71-9683

STRECKER, William Daniel, 1944-
AN ANALYSIS OF THE INSTRUCTION EXECUTION RATE
IN CERTAIN COMPUTER STRUCTURES.

Carnegie-Mellon University, Ph.D., 1970
Engineering, electrical

University Microfilms, A XEROX Company, Ann Arbor, Michigan

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Carnegie-Mellon University

CARNEGIE INSTITUTE OF TECHNOLOGY

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF Doctor of Philosophy

TITLE An Analysis of the Instruction Execution Rate in Certain Computer Structures

PRESENTED BY William Daniel Strecker

ACCEPTED BY THE DEPARTMENT OF Electrical Engineering

C. Gordon Bell — 28 July 1970
MAJOR PROFESSOR — DATE

A. G. Jordan — 28 July 1970
DEPARTMENT HEAD — DATE

APPROVED BY THE COMMITTEE ON GRADUATE DEGREES

[signature] — 9/a/70
CHAIRMAN — DATE

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Carnegie-Mellon University

An Analysis of the Instruction Execution Rate in Certain Computer Structures

William Daniel Strecker

Abstract:

The purpose of the thesis is to present a series of models of digital computers at the level of the memory processor interface. A discussion of computer instructions is presented and the single address format is taken as the prototype instruction. The execution rate for instructions of this type is then determined for several computer structures of the single processor and general multiprocessor types. The effect on the execution rate of a specialized processing activity, input/output handling, is considered. Analytic models relate the instruction execution rate to the memory and processor speeds, their number, and their interconnection. Simulation studies serve to verify the results of the analysis. A simple automatic design program is proposed which optimally configures computer structures from a set of available components.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Acknowledgement

I would like to express my appreciation to Professor Gordon Bell for his guidance during the course of my research and to my wife, Carole, who greatly assited in the mechanical preparation of the thesis.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table of Contents

	Page
Abstract	ii
Acknowledgement	iii
Figures	vi
Tables	vii
Chapter I Introduction	1
A. Computer Modelling	1
B. Computer Analysis	2
C. Computer Synthesis	5
Chapter II Computer Components and Instructions	8
A. Memories	8
B. Processors	11
C. The Computer Instruction	13
D. Instruction Timing Diagram	14
E. The Instruction Execution Rate	17
Chapter III Single Processor Computers	19
A. Single Memory	19
B. Interleaved Memory	21
C. Interleaved Memory--Alternative Analysis	24
D. Instruction Buffering	27
E. Instruction Prefetch	31
Chapter IV Multiprocessor Computers	35
A. The Multiprocessor Problem	35
B. Modified Instruction Format	38
C. Multiprocessor with $t_p = t_w$	41

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

D. Multiprocessor with $t_p < t_w$ 46

E. Multiprocessor with $t_p > t_w$ 49

Chapter V Simulation 54

A. Reasons for Simulation 54

B. The Simulator 54

C. Results 54

D. A Comparison with Other Simulation Results 59

Chapter VI An Analysis of I/O Effects on Processor Performance 69

A. I/O Activity 69

B. Simple I/O Handling 72

C. Dynamic I/O Handling 74

D. Example 77

Chapter VII Computer Design 80

A. Optimization Approach 80

B. Costs and Problem Formulation 80

C. Example 1: Minimization of System Cost 84

D. Example 2: Minimization of Cost/Performance Ratio 87

Chapter VIII Conclusion 91

A. Summary 91

B. Extentions of the Models 92

C. A Proposal for Continued Work 94

References 96

Bibliography 98

Appendix 99

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figures

		Page
1.	General Computer Configuration	4
2.	Synthesis Procedure	7
3.	Instruction Timing Diagram	15
4.	Instruction Execution Tree	26
5.	Prefetch Execution Tree	34
6.	Simulator	56
7.	Results for $t_p = t_w$	61
8.	Results for $t_p = 0.1t_c$	62
9.	Results for $t_p = 0.2t_c$	63
10.	Results for $t_p = 2t_c$	64
11.	Results for Varying t_c	65
12.	A Comparison with Rosenfeld's Results	68

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Tables

		Page
1.	Example 1 Costs	85
2.	Example 1 Designs	86
3.	PDP-10 Costs	88
4.	PDP-10 Designs	90

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter I Introduction

A. Computer Modelling

The purpose of this thesis is to present a series of analytic models of digital computers. The models relate the performance of the computer as measured by the rate of instruction execution to the specifications of its major high level components, their number, and their interconnection. The main components considered are memories characterized by their cycle and access times and processors characterized by the times required to perform each of their operations. (A detailed consideration of the computer components is given in chapter II.)

There are two reasons for doing the modelling. The first is to gain a quantitative understanding of those factors which govern the performance of digital computers - analysis. The second is to assist in the design of digital computers - synthesis.

A review of the computer literature indicates that computer modelling at the level of the memory processor interface has been neglected.[1] The probable reason for the neglect is the mathematical difficulties associated with analytic solutions of suitable models. A major contribution

[1] There has been some analysis; these earlier results are discussed in the relevent chapters of the thesis.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of the thesis is the development of reasonably simple approximate solution methods for the models proposed. As is indicated in chapter V, simulation studies suggest that the approximate solutions are quite satisfactory.

B. Computer Analysis

The digital computer is an information processing device and an appropriate measure of its performance is the rate at which the information is processed. The primitive computer activity (as opposed to computer component activity) is the execution of an instruction. If we assume certain things constant over a class of computer structures to be analyzed - specifically, the instruction set and the memory word size - then the performance of the computers can be taken as equal to the IER where the IER is the instruction execution rate. The IER mainly depends on two factors: component speed and concurrency. The execution of an instruction (A discussion of computer instructions appears in chapter II.) involves a sequence of operations by a memory and a processor. The IER is determined not only by how fast these operations are carried out but also by the number of operations being carried out simultaneously. (As a practical matter the subject of concurrency is a rather important one. Technology imposes, at any given time, limits on how fast basic operations can take place and the only remaining factor that can be used to increase performance

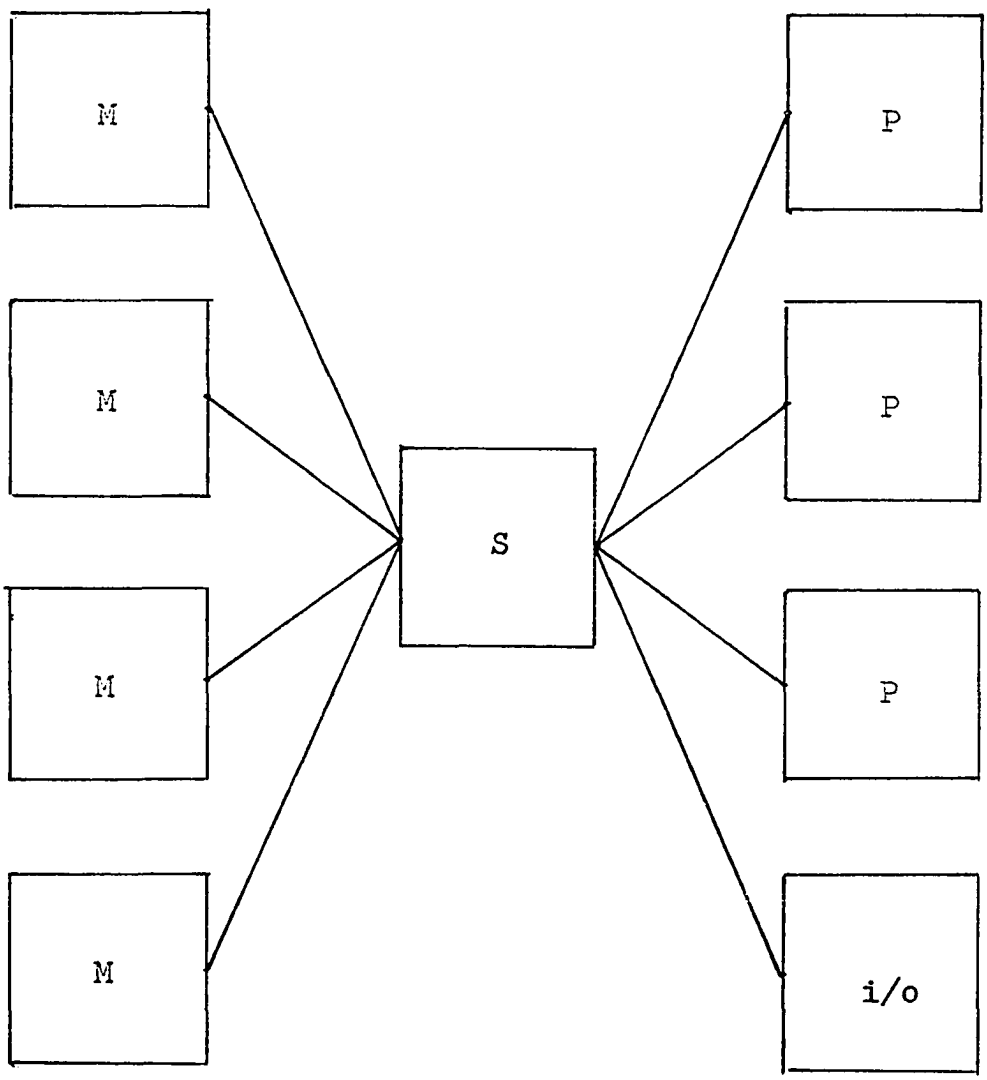
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is concurrency and contemporary large computers such as the CDC 6600 (Thornton, 1970) and IBM 360/91 (Anderson, et al., 1967) use concurrency in several parts of the computer.)

Since the operations comprising a single instruction are normally intended to be carried out sequentially, the presence of concurrent operations implies that for at least some of the time more than one instruction is in the process of being executed. The multiple instructions usually appear in either of two ways: in a multiprocessor computer (with multiple instruction streams being simultaneously executed) or in a single processor computer simultaneously executing successive instructions of a single instruction stream.

The general class of computer structures is indicated diagramatically in figure 1. A group of memories (each indicated by an M) is connected through a switch (S) to a group of processors (P). The memories are also connected through the switch to a specialized processor, an input/output channel (i/o), which is characterized by a constant memory access rate. In the general case each of the memories and each of the processors can be different and the extent to which any given memory is used by any given processor can be specified independently. For the analysis in the thesis we consider several special structures of this general class. We assume that all the memories and all the processors are alike and we assume that there is an equal likelihood that any given memory is used by any given pro-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



Legend:
M-- memory
P-- processor
S-- switch
i/o -- i/o channel

Figure 1. General Computer Configuration

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

cessor. We divide the structures into two classes: single processors (a single P on figure 1) and multiprocessors and provide a different type of analysis for each. For the single processors (analyzed in chapter III) we introduce a notation called an instruction timing diagram and with the aid of it directly compute the instruction execution time and then the IER. The basic approach taken is to add the average delay in accessing memory to the processor time to get the total instruction execution time. This approach, while both straightforward and conveniently used to examine processor features in detail, is very awkward to apply to multiprocessor computers, and another approach is indicated. For multiprocessors (analyzed without i/o in chapter IV and with i/o in chapter VI) we introduce a special instruction form called a unit instruction which allows us to determine the IER directly in terms of the rate of memory cycle utilization. The utilization is determined by an approach related to the occupancy problem of combinatorial analysis.

C. Computer Synthesis

The models developed in the thesis relate the performance of the computer to the number, specifications, and interconnection of its components. These variables are also those to which the cost of the computer is related. If both cost and performance are related to common variables, it is possible to formulate a design procedure that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

will choose computer configurations that are optimum with respect to certain design criteria.

Synthesis procedures are usually iterative as indicated in figure 2. A series of potential design configurations are generated, analyzed, and tested against the design criteria. The best of the configurations (as measured against the design criteria) is chosen. The heart of the synthesis procedure is the analysis part. The generation of the proposed configurations can be either simple in that all configurations (in some prespecified design space) are generated or more complex in that the configurations generated are dependent on the results of analysis and testing of earlier configurations.

The purpose of this thesis is not to consider design procedures in detail. However, to illustrate the utility of the thesis analysis in design, a simple design program is presented in chapter VII. The program chooses a configuration (in other words, it picks the number of memories and processors and their speeds) so as to realize a desired IER at a minimum cost.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

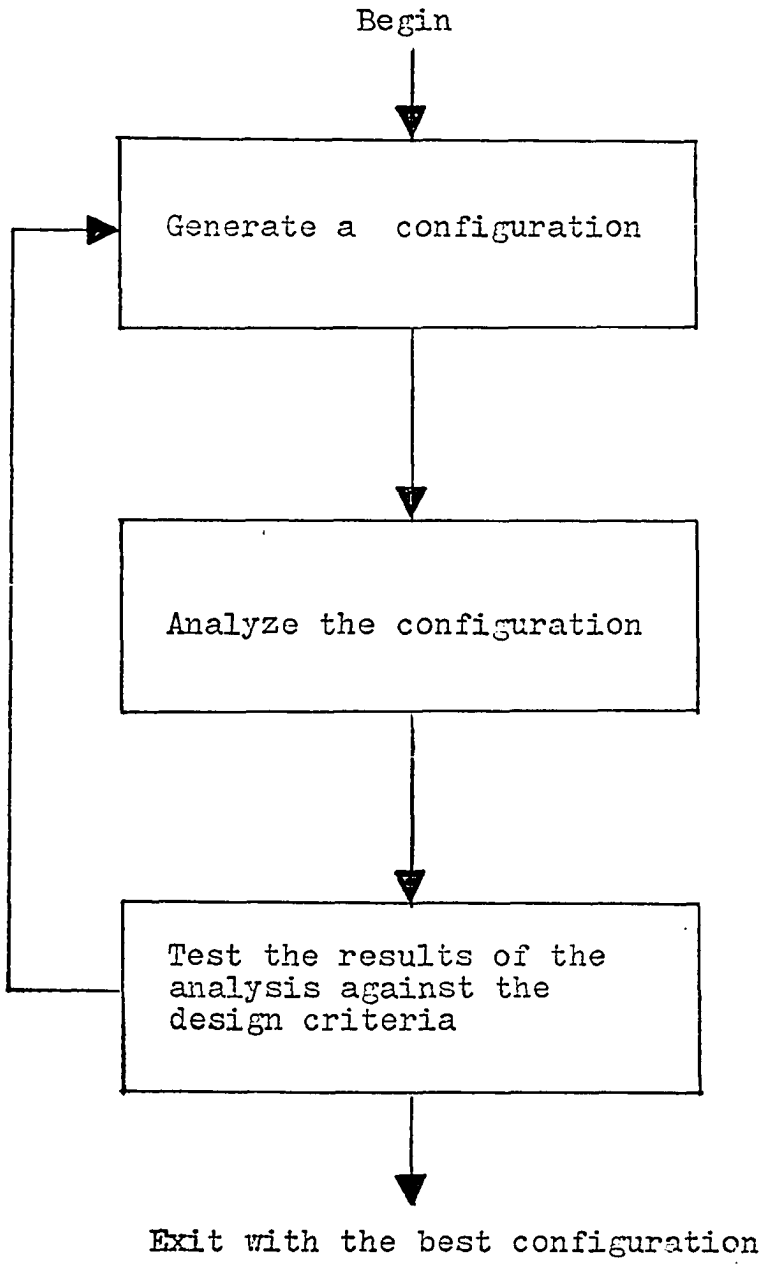


Figure 2. Synthesis Procedure

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter II Computer Components and Instructions

A. Memories

The structure and organization of the digital computer is influenced by both what we think the computer should be and by the technology of the computer components. Probably memory technology has had more influence, historically, on computer design than any other factor.

The purpose of the memory is to hold programs (sets of instructions) and the data (sets of operands) to be processed. The time required to obtain information from memory thus strongly influences (limits) the instruction execution rate. For economic reasons it is generally impossible to provide, in a single memory, both sufficient speed of operation to realize an adequate IER and still have adequate memory size (number of memory words) to hold all the programs and data associated with a computer system. At times even a single program and its related data may face this limitation. Consequently, it is conventional that at least two forms of memory be present in a computer system: primary and secondary.

The primary memory is a relatively small but fast storage area for instructions and operands that can be directly operated on by the processor. The information

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

stored in the secondary memory cannot be acted on directly by the processor; it must first be transferred to primary memory. The transfer of information between primary and secondary memories is an important activity in digital computer systems, both because it interferes with processor access to primary memory and because there is usually a large access time associated with secondary memories.

Because of the general difficulty (or impossibility) of predicting the processor accessing pattern, an important requirement for primary memories is a random access characteristic. For such memories, the time required to access any word of information is independent of its location in memory; in particular it is independent of the relation between the location referenced and the last referenced location.

At the time of writing the most common form of primary memory is the magnetic core type. Magnetic core memories typically used in computers have word sizes from eight to over 100 bits (possibly 500 bits) and have total word capacities from about 4K (K = 1024) to 64K. (The sizes given are typical for single memory units. The entire computer primary memory may be made up from a number of memory units.) Core memories have complete operation or cycle times in the range of 0.5 usec. to 10 usec. The readout of information in a magnetic core is inherently a destructive process: that is, the contents of the memory lo-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

cation are lost when read. Since this is usually undesired, the information must also be restored in the memory after being read. This re-writing takes an additional amount of time. Once the information is read out, however, it is immediately available to a processor; the latter need not wait for the restore time. The time elapsing between the initiation of a memory request and the time the information becomes available is the access time; it is typically 30% to 50% of the cycle time.

The cost of magnetic core memories is generally related to the cycle time and a very rough approximation would give the cost proportional to the reciprocal of the cycle time. Core memories are frequently of the coincident current type and in these the cost of the electronic part of the memory is roughly proportional to the square root of the memory size. The balance of the memory cost is directly proportional to the memory size. Thus, the cost per word is lower in large memories than in small; in particular a memory of w words costs less than m memories of w/m words.

Another form of primary memory is the transistor register type. These memories are characterized by very fast access times of about 25 to 100 nsec. Their cost, though, is such as to preclude their general use as primary memory. It is common, however, for contemporary computers to provide a small amount (typically 8 to 64

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

words) of primary memory in registers. The registers are often addressable as if they were locations in core memory.

The secondary memories store the large bulk of information in the digital computer system. The principal types of secondary memories are rotating discs and drums and linear magnetic tapes. The cost of storage per bit in secondary memories is about 1% to 10% of that in primary memory. The low cost and large capacity of secondary memories is due to the fact that they are not of the random access type. The access time is dependent on the relation between the last and currently accessed data and the time elapsed since that last access. The average access time for randomly located information is half the time for a revolution (about 10 msec.) in rotating memories and the time to search half the tape (a number of seconds) in magnetic tape memories. The maximum rate of information flow in a random access memory is the reciprocal of the cycle time; in a non-random access memory the maximum is obviously not the reciprocal of the average access time. For the type of highly structured information flows that take place between the primary and secondary memories, the word flow rate (particularly from drums) may appraoch that obtainable from primary memory.

