24012

National Library of Canada / Bibliothèque nationale du Canada

CANADIAN THESES ON MICROFICHE / THÈSES CANADIENNES SUR MICROFICHE

NAME OF AUTHOR/*NOM DE L'AUTEUR* Domaschenko John

TITLE OF THESIS/*TITRE DE LA THÈSE* An Implementation Study On Microprogram Optimization

UNIVERSITY/*UNIVERSITÉ* U. of Alberta

DEGREE FOR WHICH THESIS WAS PRESENTED/*GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE* M. Sc.

YEAR THIS DEGREE CONFERRED/*ANNÉE D'OBTENTION DE CE GRADE* 1975 (Spring)

NAME OF SUPERVISOR/*NOM DU DIRECTEUR DE THÈSE* Dr. J. Tartar

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

DATED/*DATÉ* May 2/75 SIGNED/*SIGNÉ* [signature]

PERMANENT ADDRESS/*RÉSIDENCE FIXE* 246 20th AVE N.E.
Calgary Alta
T2E 1P3

THE UNIVERSITY OF ALBERTA

AN IMPLEMENTATION STUDY ON MICROPROGRAM OPTIMIZATION

by

© JOHN DOMASCHENKO

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

SPRING, 1975

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled "An Implementation Study On Microprogram Optimization," submitted by John Domaschenko in partial fulfilment of the requirements for the degree of Master of Science.

.....

Supervisor

.....

.....

DATE 28 April 1975

To Ruth

ABSTRACT

An implementation study on the cptimization of sequentially executable microprogram segments is discussed. This is a practical extension of earlier work presented in a PhD Thesis by W. G. Sitton. Optimization in this context means the deletion of a maximum number of nonessential micro-operations. These nonessential micrc-operations are identified by the constfuction of all alternate forms of each micro-operation.

Theorems originally presented by Sitton have been extended, transformed into algorithms, and verified by their application on microprogram examples. Data structures have been developed in conjunction with the requirements of the algorithms developed. These algorithms have been designed to accept general machine architecture and language definitions in order not tc restrict the application of the implementation. The problems and peculiarities encountered during the development of the implementation will be discussed and some examples will be presented.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Professor John Tartar, my supervisor, for his advice, criticism, and guidance throughout the duration of the research and the preparation of this thesis.

I am also grateful to the Department of Computing Science staff who have made my stay at The University of Alberta most enjoyable.

Finally, I would like to thank Ms. Ruth Hendricks for her faithful support during the preparation of this thesis.

Table Of Contents

Table Of Contents (Cont'd)

Table Of Contents (Cont'd)

Table Of Contents (Cont'd)

Table Of Contents (Cont'd)

Table Of Contents (Cont'd)

Table Of Contents (Cont'd)

List Of Figures

List Of Figures (Cont'd)

CHAPTER I

Introduction

The subject of this thesis is an implementation study on the optimization of microprogram segments consisting of sequentially executable micro-operations. The purpose was to investigate the practical implications of optimization strategies which have been theoretically developed in Sitton[S1]. Optimization techniques, in general, can be applied to almost every area of software research [A2, G1, L1, S1, T3]. Within this study, the term is used to mean the techniques developed to optimize microprogram segments.

The areas of software development and software optimization are relatively unexplored and unformalized, as has been pointed out in Agrawala and Rauscher[A1] by the statement:

> "While hardware developments for supporting microprogramming have been dramatic, developments in software and applications aspects of microprogramming are still in their infancy. Programming language techniques, such as intermediate language design, interpretation, and optimization need to be applied and extended. Application areas, somewhat hampered by lack of software support, need to adapt microprogramming to specific problems to realize this full potential of microprogramming."

Basically, the research is involved with the implementation and study of a program which finds and deletes nonessential operations from a set of sequentially executable micro-operations. In order to accomplish this study, theorems and theoretical algorithms have been extended, transformed intc applicable algorithms, and implemented for the purpose of practical application and verification.

Previous attempts at the deletion of nonessential operations have relied upon an evaluation of only the source form of each operation. However, the deletion of all nonessential operations requires the identification and evaluation of each alternate operation form. The significance of the research is based on the premise that nonessential operations are most likely to occur in operation forms which are alternate and equivalent to the source forms.

In general, nonessential micro-operations have been divided into the three classes: negated, redundant, and parallel [K2, S1]. The identification of nonessential operations involves the recognition of alternate and equivalent operation forms. In this study, the identification is done by the analysis of equivalently defined

memory units. It is possible to force operations to assume nonessential status upon meeting a minimal set of conditions applied to operations independently. This is accomplished through the assignment of alternate source and sink units for each operation.

With the application of an alternate unit identification algorithm, an alternate unit assignment table is constructed in order to represent all possible forms of each operation. This is accomplished by specifying all prime and alternate source and sink units. Information on how units are utilized within the given microprogram segment, and how they will be used after the given segment, is also stored in tabular form within a unit assignment table.

By using the accumulated information on all operation forms and other available information, all initial operation deletion candidates are identified. This process of recognizing the operation deletion candidates involves the generation of one or more operation deletion strategies. This is necessary for each operation deletion candidate. They are later evaluated independently as feasible or not.

The approach used to evaluate each operation deletion strategy is: first, to determine at which locations other strategies are required; second, to determine the conflict

relaticnships with other strategies; and third, to determine which strategies may be used to alter each location or point. Informaticn cbtained upon the evaluation of strategies is stored in tabular form. This is done in order to represent all feasible operation deletion strategies, all feasible alternate source unit assignment strategies, strategy conflict relationships, and all locations which require a strategy.

Once the evaluation of strategies is complete, the problem is transformed into one of optimal selection of operation deletion strategy and alternate source unit assignment strategy sets. Each set if applied, will result in the deletion of a maximum number of operations. The outcome of this final phase is a set of optimized microprogram segment versions. These versions are logically correct with respect to the given microprogram segment.

The subject matter discussed above is organized into eight chapters. Chapter II introduces the prcblem for which the program has been implemented. Chapter III describes the general solution framework followed by the program and existing limitations. Chapters IV through VII describe each major phase of the solution framework in detail. Finally,

Chapter VIII summarizes the study and makes recommendations for further research. Chapter VIII is followed by an Appendix of optimization examples generated by the implemented program.

CHAPTER II

Problem Analysis

2.1 Introduction

The objective of this chapter is to describe the problem for which a program has been implemented. Also, it is to introduce related information and definitions referred to throughout this study. The general order of entries consist of: first, a description of the general representation of microprograms; second, a description of the problem; third, the range of optimization; fourth, related information required for the definition of nonessential operations; and fifth, definitions leading to the description of nonessential operations.

2.2 Microprogram Representation

2.2.1 Microprogram Notation

Following the definitions given in Sitton[S1] and later used in Jackson and Dasgupta[J1], the notation for the general representation of microprograms will now be

described and used throughout this study. This notation is very general in order to prevent restrictions when transition is made to or from different degrees of encoding and/or parallelism.

Basically, a microprogram is an ordered set of micro-instructions, each of which is to be executed for a particular time period. Each microinstruction in turn consists of an ordered set of micro-operations which are invoked during the execution of the microinstruction.

Definition 2.1: A Source Unit (SC) is a memory unit used within a micro-operation as data input.

Definition 2.2: A Sink Unit (SK) is a memory unit used within a micro-operation as data output.

Definition 2.3: A Micro-Operation (MO) is a 6-tuple (OP, SC, SK, C, CU, TV)

where

OP identifies the primitive (hardware) operation, e.g. ADD, SUB, GATE, etc;

SC is a set of units used as the data source for 'OP', e.g. Registers;

SK is a set of units used as the data sink for 'OP';

C is a set of control information qualifying 'OP', e.g. Shift Amounts, etc;

OU is the set of operational unit(s) required to execute 'OP', e.g. Adder, Shifter, etc;

TV denotes the time validity set for executing 'OP' using 'SC', 'SK', 'C' and 'OU'.

Each micro-operation is distinctly identified in order, by a two part index I, I = (j,k). The sub-indices are interpreted as the micro-operation, MO(k), contained within the particular microinstruction, MI(j). In order to simplify discussion, a one part index I, where I is an integer, will be used throughout this study. Transition from one form to the other is trivial.

The information sets 'OU' and 'TV' are not used in the optimization of microprogram segments. It will be assumed throughout this study, that a given microprogram is ordered logically as well with respect to the 'OU' and 'TV' sets. That is to say, the 'OU' sets are not used in the recognition of equivalent micro-operations. Also, corrections are not made to the assignment of operational units, nor to the time validity sets when operations are deleted, or when unit definitions are moved. The author has come to learn that such corrections and alterations should be made at the time of micro-code generation for a completely optimized microprogram. At this time, operation units can be assigned optimally and allowances made for the

timing restrictions. Code compaction, parallelism constraints, etc. will further affect the 'OU' and 'TV' sets.

The following notation will be used when referring to particular source and sink unit(s):

<u. type> (<op. id.>),

<u. type> (<op. id.> , <comb. id.>), or

<u. type> (<op. id.> , <comb. id.> , <u. index>)

where

<u. type> is either 'SC' or 'SK', indicating the source or sink set;

<op. id.> identifies the particular micro-operation being examined;

<comb. id.> identifies the particular combination (This will later be clarified through examples. If absent, the notation is to be interpreted as the set of all possible source or sink unit combinations for the particular micro-operation);

<u. index> identifies the <u. index>th unit within the particular combination referred to.

Within this study, the notation |S| should be interpreted as the number of elements within the set S.

In order to clarify the given notation, consider the following pair of microinstructions:

1). R3:=R1+R2, R4:=R2.

2). R4:=R3+R4.

Figure 2.1 Microprogram example

During the execution of the above code, MI(1) results in the addition of registers R1 and R2. The result is gated to register R3, and there is a simultanecus gating of the contents of register R2 to register R4. Then MI(2) results in the addition of registers R3 and R4 with the result gated to register R4.

The code of Figure 2.1 translated into the given notation, 'OU', 'C', and 'TV' sets ignored, would have the form fcllowing in Figure 2.2.

Index	OP	SC	SK
1	ADD	{R1,R2}	R3
2	GATE	R2	R4
3	ADD	{R3,R4}	R4

Figure 2.2 Example of the general notation using the microprogram of Figure 2.1

Further reviewing the notation:

CF(1) = ADD [operation code for MO(1)].

SC(1) = {R1,R2} [all source unit combinations for MO(1)].

SC(1,1,2) = R2 [second source unit within the first source unit combination of MC(1)].

|SC(1)| = 1 [number of source unit combinations for MO(1)].

|SC(1,1)| = 2 [number of source units within the first source unit combinatien of MO(1)].

By examining Figure 2.2 and observing that units R2 and R4 contain the same value after MO(2) is executed, two possible source unit combinations for MO(3) are obtained.

SC(3) = {R3,R4}, {R3,R2} [the set of all source unit combinations for MO(3)].

SC(3,1,1) = R3 [the first unit within the first source unit combination of MO(3)].

|SC(3)| = 2 [two possible source unit combinations for MO(3)].

|SC(3,1)| = 2 [two source units within the first combination of source units for MO(3).].

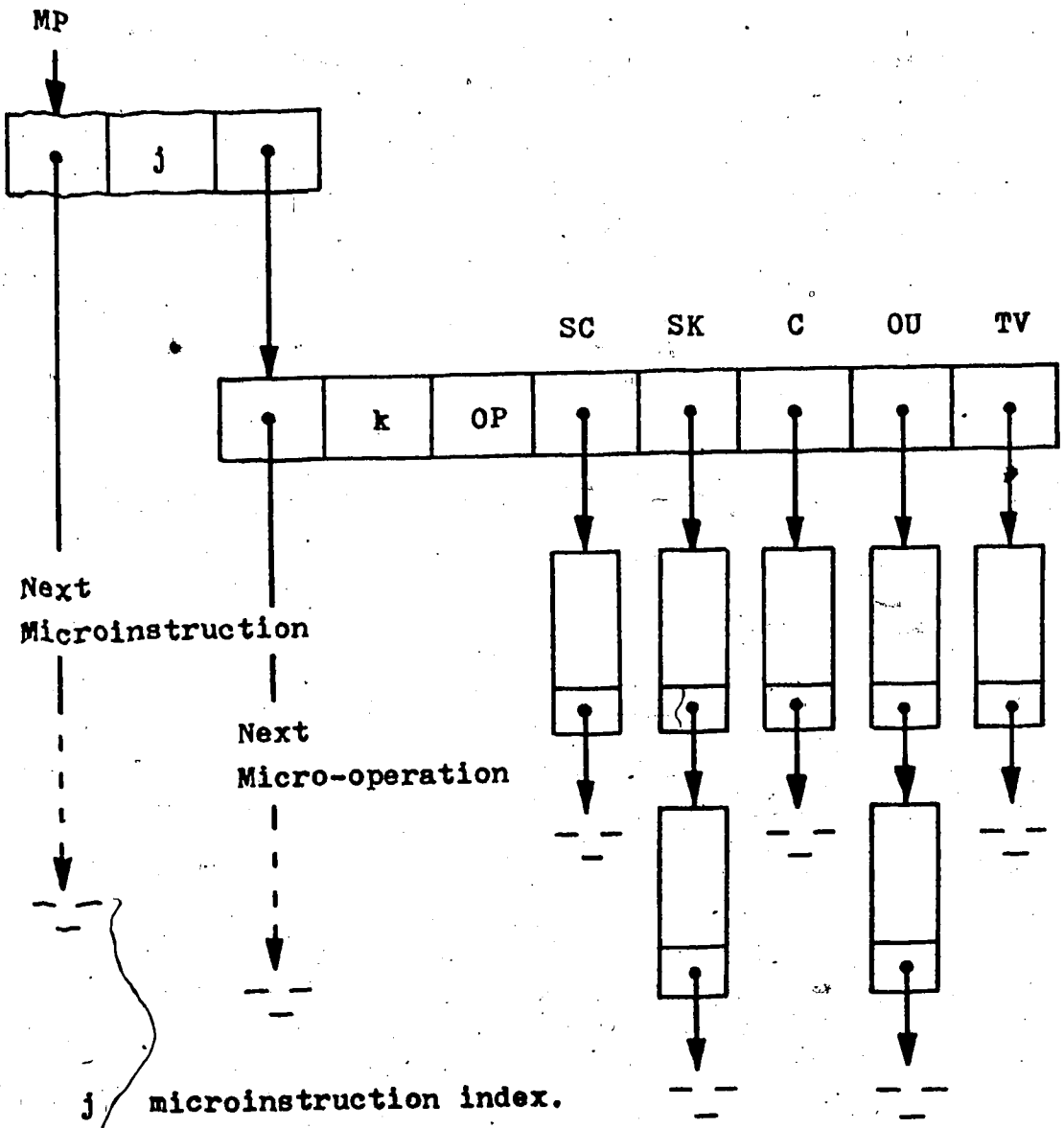
2.2.2 Implementation of Microprogram Representation

The main objective in the design of the internal representation was to keep the internal organization and representation of a microprogram as general as possible. The internal form was designed as a list of lists

$$MP = mi_1, mi_2, \ldots, mi_n,$$

where each sub-list, mi_j, is a record representing a microinstruction. Each record was designed to contain: a reference to the next microinstruction record, a microinstruction index, and a reference to a list of micro-operation records. Each record representing a micro-operation was designed to consist of: a reference to the next micro-operation record, a micro-operation index, a

micro-operation code, and references to string vectors of information representing the sets as described in definition 2.3.



j microinstruction index.
k micro-operation index.
OP operation code.
SC source unit set.
SK sink unit set.
C control information set.
OU operational unit set.
TV time validity information set.

Figure 2.3 Microprogram implementation

2.3 Problem Description

Basically, as defined in Sitton[S1], the problem for which the program has been implemented is that of identifying and deleting nonessential operations by identifying and evaluating all possible micro-operation forms. The objective is to derive a set of optimized micro-program segment versions which are logically correct with respect to the original microprogram segment. For example, consider the following four sequentially executable micro-operations:

1	ADD	{1,2}	3
2	GATE	2	4
3	ADD	{1,4}	3
4	SUB	{5,3}	5

Figure 2.4 Microprogram segment example

Examination of only the source form of each micro-operation yields no nonessential operations. Yet, evaluation of equivalent units (i.e., by observing the gating of the contents of register 2 into register 4) yields another form for MC(3) as follows:

```
ADD     {1,2}     3
```

Figure 2.5 Alternate operation form example

The prior form is nonessential with respect to MO(1). This generally makes it possible to optimize the code in Figure 2.4 to the following two micro-operations in Figure 2.6.

```
1     ADD     {1,2}     3
4     SUB     {5,3}     5
```

Figure 2.6 Optimized version of microprogram segment given in Figure 2.4

Naturally it is assumed that only the contents of registers 3 and 5 are later required.

The significance of this problem has been stated as being: nonessential operations are most likely to occur from operation forms which are equivalent with respect to their 'OP', 'SC', and 'C' sets. Also, they are most likely to occur in forms which are alternate (i.e., give the same result).

2.4 Optimization Range

The search for nonessential operations is performed only in the given microprogram segment. This consists of operations which are sequentially executable, none are branch operations, and none of which are branched to from other regions except for the first operation.

2.5 Unit Assignment Table (UAT)

2.5.1 Introduction

In order to define nonessential operations, the unit assignment table will now be described. The purpose of this table is to store information on where and how each unit of a given microprogram segment is used, and how it will first be used within the following segment(s).

The unit assignment table will be at most an (Nop+1) by Nu array, where Nop equals the number of operations and Nu equals the number of units. The elements of rows 1 through Nop will contain one of the values: 0, if the corresponding unit is not used; 1, if used as a data source; 2, if used as a data sink; or 3, if it is used as a data source and as a data sink. Each element of the last row, (Nop+1), will contain the value 1 if the corresponding unit is first used as a data source; or 2, if first used as

a data sink after the given microprogram segment. The reason a 2 is used for the last row is to assure that if global optimization was applied, then the corresponding unit would not be chosen as an alternate source unit.

The unit assignment table is also used to identify non-essential operations within the given microprogram segment.

The fcllowing figures are given in order to clarify the organization of the unit assignment table.

1	ADD	{1,2}	3	
2	SUB	{3,4}	4	
3	GATE	3	5	
4	SHIFT	5	7	[4]
5	GATE	5	6	
6	AND	{6,1}	5	
7	ADD	{1,2}	3	
8	ADD	{3,1}	7	

Figure 2.7 Microprogram segment example

Presume that only the information within units 4, 5, and 7 are later required. Then the following UAT would be generated.

UNIT	1	2	3	4	5	6	7
MO(1)	1	1	2	0	0	0	0
MO(2)	0	0	1	3	0	0	0
MO(3)	0	0	1	0	2	0	0
MO(4)	0	0	0	0	1	0	2
MO(5)	0	0	0	0	1	2	0
MO(6)	1	0	0	0	2	1	0
MO(7)	1	1	2	0	0	0	0
MO(8)	1	0	1	0	0	0	2
EXIT	2	2	2	1	1	2	1

Figure 2.8 UAT of microprogram segment given in Figure 2.7

2.5.2 Generation of the UAT

The current program generates rows 1 through Nop on input of the given microprogram segment. Row (Nop+1) is then later generated using separate parameters on input.

Algorithm 2.A

The following four steps are executed in the generation of the UAT.

A.1 For each micro-operation i = 1, ..., Nop do steps A2

and A3.

A.2 For j:=1, ..., |SC(i,1)| do

UAT(i,SC(i,1,j)):=1.

A.3 For j:=1, ..., |SK(i,1)| do

If UAT(i,SK(i,1,j)) = 1 then

UAT(i,SK(i,1,j)):=3

else

UAT(i,SK(i,1,j)):=2.

A.4 Set each value in row (Nop+1) to 2. Then for each unit specified on input as required after the given microprcgram segment, set the corresponding value in row (Nop+1) to 1.

