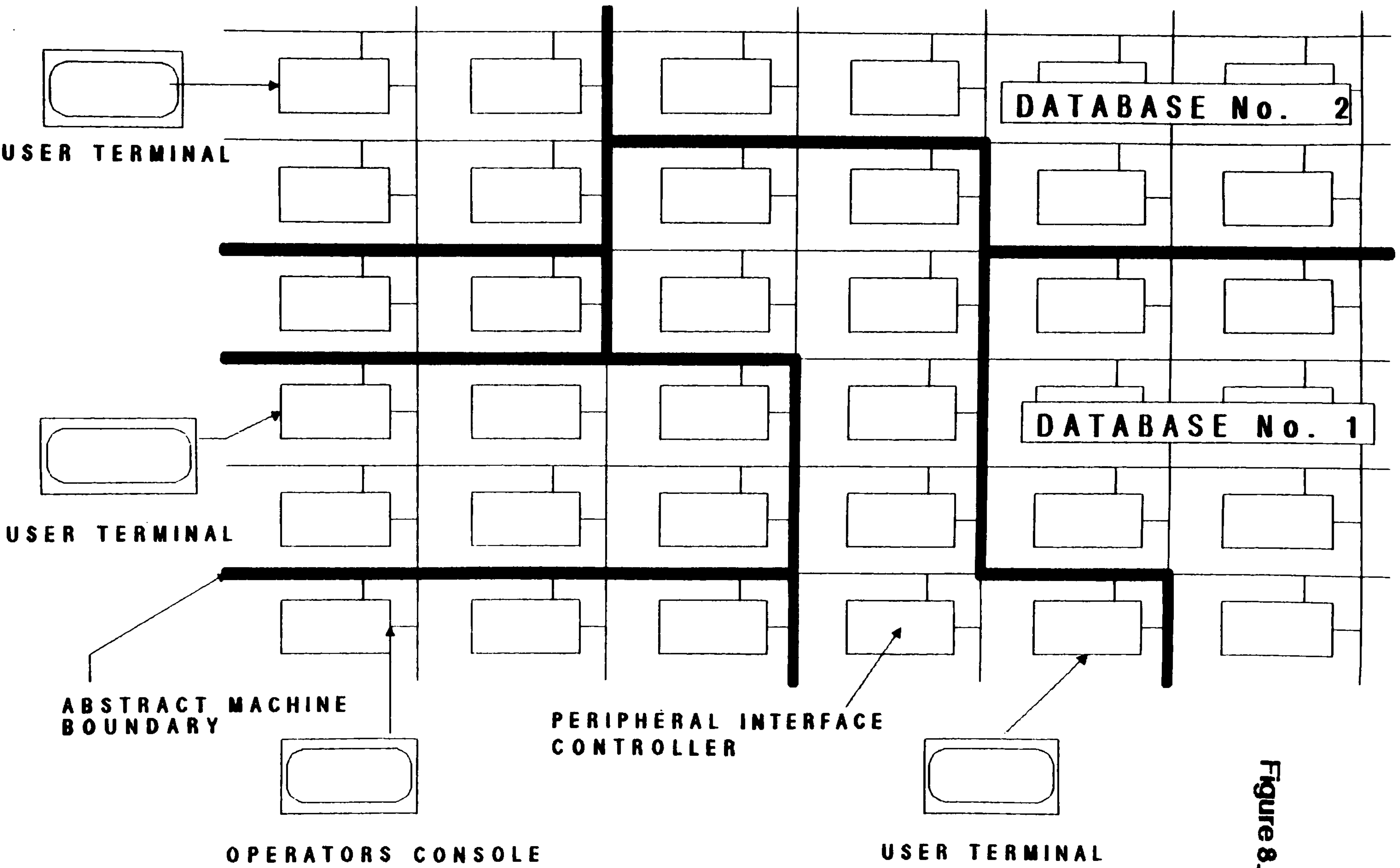


Figure 8.1

inter-connection. An additional bus system is added to the front-end, that is the interface to the user, which provides the flexible hierarchy needed in the Input-Output system as well as providing a number of extra benefits. This system is the Transaction Environment Switching System, or T.E.S.S. for short.

T.E.S.S. allows the user to link together many modules whose location is often random in location. T.E.S.S. has also changed the way in which the user's terminal is inter-connected to the modules. The terminal is no longer directly coupled to the module but is linked indirectly through a series of Input-Output switches mounted directly on each module. These Input-Output switches perform a form of Input-Output segmentation similar to that of the Inter Module bus.

A general comparison may be made between the basic Group Processor System, shown in figure 5.8, and the T.E.S.S. architecture shown in figure 8.2. The Terminal Crossbar Switch Network allows any terminal to access any one or more terminal Input-Output channels. It will be shown later that many complex Input-Output switch inter-connections may be set up.

8.3 T.E.S.S. Objectives

The T.E.S.S. provides the Group Processor System with two methods of Terminal inter-action. The first; communication paths are available to the users along the

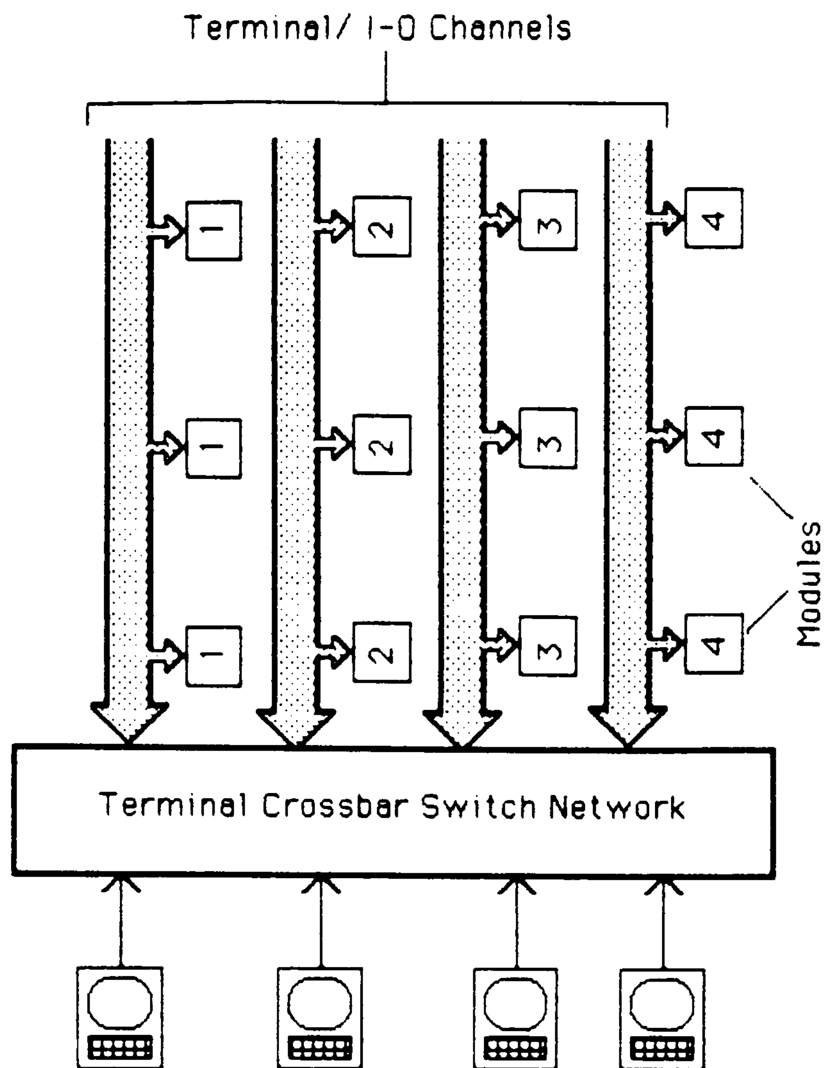


Figure 8.2 T.E.S.S. Architecture Outline

system-wide Input-Output route, that is the terminal Input-Output channels, providing users with a method of global interaction with cells. This provision is important where the user is interfacing with a database system for query or updating, this method is not supported in the basic Group Processor System. The second; direct terminal input to the module can result in information being transmitted between the machine and the user in the fastest possible time. These two dissimilar routes for terminal communication allows maximum flexibility for global interaction while maintaining direct access to cells for immediate update or query.

The application of these approaches can be visualised by considering the use of such a system in a fighting ship, an application not too dissimilar from the system outlined in figure 8.1. The Captain and senior members of the crew need instant access to information regarding the whole theatre of war. This information would be stored in cells immediately accessible to the peripheral devices, these cells would contain the software functions dedicated to updating the database. Both the Captain and the direct input devices use different methods of Input-Output communication paths because each user, either the Captain or the peripheral, have different speed requirements.

The Captain has a slow speed requirement compared to the high speed data input from, for instance, the ship's radar system. Therefore, the Captain uses the system wide Input-Output bus, that is the terminal Input-Output channel,

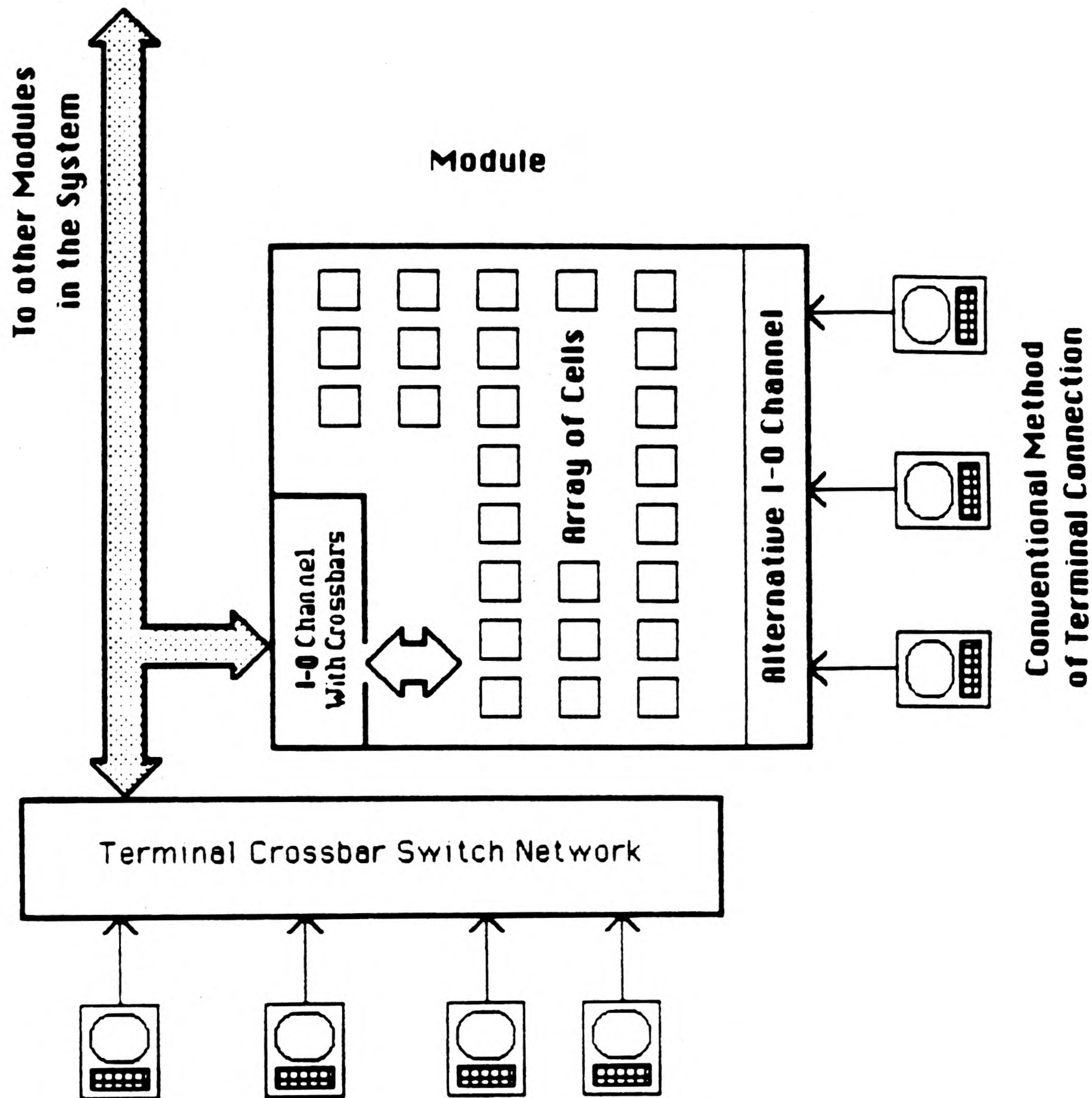


Figure 8.3 Terminal Alternatives With T.E.S.S

whereas the peripheral uses the conventional method of direct access to the module. Both of these two methods of Input-Output are shown in figure 8.3. The important objective here is, the requirement for a dedicated function of the peripheral resulting in optimised performance, and the generality and speed reduction of the T.E.S.S.

8.4 T.E.S.S. Operation

Each cell within the Group Processor System has an Input-Output bus. This bus handles most Input-Output traffic and serves Input-Output requests on a first in first out basis. This first in first out queue is identical to the queuing processor on the other system buses as described by Quick.

To understand the operation of the T.E.S.S, it is useful to study the actions of a crossbar switching system. A crossbar switching system allows many processors to communicate with many memory units. Figure 8.4.a. shows the crossbar architecture as many processors on the left side of the diagram, while the memory units are located at the bottom of the diagram.

Additional units e.g. Input-Output channels may also be coupled to the crossbar so as to form an integrated system environment which conforms to the polymorphic rules, that is, any computing requirement may be constructed from a mixture of processors, memory and Input-Output unit by the

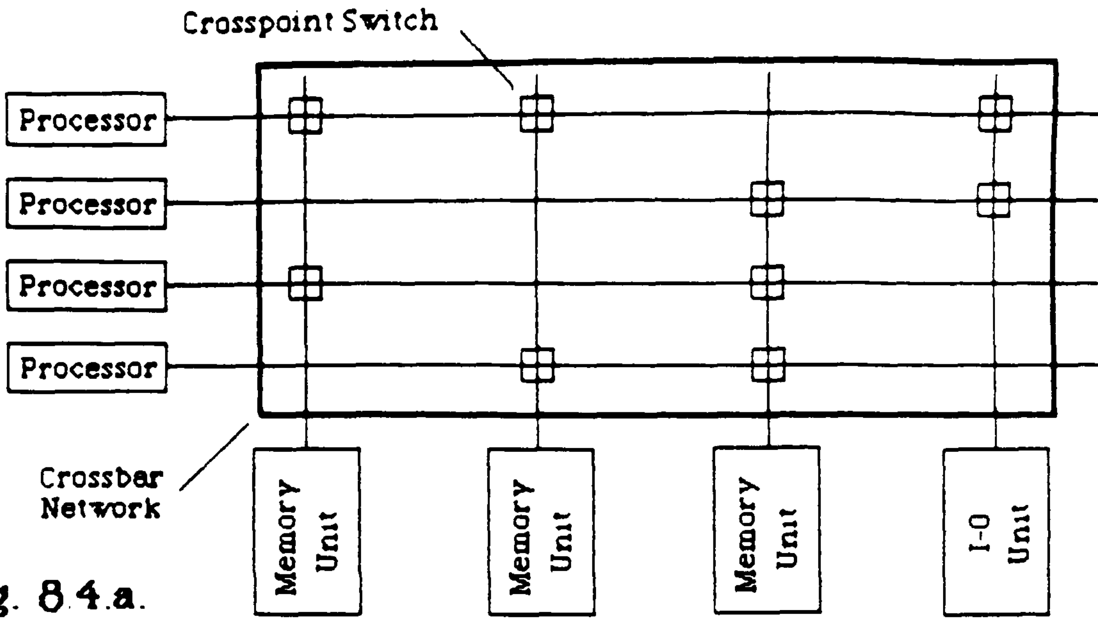


Fig. 8.4.a.

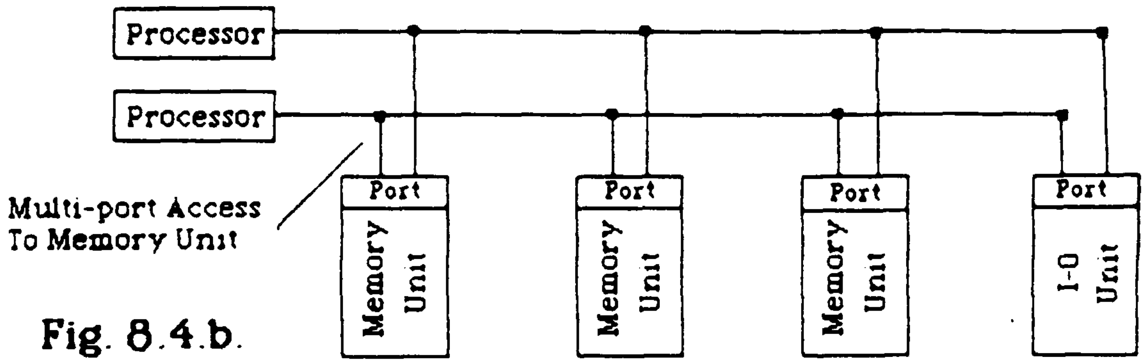


Fig. 8.4.b.

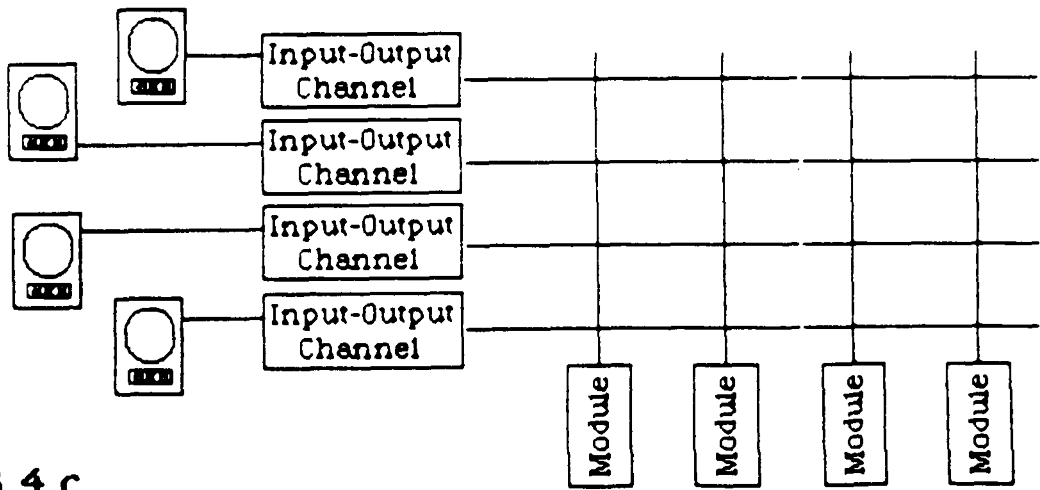


Fig. 8.4.c

Figure 8.4 Ported Memory Concepts

enabling/disabling the communication between these units. The crosspoint switch performs this switching function.

The main problem with this classic architecture is that the crossbar is a single point of failure. A more normal implementation of the crossbar mechanism is to partition the crossbar into it's many crosspoint constituent parts, and distribute these to the memory units. This forms the basis of the multi-ported memory concept of figure 8.4.b. In this Thesis the processor unit no longer exists, as the many processors are distributed to the cells within the Group Processor System Module.

The overall architecture of the T.E.S.S, shown in figure 8.4.c., illustrates a typical structure of the Input-Output bus, and its inter-connection to each module. Each cell has a number of serial Input-Output lines which provide the many inter-connection patterns that are needed in the T.E.S.S. These Input-Output lines are driven from a single Input-Output channel within the cell, and the Input-Output line is mapped to the cell when the cell receives its dedicated function. This Input-Output mapping informs each cell which has to perform user Input-Output operations, and which of the cell Input-Output ports to use when performing Input-Output functions. Technological limitations are the only consideration which the computer architect has to contemplate when designing T.E.S.S. into any Group Processor System.

Future Terminal Expansion

Crossbar Switch

Crossbar Switch

Crossbar Switch

Crossbar Switch

To Terminals & System I-O Ports

Module I-O Inter-connection Network

Global Bus

Module Buses

To I-O Ports on Cells

Figure 8.5 Four Terminal/Crossbar Module Inter-connection

The four serial lines to the Terminal & System Input-Output Ports, figure 8.5, may provide maximum triple modular redundancy to the terminal. This can be achieved by either sending output along three serial lines from the cell or, by mapping any of the Inter/Intra Module buses to the serial lines radiating to the user terminals.

8.5 Module/Channel Interface

The Module/Channel Input-Output architecture, which is mounted on the module, is shown in figure 8.6. The Inter and Intra module buses, together with the serial Input-Output lines from each cell terminate at the T.E.S.S./module network interface. From there on, the communication lines are all serial, ultimately converging at the user's terminal. This parallel to serial convertion allows Off-loading to take place on the Inter/Intra Module buses. The addressing mechanism sets/resets the segmentation switches within the channel interface. These switches can have one of many configurations set, so that they can pass data along the serial lines to and from the user's terminal. These switch states are controlled by the Global Operating System by passing messages to the module Input-Output channel attached to the module containing the Global Operating System. Figure 8.8 shows these switch states and the channel interface addressing mechanism.

The module Input-Output Inter-connection Network is also a crossbar system. The operation of this inter-connection network is very similar to the crossbar

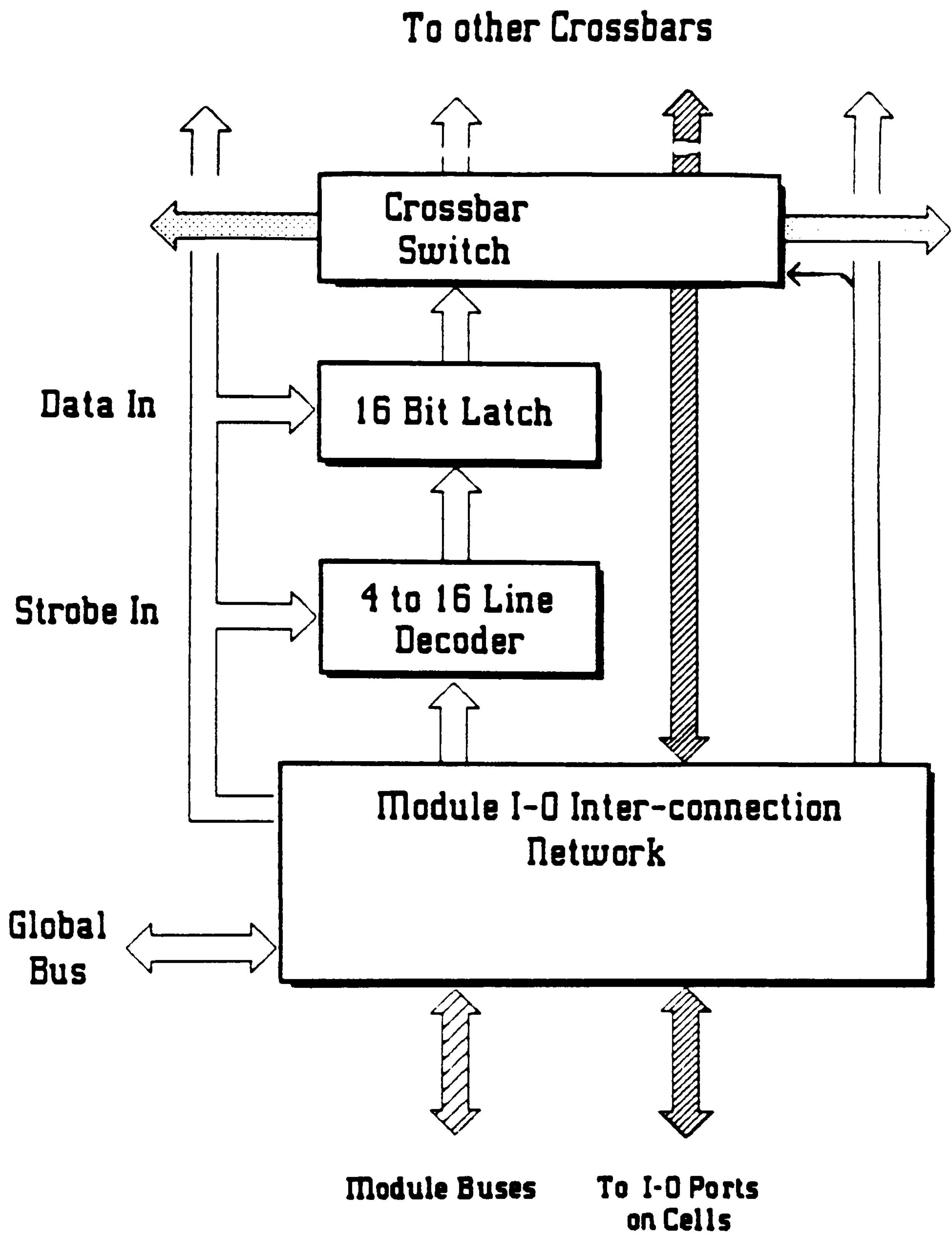


Figure 8.6 Module/Crossbar Inter-connection Network

operation described above. Whereas the mapping of the Terminal Communication Lines to each Module is performed through the Module Input-Output Inter-connection Network, the Module Input-Output Inter-connection Network itself is controlled and set up by the Global Operating System via the Global Operating System Bus. If there is a failure in this bus, the Global Operating System may use the alternative Inter/Intra Module Buses.

8.6 Crossbar Operation

The architecture of the crossbar switch includes an assembly of crosspoints and a control memory. This architecture is shown in figure 8.6, where the crossbar contains a 4 X 4 assembly of crosspoints and a 4 to 16 line address decoder. There are 16 latch circuits needed to maintain the crosspoints in the required state. Any one of the sixteen crosspoints may by selected by applying a logical one or zero. Any number of crosspoints can be ON at any one time.

The basic architecture of the crossbar and the module/Input-Output channel is extensible. That is, additional crossbar systems can be added to the basic architecture without any modification to the module/Input-Output channel interface. While it is possible to add as many Terminal lines as possible to the basic crossbar, in reality there is a limitation to the maximum number because it is a direct relationship with the number of select lines that make up the overall crossbar addressing

mechanism. Figures 8.7 and 8.8 show the control of the module Input-Output Inter-connection Network in more detail.

8.7 Bus Arbitration

The simple distributed bus arbitration system in the basic Group Processor System provides a high degree of bus segmentation. However, the flow of bus requests is always to the root arbiter which, in the basic system, is physically located at the module which is the root module. This system seems inflexible, if the the root module fails then the user's whole environment fails. This system limitation should be refined so that the failure of the root bus arbiter, does not produce systemic failure of the user's environment

In this Thesis it is proposed to keep the overall bus arbitration network of the basic Group Processor System, but to add a bus request line in the other direction. That is, if the root arbiter fails than the system can reconfigure itself so that the next module (root + 1) takes over as the root arbiter. The practical implementation of the additional bus request line would not result in an additional physical line, as the existing bus request line can be re-directed when it reaches the the bus arbiter.

The bus arbitration network would have maximum flexibility to respond to bus inter-connection request if the arbitration was refined. The current bus arbitration architecture does not allow for separating the functional

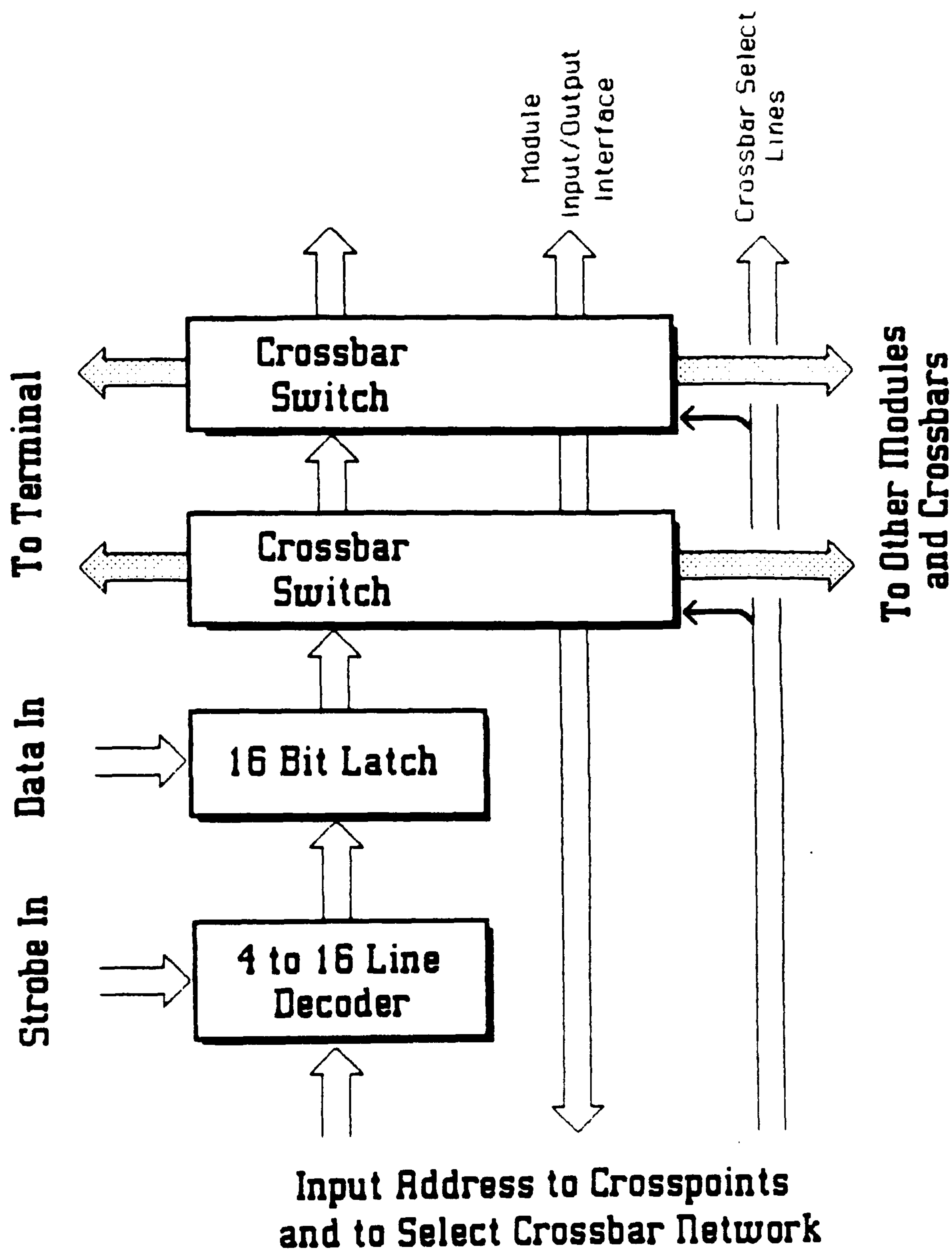
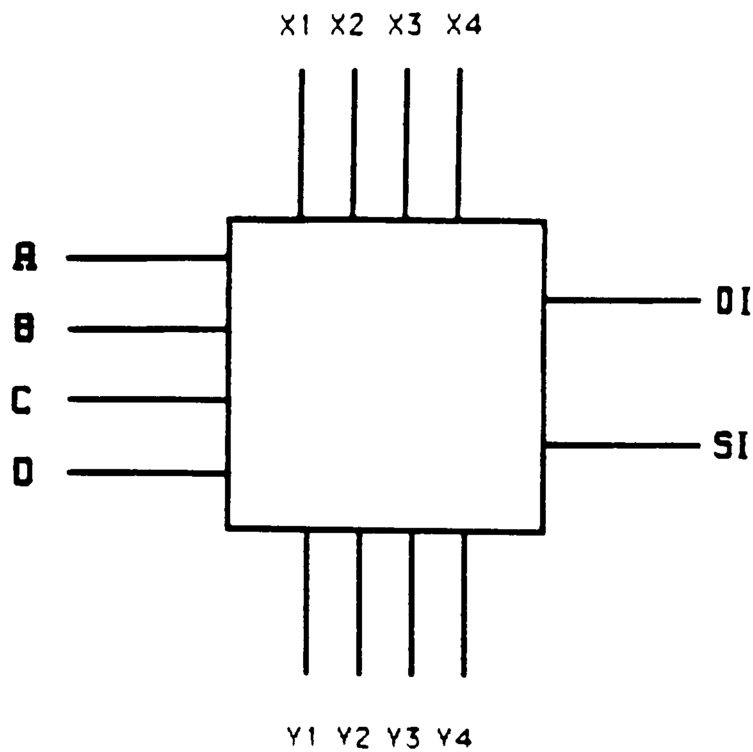


Figure 8.7 Crossbar Addressing Mechanism



Functional Diagram

Address				Select	Address				Select
A	B	C	D		A	B	C	D	
0	0	0	0	X1Y1	0	0	0	1	X1Y3
1	0	0	0	X2Y1	1	0	0	1	X2Y3
0	1	0	0	X3Y1	0	1	0	1	X3Y3
1	1	0	0	X4Y1	1	1	0	1	X4Y3
0	0	1	0	X1Y2	0	0	1	1	X1Y4
1	0	1	0	X2Y2	1	0	1	1	X2Y4
0	1	1	0	X3Y2	0	1	1	1	X3Y4
1	1	1	0	X4Y2	1	1	1	1	X4Y4

Truth Table

Figure 8.8 Crossbar Functional Diagram and Truth Table

roles of the Inter Module Bus and the bus arbitration. With the new bus arbitration mechanism, the root arbiter for any abstract processor, abstract machine, or the whole machine for that matter, can be relocated anywhere within the machine. The basic Group Processor System architecture does not have the flexibility to re-designate the priority of the arbiters. This could be important for defense systems where the centre of action within the computing system changes from a semi-leisure state, where the computation demands are usually centred on the stores and entertainments, to the war situation where the stores and liesure are of the lowest importance.