B. Processors

The purpose of the processor is to obtain instructions and operands from memory, decode the instructions, and perform the required operation on the operands. Often

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the speed at which arithmetic operations are carried out strongly influences the IER, and it is fairly conventional to characterize processors by their arithmetic capabilities. There are two principal types of arithmetic operands: fixed point (integer) and floating point (fraction plus exponent). Small to medium size computers usually have only fixed point arithmetic operations built in; floating point operations are programmed. The smaller computers may have only fixed point add and subtract and even fixed point multiply and divide must be programmed. For the purpose of discussing processor times, instruction decoding and other types of operations implemented such as logical and control operations can be grouped with the fixed point add and subtract instructions. At the present time these types of operations generally require on the order of a few tens of nsec. to a few hundred nsec. A fixed point multiply or divide takes from a few hundred nsec. to several usec. depending on the processor. Floating point operations, when implemented, have execution times in the range of about 0.4 to 15 usec. The important relations in determining IER are, as we shall see later, between the processor times and the memory restore times. From the preceeding discussion we surmise that for most contemporary computers the basic fixed point operations require less time to execute than the memory restore time. The balance of the other operations may, depending on the particular memory and processor, have operation times greater or less than the memory restore time.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

C. The Computer Instruction

We shall take as our basic computer instruction the implementation of a binary (two operand) operation. A general binary operation can be represented as:

result ← operand 1 [operator] operand 2.

In a computer we refer to information by its location in memory (specified by a memory address) and consequently the prototype instruction is written:

1. (A) ← (B) [operator] (C)
2. Take (D) as the next instruction.

The notation (X) means the contents of the memory location specified by address X. Thus B and C are the operand addresses; A is the result address. The second part of the instruction is necessary because instructions are executed as part of a sequence; the location of the next instruction of the sequence is specified by address D.

As we can see there are four memory addresses (A, B, C, D) associated with the prototype instruction. These addresses must be specified somehow, but they need not be explicitly included in the instruction; they can be specified in an implicit manner. The reasons for preferring implicit specification of addresses are due to : (1) a potential improvement in the IER by reducing the number of memory references which must be made to execute the instruction and (2) a reduced amount of memory space required to hold the instruction. (If it requires k bits to encode a memory

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

address and j bits to encode the instruction operation, then an instruction with n explicit addresses requires n k + j bits of memory to hold it.)

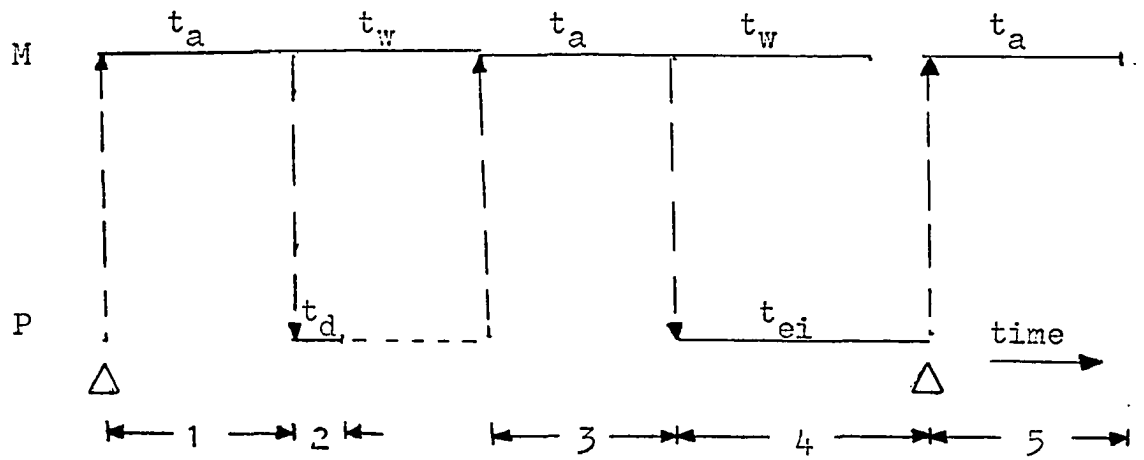
In real computers a common implementation of the basic instruction is one called the single address format. A register called the accumulator is implicitly specified as both the location of one of the operands and the location of the result. The next instruction address is implicitly taken as the address of the current instruction plus one. The single address format requires two memory references: one for the instruction itself and one for the operand. The single address format is used as a prototype instruction for the purposes of the subsequent analysis and the computer instruction set is assumed to be made up entirely of this type of instruction. In addition, each instruction and each operand is assumed to occupy exactly one memory word.

D. Instruction Timing Diagram

The execution of an instruction of the single address format involves the following steps:

1. The instruction itself is fetched from the memory. The memory address of the instruction is specified implicitly by the address of the previous instruction plus one.
2. The instruction once received from memory is decoded yielding the operation to be performed and the address of the operand to be used.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



Legend:

1 instruction fetch

2 instruction decoding

3 operand fetch

4 instruction execution

5 next instruction fetch

t_a memory access time

t_w memory restore time

t_d instruction decode time

t_{ei} processor execution time

Figure 3. Instruction Timing Diagram

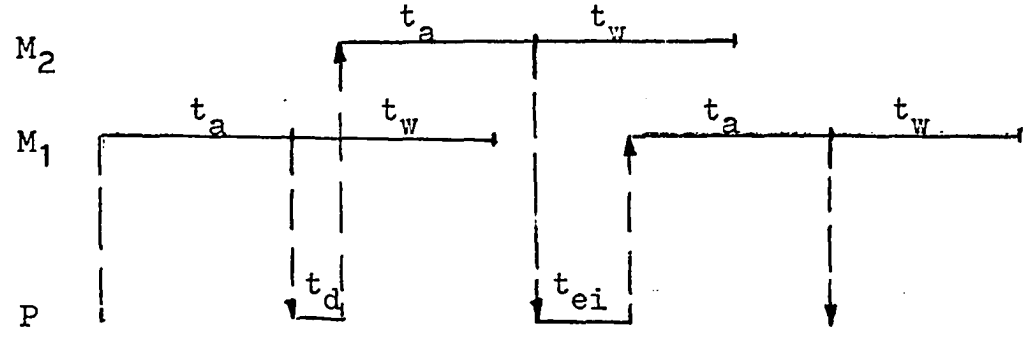
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3. The operand is fetched from memory.

4. Once the operand is received by the processor the operation is performed.

This sequence of events may be indicated diagramatically by the plotting simultaneously against time the memory and processor activities. See figure 3. This construction is termed an instruction timing diagram (ITD). The ITD will be used to visualize the instruction execution and (as will be seen in chapter III) to compute the amount of time required to execute the instruction. The general approach to be taken in computing instruction execution times from the ITD is to pick corresponding points on successive instructions and determine the time between them. (Such points are indicated by Δ's on the ITD of figure 3). Sometimes the appropriate choice of corresponding points facilitates the determination of the instruction execution time.

If there is more than just one memory capable of simultaneous operation the ITD is easily extended to handle this case. In the following we have two memories M_1 and M_2; the instruction reference goes to M_1 and the operand reference to M_2.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The interesting thing to note here is that once the instruction is decoded the operand reference can be immediately made to M_2. The time required to execute the instruction (assuming $t_d < t_w$) is shortened; this is a result of the concurrent operation for a time of both memories. Further discussion of this is deferred until chapter III.

E. The Instruction Execution Rate

The computer instruction set is assumed to consist of a set of instructions I_i of the single address instruction format each with an associated value of processor execution time t_{ei}. The value of decode time t_d is assumed to be constant for all instructions. By examining the ITD we can determine a value t_i to execute I_i where t_i is clearly a function of t_{ei}, t_d, t_a, t_w. In section D we saw that the number of memories influenced the instruction execution time; hence t_i is also a function of the memory structure. For the time being let us simply associate the memory structure with a variable S. Then we can write:

$$t_i = h(t_{ei}, t_d, t_a, t_w, S). \qquad (2.1)$$

Associated with each instruction I_i is a probability or relative frequency f_i which gives the likelihood that any given instruction is of type i. The average time E(t) required to execute an instruction is computed:

$$E(t) = \sum_i f_i t_i. \qquad (2.2)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The instruction execution rate is the reciprocal of the average execution time; hence:

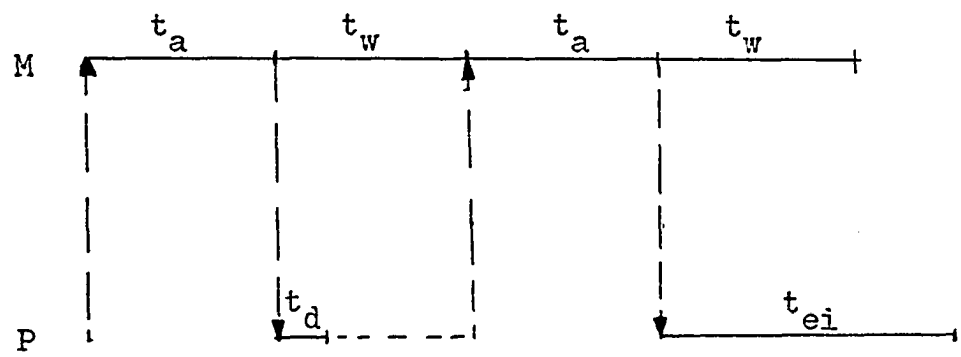
$$IER = 1/(\sum_{i} f_i t_i). \qquad (2.3)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter III Single Processor Computers

A. Single Memory

Given that we have a single processor organization and a set of instructions in the single address format with relative frequencies f_i and processor execution times t_{ei}, we now wish to compute the IER. When there is a single memory the ITD is:



Since in chapter II we indicated that for most practical situations $t_d < t_w$, the important timing relationship is that between t_{ei} and t_w. The instructions fall naturally into two classes:

class 1 - $t_{ei} \leq t_w$

class 2 - $t_{ei} > t_w$.

For instructions of class 1, the execution time for instruction I_i is:

$$t_i = 2t_a + 2t_w = 2t_c \qquad (3.1)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

where $t_c = t_a + t_w$ is the memory cycle time. For instructions of class 2, the execution time is

$$t_i = 2t_a + t_w + t_{ei} . \tag{3.2}$$

The average execution time for an instruction is defined as in chapter II:

$$E(t) = \sum_i f_i t_i$$

$$= \sum_{i \in c1} f_i (2t_c) + \sum_{i \in c2} f_i (2t_a + t_w + t_{ei}) . \tag{3.3}$$

where $i \in c1$ and $i \in c2$ mean those subscripts which apply to classes one and two respectively. Now let us make the following definitions:

$$f1 = \sum_{i \in c1} f_i \tag{3.4a}$$

$$f2 = \sum_{i \in c2} f_i \tag{3.4b}$$

$$t_e1 = (1/f1) \sum_{i \in c1} f_i t_{ei} \tag{3.4c}$$

$$t_e2 = (1/f2) \sum_{i \in c2} f_i t_{ei} . \tag{3.4d}$$

From these definitions, f1 and f2 are the relative frequencies of all instructions of class 1 and class 2 respectively. Similarly, t_e1 and t_e2 are the average respective processor execution times for instructions of class 1 and class 2. Substituting equations 3.4 in equation 3.3 gives:

$$E(t) = f1(2t_c) + f2(2t_a + t_w + t_e2) \tag{3.5}$$

The form of the result suggests a simple way to compute E(t). The average execution times (t_e1 and t_e2) are substituted

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

for t_{ei} in the general expressions for the instruction execution times given by equations 3.1 and 3.2 and then the resulting values are weighted by the respective class relative frequencies. This approach works because of the linearity of the averaging operation and applies not only to the results of this section but also to those of the subsequent sections as well. Hence we can disregard the detail of the instruction set and in the subsequent analysis perform only two computations analogous to those represented by equations 3.1 and 3.2. The instruction execution rate is obtained by taking the reciprocal of E(t) as defined by equation 3.5:

$$IER = 1/(f1(2t_c) + f2(2t_a + t_w + t_e2)). \quad (3.6)$$

Since f1 + f2 = 1 and $t_c = t_w + t_a$ the latter becomes:

$$IER = 1/(2t_c + f2(t_e2 - t_w)). \quad (3.7)$$

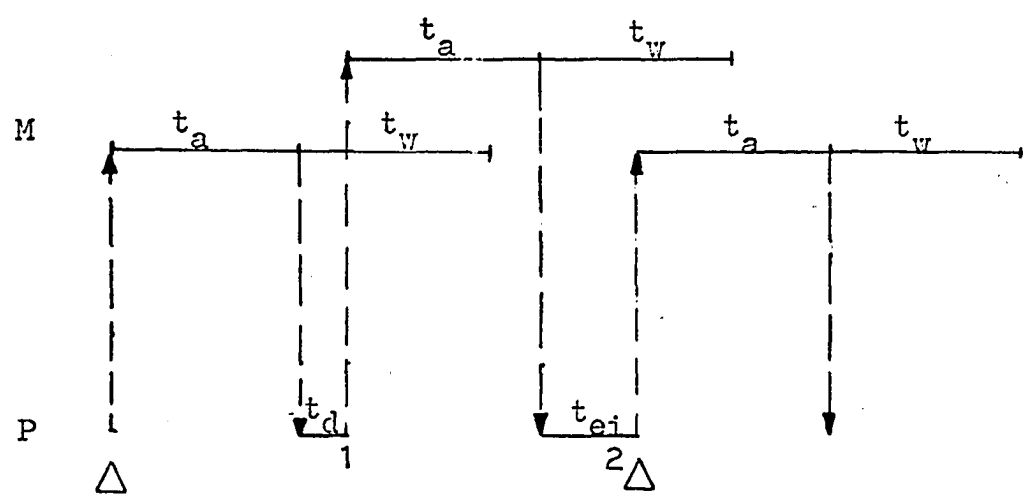
B. Interleaved Memory

We can see from equation 3.7 that even making the processor arbitrarily fast (which makes f2 = 0) cannot provide an IER greater than $1/2t_c$. The IER could be increased if it were possible to have the instruction and its operand in different memories. This cannot be done with certainty without greatly reducing the memory utility[1],

[1] The obvious way to do this is to have separate memories for the instructions and the operands. This approach eliminates the (little used) generality to use operands as instructions and conversely. More importantly it eliminates the ability to apportion freely the memory between operands and instructions as the need arises.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

but we can propose a memory organization that achieves this with a high probability. Suppose we have m independent memories. We arrange the memory addressing so that successive addresses are in different memories. In particular if the memories are denoted M_0 ,..., M_{m-1} and address 0 is in M_0 then address z is in $M_{(z \text{ modulo } m)}$. Such an addressing scheme tends to uniformly distribute the operands and the instructions among all the memories regardless of their particular addresses. We can then make the reasonable assumption that the probability of any particular memory reference being directed to any particular memory is 1/m . Equivalently, if a memory reference goes to M_j the probability that the succeeding reference also goes to M_j is 1/m . The probability that reference does not go to M_j is 1 - 1/m . This type of memory organization is called an interleaved memory. The ITD for the interleaved memory case with no addressing conflicts is:



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

We note that the value of t_i is no longer deterministic; rather t_i is a random variable. In the following we compute the average value of t_i ; however for simplicity there is no new notation introduced to indicate that it is an average value. We recall that the general idea in using the ITD to compute instruction timing is to pick corresponding points on successive instructions and then determine the time elapsed between the points. On the above diagram the points used are indicated by Δ's. The potential conflicts are indicated by the numbers 1 and 2 on the ITD. The time elapsed between the first Δ and point 1 is $t_a + t_d$. With probability 1/m a delay of $t_w - t_d$ is encountered at this point. From point 1 to point 2 a time $t_a + t_{ei}$ elapses. With probability 1/m a further delay of $t_w - t_{ei}$ is encountered before the next instruction can begin. Hence:

$$\begin{aligned} t_i &= t_a + t_d + (1/m)(t_w - t_d) + t_a + \\ &\qquad t_{ei} + (1/m)(t_w - t_{ei}) \\ &= 2t_a + t_d + t_{ei} + (1/m)(2t_w - t_d - t_{ei}). \end{aligned} \tag{3.8}$$

We have implicitly assumed above that the instructions were of class 1; for instructions of class 2 no delay can be encountered at point 2. In a manner similar to the preceding we find for instructions of class 2 that:

$$t_i = 2t_a + t_d + t_{ei} + (1/m)(t_w - t_d). \tag{3.9}$$

E(t) is now found using the method of section A:

$$\begin{aligned} E(t) = {} & f1(2t_a + t_d + t_e1 + (1/m)(2t_w - t_d - t_e1)) \\ & + f2(2t_a + t_d + t_e2 + (1/m)(t_w - t_d)) \end{aligned}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$= 2t_a + t_d + t_{em} + (1/m)(t_w - t_d) + (f1/m)(t_w - t_e1) \quad (3.10)$$

where t_{em} is the average value of processor execution time for all instructions:

$$t_{em} = f1(t_e1) + f2(t_e2). \quad (3.11)$$

We now compute the IER:

$$IER = 1/(2t_a + t_d + (1/m)(t_w - t_d) + t_{em} + (f1/m)(t_w - t_e1)). \quad (3.12)$$

For a large value of m the IER becomes:

$$IER \cong 1/(2t_a + t_d + t_{em}). \quad (3.13)$$

Hence for an arbitrarily fast processor ($t_d, t_{em} \ll t_a$) the maximum IER is about twice that obtainable with a non-interleaved memory system.