2.5.3 UAT Implementation

The UAT was implemented as a list of records. Each was designed to contain: a reference to the next UAT entry, a unit identifier, two fields for another table described later, and a reference to a vector of records. These each were designed to contain: a nonzero use value for each unit at the particular operation, an operation index, and a reference to the next such record. The reasons for this design were: to be able to index the UAT with nonnumeric unit identifiers, prevent the storage of zeros, and to merge the table with another described later.

Throughout this study data structures have been designed to be pseudo sparse, well ordered, and very flexible with respect to the ability to add information. The objective of this approach was to save space, save search time, and to take advantage of equivalent information. Also, it was to enhance the ability to add information for future improvements and extensions. The utilization of ALGOLW record and reference facilities enhanced the ability to use equivalent information by cross referencing. The following Figure 2.9 is given in crder to clarify the organization of the implemented UAT structure.

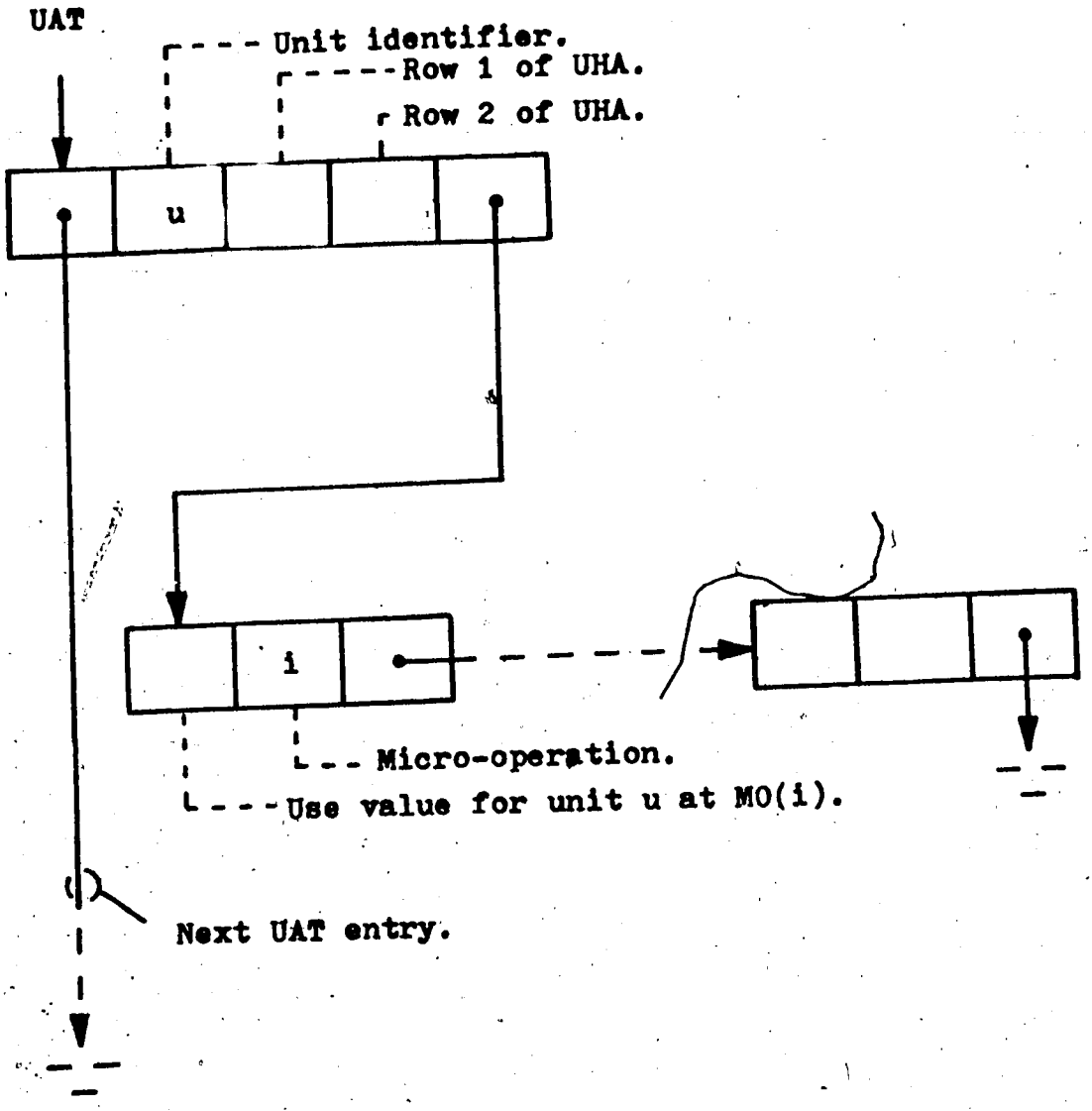


Figure 2.9 UAT implementation

2.6 Nonessential Operations

2.6.1 Introduction

Nonessential operations are basically those operations whose outputs are never used, or operations which are equivalent in result with respect to other operations within the given microprogram segment. In this study we are basically concerned with the three classes: negated-forward, parallel-forward, and parallel-backward. These classes may be defined in terms of the unit assignment table and equivalent relations.

2.6.2 Negated-Forward Operations

Basically, a MO(i) may be negated-forward, iff, each data sink unit of MO(i) is not used as a data source unit until after it is again used as a data sink.

Definition 2.4: MO(i) is a negated-forward deletion candidate if there exists a minimal set of micro-operations

$$M = \{ MO(k_1), MO(k_2), \ldots, MO(k_n) \}$$

such that

$$i < k_1 < k_2 \ldots < k_n$$

where for each sink unit $l = 1, \ldots, |SK(i,1)|$, there exists

a MO(j) contained within the set M, such that UAT(j,SK(i,1,1)) > 1.

The latter definition 2.4 defines the conditions which must be met for an operation to be a deletion candidate. In order for an operation to be a feasible deletion candidate such that it may be deleted, the following conditions in definition 2.5 must be met, or changes made such that they are met.

Definition 2.5: MO(i) is a negated-forward deletion candidate and may be deleted, iff, for each sink unit l = 1, ..., |SK(i,1)|, UAT(m,SK(i,1,l)) = 0, for all micro-operations m, i < m < n, and where MO(n) is the first micro-operation within the set M such that UAT(n,SK(i,1,l)) = 2.

For example, a MO(i) with two sink units, would be identified as a negated-forward deletion candidate if both units were found being redefined by either one or two of the following operations (i.e., the sink units are redefined by a minimum number of micro-operations). Furthermore, MO(i) would be a feasible deletion candidate if conditions exist, or were made such, that each sink unit was not used as a source unit until after it was redefined.

2.6.3 Parallel Operations

In order to define parallel operations, equivalent units and equivalent operations must first be defined as follows:

Definition 2.6: A QMSU (memory unit qualified by its index of definition) is an ordered pair, (u,oid), where u is the identifier of the unit, and oid is the index of the operation which defines unit u. For example, (R3,10), would qualify unit R3 to be defined by micro-operation 10. All units upon entry into a microprogram segment are qualified by the index of definition 0.

Definition 2.7: Unit i, defined at MO(s), is equivalent (<=>) to unit j, defined at MO(t), if their contents are equivalent; i.e.,

(i,s) <=> (j,t)

if MO(s) <=> MO(t).

In the following definition 2.8 the term, data-flow, should be interpreted as the transfer of information between units without any change made to the information itself.

Definition 2.8: MO(i) is equivalent (<=>) to MO(j) if their outputs are identical with respect to value; i.e.,

MO(i) <=> MO(j)

if OP(i) = OP(j), C(i) = C(j), and

for 1:=1, ..., max(|SC(i,1)|,|SC(j,1)|),
    SC(i,1,1) <=> SC(j,1,1);
or OP(i) = data-flow, C(i) = ø, and
for 1:=1, ..., max(|SK(i,1)|,|SK(j,1)|),
    SK(i,1,1) <=> SK(j,1,1);
or OP(j) = data-flow, C(j) = ø, and
for 1:=1, ..., max(|SK(i,1)|,|SK(j,1)|),
    SK(i,1,1) <=> SK(j,1,1).

2.6.3.1 Parallel-Forward Operations

Generally, MO(i) is a parallel-forward deletion candidate if all sink units of MO(i) may be moved forward to another equivalent micro-operation.

Definition 2.9: MO(i) is a parallel-forward deletion candidate with respect to MO(j) and may be deleted iff i < j;
MO(i) <=> MO(j);
for 1:=1, ..., |SK(i,1)|
    UAT(m,SK(i,1,1)) = 0 for all m, i < m < j, and
    UAT(j,SK(i,1,1)) = 0 or 2;
and either SK(i,1) may replace SK(j,1) at MO(j), or SK(i,1) can be fanned out at MO(j).

2.6.3.2 Parallel-Backward Operations

In general, MO(j) is a parallel-backward deletion candidate if all sink units of MO(j) may be moved back to another equivalent micrc-cperation.

Definition 2.10: MO(j) is a parallel-backward deletion candidate with respect to MO(i) and may be deleted iff i < j;

MO(i) <=> MO(j);

for l:=1, ..., |SK(j,1)|

UAT(m,SK(j,1,1)) = 0 or 1 for all m, i < m < j, and

UAT(j,SK(j,1,1)) = 2;

and either SK(j,1) may replace SK(i,1) at MC(i), or SK(j,1) can be fanned out at MO(i).

2.7 Summary

The major objective of this chapter has been to describe the problem for which a program has been implemented. Also, to introduce related information and definitions referred to throughout this study.

The negated-forward definitions given in section 2.6.2 are different from those found in Sitton[S1]. Sitton's definitions have been expanded to include multiple sink units. The difference is Sitton's definition refers to only

one micro-operation redefining the sink set which is correct for single sink unit operations. Yet, for multiple sink unit sets, it may be possible to redefine the sink set using several micro-operations as long as each sink unit is not used as a source unit until after it is redefined.

The data structures for the internal representation of microprograms and the UAT have also been discussed. They were developed and implemented using records and references available in ALGOLW. For the purpose of future extensions into global optimization, it may be of value to consider extending the structure and/or concept of the UAT. The extension would be to include fields in the UAT or to develop another structure, in order to indicate which units will be equivalent at the end of each optimized micro-program segment version. This may aid in the generation of alternate region forms by making available equivalence information.

The following chapter will describe the general framework of the solution developed and existing limitations.

CHAPTER III

Solution Framework

3.1 Introduction

The purpose of this chapter is to describe, in general, the solution framework for the problem outlined in Chapter 2. The limitations within the analysis of nonessential operations are also given.

3.2 General Solution Approach

Given a microprogram segment, a unit assignment table is generated. All equivalent units and operations are identified by the analysis and generation of all alternate operation forms of each micro-operation. The unit assignment table is then examined as well as all variables generated from the identification of equivalent operations. This is necessary in order to identify all strategies. Each strategy if applied, will result in the deletion of an operation.

Each strategy is then examined in order to determine: all conflicts with other strategies, the points requiring alteration for the strategy to be made applicable, and new strategies (i.e., Alternate Source Unit Assignment

Strategies) and other options (i.e., Operation Deletion Strategies) which may be used to alter each point. Those strategies requiring the alteration of points for which there exists no option, are deleted.

The final phase involves a search using the set of all applicable strategies. This search involves examining the conflicts, points, and options in order to determine all strategy sets which will result in the deletion of at least n operations, where n is an upper bound. The result is a set of optimized microprogram segment versions which are logically correct with respect to the given microprogram version.

3.3 Description of Solution Framework

3.3.1 Identification of Alternate Operation Forms

The identification of alternate operation forms is a two part process. The objective is the generation of an Alternate Unit Assignment Table (ALT). The ALT was designed to represent all possible operation forms of each operation within the given microprogram segment.

During the first part, two types of alternate source units and two types of alternate sink unit references are identified and stored into the ALT. The prime (i.e.,

given) source and sink units are also stored in this part. The identification of the alternate units is made as a result of one examination of each operation, the generation of alternate operation forms, and by using the prime and currently identified alternate source units. It should be noted that alternate sink units differ from the alternate source units in that they are actually references to equivalent micrc-operations.

In the second part, a third type of alternate source unit is identified and stcred into the ALT. This third type is identified by the examination of results in the ALT, indicating which cperations are equivalent. For if MO(i) <=> MO(j), then the sink set of MO(i) may replace the set at MO(j), or the sink set at MO(j) may replace the set at MO(i).

To be concise, equivalent sink units are combined to identify alternate source units. Prime and alternate source units are used to generate alternate operation forms which in turn are combined to generate equivalent sink units.

3.3.2 Identification of Operation Deletion Strategies

The identification of strategies which describe the deletion of an operation is also a two part process. One part involves the examination of the unit assignment table

entries in order to determine all possible negated-forward operation deletion strategies. It has been proved in Sitton[S1], that only adjacent definitions of each unit need be identified in order to determine all possible negated-forward operation deletion strategies.

The second part requires the examination of all alternate sink unit references generated from the identification of equivalent operations. These references refer to all possible parallel-forward and parallel-backward operation deletion candidates.

3.3.3 Analysis of Operation Deletion Strategies

Once all strategies have been identified, each strategy is analyzed individually. This is done in order to find all points requiring alteration, conflicts with other strategies, all options which may be used to alter each point, and to delete those strategies which may not be applied. They require the alteration of a point for which there exists no option.

The determination of strategy points involves the identification of the prime source unit(s) which must be altered or the operation(s) which must be deleted, in order that a strategy may be made applicable. When a strategy point is identified, it is classified and stored if not

already done so. Then all options which may be used to alter each point, with respect to the strategy requiring the alteration, are evaluated and stored if not already done so. Information on which strategy may use a particular option is also stored for the purpose of identifying conflicts.

The identification of strategy points may result in the identification and examination of a second type of strategy, Alternate Source Unit Assignment Strategies, which are used to change prime source units. The identification and evaluation of alternate source unit assignment strategies occurs when there exists one or more possible alternate source units which may be used to replace the prime source unit.

Strategy conflicts are identified and information is stored for later use, during the analysis of each strategy. This is done in order to combine strategies for the deletion of a maximum number of operations. Conflicts arise when strategies differ in how points may be altered. The identification involves examining variables indicating which option each strategy may use at each point, and the examination of other available information to be later described in detail.

It will later be shown that the evaluation of strategies is a recursive process, requiring only one

analysis of each strategy.

3.3.4 Strategy Selection

The major objective of this program is to determine all strategy sets which result in the deletion of at least n operations, where n is an expected maximum. This is done by finding and ordering the largest possible sets of non-conflicting operation deletion strategies in decreasing order of size. Each set is then evaluated, starting with the largest set. This process continues until all resulting sets of operation deletion strategies, and if needed, alternate source unit strategies; resulting in the deletion of at least the expected number of operation deletions are found, or until no sets are left.

Since there may exist more than one option per point, the resulting strategy sets may not result in distinct optimized microprogram segment versions. All that is assured is that each optimized version will have at least n, where n > 0, deleted operations. Each version will be logically correct with respect to the original microprogram.

It is important to note that the selection of strategy sets is the most time consuming process of all. This is primarily due to the possibility of many options available to alter any one point.

3.4 Limitations Within Solution Framework

The limitations are listed as follows:

1). Only segments of sequentially executable code and containing no branch operations or operations which are branched to, except for the first operation, may be optimized. That is to say, global analysis of all regions is not performed.

2). Operations which equivalently define units other than those which involve data-flow operations, or equivalent operations, are not identified or analyzed. For example, in the following code Figure 3.1, units 2 and 3 are equivalent after the given two micro-operations. These would not be so recognized by the existing program (presume that the subtraction function is interpreted as unit 3 minus unit 1).

1	ADD	{2,1}	3
2	SUB	{3,1}	3

Figure 3.1 Microprogram segment example

3.5 Summary

The purpose of this chapter has been to describe, in general, the solution framework applied by the program and limitations within the framework. The following chapters will describe in detail, give definitions, and present algorithms for each major phase of the solution framework.

CHAPTER IV

Identification of Alternate Operation Forms

4.1 Introduction

Within this chapter the process of identifying all possible operation forms is described in detail. The general order of entries consist of: first, a description of the unit types; second, a general description of the algorithm approach; third, a description of the tables used; and fourth, a detailed description of the algorithm for which a proof of correctness can be found in Sitton[S1]. The basic concept in this chapter is the application of an alternate unit identification algorithm. An alternate unit assignment table is constructed in order to represent all possible forms of each operation.

4.2 Unit Types

4.2.1 Introduction

Besides the prime units given in the source form of each operation, two types of alternate source units are used in the generation of alternate operation forms. The two

types of alternate units are defined as follows:

Definition 4.1: u is an Alternate Source Unit for MO(k) if it is equivalent to a unit specified in the source form of MO(k); i.e.,
(u,i) is an alternate source unit for (v,j) at MO(k),
iff i < k and MO(i) <=> MO(j).

Definition 4.2: Alternate Sink Units are those sink units which are equivalently defined; i.e.,
(u,i) and (v,j) are alternate sink units for each other, iff MO(i) <=> MO(j).

4.2.2 Alternate Source Units

Alternate source units are classified into three types. These are: Type C, which are not redefined before the appearance of the operation for which they are alternates; Type R, which are redefined before the appearance of the operation for which they are alternates; and Type A, which are defined by the assignment of alternate sink units. The three classes are defined as follows:

Definition 4.3: (u,i) is a Type C alternate source unit for (v,j) at MO(k), iff i < k;
MO(i) <=> MO(j);
and for all micro-operations l, i < l < k,

UAT(l,u) < 2.

Definition 4.4: (u,i) is a Type R alternate source unit for (v,j) at MO(k), iff i < k;

MO(i) <=> MO(j);

and for at least one micrc-operation l, i < l < k,

UAT(l,u) > 1.

Definition 4.5: (u,i) is a Type A alternate source unit for (v,j) at MO(k), iff i < k;

u ∈ SK(l,1);

MO(l) <=> MO(i);

MO(i) <=> MO(j).

Basically, the above definition 4.5 means that unit u is an alternate for (v,j), if it is moved from the sink set of MO(l) to the sink set of MO(i), where MO(i), MO(j), and MO(l) are equivalent operations.

4.2.3 Alternate Sink Unit References

The identification of alternate sink units involves the recognition of references for each operation to other operations which can equivalently define the sink set of that operation. Alternate sink unit references are classified into two types: Type P, which reference equivalent predecessor operations; and Type S, which

reference equivalent successor operations. Each type is defined as follows:

Definition 4.6: MO(i) is a Type P alternate sink unit reference at MO(j), iff i < j;
i > 0;
and MC(i) <=> MO(j).

Definition 4.7: MO(j) is a Type S alternate sink unit reference at MO(i), iff i < j;
and MC(i) <=> MO(j).

4.2.4 Unit Representation

Throughout this study, the following notation will be used when representing prime and alternate units or alternate sink unit references.

Definition 4.8: A memory unit is represented as a 4-tuple
(t, u, i, j)
where

t defines the type of unit and may be any of 'C', 'R', 'A', 'S', 'P', or Ø in the case of prime units;

u identiffes the unit, e.g. R2;

i identifies the operation which will define the

unit, or the equivalent operation in the case of alternate sink unit references;

j identifies the operation which initially defined unit u, in the case of Type A alternate source units.

4.3 General Description of Algorithm Approach

The algorithm uses one pass of the given microprogram segment in order to generate all operation forms. This uses all combinations of prime and alternate source units and finds all Type C and Type R alternate source units, and all Type P and Type S alternate sink unit references. The actual identification of alternate operation forms involves the use of a hashing function on the 'OP', 'SC', and 'C' sets of each operation. It also uses two tables in order to store equivalence information and current information on each unit. The algorithm then uses one pass of the ALT generated during the first part in order to find all Type A alternate source units. To be concise, equivalent operations are combined to identify alternate sink units. Equivalent sink units are combined to identify alternate source units.

4.4 Alternate Unit Assignment Table (ALT)

4.4.1 Introduction

The alternate unit assignment table is designed to hold all prime and alternate unit information for each micro-operation in order to represent all possible forms of each operation. The unit information within each ALT entry is of the form described in definition 4.8.