8.8 Summary

The simple bus architecture presented by Quick is suitable for VLSI implementation. With this Thesis, the bus architecture is further refined by adding increased Input-Output reliability and increased flexibility with T.E.S.S.

The provision of the Transaction Environment Switching System gives the system architect a useful method of swapping between abstract machines, as well as coupling many abstract machines together. T.E.S.S. has the ability to build many hierarchical abstract machines within the Group Processor System. The simulation has proved that T.E.S.S. can work in a real time application, such as those found in IKBS.

The architecture of T.E.S.S. solves a number of problems for potential users of Group Processor architectures. While these problems have been discussed above, there still remains the problem of whether the addition of T.E.S.S. will enhance or degrade the performance of the Group Processor System. As well as modelling the overall architecture of the Group Processor System, the simulation of the Group Processor System has given a realistic feedback on the potential performance of T.E.S.S.

The refinement of the bus arbitration system makes the Group Proceesor System a highly desirable re-configureable fully distributed machine. The enhanced capability of the new arbitration system makes the Group Processor System more flexible from the system programmers viewpoint, as the user's execution environment can be manipulated to support almost any module inter-connection pattern.

The most important addition to the Group Processor's Cell is the addition of a dedicated Input-Output bus, which is connected to each and every cell in the system. While this further complicates an already complex integrated circuit design, it is important that at least one dedicated Input-Output bus is incorporated in the final cell design.

CHAPTER NINE

CHAPTER NINE

CONCLUSIONS AND FURTHER RESEARCH

9.0 Introduction

This research has considered many existing and new issues relating to fifth generation computing systems. The Thesis has continued the abandonment of the Von-Neumann architecture by considering the Group Processor System as a viable system for further refinement, suitable for applications in Intelligent and Knowledge Based Systems.

In this chapter, we conclude the Thesis by identifying the design enhancements made to the Group Processor System concept. Typical performance characteristcs are also summarised for cellular computer systems, as applied to the Group Processor System architecture. The chapter also summarises the significant issues which computer architects must overcome if they are to design very high performance fifth generation computer systems.

9.1 Research Initatives

The research initiatives of Japan, the U.S. and Alvey seem to approach different areas of investigation. While this is useful from a pure research point of view, there is a possibility that the final battle for the commercial exploitation of the research may go, by default, to Japan.

Japan has concentrated on the hardware aspects of the research, while Alvey seem content to concentrate on software issues. Consequently, the Japanese are likely to be the major manufacturers of computing equipment in the future.

The ten major application catagories outlined by the Japanese demand very flexible Input-Output systems which must respond to user interaction in real time. One of the conclusions drawn from this research is that machines which have the power to process Intelligent and Knowledge Based System programs are not that responsive to terminal interaction. At least one paper "Down grading to a VAX", published by the DEC User's Group, suggests that the older PDP 11/70 was more of an interactive machine than the more modern VAX. It is surprising therefore, that the current Intelligent and Knowledge Based System community, within in the U.K., sees the VAX machine as THE machine for Intelligent and Knowledge Base System research.

The direction that the British research community is currently taking must change. The direction with which to approach the next five years, if the U.K. wishes to be considered a manufacturer of computing equipment, is to provide more initiative for the development of hardware. A starting point for this would be the stimulation of research into computer architecture, and not to concentrate purely on VLSI development. Failure to do this will almost certainly result in the U.K. being a net importer of computing equipment.

Future research must consider a wider range of activity if the British computer industry is to have a future on as wider front as possible.

9.2 Computer Architecture

There has been little advance in the design of new computer architectures. Current state of the art designs have been refinements of advances made many years ago. The only advances that have been made are related to technological advances made in VLSI. Computer manufacturers have capitalised on VLSI advances at the expense of maintaining architectural advances. The exception to this has been INMOS's TRANSPUTER.

It has been shown in this Thesis that the TRANSPUTER suffers from major design imperfections which will limit the device's application in many Intelligent and Knowledge Based System areas. The TRANSPUTER has an application base, but the simulations undertaken in this research have clearly indicated that the TRANSPUTER is not a device for modern interactive Intelligent and Knowledge Based Systems. The TRANSPUTER can work effectively only when configured as a backend processor attached to a conventional computing system.

The TRANSPUTER has to be refined in a number of areas before it can be accepted for Intelligent and Knowledge Based Systems. The number of communication buses has to be dramatically increased before the device can be thought of

as a useful fifth generation product. However, the TRANSPUTER does provide researchers will a starting point with which to design and build cellular computing equipment. As these systems will be inadequate in terms of processing power, it is excuseable that many researchers many see the development of cellular systems based on the TRANSPUTER as a waste of time.

9.3 The Group Processor System

The basic Group Processor System suggested by Quick has a number of limitations. The most important of these limitations are the severe restrictions on performance. By comparision, the basic Group Processor System does have major performance advantages over the TRANSPUTER.

A wide range of Group Processor architectures were simulated in this Thesis. The result of the simulations can summarised as follows.

The basic Group Processor System, as described by Quick, has little opportunity for varying the state variables within either the architecture or the operating system. The major problem associated with the basic Group Processor System is the relatively poor bus service rate. This basic Group Processor architecture is seen only as a fundamental building block for design variations on the original Group Processor concept. The only variation that can take place with this configuration is routing Input-Output operations to all buses.

One important requirement in any cellular system is that of immediate communication with other cells within the execution environment. Another requirement is to provide a very fast Input-Output system both to the user and to the program execution environment.

The need for fast turn-around in Input-Output traffic cannot be met with the current cell design. With this in mind, the provision of an dedicated Input-Output bus offers a "potential" speedup in Input-Output communication. It is important to note here that an additional bus has been provided to each cell within the system.

The extended Group Processor architecture represents the first variation on the basic Group Processor System. The architecture found in this system is more complex than the basic Group Processor System. In this configuration, the basic cell has an extra bus added in the form of a dedicated Input-Output bus. This configuration provides the systems programmer with a wider range of variables with which to fine tune system performance. This architecture provides a compromise in system complexity, and the results of the simulation proves that the variations in bus scheduling, have a profound effect on system performance.

The multi user Group Processor highlights one of the Group Processor's basic features, that is system segmentation both at the program execution level within the Group Processor System, as well as in the Input-Output system. The performance of the Input-Output system is the

one factor which limits the Group Processor System's application in Intelligent and Knowledge Based Systems. The results of the simulation have clearly shown that system segmentation greatly reduces bus contention, and consequently improves system performance.

The provision of an Off-loading factor was critical for improving system performance. The most important consideration when incorporating an Off-loading Factor is maintaining moderation in the length of the Input-Output queue. The lower the Off-loading Factor the better the Input-Output performance, but this has the effect of decreasing other bus performance. For systems that need higher Input-Output throughput an Off-loading Factor of 30 is recommended.

9.4 T.E.S.S.

The major architectural contributions made by this research is in the design of T.E.S.S.. T.E.S.S., the Transaction Environment Switching System, provides the solution to a number of important problems which have limited advances in the Group Processor System. The most interesting results relate to the segmentation of the Input-Output bus. The segmentation of the Input-Output system enables each module to have its own dedicated Input-Output channel. This Input-Output system can be coupled together to provide either general purpose networking facilities or dedicated individual user interfaces to the Group Processor System.

The concepts behind the Input-Output architecture of the Group Processor System is very similar to more conventional Input-Output systems. The main dissimilar feature is the inter-connection of terminals to each of the many Modules that would make up a typical system design. The extension of the Input-Output system provides much needed flexibility for the whole Group Processor System.

The provision of the Transaction Environment Switching System gives the system architect a useful method of swapping between abstract machines, as well as coupling many abstract machines together. T.E.S.S. has the ability to build many hierarchical abstract machines within the Group Processor System. The simulation has indicated that T.E.S.S. can work in a real time application, such as those found in IKBS.

9.5 Future Research

This research has highlighted a number of important issues which can be further developed. On the software side, there is a need for a more realistic execution environment within the simulated Group Processor System. Such realistic execution will provide more accurate figures with which to fine tune the Group Processor System. The simulation at the instruction level, while being very desirable, is not realistic with the current computing machinery available to the research team. This simulation would give very accurate feedback on the performance of a real Group Processor

System, particularly if the simulation was supporting the language and operating system level.

On the hardware side there is a need for more detailed design. While the current researchers have the capability to undertake a more detailed study, it would be a waste of research effort if the finalised design were not integrated into silicon. Therefore, very little detailed hardware need progress until there is a commitment to implement the final design. While designing a cellular system around the TRANSPUTER would be of limited value; there would be a definite advantage for designing a very general bus inter-connection system which could support may devices of various kinds. These devices would be specialist processors dedicated to either general purpose computing functions, or more specialist design to optimise performance e.g. sorting or search engines.

On the wider aspects of software development for the Group Processor System; research into language translation should consider the relative performance of data driven and demand driven systems. While the computer architect can optimise the hardware facilities within the overall system design, it is the programmer who provides the useful features to the system users. Inefficient programming and/or concepts at the software level can render the hardware optimisation useless. Therefore, the analysis of the general performance of the Group Processor System is considered complete with this Thesis.

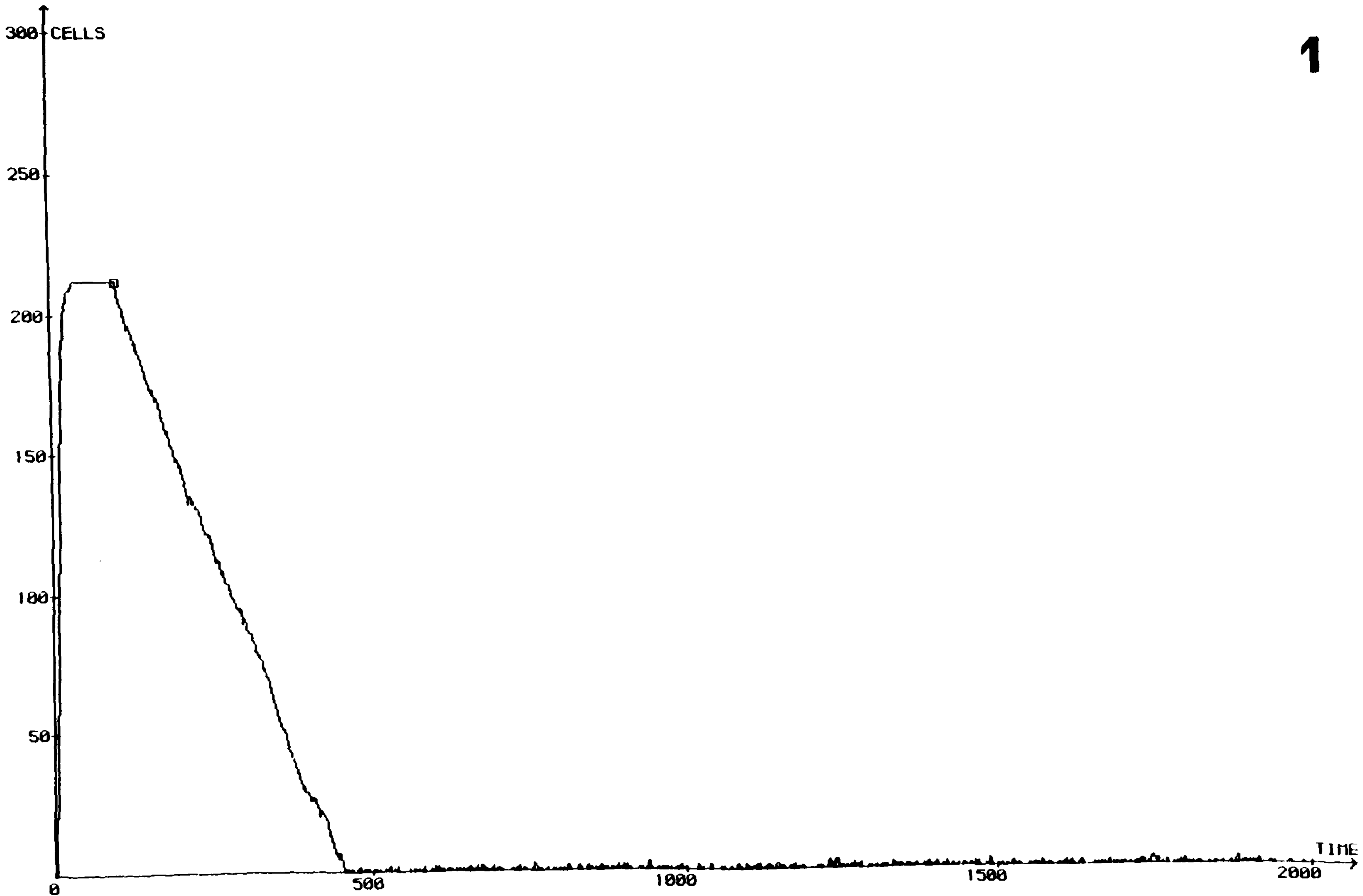
9.6 Summary

The Thesis has proposed a number of new ideas which have considerably increased the overall performance of the Group Processor System.

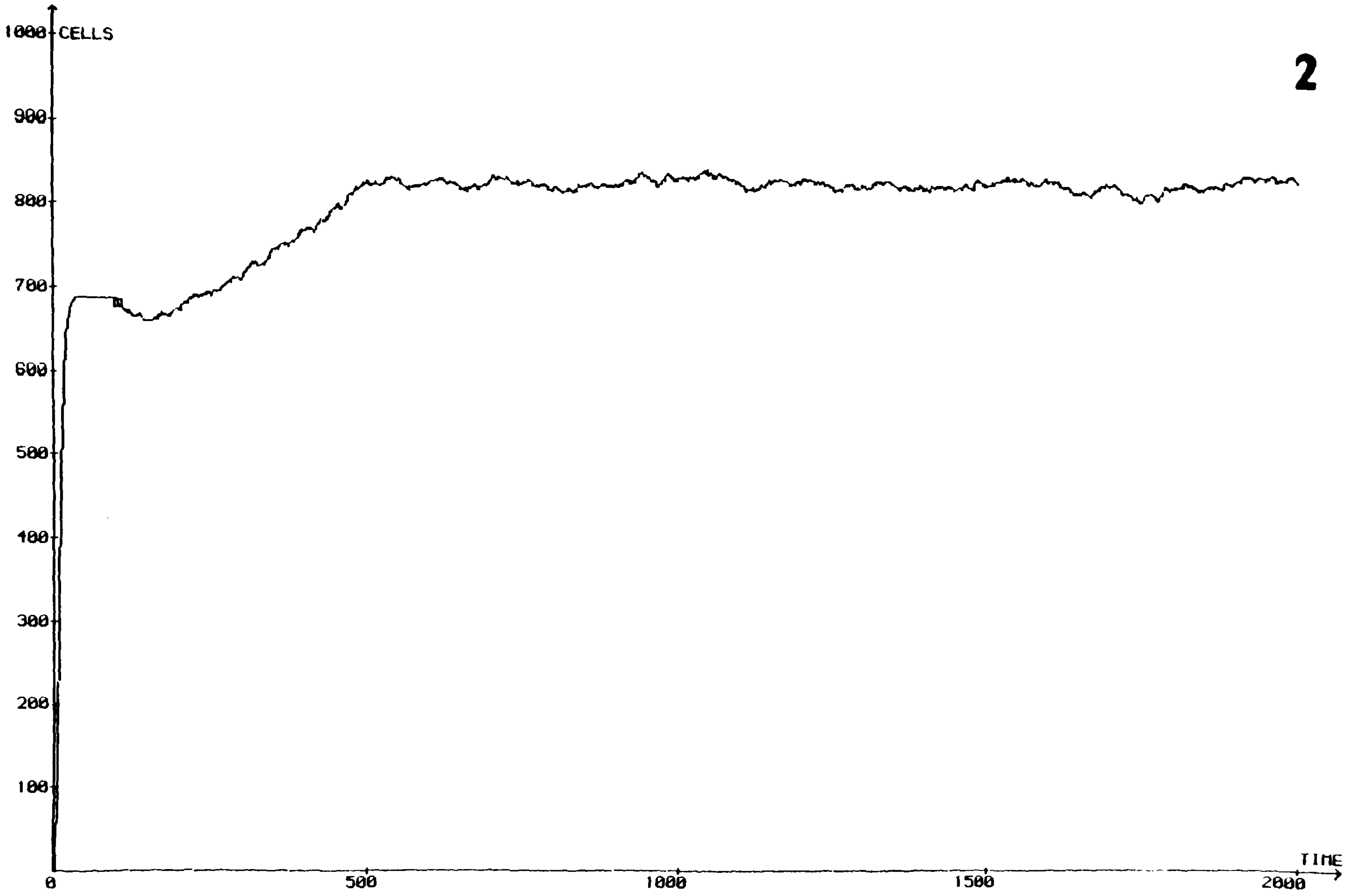
This research has concluded an important phase in the development of cellular systems of the Group Processor System type, and a number of important advances made. Most of the problems encountered during this time have been either limitation of current computer systems, or the problems associated with communicating ideas with other researchers. The problems associated with the computing machinery can only be solved by building a Group Processor System, thereby providing the ideal environment with which to investigate a massively parallel Group Processor System. The problems associated with communication can only be solved by maintaining a team effort in developing Group Processor Systems.

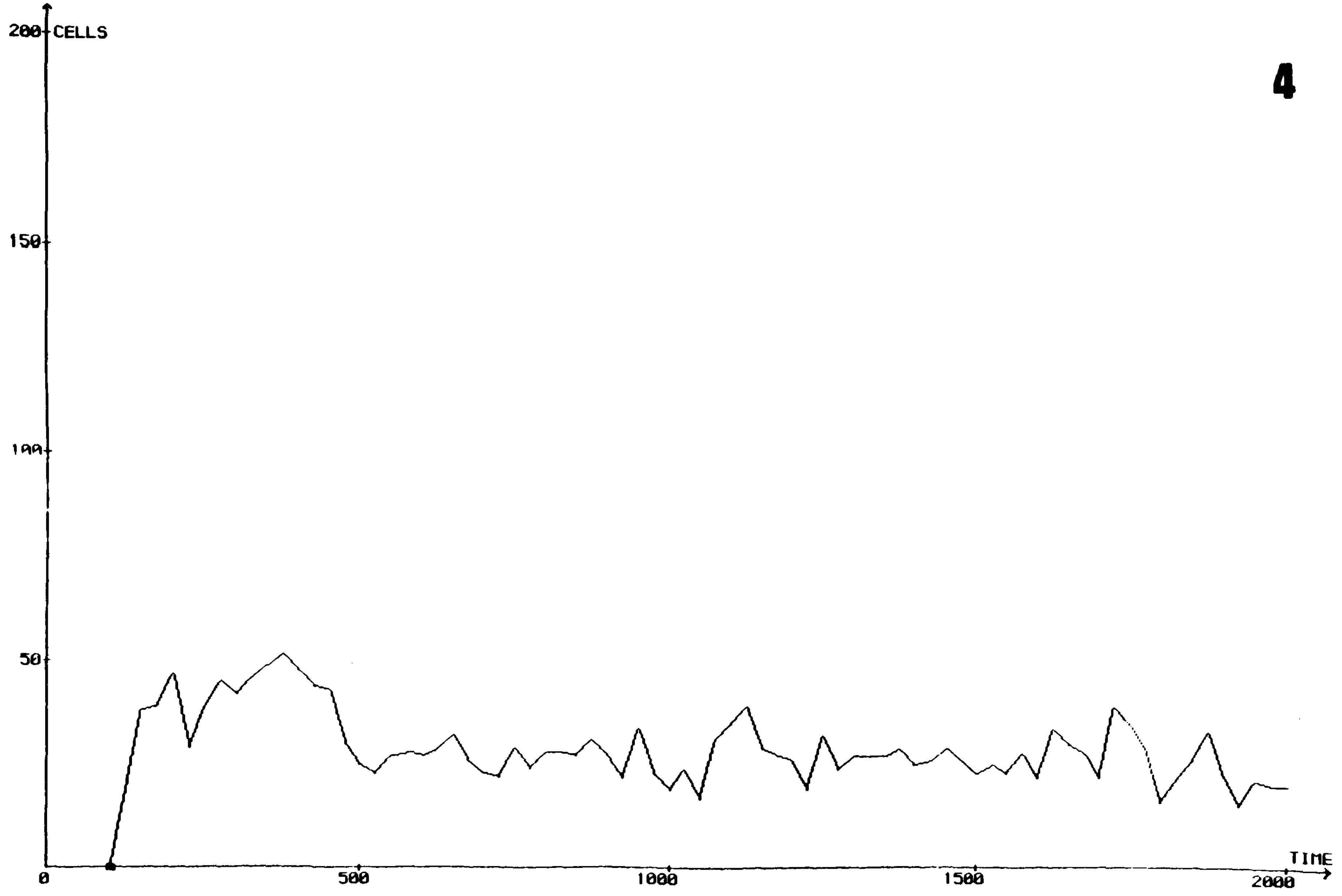
APPENDIX ONE

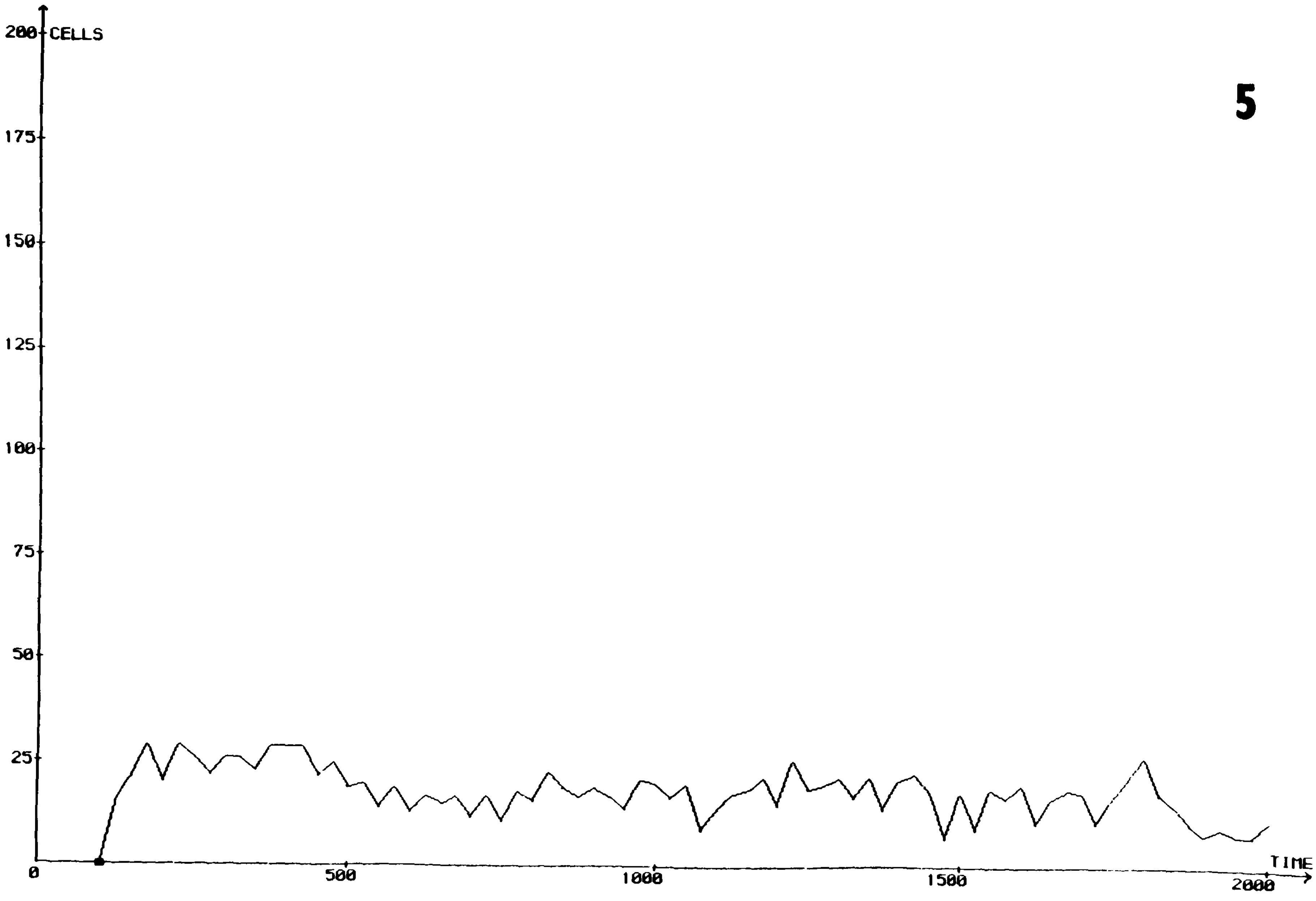
GRAPH No.	DESCRIPTION	BUS CONFIG	IO - COUNT	CONST	CONFIG
1	Global Bus Queue	1,3,3,0	0	0	0
2	Intra Bus Queue	1,3,3,0	0	0	0
3	Inter Bus Queue	1,3,3,0	0	0	0
4	Free Cells	1,3,3,0	0	0	0
5	Active Cells	1,3,3,0	0	0	0
6	Waiting Cells	1,3,3,0	0	0	0
7	Communicating Cells	1,3,3,0	0	0	0
8	Queue Waiting Time	1,3,3,0	0	0	0
9	Global Bus Queue	1,3,3,1	0	0	0
10	Intra Bus Queue	1,3,3,1	0	0	0
11	Inter Bus Queue	1,3,3,1	0	0	0
12	Input/Output Bus Queue	1,3,3,1	0	0	0
13	Free Cells	1,3,3,1	0	0	0
14	Active Cells	1,3,3,1	0	0	0
15	Waiting Cells	1,3,3,1	0	0	0
16	Communicating Cells	1,3,3,1	0	0	0
17	Queue Waiting Time	1,3,3,1	0	0	0
18	Global Bus Queue	1,3,3,1	0	0	0
19	Intra Bus Queue	1,3,3,1	0	0	0
20	Inter Bus Queue	1,3,3,1	50	20	0
21	Input/Output Bus Queue	1,3,3,1	50	20	0
22	Free Cells	1,3,3,1	50	20	0
23	Active Cells	1,3,3,1	50	20	0
24	Waiting Cells	1,3,3,1	50	20	0
25	Communicating Cells	1,3,3,1	50	20	0
26	Queue Waiting Time	1,3,3,1	50	20	0
27	Global Bus Queue	1,3,3,1	50	20	0
28	Intra Bus Queue	1,3,3,16	50	20	0
29	Inter Bus Queue	1,3,3,16	50	20	0
30	Input/Output Queue	1,3,3,16	50	20	0
31	Free Cells	1,3,3,16	50	20	0
32	Active Cells	1,3,3,16	50	20	0
33	Waiting Cells	1,3,3,16	50	20	0
34	Queue Waiting Time	1,3,3,16	50	20	0
35	Global Bus Queue	1,3,3,16	50	20	0
36	Intra Bus Queue	1,3,3,16	50	20	0
37	Inter Bus Queue	1,3,3,16	50	20	0
38	Input/Output Bus Queue	1,3,3,16	50	20	1,(1,2)
39	Input/Output Bus Queue	1,3,3,16	50	20	3,(3,4,5)
40	Input/Output Bus Queue	1,3,3,16	50	20	6,(6,7,8)
41	Input/Output Bus Queue	1,3,3,16	50	20	9,(9,10)
42	Input/Output Bus Queue	1,3,3,16	50	20	11,(11.12,13)
43	Input/Output Bus Queue	1,3,3,16	50	20	14,(14)
44	Input/Output Bus Queue	1,3,3,16	50	20	15(15,16)
45	Free Cells	1,3,3,16	50	20	0
46	Active Cells	1,3,3,16	50	20	0
47	Waiting Cells	1,3,3,16	50	20	0
48	Communicating Cells	1,3,3,1	50	20	0
49	Queue Waiting Time	1,3,3,1	50	20	0

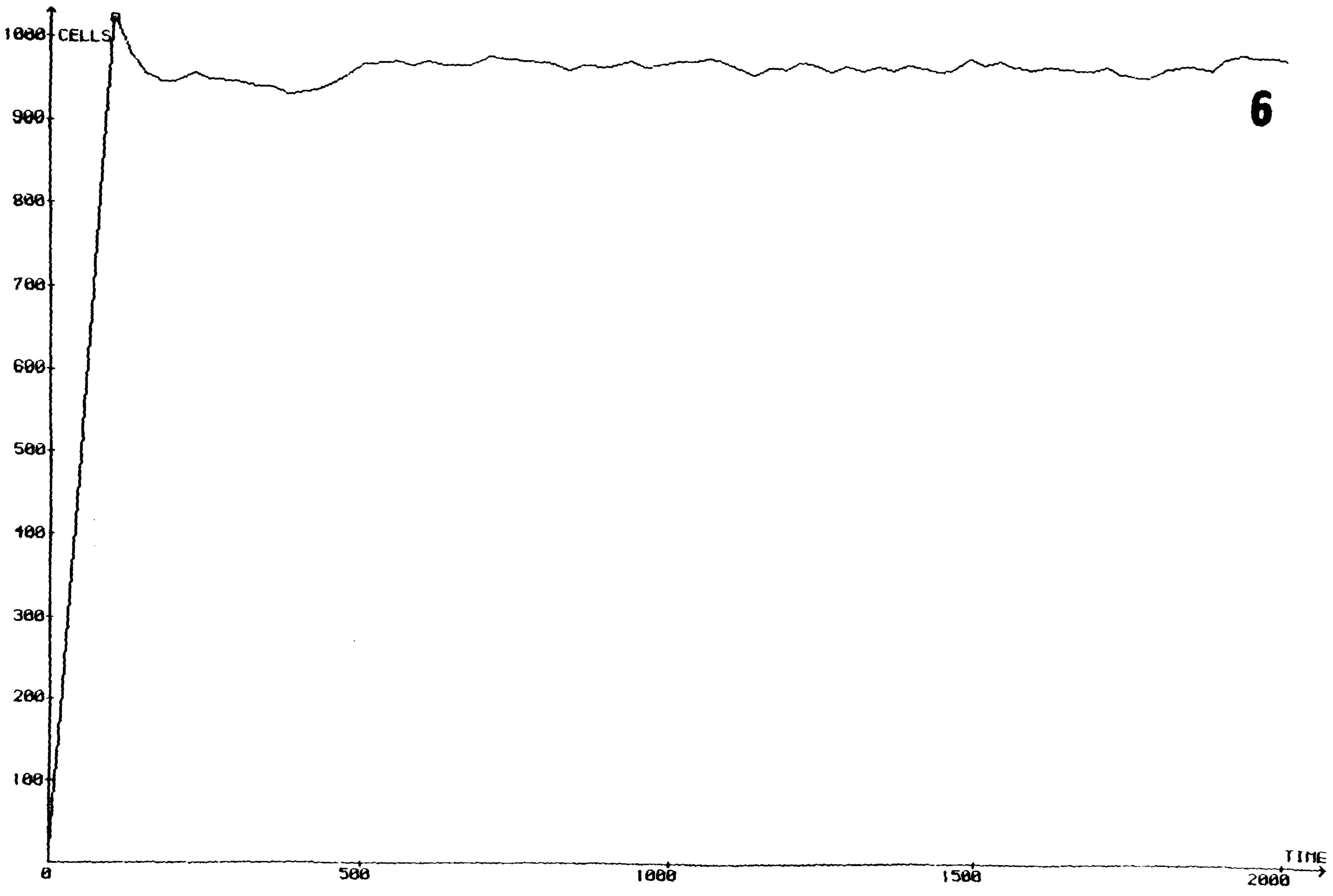


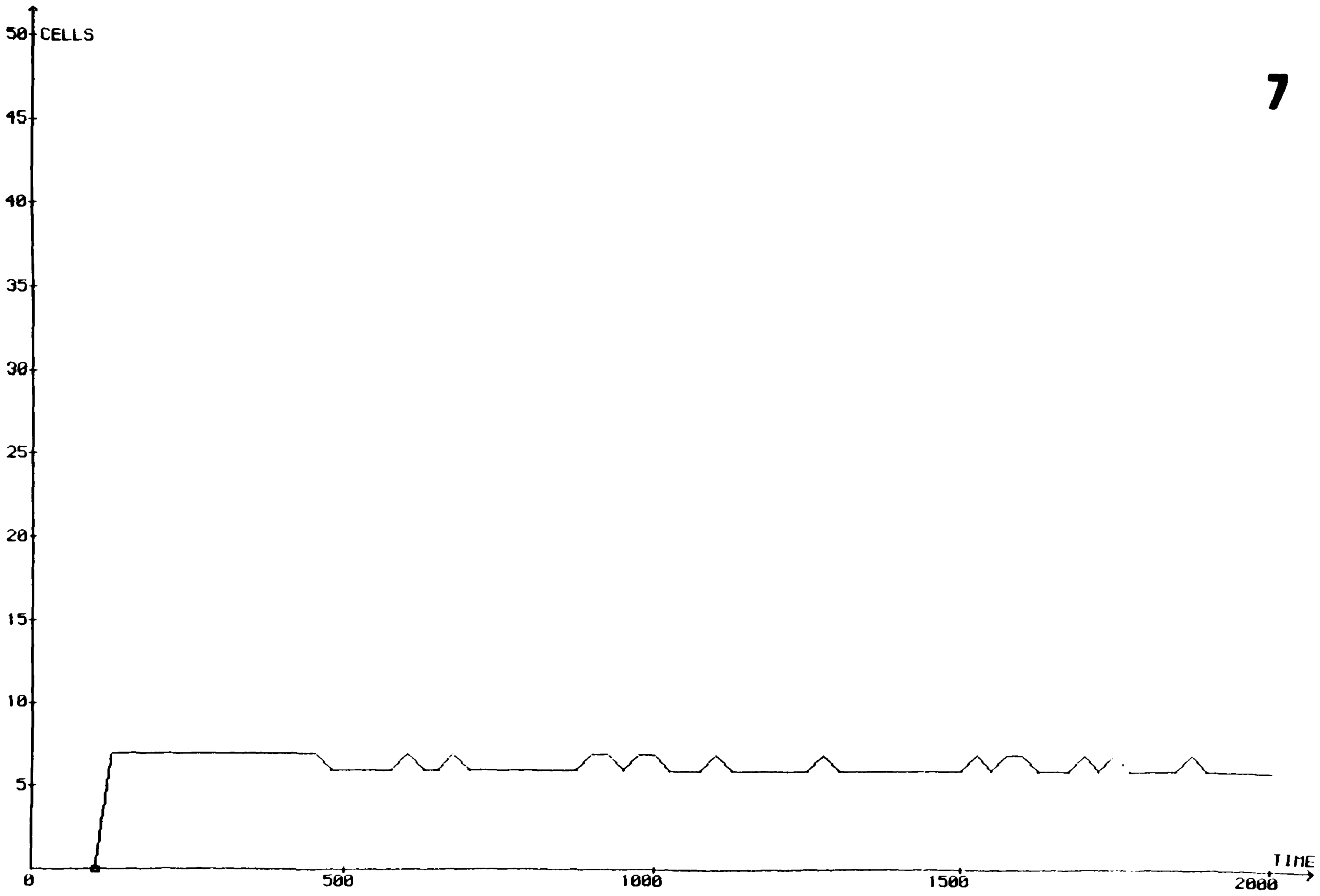
1
300 CELLS
250
200
150
100
50
0
500
1000
1500
2000
TIME