C. Interleaved Memory -- Alternative Analysis

In the last section we considered all memory references to be random with the probability of a reference to a particular memory 1/m. Hence, with this definition, there is a non-zero probability $1/m^2$ that an operand reference conflicts with the instruction reference and that the succeeding instruction reference conflicts with that operand reference. This implies that two successive instructions (which occupy successive memory locations) are located in the same memory. But the interleaving scheme we have suggested generally avoids this. It is of interest then to compute E(t) if these double conflicts were eliminated. (The double conflicts can occur only when $t_{ei} < t_w$ and we shall

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

assume this for the following discussion.) Figure 4 shows a tree structure of possible operation sequences. This is presented to aid in the analysis. The probability of getting from one node of the tree to another is the product of the probabilities of all the branches connecting the nodes. The probabilities of the branches are obtained as follows:

1. The probabilities of branches 1-2 and 1-3 are just those normally associated with the conflict of an operand reference with the preceeding instruction reference. Hence they are 1/m and 1 - 1/m respectively.
2. From the preceeding discussion branch 2-4 represents an impossible situation. Hence the probability of branch 2-4 is zero and the probability of branch 2-5 is one.
3. The probability of branch 3-6 is not (as might be expected) 1/m but rather 1/(m-1). Once an instruction is obtained from a memory, say M_j, and the operand reference is known not to conflict with that of the instruction, the operand must have been chosen from one of the remaining memories not including M_j. Since the next instruction reference is also made to one of these memories, the probability of a conflict is 1/(m - 1). The probability of branch 3-7 is then 1 - 1/(m - 1).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

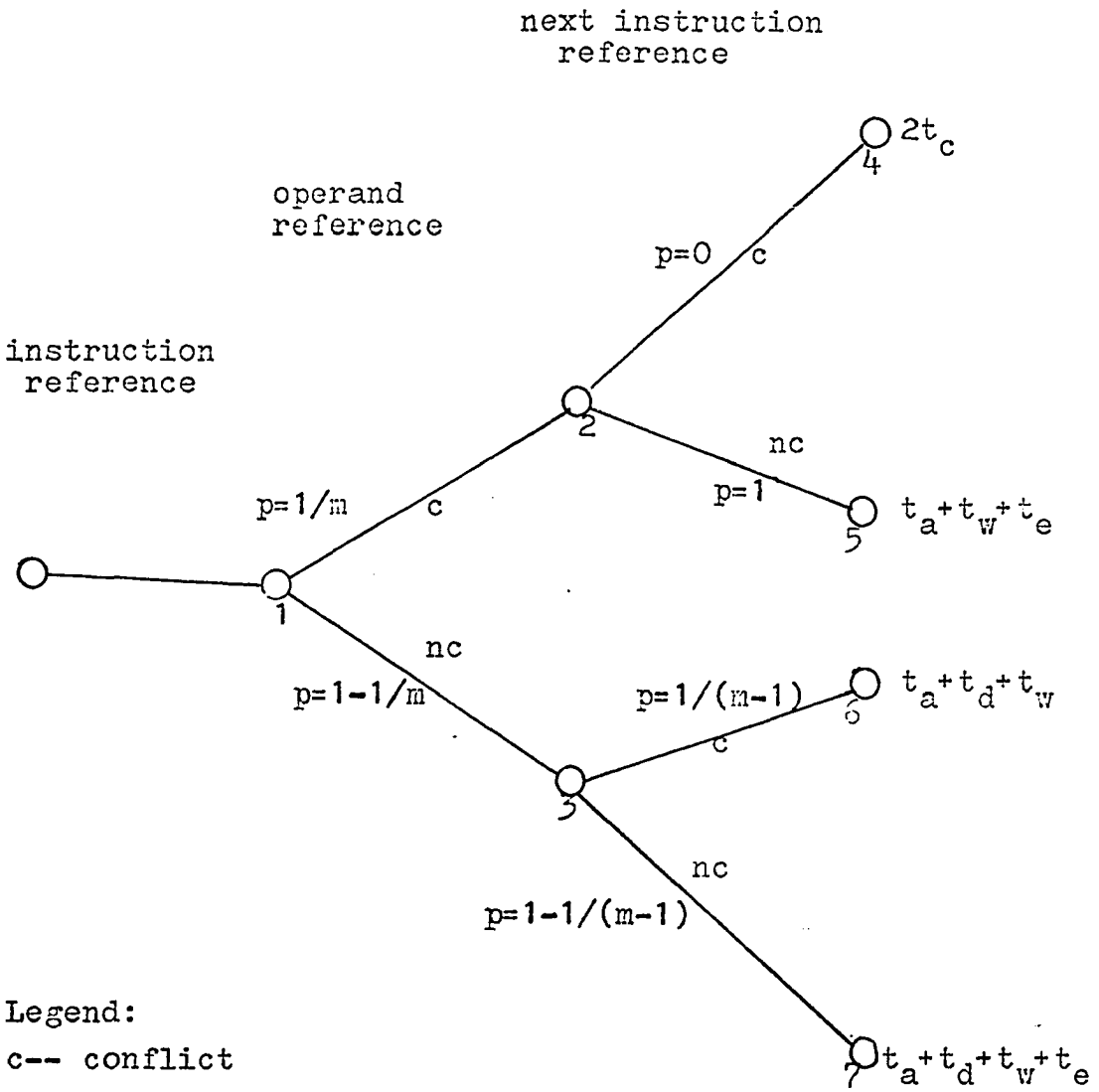


Figure 4. Instruction Execution Tree

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The times next to the terminal nodes indicate the instruction times for the sequence ending at the node; they are readily obtained from the ITD. We now compute t_i:

$$t_i = \sum_{j=4,5,6,7} (\text{time from node 1 to node } j)(\text{probability of getting from node 1 to node } j)$$

$$= (1/m)(t_a + t_w + t_{ei}) + (1 - 1/m)(1/(m-1))(t_a + t_d + t_w) + (1 - 1/m)(1 - 1/(m-1)) \times (t_a + t_d + t_a + t_{ei})$$

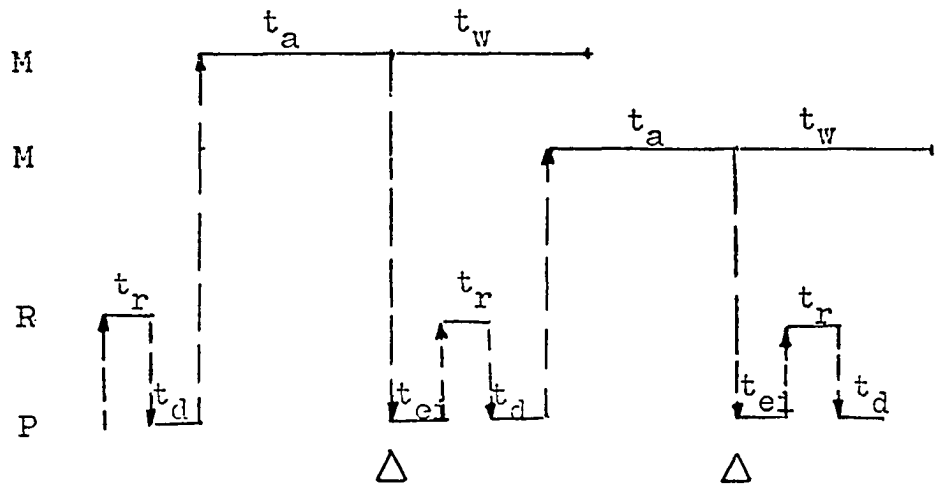
$$= 2t_a + t_d + t_{ei} + (1/m)(2t_w - t_{ei} - t_d). \qquad (3.14)$$

which is exactly the same as is obtained by the previous, simpler analysis.

D. Instruction Buffering

With an m-way interleaved memory it is possible to simultaneously obtain the contents of m successive memory locations. Since successive instructions are normally located in successive memory locations, it is possible to organize the processor to perform the instruction references for m instructions simultaneously (the current instruction reference and the next m - 1 instruction references). Let us assume that the m instructions obtained are stored in m fast processor registers with an access time $t_r \ll t_a$. The instructions are then obtained from the registers as needed. Now up to $s<m$ instructions (why s can be less than m is dicussed shortly) can be executed which have an ITD as follows:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



We see that (except for the first instruction) the fetching and decoding operations of the subsequent instruction can be grouped with the processor execution of the current instruction. This makes it appropriate to redefine (for this section only) classes 1 and 2:

class 1: $t_{ei} + t_r + t_d \leq t_w$

class 2: $t_{ei} + t_r + t_d > t_w$.

Using the approach of section B the values of t_i are readily obtained:

$$t_i = t_a + (t_{ei} + t_r + t_d) + (1/m)(t_w - t_{ei} - t_r - t_d) \qquad (3.15)$$

for instructions of class 1 and

$$t_i = t_a + t_r + t_d + t_{ei} \qquad (3.16)$$

for instructions of class 2. The time required to fetch the m instructions is t_c because the operand reference of the first instruction necessarily conflicts with the instruction fetch. To compute E(t) for a single instruction, we

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

apportion the initial instruction fetch time among the s instructions executed. Hence using equations 3.15 and 3.16 we compute E(t):

$$E(t) = t_c/s + f2(t_a + t_d + t_e2) + f1(t_a + t_e1 + t_d + t_r + (1/m) \times (t_w - t_e1 - t_a - t_d))$$
$$= t_c/s + (t_a + t_d + t_{em} + t_r) + (f1/m)(t_w - t_e1 - t_d - t_r). \quad (3.17)$$

The IER is then computed:

$$IER = 1/(t_c/s + (t_a + t_d + t_{em} + t_r) + (f1/m)(t_w - t_e1 - t_d - t_r)). \quad (3.18)$$

The reason that less than m instructions can be executed is due to the presence of branch instructions in the instruction stream. Such instructions cause the processor to take the next instruction from a memory location that is not the next successive memory location after the branch instruction. Suppose that the relative frequency of branch instructions is f_b. We shall characterize the instruction set by assuming that there is a constant probability $p_b = f_b$ that any given instruction is a branch instruction. Hence there is a probability $1 - p_b$ that an instruction is not a branch instruction. We now compute the probability that a sequence of k instructions are executed. For $1<k<m$ there must be $k - 1$ instructions that are not branches followed by a branch instruction. Let X be a random variable equal to the number of instructions

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

executed. From the preceeding discussion:

$$p(X = k) = p(k) = (1 - p_b)^{k-1} p_b;\ k = 1,\ldots,m - 1; \tag{3.19}$$

where $p(k)$ is the probability of the k instruction sequence. Regardless of whether the m^{th} instruction is a branch instruction or not, the execution sequence terminates at the m^{th} instruction if it has not terminated earlier. Thus:

$$\begin{aligned} p(m) &= 1 - p(X<m) \\ &= 1 - \sum_{k=1}^{m-1} (1-p_b)^{k-1} p_b. \end{aligned} \tag{3.20}$$

Using the fact that the sum of a finite geometric series is:

$$\sum_{k=1}^{n} a^k = (a - a^{n+1})/(1 - a)$$

we reduce equation 3.20 to

$$p(m) = (1 - p_b)^{m-1}. \tag{3.21}$$

The expected number of instructions executed E(X) is then:

$$\begin{aligned} E(X) &= \sum_{k=1}^{m} kp(k) \\ &= \sum_{k=1}^{m-1} k\ p_b(1 - p_b)^{k-1} + m(1 - p_b)^{m-1}. \end{aligned} \tag{3.22}$$

Writing the summation as

$$p_b \frac{d}{d(1-p_b)} \sum_{k=1}^{m-1} (1 - p_b)^k$$

and using the previously mentioned relation for the sum of a finite geometric series, we reduce equation 3.22 to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$E(X) = (1 - (1 - p_b)^m)/p_b \ . \qquad (3.23)$$

The expression for E(X) can now be substituted for s in equation 3.18 yielding the IER for the buffered instruction case:

$$IER = \frac{1}{\frac{p_b t_c}{1 - (1 - p_b)^m} + t_a + t_d + t_r + t_{em} + \frac{f1}{m}(t_w - t_e 1 - t_d - t_r)} \ . \qquad (3.24)$$

For large values of m and small values of t_d and t_r, equation 3.25 becomes:

$$IER \cong 1/(p_b t_c + t_a + t_{em}). \qquad (3.25)$$

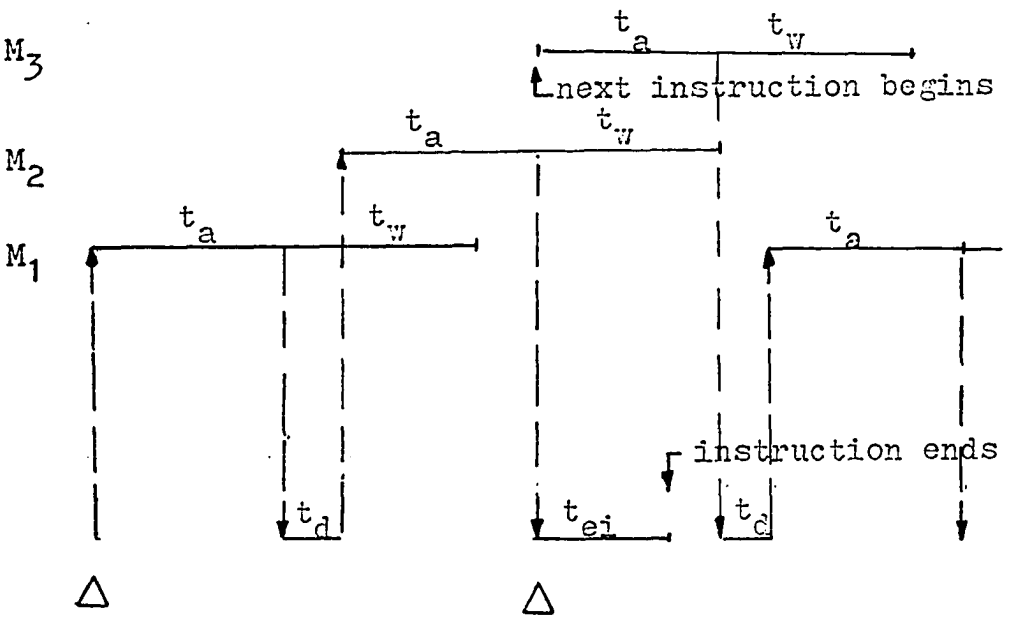
Finally, for a fast processor and a low value of p_b (for scientific computing p_b probably lies in the range of about 0.05 to 0.3) the IER approaches $1/t_a$ which is about twice the IER in the results of section B.

D. Instruction Prefetch

Normally computer instructions are intended to be sequentially executed: in a stream of instructions, the execution of instruction x + 1 does not begin until the execution of instruction x is completed. However, it is possible to organize the processor so that more than one instruction is being executed at a time. It is possible to go to processors of considerable complexity (as for example in the CDC 6600 and the IBM 360/91) in order to maximize the overlapping of instruction execution. In this

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

section, however, we will discuss a modest form of concurrency of instruction execution: the instruction prefetch. The idea of the instruction prefetch is quite simple: the overlapping of the subsequent instruction fetch with the processor execution time of the current instruction. The ITD for this case with no addressing conflicts is:



From the ITD we observe that the value of t_{ei} is not going to appear in the expression for t_i. This perhaps surprising result is a general feature of this type of concurrency. The rate at which instructions are executed is dependent on the time which elapses between instructions commencing execution and not on the time required to execute a given instruction. There are, however, some side effects to be considered. If the value of t_{ei} is such that it extends to overlap the processor execution time of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the next instruction, then multiple processor execution units must be provided. Also there must be checking to insure that the result of the first instruction is not an operand of the subsequent instruction.

Since t_{ci} is not going to appear in the final relations we do not have to consider f_i and t_i and we can compute E(t) directly. Probably the best way to find E(t) is to use the instruction execution tree of section C. The probabilities associated with the branches of the tree are the same; only the instruction execution times have to be changed. The tree is presented in figure 5. We now, using the approach of section C, compute E(t):

$$\begin{aligned} E(t) &= (1/m)(t_w + t_w + t_a) + \\ &\quad (1 - 1/m)(t_a + t_d + t_a + t_w) + \\ &\quad (1 - 1/m)(t_a + t_d + t_a) \\ &= 2t_a + t_d + (1/m)(2t_w + t_a - t_d). \end{aligned} \tag{3.26}$$

The IER can now be computed:

$$IER = 1/(2t_a + t_d + (1/m)(2t_w + t_a - t_d)) \tag{3.27}$$

which for small t_d and large m goes to $1/2t_a$.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

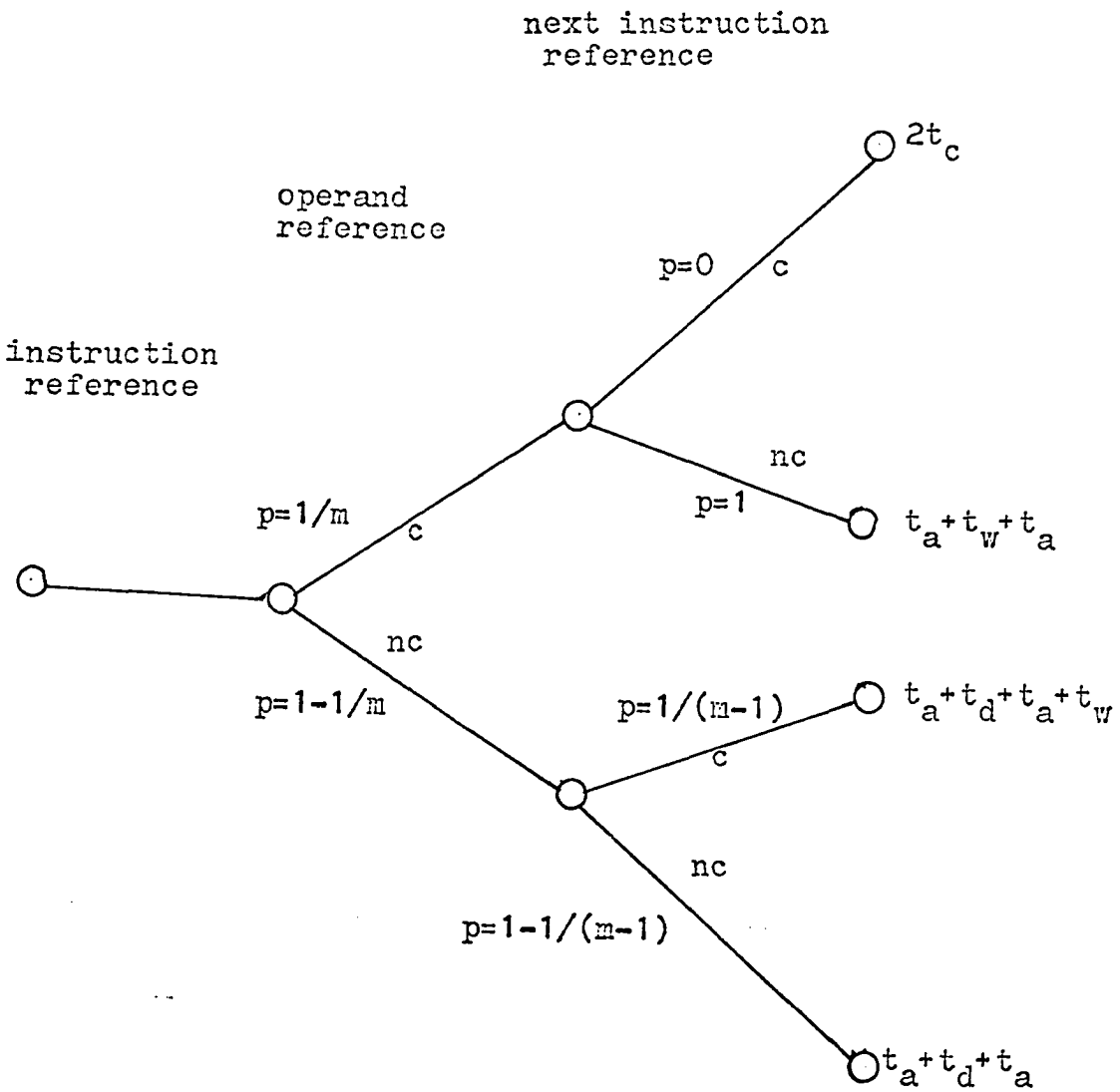


Figure 5. Prefetch Execution Tree

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter IV Multiprocessor Computers

A. The Multiprocessor Problem

One of the main reasons for the anaylsis presented in chapter III is to indicate some limits on the IER obtainable in a single processor computer. To get a higher IER in a single processor computer (single instruction stream) it is probably necessary to have a definite structure in the information to be processed. For example, if the information can be structured as n-component vectors and we organize the processor so as to have n execution units which are capable of performing simultaneous operations on each of the n components (and provide a suitable memory organization), then we can obtain an IER which is about n times that which would be obtained if the data were treated in a scalar form. This is essentially the approach taken in the Illiac IV (Barnes, et al., 1968). As might be expected there are considerable difficulties in realizing an IER that high for many practical problems.