4.4.2 ALT Structure

If given a microprogram segment consisting of Nop operations, the ALT will consist of an ordered set of Nop entries. Each corresponds to a particular micro-operation, MO(i). Each entry corresponding to a MO(i) will consist of: a set of |SC(i,1)| ordered sets of prime and alternate source units, an ordered set consisting of the first prime sink unit and the alternate sink unit references, and an ordered set of |SC(i,1)| - 1 sink sets. Each of the latter sink unit sets will consist of one of the remaining prime sink units.

4.4.3 ALT Notation

When referring to a particular set of elements or an element within the ALT, the following notation will be used:

ALT(i, t, j, k)

where

i indicates the particular ALT entry corresponding to MO(i);

t is either 'SC' or 'SK', indicating one of the source or sink sets of prime and alternate units;

j identifies the jth set of type indicated by t;

k indicates the kth element within the jth set.

When referring to a particular field within an element located within the ALT, each element of the form described in definition 4.8. The following notation will be used:

ALT(i, t, j, k)[l]

where l indicates the lth field of the element ALT(i, t, j, k).

The following two figures are given in order to clarify the description of the ALT.

1	ADD	{1,2}	3
2	SUB	{3,4}	4
3	GATE	3	5
4	GATE	5	6
5	AND	{6,1}	5
6	ADD	{3,1}	7

Figure 4.1 Microprogram segment example

	SOURCE 1	SOURCE 2	SINK 1
MO(1)	(1, 0)	(2, 0)	(3, 1) (S, 4) (S, 3)
MO(2)	(3, 1) (A, 6, 1, 4) (A, 5, 1, 3)	(4, 0)	(4, 2)
MO(3)	(3, 1) (A, 6, 1, 4) (A, 5, 1, 3)		(5, 3) (S, 4) (P, 1)
MO(4)	(5, 3) (A, 6, 3, 4) (A, 6, 1, 4) (A, 5, 1, 3) (A, 3, 3, 1) (C, 3, 1)		(6, 4) (P, 1) (P, 3)
MO(5)	(6, 4) (A, 6, 3, 4) (A, 6, 1, 4) (A, 5, 1, 3) (A, 5, 4, 3) (A, 3, 3, 1) (A, 3, 4, 1) (C, 3, 1) (C, 5, 3)	(1, 0)	(5, 5)
MO(6)	(3, 1) (A, 6, 3, 4) (A, 6, 1, 4) (A, 5, 1, 3) (A, 5, 4, 3) (A, 3, 3, 1) (A, 3, 4, 1) (R, 5, 3) (C, 6, 4)	(1, 0)	(7, 6)

Figure. 4.2 ALT of microprogram segment in Figure 4.1

The following is given in order to review the given notation.

|ALT(2,SC,1)| = 3 [the number of prime and alternate source units corresponding to the first source unit of MO(2)].

ALT(2,SC,1) = (3,1), (A,6,1,4), (A,5,1,3) [the set of prime and alternate source units corresponding to the first source unit of MO(2)].

ALT(2,SC,1,1) = (3,1) [the first prime source unit of MO(2) is unit 3 and was last defined at MO(1)].

ALT(2,SC,2,1) = (4,0) [the second prime source unit of MO(2) is unit 4 and was last defined prior to entry into the given microprogram segment].

ALT(4,SK,1,1)[2] = 6 [the first sink unit of MO(4)].

ALT(6,SC,1,4)[1] = A, ALT(6,SC,1,4)[2] = 5, ALT(6,SC,1,4)[3] = 1, ALT(6,SC,1,4)[4] = 3 [the fourth unit corresponding to the first source unit set of MO(6) is a Type A alternate unit; the unit is 5; to be defined at MO(1); and initially defined in the source form of MO(3)].

4.4.4 ALT Implementation

The ALT has been designed and implemented as a list of records. Each record was designed to contain: a reference to the next such record, a reference to the corresponding micro-operation, a reference to a list of prime and alternate source unit sets, and a reference to a list of sink unit sets. The first sink set contained the alternate sink unit references. Each set of prime and alternate units was designed as a list of records. Each record contained a reference to a vector of records with fields as defined in definition 4.8, plus a reference to the next such record.

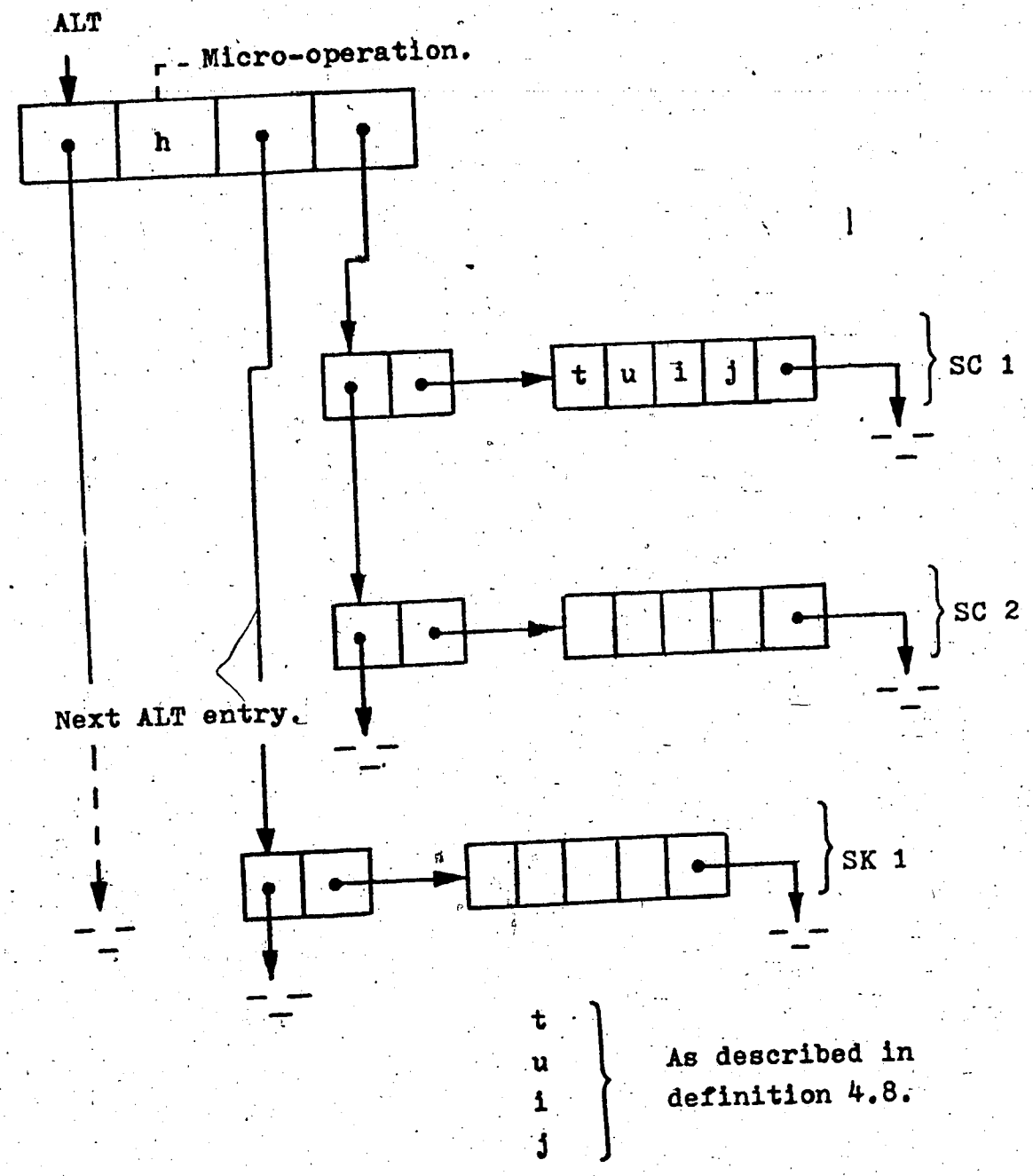


Figure 4.3 ALT implementation

4.5 Hash Function

Distinct operation forms are identified by the application of a hashing function on all generated operation images. Each image consisted of a concatenation of the strings representing an operation code, a source unit combination, and a control set.

The current implementation applies a quadratic hash modulo p, where p is a prime, on the first ten nonblank characters of each image. If a clash occurs, it is resolved by squaring a counter and adding it to the last hash code, modulo p. The actual value of p is of no importance except that it be a prime number. It also defines the maximum number of distinct operation images allowed.

Within this text the notation, H(ST), will denote a code generated from hashing the string ST. The notation, ST+ST1, will denote the concatenation of the strings ST and ST1. Also, single quotes will be used around string constants. Examples of operation images hashed and hash results can be found in the Appendix.

4.6 Operation Hash Table (OHT)

The function of the OHT is to represent all the equivalent information on each unit of the given micro-program segment. That is to say, to represent those QSMUs which are equivalently defined.

The OHT is designed to hold entries consisting of a hash code, an operation form, and a vector. Each vector consisted of ordered pairs, QMSUs, which are equivalently defined by the operation form; or, a vector of OHT entry addresses where a unit u defined by MO(j) is uniquely represented by the operation form 'NOP,(u,j)'. The OHT is addressed by using the codes generated from the hashing function.

4.6.1 OHT Example

Using the first three operations within the micro-program segment of Figure 4.1, the OHT addresses would be as follows:

H('ADD,(1,0),(2,0)') = 277;

H('SUB,(3,1),(4,0)') = 477;

H('GATE,(3,1)') = 834;

and at each address would be

OHT(277) = (3,1), (5,3);

OHT(477) = (4,2);

OHT(834) = (5,3), (3,1).

There would also exist the following:

OHT(H('NOP,(3,1)')) = 277,834;

OHT(H('NOP,(4,2)')) = 477;

CHT(H('NOP,(5,3)')) = 834,277.

4.6.2 OHT Implementation

The operation hash table was implemented as a binary tree for quick access. Each entry was designed to consist of: a pair of left and right reference links, a distinct hash code, a string representing the operation form which hashed to the entry address, and a reference to a vector of records. Each vector of records was designed as a linked list of records each containing an OHT address, or QMSU fields as defined in definition 2.6.

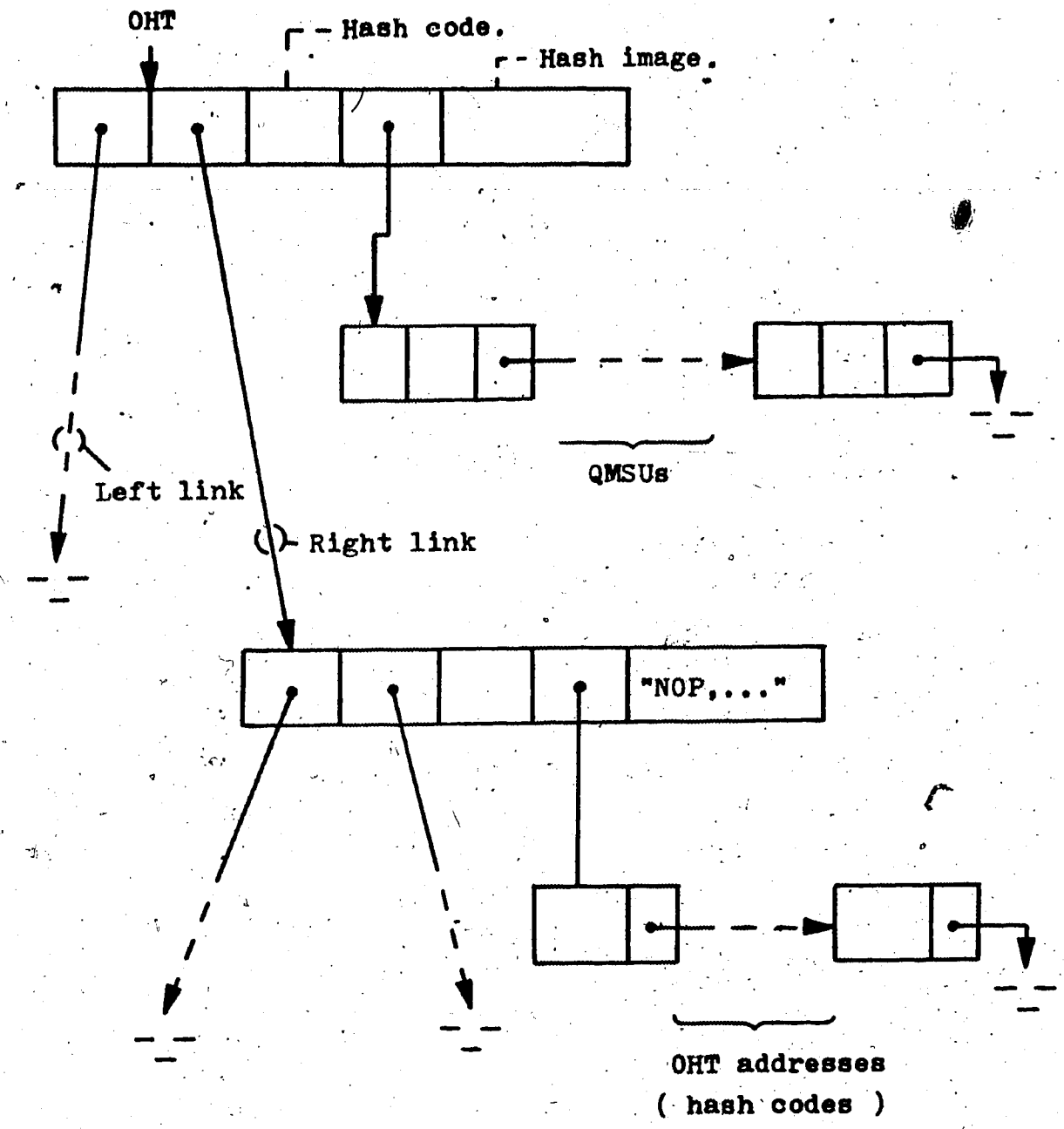


Figure 4.4 OHT implementation

4.7 Unit Hash Address Table (UHA)

The unit hash address table is used for each unit in the given microprogram segment. It is used to store the index of the operation which last defined the corresponding unit, and the OHT address of the entry containing the image of the operation form which last defined the corresponding unit. The UHA table is at most a 2 by Nu array, where Nu is the number of distinct units within the given microprogram segment. Row 1 contains the operation index and row 2 contains the OHT address.

4.7.1 UHA Example

By using the same first three operations within the microprogram segment of Figure 4.1, the UHA would contain the values as in Figure 4.5, after MO(3) was examined.

ROW	UNITS						
	1	2	3	4	5	6	7
1	0	0	1	2	3	0	0
2	708	709	277	477	277	713	714

Figure 4.5 UHA example

Note that upon entry into a given microprogram segment, each unit u is given the index of definition 0, and is initialized as defined by the operation form 'SHIFT,(u,0)'.

4.7.2 UHA Implementation

The unit hash address table was merged together with the unit assignment table. Thus it makes up the other two fields described in section 2.5.3.

4.8 Detailed Description of the Alternate Unit Identification Algorithm

A detailed description of each part of the algorithm is presented next. At this time the following notation will be used: the representation, D • S, is used to denote the insertion of an element S into a set D; and, D ≠ S, represents all elements of D, except those equal to the element S.

4.8.1 Part 1

To be able to find all Type C and Type R alternate source units and all Type P and Type S alternate sink unit references, with only one examination of each operation, the algorithm must provide the following information upon examination of each operation. Each sink unit of each micro-

operation must be "matched" with all QMSUs equivalently defined to contain the same value. All possible combinations of prime and/or alternate source units must be used to find all possible forms of each operation.

This identification process involves the following three phases:

Phase 1.1: Identifies all Type C and Type R alternate source units for each operation.

Phase 1.2: Determines alternate sink unit references from each unmodified data-flow operation; e.g. a GATE operation with a null control field.

Phase 1.3: Determines alternate sink unit references from operations which are not unmodified data-flow operations.

4.8.1.1 Part 1:Phase 1.1

The major objective of phase 1.1 is to retrieve, from the OHT, all Type C and Type R alternate source unit information for every prime source unit of each micro-operation.

Algorithm 4.8

The following steps are executed during phase 1.1 using

the given microprcgram segment.

A.1 Initialize the OHT and then initialize row 1 of the UHA to zeros.

A.2 Identify each unit within the microprogram segment as if it were defined by a trivial SHIFT operation with an index of 0; i.e., OP(0) = SHIFT, SC(0,1) = j, and SK(0,1) = ∅, for each unit j so that references to units defined in a predecessor segment are distinct from units defined in the current segment.

For i:=1, ... , |U| do

UHA(2,u_i):=H('SHIFT,('+u_i +',0)');

OHT(UHA(2,u_i)):=(u_i ,0); and

OHT(H('NOP,('+u_i +',0)')):=UHA(2,u_i),

where U = { u_1 ,u_2 ,... , u_n },

is the set of all units used in the given mircoprogram segment.

A.3 Do steps A.4 through A.11 for each micro-operation i = 1,...., Nop where Nop equals the number of operations.

A.4 Initialize a string X to be used to gather the operation form of each micro-operation i.

X:=OP(i).

A.5 Execute steps A.6 through A.9 for each prime source unit of MO(i). Then continue with step A.10.

For j:=1, ..., |SC(i,1)| do steps A.6 to A.9.

A.6 Initialize the ith AIT entry if not already done so. Then initialize a set for the jth source unit by inserting an entry for source unit SC(i,1,j), last defined at operation UHA(1,SC(i,1,j)).

AIT(i,SC,j):= (SC(i,1,j),UHA(1,SC(i,1,j))).

A.7 Concatenate the QMSU for the current prime source unit cnto the string X.

X:=X+',('+SC(i,1,j)+','+UHA(1,SC(i,1,j))+')'.

A.8 Identify all Type C and Type R alternate source units for the current prime source unit by first letting Q consist of the set of all equivalent but not equal to the QMSU for the current prime source unit.

Q:=OHT(UHA(2,SC(i,1,j))) ≠

(SC(i,1,j),UHA(1,SC(i,1,j))).

A.9 Then for each QMSU in Q, determine if the unit within has been redefined since the last operation, and appropriately insert an alternate Type C or Type R element into the ALT(i,SC,j).

For k:=1, ..., |Q| do

Let s:=first value in Q(k) and

t:=second value in Q(k);

If UHA(1,s) = t then

ALT(i,SC,j) := ALT(i,SC,j) ▪ (C,s,t, Ø)

else

ALT(i,SC,j) := ALT(i,SC,j) ▪ (R,s,t, Ø).

A.10 Concatenate the control set onto X. Identify and place each prime sink unit into the ALT. Update the UHA entry for each sink unit in order to reflect it's current index of definition.

X:=X+C(i);

For k:=1, ..., |SK(i,1)| do

UHA(1,SK(i,1,k)):=i;

UHA(2,SK(i,1,k)):=H(X); and

ALT(i,SK,k,1):=(SK(i,1,k),i).

A.11 If MO(i) is an unmodified data-flow operation, then execute Phase 1.2, or else execute Phase 1.3.

4.8.1.2 Part 1:Phase 1.2

If the current operation being examined is an unmodified data-flow operation, then the sink unit set of the operation is made equivalent to the source unit set. The sink units are made equivalent by placing QSMUs of each sink unit at all OHT entry addresses containing the current QMSU of the source unit. Then by determining where the sink units of the current operation can be used alternatively, alternate sink unit references are entered into the ALT

accordingly.

Algorithm 4.B

The following steps are executed for each unmodified data-flow operation, MO(i).

B.1 Update the UHA for each sink unit of MC(i) so the sinks are equivalent to the source unit.

For j:=1, ..., |SK(i,1,1)| do

UHA(2,SK(i,1,j)):=UHA(2,SC(i,1,1)).

B.2 Let D become equal to the set of OHT addresses containing operation forms which contain the source unit SC(i,1,1).

D:=OHT(H('NOP,('+SC(i,1,1)+','+UHA(1,SC(i,1,1))+')')).

B.3 Place a QMSU for each sink unit of MO(i) at each address containing a QMSU of the source unit of MO(i).

For j:=1, ..., |D| do

For k:=1, ..., |SK(i,1)| do

OHT(D(j)):=OHT(D(j)) ▪ (SK(i,1,k),i).