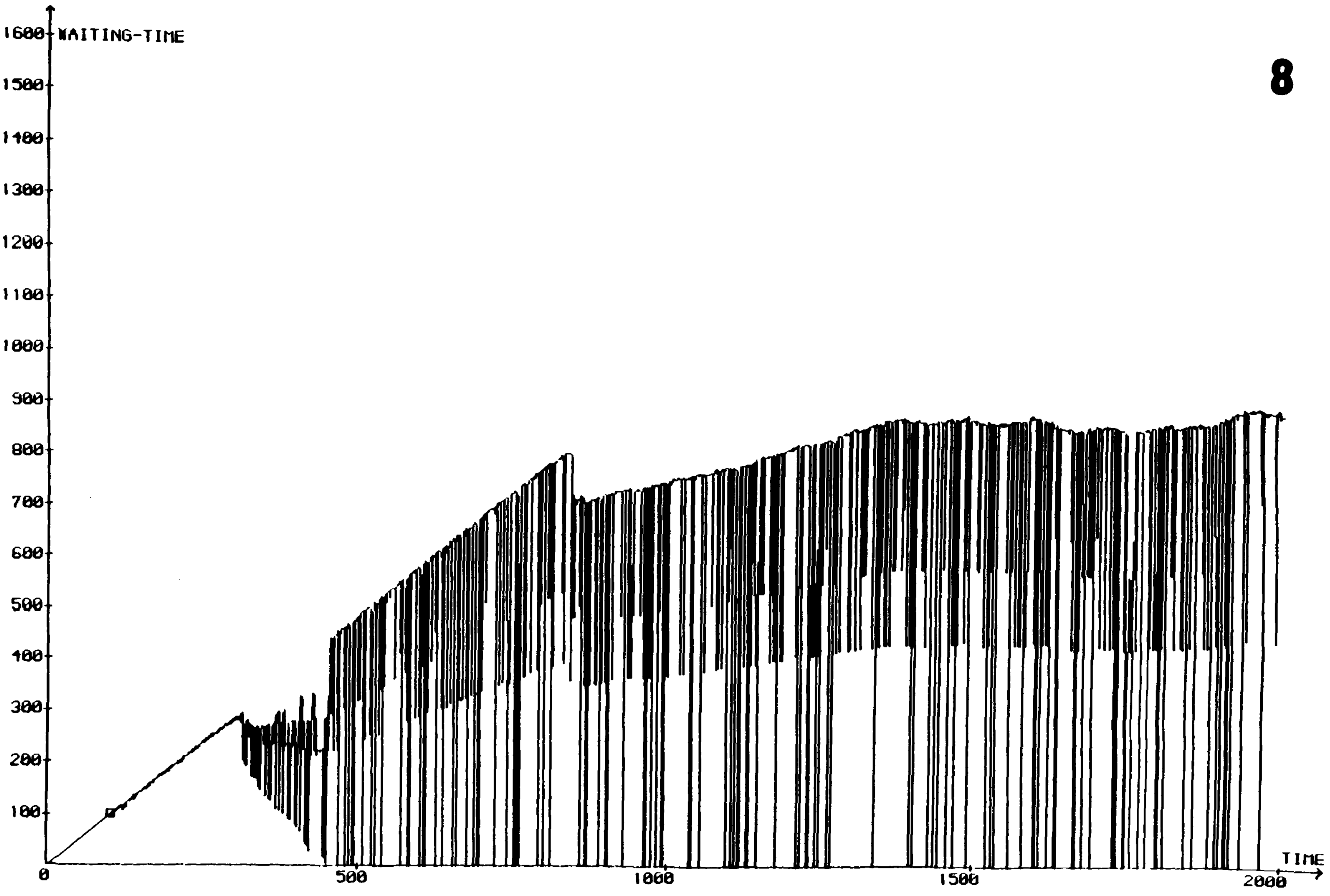

2
CELLS
1000
900
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
TIME

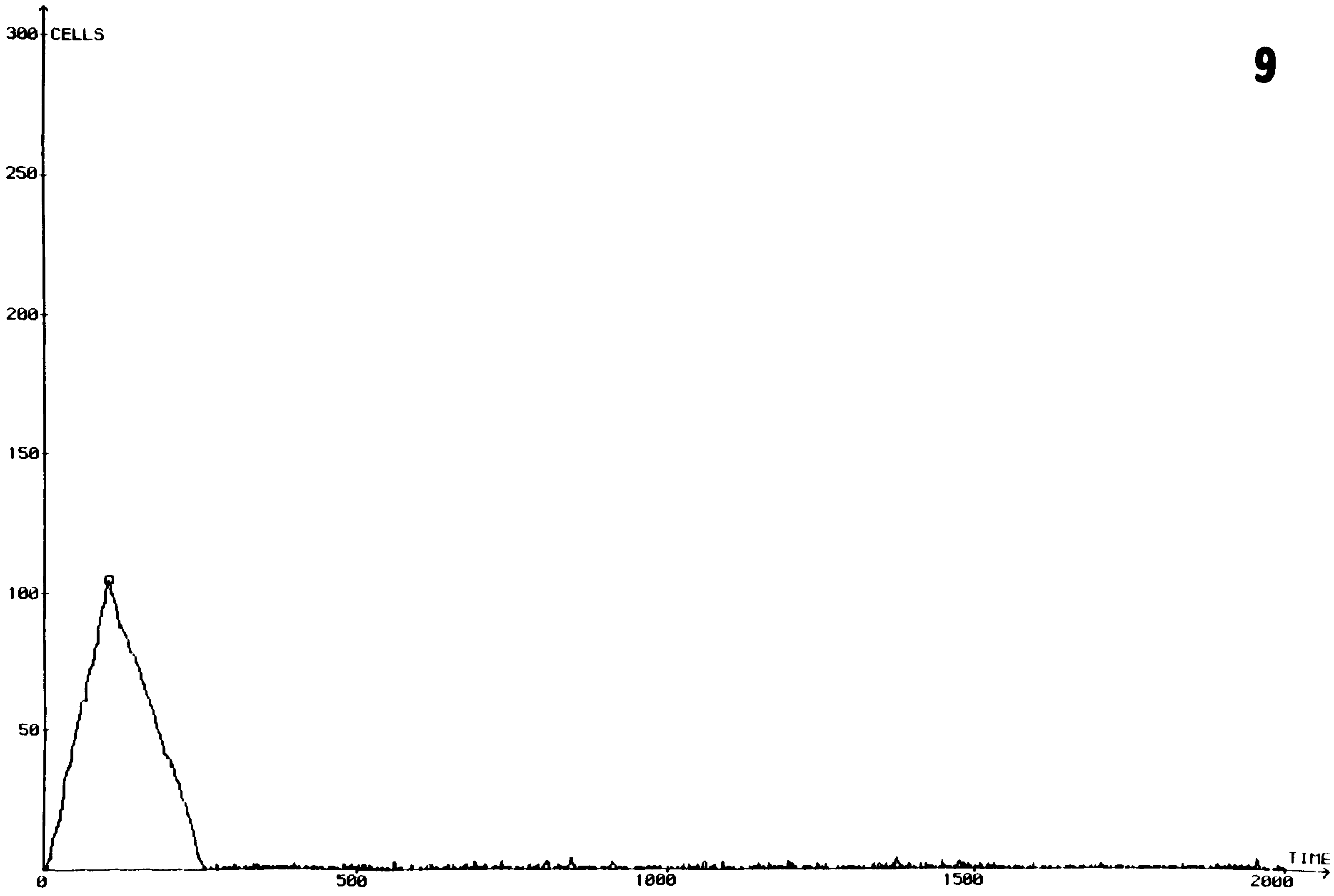

4
CELLS
200
150
100
50
0
500
1000
1500
2000
TIME

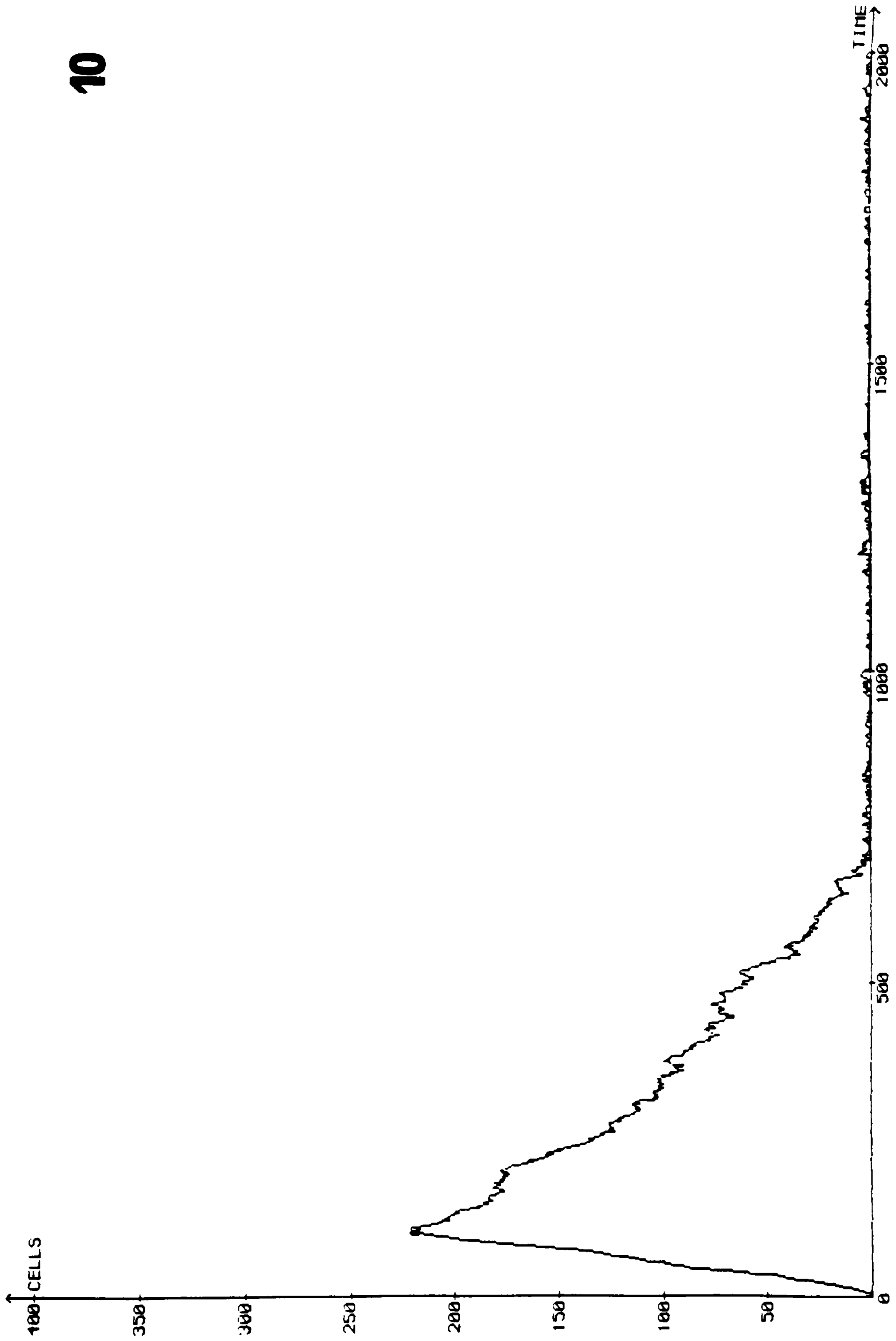

200
CELLS
175
150
125
100
75
50
25
0
500
1000
1500
2000
TIME
5

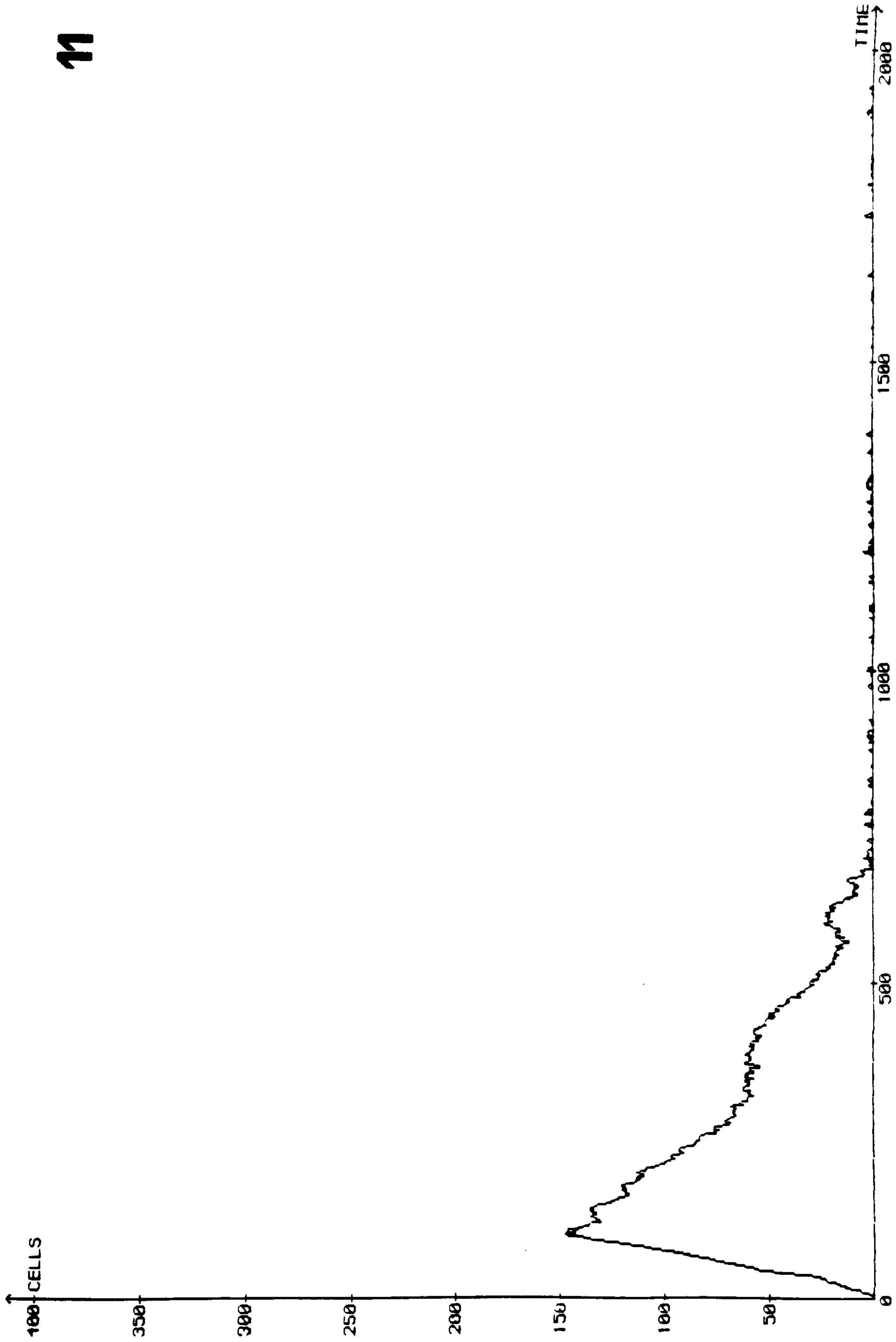

CELLS
1000
900
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
TIME
6

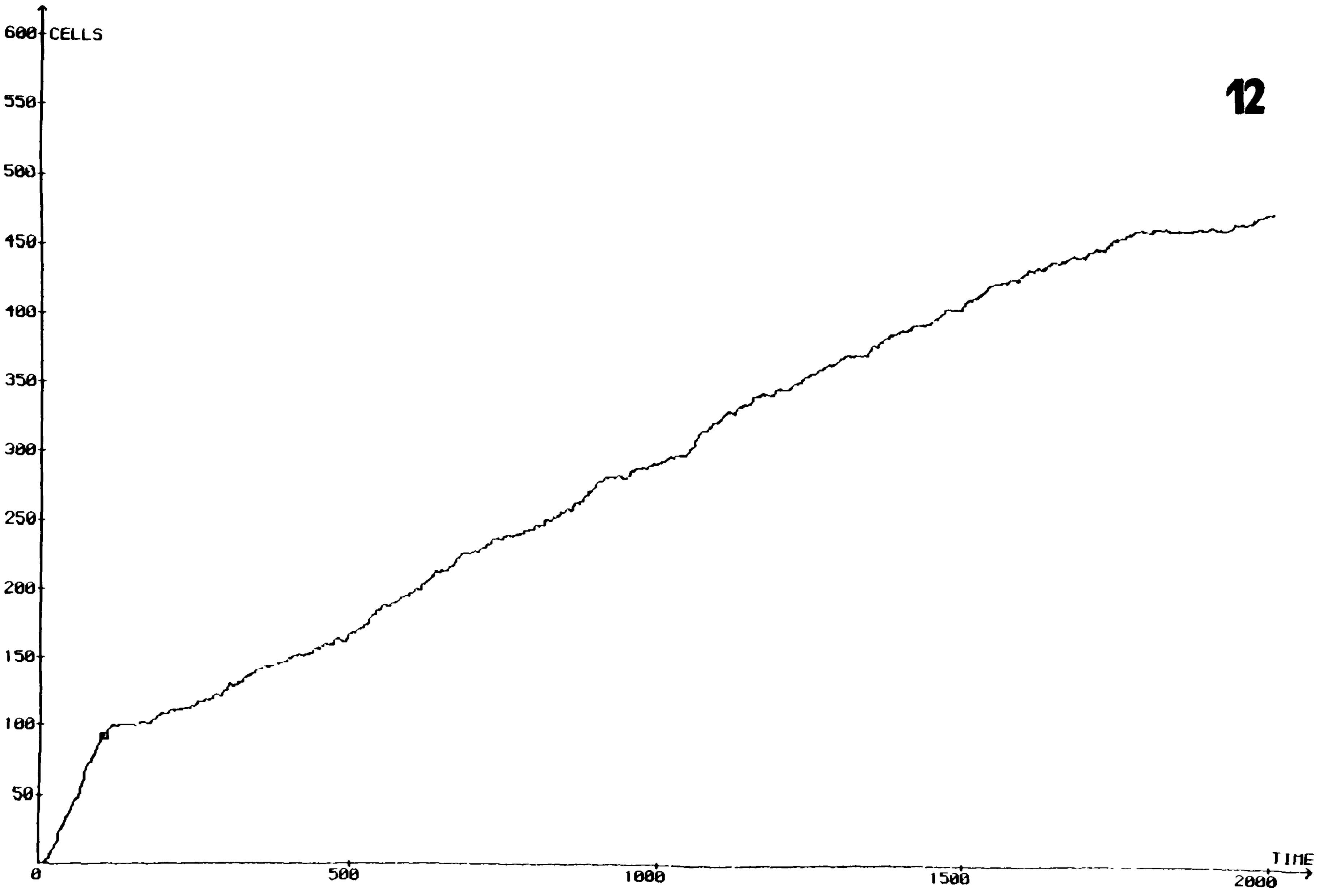

7
CELLS
50
45
40
35
30
25
20
15
10
5
0
500
1000
1500
2000
TIME

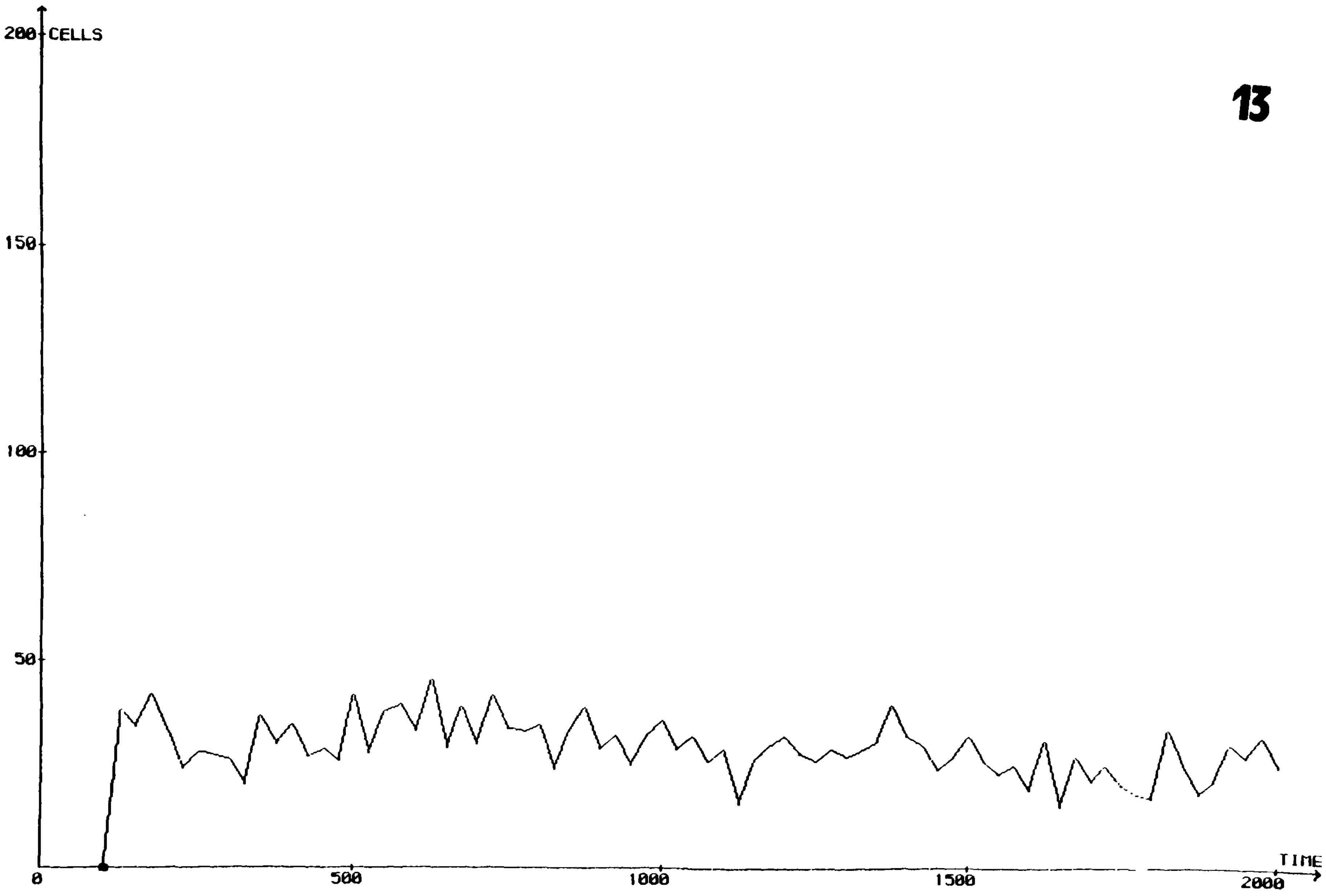

WAITING-TIME
1600
1500
1400
1300
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
TIME
8