If the information to be processed cannot be so structured, then to get a higher IER, it is necessary to go to a multiprocessor organization (with multiple instruction streams). We should note at this point that it is not the purpose of this thesis to indicate how the multiprocessor is to be used: in particular, how a single, inherently sequential task can be broken down into n tasks that can be

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

run on an n-processor computer. For a discussion of this see Rosenfeld (1969). A typical multiprocessor organization is presented in figure 1. The most important aspect of the multiprocessor organization is the sharing of a common memory system by all the processors. As the processors randomly direct requests to the memories, it is inevitable that conflicts will arise in that a processor will request service from a memory that is busy servicing another processor request. The function of the switch in figure 1 is to direct processor requests to the correct memory and to resolve conflicts by deferring requests to busy memories to subsequent memory cycles. Since we assume that the processor requests to the memories are random, we have what is termed a stochastic service system. The study of such systems is called queueing theory and in queueing theory terminology the multiprocessor system is an m-server system with a finite service requesting population (the n processors). The servers are unique in the sense that they can handle only requests directed specifically toward them. (Usually a m-server system is taken to be one in which any server can service any request.) The memories are characterized by constant service time t_a followed by an interval t_w when they are unavailable to service requests. New requests for service are generated by processors after some interval (t_d or t_{ei}) has elapsed since their last request was serviced. These combined aspects of the multiprocessor

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

do not allow it to be handled by the common models of queueing theory. It does not appear to the author that a rigorous solution of this queueing situation can be readily obtained. Given this, there are basically two approaches that can be taken: (1) simplify the model sufficiently so that it can be solved by rigorous methods or (2) attempt an approximate solution. The latter approach is taken in this thesis; the analysis appears in the subsequent sections of this chapter. The former approach is taken by Skinner and Asher, (1969). They model the multiprocessor as a discrete Markov chain. The basic time interval is a memory cycle time. They assume a matrix of probabilities which express the likelihood that a given processor requests service from a given memory at the beginning of the memory cycle. They also assume matrices of probabilities which express the likelihood of the various outcomes that can arise when there are simultaneous requests to one memory by several processors. The states of the modelled system are characterized by the processors delayed and the memories for which they are delayed. A state transition matrix is formed from the previously mentioned probabilities and from this matrix the steady-state probabilities of the various states are determined. With this information the average amount of delay experienced by a processor in making a memory request is computed.

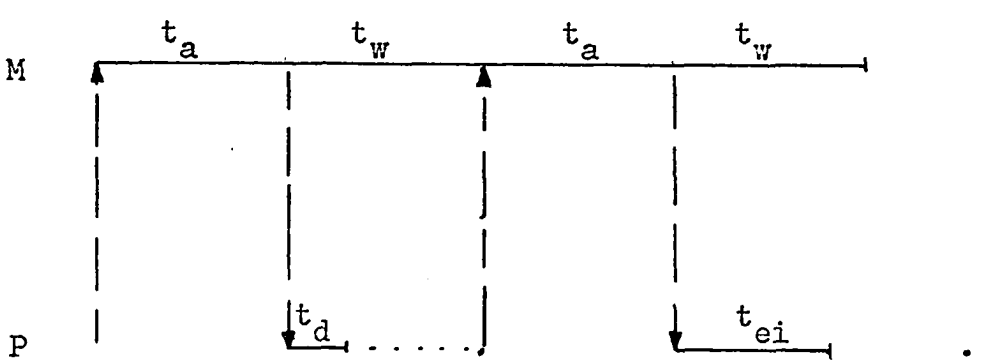
There are two problems with this approach. The first

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is that as the number of memories and processors increases, the number of potential states of the system becomes quite large and it is difficult to obtain other than a numerical solution. The second problem is obtaining the required probabilities. The probability that a processor directs a request to memory during a given time interval is dependent not only on the relation of the memory speed to the processor speed but also on the amount of delay a processor experiences in getting a memory request serviced. Since, in essence, that delay is what the analysis is supposed to determine, it is difficult to see how the required probabilities can be obtained in an analytic manner. (Skinner and Asher obtain the probabilities that they use in their model by first simulating the system and then making measurements on the simulated system. The necessity of doing this would seem to diminish the utility of the model.)

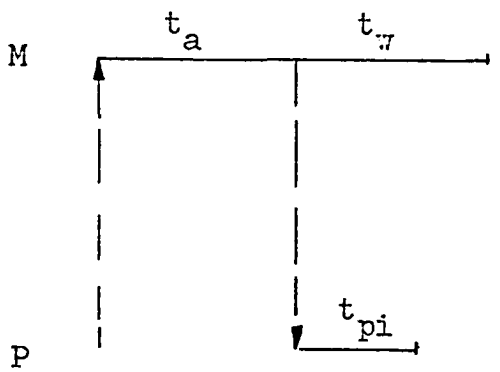
B. Modified Instruction Format

The previous discussion has assumed the single address format instruction as the model of a computer instruction. This instruction format has an ITD as follows:



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The instruction consists of two instances of the following operation sequence: the accessing of memory followed by an interval of processor activity. Hence an execution of a single address instruction can be approximated as two successive executions of a simple instruction with the following ITD:



where t_{pi}, the average processor activity time, is defined:

$$t_{pi} = (t_d + t_{ei})/2. \qquad (4.1)$$

This instruction is termed a unit instruction. The execution rate for unit instructions is termed UER to distinguish it from the IER. For the situation here UER = 2 x IER. We will now average over the instruction set and compute a single value t_p defined:

$$t_p = \sum_I f_i t_{pi}. \qquad (4.2)$$

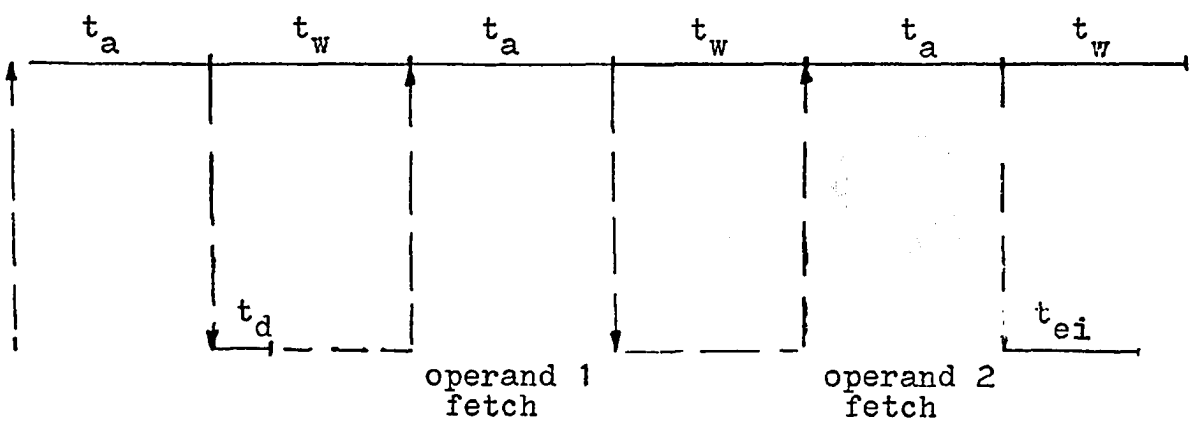
We will henceforth assume that all the instructions of the instruction set are made up of unit instructions with a single value t_p. As the analysis of chapter III would

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

suggest, the important relation to consider is that between t_p and t_w. There are three cases of interest and they are discussed individually in the following sections:

1. $t_p = t_w$
2. $t_p < t_w$
3. $t_p > t_w$.

To mention the unit instruction only in relation to the single address format instruction is to overlook its considerable generality. Obviously, instructions with no operand reference map directly into unit instructions, but the operation sequence of the unit instruction is sufficiently basic - a memory access followed by processor activity - that nearly any instruction format can be easily mapped into a series of them. For example, consider a two (operand) address instruction format which has the following ITD:

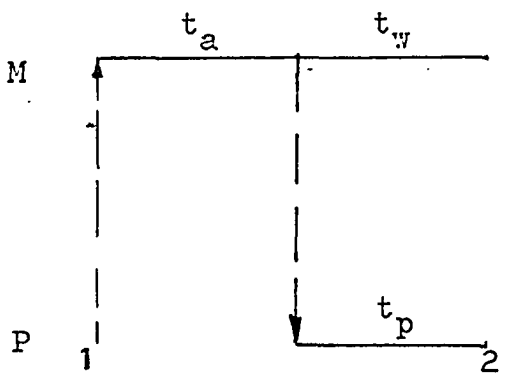


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

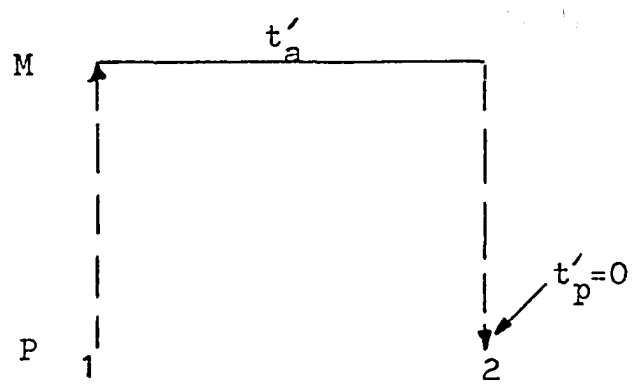
This is mapped into three unit instructions each with an average processor activity time $t_p = (t_d + t_{ei})/3$. Since three unit instructions are required the UER = 3 x IER. Other instruction formats may be handled in a similar manner.

C. Multiprocessor with $t_p = t_w$

In order to facilitate the discussion a further change in the instruction format is indicated. Unlike the change in section B the following is purely a conceptual transformation which introduces no approximation in the analysis. The ITD for the unit instruction when $t_p = t_w$ is:



This is transformed to the following ITD:



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The memory now has an access time of t_c and zero recycle time; the processor execution time is also zero. This transformation introduces no change in the sense that the performance of a system with either ITD is the same. For both ITD's the memory access begins at point 1 and the processor execution and the memory recycling are completed at point 2. With this new instruction format we are now ready to determine the UER for the multiprocessor.

Let us assume that we have m memories (m-way interleaved) denoted M_j; j = 1,...,m; and n processors denoted P_i; i = 1,...,n. From the instruction format, we can see that one unit instruction is executed for each memory cycle which is utilized by a processor. The maximum rate at which memory cycles are available is m/t_c and this represents an upper bound on the UER. The problem of finding the UER reduces, in essence, to that of determining that fraction of the total number of memory cycles which are utilized by all the processors. Notice that this represents quite a different approach from that used in chapter III. The analysis in chapter III might be termed processor oriented since the time required to access memory is simply considered a delay which is added to the processor execution times in order to get the total instruction execution time. The analysis of this chapter is memory oriented since the processors are considered only to the extent that they affect the memory utilization.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

We will term a processor queued if it is either waiting for or in the process of receiving memory service. A memory is termed occupied if it has one or more processors queued and unoccupied if it has no processors queued. A processor is termed active if it is currently being serviced by a memory. Let us consider an interval of time equal to t_c. For each memory which is occupied at the beginning of the interval, there is exactly one memory request serviced during that interval and hence exactly one unit instruction executed. For each memory that is unoccupied at the beginning of the interval there are no memory requests serviced (and hence no unit instructions executed). Because the modified instruction format has $t_p' = 0$ there are always n processors queued. Let us now define a random variable Z_j; $j = 1,\ldots,m$; where:

$$Z_j = \begin{cases} 0 & \text{if } M_j \text{ is unoccupied} \\ 1 & \text{if } M_j \text{ is occupied} \end{cases} \tag{4.3}$$

If X is a random variable which takes on values equal to the number of occupied memories, then:

$$X = \sum_j Z_j \tag{4.4}$$

The expected value of X, E(X), is the average number of occupied memories. From the previous discussion it should be clear that:

$$UER = E(X)/t_c \ . \tag{4.5}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

From equation 4.4 we have:

$$E(X) = E(\sum_j Z_j) = \sum_j E(Z_j) \qquad (4.6)$$

where $E(Z_j)$ is the expected value of Z_j. Since all the memories are identical, equation 4.6 reduces to:

$$E(X) = mE(Z_j) \quad \text{for any } j. \qquad (4.7)$$

We now wish to focus on one memory M_j and determine Z_j. The approach used here is related to the occupancy (or distribution) problems of combinatorial analysis (Feller, 1968). From the foregoing discussion we know that there are always n processors queued. The probability of any given processor memory request going to any given memory and hence queued for that memory is, as in chapter III, 1/m. In particular, the probability that any given processor is queued for M_j is 1/m and the probability that any given processor is not queued for M_j is 1 - 1/m. If Y is a random variable equal to the number of processors queued for M_j, the probability that Y = r is given by a binomial distribution:

$$p(Y = r) = p(r) = \binom{n}{r}(1/m)^r(1 - 1/m)^{n-r}. \qquad (4.8)$$

From the definition of Z_j and Y we now compute $E(Z_j)$:

$$\begin{aligned} E(Z_j) &= (0)p(0) + \sum_{r=1}^{n}(1)p(r) \\ &= \sum_{r=0}^{n} p(r) - p(0) \\ &= 1 - p(0). \end{aligned} \qquad (4.9)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Using equation 4.8 in equation 4.9 we find:

$$E(Z_j) = 1 - (1 - 1/m)^n \ . \qquad (4.10)$$

We then use equations 4.5 and 4.10 to compute the UER:

$$UER = (m/t_c)(1 - (1 - 1/m)^n) \ . \qquad (4.11)$$

E(X), the average number of occupied memories, is a function of m and n; let us call this function g(m,n):

$$g(m,n) = m(1 - (1 - 1/m)^n). \qquad (4.12)$$

The function g(m,n) has certain properties of interest:

1. For m,n>1, g(m,n) is monotonically increasing in m and n. This shows that we always get an improvement in the UER by adding another memory or processor.
2. $g(m,n) \leqslant$ minimum (m,n). The number of unit instructions executed during an interval t_c cannot exceed the number of memories or processors.

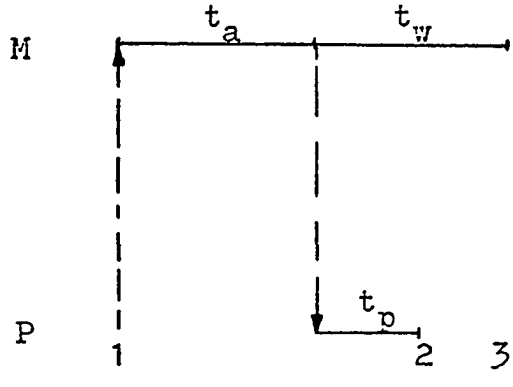
We might have stated the problem of finding the UER as follows. Let us randomly distribute n processors among m memories. The UER is the average number of memories which receive processors multiplied by $1/t_c$. Riordan (1958) shows by quite different methods than we have employed that the average number of memories which would receive processors is g(m,n). (Riordan's work is in combinatorial analysis; he speaks of balls and cells rather than processors and memories.) This method of problem formulation shows the approximate nature of the analysis. It has been implicitly assumed that all n processors make random requests

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

during each interval of time t_c. In a real computer $n' \leq n$ processors make requests during t_c: if there are several processors queued for service at a memory, only the one serviced during the interval t_c makes a new request at the end of that interval. Consequently, unfavorable (in terms of the effect on the UER) distributions of processors (a number of processors queued for one memory) tend to be more frequent in an actual computer than would be suggested by the analysis. The result of this is that the UER specified by equation 4.11 is somewhat higher than would be actually observed. We might expect that the most significant deviation between the actual and the computed UER to be the greatest when there is a high probability of a number of processors being queued for a single memory. This would occur when $n/m \geq 1$ and m is small.

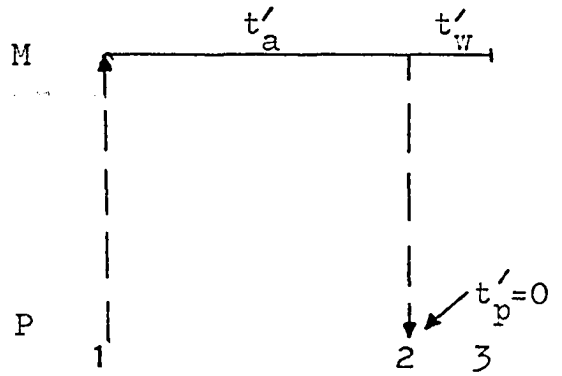
D. Multiprocessor with $t_p < t_w$

When $t_p < t_w$ the ITD is:



As in section C we perform a transformation on the ITD to get:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



The access time becomes $t'_a = t_a + t_p$, the memory restore time becomes $t'_w = t_w - t_p$ and the processor execution time t'_p goes to zero. The transformation is such that the performance of a system with either ITD is the same. For both ITD's the memory access starts at point 1, the processor execution is through at point 2, and memory restore is completed at point 3.

We recall the definition of an active processor as one whose memory request is currently being serviced. When that service is completed, an active processor can make a new request to either an occupied or an unoccupied memory. If it makes a request to an occupied memory, there is no appreciable advantage gained from the fact that $t_p < t_w$; the processor must wait anyway. On the other hand, if the request is made to an unoccupied memory the processor's request is serviced immediately; and there is an advantage associated with t_p being less than t_w. The proba-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

bility of an active processor making a request to an occupied memory is defined as:

$$p(occ) = \frac{\text{average number of occupied memories}}{m}. \tag{4.13}$$

The probability of a request to an unoccupied memory is:

$$p(unocc) = 1 - p(occ). \tag{4.14}$$

We estimate the number of occupied memories as $g(m,n)$. Hence:

$$p(occ) = 1 - (1 - 1/m)^n \tag{4.15}$$

and

$$p(unocc) = (1 - 1/m)^n. \tag{4.16}$$

Using the ideas of chapter III, we compute the average amount of time required to execute a unit instruction by an active processor:

$$\begin{aligned} E(t) &= p(occ)(t_c) + p(unocc)(t_a') \\ &= p(occ)(t_c) + p(unocc)(t_a + t_p) \\ &= t_c + (1 - 1/m)^n(t_p - t_w). \end{aligned} \tag{4.17}$$

The rate of execution R is just $1/E(t)$:

$$R = \frac{1}{t_c} \frac{1}{1 + (1 - 1/m)^n \frac{t_p - t_w}{t_c}}. \tag{4.18}$$

Now this is the unit instruction execution rate for one of the active processors. The UER for the multiprocessor is just R multiplied by the number of active processors which is also estimated as $g(m,n)$. Thus:

$$UER = \frac{m}{t_c} \frac{1 - (1 - 1/m)^n}{1 - (1 - 1/m)^n \frac{t_w - t_p}{t_c}}. \tag{4.19}$$

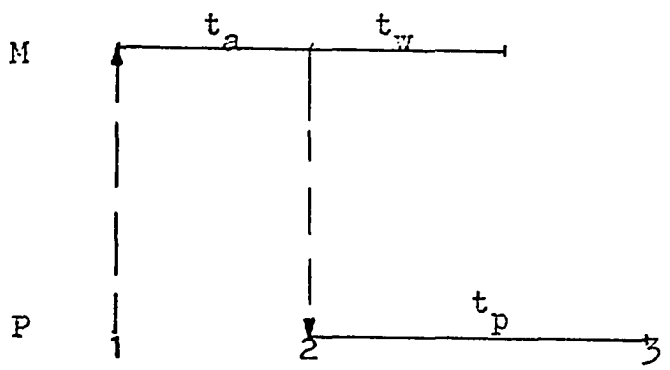
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Since the denominator of the fraction is less than one, the UER is greater for the case of $t_p < t_w$ than for the case of $t_p = t_w$ when m and n are the same.