B.4 Let E become equal to a vector of all QMSUs which are equivalent to the QMSU of the source unit of MO(i).

E:=OHT(D(1)).

B.5 Using E, determine all operations to which the sink units of MC(i) are equivalent. Insert alternate sink unit references accordingly.

For j:=1, ..., |E| do

For k:=1, ...,|SK(i,1)| do

ALT(E(j)[2],SK,1):=ALT(E(j)[2],SK,1) * (S,i), and

ALT(i,SK,1):=ALT(i,SK,1) * (P,E(j)[2]).

B.6 Update the OHT for each unit with OHT addresses of equivalent QMSUs.

For j:=1, ..., |SK(i,1)| do

OHT(H('NOP,('+SK(i,1,j)+','+i+')')):= D.

4.8.1.3 Part 1:Phase 1.3

If the current operation is not an unmodified data-flow operation, then equivalent sink unit references are determined. This is accomplished by hashing all possible forms of the current operation using all combinations of prime and/or alternate source units for the current operation. This phase is executed in order to identify all equivalent QMSUs defined earlier by equivalent operation forms.

The following operation forms for MO(6) found in Figure 4.1, are presented as an example of the described generation process:

ADD,(3,1),(1,0)

ADD,(5,3),(1,0)

ADD,(6,4),(1,0)

Algorithm 4.C

The following steps are executed for each operation which is not an unmodified data-flow operation.

C.1 Generate all operation forms of MO(i) using all the prime and/or alternate source units found in the ALT entry for MO(i). At the same time let D become equal to the vector of OHT addresses generated and which contain images of operations that can equivalently define the sink units of MO(i). Let E become equal to the vector of all equivalent QMSUs.

D:= ∅;

E:= ∅;

For each combination V of prime and/or alternate source units from ALT(i,SC) do

D:=D ▪ H(OP(i)+V+C(i)) and

E:=E ▪ (OHT(H(OP(i)+V+C(i))) ≠ E).

C.2 Update the OHT with the accumulated vector D, with each QMSU within the vector E.

For j:=1, ..., |E| do

OHT(H('NOP,('+E(j)[1]+','+E(j)[2]+')')):=D.

C.3 Using those QMSUs within E that are equivalent to the QMSUs of the sink unit(s) of MO(i), determine all equivalent operations and place alternate sink unit references into the ALT accordingly.

For j:=|SK(i,1)|+1, ..., |E| do

ALT(E(j)[2],SK,1):=ALT(E(j)[2],SK,1)+(S,i), a

ALT(i,SK,1):=AIT(i,SK,1)+(P,E(j)[2]).

C.4 Update the OHT with the accumulated QMSU vector E at all OHT addresses within D.

For j:=1, ..., |D| do

OHT(D(j)):=E.

4.8.2 Part 2

With all alternate sink unit references determined, it is possible to identify all Type A alternate source units. This is done by examining each Type P and Type S alternate sink unit reference. Part 2 involves one sequential examination of the AIT. At this time each alternate sink unit reference is selected. Each prime sink unit for the operation to which an alternate sink unit may be assigned is determined. The ALT is then examined from the following entry for the referenced operation to the last entry in the ALT. If the prime sink unit of the referenced operation is used as a prime or non-Type A alternate source unit, then the prime sink unit of the referencing operation is entered as a Type A alternate sink unit.

Algorithm 4.D

Part 2 involves the execution of the following steps using the ALT generated in part 1.

D.1 Execute step D.2 for each ALT entry.
For i:=1, ..., Nop do step D.2.

D.2 Execute D.3 for each alternate sink unit reference within ALT(i,SK,1).
For j:=2, ..., |ALT(i,SK,1)| do step D.3.

D.3 Execute step D.4 for each prime sink unit u of MO(i).

D.4 Execute step D.5 for each corresponding sink unit v of MO(k); k = ALT(i,SK,1,j)[3].

D.5 Execute step D.6 for each ALT entry of MO(k+1) through MO(Nop).
For l:=k+1, ..., Nop do step D.6.

D.6 Insert unit u as an alternate Type A source unit each time unit v is found as a prime or non-Type A alternate source unit within MO(l).

```
For m:=1, ..., |SC(l,1)| do
For n:=1, ..., |ALT(l,SC,m)| do
    If ALT(l,SC,m,n)[1] ≠ A and
       ALT(l,SC,m,n)[2] = v then
         ALT(l,SC,m):=ALT(l,SC,m) ▪ (A,u,k,i).
```

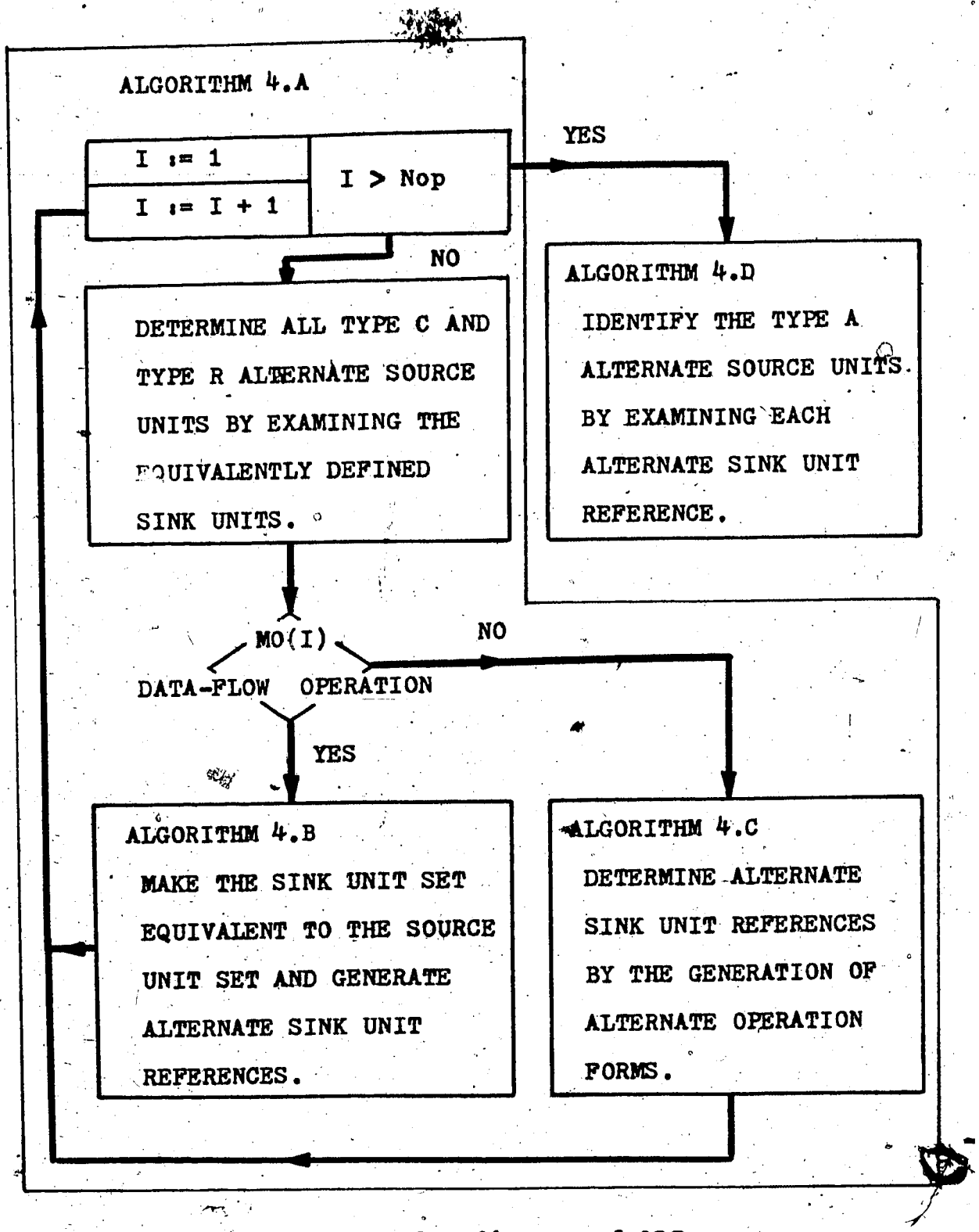


Figure 4.6 Flow diagram of ALT generation

4.9 Summary

The purpose of this chapter has been to describe the algorithm used to build the ALT such that all possible forms of each operation may be represented. The algorithm has been programmed successfully and a proof of the correctness of the algorithm can be found in Sitton[S1].

It has been pointed out in Sitton[S1] that the ordering of the source units, where primitive operations conform to commutative laws, eliminates the chance of nonmatches on operation forms which generate the same result (i.e., source units for primitive operations such as SUB should have a defined order of appearance in the hashing source image). This is correct, but slightly misleading, in that the reordering of units will also increase matches. For example, an image 'ADD,{R1,R2}' may operationally give the same result as 'ADD,{R2,R1}'.

CHAPTER V

Deletion Candidacy

5.1 Intoduction

The purpose of this chapter is to describe the conditions which must be satisfied in order to delete an operation and to describe how initial operation deletion candidates are identified. It is also to describe the evaluation of the feasibility of each strategy used to delete an operation. Conditions for the deletion of operations have been described in Chapter 2, and the process of identifying all possible forms of each operation has been described in Chapter 4. The general problem now is to first identify all initial operation deletion candidates. These are to be evaluated in order to determine all points where alterations must be performed (i.e., where strategies must be applied). This is done in order to meet the condidtions as described in Chapter 2. The identification and evaluation involves the organization of the information describing the operation deletions and prime source unit alterations.

The basic result of this chapter is the presentation of algorithms developed and used for the identification and

evaluation of the various types of strategies which will be discussed. These algorithms were developed from theorems and theoretical conditions, and were verified through experimentation with the implementation.

5.2 General Description of Operation Deletion Candidacy

Before an operation may be a feasible candidate for deletion, a minimal set of conditions as defined in Chapter 2 must be satisfied. The process involves the examination of the UAT and ALT tables in order to identify all initial operation deletion strategies. It then involves the independent evaluation of each strategy in order to determine the points where operation(s) must be altered or deleted, to evaluate all options which may be used to alter each point, and to determine which strategies may be deleted.

Note that conditions for operation deletion candidacy are based on the examination of each strategy independently. Still, the deletion of [illegible]ation may be a dependent event, as it may af[illegible] deletion status of other operations.

5.3 Fan-Out

It is easily seen that if MO(i) <=> MO(j), then either operation theoretically could define either or both sink unit sets. The practical application depends on the parallel hardware fan-out capabilities of the particular machine. Specifically, fan-out capabilites may affect the deletion candidacy of parallel operations. In order to accommodate this, deletion candidates will be determined with and without fan-out capabilities.

It is interesting to point out that the allowance of fan-out brings about a slight problem. The problem is that the utilization of fan-out does enhance the deletion of non-essential operations, but it does not directly enhance the the efficient utilization memory units. With the allowance of fan-out, resulting optimized microprogram segment versions must be evaluated by the user in order to choose those versions which utilize memory units most efficiently. There may exist an excessive duplication of information in order to delete a maximum number of operations. For example, consider the following microprogram segment in Figure 5.1.

1	ADD	{1,2}	3
2	SUB	{3,4}	4
3	GATE	3	5
4	ATE	5	6
5	AND	{6,2}	5
6	AND	{3,1}	7

Figure 5.1 Microprogram segment example

In this case, with fan-out, both MO(3) and MO(4), could generally be deleted by parallel-backward movement. This deletion would result in the following optimized micro-program segment version in Figure 5.2.

1	ADD	{1,2}	{5,6}
2	SUB	{5,4}	4
5	AND	{6,2}	5
6	AND	{6,1}	7

Figure 5.2 Optimized segment example

A more efficient version would be as in the following Fig 5.3.

1	ADD	{1,2}	3
2	SUB	{3,4}	4
5	AND	{3,2}	5
6	AND	{3,1}	7

Figure 5 Cptimized segment example

On the other hand, fan-out brings forth a secondary benefit. The application and study of fan-out possibilities could be used to identify those operational units which could be altered and improved. This would be done by increasing the number of data paths leading out of them, in order to increase the parallelism and hence the processing power of a particular machine.

5.4 Strategies

5.4.1 Introduction

This study is concerned with two types of strategies: those which result in the deletion of an operation, and those which result in the replacement of a prime source unit by an alternate source unit. These two classes are the options available for the alteration of a strategy point.

Including fan-out, the operation deletion strategies are further classified into five possible types. Hence, there exist six distinct types of strategies which may be used to delete an operation.

5.4.2 Source Unit Replacement Strategies (SR)

Source unit replacement strategies are those which, if applied, result in the assignment of an alternate source unit. The identification of such strategies occurs during the evaluation of operation deletion strategies. Such strategies are applied in conjunction with operation deletion strategies to satisfy the conditions for deletion candidacy.

5.4.3 Operation Deletion Strategies

This class of strategies if applied, each result in the deletion of an operation and are described as follows:

1). Negated-Forward Deletion Strategies (NF): Those strategies which result in the negated-forward deletion of an operation. This is an exceptional class in that there may exist more than one strategy for each operation, depending on the number of sink units of the particular operation

being deleted. All are combined to delete the particular operation.

2). Parallel-Forward Deletion Strategies Without Fan-out (PF): Those strategies which result in the parallel-forward deletion of an operation.

3). Parallel-Forward Deletion Strategies With Fan-out (PF F): Those strategies which result in the parallel-forward deletion of an operation such that sink unit fan-out is performed.

4). Parallel-Backward Deletion Strategies Without Fan-out (PB): Those strategies which result in the parallel-backward deletion of an operation.

5). Parallel-Backward Deletion Strategies With Fan-out (PB F): Those strategies which result in the parallel-backward deletion of an operation such that sink unit fan-out is performed.

5.4.4 Strategy Representation

In order to discuss the feasibility of each type of strategy, the notation for the representation of a strategy will now be presented.

Definition 5.1: A strategy is a 7-tuple

(i, c, v, j, t, u, l)

where

i is an index which distinguishes the strategy from the rest;

c indicates whether the strategy has been checked;

v indicates whether the strategy is applicable;

j references the operation to be deleted, or references an alternate source unit within the ALT which is to be used as a replacement for a prime source unit within MO(l);

t identifies the type of strategy and may be any of 'SR', 'NF', 'PF', 'PF F', 'PB' , or 'PB F';

u indicates the uth prime source unit of MO(j) is to be altered, that the sink units of MO(j) and MO(l) are equal, or references a prime sink unit of MO(j) in the case of a 'NF' type strategy;

l references the operation with respect to which MO(j) is to be deleted, within which an alternate unit is to be assigned, or the next operation which affects unit u in the case of a 'NF' type strategy.

5.4.5 Storage of Strategy Information

Each strategy once identified, is stored within a Strategy Table (ST) to be discussed in the next chapter. All that is needed now, is that the strategies are stored in the order operation deletion strategies followed by source unit replacement strategies. The operation deletion strategies are further ordered in increasing order of the operation indices indicating the operation to be deleted. Then in the order of negated operation deletion strategies which are [illegible] ordered in increasing order of the 1 field, followed by the parallel operation deletion strategies. Source unit replacement strategies are also ordered in increasing order of the operation indices specifying the locations at which a prime unit is to be altered.

This partitioning and ordering is done primarily to reduce the search times for strategies. It is also done because alternate unit assignment strategies are identified during the feasibility evaluation of operation deletion strategies.

5.5 Alternate Unit Assignment Strategy Feasibilty

An alternate source unit, ALT(i,SC,j,k), is feasible for assignment only if conditions exist which insure that the unit is equivalent at MO(i) to the prime source unit ALT(i,SC,j,1).

Definition 5.1: Unit u is a feasible alternate source unit for the lth prime source unit v, at [redacted] one of the following conditions is satisfied.

1). (C,u,i) ∈ AIT(k,SC,1) such that
ALT(k,SC,1,1)[2] = v.

2). (R,u,i) ∈ ALT(k,SC,1) such that
ALT(k,SC,1,1)[2] ≠ v;
for j:=i+1, ..., k-1
where UAT(j,u) > 1,
MO(j) is a feasible negated-forward candidate for deletion; or, MO(j) is a feasible parallel-forward candidate for deletion with respect to a MO(m) such that m ≥ k.

3). (A,u,i,j) ∈ ALT(k,SC,1) such that
ALT(k,SC,1,1)[2] = v; MO(j) is a feasible parallel candidate for deletion with respect to MO(i);
for m:=i+1, ..., k-1
where UAT(m,u) > 1,

MO(m) is a feasible negated-forward candidate for deletion; or, MO(m) is a feasible parallel-forward candidate for deletion with respect to a MO(n) such that $n \geq m$.

All alternate source unit strategies which meet the conditions within Definition 5.1 are recorded as feasible after only one examination of each strategy.

5.6 Operation Deletion Strategy Feasibility

The minimal set of conditions for each operation deletion strategy type will be described within this section. Note that conditions for deletion candidacy are based upon one examination of each strategy independently of the others. Yet, the deletion of an operation is a dependent event, as it may affect the deletion status of other operations.

Definition 5.2: MO(i) is a feasible negated-forward candidate for deletion, iff the following minimal set of conditions are satisfied and no condition conflicts occur.

For each sink unit $u \in SK(i,1)$, there exists a MO(m) such that the following conditions are satisfied:

1). $i < m$.

2). UAT(m,u) > 1.

3). UAT(j,u) < 2 for all operations j, i < j < m.

4). For j:=i+1, ..., m

where u = ALT(j,SC,1,1)[2],

1 < l <= |SC(j,1)|, at least one of the following conditions is satisfied:

a). There exists a feasible alternate within ALT(j,SC,l);

b). MO(j) is a negated-forward candidate for deletion;

c). MO(j) is a parallel-forward candidate for deletion;

d). MO(j) is a parallel-backward candidate for deletion.

Definition 5.3: MO(i) is a feasible parallel-forward candidate for deletion with respect to MO(j), iff the following minimal set of conditions are satisfied and no condition conflicts occur.

1). SK(i,1) = SK(j,1),

or SK(i,1) may be fanned-out at MC(j),

or MO(j) is a feasible candidate for deletion and the sink set SK(i,1) can be used as the sink set for MO(j).

2). For k:=i+1, ..., [illegible] and

For l:=1, ..., [illegible],1)|

where ALT(i,SK,l,1)[2] = ALT(k,SC,m,1)[2],

1 < m <= |SC(k,1)|, at least one of the following conditions is satisfied:

a). There exists a feasible alternate within ALT(k,SC,m);

b). MO(k) is a feasible negated-forward candidate for deletion;

c). MO(k) is a feasible parallel-forward candidate for deletion;

d). MO(k) is a feasible parallel-backward candidate for deletion.

3). For k:=i+1, ..., j-1 and

For l:=1, ..., |SK(i,1)|,

where ALT(i,SK,l,1)[2] = ALT(k,SK,m,1)[2],

1 < m <= |SK(k,1)|, at least one of the following conditions is satisfied:

a). MO(k) is a feasible negated-forward candidate for deletion;

b). MO(k) is a feasible parallel-forward candidate for deletion with respect to a MC(r), such that r ≥ j;

c). MO(k) is a feasible parallel-backward

candidate for deletion with respect to a MO(r), such that r < i.

Definition 5.4: MO(j) is a feasible parallel-backward candidate for deletion with respect to MO(i), iff the following minimal set of conditions are satisfied and no condition conflicts occur.

1). SK(j,1) = SK(i,1),
or SK(j,1) may be fanned-out at MC(i),
or MO(i) is a feasible candidate for deletion and the sink set SK(j,1) can be used as a sink set for MO(i).

2). For k:=i+1, ..., j and
For l:=1, ..., |SK(j,1)|
where ALT(j,SK,l,1)[2] = ALT(k,SC,m,1)[2],
1 < m <= |SK(k,1)|, at least one of the fcllowing conditions is satisfied:

a). There exists a feasible alternate in ALT(k,SC,m);

b). MO(k) is a feasible negated-forward candidate for deletion;

c). MO(k) is a feasible parallel-forward candidate for deletion;

d). MO(k) is a feasible parallel-backward candidate for deletion.