9
CELLS
300
250
200
150
100
50
0
500
1000
1500
2000
TIME

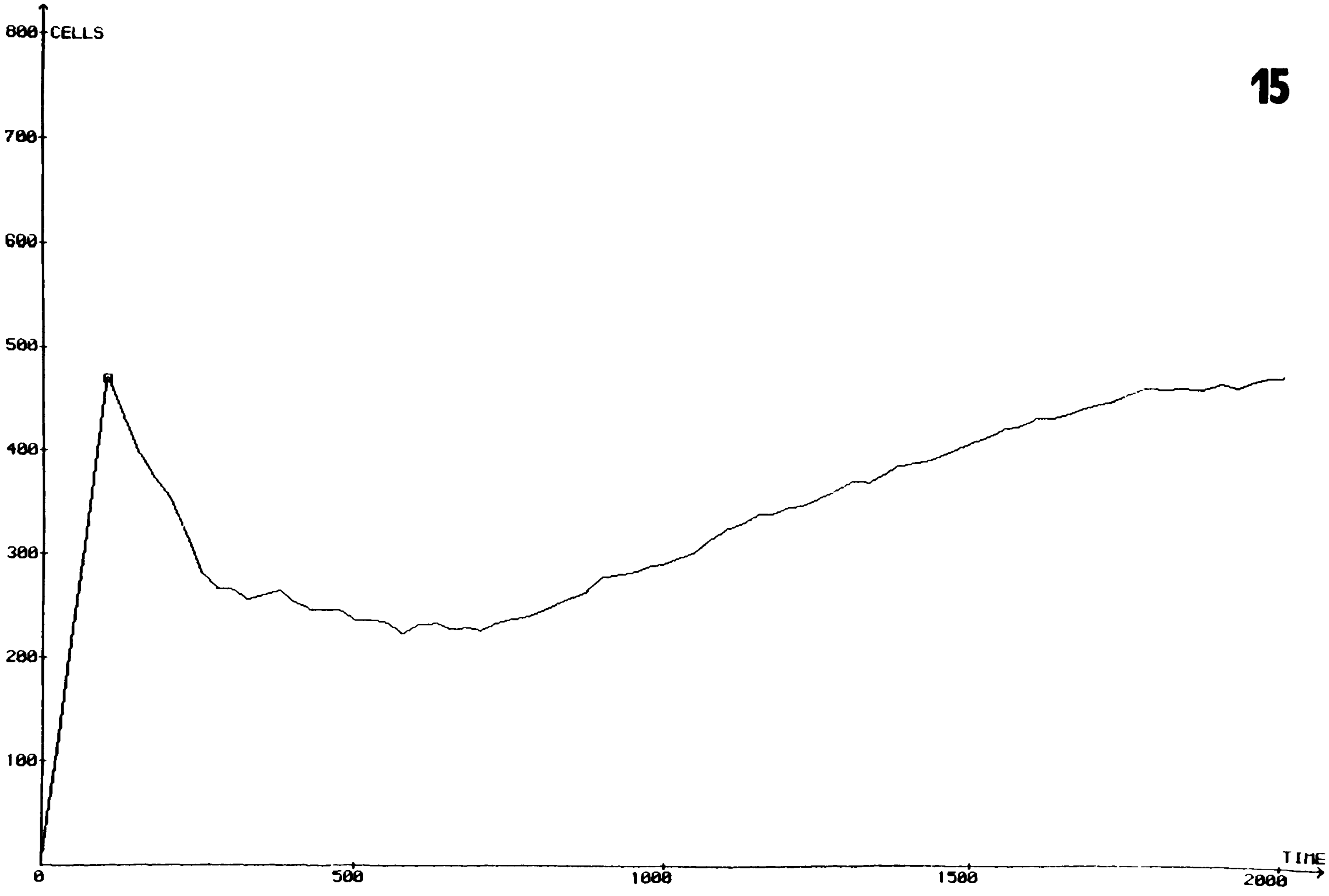

10
CELLS
TIME
400
350
300
250
200
150
100
50
0
500
1000
1500
2000

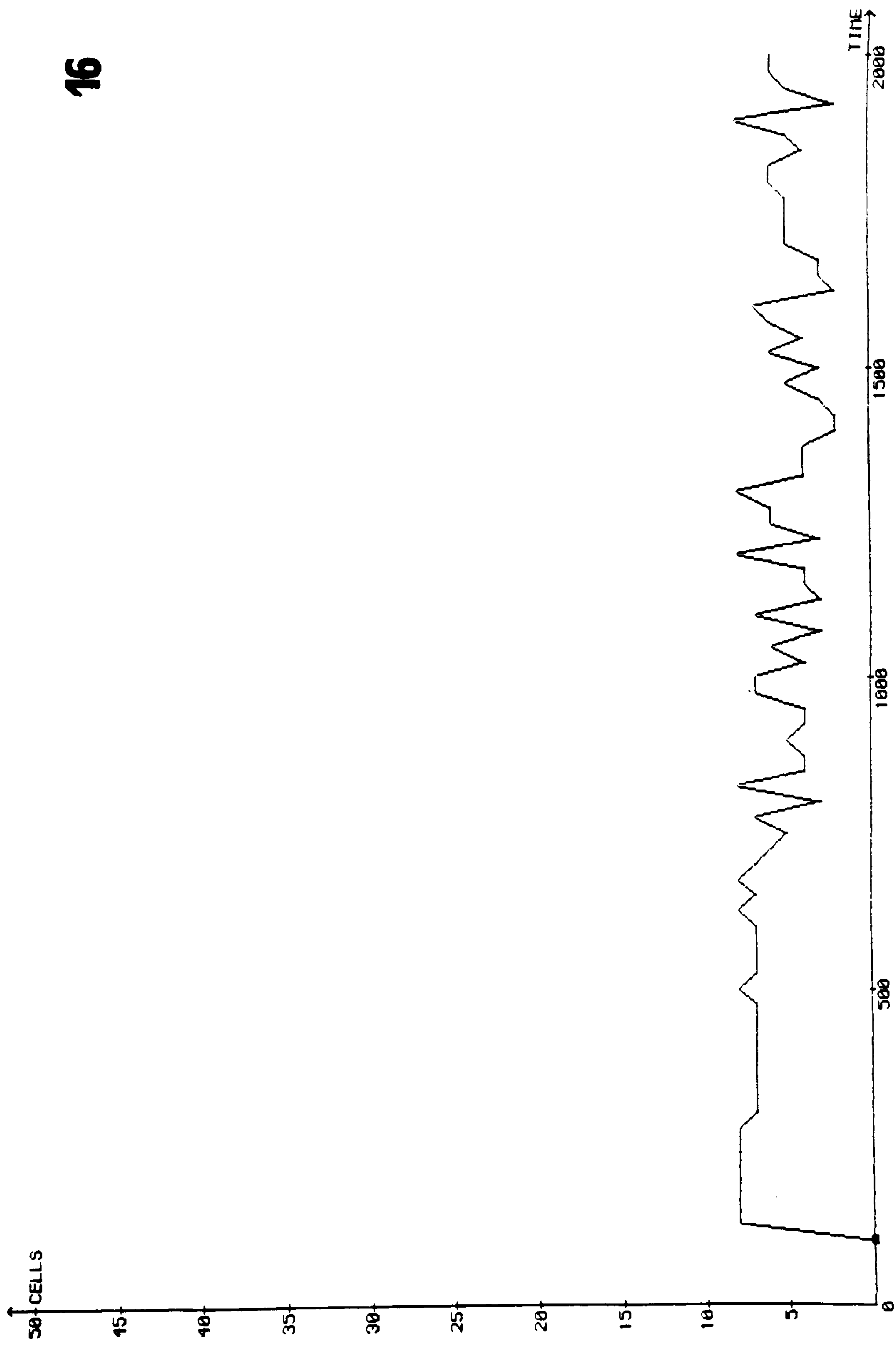

11
CELLS
400
350
300
250
200
150
100
50
0
500
1000
1500
2000
TIME

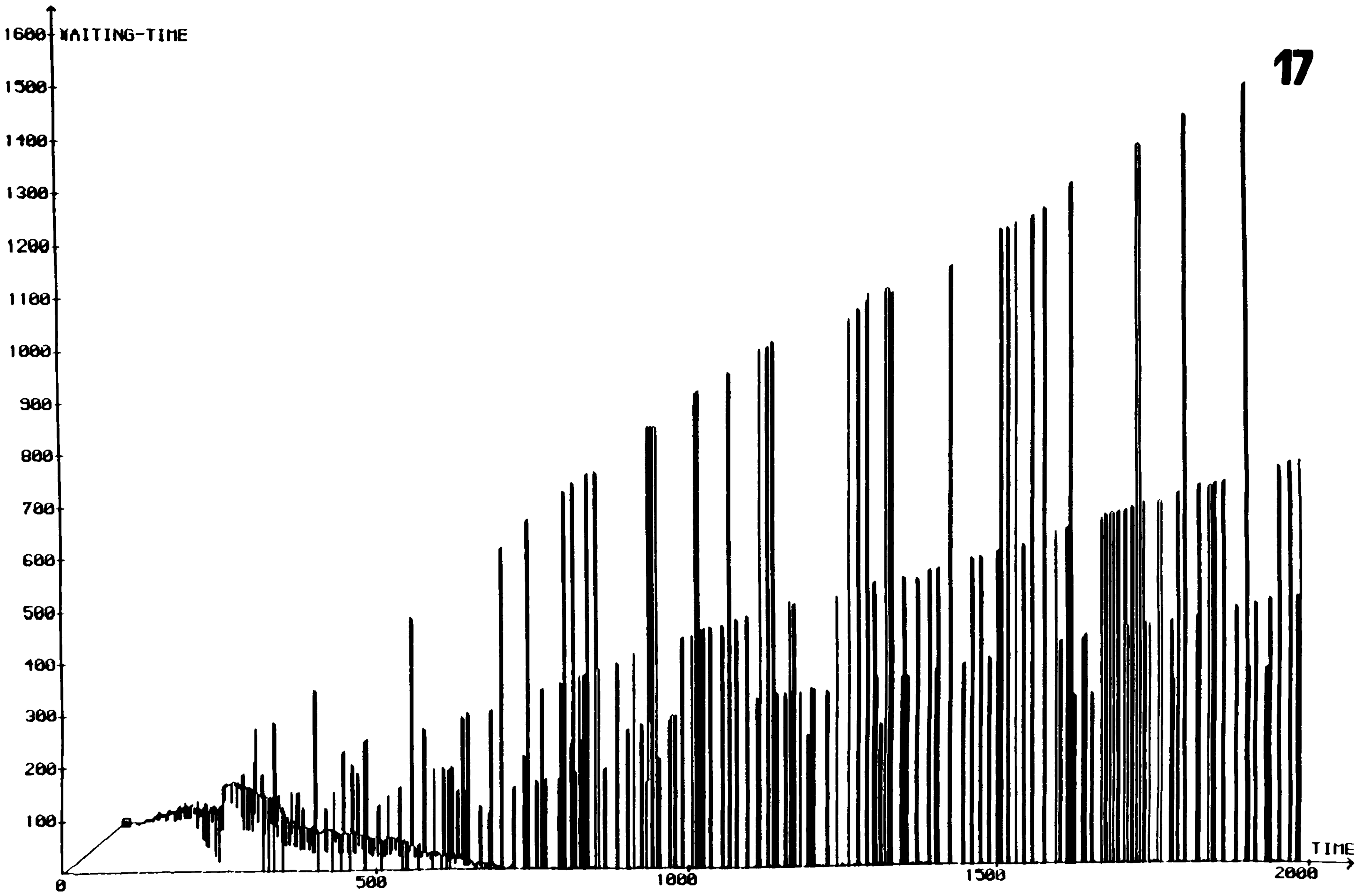

12
CELLS
600
550
500
450
400
350
300
250
200
150
100
50
0
500
1000
1500
2000
TIME

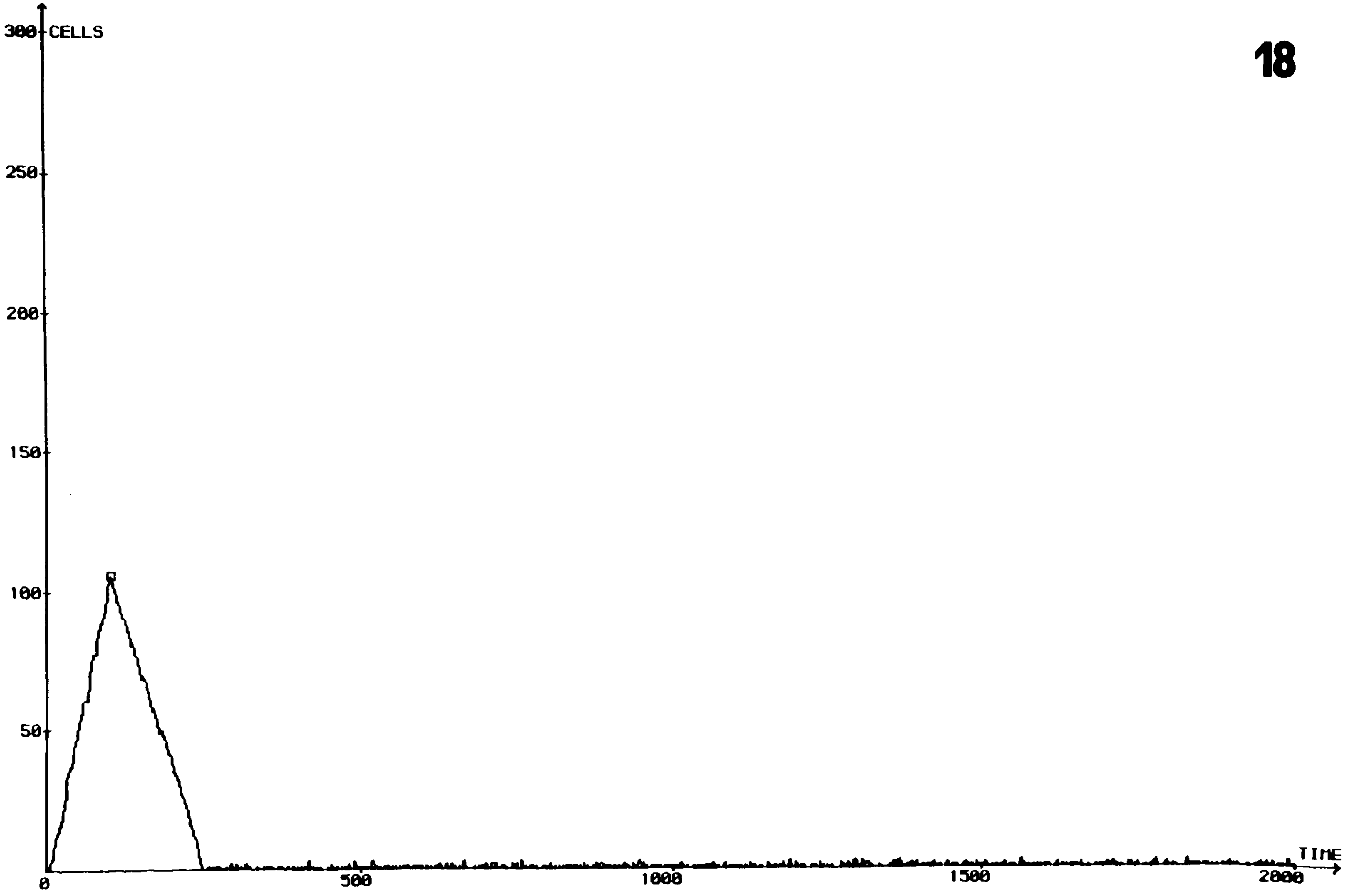

13
CELLS
200
150
100
50
0
500
1000
1500
2000
TIME

CELLS
14
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
TIME


15
CELLS
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
TIME

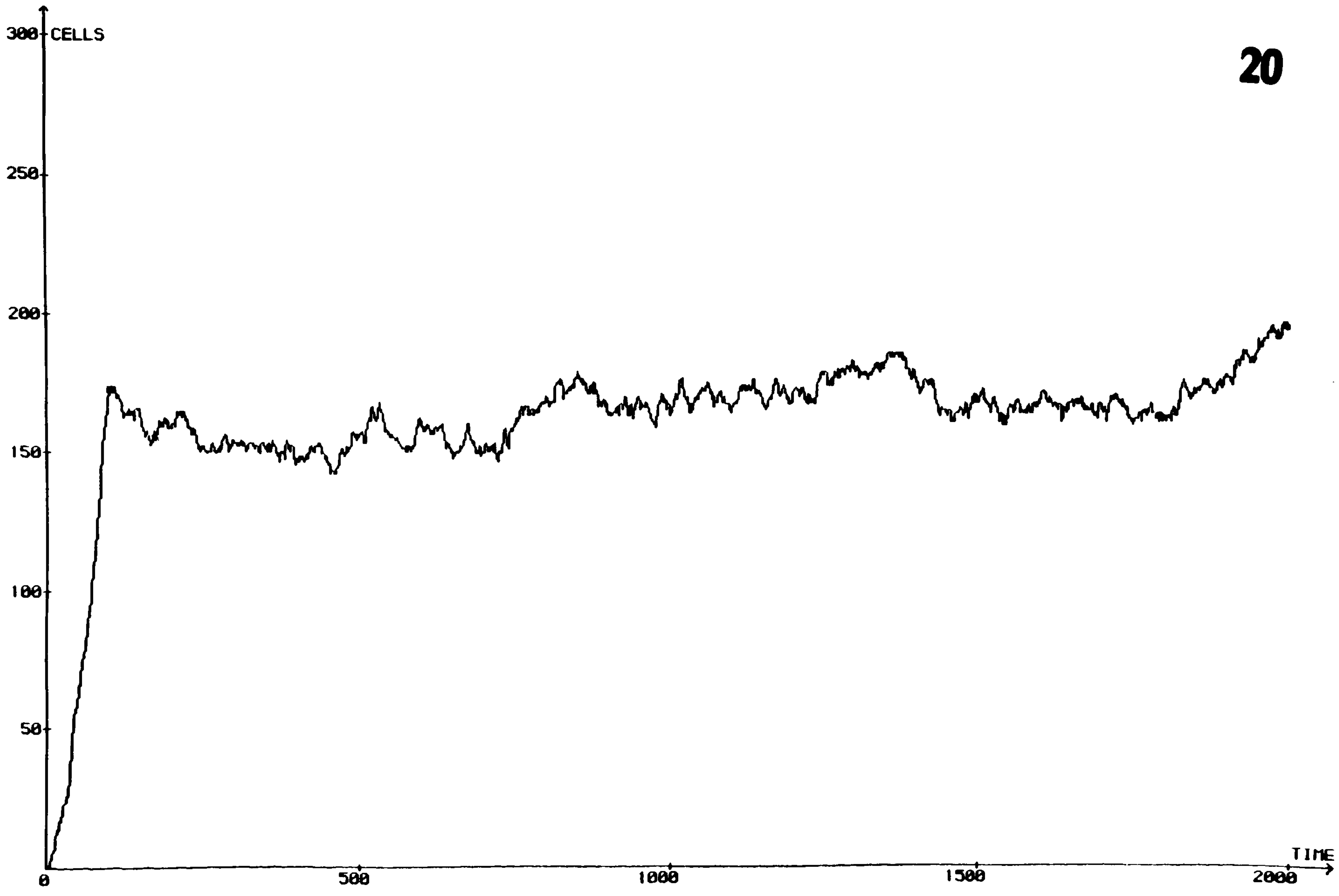

16
CELLS
TIME
50
45
40
35
30
25
20
15
10
5
0
500
1000
1500
2000

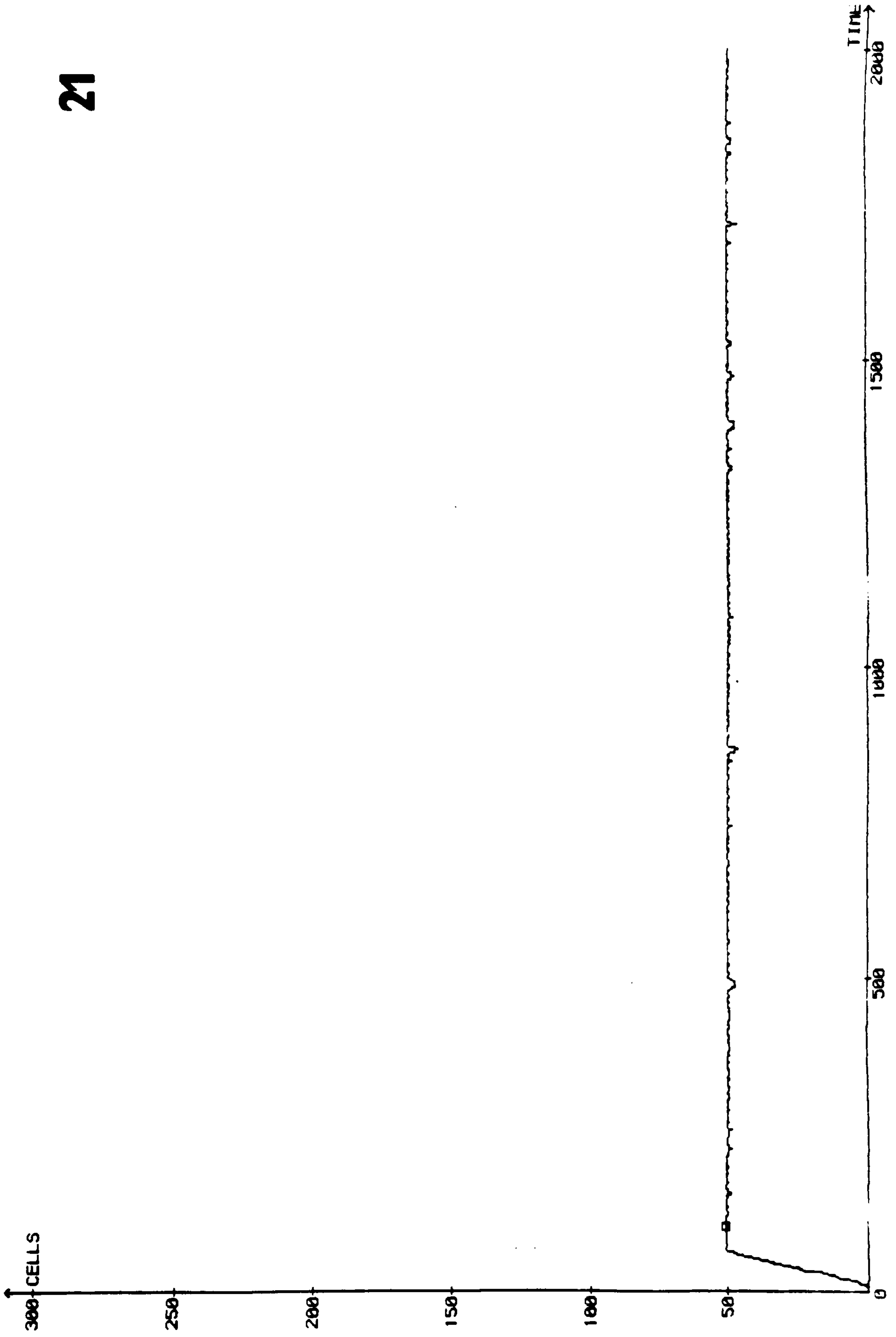

17
WAITING-TIME
TIME
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
500
1000
1500
2000

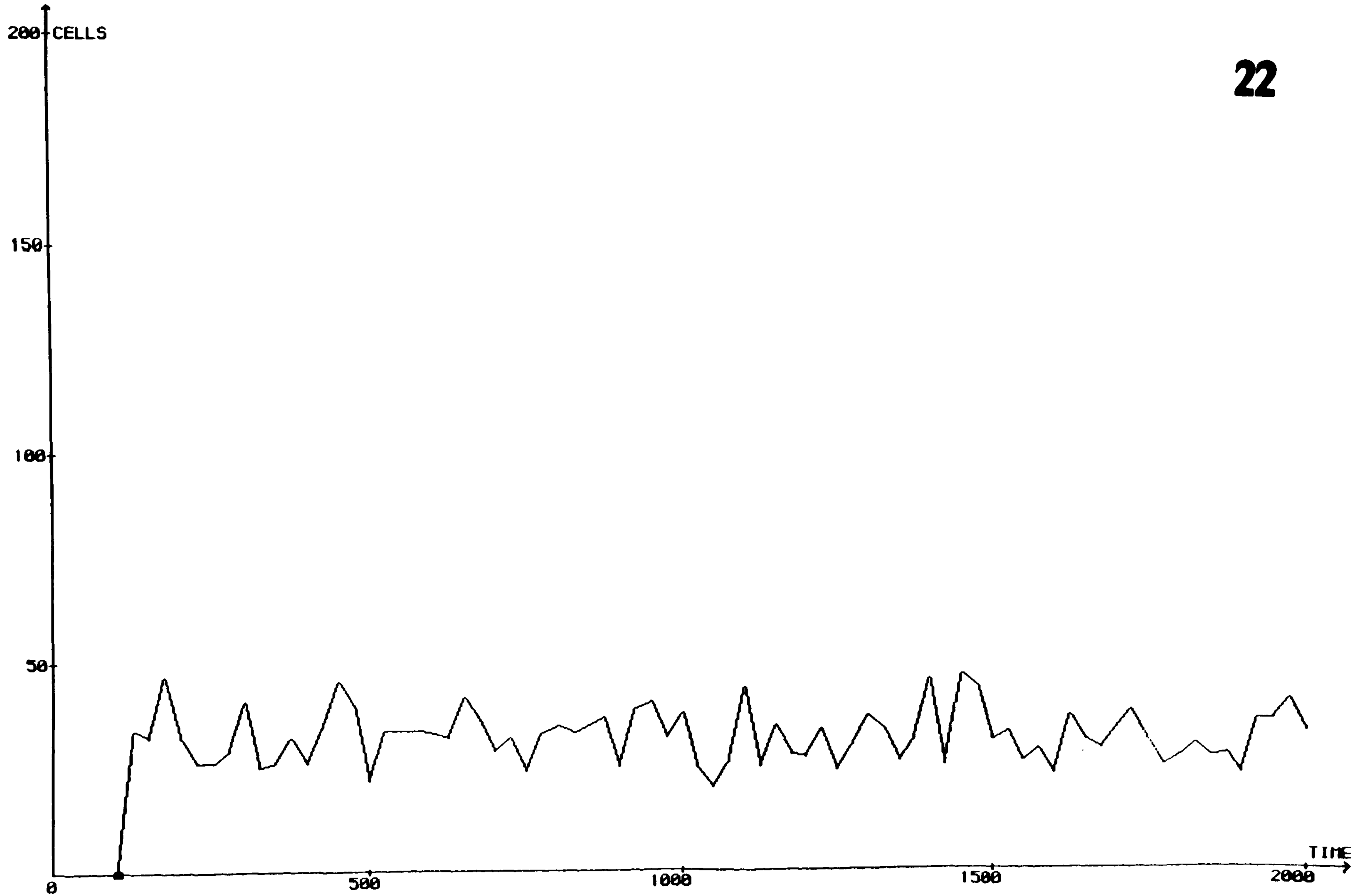


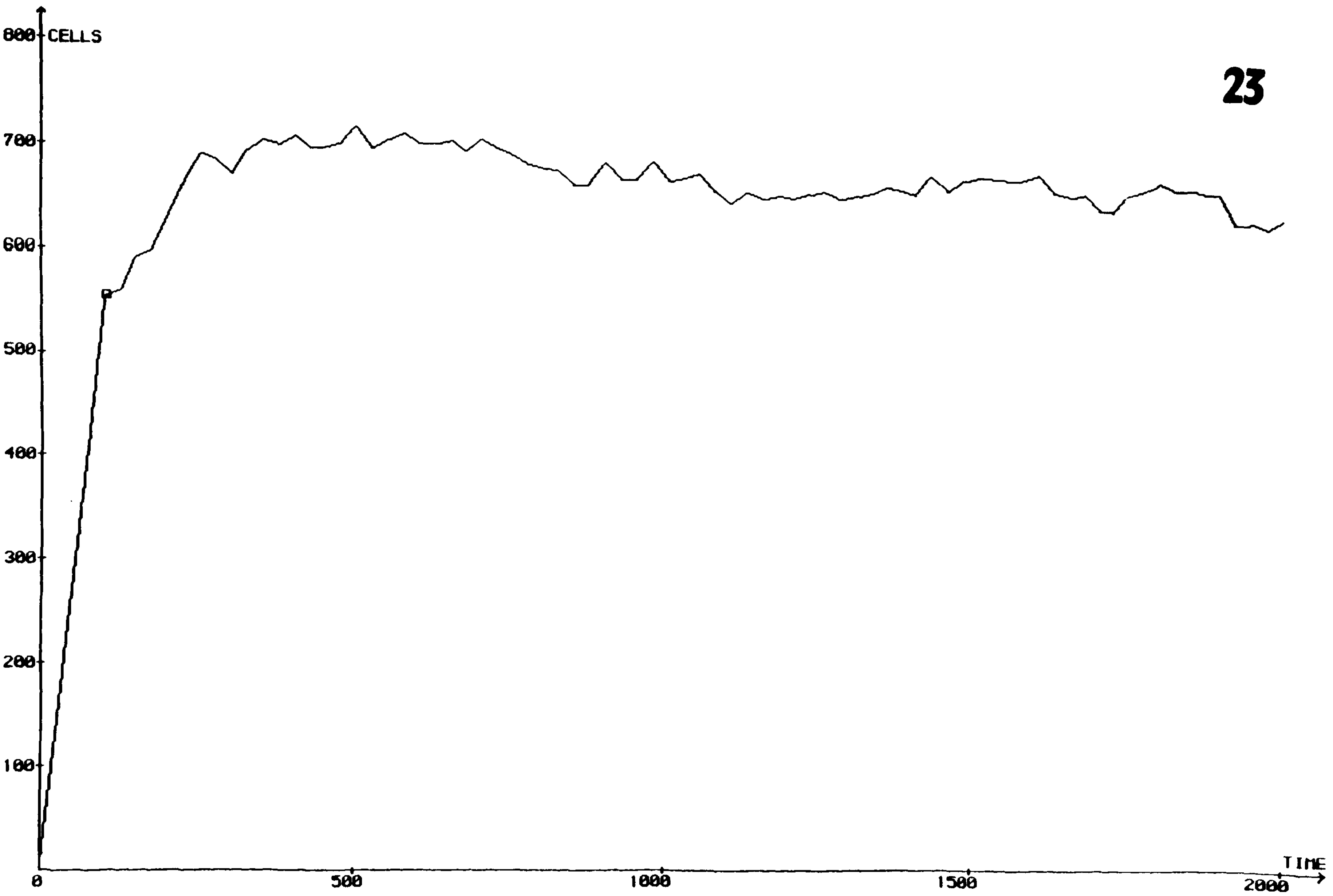
18
300 CELLS
250
200
150
100
50
0
500
1000
1500
2000
TIME

300
CELLS
19
250
200
150
100
50
0
500
1000
1500
2000
TIME

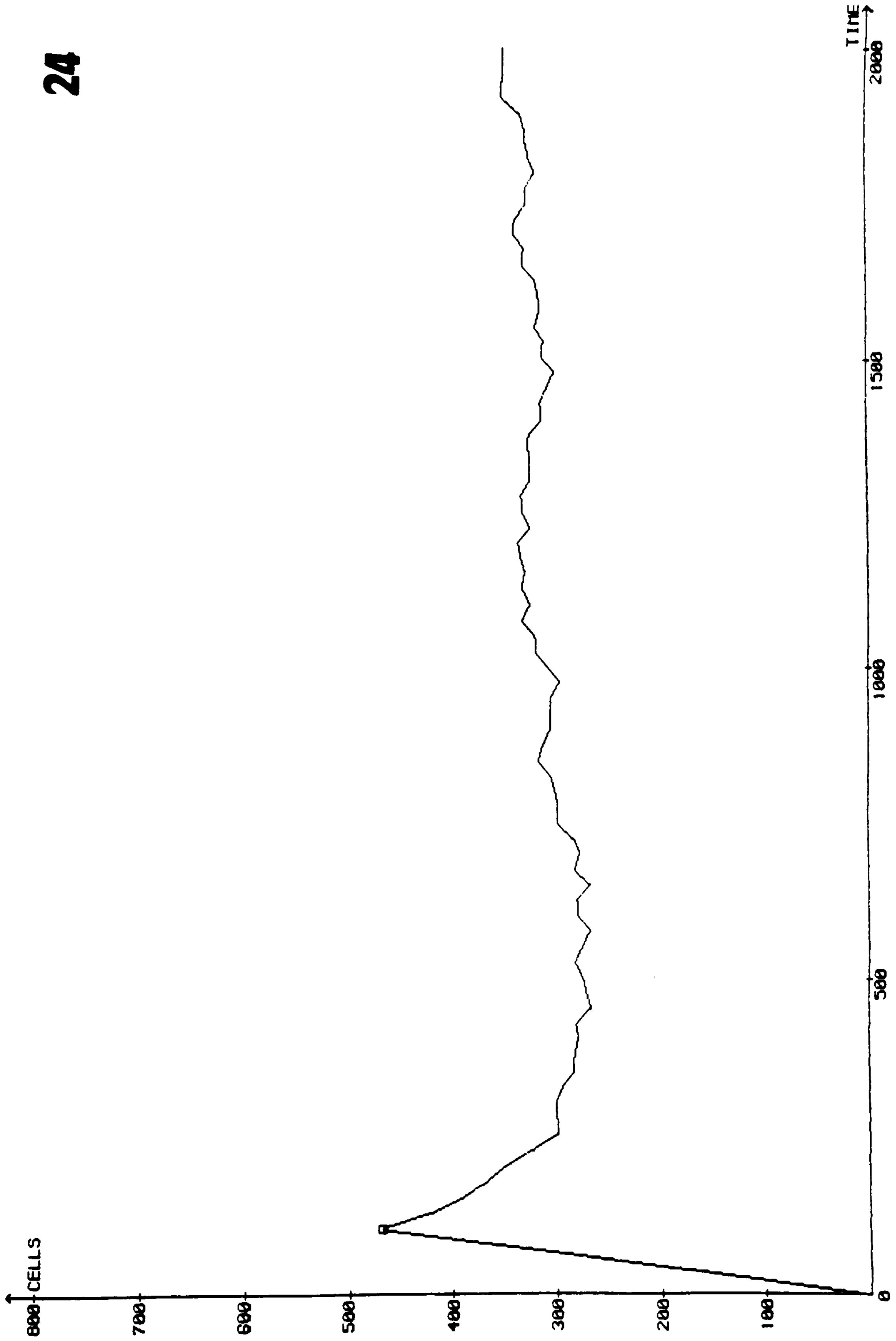

20
CELLS
TIME
300
250
200
150
100
50
0
500
1000
1500
2000


21
CELLS
300
250
200
150
100
50
0
500
1000
1500
2000
TIME

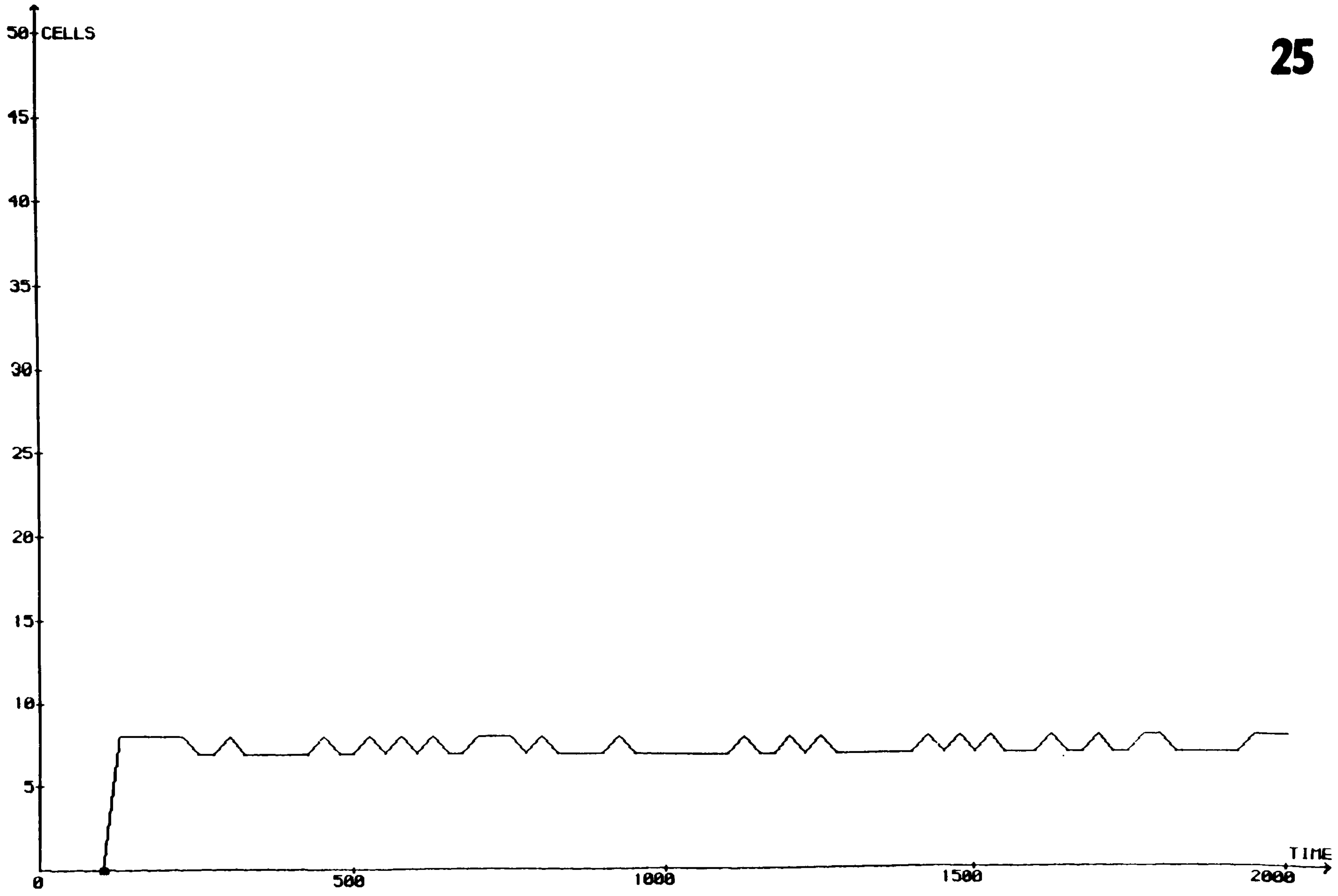

200
CELLS
22
150
100
50
0
500
1000
1500
2000
TIME

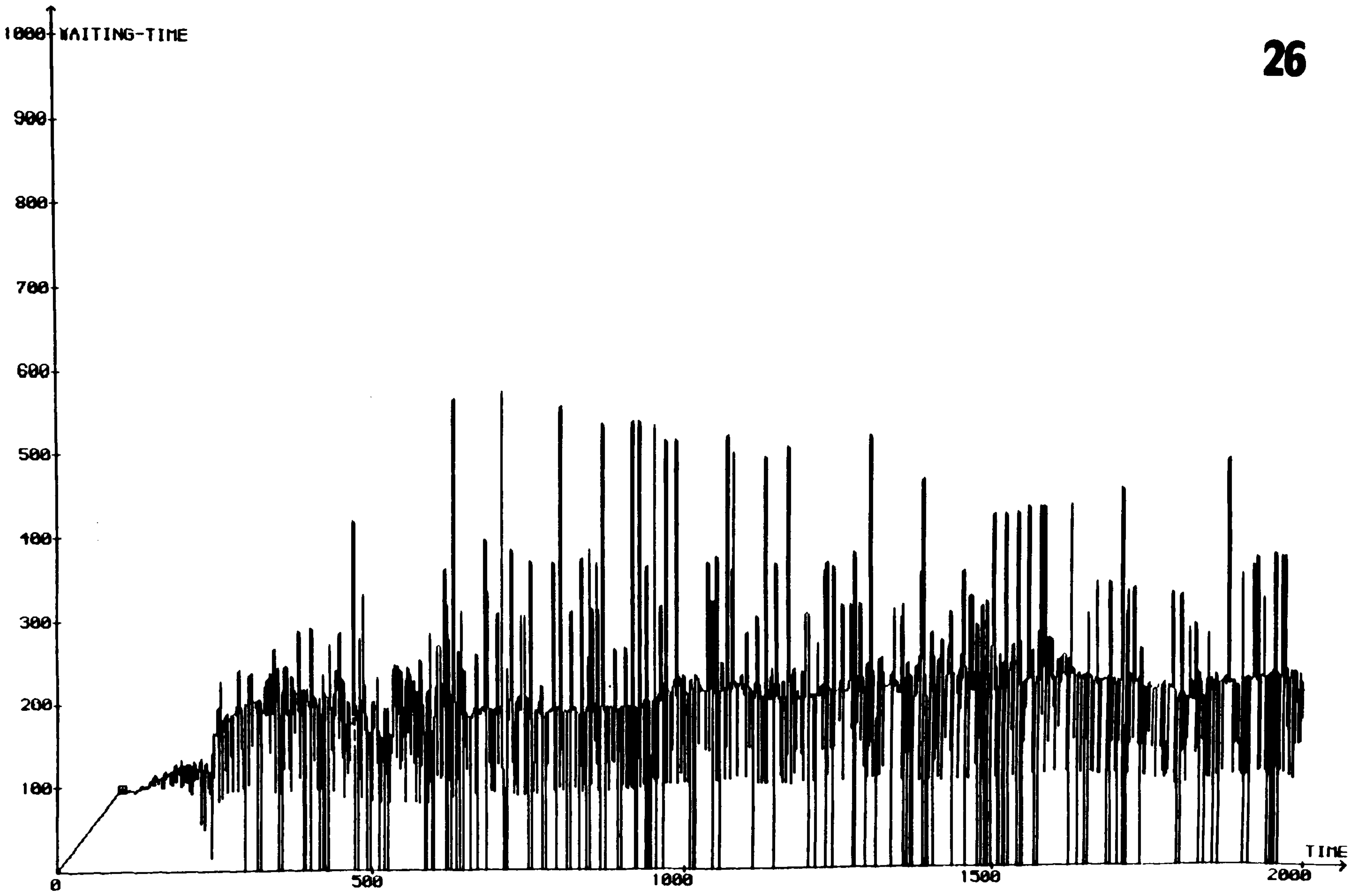


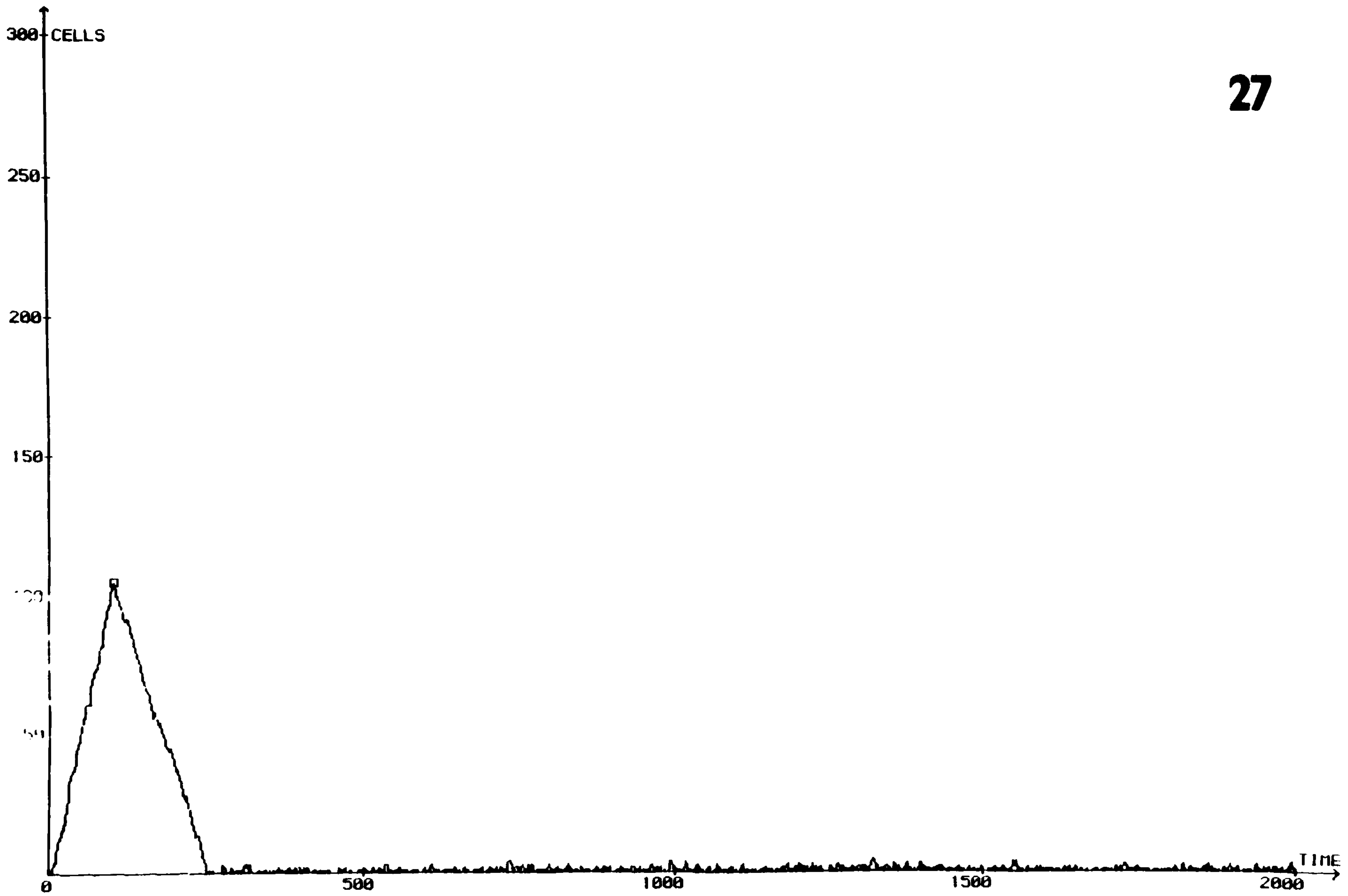
23
CELLS
TIME
800
700
600
500
400
300
200
100
0
500
1000
1500
2000