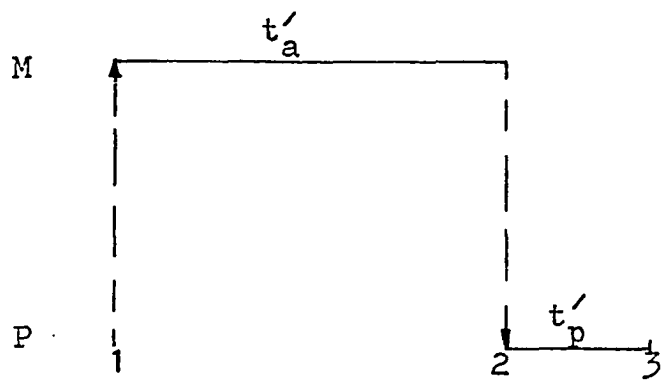
We have used g(m,n) as an estimate of both the average number of active processors and occupied memories. Actually for $t_p < t_w$ the average number would be somewhat higher than g(m,n). The increased number of occupied memories would tend to decrease the performance (since there is a reduced probability of a request to an unoccupied memory) while the increased number of active processors would tend to increase it. Simulation studies suggest that the effects almost cancel (chapter V).

E. Multiprocessor with $t_p > t_w$

When $t_p > t_w$ the following ITD applies:



This ITD can be transformed to the following:



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The memory access time becomes t_c, the memory recycle time goes to zero, and the processor time goes to $t'_p = t_p - t_w$. Again the transformation is such that the performance of a system with either ITD is the same. For both ITD's the memory access begins at point 1, the memory is restored at point 2, and the processor execution is completed at point 3.

For the previous two cases there were always n processors queued. For this case, because $t'_p > 0$, there will be, in general, less than n processors queued. This introduces an additional complication into the analysis. Let us suppose that there is a constant (in other words, independent of time and the state of the memory queues) probability p_m that a given processor is queued for memory service. Let Q be a random variable which takes on values equal to the number of processors so queued. The probability that k processors out of n are queued is given by a binomial distribution:

$$p(Q = k) = p(k) = \binom{n}{k}(p_m)^k(1 - p_m)^{n-k} \qquad (4.20)$$

When k processors are queued the average rate which at unit instructions are executed is given by equation 4.11 with n replaced by k. Defining this as R(k) we have:

$$R(k) = (m/t_c)(1 - (1 - 1/m)^k). \qquad (4.21)$$

The non-zero value of t'_p does not affect in any direct way the rate at which instructions are executed; however its effect is felt indirectly through its influence on the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

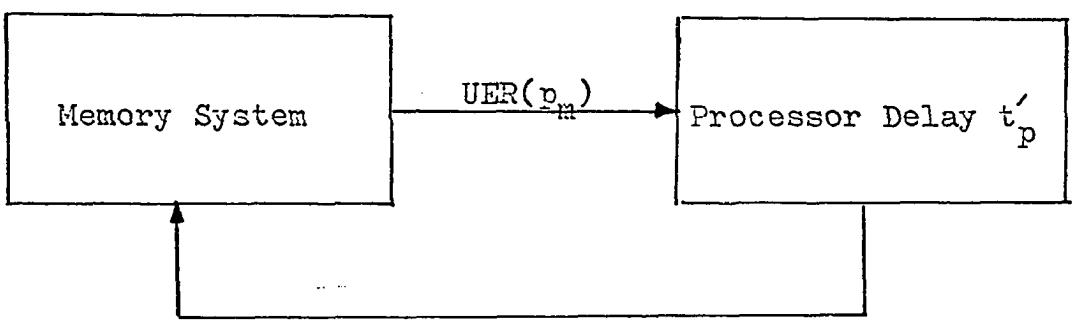
on the value of p_m. We now compute the UER as the expected value of R. From equation 4.20 and 4.21 we have:

$$\begin{aligned} UER &= \sum_k R(k)p(k). \\ &= \sum_k (m/t_c)\binom{n}{k}(1-(1-1/m)^k)(p_m)^k(1-p_m)^{n-k} \\ &= (m/t_c)(1-\sum_k \binom{n}{k}(p_m(1-1/m))^k(1-p_m)^{n-k}) \\ &= (m/t_c)(1-(1-p_m/m))^n \end{aligned} \qquad (4.22)$$

the latter resulting from applying the binomial theorem to the summation. We note the UER specified by equation 4.22 is a function of p_m. Equation 4.22 is identical to equation 4.11 except for the replacement of $1/m$ with p_m/m. Since $p_m < 1$, the UER for $t_p = t_w$ is greater than the UER for $t_p > t_w$ assuming the same values for m and n. Because the number of processors queued is binomally distributed, the average number of processors queued is np_m. Hence:

$$p_m = \frac{\text{average number of processors queued}}{n} \qquad (4.23)$$

A flow diagram of the instruction execution is as follows:



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Serviced memory requests leave the memory system at a rate specified by equation 4.22. They then experience a delay t_p' before making a new memory request. Let n_p be the average number of processors not queued in the memory system and n_m the average number queued. Necessarily:

$$n_p + n_m = n \tag{4.24}$$

and thus substituting in equation 4.23 we have:

$$p_m = (n - n_p)/n. \tag{4.25}$$

From the flow diagram the average number of processors not queued must be the product of the average unit instruction processing rate and the delay t_p'. Thus:

$$n_p = UER(p_m)t_p'. \tag{4.26}$$

Using equation 4.26 and the relation $t_p' = t_p - t_w$ we have:

$$p_m = 1 - \frac{m}{nt_c}(1 - (1 - p_m/m)^n)(t_p - t_w) \tag{4.27}$$

or

$$0 = p_m + (m/n)\left(\frac{t_p - t_w}{t_c}\right)(1 - (1 - p_m/m)^n) - 1 \tag{4.28}$$

which is an n^{th} order polynomial equation in p_m. It must be solved for the value of p_m in the interval (0,1). That there exists one and only one solution of equation 4.28 in (0,1) can be seen by considering equation 4.27. As p_m goes from zero to one, the left hand side of equation 4.27 increases monotonically from zero to one while the right hand side decreases monotonically from one. There is one and only one value of p_m in (0,1) for which the right and left hand sides are equal. Once a value of p_m is obtained,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

it is substituted in equation 4.22 to obtain the UER.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter V Simulation

A. Reasons for Simulation

In the multiprocessor analysis of chapter IV two principal approximations are made. The first is the replacement of the single address format instruction with two successive unit instructions and the associated averaging over the instruction set to get a value of t_p. The second approximation is the treating of the inherent multiprocessor queueing problem as a distribution or occupancy problem. The intent of the simulation studies is to ascertain the effects of the approximations over a limited set of cases.

B. The Simulator

The simulator is the "next most imminent event" type. In the simulator certain rules are applied to determine the sequence of events in the simulated system and the timing of the events is determined accordingly. A simulator of this type is quite simple and executes rather rapidly.

The simulator is set up to handle n processors denoted P_i; $i = 1,...n$; and m memories denoted M_j; $j = 1,...,m$; where m and n are arbitrary and specified at run time. Associated with each memory M_j is a time t_{mj} which is the earliest time M_j can initiate servicing a new

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

memory request. Similarly, associated with each processor P_i is a time t_{pi} which is the earliest time processor P_i can initiate a new request for memory service. The simulator is arranged so that one cycle of simulation corresponds to the execution of one unit instruction by one processor. There are two basic rules which govern the sequence of events in the simulator. First, the instruction unit execution of any given simulation cycle is always associated with the processor P_i for which the value of t_{pi} is a minimum at the beginning of the cycle. (If there is more than one value of i for which t_{pi} is a minimum; then the largest value of i is arbitrarily chosen.) Second, an instruction unit execution involving P_i and M_j always commences at the maximum of the times t_{pi} and t_{mj} since that is the earliest time at which both P_i and M_j are available. With these rules in mind we can follow step by step the action of the simulator whose flowchart appears in figure 6.

1. The simulator is initialized. The values m, n, t_c, and t_w, are specified; t_{mj} and t_{pi} are set to zero for all i and j. Y is set to the total number of unit instruction executions to be simulated.

2. The value i is selected so that t_{pi} is a minimum. This corresponds to the selection of the processor for the current simulation cycle.

3. The value of j is selected. Similarly, this corresponds to the selection of the memory for

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

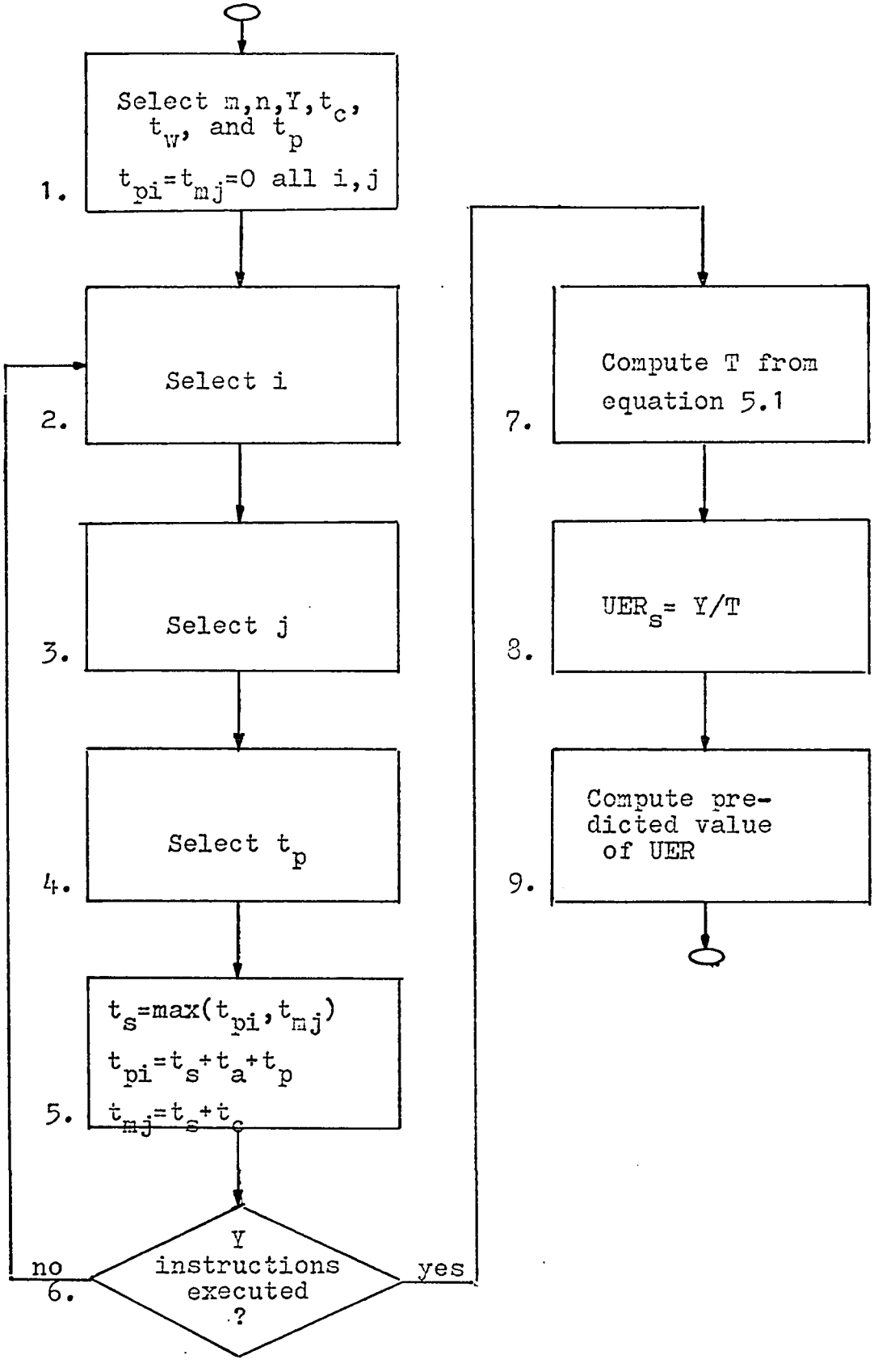


Figure 6. Simulator

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the current simulation cycle. A multiplicative congruent random number generator (Kruskal, 1969) is used to uniformly generate integers in the range 1,...,m.

4. The value of t_p is selected. Normally it is a constant but in one simulation, however, it is is selected so that for any given processor it oscillates between two values whose average is t_p.

5. A start time (for the unit instruction execution of the current simulation cycle) t_s is computed as the maximum of t_{pi} and t_{mj}. The time t_{mj} is then set to the sum of the start time and the memory cycle time t_c. The time t_{pi} is set to the sum of the start time, the memory access time t_a and the processor execution time t_p.

6. If less than Y unit instructions have been executed, another simulation commences at step 2. Otherwise the computation of the results begins at step 7.

7. The exact time at which the simulation ends is not precise; there are m + n times in the simulator. Probably the most reasonable estimate of the end time is:

$$T = \frac{1}{m + n}\left(\sum_i t_{pi} + \sum_j t_{mj} \right).$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

8. The simulation unit instruction execution rate UER_S is the ratio of the total number of executions to the time required to execute them; hence:

$$UER_S = Y/T \ .$$

9. The UER is computed from either equation 4.11, 4.19, or 4.22 depending on the relation of t_p to t_w. If equation 4.22 is approriate then the value of p_m must be computed. A Newton-Raphson search technique (Pierre,1969) is employed; it converges rapidly to the value of p_m in the interval (0,1).

As can be seen from the above, the amount of simulator activity per unit instruction execution is independent of m. The value of n only determines the number of values of i which must be searched to find the minimum in step 2.

The actual simulator is more involved than described here and has provision for several instruction formats (including the simple unit instruction used here). It accepts input in the form of an instruction set where the format, execution times and relative frequency of each instruction may be specified. The sequence of events in the simulator when executing unit instructions is precisely just those described above. The simulator is written in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Algol[1] for the Univac 1108 and it simulates the execution of about 2000 unit instructions per second.

C. Results

A set of simulation results is presented in figures 7 through 11. The results are presented in a normalized form: the performance of a one processor, one memory system with $t_p = t_w$ is taken to be one. We use figures 7 through 10 to verify the basic multiprocessor analysis; figure 11 is used to verify the instruction reduction. The figures are discussed individually and represent various cases of interest:

Figure 7: For this case $t_p = t_w = 0.5t_c$. The difference between the predicted and the simulated values is small. The maximum deviation is about 8% with the predicted value higher than the simulated and occurs when the ratio of processors to memories is one or greater. This is in accordance with the observation of chapter IV, section C.

Figure 8: Here $t_p = 0.1t_c$ and $t_w = 0.5t_c$. The maximum deviation observed is about 10%; the predicted performance again higher than the simulated performance. The worst deviations

[1] A text of the simulator appears in the appendix.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

occur when 3 or 4 very fast processors ($t_p = 0.1t_c$) are used with 2 to 4 memories. (This is a situation which would not likely occur in practice because it probably would be uneconomical to configure a system in this fashion.) This is again in accordance with the observation of chapter IV, section C. The slope of the four processor performance curve is still high even when n = 16 suggesting that the performance of the system can be significantly improved by adding memories (This may or may not be economical though.) Although figure 8 does not show it, the curves for the simulated and predicted results converge for n = 3 and n = 4 when $m > 64$.

Figure 9: This is the same as figure 8 except that $t_p = 0.2t_c$. Basically the same comments apply.

Figure 10:Here t_p is greater than t_w: $t_p = 2t_c = 4t_w$. The results show excellent agreement of the simulated and the predicted results. For $m > 1$ the corresponding curves are nearly indistinguishable.

Figure 11:These simulation results are presented to verify the reduction of the instruction set to a unit instruction with a single value of t_p. The curves present two different simulations. For one the value of t_p oscillates between $0.1t_c$ and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

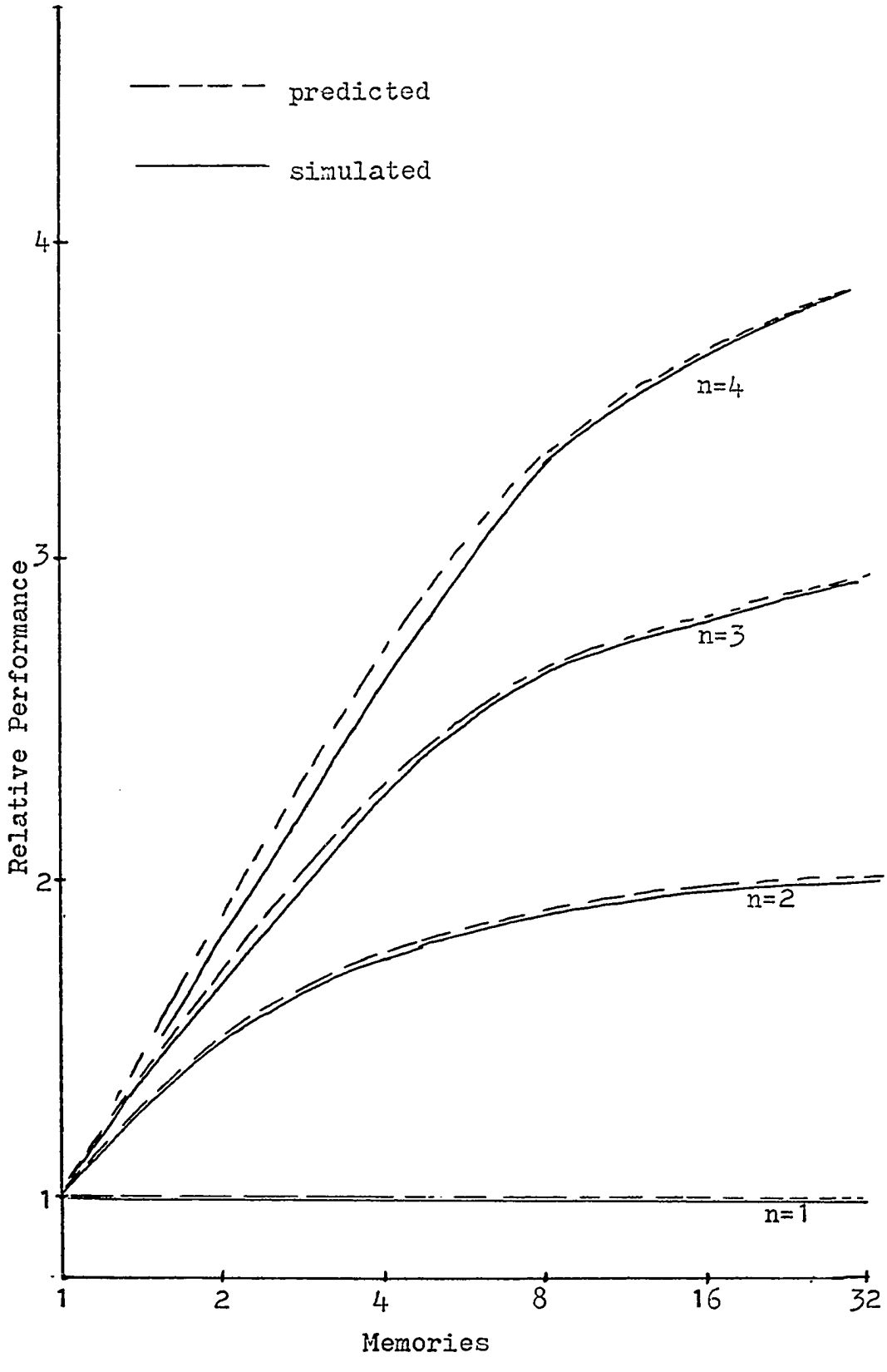


Figure 7. Results for $t_p = t_w$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