3). For k:=i+1, ..., j-1 and

For l:=1, ..., |SK(j,1)|

where ALT(j,SK,l,1)[2] = ALT(k,SK,m,1)[2],

1 < m <= |SK(k,1)| at least one of the following conditions is satisfied:

a). MO(k) is a feasible negated-forward candidate for deletion;

b). MO(k) is a feasible parallel-forward candidate for deletion with respect to MO(r), such that r ≥ j;

c). MO(k) is a feasible parallel-backward candidate for deletion with respect to MO(r), such that r < j.

5.7 Reciprocally Dependent Deletion Conditions

It may be shown that the recursively defined conditions used for the evaluation of the feasibility of strategies may never converge. This condition has been termed reciprocal dependency. It may occur during the analysis of a MO(i) as a deletion candidate, if it is found that MO(j) must be a feasible deletion candidate. Then, during the analysis of MO(j) as a deletion candidate, it is found that MO(i) must be a feasible deletion candidate. For example, consider the following program segment:

1	ADD	{1,2}	5	
2	SHIFT	5	4	[2]
3	ADD	{1,2}	6	

Figure 5.4 Microprogram segment example

If unit 5 is not required upon exit, then MO(1) is examined as a negated-forward candidate for deletion. In which case, the following conditions must be satisfied:

a). Unit 5 at MO(2) must be replaced by unit 6.

b). The MO(3) definition of unit 6 must be moved to MO(1).

Without fan-out, examination of the latter microprogram segment would first consist of examining the negated-forward deletion candidacy of MO(1). This in turn would result in the examination of MO(3) as a feasible parallel-backward deletion candidate, during the analysis of unit 6 as a feasible alternate source unit for unit 5 at MO(2). The examination of the feasible parallel-backward deletion candidacy of MO(3), would thus lead to the examination of the feasible negated-forward deletion candidacy of MO(1), in order to move unit 6 for definition at MO(1). Hence, MO(1) and MO(3) are reciprocally dependent with respect to the deletion conditions.

It has been proved in Sitton[S1], that reciprocal

dependency need not impede the finite analysis of deletion candidacy; i.e., deletion candidacy analysis will converge and thus strategies can be evaluated by a finite analysis.

This problem has been resolved within the implementation by keeping a list of strategies currently being evaluated. If during the analysis of a strategy, the alteration of a point is required for which an option(s) exists within the current list. Each option is recognized as valid for that point and the search for valid options is continued with those options not within the current list.

5.8 Identification of Operation Deletion Candidates

The identification of all initial strategies which can result in the deletion of an operation is a two part process. This first invclves the examination of the UAT and then the examination of the AIT.

In order to identify all negated-forward operation deletion strategies, it has been proved in Sitton[S1] that all that needs to be done, is to identify all adjacent definitions of each unit within the given microprogram segment. The following algorithm is used to identify all operations to be evaluated for negated-forward deletion.

Algorithm 5.A

The following steps are executed in order to identify all initial negated-forward operation deletion strategies.

A.1 Do steps A.2 through A.6 for each unit within the given microprogram segment.

For i:=1, ..., |U| do A.2 to A.6

where $U=\{u_1, u_2, \ldots, u_n\}$, the set of all units.

A.2 Let j:=1.

A.3 While j <= Nop+1 and UAT(j,u_i) < 1 do j:=j+1.

A.4 If j > Nop+1, then go to step A.1 and restart with the next unit, or else let k:=j+1 and do steps A.5 through A.6.

A.5 While k <= Nop+1 and UAT(k,u_i) < 1 do k:=k+1.

A.6 If k > Nop+1, then go to step A.1 and repeat with next unit; or else, MO(j) is a negated-forward deletion candidate, hence store the strategy (x, false, false, j, 'NF', u_i, k.) such that x is the next strategy identifier, let j:=k, k:=k+1, and go to step A.5.

In order to identify all initial parallel operation deletion strategies, each entry within the ALT is examined for alternate sink unit references.

If fan-out is allowed, a similar 'PF F' or 'PB F' type strategy would also be generated with each 'PF' or 'PB' strategy found.

Algorithm 5.B

The following three steps are executed in order to identify all initial parallel operation deletion strategies.

B.1 Do step B.2 for each ALT entry.
For i:=1, ..., Nop do step B.2.

B.2 Execute step B.3 for each alternate sink unit reference within ALT(i,SK,1).
For j:=2, ..., |ALT(i,SK,1)| do step B.3.

B.3 If ALT(i,SK,1,j)[1] = 'S' then store the parallel-forward strategy (x, false, false, i, 'PF', Ø, ALT(i,SK,1,j)[3]), or else store the parallel-backward strategy (x, false, false, i, 'PB', Ø, ALT(i,SK,1,j)[3]).

If the latter two algorithms are applied to the tables generated for the microprogram segment in Figure 4.1, the following list in Figure 5.5 of operation deletion candidates would be obtained.

Negated-forward Deletion Candidates

1). MO(1) with respect to MC(N+1).

2). MO(3) with respect to MC(5).

3). MO(4) with respect to MC(N+1).

Parallel-forward Deletion Candidates

1). MO(1) with respect to MC(4).

2). MO(1) with respect to MC(3).

3). MO(3) with respect to MC(4).

Parallel-backward Deletion Candidates.

1). MO(3) with respect to MC(1).

2). MO(4) with respect to MC(1).

3). MO(4) with respect to MC(3).

Figure 5.5 Operation deletion candidates example

5.9 Feasibility Examination of Identified Strategies

The purpose of this section is to describe the process involved in evaluating the feasibility of each strategy. The process as a whole, involves a sequential examination of the strategy table. At this time each operation deletion strategy is evaluated as feasible or not, if not already done so. Nonfeasible strategies are deleted after the evaluation process is completed.

Each of the fcllowing algorithms are used to evaluate the operation deletion strategies found by the algorithms of section 5.8. Examples of the feasible operation deletion candidates as described in Figure 5.5 for the microprogram segment given in Figure 4.1, can be fcund within the Appendix.

Algorithm 5.C

Each operation deletion strategy within the ST is examined in order, as follows:

C.1 Let i:=0.

C.2 Iet i:=i+1.

C.3 Go to step C.5 if strategy i does nct exist or if strategy i is an alternate source unit assignment strategy, or else continue with step C.4.

C.4 Go to step C.2 if strategy i has been checked, or else examine strategy i depending on its type and then go tc step C.2.

C.5 Iet i:=0.

C.6 Iet i:=i+1.

C.7 Continue with step C.8 if strategy i exists, or else stop.

C.8 Delete strategy i if it is not checked as feasible, and then continue with step C.6.

The following algorithm is used to evaluate the feasibility of alternate unit assignment strategies. These are generated during the examination of operation deletion strategies and are stored following the operation deletion strategies.

Algorithm 5.D

An alternate unit assignment strategy (i, c, v, j, 'SR', u, 1) is checked as feasible or not by executing the fcllowing steps.

D.1 Continue with step D.7 if ALT(j)[1] = C, else go to step D.4 if ALT(j)[1] = R, or else continue with step D.2.

D.2 ALT(j)[1] = A, so evaluate all parallel deletion

strategies for the deletion of MO(ALT(j)[4]) which have not yet been examined.

D.3 Go to step D.8 if there exists no feasible parallel deletion strategy for the deletion of MO(ALT(j)[4]), or else continue with step D.4.

D.4 Check if unit ALT(j)[2] is used as a sink unit between MO(ALT(j)[3]+1) through MO(l-1) and do steps D.5 and D.6 if it is. Then continue with step D.7.

For m:=ALT(j)[3]+1, ..., l-1 do.

If UAT(m,ALT(j)[2]) > 1 then do steps D.5 and D.6.

D.5 Examine the negated-forward deletion of MO(m) if it is a candidate and if not already done so. Also, examine all unchecked parallel-forward deletion strategies of MO(m) with respect to a MO(p), such that $p \geq 1$.

D.6 If there exists a valid negated-forward deletion strategy for MO(m), or a parallel-forward deletion strategy for MO(m) with respect to a MO(p), such that $p \geq 1$, then continue with step D.4, or else go to step D.8.

D.7 Let v:=true.

D.8 Let c:=true and stop.

It has been stated that there must exist a negated-forward deletion strategy entry for MO(j), for each sink unit of MO(j). This difference with respect to parallel strategies has come about from the definition for the

negated-forward deletion candidacy of an operation, and from the method of identification of negated-forward deletion candidates. However, after the examination of the negated-forward deletion candidacy of a MO(j), only the first strategy entry within the ST is checked as valid if feasible, since the strategy set is merged into one.

Algorithm 5.E

The negated-forward strategy set for a MO(j) is examined by executing the following steps.

E.1 Go to step E.8 if the number of strategies within the set is less than the number of sink units of MO(i).

E.2 Do step E.3 for each strategy (i, c, v, j, 'NF', u, 1) within the negated-forward strategy set for MO(j) and then continue with step E.7.

E.3 Examine each operation from MO(j+1) through MO(l) and do steps E.4 through E.6 each time unit u is found within one of the prime source unit sets of one of these operations. Then continue with step E.2.

For m:=j+1, ..., l do

For n:=1, ..., |SC(m,1)| do

If u = ALT(m,SC,n,1)[2] then do steps E.4 through E.6.

E.4 Generate, store, and evaluate all alternate source unit

assignment strategies using the ALT entries ALT(m,SC,n,2) through ALT(m,SC,n,|ALT(m,SC,n)|).

E.5 Examine all nonchecked negated-forward and parallel deletion strategies for MO(m).

E.6 If no feasible alternate unit assignment strategy exists for the prime source unit ALT(m,SC,n,1)[2] and no feasible operation deletion strategy exists for MO(m), then go to step E.8, or else continue with step E.2.

E.7 Let v of the first strategy become equal to true.

E.8 Let c:=true and stop.

The following algorithm is used to examine all parallel-forward deletion strategies with or without fan-out.

Algorithm 5.F

A parallel-forward strategy (i, c, v, j, 'PF' or 'PF F', u, l) is checked as feasible upon satisfactory completion of the following steps.

F.1 If the sink units of MO(j) equal those of MO(1) or if fan-out is allowed, then continue with step F.3, or else continue with step F.2.

If SK(j,1,m) = SK(1,1,m) for all m,

$1 < m <= \max(|SK(j,1)|, |SK(1,1)|)$, or if fan-out is allowed, then go to step F.3.

F.2 Examine all deletion strategies for MO(1), and if there exist at least one feasible deletion strategy, then continue with step F.3, or else go to step F.12.

F.3 Check each operation from MO(j+1) through MO(1) and do steps F.4 through F.6 each time a sink unit as defined by MO(j) is found as a prime source unit within one of these micro-operations. Then continue with step F.7.

```
For m:=j+1, ..., 1 do
For n:=1, ..., |SK(j,1)| do
For p:=1, ..., |SC(m,1)| do
    If ALT(j,SK,n,1)[2] = ALT(m,SC,p,1)[2] and
       ALT(j,SK,n,1)[3] = ALT(m,SC,p,1)[3] then
        do steps F.4 through F.6.
```

F.4 Generate, store, and evaluate all alternate source unit assignment strategies using the entries ALT(m,SC,p,2) through ALT(m,SC,p,|ALT(m,SC,p)|).

F.5 Examine all nonchecked operation deletion strategies for MO(m).

F.6 If no feasible alternate unit assignment strategy exists for unit ALT(m,SC,p,1)[2] and no feasible operation deletion strategy exists for MO(m), then go to step F.12, or else continue with step F.3.

F.7 Examine each operation from MO(j+1) through MO(1-1) and

do steps F.8 through F.10 each time a sink unit of MO(j) is found to be defined by one of these operations. Then continue with step F.11.

For m:=j+1, ..., l-1 do

For n:=1, ..., |SK(j,1)| do

If UAT(m,ALT(j,SK,n,1)[2]) > 1 then do steps F.8 through F.10.

F.8 If a strategy exists and it is not checked, examine the negated-forward deletion feasibility of MO(m).

F.9 Examine all nonchecked parallel-forward strategies of MO(m) with respect to a MO(s), such that $s \geq j$. Also, examine all nonchecked parallel-backward strategies for MO(m) with respect to a MO(s), such that $s < j$.

F.10 If a feasible deletion strategy for MC(m) as specified in steps F.8 and F.9 exists, then continue with step F.7, or else continue with step F.12.

F.11 Let v:=true.

F.12 Let c:=true and stop.

The feasibility of each parallel-backward deletion strategy, with or without fan-out, is examined by the application of the following algorithm.

Algorithm 5.G

A parallel-backward strategy (i, c, v, j, 'PB' or 'PB F', u, 1) is checked as feasible upon satisfactory completion of the following steps.

G.1 If the sink units of MO(j) equal those of MO(1) or fan-out is allowed, then continue with step G.3, or else continue with step G.2.
If SK(j,1,m) = SK(1,1,m) for all m,
1 < m <= max(|SK(j,1)|,|SK(1,1)|) or if fan-out is allowed, then go to step G.3.

G.2 Examine all operation deletion strategies for MO(1) and if there exists at least one feasible deletion strategy, then continue with step G.3, or else go to step G.12.

G.3 Check each operation from MO(1+1) through MO(j), and apply steps G.4 through G.6 each time a sink unit of MO(j) is found as a prime source unit within one of these micro-operations. Then continue with step G.7.
For m:=1+1, ..., j-1 do
For n:=1, ..., |SK(j,1)| do

For p:=1, ..., |SC(m,1)| do

If ALT(j,SK,n,1)[2] = ALT(m,SC,p,1)[2] then do steps G.4 through G.6.

G.4 Generate, store, and examine all alternate source unit assignment strategies using the entries ALT(m,SC,p,2) through ALT(m,SC,p,|ALT(m,SC,p)|).

G.5 Examine all nonchecked operation deletion strategies for MO(m).

G.6 If no feasible alternate unit assignment strategy exists for unit ALT(m,SC,p,1)[2] and no feasible operation deletion strategy exists for MO(m), then go to step G.12, or else continue with step G.3.

G.7 Examine each operation from MO(l+1) through MO(j-1) and do steps G.8 through G.10 each time a sink unit of MO(j) is found to be defined by one of these operations. Then continue with step G.11.

For m:=l+1, ..., j-1 do

For n:=1, ..., |SK(j,1)| do

If UAT(m,ALT(j,SK,n,1)[2]) > 1 then do steps G.8 through G.10.

G.8 If a strategy exists and it is not checked, examine the negated-forward feasibility of MO(m).

G.9 Examine all nonchecked parallel-forward strategies of MO(m) with respect to a MO(s), such that $s \geq j$. Also, examine all nonchecked parallel-backward strategies of

MQ(m) with respect to a MO(s), such that s < j.

G.10 If a feasible deletion strategy for MC(m) as specified in steps G.8 or G.9 exists, then continue with step G.7, or else continue with step G.12.

G.11 Let v:=true.

G.12 Let c:=true and stop.

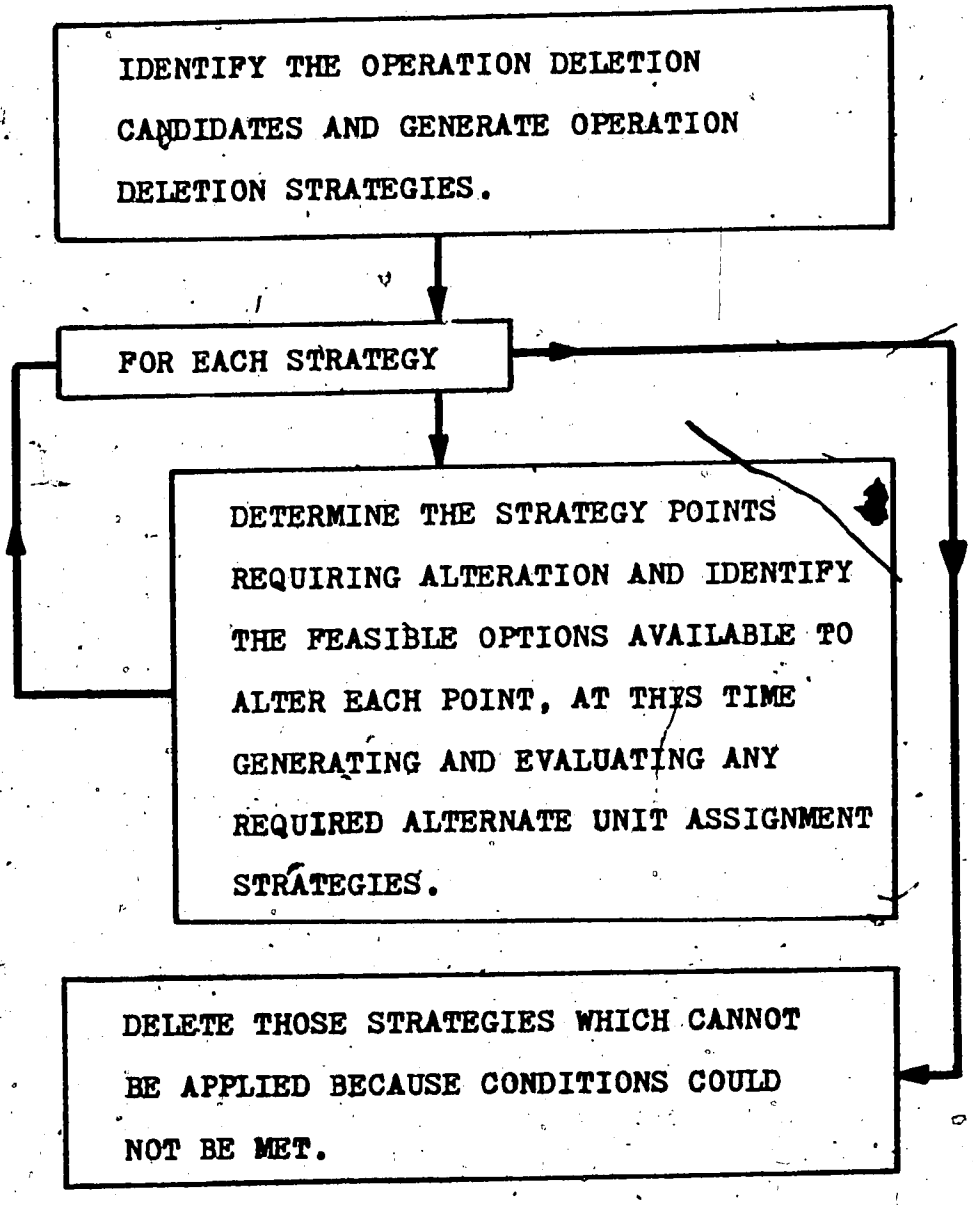


Figure 5.6 Flow diagram of strategy identification and evaluation

5.10 Summary

The purpose of this chapter has been to describe the develcped process for the identification and evaluation of strategies used in the deletion of operations. It has been shcwn that the evaluation of strategies involves only one examination of each strategy.

Each of the algcrithms described has been programmed successfully in ALGOLW. In order not to restrict the additicn of possible future extensions and constraints, the algorithms have been implemented modularly and with easily modifiable data structures.

The utilization of fan-out and the prcblem it gives in its applicatien in micrcprcgram optimization has also been discussed and clarified within this chapter. The problem was observed during the experimentation with the implementation. Optimization examples with fan-out can be found within the Appendix.

The following chapter will discuss the analysis variables used tc delete operations. The means by which the analysis variables are identified and stored will also be discussed.

CHAPTER VI

Deletion Candidate Analysis

6.1 Introduction

The purpose of this chapter is to describe the analysis of operation deletion strategies and alternate unit assignment strategies. It has been described previously that operation deletion candidates are identified and evaluated independently. In order to effect operation deletion candidacy, it is necessary to evaluate the dependencies among the deletion candidates. The objective of the analysis is to determine the greatest number of operations which can be deleted from the given microprogram segment.

The basic result of this chapter is the presentation of algorithms developed to select and apply the best strategy sets. These have been developed from theorems and theoretical conditions, and were verified through experimentation with the implementation.