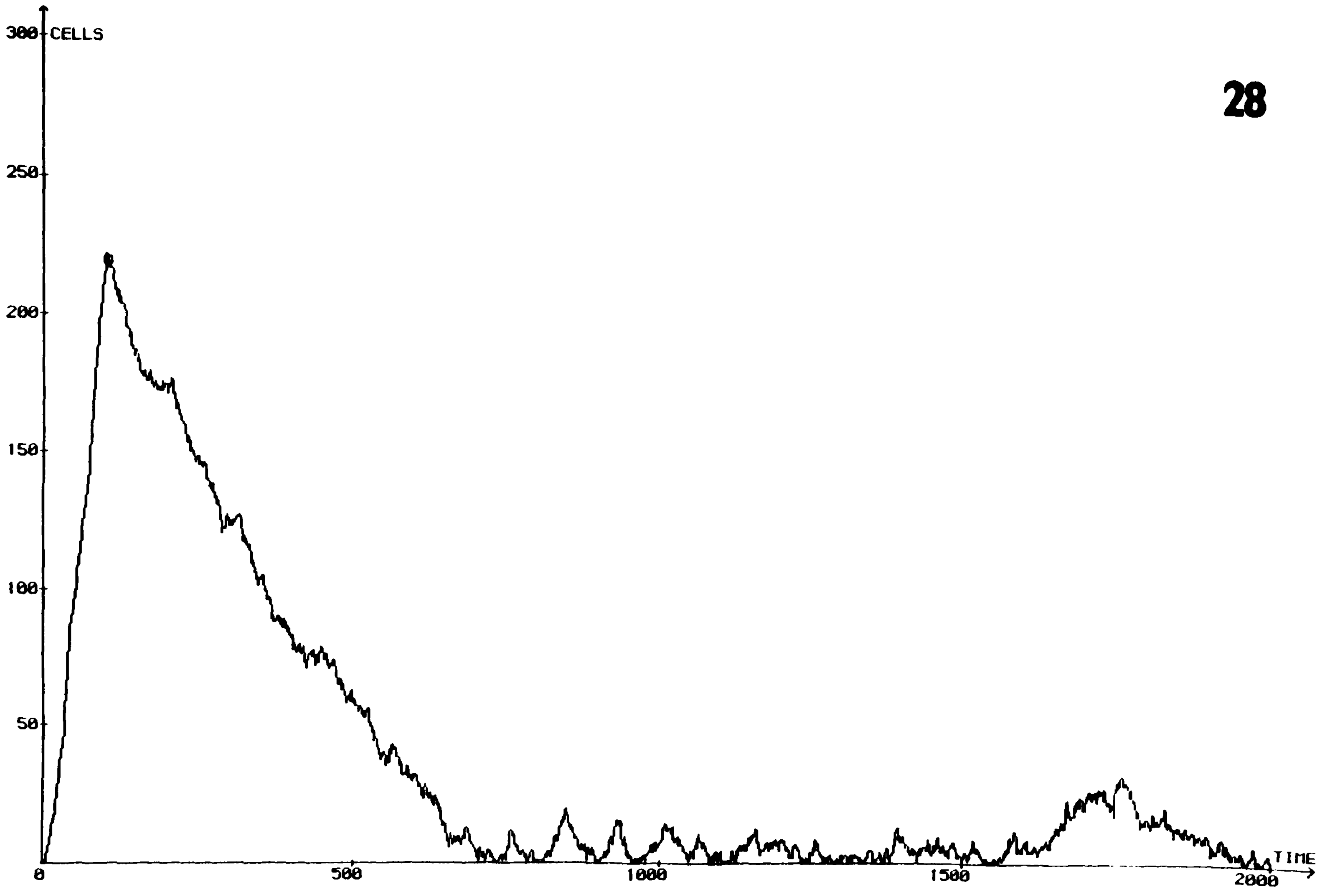


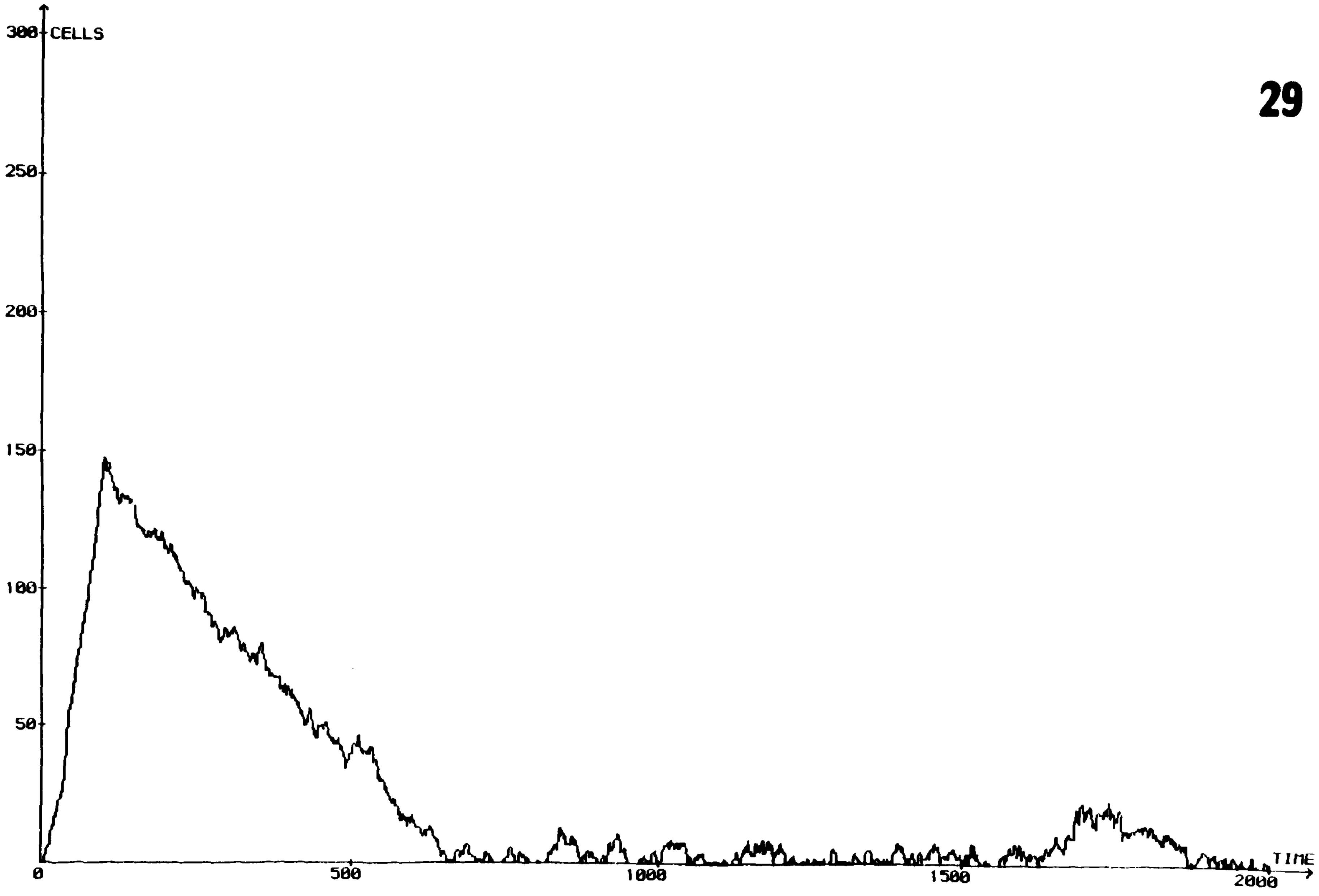
24
CELLS
TIME
800
700
600
500
400
300
200
100
0
500
1000
1500
2000

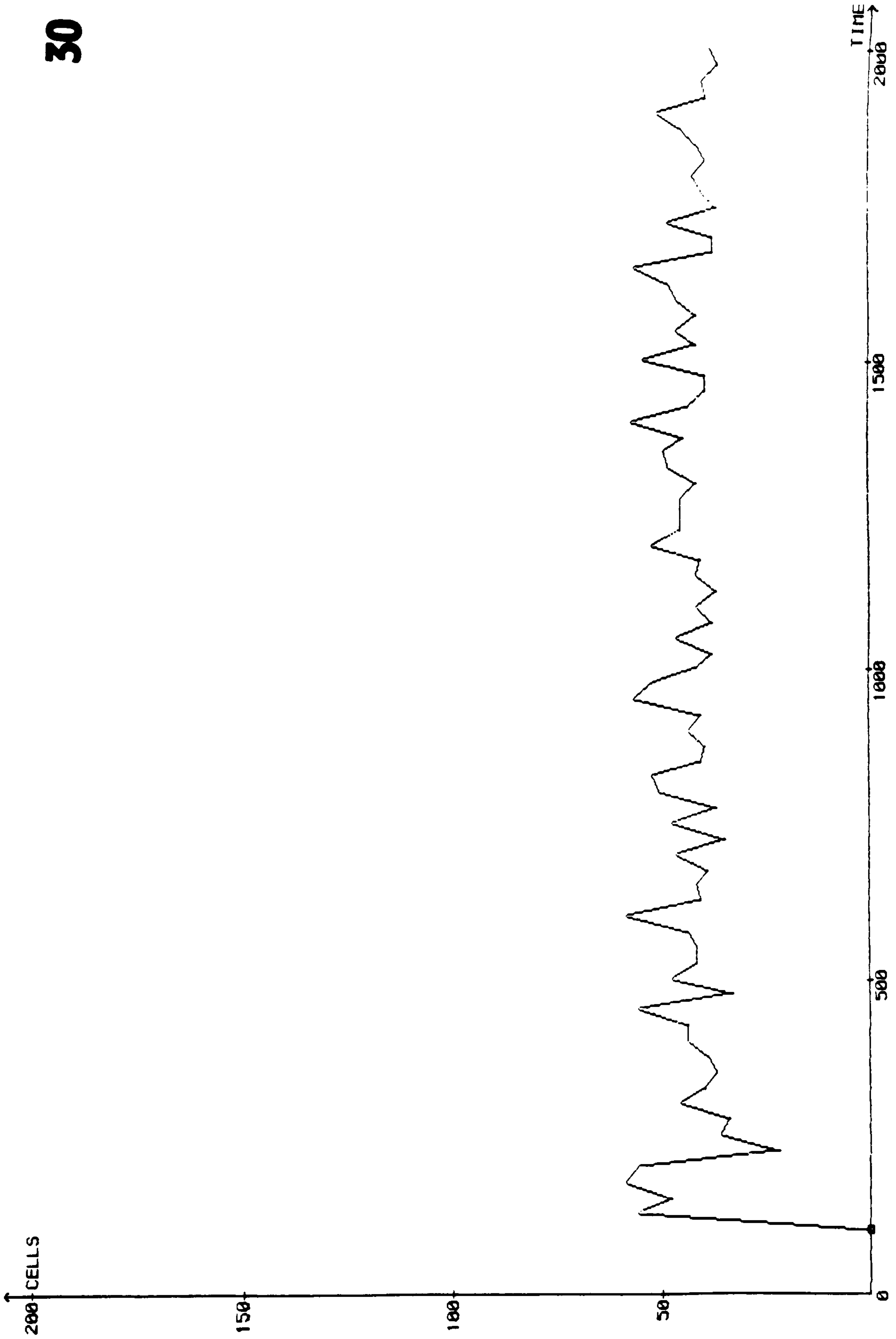

25
CELLS
50
45
40
35
30
25
20
15
10
5
0
500
1000
1500
2000
TIME

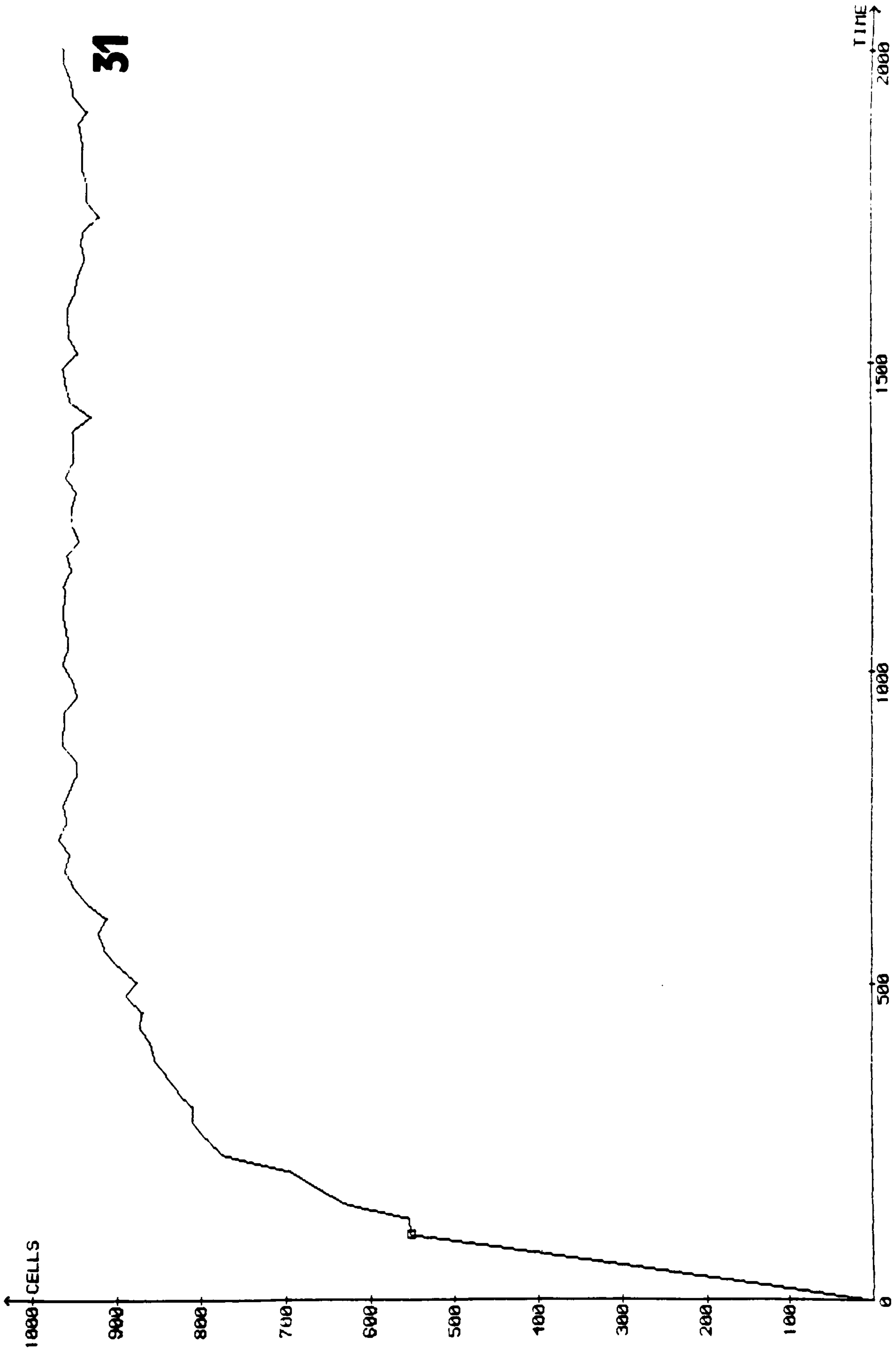

26
WAITING-TIME
TIME
1000
900
800
700
600
500
400
300
200
100
0
500
1000
1500
2000


27
CELLS
300
250
200
150
0
500
1000
1500
2000
TIME

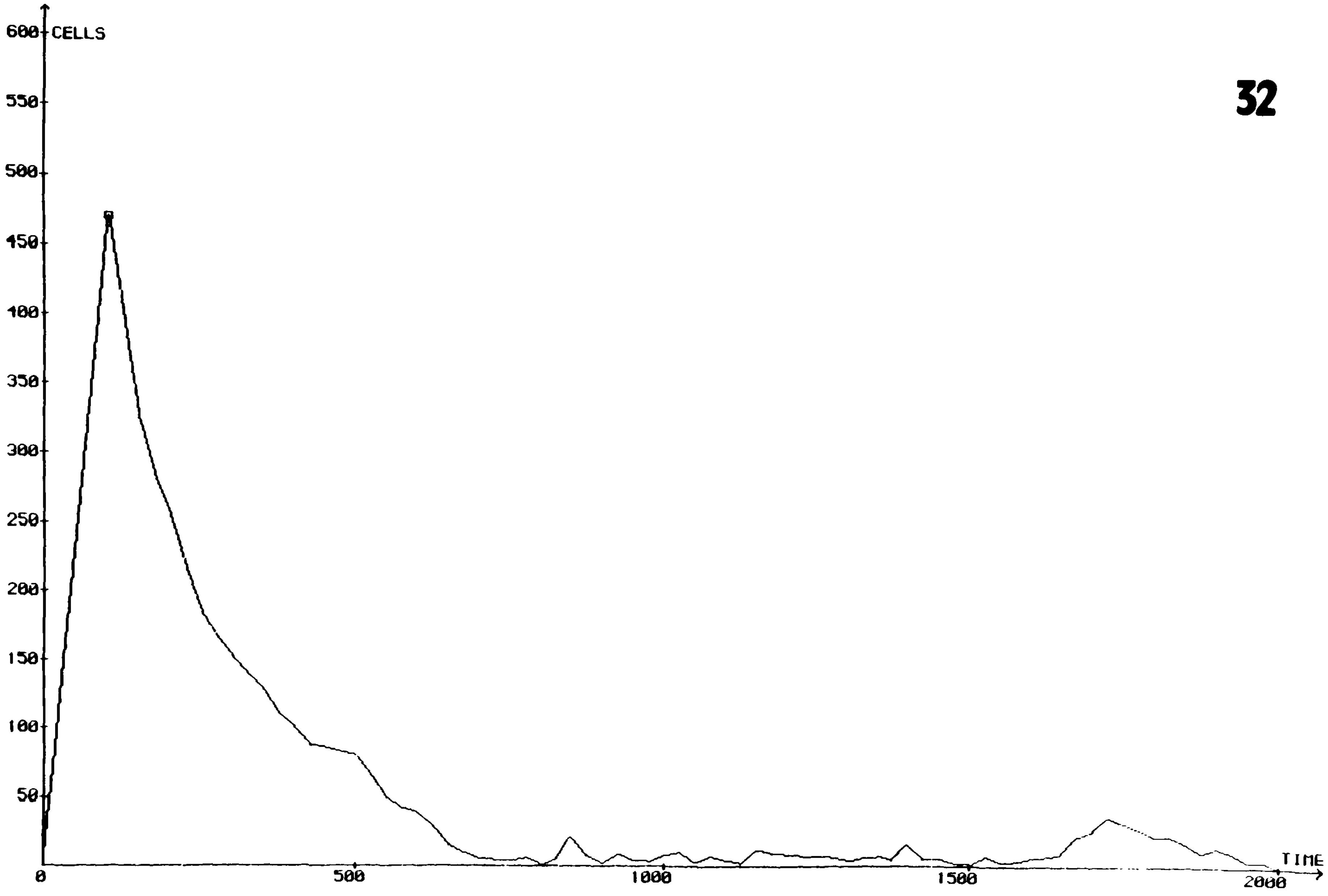

28
CELLS
300
250
200
150
100
50
0
500
1000
1500
2000
TIME

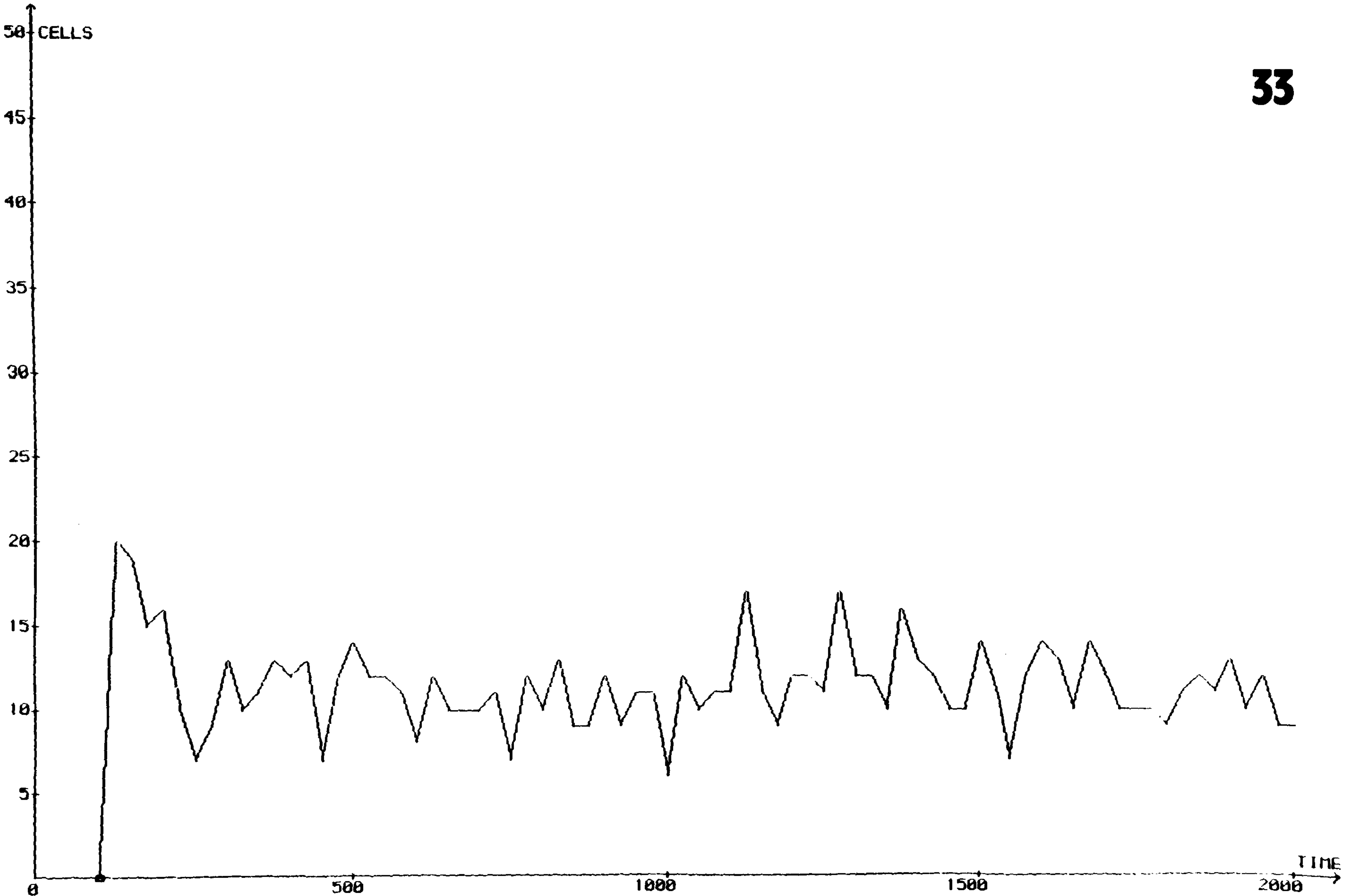

29
CELLS
300
250
200
150
100
50
0
500
1000
1500
2000
TIME

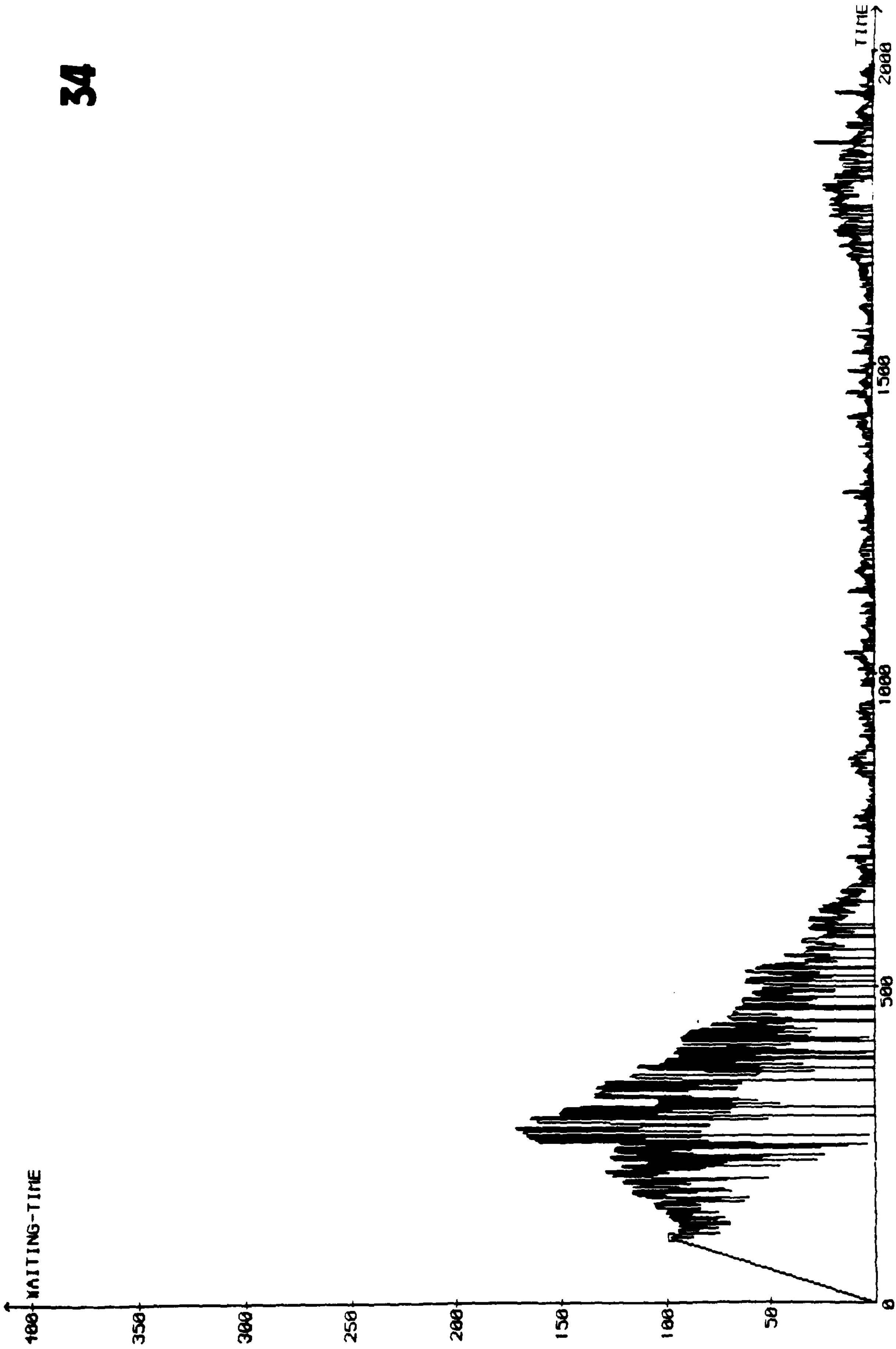

30
CELLS
TIME
200
150
100
50
0
500
1000
1500
2000

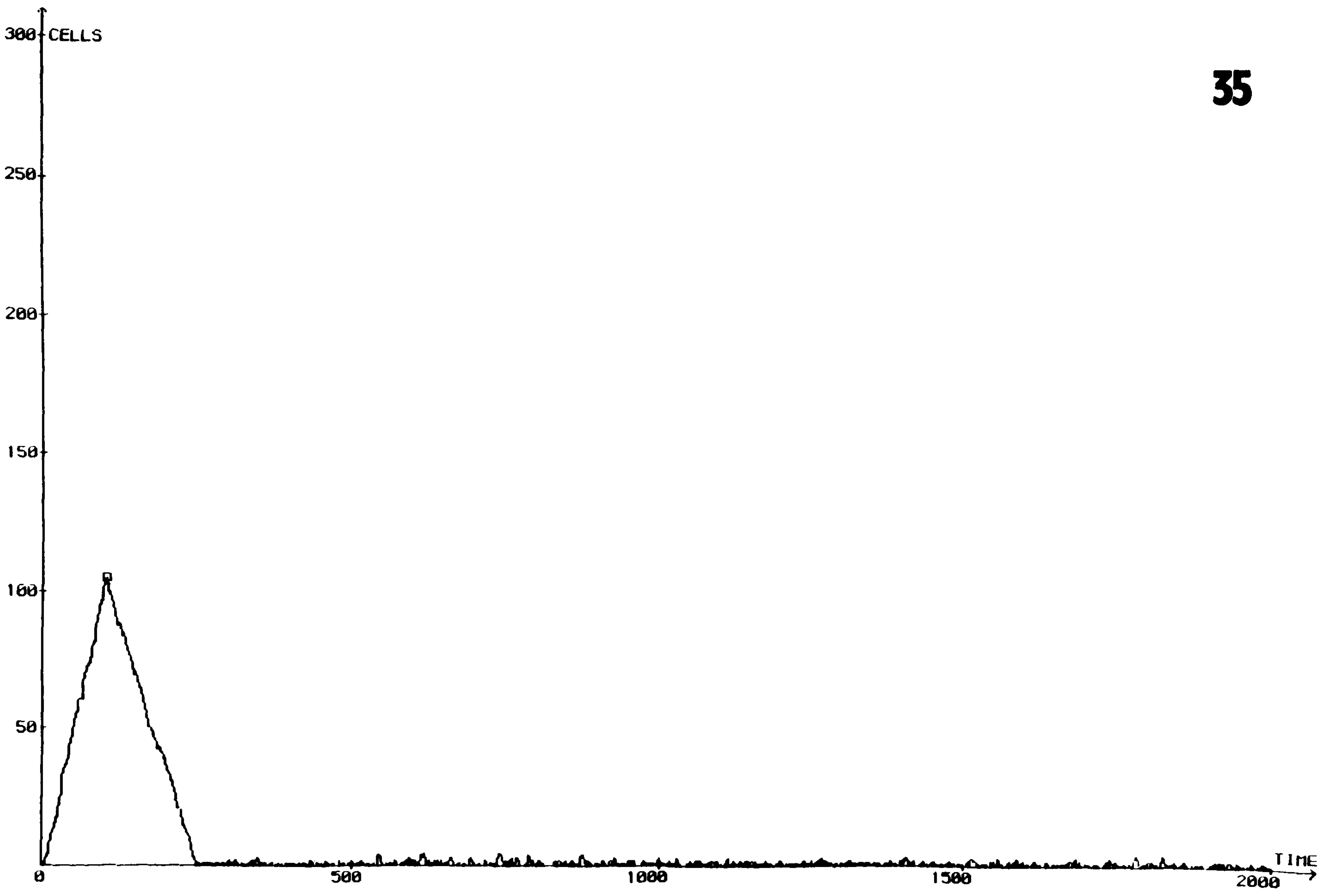


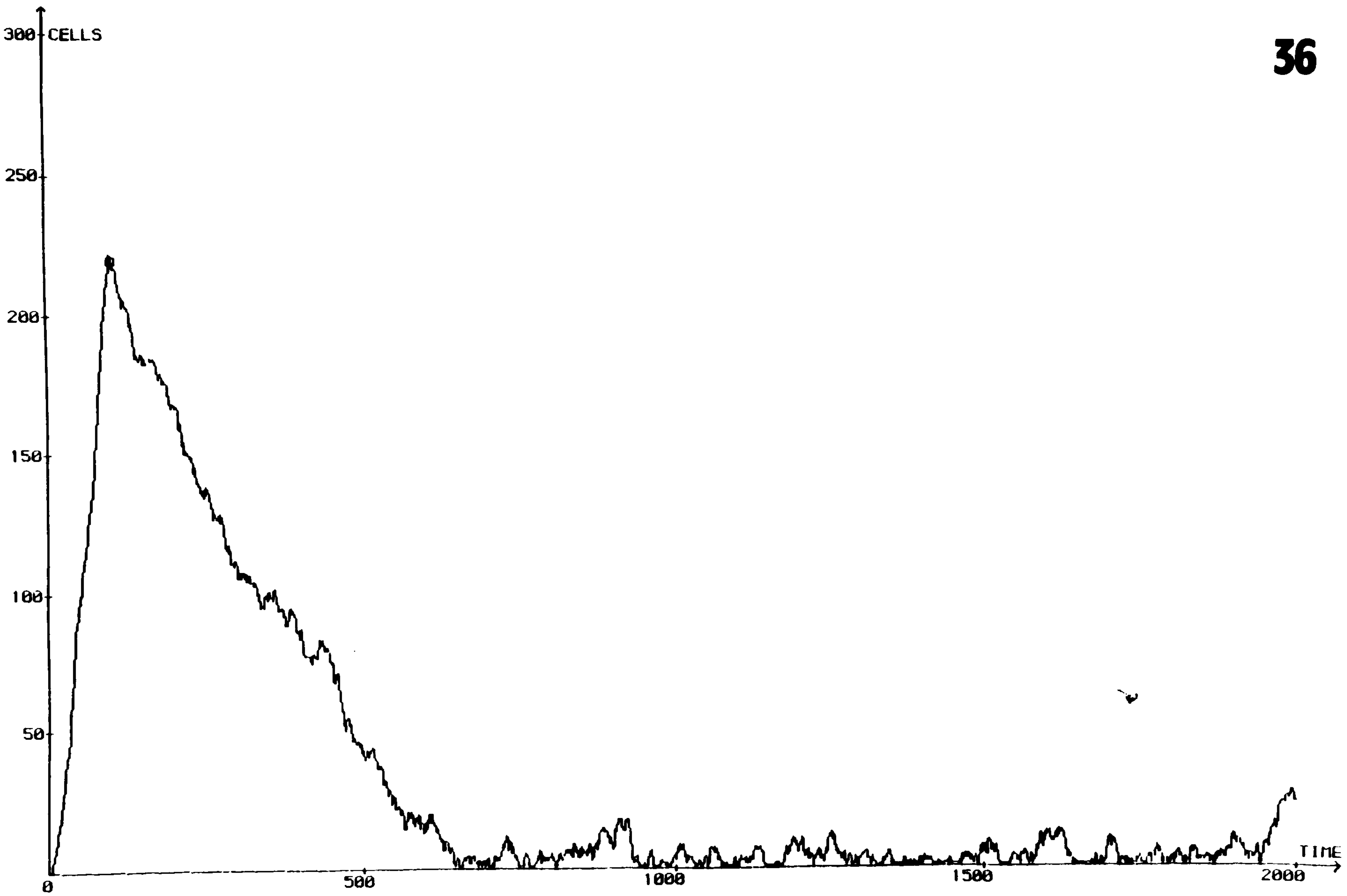
31
CELLS
1000
900
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
TIME