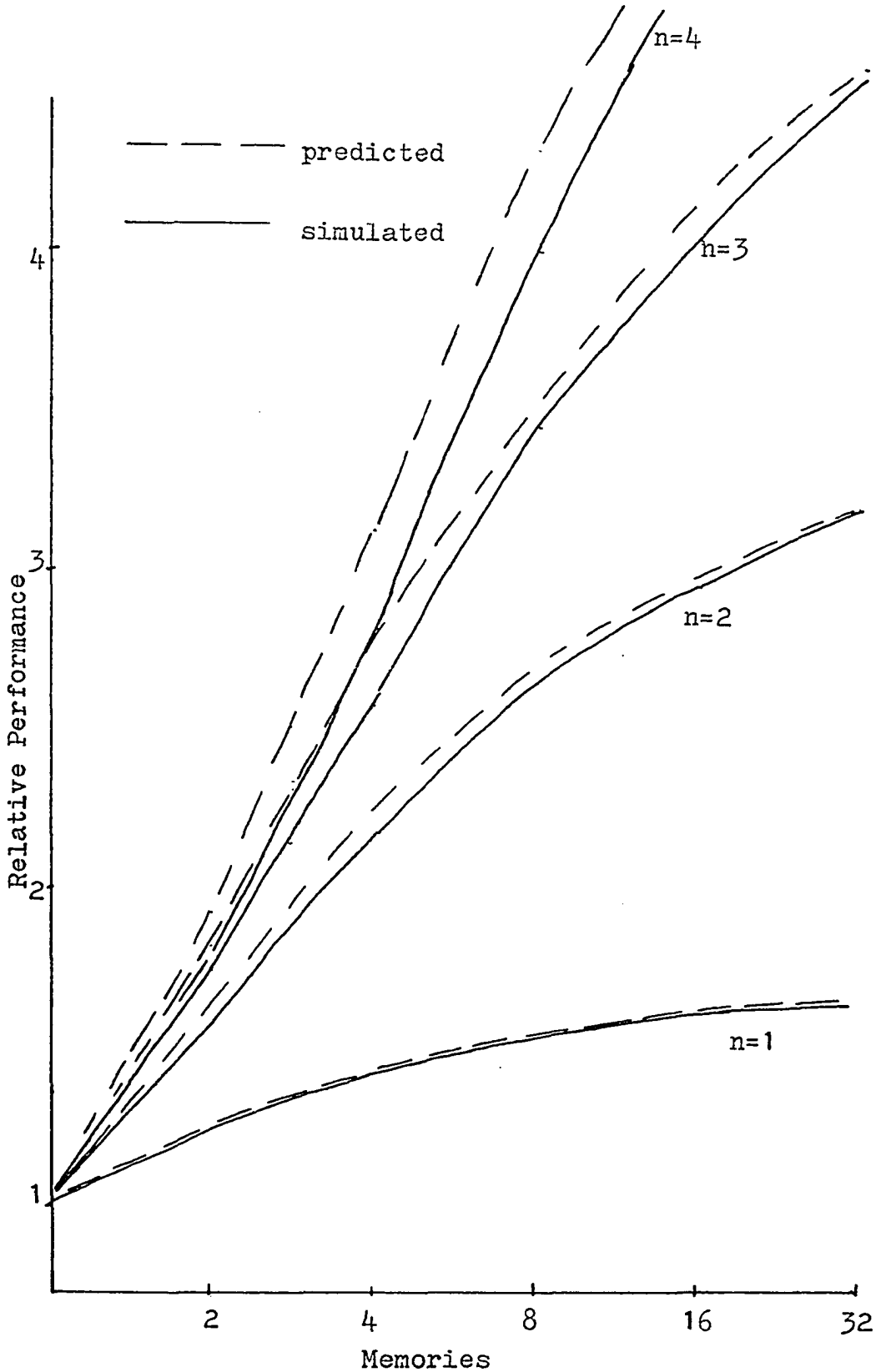


Figure 8. Results for t_p=0.1 t_c

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

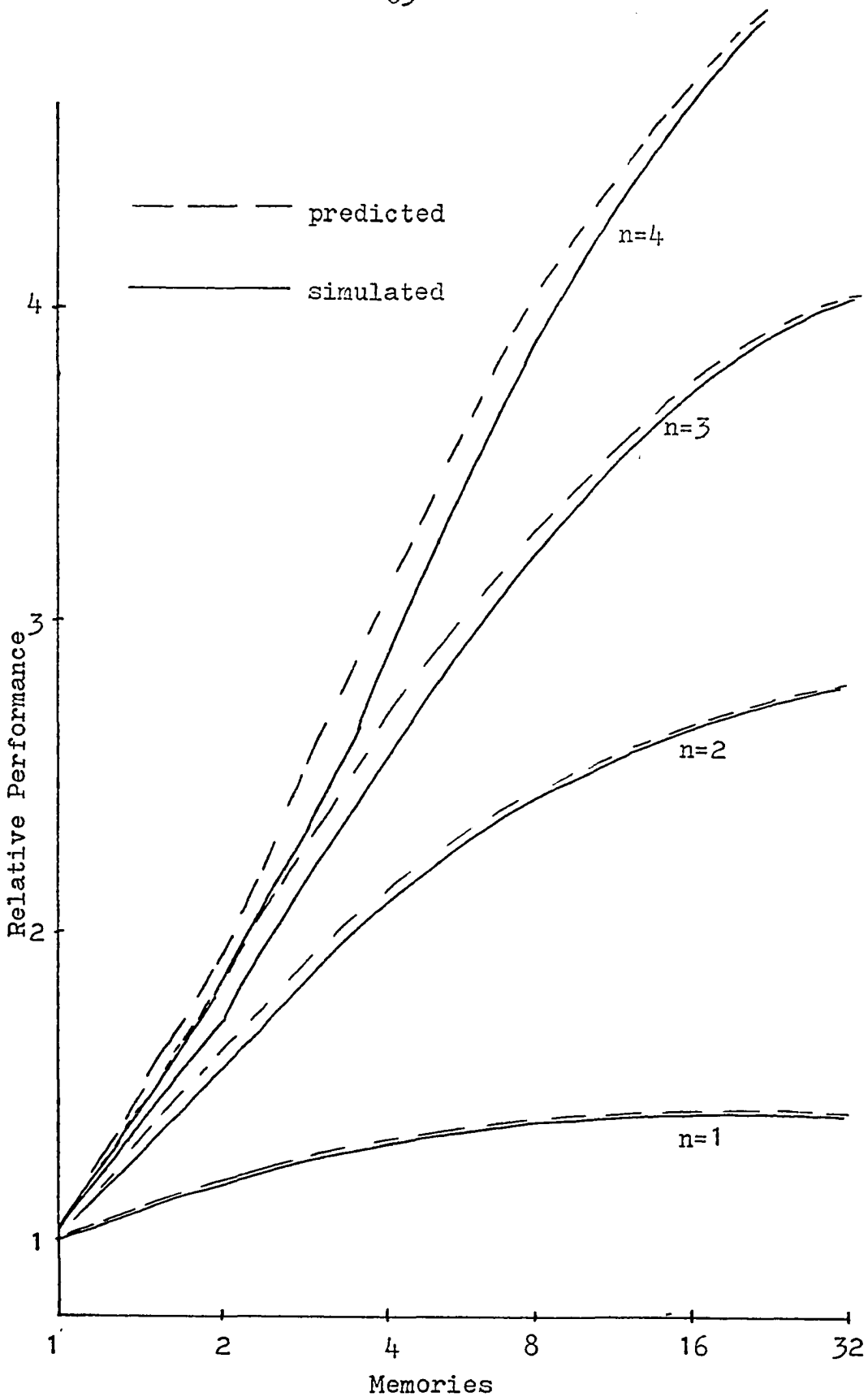


Figure 9. Results for $t_p = 0.2t_c$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

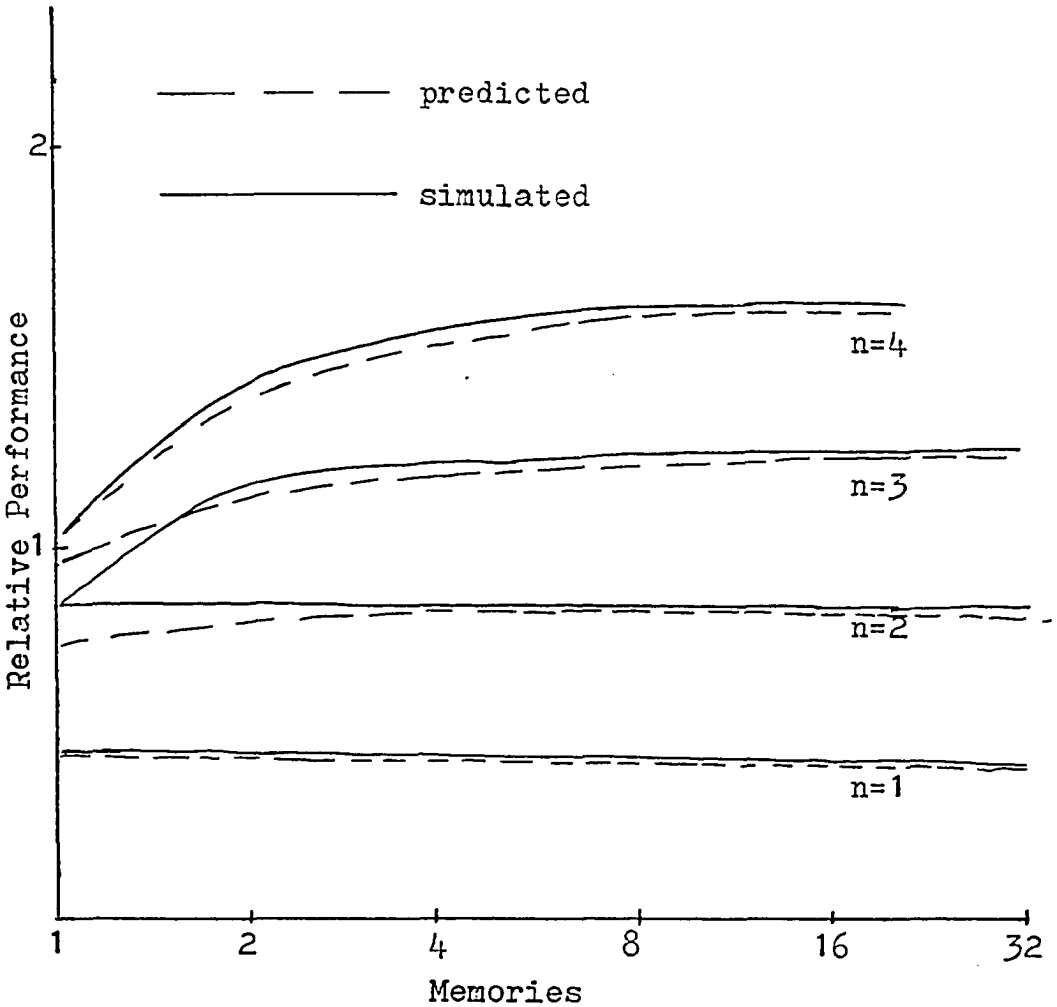


Figure 10. Results for $t_p=2t_c$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

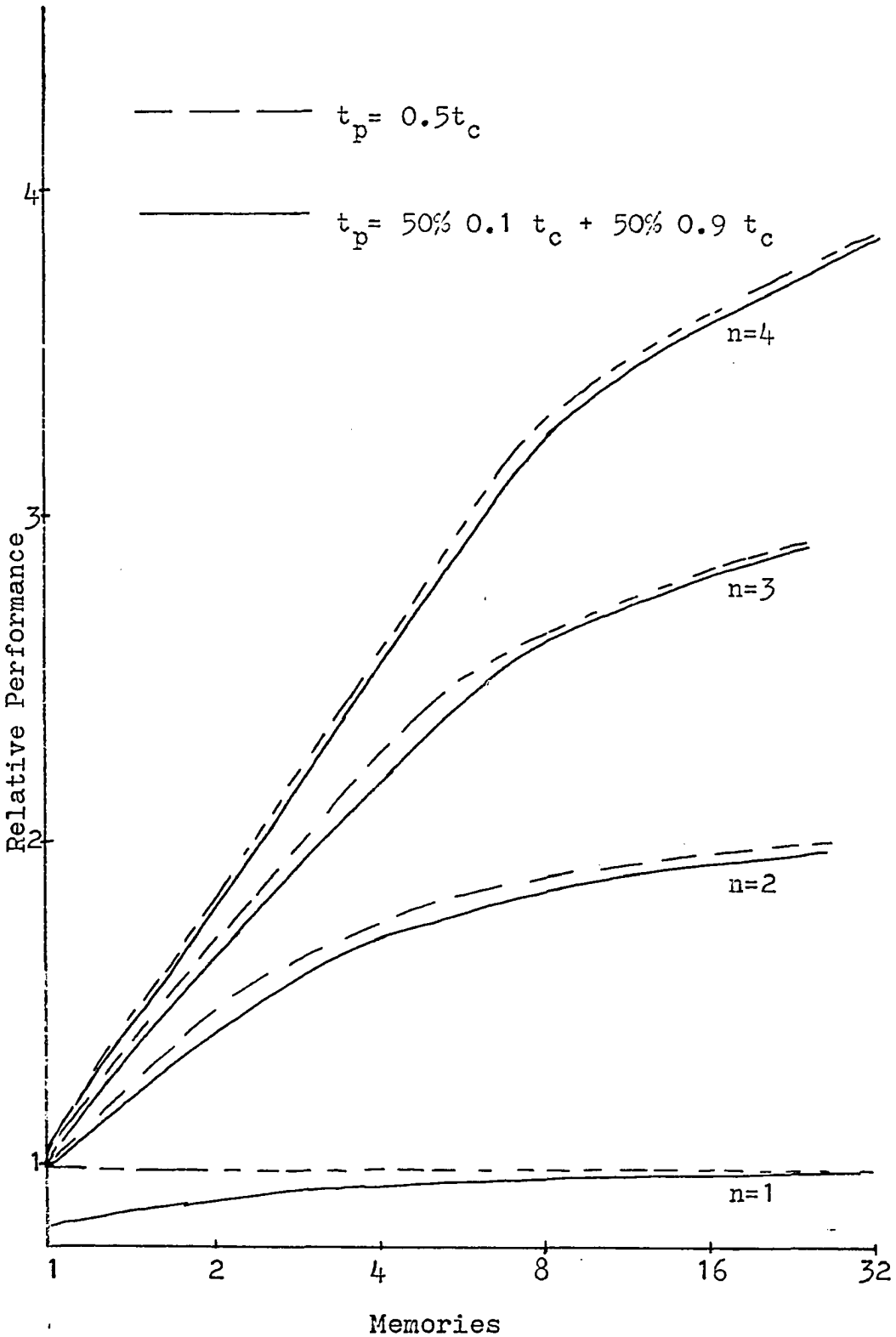


Figure 11. Results for Varying t_c

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$0.9t_c$ for a given processor with a mean $0.5t_c$; for the other a constant value of $t_p = 0.5t_c$ is used. The results suggest that the reduction is probably a reasonable one. The relative performance is slightly lower for the case where t_p varies than for the case where t_p is fixed; this is generally in accordance with what we would expect for a stochastic service system.

D. A Comparison with Other Simulation Results

By using some published multiprocessor simulation results it is possible to provide a form of independent verification of the analytic results of chapter IV. Rosenfeld (1969) discusses the results of simulation of the solution (by Gauss-Seidel iteration) of a set of simultaneous linear algebraic equations on a multiprocessor computer.

The processors simulated have the general characteristics and instruction set of the IBM 360 computer series. Although the relative frequencies and processor execution times for the instruction set are not given, a set of total instruction execution times (which are presumably nominal times for a single processor computer) are given. From these times it appears that the instruction execution time is roughly equal to the memory cycle time multiplied by the number of memory cycles needed to execute the instruction. Thus one can reasonably estimate that the average processor

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

activity time is about equal to the memory restore time, and hence Rosenfeld's system can be described in our terminology as a multiprocessor with $t_p = t_w$.

As we discussed in chapter IV the IER is directly determined by the extent of memory cycle utilization. Fortunately, one of the measurements Rosenfeld makes on his simulated system is the memory cycle utilization and this makes a direct comparison with his results quite simple.