6.2 General Analysis Approach

6.2.1 Introduction

The general approach used to evaluate operation deletion strategies is to: first, identify all possible deletion strategies; second, to evaluate their feasibility and to determine all points requiring alteration for each strategy; third, to determine the feasible options available for each point; and fourth, to determine the conflicts among the strategies. This analysis involves the evaluation of the following three variables:

1). Strategy Options: Those strategies which may be employed in the deletion of an operation.

2). Strategy Points: Locations within operations of the program segment where a strategy must be applied in order to delete an operation.

3). Strategy Conflicts: Those strategies which cannot be used together for the deletion of an operation or set of operations.

6.2.2 Strategy Options

Strategy options, as a whole, consist of all feasible strategies which may be used to delete an operation and alter prime source units. These may be used in conjunction

with other strategies for the purpcse of deleting operations. The various types have been described in section 5.4.

It is important to note that the ay not be a unique method for the deletion of an operation. There may exist more than one strategy to delete an operation as well as more than one option to alter a point. Hence, it is necessary to identify all possible sets of strategies which may be employed in the deletion of an operation.

6.2.3 Strategy Points

In crder to delete an operation or set of operations, it may be necessary to alter another or other operations. The concept of strategy points is helpful for this alteration process. Basically, strategy pcints are those locations at which units must be altered by replacement, moved, or the operation containing the point must be deleted, in order to delete an operation.

Definition 6.1: A strategy point is a 3-tuple

(i , t, u)

where

i is the index of the operation which contains a strategy point;

t identifies the point as a source or sink unit

alteration and may be 'SC' or 'SK', respectfully;

u is the identifier of the unit to be altered (i.e., the uth unit of Type t, within MO(i)).

In the strategy table examples within the Appendix the notation, MO(i) SC j, is used to indicate that the strategy point invclves the alteration of the jth prime source unit. Also the notation, MO(i) SK ALL, is used to indicate that the strategy point involves the deletion of MO(i).

6.2.4 Strategy Conflicts

In order to delete a maximum number of operations, all relationships among the strategies must be determined. For this reason, strategy conflicts have been divided into the two catagories: deletion impedance conflicts and absolute strategy conflicts which are defined as follows:

Definition 6.2: Deletion Impedance Conflicts exist between those strategy pairs where one requires the null form of MO(i). The other strategy requires the nonnull form of Op(i), SC(i), and C(i).

An example of deletion impedance follows. If we had two strategies, where the first described the negated-forward deletion of a MO(i), and the second described the parallel-backward deletion of a MO(j) with respect to MO(i). Then the

second strategy would be recorded as impeding the first strategy.

Definition 6.3: Absolute Strategy Conflicts are those strategy pairs which imply different deletion approaches of the same unit definition. Also, those strategy pairs which describe different alternate source unit assignments for the same prime source unit of an operation.

For example, if a unit u was defined at a MO(i) and MO(i) was a deletion candidate. All alternate unit assignment strategies describing the replacement of a prime source unit by unit u defined at MO(i), would be in absolute conflict with any operation deletion strategy for MO(i).

6.3 Representation of Analysis Variables

All strategy variables are represented independently, in tabular form, within a strategy table (ST).

6.3.1 Strategy Table

The strategy table is a four part table containing: first, the information on each strategy and strategy point; and second, three matrices designed to represent the variables described previously. The first part consists of a list of strategy information with fields as defined in

section 5.4.4, and strategy point information as defined in section 6.2.3. The three matrices consist of a strategy conflict matrix, a strategy point matrix, and a strategy option matrix.

6.3.1.1 Strategy Conflict Matrix (SCM)

This matrix is used to represent the conflict relationships among all the strategies. Given that Ns feasible strategies are found, the SCM will then be at most an $(Ns)^2$ array. Each element will contain one of the following values:

SCM(i,j) = 0, if there exists no conflict or impedence between strategies i and j;

SCM(i,j) = 1, if strategy i is in absolute conflict with respect to strategy j;

SCM(i,j) = 2, if strategy i impedes the operation deletion described by strategy j;

SCM(i,j) = 3, if strategy j impedes the operation deletion described by strategy i;

SCM(i,j) = 4, if strategy i impedes the operation deletion described by strategy j, and strategy j impedes the operation deletion described by

strategy i.

6.3.1.2 Strategy Point Matrix (SPM)

The purpose of this matrix is to represent those strategy points which require alteration in order that a strategy may be applied. Given that there are Ns strategies and Np points, the SPM will be at most an Ns by Np array. Each element of the SPM will contain one of the following values:

SPM(i,j) = Ø, if strategy i does not require the alteration of strategy point j;

SPM(i,j) = j, if strategy i requires the alteration of strategy point j.

6.3.1.3 Strategy Option Matrix (SOM)

The strategy option matrix is designed to represent all options available for the alteration of each strategy point. If it is found that there exists Np points and Ns strategies, then the SOM will be at most an Np by Ns array. Each element will contain one of the following values:

SOM(i,j) = Ø, if strategy j may not be used to alter strategy point i;

SPM(i,j) = j, if strategy j may be used to alter strategy point i.

6.3.2 Strategy Table Implementation

The strategy table was designed and implemented as two linear lists. The first list contained the strategy type, strategy conflict, and strategy point information. The second list contained the strategy point and strategy option information.

The first list structure was designed as a linked list of records. Each record consisted of: a link to the next such record, strategy type information as described in definition 5.4.4, a reference to a vector of records representing a row of the SCM, and a reference to a vector of records representing a row of the SPM. Each element within a vector representing a row of the SCM consisted of: a strategy identifier, a variable indicating the type of conflict, and a reference to the next element. The elements within a vector representing a row of the SPM consisted of a strategy identifier and a reference to to the next element.

The second list structure was also designed as a linked list of records. However in this case, each record consisted of a link to the next such record, strategy point information as described in definition 6.1, and a reference

to a list of vectors. Each vector contained a reference to the next vector in the list, an identifier of the strategy which may be used to alter the specified point, and identifiers of those strategies which may use the particular option.

6.4 Identification of the Strategy Conflict Variables

The identification of the conflict relationshps among strategies is performed during the analysis of the feasibility of each strategy. This process, as a whole, involves three phases depending on the type of strategy and whether the strategy requires the alteration of a strategy point. The three phases described next were not merged into the algorithms of Chapter 5 in order to simplify their discussion.

6.4.1 Phase 1

This phase involves the recognition of absolute strategy conflicts which occur when strategy pairs differ in how a strategy point may be altered. Phase 1 is executed each time a set of strategy options is assigned to a strategy point for a particular strategy. At this time, the strategy is set as being in absolute conflict with those strategies which do not use the same option(s). This is

trivial to do, for as described previously in section 6.3.2, each allocated strategy option is followed by a vector of strategy identifiers. These are the strategies which may use the particular option to alter the strategy point.

6.4.2 Phase 2

Phase 2 involves the examination of each operation deletion strategy each time an operation deletion strategy is aralyzed. In the case of parallel operation deletion strategies, this phase is executed after it has been determined whether or not the operation being deleted has the same sink set as the operation with respect to which it is being deleted. During the analysis of an operation deletion strategy i, the following is executed with each feasible operation deletion strategy j, i ≠ j:

Algorithm 6.A

Strategy i is set as being in absolute conflict with strategy j, if one of the following conditions is satisfied:

1). If strategy i and j describe the deletion of the same operation;

2). Both strategies i and j are parallel operation deletion strategies without fan-out, both describe

an operation deletion with respect to the same operation, and neither operation being deleted has an equal sink set as the operation with respect to which the operation is being deleted.

Otherwise, strategy i is set as impeding strategy j, if both of the following conditions are satisfied:

1). Strategy i is not a negated-forward operation deletion strategy;

2). Strategy j describes the operation deletion of the operation with respect to which the operation described by strategy i is to be deleted.

6.4.3 Phase 3

This phase involves the examination of all strategies each time a different alternate unit assignment strategy is analyzed. Also, this phase is in two parts depending on whether or not the strategy describes a Type A alternate unit assignment. When an alternate unit assignment strategy i is analyzed and strategy i describes the assignment of a Type A alternate source unit, the following is executed with each strategy j, i ≠ j:

Algorithm 6.B.

Strategy i is set as being in absolute conflict with strategy j, if one of the following conditions is satisfied:

1). Strategy j describes the operation deletion of the operation which initially defined the unit. Also, strategy j is a negated-forward operation deletion strategy or a parallel deletion strategy not with respect to the operation which will define the unit of strategy i;

2). Strategy j descibes the assignment of another alternate unit for the same prime source unit as described in strategy i;

3). Strategy j describes the assignment of a non-Type A alternate source unit defined by the operation which initially defined the alternate Type A unit described in strategy i.

Strategy i is set as impeding strategy j, if one of the following conditions is satisfied:

1). Strategy j describes the operation deletion of the operation which will define the alternate source unit described in strategy i;

2). Strategy j describes the operation deletion of the

operation at which the alteration is to be made, and is not the same operation which initially defined the alternate source unit described in strategy i.

Otherwise when a strategy i is being analyzed and strategy i describes the assignment of a non-Type A alternate source unit, the following is executed with each strategy j, i ≠ j:

Algorithm 6.c

Strategy i is set as being in absolute conflict with strategy j, if one of the following conditions is satisfied:

1). Strategy j describes the deletion of the operation which defined the alternate source unit of strategy i;

2). Strategy j describes the assignment of an alternate source unit to the same prime source unit as strategy i;

3). Strategy j describes the assignment of a Type A alternate unit defined by an operation which initially defined the alternate source unit of strategy i.

Otherwise, strategy i is set as impeding strategy j, if

strategy j describes the operation deletion of the operation at which strategy i is to be applied.

6.5 Summary

The purpose of this chapter has been to describe the break down of information on deletion candidates such that sub-sets of strategies may be combined in order to delete a maximum number of operations.

A description of the identification of the conditions for strategy pairs to be in conflict has also been given for the first time. These were tested during the implementation of the program.

The data structure designed to store the analysis variables has also been discussed.

The following chapter will discuss how the program combines strategies in order to delete a maximum number of operations.

CHAPTER VII

Strategy Selection and Application

7.1 Introduction

In this chapter, the process of determining and applying those strategy sets which will result in the deletion of a maximum number of operations will be described in detail.

7.2 General Strategy Selection Approach

The objective function of the strategy selection process is to determine those combinations of operation deletion strategies and if necessary, also alternate unit assignment strategies, which if applied, result in the deletion of a maximum number of operations. The outcome will be a set of optimized microprogram versions which are logically correct with respect to the original microprogram.

In general, a set D of all feasible operation deletion strategies and a set FA, of all feasible alternate unit assignment strategies have been determined. Also, the conflict relationships among all the strategy pairs, the

strategy points, and all the options to alter each strategy point are known.

Using the set D, a set G, consisting of a maximum number of nonconflicting operation deletion strategies which result in the deletion of s operations is determined. Then for the set G, another set G' is determined, consisting of all strategies required to alter the points as described by the strategy set G, and also by the strategy set G' itself. During this time a set G" is also accumulated and will consist of all operation deletion strategies used to alter prime source unit points.

Once the set G' is determined, the sets G and G' are merged together into one set H of distinct strategies. The set H if applied, will result in the deletion of (t - v) operations. The variable t is the number of operation deletion strategies in the set H, and v is the number of operation deletion strategies which are impeded by another operation deletion strategy in the set H.

In order to insure that s operations will be deleted, the following four constraints are checked during and after the latter process:

1). All points defined for G and G' are altered by G'.

2). All operation deletion strategies accumulated in G" must not be impeded by any strategy in H.

3). No absolute conflicts may exist in H.

4). $(t - v) \geq s$.

The latter four constraints are also necessary and sufficient for the determination of the set G^s.

7.3 Solution Framework of the Strategy Selection Algorithms

The process of selecting and applying those strategy sets which result in the deletion of a maximum number of operations involves three general phases.

7.3.1 Phase 1

By using the set D, consisting of all feasible operation deletion strategies, all sets of nonconflicting operation deletion strategies are determined. Each set is placed according to its size, into a list of lists

$$L = l_n , l_{n-1} , \ldots , l_1 ,$$

such that the sub-lists are in decreasing order of the number of operations to be deleted by the strategy sets within each sub-list. Each sub-list

$$l_m = k_1 , k_2 , \ldots , k_p ,$$

is a list of sets containing m strategy identifiers of strategies to be used to delete m operations (i.e., distinct sets as G described in section 7.2).

In this phase, nonconflicting strategy sets means sets in which there exists no absolute or impedance conflicts among all strategy pairs.

7.3.2 Phase 2

This phase involves the execution of phase 3 with each sub-list generated by phase 1. The process starts with the sub-list of strategy sets which describe the deletion of a maximum number of operations. It continues accordingly while phase 3 has not found any optimized microprogram version(s), or until no sub-list is left.

7.3.3 Phase 3

In this phase, each set within the given sub-list is evaluated in order to determine if one or more G' sets exist, as described in section 7.2. If a set G' is determined, and if the constraints hold for G, G', and H; then the final set H is applied to a copy of the given microprogram segment and the copy is printed. Again note that this process is highly recursive as there may exist many options to alter any one point. Hence a set G may combine with several distinct G's and thus more than one final strategy set may be obtained with each set G. Each optimized microprogram version presented in the Appendix is

given by showing the initial strategy set G, and also the final strategy set H.

7.4 Implementation of the Nonconflicting Strategy Set List

The list of lists as described in section 7.3.1, was implemented as a linked list of records. Each record was designed to contain: a reference to the next such record, an integer indicating the size of the sets, and a reference to the list of records representing the strategy sets. Each set of records representing a strategy set, consisted of a record containing a reference to the next set of records and a reference to a vector of records. Each record of a vector contained a strategy identifier and a reference to the next vector entry.

In the following section, the notation L(i) will be used to identify the ith sub-list, L(i,j) to denote the jth set in sub-list i, and L(i,j)[k] to denote the kth strategy within L(i,j).

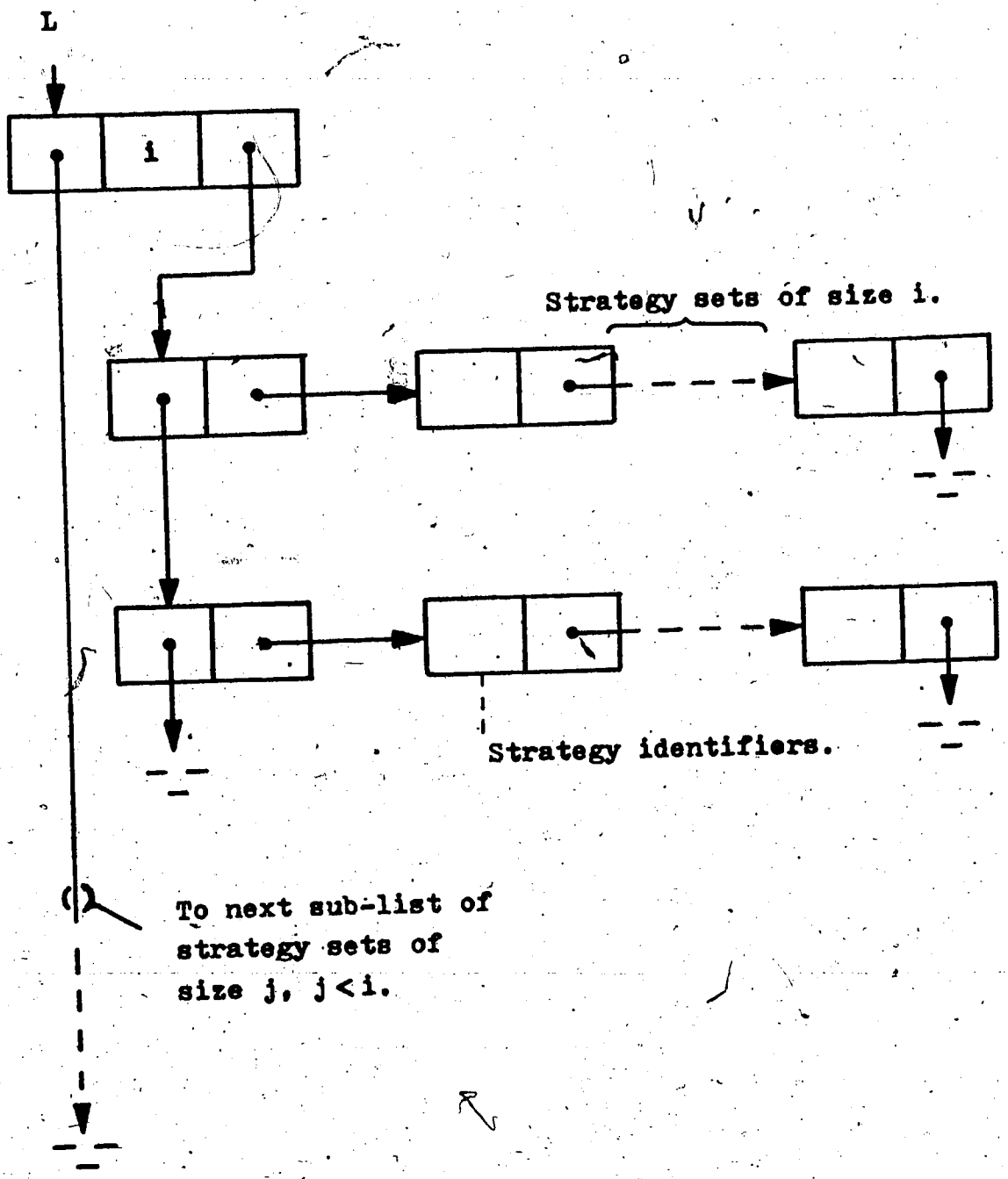


Figure 7.1 Implementation of nonconflicting strategy set list

7.5 Strategy Selection and Application Algorithms

The purpose of this section is tc present those algorithms used to determine and apply the strategy sets which delete the greatest number of operations. The following recursive algorithm was designed to determine the nonconflicting operation deletion strategy sets. By nonconflicting it is meant that no strategy in a resulting set impedes another strategy or is in absolute conflict with another strategy in the set.

Algorithm 7.A

By using the set of all feasible operation deletion strategies

$D = \{ d_1, d_2, \ldots d_n \}$,

found in the ST. The sets of nonconflicting operation deletion strategies are found by executing the following steps:

A.1 Let i:=0.

A.2 Let i:=i+1.

A.3 Store D into the sub-list L(i-1) and stop if i > |D|, or else continue with step A.4

A.4 Let j:=0.

A.5 Let j:=j+1.

A.6 Go to step A.2 if j > |D|.

A.7 If SCM(D(i),D(j)) = 0, then go to A.5, or else let D' become equal to the elements of D except for D(i), D" become equal to the elements of D except for D(j), then recursively apply this algorithm to the strategy sets D' and D", and then stop.

The following algorithm was designed and is used to pass each sub-list as described in phase 2 of section 7.3.2 to algorithm 7.C. At this time it should be pointed out that algorithm 7.A generates primarily the largest nonconflicting strategy sets for which possibly there may not exist a final strategy set H. Hence, in algorithm 9.B, step B.5, all distinct sets of size i-1 are generated using the sets of size i if no optimized microprogram version was found.

Algorithm 7.B

B.1 Let i:=0.

B.2 Let i:=i+1.

B.3 Stop if i > |L| or if an optimized microprogram version has been found.

B.4 Execute algorithm 7.C using the list L(i).

B.5 Using L(i), generate all subsets of size (i - 1) and store them into L(i - 1) if not already done so, and if no optimized microprogram version has been found.

B.6 Repeat with step B.2.

Algorithm 7.C is the first of a set of recursive algorithms designed and used to find the set G' for each set of nonconflicting operation deletion strategies. In the following algorithm the variables G, G', and G" are as described in section 7.2. The variables E and E' represent the set of strategies whose points are currently being resolved and the set of strategies whose points will be resolved, respectively.

Algorithm 7.C

C.1 Let j:=0.

C.2 Let j:=j+1,

C.3 Stop if j > |L(i)|.