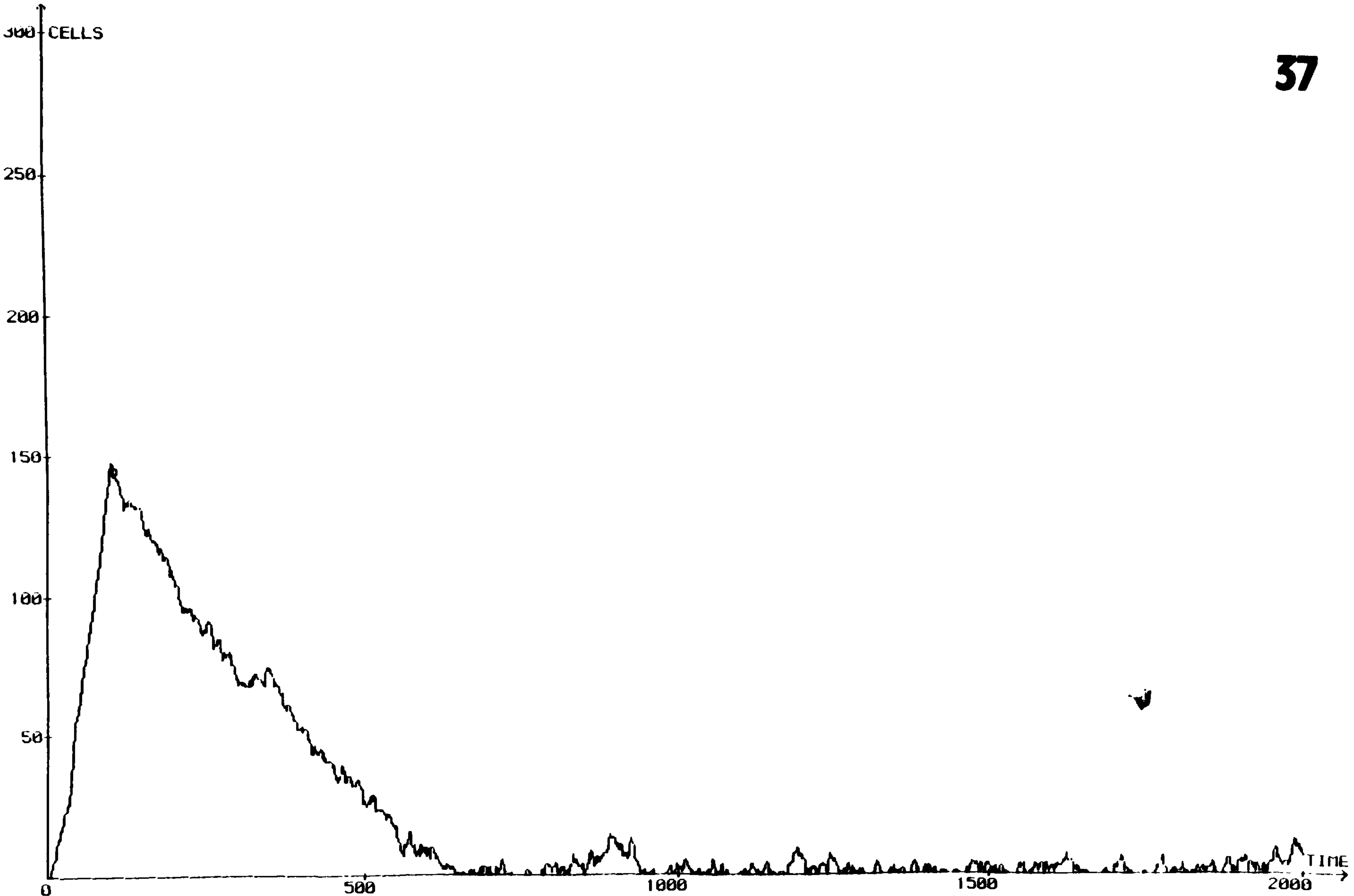

CELLS
600
550
500
450
400
350
300
250
200
150
100
50
0
500
1000
1500
2000
TIME
32

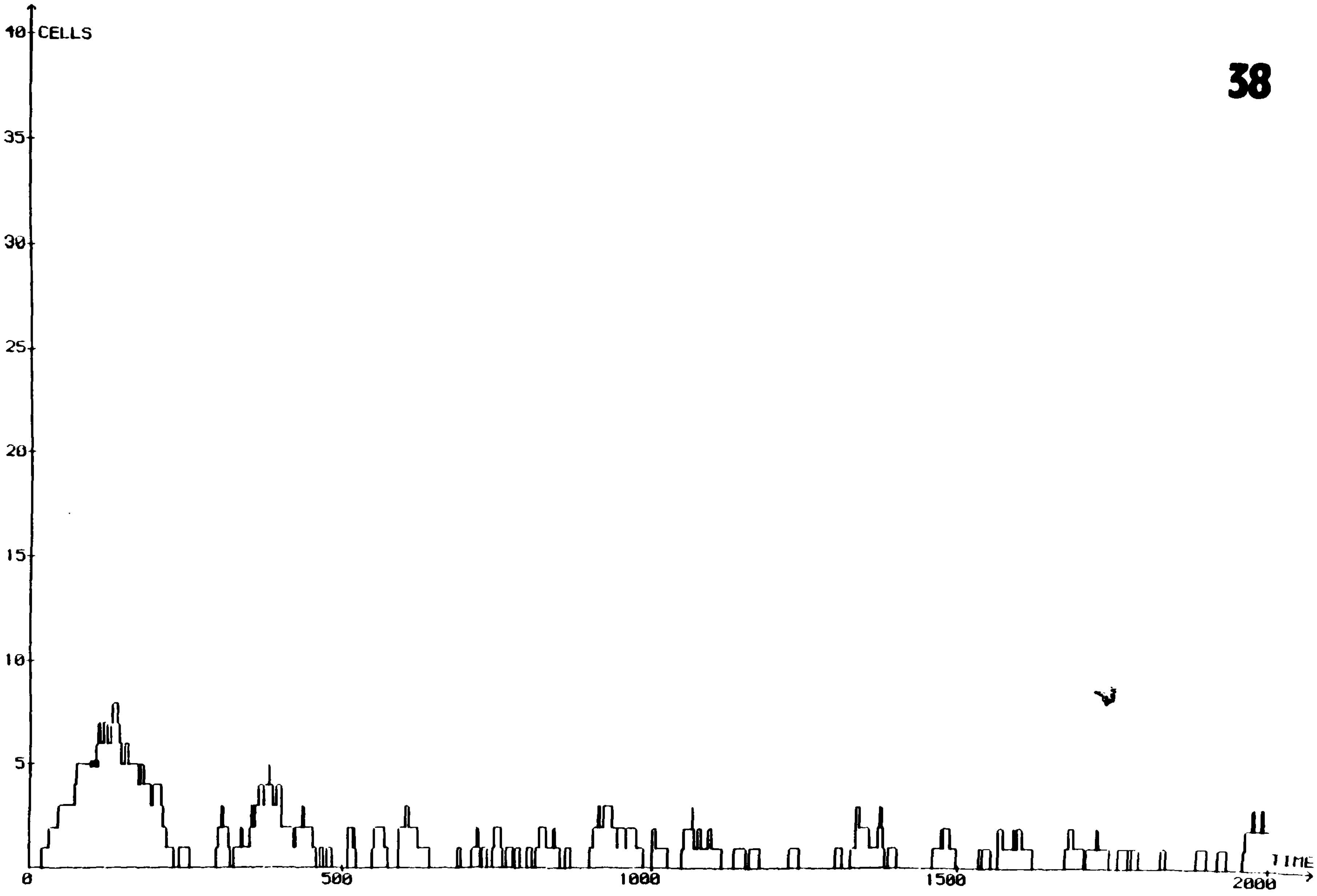

33
CELLS
TIME
50
45
40
35
30
25
20
15
10
5
0
500
1000
1500
2000

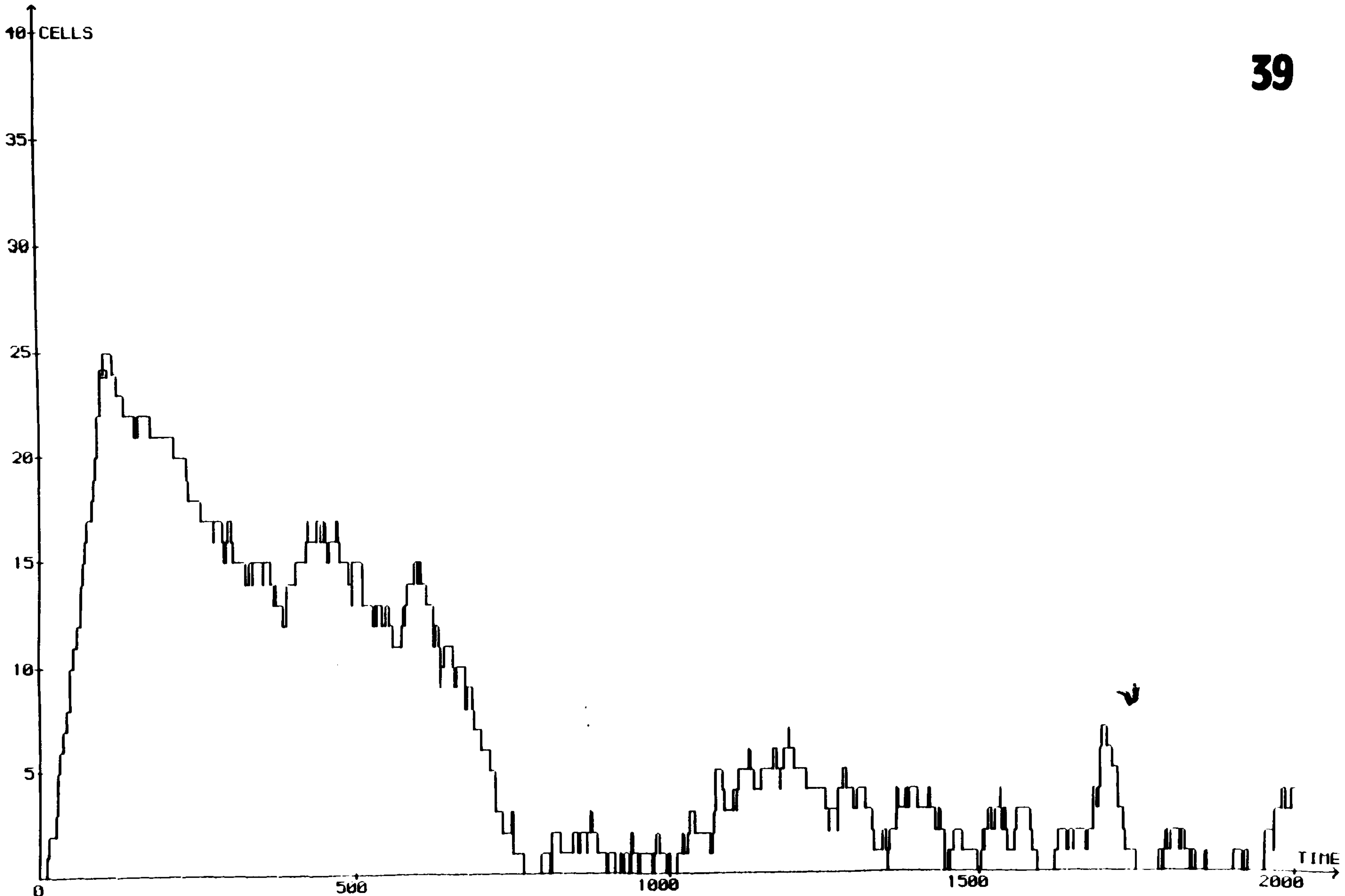

34
WAITING-TIME
TIME
400
350
300
250
200
150
100
50
0
500
1000
1500
2000

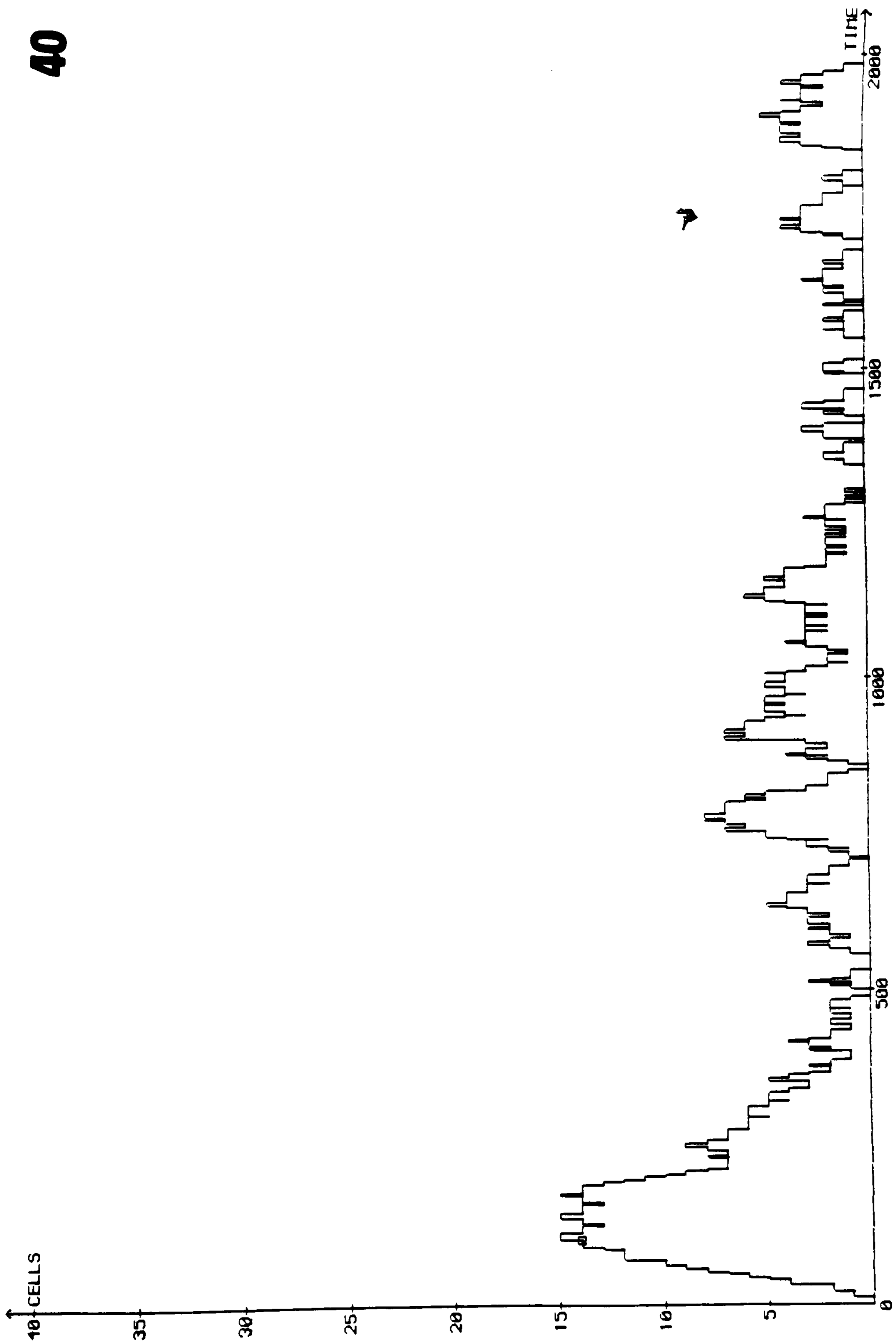

35
CELLS
300
250
200
150
100
50
0
500
1000
1500
2000
TIME

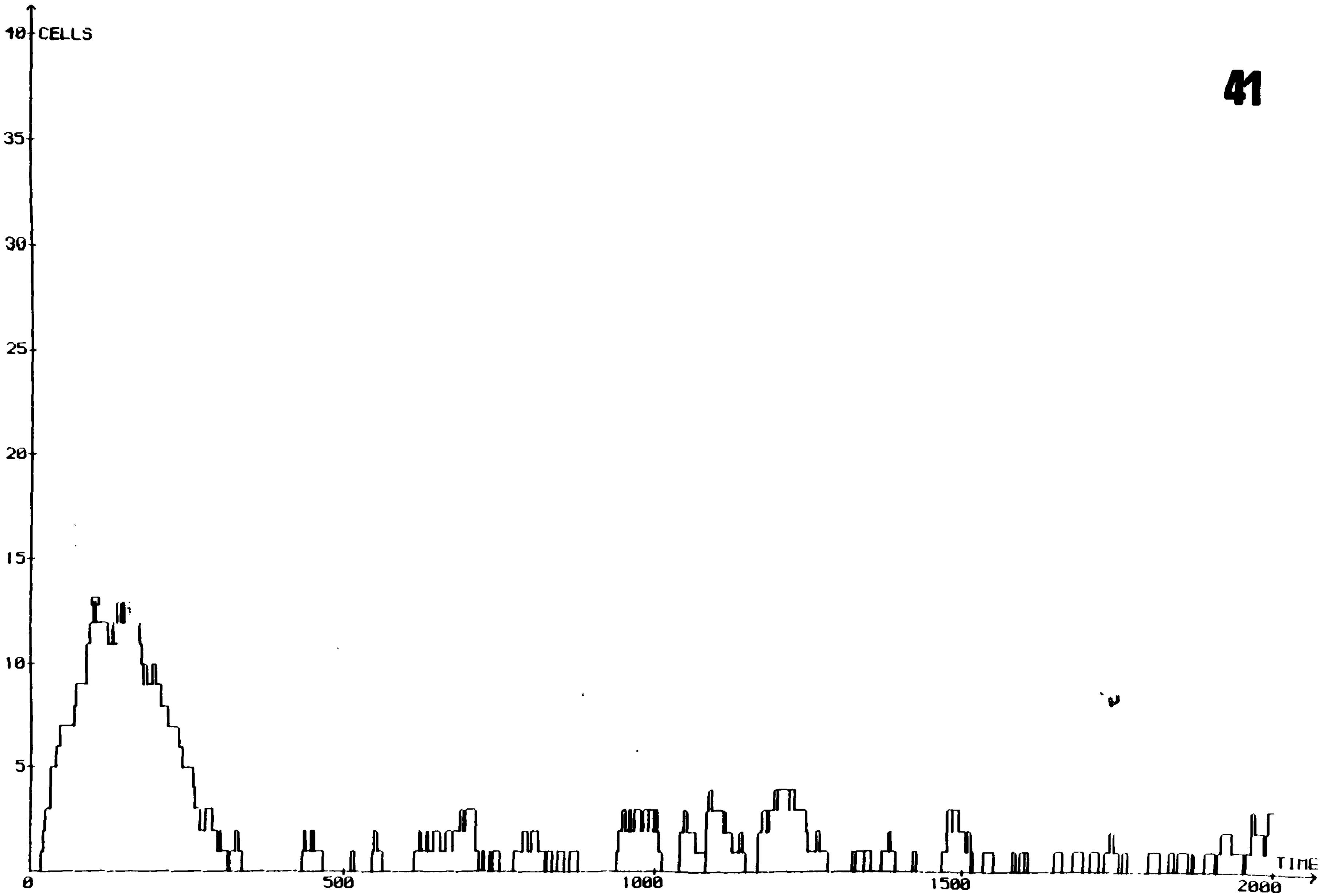

36
CELLS
300
250
200
150
100
50
0
500
1000
1500
2000
TIME

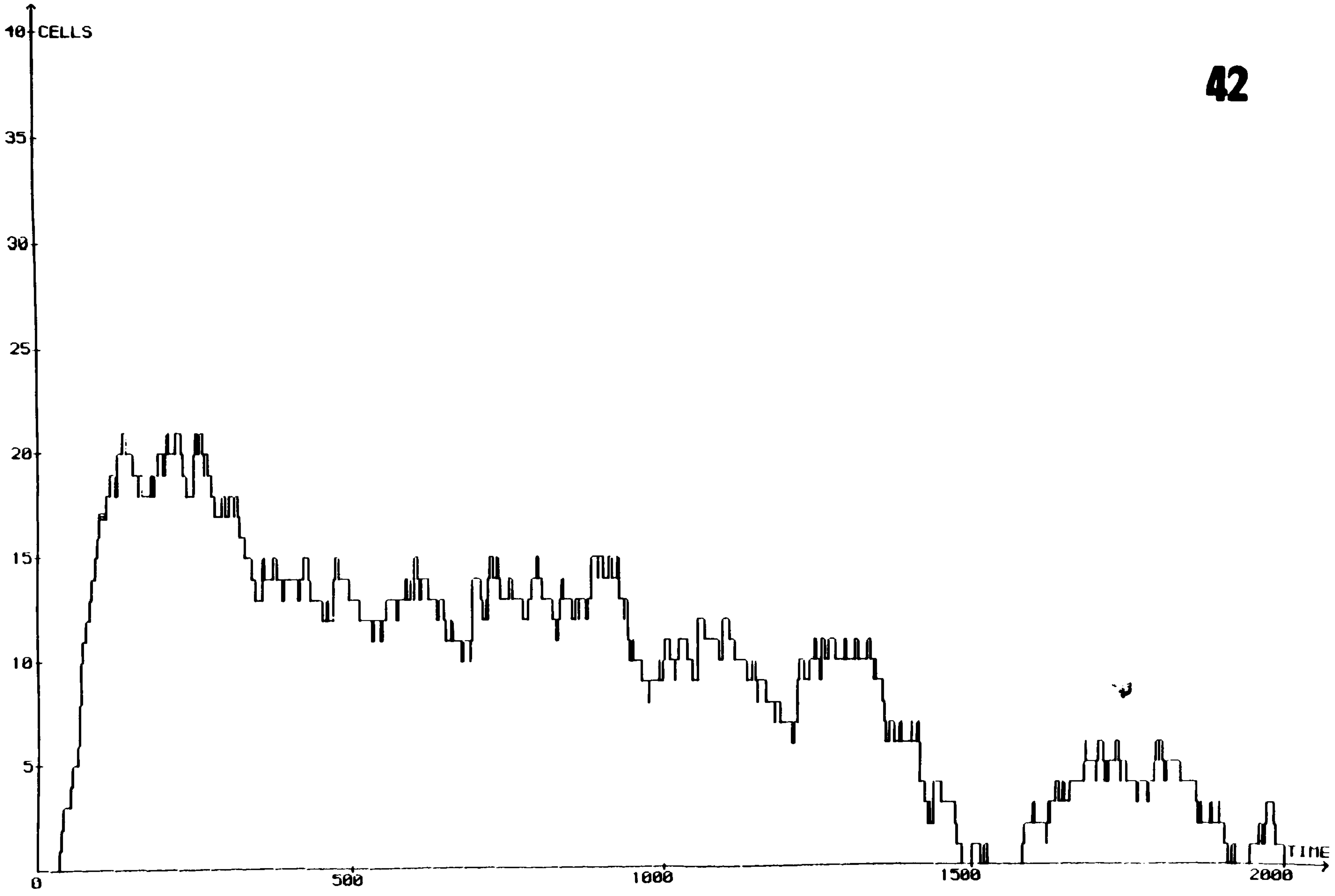

37
CELLS
300
250
200
150
100
50
0
500
1000
1500
2000
TIME


38
CELLS
TIME
40
35
30
25
20
15
10
5
0
500
1000
1500
2000

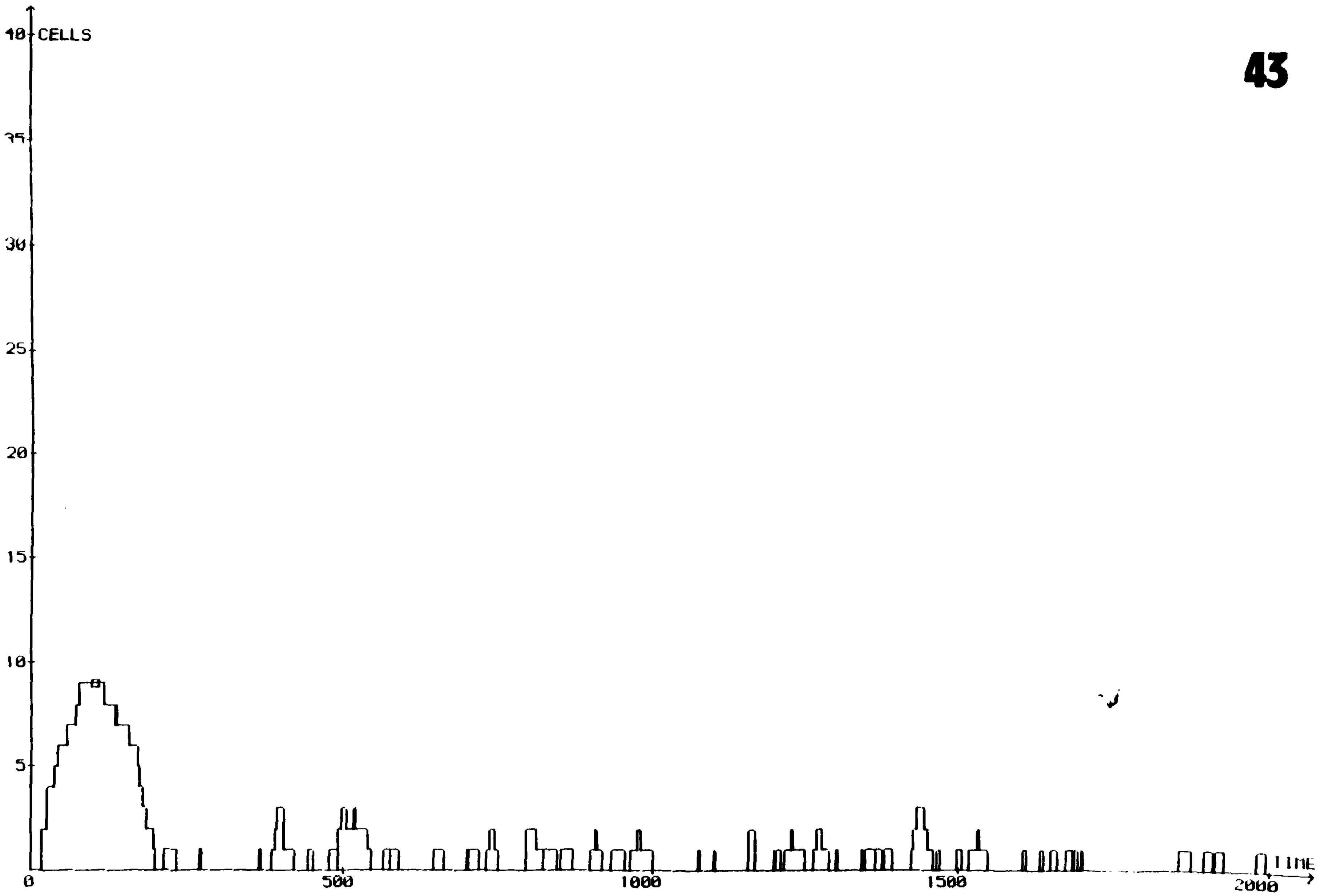

CELLS
40
35
30
25
20
15
10
5
0
500
1000
1500
2000
TIME
39

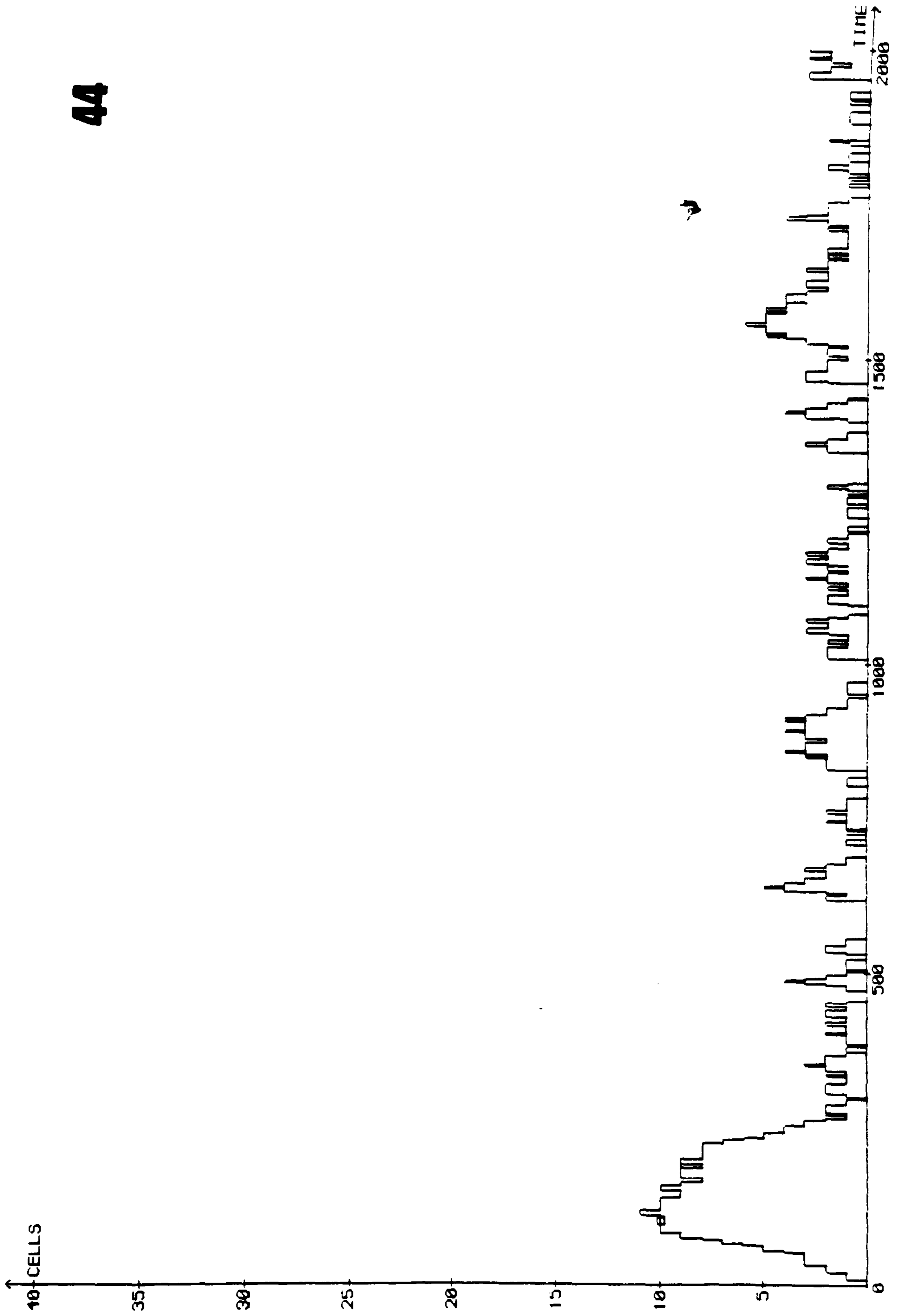

40
CELLS
TIME
0
5
10
15
20
25
30
35
40
0
500
1000
1500
2000

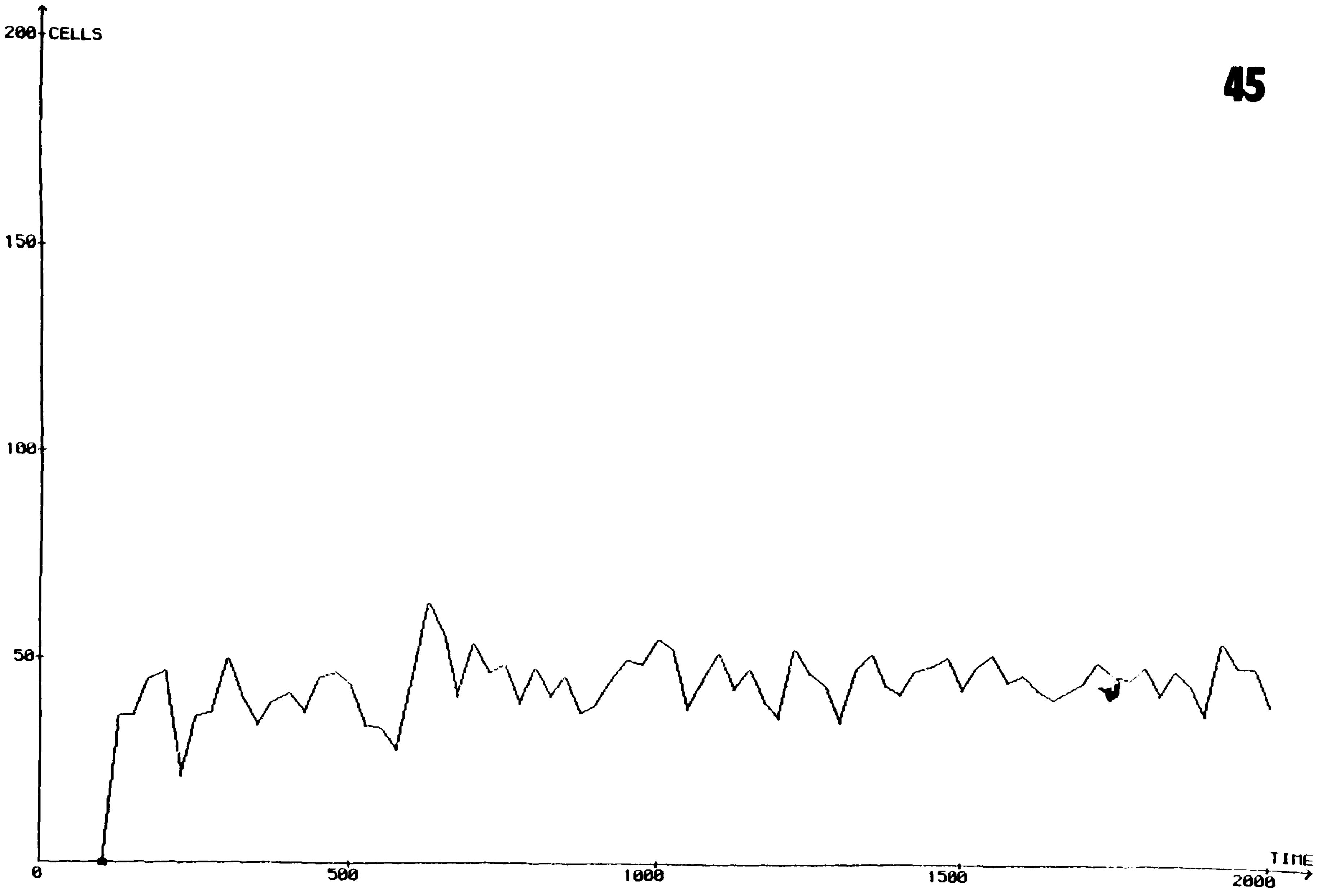

41
40
CELLS
35
30
25
20
15
10
5
0
500
1000
1500
2000
TIME

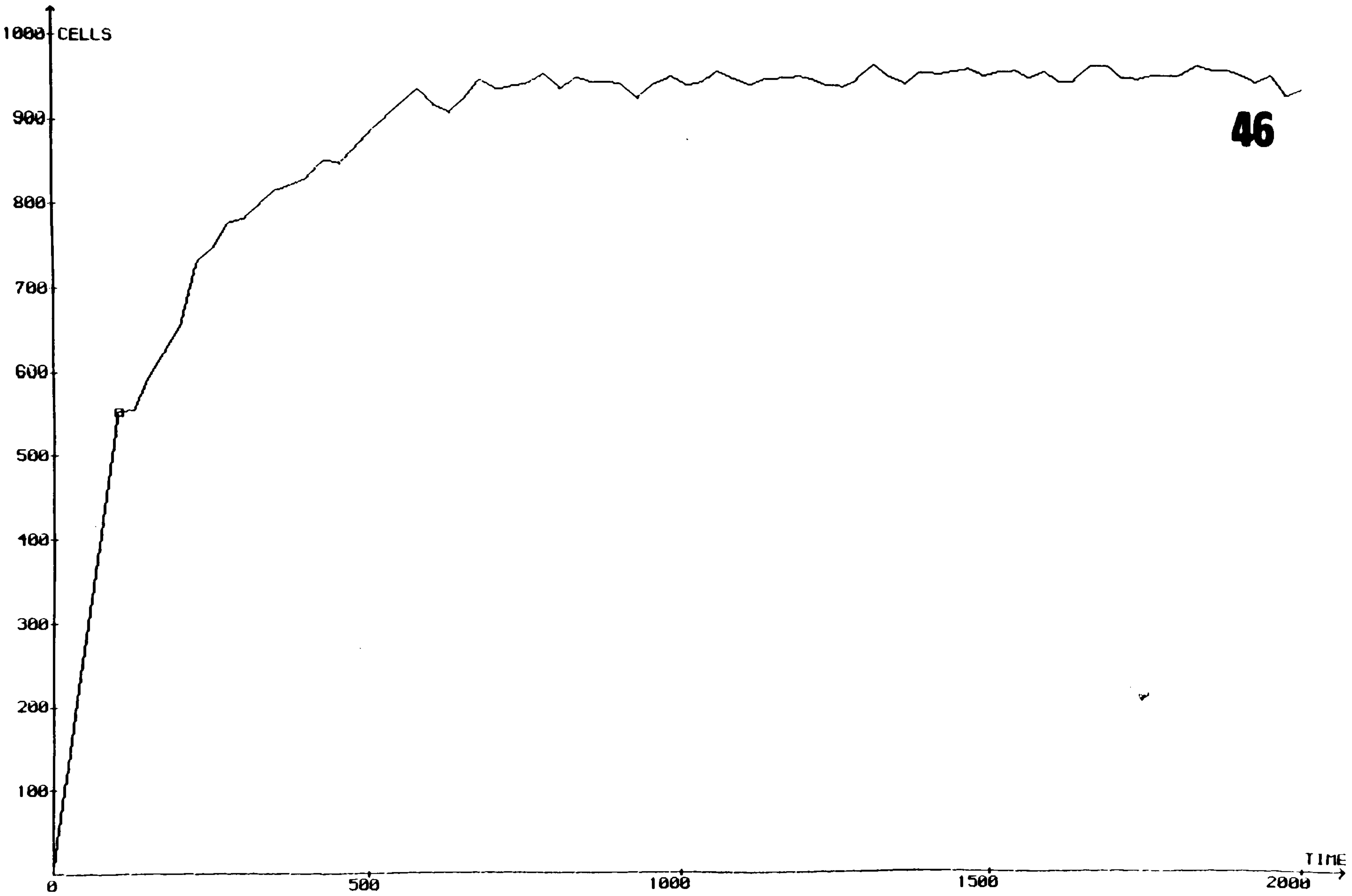


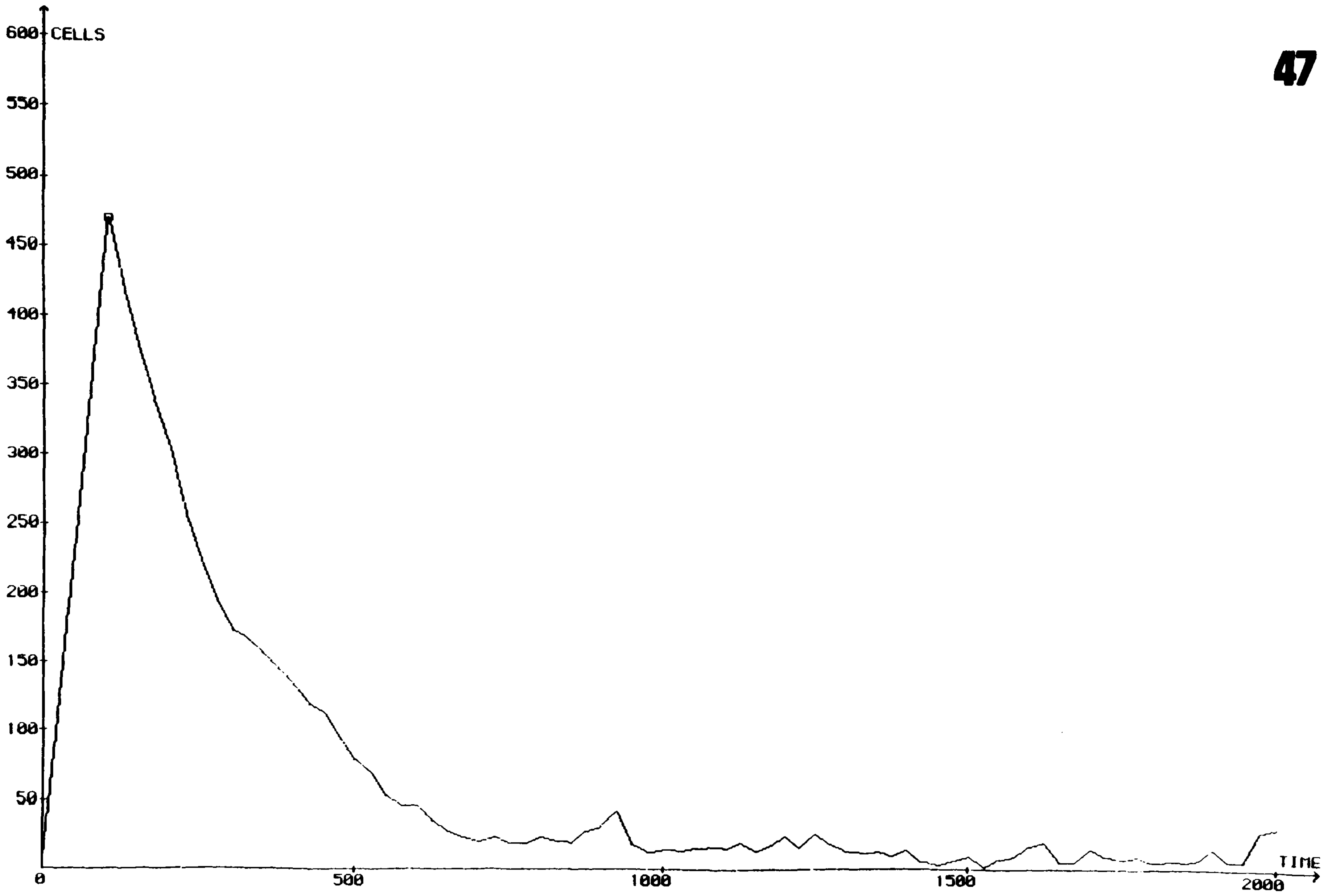
42
CELLS
40
35
30
25
20
15
10
5
0
500
1000
1500
2000
TIME