For the multiprocessor with $t_p = t_w$ the memory cycle utilization is speciifed by the function g(m,n) defined by equation 4.12:

$$g(m,n) = m(1 - (1 - 1/m)^n).$$

Figure 12 shows Rosenfeld's observed memory utilization (solid lines) plotted together with g(m,n) (broken lines). The agreement between the simulated and the predicted results is rather good with the utilization in the simulated generally somewhat higher. At least one reason may be advanced to account for this: an incorrect assumed value of t_p. If t_p were assumed somewhat less than t_w, the analytically predicted value of memory utilization would increase and the curves for the simulated and the predicted results would become nearly identical. Regardless of the value of t_p assumed the general shape of the curves reflecting the simulated and predicted results is the same.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

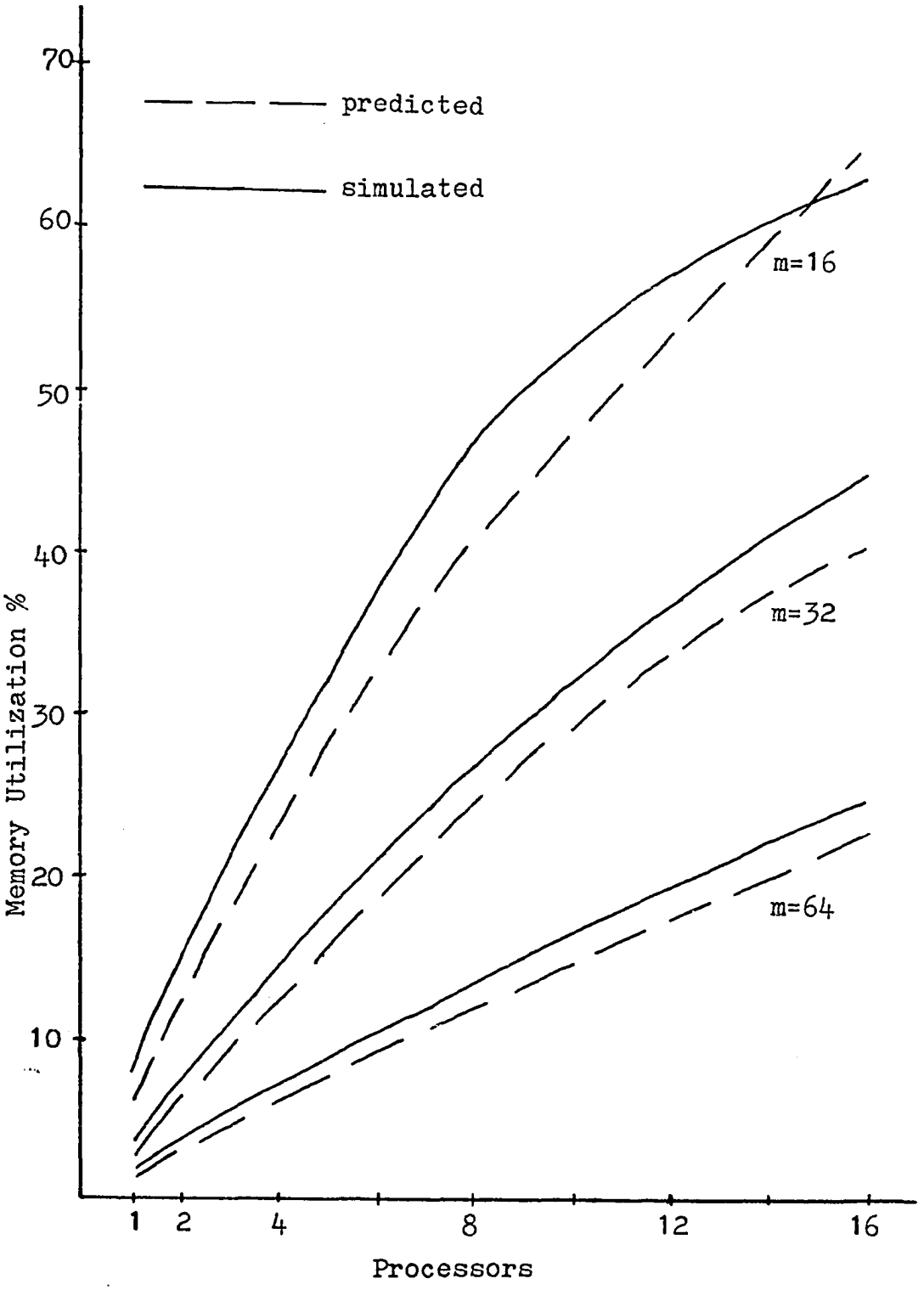


Figure 12. A Comparison with Rosenfeld's Results

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter VI An Analysis of I/O Effects on Processor Performance

A. I/O Activity

In chapter II we discussed the technological reasons necessitating the presence of primary and secondary memories in computer systems. The information stored in a secondary memory is moved into the primary memory only when it is actually ready to be used by the processor and after processing it is returned to the secondary memory. The information flow between the memories is generally called input/output (i/o) activity and this activity has a degrading effect on the UER. Each word of information transferred between the primary and the secondary memory usually uses one cycle of the primary memory.[1] If both the i/o and the processors are active simultaneously, conflicts arise when both direct a request to the same memory simultaneously. Normally if a conflict occurs the i/o request is served first and the processor request is deferred until the subsequent memory cycle. In other words, an i/o service request has a higher priority than a processor request. The reason for granting priority to the i/o is due to the rotating character of commonly used secondary memories (drums and discs). For each i/o request that is not serviced

[1] In some computers additional cycles are used to count the number of i/o transfers and to specify the memory locations to which the transfers go.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

sufficiently rapidly, the i/o transfer process must be delayed by the time required for one full rotation of the memory device thus delaying the processor waiting for the i/o the same amount of time. For some types of i/o it is possible, however, to implement a dynamic priority scheme where sometimes a processor request has higher priority than an i/o request and it is shown in section C that this approach leads to a smaller degradation in processor performance than the simple priority scheme.

Several authors have given analyses of the effects of i/o activity.[2] Flores (1964) determines the extent of queueing of i/o requests on memories. (His analysis does not consider the processors.) Flores' model is developed from the following ideas. The i/o requests for memory service are assumed to be generated by a Poisson process with a mean request rate of RIO. The requests are considered to be uniformly distributed among the m memories and hence each memory has a mean request rate of RIO/m. The memory is considered to be a server (in a queueing sense) with a constant service time t_c. The result is a simple queueing situation with Poisson input and constant service time. The mean time elapsing between the initiation of a memory request and the time the service of that request begins is computed. Flores does not propose, however, a purpose to which that time, once computed, can be put. Shemer and Gupta (1969) extend Flores' model to consider the effect

[2]The notation in the following discussion is not that of the original authors.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of i/o activity on the performance of a single processor. In their model a processor with an average processing time t_p generates random requests to the m memories. Simultaneously, i/o requests generated by a Poisson process with mean rate RIO compete with the processor for the available memory cycles. Their rather involved analysis allows for i/o queueing and they compute the average time required to complete a memory request initiated by the processor.

In order to understand the relation of the above authors' analysis to that of this chapter, it is necessary to look at the nature of the Poisson process (Hillier and Lieberman, 1967) used as a model of the source of i/o requests. Each memory experiences a mean request rate of RIO/m and hence during an interval of time t_c the probability distribution for the number q of i/o request received is:

$$p(q,t_c) = \frac{(RIO/(m/t_c))^q \, e^{-RIO/(m/t_c)}}{q!} .$$

The average number of requests received during t_c is $RIO/(m/t_c)$, but the above equation associates a non-zero probability for any finite number q of requests. (Although when $RIO/(m/t_c)$ is small the probabilities associated with large values of q fall off very rapidly.) The type of i/o activity which is likely to use a significant portion of the primary memory cycles (and thus significantly affect the UER) is that from very high speed discs and drums

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(or low speed core used as a secondary memory) and each of these is characterized by a regular periodic flow rate. The number of such devices likely to be in simultaneous operation in a computer system is small: often one; perhaps as many as three or four. While the Poisson process is a satisfactory model for representing the generation of requests from a number of (unsynchronized) periodic sources, probably an equally satisfactory model when the number of sources is small is simply to assume that there is a probability $RIO/(m/t_c)$ that one i/o request per memory is received during an interval t_c and a zero probability of more than one request. This is especially suitable when each of the i/o devices has a small amount of buffering (as it usually does). This assumption is used in section B. When there is only one i/o device with a periodic flow rate in operation, an advantageous i/o handling scheme can be implemented. This situation is assumed in section C.

B. Simple I/O Handling

In at least one way the i/o activity looks like the $n + 1^{st}$ processor in the multiprocessor system and it would be attractive to be able to handle it as such. However the multiprocessor analysis is derived on the basis of identical processors and because of the priority granted i/o requests, the i/o activity looks rather different from a processor. The i/o activity does, like the processors,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

contribute to the occupancy of the memory system and hence increases the rate at which memory cycles are utilized. Our general approach in the subsequent analysis is to compute occupancy of the memories with i/o, determine the rate of memory cycle utilization, and then apportion that rate between the processors and the i/o. (We shall present an analysis only for the multiprocessor case where $t_p = t_w$; extensions to cover the other cases are not difficult.)

Let us consider a particular memory, say M_j. Let A be the event that M_j is occupied by a processor request and let B be the event that M_j is occupied by an i/o request. From the previous discussion the probability of B is:

$$p(B) = \frac{RIO}{m/t_c} \tag{6.1}$$

where necessarily RIO must be such that $p(B) < 1$. We will assume that the probability of A is not affected significantly by the i/o activity. This is equivalent to the assumption that A and B are independent events. Thus when $t_p = t_w$, p(A) can be computed from equation 4.11:

$$p(A) = 1 - (1 - 1/m)^n. \tag{6.2}$$

The probability of a memory being occupied by either a processor or an i/o request is the probability of the event A or B. When A and B are independent the probability of the event A or B is:

$$p(A \text{ or } B) = p(A) + p(B) - p(A)p(B). \tag{6.3}$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Substituting equations 6.1 and 6.2 in 6.3 we have:

$$p(A \text{ or } B) = 1 - (1 - 1/m)^n - (1 - 1/m)^n \frac{RIO}{m/t_c} \cdot \tag{6.4}$$

Now, having determined the probability of occupancy of one memory, we can determine the rate R at which memory requests are serviced by multiplying by m/t_c:

$$\begin{aligned} R &= (m/t_c)p(A \text{ or } B) \\ &= (m/t_c)(1 - (1 - 1/m)^n - (1 - 1/m)^n \frac{RIO}{m/t_c}) \cdot \end{aligned} \tag{6.5}$$

The rate R includes the service of both i/o and processor memory requests. But since the i/o requests are served first, the rate R includes exactly a rate RIO of serviced i/o requests. Hence, the UER can then be determined by subtracting RIO from R:

$$\begin{aligned} UER &= R - RIO \\ &= (m/t_c)(1 - \frac{RIO}{m/t_c})(1 - (1 - 1/m)^n) \cdot \end{aligned} \tag{6.6}$$

We note that this is just the UER that would be observed without i/o multiplied by the factor $(1 - \frac{RIO}{m/t_c})$.

C. Dynamic Priority I/O Handling

We will assume for this analysis that the i/o requests originate from a single periodic source, and we will see that it is possible to specify a method of handling i/o requests which results in less degradation of the UER

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

than that specified by equation 6.6 above. In order to simplify the discussion and the following derivation, let us assume that the ratio $(m/t_c)/RIO$ is an integer whose value is $N>1$. This means that exactly $1/N$ of the total memory cycles of any given memory, say M_j, are used by the i/o. Furthermore, let us assume that the i/o requests are generally for sequential memory locations and hence M_j receives on i/o request exactly once for every N cycles. Let us assume that there is associated with each memory a one word buffer to hold the i/o information and a control mechanism to implement the following strategy:

1. If less than N - 1 cycles have elapsed since the i/o request was received, the processor requests have priority; an i/o request is serviced only if there is no processor waiting for service.
2. If N - 1 cycles have elapsed and the i/o request has not yet been serviced, the i/o request gets the current memory cycle.

We term this dynamic priority i/o handling.

Let us consider a sequence of N cycles for M_j. The probability that a given cycle is occupied by a processor request is p(A) specified by equation 6.2. The probability that a given cycle is not occupied by a processor request is 1 - p(A). In the absence of i/o requests the probability that k of the cycles are used by processor requests is specified by a binomial distribution:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$p(k) = \binom{N}{k}(p(A))^k(1 - P(A))^{N-k};\ k=0,\ldots,N. \quad (6.7)$$

Let C be a random variable equal to the number of cycles used to service both i/o and processor requests during the N cycle sequence. The expected value of C is:

$$E(C) = 1 + \sum_{k=0}^{N-1} \binom{N}{k}(p(A))^k(1 - p(A))^{N-k}k + (N - 1)p(N). \quad (6.8)$$

In the above expression, the first term accounts for the cycle received by the i/o, and the second term accounts for the cycles received by up to N - 1 processor requests. The last arises because even if the processors request all N cycles of the N cycle sequence, they only get N - 1; the i/o gets the remaining one. Equation 6.8 may be rewritten:

$$E(C) = 1 + \sum_{k=0}^{N} \binom{N}{k}(p(A))^k(1 - p(A))^{N-k} - p(N). \quad (6.9)$$

The summation represents the expected value of the number of processor requests during the N cycle sequence; hence it is just Np(A). Thus:

$$E(C) = 1 + Np(A) - p(N). \quad (6.10)$$

The average occupancy of a memory cycle over the N cycle sequence is E(C)/N; we can then compute R for this case:

$$R = (m/t_c)(E(C)/N)$$

$$= (m/t_c)(1/N + p(A) - p(N)/N). \quad (6.11)$$

Since we assumed $N = (m/t_c)/RIO$ we have:

$$R = (m/t_c)\left(\frac{RIO}{m/t_c} + (1 - (1 - 1/m)^n - \frac{RIO}{m/t_c}(1 - (1 - 1/m)^n)^N\right). \quad (6.12)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

As before the UER is determined by subtracting RIO from R:

$$UER = R - RIO$$
$$= (m/t_c)(1 - \frac{RIO}{m/t_c}(1 - (1 - 1/m)^n)^{N-1}) \times (1 - (1 - 1/m)^n). \quad (6.13)$$

The latter is just the UER without i/o multiplied by the factor $1 - (RIO/(m/t_c)(1 - (1 - 1/m)^n)^{N-1}$. Since $(1 - (1 - 1/m)^n)^{N-1} < 1$ this method of i/o handling results in a lower degradation of the processor performance. If $(m/t_c)/RIO$ is not an integer, then substituting for N in equation 6.13 the largest integer not greater than $(m/t_c)/RIO$ gives a satisfactory approximation for the UER.

D. Example

Consider a 4 processor, 4 memory system with $t_p = t_a = t_w = 0.5$ usec. and RIO = 10^6 requests/sec. The UER without i/o is computed from equation 4.11:

$$UER(\text{without i/o}) = (4/1.0 \times 10^{-6} \text{ sec.}) \times (1 - (1 - 1/4)^4)$$
$$= 2.73 \times 10^6/\text{sec.}$$

When the i/o is considered without the dynamic priority scheme we use equation 6.6 to find:

$$UER(\text{simple i/o}) = (1 - 1/4)(2.73 \times 10^6/\text{sec.})$$
$$= 2.02 \times 10^6/\text{sec.}$$

If we implement the dynamic priority i/o handling we find $N = (m/t_c)/RIO = 4$ and hence using equation 6.13 we have:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$\text{UER(dynamic i/o)} = (1 - (1/4)(1 - (1 - 1/4)^4)^3) \times (2.73 \times 10^6/\text{sec.})$$
$$= 2.52 \times 10^6/\text{sec.}$$

The UER with the dynamic priority scheme is about 25% higher than that obtained without it: a substantial improvement.

Let us define a memory efficiency e_m as:

$$e_m = \frac{\text{total memory cycles used/sec.}}{\text{total memory cycles available/sec.}}$$

$$= \frac{\text{UER} + \text{RIO}}{m/t_c} \qquad (6.14)$$

and a processor efficiency e_p as:

$$e_p = \frac{\text{total instructions executed/sec.}}{\text{total number of instructions executed if there were no memory delays/sec.}}$$

$$= \frac{\text{UER}}{n/t_p} . \qquad (6.15)$$

We can now compute the efficiencies for the preceeding example:

1. No i/o:

$$e_m = \frac{2.73 \times 10^6}{4.00 \times 10^6} = 0.67$$

$$e_p = \frac{2.73 \times 10^6}{8.00 \times 10^6} = 0.34$$

2. Simple i/o :

$$e_m = \frac{3.02 \times 10^6}{4.00 \times 10^6} = 0.75$$

$$e_p = \frac{2.02 \times 10^6}{8.00 \times 10^6} = 0.25$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3. Dynamic i/o:

$$e_m = \frac{3.52 \times 10^6}{4.00 \times 10^6} = 0.88$$

$$e_p = \frac{2.52 \times 10^6}{8.00 \times 10^6} = 0.32 \ .$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter VII Computer Design

A. Optimization Approach

The purpose of this chapter is to indicate how the multiprocessing models of chapter IV can be used in a simple automatic design program. An appropriate context in which to consider the design process is that of an optimization problem. The nature of the optimization problem is to relate the costs of a proposed design to the variables which reflect its structure and then choose the values of the variables so that required performance is obtained and the cost of the design is minimized. We have taken the UER as the basic measure of computer performance and the formulas of chapter IV relate the UER to the variables t_w, t_a, t_d, t_p, m, and n. If we can also relate the costs of the design to these variables we have the necessary relationships to formulate the optimization problem and hence to implement an automatic design program.

B. Costs and the Problem Formulation

The three types of components whose costs we consider to enter into the overall cost of the multiprocessor system are memories, processors, and switches. While it is interesting to consider the possibilities of relating by formula the costs of the components to their specifications, the relations would be both rather difficult to obtain and probably (because of the discrete nature of the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

manufacturing process) not very meaningful. Hence we shall assume that the costs of the components are related to the variables in a tabulated form.

There are several considerations that enter into the determination of the individual component costs:

1. Switches: The switch has to connect n processors to m memories. Hence there are m x n potential connections implied in the structures we are considering and if the cost to realize a simple switch is C_s, the total cost for the multiprocessor switch is about n x m x C_s.
2. Memories: As indicated in chapter II, the cost per word of a coincident current magnetic memory is lower in memories of a larger number of words than in one of a smaller number of words. We assume that the total memory system has been specified in advance to have w words. If the cost of a memory of w words and cycle time t_c is $C_m(w, t_c)$, then the total memory cost with an m- way interleaved memory is m x $C_m(w/m, t_c)$ assuming that a memory is available in that size and speed. The value of t_w is determined once t_c is specified and the former is not considered a design variable.
3. Processors: The cost of the processor is dependent on the many different speeds associated with its internal operations. Once an instruc-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

tion mix has been specified a single value t_p is determined by equation 4.2 . The cost of the processors is then n x $C_p(t_p)$.

The above relations allow the cost C of the multiprocessor computer to be expressed as:

$$C = m \times C_m(w/m,\ t_c(t_w)) + n \times C_p(t_p) + m \times n \times C_s \qquad (7.1)$$

The performance of the multiprocessor computer is specified by equation (4.11),(4.19) or (4.22) depending on the relationship of t_p to t_w. We symbolically include all three equations in the following:

$$UER = UER(t_p,\ t_w,\ t_c,\ m,\ n). \qquad (7.2)$$

We now state the optimization problem as:

minimize C

such that $UER \geq UER_{required}$

where C and UER are specified by equations 7.1 and 7.2. To this other constraints may be added; for example, one limiting the number of processors or stating that the number of processors must be greater than two.

The approach we have taken to solve the optimization problem is an exhaustive search over the possible values (as tabulated) of t_p and t_c (implying t_w) and over a specified set of values for m and n. A search space of no more than 10^4 points exists if we assume about 5 to 10 values for each of the variables.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

An Algol program was written for the Univac 1108 to evaluate equations 7.1 and 7.2 over a specified set of values of t_p, t_c, m, and n and pick the optimum. Despite the fact that the exhaustive search approach lacks sophistication (although it is difficult to think of other tecniques that could be used) it has the definite advantage that all potential structures are evaluated. Furthermore, the search is carried out sufficiently rapidly (about 0.2 sec./100 structures) that there is little incentive to consider other methods. Thus, in addition to choosing the optimum structure, the costs and performances of the suboptimal structures are also available and it is interesting to group them according to their performance and the constraints violated. It is always important to the designer to know what the sensitivity of a proposed design is to the design constraints and objectives; that is, how the design would change if the constraints and objectives were altered somewhat. This is readily determined if an evaluation of all potential structures in the design space is available.

We should that the above formulation of the optimization problem is not the only one possible. A design goal might be to design a system that has the maximum UER possible but does not exceed a cost $C_{max.}$. Another design goal might be to design a system which has the minimum cost/performance ratio (the cost of executing an instruction per unit time is a minimum). The reformulation of the optimization problem to handle these cases is per-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

fectly straightforward. In the subsequent sections we present two examples: one minimizes the system cost for a given UER; the other minimizes the cost/performance ratio.