C.4 Let E:=L(i,j), E':= ∅, G':= ∅, G":= ∅, k:=1, and execute algorithm 7.D with the parameter set (E, E', L(i,j), G', G", k). Then continue with step C.2.

Algorithm 7.D

D.1 Execute algorithm 7.F if E = ∅ and E' = ∅ using the parameter set (L(i,j), G').

D.2 Let E:=E', E':= ∅, k:=1, and recursively apply this algorithm using the parameter set (E, E', L(i,j), G',

G", k), if k > |E|.

D.3 let m:=1.

D.4 Apply algorithm 7.E using the parameter set (E, E', L(i,j), G', G", k, m) and then stop.

Algorithm 7.E

E.1 If (m-1) was the last strategy point, let k:=k+1 and apply algorithm 7.D using the parameter set (E, E', L(i,j), G', G", k) and stop, or else continue with step E.2.

E.2 Go to step E.3 if SPM(E(k),m) ≠ Ø, or else let m:=m+1 and go to step E.1.

E.3 Execute steps E.4 through E.8 with each option at strategy point m and then stop.

E.4 Continue with E.3 if the current option is in absolute conflict with a strategy in L(i,j) or G'; if the option impedes a strategy in G"; or if the strategy point is a prime source unit alteration, the current option is an operation deletion strategy, and the option is impeded by a strategy within L(i,j) or G'.

E.5 Let NG' become equal to G' plus the current strategy option if it is not already in G'.

E.6 Let NG" become equal to G" plus the current strategy option if the point is a prime source unit alteration,

if the option is an operation deletion strategy, and if it is not already in G".

E.7 Let NE', become equal to E' plus the new option if it is not within E, E' or L(i,j) already.

E.8 Let m:=m+1 and recursively apply this algorithm with the parameter set (E, NE', L(i,j), NG', NG", k, m).

Algorithm 7.F

F.1 Let H become equal to the set of all distinct strategies within L(i,j) and G'.

F.2 If the number of operation deletion strategies in H minus the number of operation deletion strategies which are impeded by another operation deletion strategy in H, is greater than or equal to the number of operation deletion strategies in L(i,j), then execute F.3, or else stop.

F.3 Make a copy of the given microprogram node, apply the strategies to the copy, print the copy as the next microprogram version, and then finally stop.

7.6 Application of Strategies

The application of the final strategy set H to a copy of the given microprogram involves the application of one of the following steps for each strategy in H, depending on the

the type of strategy.

1). For negated-forward operation deletion strategies and parallel strategies of operations which have equal sink units as the operation with respect to which they are being deleted, all that needs to be done is to delete the sink units of the operation being deleted.

2). For the rest of the operation deletion strategies, the sink set of the operation being deleted is erased. Also, by using the original microprogram, the sink set of the operation being deleted is copied to the operation with respect to which the operation is being deleted.

3). Alternate unit assignment strategies involve only the replacement of the specified prime source unit by the referenced alternate unit.

Note that care must be taken to delete the correct number of sink units. The application of a parallel strategy may have already resulted in the movement of sink units to a particular operation. This is avoided by using the original microprogram as a template.

After each strategy has been applied, those operations which have had their sink set altered, are checked and

deleted if no sink set is found. Samples of optimized micro-program versions can be found in the Appendix and more versions may exist for the examples as a cut off level was imposed on the computer output.

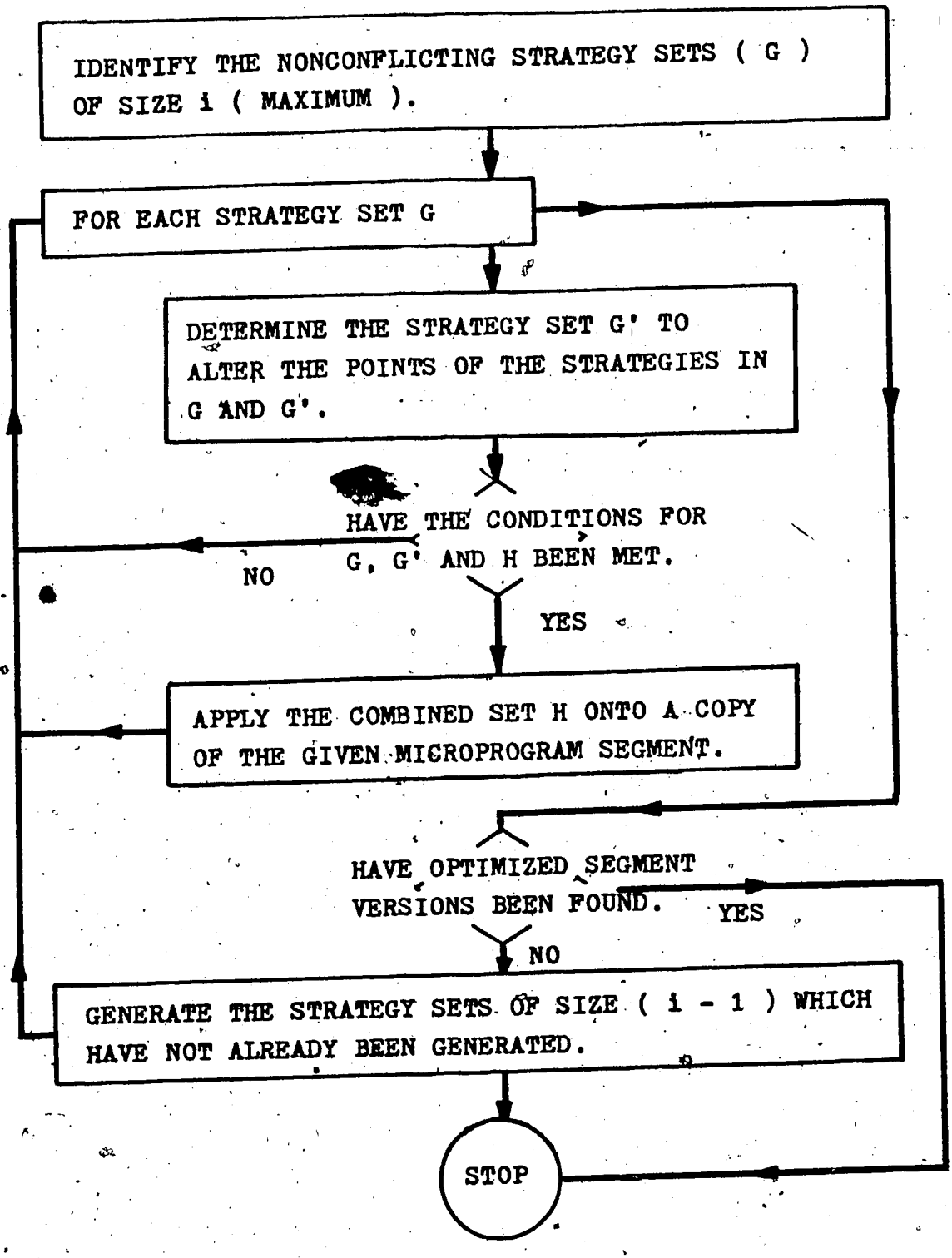


Figure 7.1 Flow diagram of strategy selection

7.7 Summary

The purpose of this chapter has been to describe and present the algorithms developed in order to find and apply those strategy sets which result in the deletion of a maximum number of operations.

From the experimentation with the implementation, it was learned that this final phase of determining the strategy sets requires the greatest amount of time. This time was far in excess of the time required for the phases leading up to the selection and determination of strategy sets. The reason is primarily due to the generation of numerous options to alter prime source units from the identification of equivalent memory units.

CHAPTER VIII

Summary And Recommendations

In this thesis, an implementation study on the optimization of microprogram segments has been presented as a practical extension of the work found in Sitton[S1]. It is felt that much has been learned about the peculiarities of microprogram optimization.

As a result of the preliminary work required for this research, the author has come to feel that previous work on microprogram optimization has been rather unfruitful. That is to say, optimization strategies have not been adequately explored, formalized, nor incorporated into any existing compiler to any reasonable degree. The reasons for this lack of results in microprogram optimization seem to be: the relative infancy of microprogramming in practice, the limited development of formal definitions describing the microprogramming environment, and because of the lack of productive results in the development and especially implementation of microprogramming languages.

With the available knowledge of software optimization and microprogram optimization, it would not be unreasonable to see the development of a general but highly intensive

optimizer within a very short period of time. The optimizer would be capable of accepting machine architectural definitions, language definitions, user interactive information, and user micro-code; in order to generate truly optimized micro-code. The implementation would probably use developed software techniques, engineering flow analysis techniques, concepts developed within this study, etc.

One of the next most probable undertakings in this field will be the analysis of branch points. The author feels that this research will probably be along the lines of ordering microprogram forms [E1]. This would be done by varying the number of loops and the order of loops as well as by the localization of regions and segments.

It should be pointed out that the process of identifying alternate operation forms need not be limited to single operations. The process could easily be applied to the generation of alternate region forms and/or segments including procedures (i.e. subroutines). This would possibly increase the ability to identify nonessential regions and even possibly lead the way to altering the form of procedures or common code. This might also decrease the number of branches and/or increase the efficient utilization of control space.

A very interesting line of research would be on the

identification of equivalent microprograms, and the testing of the correctness of microprograms using the concept of identifying alternate operation forms as described within this study. If one could generate all possible alternate operation forms of a pair of microprograms, some degree of equivalence among them may be obtained. By using a similar technique, it may be possible to determine a degree of correctness of a microprogram. An example of this type of research on loop-free microprograms can be found in Ramamoorthy and Shankar[R1]. Still, much work remains and the basic principles could be transferred to the study of the equivalence of machines at the microprogramming level.

Also, much work using the concepts discussed within this study needs to be done at the global optimization level. The end result would make it possible to optimize whole microprograms rather than just a microprogram segment.

Having studied the optimization of microprogram segments thoroughly, the author feels that the consideration of the following steps might be of some value in the design of a complete optimizer.

1). The first step would break a given microprogram into computationally simpler parts (i.e., regions) in order to identify nonessential regions which would be deleted immediately. This

would be done by evaluating branch points.

2). The second step would attempt to reorder the regions by analyzing the branch points and would generate one or more microprogram forms with the original form included in the count. The resulting forms would also be ordered in order of the most probable optimal forms.

3). The third step would generate old and new microprogram forms by using all possible combinations of the given and optimized segments. This step would us the process described within this study, on each microprogram form obtained from step 2.

4). The fourth step would generate old and new forms by applying global analysis to each microprogram form obtained from step 3.

5). The fifth and final step would generate one or more sets of micro-code for each microprogram form obtained from step 4. It would make sure that each set of generated code utilizes the hardware resources and data paths as efficiently as possible. The parallelism of the machine and the time validities would be taken into account during this step.

It would seem that the latter steps would require a considerable amount of processing. This may be true but it

would naturally be assumed that consistent attempts would be made to delete those forms having a lesser degree of optimality. Also, run time statistics, architectural constraint information, user interactive information, etc. would also be used during the whole optimization process.

It is strongly felt that research on the formalization and application of the steps just described will result in the development of an optimizer which is desperately needed today. The knowledge gained from such an endeavor would be invaluable in itself.

Bibliography

[A1] A. K. Agrawala and T. G. Rauscher, "Microprogramming: Perspective and Status," IEEE Trans. On Comp., Vol. C-23, No. 8, August 1974, pp. 817-837.

[A2] F. E. Allen and J. Cocke, "A Catalogue of Optimizing Transformations," in Design and Optimization of Compilers, R. Rustin (Ed), Courant Computer Science Symp. No. 5, Prentice-Hall, Englewood-Cliffs, N.J., 1972.

[D1] S. Dasgupta and J. Tartar, "On the Minimization of Control Memories," Information Prccessing Letters, Vol. 3, No. 3, January 1975, pp. 71-74.

[E1] C. P. Earnest, K. G. Blake, and J. Anderson, "Analysis of Graphs by Ordering of Nodes," Journal of the ACM, Vol. 19, No. 1, January 1972, pp. 23-42.

[G1] D. Gries, Compiler Construction for Digital Computers. John Wiley and Sons, Ltd., 1971.

[H1] S. S. Husscn, Microprogramming: Principles and Practices. Prentice-Hall, Englewood Cliffs, N.J., 1970.

[J1] L. W. Jackson and S. Dasgupta, "An Algorithm for

Identifying Parallel Micro-operations," Tech. Rep. TR73-20, Dept. Of Comp. Sci., Univ. Of Alberta, Edmonton, Alberta, Canada, December 1973.

[J2] L. Jones,et.al., "Microprogramming Bibliography, 1951-Early 1974," SIGMICRO Newsletter, Special Issue, September 1974.

[K1] R. L. Kleir, "A Representation for the Analysis of Microprcgram Operations," The Seventh Annual Workshop Microprogramming, Palo Alto, California, September 30, 1974.

[K2] R. L. Kleir and C. V. Ramamoorthy, "Optimization Strategies for Microprograms," IEEE Trans. On Comp., Vol. C-20, No. 7, July 1971, pp. 783-795.

[L1] E. S. Lowry and C. W. Meddock, "Object Code Optimization," Comm. Of the ACM, Vol. 12, No.1, pp. 13-22, January 1969.

[P1] J. L. Pfaltz, "Graph Structures," Journal of the ACM, Vol. 19, No. 3, July 1972, pp. 411-422.

[R1] C. V. Ramamoorthy and K. S. Shankar, "Automatic Testing for the Correctness and Equivalence of Loopfree Microprograms," IEEE Trans. On Comp., Vol C-23, No. 8, Agust 1974.

[R2] C. V. Ramamoorthy and M. Tsuchiya, "A High-Level Language for Horizontal Microprcgramming," IEEE Trans. On Comp., Vol. C-23, No. 8, August 1974,

pp. 791-801.

[R3] S. R. Redfield, "A Study In Microprogrammed Processors: A Medium Sized Microprogrammed Processor," IEEE Trans. On Comp., Vol. C-20, No. 7, July 1971.

[R4] R. Rosin, R. F. Frieder, and R. H. Eckhouse Jr., "An Environment For Research In Microprogramming And Emulators," Comm. Of The ACM, Vol. 15, No. 8, August 1972, pp. 748-760.

[S1] W. G. Sitton, "Strategies for Microprogram Optimization," Tech. Rep. TR73-4, Dept. Of Comp. Sci., Univ. Of Alberta, Edmonton, Alberta, Canada, October 1973.

[T1] J. Tartar, "Analysis And Optimization Techniques For Microprogram Forms," Second Texas Conference On Computing Systems, Austin Texas, June 1973.

[T2] J. Tartar and S. Dasgupta, "A Probabilistic Model for the Evaluation of Microprogram Performance," The Seventh Annual Workshop on Microprogramming, Palo Alto, California, September 30th, 1974.

[T3] M. Tsuchiya and M. J. Gonzalez, Jr., "An Approach to Optimization of Horizontal Microprograms," The Seventh Annual Workshop on Microprogramming, Palo Alto, California, September 30th, 1974.

APPENDIX

Optimization Examples

The following consists of results obtained from optimizing two sample microprograms. Optimization was performed without and with fan-out for each sample microprogram, respectively. The optimization results where obtained using the implementation developed and written in ALGOLW. The implementation was run on an IBM 360/67 at The University of Alberta.

The first microprogram was optimized from 8 micro-operations to 6 operations and the second was optimized from 6 micro-operations to 4 micro-operations. These results show approximately a 25 percent and 33 percent reduction in the number of operations, respectively. The approximate time required for the results was between 10 to 15 seconds without fan-out and from 60 to 80 seconds with fan-out. This amount of time may seem excessive but it must be stressed that the implementation is very general and also, practical constraints would limit the alternate possibilities considerably.

Diagrams of all the tables are presented with the data for each set of results. Also, a translation of each strategy table is presented.

Example from "Strategies for Microprogram Optimization," by Wesley Gary Sitton, Technical Report TR73-4, April 1973, pp. 36.

ORIGNINAL MICROPROGRAM

MICRO-INST. MICRO-OP., OP. CODE, SOURCE SET, SINK SET, CONTROL SET, OPERATION UNITS SET, TIME VALIDITY SET.

1.1, GATE, {2}, {3}.
2.1, GATE, {4}, {6}.
3.1, ADD, {3,6}, {7}.
4.1, ADD, {1,2}, {2}.
5.1, ADD, {2,4}, {8}.
6.1, GATE, {7}, {5}.
7.1, ADD, {2,6}, {9}.
8.1, SUB, {7,3}, {3}.

REGION PARAMETERS

FIRST MICRO-INS. = 1
FIRST MICRO-OP. = 1
LAST MICRO-INS. = 8
LAST MICRO-OP. = 1

(WITH NO FAN-OUT)

UNIT ASSIGNMENT TABLE

INDEX — MEMORY SUB-UNITS .....

	1	2	3	4	5	6	7	8	9
MO(1.1)	0	1	2	0	0	0	0	0	0
MO(2.1)	0	0	0	1	0	2	0	0	0
MO(3.1)	0	0	1	0	0	1	2	0	0
MO(4.1)	1	3	0	0	0	0	0	0	0
MO(5.1)	0	1	0	1	0	0	0	2	0
MO(6.1)	0	0	0	0	2	0	1	0	0
MO(7.1)	0	1	0	0	0	1	0	0	2
MO(8.1)	0	0	3	0	0	0	1	0	0
EXIT	2	2	1	2	2	2	2	1	2

ALTERNATE UNIT ASSIGNMENT TABLE.