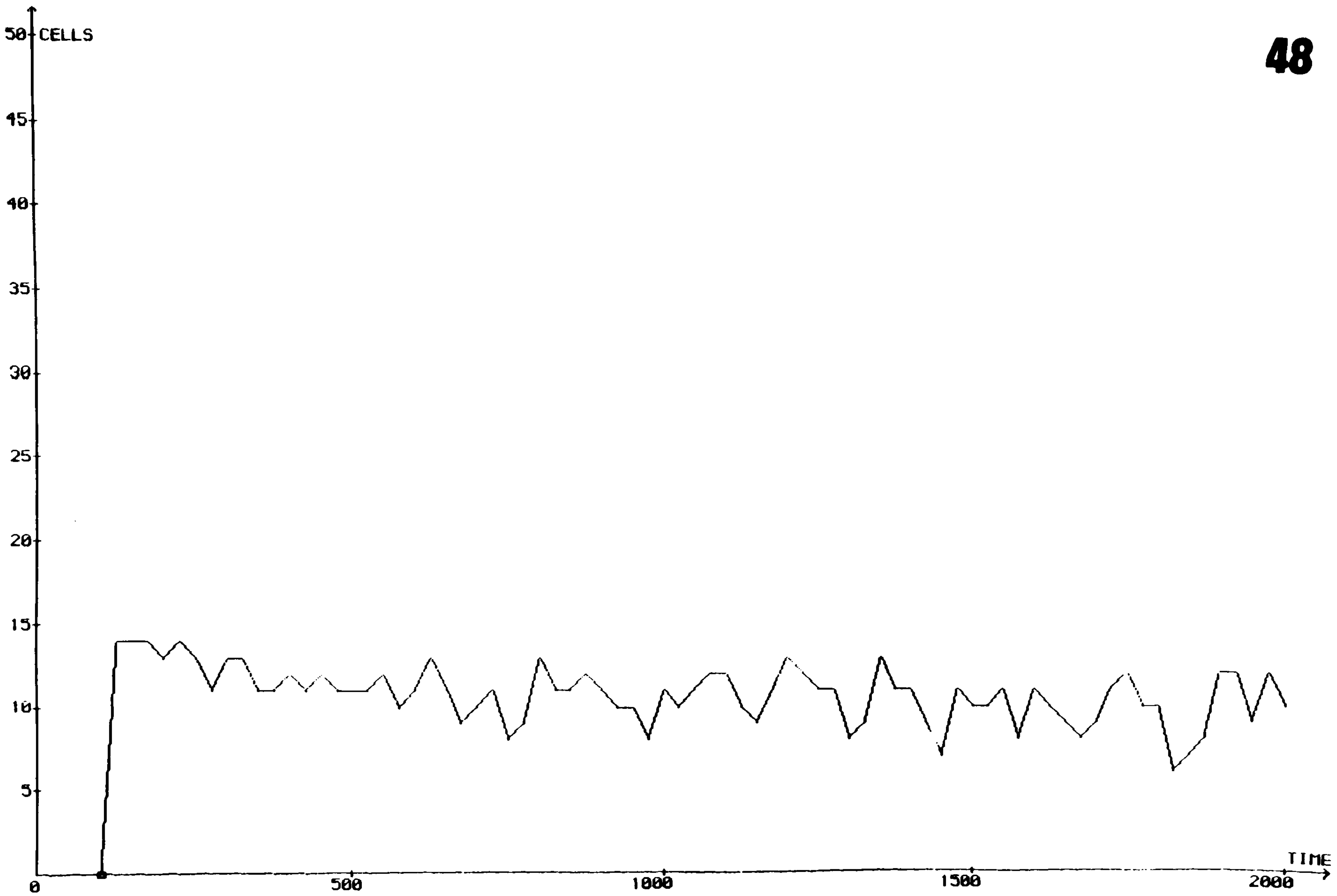

43
CELLS
TIME
40
30
25
20
15
10
5
0
500
1000
1500
2000

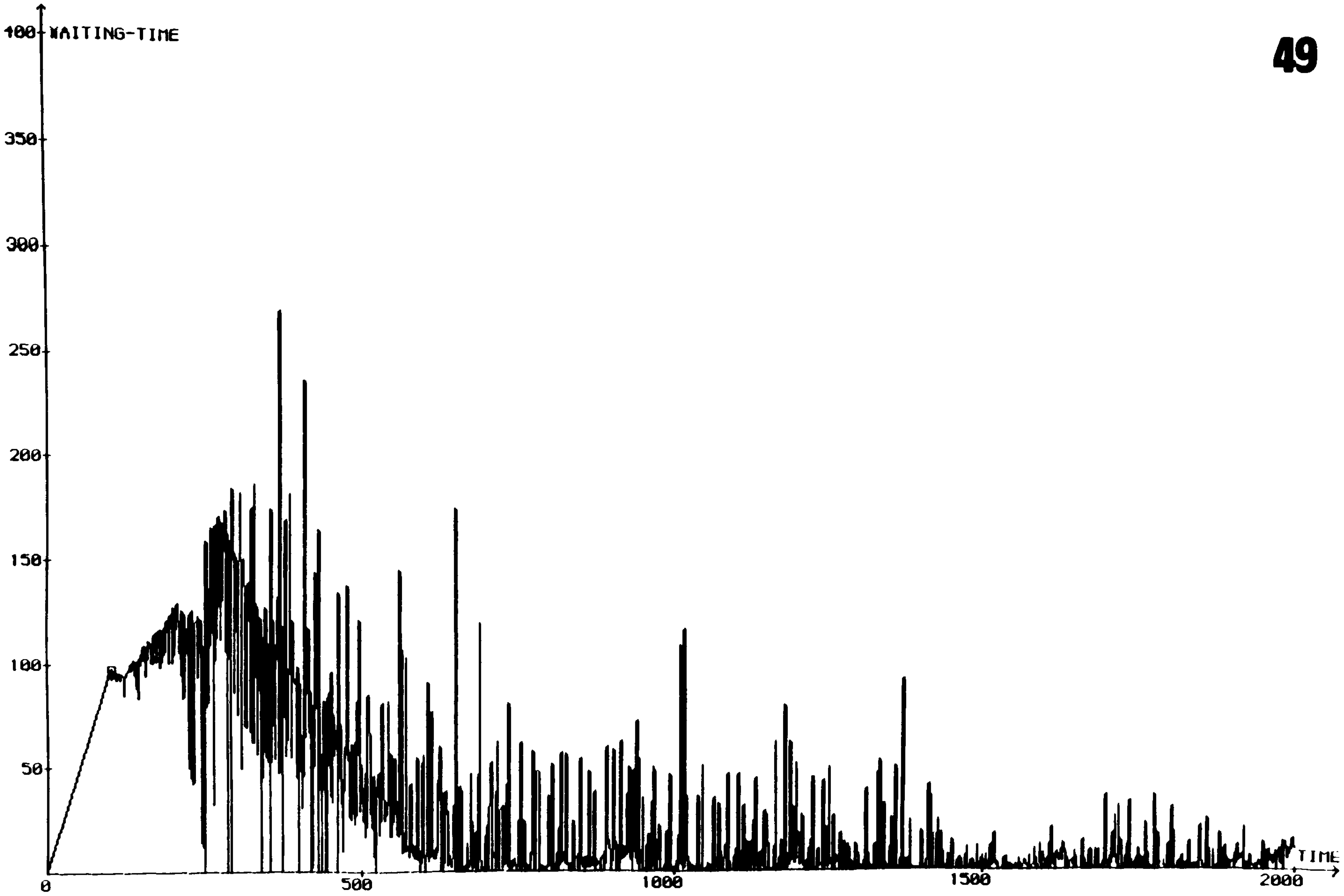

44
CELLS
40
35
30
25
20
15
10
5
0
500
1000
1500
2000
TIME


45
CELLS
200
150
100
50
0
500
1000
1500
2000
TIME


CELLS
1000
900
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
TIME
46


47
CELLS
TIME
600
550
500
450
400
350
300
250
200
150
100
50
0
500
1000
1500
2000


48
CELLS
50
45
40
35
30
25
20
15
10
5
0
500
1000
1500
2000
TIME


49
WAITING-TIME
TIME
400
350
300
250
200
150
100
50
0
500
1000
1500
2000

APPENDIX TWO

UNIVERSITY OF SHEFFIELD PASCAL COMPILER (1.8.1) ON 16/09/85 AT 10:42:37
OPTIONS CHOSEN : 64V WITH CHECKS NODEBUG

```
  1 001000 program simulator(input, output, rsystem, rcell, qfile, qlength, cfile, snapshot);
  2
  3
  4        (***************************************************************************)
  5        (*                                                                         *)
  6        (*    This is a simulation of the CELLULAR COMPUTER SYSTEM                 *)
  7        (*                                                                         *)
  8        (*    It simulates a CELLULAR computer in that it can have an arbitary     *)
  9        (*    number of MODULES, CELLS and BUSES                                   *)
 10        (*                                                                         *)
 11        (*    This program is written in SHEFFIELD PASCAL running under PRIMOS     *)
 12        (*                       in WGIHE.  Summer 1985                            *)
 13        (*                                                                         *)
 14        (***************************************************************************)
 15
 16
 17        const
 18            startseed = 49631;
 19            maxcells = 64;                  (* Number of cells per module *)
 20            maxmodules = 16;                (* Number of modules in system *)
 21            maxglobalbus = 1;               (* Number of global buses *)
 22            maxintrabus = 3;                (* Number of intra buses *)
 23            maxinterbus = 3;                (* Number of inter buses *)
 24            maxiobus = 1;
 25            tickperiod = 0.000001;          (* One microsecond clock *)
 26            maxsimtime = 0.002;             (* Duration of simulation *)
 27            maxmemory = 100000000;          (* 100 Megabytes of main memory *)
 28            sendconst = 0.000005;           (* Time range constant for sending message requests *)
 29            ioconst = 0.000020;             (* Time range constant for io requests *)
 30            cellconst = 0.000010;           (* Time range constant for cell requests *)
 31            osconst = 0.000001;             (* Time range constant for operating system requests *)
 32            memsizeconst = 1000;            (* Memory requirement constant *)
 33            modfailconst = 999;             (* Chance of a module not powering up - out of 1000 *)
 34            cellfailconst = 990;            (* Chance of a cell not powering up - out of 1000 *)
 35            processingconst = 0.000200;     (* Time range for cell processing *)
 36            requestconst = 5;               (* 10-n out of 10 requests will be looked at *)
 37
 38        type
 39            message = packed array [1..15] of char;                (* For passing messages *)
 40            statustype = (idle, acti, wait, comm, dead);            (* The states a cell can be in *)
 41            requesttype = (io, sendre, oscall, celreq, cpujob);     (* What a cell wants to do *)
 42            bustype = (global, intra, inter, ioline);               (* Which bus a cell requires *)
 43            qhead = ^ event;
 44
 45            event =                                                 (* This record defines a request for
*)
 46                record
 47                    busrequest:  real;
 48                    busrelease:  real;
 49                    busgrant:    real;
 50                    waitingtime: real;
 51                    mastertime:  real;
 52                    moduleno: integer;
 53                    cellno:   integer;
 54                    calltype: requesttype;
 55                    link: qhead
 56                end;
 57
 58
 59
 60            (***************************************************************************)
 61            (*                                                                         *)
```

```
62          (*    All the queues in the system are implemented as a linear linked      *)
63          (*    list. With pointers HEAD and TAIL indicating the front and rear      *)
64          (*    of each queue                                                        *)
65          (*                                                                         *)
66          (***************************************************************************)
67
68
69          queuepointers =
70              record
71                  head: qhead;
72                  tail: qhead;
73                  status: statustype
74              end;
75
76          cellelement =                       (* This record defines the structure of a cell *)
77              record
78                  status:     integer;
79                  prevstatus: integer;
80                  globalcall: integer;
81                  intracall:  integer;
82                  intercall:  integer;
83                  systimes:   integer;
84                  qtimes:     integer;
85                  glocall:   integer;
86                  celcall:   integer;
87                  sencall:   integer;
88                  iocall:    integer;
89                  memreqsize: integer;
90                  entersys:   real;
91                  enterqueue: real;
92                  exitqueue:  real;
93                  processing: real;
94                  lastreq: requesttype;
95                  queue:   boolean
96              end;
97
98          moduleelement =                     (* What is the module doing *)
99              record
100                 status: statustype
101             end;
102
103         modules = array [1..maxcells] of cellelement;       (* How many cells per module *)
104
105
106
107
108
109     Var
110         seed:       integer;
111         simtime:    real;                   (* System clock *)
112         globalbus:  integer;
113         intrabus:   integer;
114         interbus:   integer;
115         iobus:      integer;
116
117
118
119         qfile:   text;                  (* File of time spent in queue *)
120         cfile:   text;                  (* File of time spent in system *)
121         qlength: text;                  (* File of number in queues *)
122         rcell:   text;                  (* File of cell processing results *)
123         rsystem: text;                  (* File of system results *)
124         snapshot:text;                  (* File of cell states *)
125
126
127         globalq:  array [1..maxglobalbus] of queuepointers;
```

```
128            intraq:   array [1..maxintrabus] of queuepointers;
129            interq:   array [1..maxinterbus] of queuepointers;
130            iobusq:   array [1..maxiobus] of queuepointers;
131            computer: array [1..maxmodules] of modules;
132
133            snap:            integer;
134            time:            integer;
135            snapcount:       integer;
136            averageacti:     integer;
137            averageidle:     integer;
138            averagewait:     integer;
139            averagecomm:     integer;
140            averagedead:     integer;
141
142            avsystimes:      array [1..maxmodules] of integer;
143            avqtimes:        array [1..maxmodules] of integer;
144            avglocall:       array [1..maxmodules] of integer;
145            avcelcall:       array [1..maxmodules] of integer;
146            avsencall:       array [1..maxmodules] of integer;
147            aviocall:        array [1..maxmodules] of integer;
148            avcpujobs:       array [1..maxmodules] of integer;
149            avglobalcall:    array [1..maxmodules] of integer;
150            avintracall:     array [1..maxmodules] of integer;
151            avintercall:     array [1..maxmodules] of integer;
152
153            module:          integer;
154            cell:            integer;
155            loadfactor:      integer;
156            run:             integer;
157            globalmem:       integer;
158            numofremovals:   integer;
159            cellmiss:        integer;
160
161            iocount:         integer;
162            sencount:        integer;
163            oscount:         integer;
164            celcount:        integer;
165
166            iorequest:       integer;
167            sendrequest:     integer;
168            osrequest:       integer;
169            cellrequest:     integer;
170
171            ioqcount:        integer;
172            morerequests:    boolean;
173            param:           array [1..10] of integer;
174
175
176
177
178
179
180
181
182        (*****************************************************************)
183        (*                                                               *)
184        (*      Now start all the functions useful to the system         *)
185        (*                                                               *)
186        (*****************************************************************)
187
188
189            function random(x:real):real;
190 001420     begin
191 001445        seed:=(13849+25173*seed) mod 65536;
192 001466        random:=seed/65536;
193 001475     end;
```

```
194
195
196
197            function modulefail: boolean;
198            var
199                dummy: real;
200                rnd:   integer;
201
202 001500     begin
203 001520         dummy := 32.31;
204 001524         rnd := trunc(random(dummy) * 1000);
205 001535         if rnd > modfailconst then
206 001545             modulefail := true
207 001545         else
208 001550             modulefail := false
209 001550     end; { modulefail }
210
211
212
213            function cellfail: boolean;
214            var
215                dummy: real;
216                rnd:   integer;
217
218 001554     begin
219 001574         dummy := 32.14;
220 001600         rnd := trunc(random(dummy) * 1000);
221 001611         if rnd > cellfailconst then
222 001621             cellfail := true
223 001621         else
224 001624             cellfail := false
225 001624     end; { cellfail }
226
227
228
229            function bustime(request: requesttype): real;
230            (* This function generates the bus time required by a cell *)
231            var
232                 rnd:        real;
233                 dummyvalue: real;
234
235 001630     begin
236 001654          dummyvalue := 29.1;
237 001660          case request of
238 001661              (* Decide what type of request it is *)
239 001661              (* The time constants need to be different for each type of request *)
240 001661              io:
241 001661                  rnd := random(dummyvalue) * ioconst;
242 001672              sendre:
243 001672                  rnd := random(dummyvalue) * sendconst;
244 001703              oscall:
245 001703                  rnd := random(dummyvalue) * osconst;
246 001714              celreq:
247 001714                  rnd := random(dummyvalue) * cellconst;
248 001725              cpujob:
249 001725                  (*  do nothing *)
250 001725          end; (* End the case *)
251 001743          bustime := rnd
252 001743     end; { bustime }
253
254
255
256            function jobselection: integer;
257            (* This determines if a bus request is required if so the *)
258            (* function returns the value of the type of bus request *)
259            var
```

```
260                 test:       integer;
261                 dummyvalue: real;
262
263 001752      begin
264 001772          dummyvalue := 23.1;
265 001776          test := trunc(random(dummyvalue) * 10);
266 002007          if test >= requestconst then
267 002017             test := trunc(random(dummyvalue) * 10)
268 002026          else
269 002031             test := 0;
270 002034          jobselection := test;
271 002040      end; { jobselection }
272
273
274
275             function memsize: integer;
276             (* This calculates the amount of global memory required by a cell *)
277             var
278                 dummyvalue: real;
279
280 002043      begin
281 002063          dummyvalue := 32.1;
282 002067          memsize := trunc(random(dummyvalue) * memsizeconst)
283 002076      end; { memsize }
284
285
286
287             function processingload: real;
288             (* This function generates the processing time required by a cell *)
289             var
290                 rnd:        real;
291                 dummyvalue: real;
292
293 002103      begin
294 002123          dummyvalue := 29.1;
295 002127          rnd := random(dummyvalue) * processingconst;
296 002137          processingload := rnd
297 002137      end; { processingload }
298
299
300
301
302
303
304         (*****************************************************************)
305         (*                                                               *)
306         (*      Now start all the procedures used in the simulator       *)
307         (*                                                               *)
308         (*****************************************************************)
309
310
311             procedure clear;
312 002146      begin
313 002166          write(chr(155),'+')
314 002205      end; { clear }
315
316
317
318             procedure home;
319 002206      begin
320 002226          write(chr(158))
321 002240      end; { home }
322
323
324
325             procedure openfiles(directory: integer);
```

```
326             (* Prepare all file for a run of the program *)
327
328 002241      begin
329 002266           case directory of
330 002267                1:
331 002267                     begin
332 002267                          rewrite(qfile,    '/R1/Qfile.dat');
333 002277                          rewrite(cfile,    '/R1/Cfile.dat');
334 002307                          rewrite(qlength,  '/R1/Qlength.dat');
335 002317                          rewrite(rcell,    '/R1/Rcell.dat');
336 002327                          rewrite(rsystem,  '/R1/Rsystem.dat');
337 002337                          rewrite(snapshot, '/R1/Snapshot.dat')
338 002347                     end;
339
340 002350                2:
341 002350                     begin
342 002350                          rewrite(qfile,    '/R2/Qfile.dat');
343 002360                          rewrite(cfile,    '/R2/Cfile.dat');
344 002370                          rewrite(qlength,  '/R2/Qlength.dat');
345 002400                          rewrite(rcell,    '/R2/Rcell.dat');
346 002410                          rewrite(rsystem,  '/R2/Rsystem.dat');
347 002420                          rewrite(snapshot, '/R2/Snapshot.dat')
348 002430                     end;
349
350 002431                3:
351 002431                     begin
352 002431                          rewrite(qfile,    '/R3/Qfile.dat');
353 002441                          rewrite(cfile,    '/R3/Cfile.dat');
354 002451                          rewrite(qlength,  '/R3/Qlength.dat');
355 002461                          rewrite(rcell,    '/R3/Rcell.dat');
356 002471                          rewrite(rsystem,  '/R3/Rsystem.dat');
357 002501                          rewrite(snapshot, '/R3/Snapshot.dat')
358 002511                     end;
359
360 002512                4:
361 002512                     begin
362 002512                          rewrite(qfile,    '/R4/Qfile.dat');
363 002522                          rewrite(cfile,    '/R4/Cfile.dat');
364 002532                          rewrite(qlength,  '/R4/Qlength.dat');
365 002542                          rewrite(rcell,    '/R4/Rcell.dat');
366 002552                          rewrite(rsystem,  '/R4/Rsystem.dat');
367 002562                          rewrite(snapshot, '/R4/Snapshot.dat')
368 002572                     end;
369
370 002573                5:
371 002573                     begin
372 002573                          rewrite(qfile,    '/R5/Qfile.dat');
373 002603                          rewrite(cfile,    '/R5/Cfile.dat');
374 002613                          rewrite(qlength,  '/R5/Qlength.dat');
375 002623                          rewrite(rcell,    '/R5/Rcell.dat');
376 002633                          rewrite(rsystem,  '/R5/Rsystem.dat');
377 002643                          rewrite(snapshot, '/R5/Snapshot.dat')
378 002653                     end;
379
380 002654                6:
381 002654                     begin
382 002654                          rewrite(qfile,    '/R6/Qfile.dat');
383 002664                          rewrite(cfile,    '/R6/Cfile.dat');
384 002674                          rewrite(qlength,  '/R6/Qlength.dat');
385 002704                          rewrite(rcell,    '/R6/Rcell.dat');
386 002714                          rewrite(rsystem,  '/R6/Rsystem.dat');
387 002724                          rewrite(snapshot, '/R6/Snapshot.dat')
388 002734                     end;
389
390 002735                7:
391 002735                     begin
```

```
392 002735                          rewrite(qfile,    '/R7/Qfile.dat');
393 002745                          rewrite(cfile,    '/R7/Cfile.dat');
394 002755                          rewrite(qlength,  '/R7/Qlength.dat');
395 002765                          rewrite(rcell,    '/R7/Rcell.dat');
396 002775                          rewrite(rsystem,  '/R7/Rsystem.dat');
397 003005                          rewrite(snapshot,'/R7/Snapshot.dat')
398 003015                     end;
399
400 003016                8:
401 003016                     begin
402 003016                          rewrite(qfile,    '/R8/Qfile.dat');
403 003026                          rewrite(cfile,    '/R8/Cfile.dat');
404 003036                          rewrite(qlength,  '/R8/Qlength.dat');
405 003046                          rewrite(rcell,    '/R8/Rcell.dat');
406 003056                          rewrite(rsystem,  '/R8/Rsystem.dat');
407 003066                          rewrite(snapshot,'/R8/Snapshot.dat')
408 003076                     end;
409
410 003077                9:
411 003077                     begin
412 003077                          rewrite(qfile,    '/R9/Qfile.dat');
413 003107                          rewrite(cfile,    '/R9/Cfile.dat');
414 003117                          rewrite(qlength,  '/R9/Qlength.dat');
415 003127                          rewrite(rcell,    '/R9/Rcell.dat');
416 003137                          rewrite(rsystem,  '/R9/Rsystem.dat');
417 003147                          rewrite(snapshot,'/R9/Snapshot.dat')
418 003157                     end;
419
420 003160                10:
421 003160                     begin
422 003160                          rewrite(qfile,    '/R10/Qfile.dat');
423 003170                          rewrite(cfile,    '/R10/Cfile.dat');
424 003200                          rewrite(qlength,  '/R10/Qlength.dat');
425 003210                          rewrite(rcell,    '/R10/Rcell.dat');
426 003220                          rewrite(rsystem,  '/R10/Rsystem.dat');
427 003230                          rewrite(snapshot,'/R10/Snapshot.dat')
428 003240                     end;
429 003241          end;            (* end case *)
430
431 003271          writeln(qfile,' simtime   module  cell  busrequest  busgrant  busrelease  waitingtime  mas
ime
          request');
432 003306          writeln(qfile);
433 003315          writeln(cfile,' simtime   module  cell  entersys  enterqueue  exitqueue  exitsys  processi
requ
         est');
434 003332          writeln(cfile);
435 003341          writeln(qlength,' simtime   globalq  intral  intra2  intra3  interl  inter2  inter3  iobus
436 003356          writeln(qlength);
437 003365          writeln(snapshot, ' simtime  free  acti  wait  comm');
438 003402          writeln(snapshot);
439 003411          writeln(rcell, 'module  cell  C.active  qtimes  gloscall  c.request  sendcall  iocall  glc
all
          intracall  intercall');
440 003426          writeln(rcell)
441 003435      end; { openfiles }
442
443
444
445             procedure closefiles;
446 003436      begin
447 003456          { close all the files }
448 003456      end; { closefiles }
449
450
451
```

```
452             procedure displaytime:
453             (* Write on the screen the current simulation time *)
454
455 003457      begin
456 003477          writeln:
457 003506          writeln('                  Simulation time is '. simtime: 1: 6. '  Run number '. run):
458 003545          writeln
459 003545      end: ( displaytime )
460
461
462
463             procedure printheader :
464             (* Show current Configuration *)
465
466 003555      begin
467 003575          writeln('         Cellular Computer Simulation Version 1.0'):
468 003612          writeln('         ----------------------------------------'):
469 003627          writeln:
470 003636          writeln('         Current configuration is:-'):
471 003653          writeln('         --------------------------'):
472 003670          writeln:
473 003677          writeln('              '. maxcells: 4. ' Cells per Module'):
474 003730          writeln('              '. maxmodules: 4. ' Modules per Machine'):
475 003761          writeln('              '. maxglobalbus: 4. ' Global Bus'):
476 004012          writeln('              '. maxintrabus: 4. ' Intra Buses'):
477 004043          writeln('              '. maxinterbus: 4. ' Inter Buses'):
478 004074          writeln('              '. maxiobus: 4. ' I/O Buses'):
479 004125          writeln
480 004125      end: ( printheader )
481
482
483             procedure startcell(module. cell: integer):
484             (* This sets up a cell as an active processor *)
485
486 004135      begin
487 004166          with computer[module][cell] do begin
488 004225              status := 1:
489 004231              entersys := simtime:
490 004237              systimes := systimes + 1:
491 004251              processing := processingload + simtime:
492 004261              lastreq := cpujob
493 004261          end
494 004267      end: ( startcell )
495
496
497
498             procedure servecell(module. cell: integer: request: requesttype):
499             var
500                 newmod:  integer:
501                 newcell: integer:
502                 found:   boolean:
503             (* This decides what a cell requires when it is bus master *)
504             (* Note: at the moment we assure infinite memory and cells *)
505
506 004270      begin
507 004324          case request of
508 004325              io:
509 004325                  begin
510 004325                     iocount := iocount + 1
511 004325                  end:
512 004334              (* Do some input/output  *)
513
514 004334              sendre:
515 004334                  begin
516 004334                      (* Send a message over modules *)
517 004334                      sencount := sencount + 1
```

```
518 004334              end;
519
520 004343          oscall:
521 004343              begin
522 004343                  (* Do a call to the operating system *)
523 004343                  oscount := oscount + 1
524 004343              end;
525
526 004352          cpujob:
527 004352              begin
528 004352                  (* Do a job within the cell *)
529 004352              end;
530
531 004353          celreq:
532 004353              begin
533 004353                  (* Cell requires a new cpu to help *)
534 004353                  celcount := celcount + 1;
535 004361                  newmod := 1;
536 004365                  newcell := 1;
537 004371                  found := false;
538 004373                  while (not found) and (newmod <= maxmodules) do begin
539 004410                      while (not found) and (newcell <= maxcells) do begin
540 004425                          if computer[newmod][newcell].status = 0 then begin
541 004464                              found := true
542 004464                          end else
543 004467                              newcell := newcell + 1
544 004467                      end;
545
546 004476                      if not found then begin
547 004503                          newcell := 1;
548 004507                          newmod := newmod + 1
549 004507                      end
550 004515                  end;
551 004516                  if not found then begin
552 004523                      cellmiss := cellmiss + 1;
553 004531                      write(rsystem,' cellmiss',cellmiss:3, ' at time', simtime:9:6);
554 004566                      writeln(rsystem,'   requested by module', module:3,'  cell', cell:3)
555 004625                  end else
556 004626                      startcell(newmod, newcell)
557 004626              end
558 004634      end (* case *)
559 004652  end; { servecell }
560
561
562
563         procedure  addto(var head, tail: qhead; request: requesttype; module,cell: integer);
564         (* Queue a new bus request *)
565         var
566              newrec: qhead;
567
568 004653  begin
569 004707       new(newrec);
570 004715       with newrec^ do begin
571 004721            (* First set up necessary fields *)
572 004721            busrequest := simtime;
573 004725            busgrant:= simtime;
574 004733            busrelease := 0;
575 004741            waitingtime := 0;
576 004747            mastertime := bustime(request) ;
577 004757            moduleno := module;
578 004765            cellno := cell;
579 004773            calltype := request;
580 005000            link := nil;
581 005005            computer[module][cell].enterqueue := simtime
582 005023       end;
583
```

```
584 005044          if head = nil then begin              (* If list empty then start it off *)
585 005050              head := newrec;
586 005054              tail := newrec
587 005054          end else begin                        (* Otherwise add on at end of list *)
588 005061              tail^.link := newrec;
589 005071              tail := newrec
590 005071          end
591 005075      end; ( addto )
592
593
594
595             procedure queuelengths(start: qhead);
596             (* See how many items in each bus queue *)
597             var
598                 next:  qhead;
599                 count: integer;
600
601 005076      begin
602 005123          next := start;
603 005127          count := 0;
604 005132          if next <> nil then begin
605 005136              while next <> nil do
606 005142                  with next^ do begin
607 005146                      count := count + 1;
608 005154                      next := link
609 005154                  end
610 005162          end;
611 005163          write(qlength, count)
612 005176      end; ( queuelengths )
613
614
615
616             procedure plotlengths;
617             var
618                 globalbus: integer;
619                 intrabus:  integer;
620                 interbus:  integer;
621                 iobus:     integer;
622
623 005177      begin
624 005217          write(qlength, simtime: 1: 6);
625 005232          for globalbus := 1 to maxglobalbus do
626 005255              queuelengths(globalq[globalbus].head);
627 005307          for intrabus := 1 to maxintrabus do
628 005332              queuelengths(intraq[intrabus].head);
629 005364          for interbus := 1 to maxinterbus do
630 005407              queuelengths(interq[interbus].head);
631 005441          for iobus := 1 to maxiobus do
632 005464              queuelengths(iobusq[iobus].head);
633 005516          writeln(qlength, ' ')
634 005532      end; ( plotlengths )
635
636
637
638
639             procedure setupqueues;
640             (* Queue requests for bus access *)
641             var
642                 globalbus:   integer;
643                 intrabus:    integer;
644                 interbus:    integer;
645                 iobus:       integer;
646                 module:      integer;
647                 cell:        integer;
648                 selector:    real;
649
```

```
650 005533      begin
651 005553          if morerequests then begin
652 005556              for module := 1 to maxmodules do begin
653 005601                  for cell := 1 to maxcells do begin
654 005624                      case jobselection of
655
656 005627                          0:
657 005627                              begin
658 005627                                 (null)
659 005627                              end;
660
661 005630                          1, 2, 3, 4:
662 005630                              begin
663 005630                                  (* Compute bound job *)
664 005630                                  if computer[module][cell].status = 0 then
665 005667                                      (* Just start a cell up *)
666 005667                                      startcell(module, cell)
667 005667                                      (* Randomize processingtime in the procedure *)
668 005667                              end;
669
670 005676                          5:
671 005676                              begin
672 005676                                  (* Cell request *)
673 005676                                  with computer[module][cell] do begin
674 005735                                      if (status = 1) and (simtime >= processing) then begin
675 005756                                      (* Queue a request for a new cell *)
676 005756                                      (* on Intra and Inter buses *)
677 005756                                          for intrabus := 1 to maxintrabus  do
678 006001                                              with intraq[intrabus]  do
679 006017                                                  addto(head, tail, celreq, module, cell);
680 006051                                          for interbus := 1 to maxinterbus  do
681 006074                                              with interq[interbus]  do
682 006112                                                  addto(head, tail, celreq, module, cell);
683 006144                                          cellrequest := cellrequest + 1;
684