C. Example 1: Minimization of System Cost

For this example we have the components available as listed in table 1. The design objectives and constraints are:

1. The UER must equal or exceed 10^6 instructions per second.
2. The total memory size must be 64 K words.
3. The number of processors must not exceed four.
4. The total system cost must be minimized.

The component costs are also indicated in table 1. They were chosen rather arbitrarily but they are probably not unrepresentative for memories in the 18 to 24 bit per word size and the related processors.

There are over 100 configurations which meet constaints two and three; 49 meet constraint one and of these four are presented in table 2. The optimum design is indicated by an asterisk; it is a three processor system. The other designs presented are the best (in terms of cost/performance ratio) using one, two, and three processors. Overall, the best cost/performance ratio is found in the four processor system. The best single processor system

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Memory:

Size	1.0 usec.	2.0 usec.	4.0 usec.
4K	$4000	$3000	$2000
8K	7000	5000	3000
16K	10000	7000	4000

Processor:

0.5 usec.	1.0 usec.	2.0 usec.
$50000	$20000	$10000

Switch:

$500/connection

Table 1. Example 1 Costs

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Design	m	n	t_c	t_p	UER	Cost	Cost/UER
1	4	3	1.0	2.0	1.15	76000	6.61 *
2	4	2	1.0	1.0	1.25	84000	6.72
3	4	4	1.0	2.0	1.49	88000	5.91
4	4	1	1.0	0.5	1.00	92000	9.20

Units:
t_c, t_p -- usec.
UER -- 10^6 instructions/sec.
Cost -- $
Cost/UER -- $10^{-2}$$/instruction/sec.

Table 2. Example 1 Designs

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

costs appreciably more than the best multiprocessor system and has an appreciably higher cost/performance ratio.

D. Example 2: Minimization of Cost/Performance Ratio

For this example we use cost data from a real computer system: the Digital Equipment Corporation PDP-10. This is a 36 bit word, single address instruction format computer which has facilities that enable it to be used in a multiprocessor configuration. The components available to build PDP-10 systems and their related costs are given in table 3. In order to put the example in realistic terms, we will deal with the actual IER of a typical but simplified instruction mix. The mix chosen has a scientific computational bias: 20% floating point multiply, 30% fixed point add, 20% branch, and 30% load/store. The branch instruction is not in the single address format: it has no operand reference. The PDP-10 System Manual (Digital Equipment Corporation, 1968) gives a rather elaborate breakdown of the processor execution times (as distinguished from instruction execution times) for the various instructions and the value of t_p for the mix above is computed to be 1.08 usec. Taking in to account the branch instruction, there are an average of 1.8 memory references per instruction; hence the actual IER = UER/1.8.

The design objectives and constraints are as follows:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Memory:

Size	1.0 usec.	1.8 usec.
8K	$40000	
16K	51000	
32K		$70000
64K		112000
128K		196000

Processor:

1.08 usec.

$151000

Switch:

$1500/connection

Table 3. PDP-10 Costs

(Costs obtained from "PDP-10 Pricing Summary", Digital Equipment Corporation,Maynard, Mass., March 30, 1969)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1. The main memory size must be 256K words.
2. The IER must equal or exceed 0.5×10^6 instructions per second.
3. The number of processors must not exceed four.
4. The cost/performance ratio must be minimized.

The eleven system configurations which meet constraints one, two, and three are given in table 4. The system which meets objective four is indicated by an asterisk. As in example 1, the best performance/cost ratio is obtained with four processors. Objective two is such that it cannot be met with less than two processors; the best two processor organization has a cost/performance ratio about 20% higher than the four processor organization.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Design	m	n	t_c	IER	Cost	Cost/IER
1	16	2	1.0	0.67	1.12	1.67
2	4	2	1.8	0.55	0.75	1.38
3	8	2	1.8	0.58	0.86	1.48
4	16	3	1.0	0.98	1.27	1.29
5	2	3	1.8	0.54	0.85	1.56
6	4	3	1.8	0.72	0.90	1.26
7	8	3	1.8	0.82	1.02	1.23
8	16	4	1.0	1.29	1.42	1.11 *
9	2	4	1.8	0.58	1.00	1.72
10	4	4	1.8	0.85	1.06	1.24
11	8	4	1.8	1.03	1.17	1.13

Units:
t_c -- usec.
IER -- 10^6 instructions/sec.
Cost -- $10^6$$
Cost/IER -- $/instruction/sec.

Table 4. PDP-10 Designs

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter VIII Conclusion

A. Summary

In the preceeding chapters we have presented a series of analytic models which quantitatively relate the performance of a certain class of computer structures to the basic component variables. We have corroborated the analysis with simulation studies and used it in a simple automatic design program. As stated in chapter I, a major goal of this thesis is to derive analytic models whose use would facilitate the design of digital computers. The question now arises as to what extent the analysis is actually useful in the computer design process and what extentions, if any, might be made to make it more useful. The answer is suggested by a review of the design program of chapter VII. In example 1 (of chapter VII) there are over 100 potential computer structures which meet the processor and memory constraints. In less than 0.2 seconds of Univac 1108 computer time these structures were determined, their performance and costs evaluated, and the optimum structure picked. The evaluations of the sub-optimal structures allow us to interact with the design process in the sense that we can see how the optimal structure changes if the design objectives or constraints are changed somewhat.

For the computer designer this type of activity is an economical tool to generate an initial set of design

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

alternatives. At the present state of development, the design program clearly does not design computers, but it does make some preliminary steps toward that objective. The generation of initial design structures and the insight gained from the models into how those structures behave certainly is valuable to the computer designer and hence this thesis has probably succeeded in its objectives. There does remain, however, a most interesting prospect for further automation of the computer design process and it is felt that with some extentions of the models and the design program that this prospect can be realized.

B. Extentions of the Models

In chapter II it was assumed that all instructions and operands occupied one memory word and that instructions were of the single address format type. In most real comuters such a situation does not exist. Normally there are at least three different operand formats: fixed point numbers, floating point numbers, and symbols (character strings); and there are multiple instruction formats. In general, each of the operand and instruction formats is of a different size. Since the primary memory word size is fixed, efficient utilization of the memory requires that one or both of the following techniques be employed: (1) pack more than one instruction or operand in a memory word or (2) use more than one memory word to hold an instruction or operand. Technique two usually slows down the IER re-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

lative to the case where there is one instruction (or operand) per memory word since more than two memory references are required to execute an instruction. Technique one may actually result in an increased IER. If two instructions are packed in a memory word, the processor can obtain the current and succeeding instructions with just one memory reference.

As an illustration of the foregoing consider a computer which has the following formats:

1. 24 bit instructions
2. 24 bit fixed point numbers
3. 48 bit floating point numbers
4. 8 bit characters.

The main memory word size for this computer might be 48, 24, or even eight bits. The choice depends on the relative use of the various formats and the level of performance desired. A high performance computer with heavy floating point usage would obviously have a 48 bit word size while a lower performance computer primarily manipulating characters would probably have an eight or a 24 bit memory word size.

The foregoing type of considerations represent a major activity in the computer design process. The design program of chapter VII could be extended to handle considerations of this type if the appropriate input information were available. This information would include the relative

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

usage of the various formats and their sizes. Various primary memory sizes could be tried by the design program and for each size a relationship r between the IER and UER would be defined (such that r x IER = UER) as well as a value of t_p. These definitions would essentially be a formalization of the ideas that were discussed in section B of chapter IV. The value of t_p would be defined as the average amount of processor time per memory reference and it is a function not only of the processor speed but also of the memory organization and the instruction and operand formats. The value of r would be defined as the average number of memory references made to execute an instruction and it is also a function of the way operands and data are mapped into the memory. With both r and t_p defined the design program would be essentially as that in chapter VII.

C. A Proposal for Continued Work

It is proposed that a design program be implemented which would specify the high level structure for a computer so that a desired IER is realized and that the cost of the design is a minimum. (Other design objectives and constraints involving costs and performance could of course be used.) Such a program would undoubtedly be interactive so that the designer could see the effect of varying the input design objectives and constraints. The inputs to the design program would be the following:

1. The component costs

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2. The desired IER and other constraints (such as total memory size, limits on the number of processors, and so forth)(The constraints might be specified in complex ways. The total memory size might be made a function of the number of processors with the memory size increasing as the number of processors increase.)

3. The instruction and operand formats and their associated relative frequencies

4. The i/o activity (possibly as a function of the total memory size and the number of processors) (Indeed if the i/o activity were known as a function of memory size, the memory size might be made a design variable rather than a constraint).

The design program would then pick the number of memories and processors, the memory word size (and possibly the total memory size), and the memory and processor speeds. Since this design program essentially includes the instruction set as one of its inputs, it might be possible to interconnect it with a design program which specifies instruction sets (Haney, 1968) thus potentially extending design automation to cover several levels of the computer design process.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

References

1. D. W. Anderson, F. J. Sparacio, and R. M. Tomasulo, "Machine Philosophy and Instruction Handling," IBM J. Research and Development, vol. 11, no. 1, pp. 25-33, January, 1967.

2. G. H. Barnes, et al., "The Illiac IV Computer," IEEE Trans. Computers, vol. C-17, no. 8, pp. 746-757, August, 1968.

3. Digital Equipment Corp., PDP-10 System Reference Manual, Maynard, Mass., 1968.

4. F. Haney, "Using a Computer to Design Computer Instruction Sets," Ph.D. Thesis, Computer Science Department, Carnegie Mellon University, 1968.

5. F. S. Hillier and G. J. Lieberman, Introduction to Operations Research, ch. 10, San Francisco: Holden-Day, 1967.

6. W. Feller, An Introduction to Probability Theory and its Applications, vol. I, third edition, ch. 2, New York: Wiley, 1968.

7. I. Flores, "Derivation of a Waiting-Time Factor for a Multiple Bank Memory," ACM J., vol. 11, no. 3, pp. 265-282, July, 1964.

8. J. B. Kruskal, "Extremely Portable Random Number Generator," ACM Communications, vol. 12, no. 2, pp. 93-94, February, 1969.

9. D. A. Pierre, Optimization Theory with Applications, ch. 6, New York: Wiley, 1969.

10. J. Riordan, An Introduction to Combinatorial Analysis, ch. 5, New York, Wiley, 1958.

11. J. L. Rosenfeld, "A Case Study in Programming Parallel Processors," ACM Communications, vol. 12, no. 12, pp. 645-655, December, 1969.

12. J. E. Shemer and S. C. Gupta, "A Simplified Analysis of Processor Look Ahead and Simultaneous Operation of a Multimodule Main Memory," IEEE Trans. Computers, vol. C-18, no. 1, January, 1969.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

13. C. E. Skinner and J. R. Asher, "Effect of Storage Contention on System Performance," IBM Systems J., vol. 8, no. 4, pp. 319-333, 1969.

14. J. E. Thornton, Design of a Digital Computer - The Control Data 6600, Glenview, Illinois: Scott, Foresman, and Co., 1970.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Bibliography

1. W. Bucholz, Planning a Computer System, New York: McGraw-Hill, 1962.

2. H. Hellerman, Digital Computer Systems Principles, New York: McGraw-Hill, 1967.

3. R. S. Ledley, Digital Computer and Control Engineering, New York: McGraw-Hill, 1960.

4. T. L. Saaty, Elements of Queueing Theory, New York: McGraw-Hill, 1961.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix

The simulator handles three classes of instructions:

1. Class 1: Single address format
2. Class 2: Instruction without operand reference (like a unit instruction)
3. Class 3: Write instructions (no processor execution time and an operand memory reference time of t_w).

An instruction execution consits of two phases: (1) the instruction reference and decode and (2) the operand reference and execute. Class two instructions have no phase two; for them the execute is accomplished in phase one. In the simulator each phase is handled as the execution of a unit instruction.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
@ ALG SIMS
COMPILED BY CMU 1108 ALGOL  DATED   31 JULY 69 (VERSION 10A)
THIS COMPILATION WAS DONE ON 22 MAY 70 AT 15:25:09
            1       BEGIN
BLOCK 1      LEVEL 1
            2       FORMAT F1(' N',X5,'N',X5,'TC',X6,'T#',X6,'TP',
            3       X7,'R',X6,'RP',X7,'P/RP',A1.2)$
            4       FORMAT F2(2(I3,X3),6(D5.2,X3),A1.1)$
            5       INTEGER NI,K,W,I,J,L,R,S,Z,Y,M,N,CC$
            6       REAL D,A,Q,TW,TA,TC,TD,TTP,V,X,H,T,R0,RP$
            7       READ(NI)$
            8       Z=8192$
            9       Y=2048$
           10           BEGIN
           11       COMMENT IC IS THE INSTRUCTION CLASS, FI IS THE
           12       INSTRUCTION RELATIVE FREQUENCY, AND IT IS THE
           13       PROCESSOR EXECUTION TIMES
BLOCK 2      LEVEL 2
           14           INTEGER ARRAY IC(1:,NI)$
           15           REAL ARRAY CP,FI,IT(1:,NI)$
           16       COMMENT READ IN THE INSTRUCTION SET VARIABLES$
           17           FOR K=(1,1,NI) DO
           18                BEGIN
           19                READ(IC(K),IT(K),FI(K))$
           20                A=A+FI(K)$
           21                END$
           22       COMMENT NORMALIZE RELATIVE FREQUENCIES AND
           23       COMPUTE CUMULATIVE PROBABILITIES$
           24           CP(1)=FI(1)=FI(1)/A$
           25           FOR K=(2,1,NI) DO
           26                BEGIN
           27                FI(K)=FI(K)/A$
           28                CP(K)=CP(K-1)+FI(K)$
           29                END$
           30       COMMENT U IS THE NUMBER OF SIMULATIONS TO BE
           31       RUNS
           32           READ(U)$
           33       COMMENT READ MEMORY RESTORE,MEMORY CYCLE AND
           34       INSTRUCTION DECODE TIMES$
           35           READ(TW,TC,TD)$
           36           TA=TC-TW$
           37           WRITE(F1)$
           38           FOR K=(1,1,U) DO
           39                BEGIN
           40                READ(N,N)$
           41                     BEGIN
BLOCK 3      LEVEL 3
           42                     INTEGER ARRAY P,C(1:,N)$
           43                     REAL ARRAY TT,TP(1:,N),TM(1:,K)$
           44                     FOR L=(1,1,N) DO P(L)=1$
           45                     R=S=1=1$
           46       COMMENT CC COUNTS THE NUMBER OF UNIT
           47       INSTRUCTIONS EXECUTED$
           48                     FOR CC=(1,1,Y) DO
           49                          BEGIN
           50       COMMENT CHOOSE A MEMORY -- DETERMINE J$
           51                     FOR L=1,2,3 DO R=MOD(5*R,Z)$
           52                     J=(R*M)//Z+1$
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
53      COMMENT CHOOSE A PROCESSOR -- DETERMINE IS
54                      FOR L=(1,1,N) DO IF TP(L)
55                      LSS TP(I) THEN I=L$
56      COMMENT IF PHASE ONE CHOOSE A NEW INSTRUCTION.
57      GENERATE A RANDOM NUMBER IN (0,1) AND CHOOSE
58      INSTRUCTION WHOSE CUMMULATIVE PROBABILITY FIRST
59      EXCEEDS THAT NUMBER$
60                      IF P(I) EQL 1 THEN
61                          BEGIN
62                          FOR L=1,2,3 DO S=MOD(S*S,Z)$
63                          Q=S/Z$
64                          L=1$
65                          FOR L=L WHILE Q GTR CP(L)
66                          DO L=L+1$
67                          C(I)=IC(L)$
68      COMMENT TT=PROCESSOR EXECUTION TIME FOR THE
69      SELECTED INSTRUCTION.  UPDATE MEMORY AND
70      PROCESSOR TIMES (INSTRUCTION REFERENCE PHASE)$
71                          TT(I)=IT(L)$
72                          TM(J)=MAX(TP(I),TM(J))+TC$
73                          TP(I)=TM(J)-TW+TD$
74      COMMENT IF INSTRUCTION I IS OF CLASS 2 NO
75      ADDITIONAL MEMORY REFERENCES NEED BE MADE.
76      UPDATE PROCESSOR TIME BY EXECUTE TIME.  PHASE
77      REMAINS ONE SO THAT A NEW INSTRUCTION IS CHOSEN
78      FOR PROCESSOR I.  IF INSTRUCTION IS NOT OF
79      CLASS 2, THE PHASE IS SET TO 2$
80                          IF C(I) EQL 2 THEN TP(I)=
81                          TP(I)+TT(I) ELSE P(I)=2$
82                          END ELSE
83                          BEGIN
84                          P(I)=1$
85      COMMENT UPDATE MEMORY AND PROCESSOR TIMES
86      (INSTRUCTION EXECUTION PHASE)$
87                          IF C(I) EQL 1 THEN
88                              BEGIN
89                              TM(J)=MAX(TM(J),TP(I))
90                              +TC$
91                              TP(I)=TM(J)-TW+TT(I)$
92                              END ELSE
93                          TM(J)=MAX(TM(J),TP(I))+TW$
94                          END$
95                      END$
96                      A=0$
97                      FOR L=(1,1,N) DO A=A+TM(L)$
98                      FOR L=(1,1,N) DO A=A+TP(L)$
99      COMMENT COMPUTE RO THE OBSERVED RATE OF
100     INSTRUCTION EXECUTIONS
101                     RO=Y*(M+N)/A$
102                     A=0$
103                     FOR L=(1,1,NI) DO A=A+IT(L)$
104                     TTP=(TD+A/NI)/2$
105     COMMENT COMPUTE THE AVERAGE PROCESSOR ACTIVTY
106     TIME.  FORM THE DIFFERENCE OF TTP AND TW$
107     COMMENT IF D IS LESS THAN OR EQUAL TO ZERO
108     COMPUTE RP THE PREDICTED RATE OF INSTRUCTION
109     EXECUTIONS
110                     D=TTP-TW$
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
111                         IF D LEQ 0 THEN
112                           BEGIN
113                           V=(1-1/M)**NS
114                           RP=M*(1-V)/(TC+V*D)S
115                           END ELSE
116                           BEGIN
117         COMMENT SUBSTITUTE II EQUATION FOR THE
118         EXECUTION RATE WHERE TP IS GREATER THAN TW THE
119         RELATION X=(1-PM/M)**N,  THEN USE NEWTON-
120         RAPHSON SEARCH TO FIND X.  THE STARTING VALUE
121         OF X IS ONE.  THE SEARCH STOPS WHEN SUCCESSIVE
122         ITERATIONS ON X DIFFER BY LESS THAN .001S
123                           X=1S
124                           T=D/(N*TC)S
125                           FOR H=(T*X**N+X-1+1/M-T)/
126                           (N*X**(N-1)+1)
127                           WHILE H GTR 0.001 DO X=X-HS
128                           RP=M*(1-X**N)/TCS
129                           ENDS
130                         WRITE(F2,M,N,TC,TW,TTP,RO,RP,
131                         RO/RP)S
          END BLOCK 3
132                     ENDS
133                   ENDS
          END BLOCK 2
134               ENDS
          END BLOCK 1
135         ENDS
 COMPILATION COMPLETE
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.