685 006152                                          prevstatus := status;
686 006160                                          status := 2;
687 006164                                          intracall := intracall + 1;
688 006176                                          intercall := intercall + 1;
689 006210                                          qtimes := qtimes + 1;
690 006222                                          celcall := celcall + 1;
691 006234                                          lastreq := celreq;
692 006242                                          queue := true
693 006242                                      end
694 006247                                  end
695 006247                              end;
696
697 006250                        (  5:
698 006250                              begin
699 006250                                  (* Cell request *)
700 006250                                  with computer[module][cell] do begin
701 006250                                      if (status = 1) and (simtime >= processing) then begin
702 006250                                      (* Queue a request on intra buses only *)
703 006250                                          for intrabus := 1 to maxintrabus  do
704 006250                                              with intraq[intrabus]  do
705 006250                                                  addto(head, tail, celreq, module, cell);
706 006250                                          cellrequest := cellrequest + 1;
707
708 006250                                          prevstatus := status;
709 006250                                          status := 2;
710 006250                                          intracall := intracall + 1;
711 006250                                          qtimes := qtimes + 1;
712 006250                                          celcall := celcall + 1;
713 006250                                          lastreq := celreq;
714 006250                                          queue := true
715 006250                                      end
```

```
716 006250                          end
717 006250                      end;  }
718
719 006250                  6:
720 006250                      begin
721 006250                          (* OS call *)
722 006250                          with computer[module][cell] do begin
723 006307                              if (status = 1) and (simtime >= processing) then begin
724 006330                              (* Queue a request for the global operating system *)
725 006330                              (* on the global bus *)
726 006330                                  for globalbus := 1 to maxglobalbus  do
727 006353                                      with globalq[globalbus]  do
728 006371                                          addto(head, tail, oscall, module, cell);
729 006423                                  osrequest := osrequest + 1;
730
731 006431                                  prevstatus := status;
732 006437                                  status := 2;
733 006443                                  globalcall := globalcall + 1;
734 006455                                  qtimes := qtimes + 1;
735 006467                                  glocall := glocall + 1;
736 006501                                  lastreq := oscall;
737 006507                                  queue := true
738 006507                              end
739 006514                          end
740 006514                      end;
741
742 006515                  7, 8:
743 006515                      begin
744 006515                          (* Send message call *)
745 006515                          with computer[module][cell] do begin
746 006554                              if (status = 1) and (simtime >= processing) then begin
747 006575                              (* Queue a request for sending message *)
748 006575                              (* on the inter or intra bus *)
749 006575                                  selector := random(seed);
750 006610                                  if selector >= 0.7 then
751 006617                                     for interbus :=1 to maxinterbus  do
752 006642                                         with interq[interbus]  do
753 006660                                             addto(head, tail, sendre, module, cell)
754 006666                                  else
755 006713                                     for intrabus :=1 to maxintrabus  do
756 006736                                         with intraq[intrabus]  do
757 006754                                             addto(head, tail, sendre, module, cell);
758 007006                                  sendrequest := sendrequest + 1;
759
760 007014                                  prevstatus := status;
761 007022                                  status := 2;
762 007026                                  qtimes := qtimes + 1;
763 007040                                  sencall := sencall + 1;
764 007052                                  lastreq := sendre;
765 007057                                  queue := true;
766 007064                                  if selector >= 0.7 then
767 007073                                     intercall := intercall + 1
768 007073                                  else
769 007106                                     intracall := intracall + 1
770 007106                              end
771 007120                          end
772 007120                      end;
773
774 007121                  9:
775 007121                      begin
776 007121                          (* I/O call *)
777 007121                          with computer[module][cell] do begin
778 007160                              if (status = 1) and (simtime >= processing) then begin
779 007201                                  if ioqcount > 50 then
780 007211                                      for interbus := 1 to maxinterbus do
781 007234                                          with interq[interbus] do
```

```
782 007252                                              addto(head, tail, io, module, cell)
783 007260                                  else begin
784 007305                                      for iobus := 1 to maxiobus do
785 007330                                          with iobusq[iobus] do
786 007346                                              addto(head, tail, io, module, cell);
787 007400                                      ioqcount := ioqcount + 1
788 007400                                  end;
789 007406                                  iorequest := iorequest + 1;
790
791 007414                                  prevstatus := status;
792 007422                                  status := 2;
793 007426                                  qtimes := qtimes + 1;
794 007440                                  iocall := iocall + 1;
795 007452                                  lastreq := io;
796 007457                                  queue := true
797 007457                              end
798 007464                          end
799 007464                      end
800 007464              end (* case *)
801 007515          end (* cell statement *)
802 007515      end (* module statement *)
803 007525  end (* if *)
804 007535 end; ( setupqueues )
805
806
807
808        procedure deleteentry(var head, tail: qhead; cell, module: integer);
809        (* This procedure deletes the queue entry from the request from *)
810        (* Module = module & Cell = cell *)
811        var
812             current: qhead;
813             last:    qhead;
814             found:   boolean;
815
816 007536 begin
817 007567      current := head;
818 007573      last := head;
819 007577      found := false;
820 007601      while ( not found ) and ( current <> nil ) do begin
821 007612           (* While correct entry not found, skip along the queue *)
822 007612           (* When we find it set flag true *)
823 007612           if (module = current^.moduleno) and (current^.cellno = cell) then begin
824 007636               found := true
825 007636           end else begin
826 007641               last := current;
827 007645               current := current^.link
828 007645           end
829 007653      end;
830
831 007654      (* If we can't find it then the program is shot !! *)
832 007654      if not found then
833 007661           writeln(' ')
834 007675      else begin
835 007676           (* if entry to be deleted is the last or the only one in the queue *)
836 007676           if current = tail then begin
837 007706              tail := last;
838 007712              if head = current then begin
839 007722                  head := nil;
840 007725                  tail := nil
841 007725              end else
842 007731                  last^.link := nil
843 007731           end else if current = head then begin
844 007747                     (* if entry to be deleted is the first *)
845 007747                     head := current^.link;
846 007755                  end else
847 007756                     (* it must be in middle of list somewhere *)
```

```
848 007756                      last^.link := current^.link
849 007756          end
850 007772      end; ( deleteentry )
851
852
853
854             procedure removeentrys(bus: bustype; busnum, cellno, moduleno: integer; call: requesttype);
855             (* This decides which of the buses to delete the superflous entries from *)
856             var
857                 count: integer;
858
859 007773      begin
860 010036          case bus of
861 010037              global:
862 010037                  begin
863 010037                      for count := 1 to maxglobalbus do
864 010062                          if count <> busnum then
865 010072                              deleteentry(globalq[count].head, globalq[count].tail, cellno, modulen
866 010126                  end;
867                                                                                                  :
868 010151              intra:
869 010151                  begin
870 010151                      for count := 1 to maxintrabus do
871 010174                          if count <> busnum then
872 010204                             deleteentry(intraq[count].head, intraq[count].tail, cellno, moduleno);
873 010262                      case call of
874 010263                          celreq:
875 010263                              begin
876 010263                              for count := 1 to maxinterbus do
877 010306                                 deleteentry(interq[count].head, interq[count].tail, cellno, module
878 010364                              end;
879 010365                          sendre, oscall:
880 010365                              begin
881 010365                                      ( do nothing )
882 010365                              end
883 010365                      end                                                                      i
884 010377                  end;
885
886 010400              inter:
887 010400                  begin
888 010400                      for count := 1 to maxinterbus do
889 010423                          if count <> busnum then
890 010433                              deleteentry(interq[count].head, interq[count].tail, cellno, modulenc
891 010511                      case call of
892 010512                          celreq:
893 010512                              for count := 1 to maxintrabus do
894 010535                                 deleteentry(intraq[count].head, intraq[count].tail, cellno, modul
895 010614                          sendre, oscall, io:
896 010614                          begin
897 010614                              ( do nothing )
898 010614                          end
899 010614                      end
900 010631                  end;
901 010632              ioline:
902 010632                  begin
903 010632                      for count := 1 to maxiobus do
904 010655                          if count <> busnum then
905 010665                              deleteentry(iobusq[count].head, iobusq[count].tail, cellno, modulen
906 010721                  end;
907 010744          end;
908 010760      end; ( removeentrys )
909
910
911
912
913             procedure startqueue(var head, tail: qhead; var qstatus: statustype; bus: bustype; busnum: in
```

```
914            (* If a queue is idle then get first entry and start it up *)
915
916 010761     begin
917 011011          qstatus := acti;
918 011014          with head^ do begin
919 011024               waitingtime := simtime - busrequest;
920 011034               busgrant := simtime;
921 011042               computer[moduleno][cellno]. status := 3;
922 011105               (* Now get rid of extra calls on rest of buses *)
923 011105               removeentrys(bus, busnum, cellno, moduleno, calltype);
924 011142               servecell(moduleno, cellno, calltype)
925 011163          end
926 011173     end; { startqueue }
927
928
929
930
931            procedure queueserver(var head, tail: qhead; var qstatus: statustype; bus: bustype; busnum: int
);
932            var
933                 temp: qhead;
934
935 011174     begin
936 011224          if qstatus = idle then
937 011230               startqueue(head, tail, qstatus, bus, busnum);
938 011244          with head^ do
939 011254               if simtime >= (busgrant + mastertime) then begin
940 011270                    (* Current cell has finished with the bus *)
941 011270                    (* Remove the queue entry and start next, if any *)
942 011270                    numofremovals := numofremovals + 1;
943 011276                    if bus = ioline then
944 011305                         ioqcount := ioqcount - 1;
945 011313                    mastertime := simtime - busgrant;
946 011325                    write(qfile, simtime:1:6, moduleno, cellno:6, busrequest:12:6, busgrant:11:6, si
e:11
          :6, waitingtime:12:6, mastertime:13:6);
947
948 011450                    case calltype of
949 011451                         io:       writeln(qfile, '    ioreq');
950 011467                         celreq:   writeln(qfile, '   celreq');
951 011505                         sendre:   writeln(qfile, '  sendreq');
952 011523                         oscall:   writeln(qfile, '   oscall');
953 011541                    end;  (* case *)
954
955 011560                    (* Once the cell has finished set it back to what it was *)
956 011560                    (* doing before the call to this bus *)
957 011560                    busrequest := 0.0;
958 011564                    computer[moduleno][cellno].status := 1;
959 011627                    computer[moduleno][cellno].exitqueue := simtime;
960
961 011672                    if link <> nil then begin
962 011700                        (* There is another entry to start up *)
963 011700                        temp := head;
964 011704                        head := link;
965 011712                        dispose(temp);
966 011720                        (* now start up new cell I/O *)
967 011720                        with head^ do begin
968 011730                            waitingtime := simtime - busrequest;
969 011740                            busgrant := simtime;
970 011746                            computer[moduleno][cellno].status := 3;
971 012011                            (* Now get rid of extra calls on buses *)
972 012011                            removeentrys(bus, busnum, cellno, moduleno, calltype);
973 012046                            (* Now see what the cell wants to do *)
974 012046                            servecell(moduleno, cellno, calltype)
975 012067                        end
```

```
 976 012077                end else begin
 977 012100                    dispose(head);
 978 012106                    tail := nil;
 979 012111                    head := nil;
 980 012114                    qstatus := idle
 981 012114                end
 982 012117            end  (* if *)
 983 012117      end; { queueserver }
 984
 985
 986
 987
 988
 989
 990             procedure servicequeues;
 991             var
 992                  globalbus: integer;
 993                  intrabus:  integer;
 994                  interbus:  integer;
 995                  iobus:     integer;
 996
 997 012120      begin
 998 012140           for globalbus := 1 to maxglobalbus do
 999 012163                with globalq[globalbus] do begin
1000 012201                if head <> nil then
1001 012205                     queueserver(head, tail, status, global, globalbus)
1002 012221                end;
1003 012245           for intrabus := 1 to maxintrabus do
1004 012270                with intraq[intrabus] do begin
1005 012306                     if head <> nil then
1006 012312                          queueserver(head, tail, status, intra, intrabus)
1007 012326                end;
1008 012352           for interbus :=1 to maxinterbus do
1009 012375                with interq[interbus] do begin
1010 012413                     if head <> nil then
1011 012417                          queueserver(head, tail, status, inter, interbus)
1012 012433                end;
1013 012457           for iobus := 1 to maxiobus do
1014 012502                with iobusq[iobus] do begin
1015 012520                     if head <> nil then
1016 012524                          queueserver(head, tail, status, ioline, iobus)
1017 012540                end
1018 012554      end; { servicequeues }
1019
1020
1021
1022             procedure cellstatus;
1023             var
1024                  module:   integer;
1025                  cell:     integer;
1026                  totacti:  integer;
1027                  totidle:  integer;
1028                  totwait:  integer;
1029                  totcomm:  integer;
1030
1031 012565      begin
1032 012605           totacti := 0;
1033 012610           totidle := 0;
1034 012613           totwait := 0;
1035 012616           totcomm := 0;
1036 012621           for module := 1 to maxmodules do
1037 012644                for cell := 1 to maxcells do
1038 012667                     with computer[module][cell] do
1039 012726                         case status of
1040 012727                              0:
1041 012727                                   begin
```

```
1042 012727                          totidle := totidle + 1
1043 012727                       end:
1044 012736                    1:
1045 012736                       begin
1046 012736                          totacti := totacti + 1
1047 012736                       end:
1048 012745                    2:
1049 012745                       begin
1050 012745                          totwait := totwait + 1
1051 012745                       end:
1052 012754                    3:
1053 012754                       begin
1054 012754                          totcomm := totcomm + 1
1055 012754                       end
1056 012762                 end:
1057 013027        writeln(snapshot, simtime:1:6, totidle: 5, totacti: 6, totwait: 7, totcomm: 5);
1058
1059 013074        averageacti := averageacti + totacti;
1060 013102        averageidla := averageidle + totidle;
1061 013110        averagewait := averagewait + totwait;
1062 013116        averagecomm := averagecomm + totcomm;
1063 013124    end; ( cellstatus )
1064
1065
1066
1067           procedure cellmanager;
1068           (* this is called every tickperiod *)
1069           var
1070                module: integer;
1071                cell:   integer;
1072
1073 013125    begin
1074 013145         for module :=1 to maxmodules do begin
1075 013170              for cell := 1 to maxcells do begin
1076 013213                   if computer[module][cell].status = 1 then begin
1077 013256                        with computer[module][cell] do begin
1078 013315                             if simtime >= processing then begin
1079 013326                                (* This cell has now finished, therefore set it idle *)
1080
1081
1082 013326                                status := 0;
1083 013331                                prevstatus := 0;
1084 013336                                enterqueue := 0.0;
1085 013344                                exitqueue  := 0.0;
1086 013352                                queue := false;
1087 013357                             end (* if *)
1088 013357                        end (* with *)
1089 013357                   end (* if *)
1090 013357              end (* cell *)
1091 013357         end (* module *)
1092 013367    end; ( cellmanager )
1093
1094
1095
1096           procedure loadsystem;
1097           (* This procedure loads the system with a large number of requests prior *)
1098           (* to startup *)
1099           var
1100                count: integer;
1101
1102 013400    begin
1103 013420         for count := 1 to loadfactor do begin
1104 013447              plotlengths;
1105 013451              setupqueues;
1106 013453              home;
1107 013455              displaytime;
```

```
1108 013457              simtime := simtime + tickperiod
1109 013457          end
1110 013465     end; ( loadsystem )
1111
1112
1113
1114            procedure startsystem;
1115            (* This examines all modules and cells on the system to see if any element *)
1116            (* has failed *)
1117            var
1118                 module: integer;
1119                 cell:   integer;
1120
1121 013476     begin
1122 013516          for module := 1 to maxmodules do begin
1123 013541               if modulefail then
1124 013545                    for cell := 1 to maxcells do
1125 013570                         computer[module][cell].status := 4
1126 013606          end;
1127
1128 013647          for module := 1 to maxmodules do
1129 013672               for cell := 1 to maxcells do begin
1130 013715                    if cellfail then
1131 013721                         computer[module][cell].status := 4
1132 013737               end
1133 013760     end; ( startsystem )
1134
1135
1136
1137            procedure results;
1138            (* Write system totals to files *)
1139            var
1140                 module: integer;
1141                 cell:   integer;
1142                 max:    integer;
1143                 n:      integer;
1144                 meansystimes:      integer;
1145                 meanqtimes:        integer;
1146                 meanglocall:       integer;
1147                 meancelcall:       integer;
1148                 meansencall:       integer;
1149                 meaniocall:        integer;
1150                 meanglobalcall:    integer;
1151                 meanintracall:     integer;
1152                 meanintercall:     integer;
1153
1154 014001     begin
1155 014021          writeln(rsystem);
1156 014030          writeln(rsystem, 'Occurences of cell unavailability ', cellmiss);
1157 014053          writeln(rsystem);
1158 014062          writeln(rsystem, 'Number of bus requests having been served');
1159 014077          writeln(rsystem, '     Removals from queues          ', numofremovals);
1160 014122          writeln(rsystem, '     I/O calls                     ', iocount);
1161 014145          writeln(rsystem, '     Sending message calls         ', sencount);
1162 014170          writeln(rsystem, '     Global OS calls               ', oscount);
1163 014213          writeln(rsystem, '     New cell requests             ', celcount);
1164 014236          writeln(rsystem);
1165 014245          writeln(rsystem, 'Number of bus requests having been made');
1166 014262          writeln(rsystem, '     I/O calls                     ', iorequest);
1167 014305          writeln(rsystem, '     Sending message calls         ', sendrequest);
1168 014330          writeln(rsystem, '     Global OS calls               ', osrequest);
1169 014353          writeln(rsystem, '     New cell requests             ', cellrequest);
1170 014376          writeln(rsystem);
1171 014405          writeln(rsystem, 'Number of bus requests not having been served');
1172 014422          writeln(rsystem, '     I/O calls                     ', iorequest-iocount);
1173 014453          writeln(rsystem, '     Sending message calls         ', sendrequest-sencount);
```

```
1174 014504          writeln(rsystem, '     Global OS calls                 ', osrequest-oscount);
1175 014535          writeln(rsystem, '     New cell requests               ', cellrequest-celcount);
1176
1177
1178 014566     end; { results }
1179
1180
1181
1182
1183
1184
1185                 (*  MAIN PROGRAM STARTS HERE  *)
1186
1187 014567 begin
1188 014721     seed := startseed;
1189 014725     for run := 1 to 1 do
1190 014750          param[run] := run * 100;
1191
1192 015007     for run := 1 to 1 do begin
1193 015032          time := 0;
1194 015035          loadfactor := param[run];
1195 015051          cellmiss := 0;
1196 015054          iocount := 0;
1197 015057          sencount := 0;
1198 015062          oscount := 0;
1199 015065          celcount := 0;
1200 015070          iorequest := 0;
1201 015073          sendrequest := 0;
1202 015076          osrequest := 0;
1203 015101          cellrequest := 0;
1204 015104          ioqcount := 0;
1205 015107          numofremovals:=0;
1206 015112          globalmem := maxmemory;
1207 015116          simtime := 0.00000000;
1208 015122          (* Reset system clock *)
1209 015122          openfiles(run);
1210 015126          (* Clear old files *)
1211
1212 015126          for globalbus:=1 to maxglobalbus do begin
1213 015151               with globalq[globalbus] do begin
1214 015167                     head := nil;
1215 015172                     tail := nil;
1216 015177                     status := idle
1217 015177               end
1218 015204          end;
1219
1220 015214          for intrabus:=1 to maxintrabus do begin
1221 015237               with intraq[intrabus] do begin
1222 015255                     head := nil;
1223 015260                     tail := nil;
1224 015265                     status := idle
1225 015265               end
1226 015272          end;
1227
1228 015302          for interbus:=1 to maxinterbus do begin
1229 015325               with interq[interbus] do begin
1230 015343                     head := nil;
1231 015346                     tail := nil;
1232 015353                     status := idle
1233 015353               end
1234 015360          end;
1235
1236 015370          for iobus := 1 to maxiobus do begin
1237 015413               with iobusq[iobus] do begin
1238 015431                     head := nil;
1239 015434                     tail := nil;
```

```
1240 015441                     status := idle
1241 015441                end
1242 015446          end;
1243
1244 015456          for module:=1 to maxmodules do
1245 015501             for cell:=1 to maxcells do
1246 015524                with computer[module][cell] do begin
1247 015563                     status := 0;
1248 015566                     prevstatus := 0;
1249 015573                     globalcall := 0;
1250 015600                     intracall := 0;
1251 015605                     intercall := 0;
1252 015612                     systimes := 0;
1253 015617                     qtimes := 0;
1254 015624                     glocall := 0;
1255 015631                     celcall := 0;
1256 015636                     sencall := 0;
1257 015643                     iocall := 0;
1258 015650                     enterqueue := 0.0;
1259 015656                     exitqueue  := 0.0;
1260 015664                     memreqsize := 0;
1261 015671                     queue := false
1262 015671                end;
1263
1264 015716          clear ;
1265 015720          writeln;
1266 015727          writeln;
1267 015736          writeln;
1268 015745          writeln;
1269 015754          printheader;
1270 015756          home;
1271 015760          displaytime;
1272
1273 015762          for module := 1 to maxmodules do begin
1274 016005              avsystimes[module] := 0;
1275 016020              avqtimes [module] := 0;
1276 016033              avglocall[module] := 0;
1277 016046              avcelcall[module] := 0;
1278 016061              avsencall[module] := 0;
1279 016074              aviocall [module] := 0;
1280 016107              avcpujobs[module] := 0;
1281 016122              avglobalcall[module] := 0;
1282 016135              avintracall [module] := 0;
1283 016150              avintercall [module] := 0;
1284 016163          end;
1285
1286 016173          averageacti := 0;
1287 016176          averageidle := 0;
1288 016201          averagewait := 0;
1289 016204          averagecomm := 0;
1290 016207          snapcount := 1;
1291 016213          morerequests := true;
1292 016215          loadsystem;
1293 016217          (* Start up machine *)
1294 016217          home;
1295 016221          displaytime;
1296 016223          cellstatus;
1297 016225          snap := 0;
1298
1299 016230          while simtime <= maxsimtime do begin
1300 016236               servicequeues;
1301 016240               plotlengths;
1302 016242               cellmanager;
1303 016244               if snap = 25 then begin
1304 016254                  cellstatus;
1305 016256                  snap := 0;
```

```
1306 016261                 snapcount := snapcount + 1;
1307 016267                 home;
1308 016271                 displaytime;
1309 016273              end;
1310 016273              snap := snap + 1;
1311 016301              simtime := simtime + tickperiod;
1312 016307              setupqueues
1313 016307          end;
1314
1315 016312          results;
1316 016314          averageidle := trunc( (averageidle / snapcount) + 0.5 );
1317 016341          averageacti := trunc( (averageacti / snapcount) + 0.5 );
1318 016366          averagewait := trunc( (averagewait / snapcount) + 0.5 );
1319 016413          averagecomm := trunc( (averagecomm / snapcount) + 0.5 );
1320 016440          writeln(snapshot);
1321 016447          writeln(snapshot, ' average', averageidle:5, averageacti:6, averagewait:7, averagecomm:5);
1322
1323 016514          mill(time);
1324 016520          writeln(rsystem);
1325 016527          writeln(rsystem, 'Time used is', time:10, ' milliseconds');
1326 016560          for snap := 1 to 17 do
1327 016603              writeln;
1328 016622          writeln('time used =', time:10);
1329 016645          writeln('numofremovals =', numofremovals:6)
1330 016670      end
1331 016670 end. ( simulator )

COMPILATION COMPLETE : 0 ERRORS REPORTED
```