	SOURCE 1	SOURCE 2	SINK 1
HO(1.1)	(2, 0.0)		(3, 1.1)
HO(2.1)	(4, 0.0)		(6, 2.1)
HO(3.1)	(3, 1.1) (C, 2, 0.0)	(6, 2.1) (C, 4, 0.0)	(7, 3.1) (S, 6.1)
HO(4.1)	(1, 0.0)	(2, 0.0) (C, 3, 1.1)	(2, 4.1)
HO(5.1)	(2, 4.1)	(4, 0.0) (C, 6, 2.1)	(8, 5.1) (S, 7.1)
HO(6.1)	(7, 3.1) (A, 5, 3.1, 6.1)		(5, 6.1) (P, 3.1)
HO(7.1)	(2, 4.1)	(6, 2.1) (C, 4, 0.0)	(9, 7.1) (P, 5.1)
HO(8.1)	(7, 3.1) (A, 5, 3.1, 6.1) (A, 7, 6.1, 3.1) (C, 5, 6.1)	(3, 1.1) (R, 2, 0.0)	(3, 8.1)

DELETION CANDIDATES

NEGATED-FORWARD DELETION CANDIDATES

1). MO(1.1) WITH RESPECT TO MC(8.1)

2). MO(2.1) WITH RESPECT TO MC(N+1)

3). MO(3.1) WITH RESPECT TO MC(N+1)

4). MO(4.1) WITH RESPECT TO MC(N+1)

5). MO(6.1) WITH RESPECT TO MC(N+1)

6). MO(7.1) WITH RESPECT TO MC(N+1)

PARALLEL-FORWARD DELETION CANDIDATES

1). MO(3.1) WITH RESPECT TO MC(6.1)

2). MO(5.1) WITH RESPECT TO MC(7.1)

PARALLEL-BACKWARD DELETION CANDIDATES

1). MO(6.1) WITH RESPECT TO MC(3.1)

2). MO(7.1) WITH RESPECT TO MC(5.1)

STRATEGY OPTIONS AND STRATEGY POINTS

INDEX	OPTIONS ACTION OR ALTERNATE	OPTION	ACTION

POINTS INDEX	POINT TYPE	

1). MO(1.1) NF MC(8.1)
(31). MO(8.1) SC 2
(19). MO(3.1) SC 1

2). MO(2.1) NF MO(N+1)
(28). MO(7.1) SC 2
(20). MO(3.1) SC 2

3). MO(3.1) NF MC(N+1)
(30). MO(8.1) SC 1
(25). MO(6.1) SC 1

4). MO(3.1) PF MO(6.1)
(25). MO(6.1) SC 1
(26). MO(6.1) SK ALL

5). MO(4.1) NF MO(N+1)
(27). MO(7.1) SC 1
(23). MO(5.1) SC 1

6). MO(5.1) [illegible] MO(7.1)
(29). MO(7.1) SK ALL

7). MO(6.1) NF MO(N+1)

8). MO(6.1) PB MO(3.1)
(21). MO(3.1) SK ALL

9). MO(7.1) NF MC(N+1)

10). MO(7.1) PB MO(5.1)
(24). MO(5.1) SK ALL

11). (C,2,0.0) SC 1 MO(3.1)

12). (C,4,0.0) SC 2 MO(3.1)

13). (A,5,3.1,6.1) SC 1 MO(6.1)

```
      (26). MO(6.1)    SK   ALL

14). (C,4,0.0)    SC  2    MO(7.1)

15). (A,5,3.1,6.1)    SC  1    MO(8.1)
     (26). MO(6.1)    SK   ALL
     (26). MO(6.1)    SK   ALL

16). (A,7,6.1,3.1)    SC  1    MO(8.1)
     (21). MO(3.1)    SK   ALL

17). (C,5,6.1)    SC  1    MO(8.1)

18). (R,2,0.0)    SC  2    MO(8.1)
     (22). MO(4.1)    SK   ALL
```

STRATEGY POINTS AND STRATEGY OPTIONS

POINTS
INDEX POINT TYPE

OPTIONS
INDEX ACTION OPTION ACTION
OR
ALTERNATE

19). MO(3.1) SC 1
(11). (C,2,0.0) SC 1 MO(3.1)
(3). MO(3.1) NF MO(N+1)
(4). MO(3.1) PF MO(6.1)

20). MO(3.1) SC 2
(12). (C,4,0.0) SC 2 MO(3.1)
(3). MO(3.1) NF MO(N+1)
(4). MO(3.1) PF MO(6.1)

21). MO(3.1) SK ALL
(3). MO(3.1) NF MO(N+1)
(4). MO(3.1) PF MO(6.1)

22). MO(4.1) SK ALL
(5). MO(4.1) NF MO(N+1)

23). MO(5.1) SC 1
(6). MO(5.1) PF MO(7.1)

24). MO(5.1) SK ALL
(6). MO(5.1) PF MO(7.1)

25). MO(6.1) SC 1
(13). (A,5,3.1,6.1) SC 1 MO(6.1)
(7). MO(6.1) NF MO(N+1)
(8). MO(6.1) PB MO(3.1)

26). MO(6.1) SK ALL
(8). MO(6.1) PB MO(3.1)
(7). MO(6.1) NF MO(N+1)

27). MO(7.1) SC 1
(9). MO(7.1) NF MO(N+1)
(10). MO(7.1) PB MO(5.1)

28). MO(7.1) SC 2
(14). (C,4,0.0) SC 2 MO(7.1)

```
( 9). MO(7.1)    NF   MO(N+1)
(10). MO(7.1)    PB   MO(5.1)

29). MO(7.1)    SK  ALL
( 9). MO(7.1)    NF   MO(N+1)
(10). MO(7.1)    PB   MO(5.1)

30). MO(8.1)    SC  1
(15). (A,5,3.1,6.1)    SC  1    MO(8.1)
(16). (A,7,6.1,3.1)    SC  1    MO(8.1)
(17). (C,5,6.1)    SC  1    MO(8.1)

31). MO(8.2)    SC  2
(18). (R,2,0.0)    SC  2    MO(8.1)
```

STRATEGY CONFLICTS AND IMPEDIMENTS.

	ACTION OR ALTERNATE	OPTION	ACTION
1).	NO(1.1)	NF	NO(8.1)
2).	NO(2.1)	NF	NO(N+1)
3).	NO(3.1)	NF	NO(N+1)
4).	NO(3.1)	PF	NO(6.1)
5).	NO(4.1)	NF	NO(N+1)
6).	NO(5.1)	PF	NO(7.1)
7).	NO(6.1)	NF	NO(N+1)
8).	NO(6.1)	PF	NO(3.1)
9).	NO(7.1)	NF	NO(N+1)
10).	NO(7.1)	PF	NO(5.1)
11).	(C, 2, 0.0)	SC 1	NO(3.1)
12).	(C, 4, 0.0)	SC 2	NO(3.1)
13).	(A, 5, 3.1, 6.1)	SC 1	NO(6.1)
14).	(C, 4, 0.0)	SC 2	NO(7.1)
15).	(A, 5, 3.1, 6.1)	SC 1	NO(8.1)
16).	(A, 7, 6.1, 3.1)	SC 1	NO(8.1)
17).	(C, 5, 6.1)	SC 1	NO(8.1)
18).	(B, 2, 0.0)	SC 2	NO(8.1)
19).	NO(3.1)	SC 1	
20).	NO(3.1)	SC 2	
21).	NO(3.1)	SK ALL	
22).	NO(4.1)	SK ALL	
23).	NO(5.1)	SC 1	
24).	NO(5.1)	SK ALL	
25).	NO(6.1)	SC 1	
26).	NO(6.1)	SK ALL	
27).	NO(7.1)	SC 1	
28).	NO(7.1)	SC 2	
29).	NO(7.1)	SK ALL	
30).	NO(8.1)	SC 1	
31).	NO(8.1)	SC 2	

POINT(S)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1).	0	0	0	0	0	0	0	0	0	0	0	0	C	0	0	C	0	0	19												31
2).	0	0	0	0	0	0	C	0	0	0	0	0	0	0	0	0	0	0		20								28			
3).	0	0	0	1	0	C	0	3	C	C	3	3	3	0	3	1	0	0							25					30	
4).	0	C	1	0	0	0	2	4	0	0	3	3	3	C	3	C	0	0							25	26					1
5).	C	0	0	0	0	0	0	0	0	0	0	0	C	C	C	0	0	0					23				27				
6).	0	0	C	0	0	C	0	0	2	4	0	C	C	0	0	C	0	C											29		
7).	0	0	0	3	0	0	C	1	C	0	0	0	1	0	1	3	1	0													
8).	0	0	2	4	0	0	1	0	C	0	0	0	C	0	0	1	1	0			21										
9).	0	0	C	0	0	3	0	0	C	1	0	0	0	3	0	0	0	C													
10).	0	0	0	0	0	4	0	0	1	C	0	0	C	3	0	C	0	0							28						
11).	C	0	2	2	0	0	0	0	0	0	0	0	C	0	0	0	0	0													
12).	0	0	2	2	0	0	C	0	0	C	0	0	0	C	0	0	C	C													
13).	0	C	2	2	0	0	1	0	0	0	0	0	0	C	0	0	1	C								26					
14).	0	0	0	0	0	0	0	0	2	2	0	0	0	0	0	0	0	0													
15).	0	0	2	2	0	C	1	0	0	0	0	C	C	0	0	1	1	0								26					
16).	0	0	1	0	0	C	2	1	0	0	0	0	C	0	1	0	1	0			21										
17).	0	C	C	0	0	0	1	1	C	0	0	0	1	0	1	1	0	0													
18).	0	0	C	0	0	0	0	0	0	0	0	0	0	C	0	0	C	C				22									
19).			3	4							11																				
20).			3	4								12																			
21).			3	4																											
22).				5																											
23).						6																									
24).						6																									
25).							7	8						13																	
26).							7	8																							
27).									9	10																					
28).									9	10				14																	
29).									9	10																					
30).																15	16	17													
31).																		18													

OPTIMIZED MICROPROGRAM VERSIONS

MICRO-INST. MICRO-OP., OP. CODE, SOURCE SET, SINK SET, CONTROL SET, OPERATION UNITS SET, TIME VALIDITY SET.

Version 1
(2,8,9)
(2,3,8,9,12,13,15)

1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {5}.
4.1, ADD, {1,2}, {2}.
5.1, ADD, {2,4}, {8}.
8.1, SUB, {5,3}, {3}.

Version 2
(2,7,9)
(2,7,9,12)

1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {7}.
4.1, ADD, {1,2}, {2}.
5.1, ADD, {2,4}, {8}.
8.1, SUB, {7,3}, {3}.

Version 3
(2,6,8)
(2,3,6,8,9,12,13,14,15)

1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {5}.
4.1, ADD, {1,2}, {2}.
7.1, ADD, {2,4}, {8}.
8.1, SUB, {5,3}, {3}.

Version 4
(2,6,7)
(2,6,7,9,12,14)

1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {7}.
4.1, ADD, {1,2}, {2}.
7.1, ADD, {2,4}, {8}.
8.1, SUB, {7,3}, {3}.

Version 5
(2,3,9)
(2,3,8,9,12,13,15)

1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {5}.
4.1, ADD, {1,2}, {2}.
5.1, ADD, {2,4}, {8}.
8.1, SUB, {5,3}, {3}.

Version 6
(2,3,6)
(2,3,6,8,9,12,13,14,15)

1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {5}.
4.1, ADD, {1,2}, {2}.
7.1, ADD, {2,4}, {8}.
8.1, SUB, {5,3}, {3}.

STRATEGY OPTIONS AND STRATEGY POINTS

(WITH FAN-OUT)

OPTIONS

INDEX	ACTION	OPTION OR ALTERNATE	ACTION

POINTS

INDEX	POINT	TYPE
1).	MO(1.1)	NF MO(8.1)
	(35). MO(8.1)	SC 2
	(23). MO(3.1)	SC 1
2).	MO(2.1)	NF MC(N+1)
	(32). MO(7.1)	SC 2
	(24). MO(3.1)	SC 2
3).	MO(3.1)	NF MO(N+1)
	(34). MO(8.1)	SC 1
	(29). MO(6.1)	SC 1
4).	MO(3.1)	PF MO(6.1)
	(29). MO(6.1)	SC 1
	(30). MO(6.1)	SK ALL
5).	MO(3.1)	PF F MO(6.1)
	(29). MO(6.1)	SC 1
6).	MO(4.1)	NF MO(N+1)
	(31). MO(7.1)	SC 1
	(27). MO(5.1)	SC 1
7).	MO(5.1)	PF MO(7.1)
	(33). MO(7.1)	SK ALL
8).	MO(5.1)	PF F MO(7.1)
9).	MO(6.1)	NF MC(N+1)
10).	MO(6.1)	PB MO(3.1)
—	(25). MO(3.1)	SK ALL
11).	MO(6.1)	PB F MO(3.1)
12).	MO(7.1)	NF MC(N+1)

13). MO(7.1) PB MO(5.1)
(28). MO(5.1) SK ALL

14). MO(7.1) PB F MO(5.1)

15). (C,2,0.0) SC 1 MO(3.1)

16). (C,4,0.0) SC 2 MO(3.1)

17). (A,5,3.1,6.1) SC 1 MO(6.1)
(30). MO(6.1) SK ALL

18). (C,4,0.0) SC 2 MO(7.1)

19). (A,5,3.1,6.1) SC 1 MO(8.1)
(30). MO(6.1) SK ALL
(30). MO(6.1) SK ALL

20). (A,7,6.1,3.1) SC 1 MO(8.1)
(25). MO(3.1) SK ALL

21). (C,5,6.1) SC 1 MO(8.1)

22). (R,2,0.0) SC 2 MO(8.1)
(26). MO(4.1) SK ALL

STRATEGY POINTS AND STRATEGY OPTIONS

POINTS
INDEX POINT TYPE

OPTIONS
INDEX ACTION OPTION ACTION OR ALTERNATE

23). MO(3.1) SC 1
(15). (C,2,0.0) SC 1 MO(3.1)
(3). MO(3.1) NF MO(N+1)
(4). MO(3.1) PF MO(6.1)
(5). MO(3.1) PF F MO(6.1)

24). MO(3.1) SC 2
(16). (C,4,0.0) SC 2 MO(3.1)
(3). MO(3.1) NF MO(N+1)
(4). MO(3.1) PF MO(6.1)
(5). MO(3.1) PF F MO(6.1)

25). MO(3.1) SK ALL
(3). MO(3.1) NF MO(N+1)
(4). MO(3.1) PF MO(6.1)
(5). MO(3.1) PF F MO(6.1)

26). MO(4.1) SK ALL
(6). MO(4.1) NF MO(N+1)

27). MO(5.1) SC 1
(7). MO(5.1) PF MO(7.1)
(8). MO(5.1) PF F MO(7.1)

28). MO(5.1) SK ALL
(7). MO(5.1) PF MO(7.1)

29). MO(6.1) SC 1
(17). (A,5,3.1,6.1) SC 1 MO(6.1)
(9). MO(6.1) NF MO(N+1)
(10). MO(6.1) PB MO(3.1)
(11). MO(6.1) PB F MO(3.1)

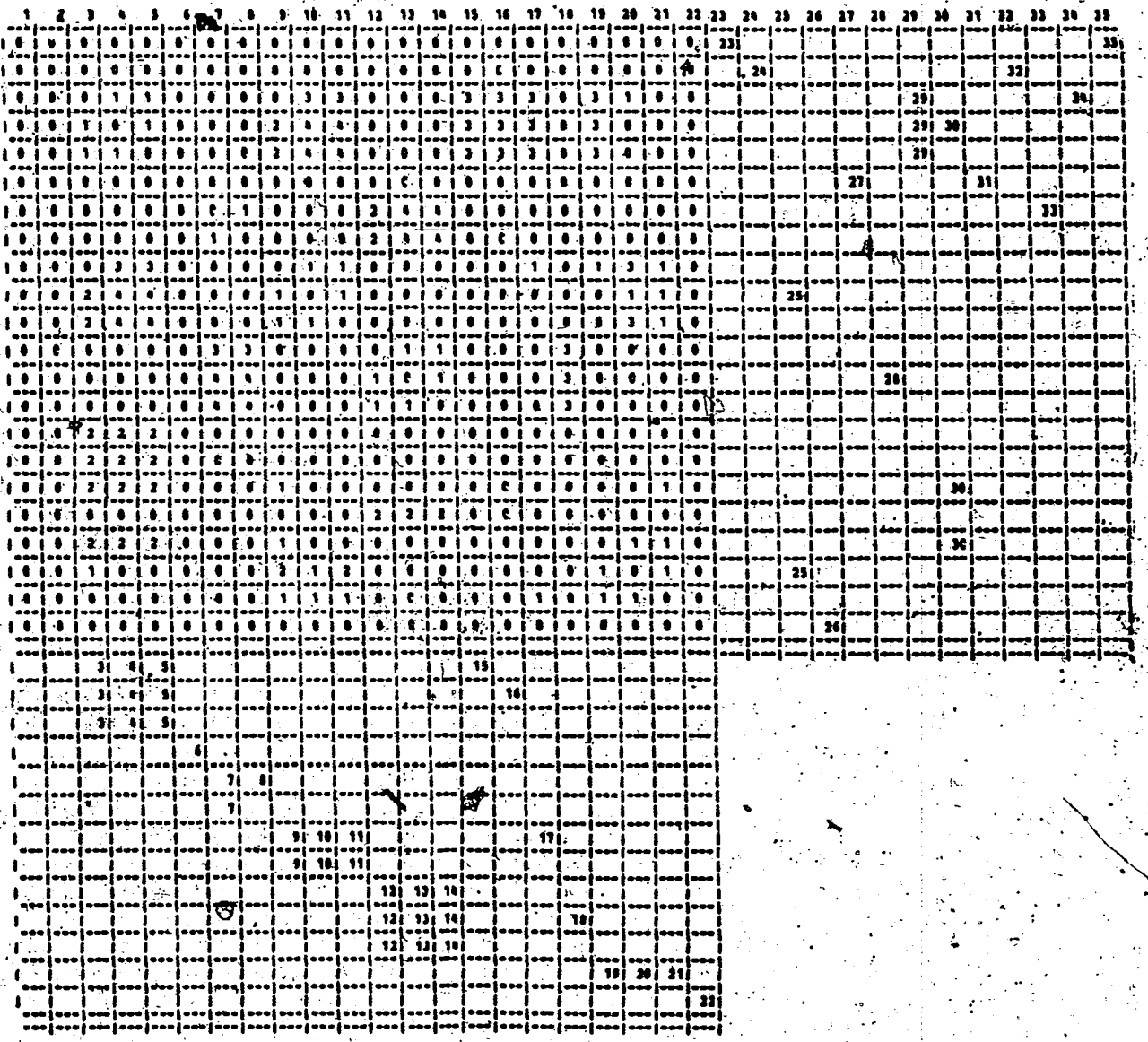
30). MO(6.1) SK ALL
(10). MO(6.1) PB MO(3.1)
(11). MO(6.1) PB F MO(3.1)
(9). MO(6.1) NF MO(N+1)

```
31). MO(7.1)    SC  1
     (12). MO(7.1)    NF    MO(N+1)
     (13). MO(7.1)    PB    MO(5.1)
     (14). MO(7.1)   PB F   MO(5.1)

32). MO(7.1)    SC  2
     (18). (C,4,0.0)    SC  2     MO(7.1)
     (12). MO(7.1)    NF    MO(N+1)
     (13). MO(7.1)    PB    MO(5.1)
     (14). MO(7.1)   PB F   MO(5.1)

33). MO(7.1)    SK  ALL
     (12). MO(7.1)    NF    MO(N+1)
     (13). MO(7.1)    PB    MO(5.1)
     (14). MO(7.1)   PB F   MO(5.1)

34). MO(8.1)    SC  1
     (19). (A,5,3.1,6.1)    SC  1    MO(8.1)
     (20). (A,7,6.1,3.1)    SC  1    MO(8.1)
     (21). (C,5,6.1)    SC  1    MO(8.1)

35). MO(8.1)    SC  2
     (22). (R,2,0.0)    SC  2    MO(8.1)
```

STRATEGY CONFLICTS AND INDEPENDENTS.

	ACTION OR ALTERNATE	OPTION	ACTION
1).	SG(1.1)	ST	SG(6.1)
2).	SG(2.1)	ST	SG(5+1)
3).	SG(3.1)	ST	SG(5+1)
4).	SG(3.1)	ST	SG(6.1)
5).	SG(3.1)	ST F	SG(6.1)
6).	SG(4.1)	ST	SG(5+1)
7).	SG(5.1)	ST	SG(7.1)
8).	SG(5.1)	ST F	SG(7.1)
9).	SG(6.1)	ST	SG(5+1)
10).	SG(6.1)	SB	SG(3.1)
11).	SG(6.1)	SB F	SG(3.1)
12).	SG(7.1)	ST	SG(5+1)
13).	SG(7.1)	SB	SG(5.1)
14).	SG(7.1)	SB F	SG(5.1)
15).	(C, 2, 0.0)	SC 1	SG(3.1)
16).	(C, 4, 0.0)	SC 2	SG(3.1)
17).	(4, 5, 3.1, 6.1)	SC 1	SG(6.1)
18).	(C, 4, 0.0)	SC 2	SG(7.1)
19).	(4, 5, 3.1, 6.1)	SC 1	SG(5.1)
20).	(4, 7, 6.1, 3.1)	SC 1	SG(5.1)
21).	(C, 5, 6.1)	SC 1	SG(5.1)
22).	(3, 2, 0.0)	SC 2	SG(5.1)
23).	SG(3.1)	SC 1	
24).	SG(3.1)	SC 2	
25).	SG(3.1)	SE ALL	
26).	SG(6.1)	SE ALL	
27).	SG(5.1)	SC 1	
28).	SG(5.1)	SE ALL	
29).	SG(6.1)	SC 1	
30)	SG(6.1)	SE ALL	
31).	SG(7.1)	SC 1	
32).	SG(7.1)	SC 2	
33).	SG(7.1)	SE ALL	
34).	SG(8.1)	SC 1	
35).	SG(8.1)	SC 2	

POINT(S)



OPTIMIZED MICROPROGRAM VERSIONS

MICRO-INST. MICRO-OP., OP. CODE, SOURCE SET, SINK SET, CONTROL SET, OPERATION UNITS SET, TIME VALIDITY SET.

Version 1
(2,11,14)
(2,11,14,16)

1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {7,5}.
4.1, ADD, {1,2}, {2}.
5.1, ADD, {2,4}, {8,9}.
8.1, SUB, {7,3}, {3}.

Version 2
(2,11,12)
(2,11,12,16)

1.1, GATE, {2} {3}.
3.1, ADD, {3,4}, {7,5}.
4.1, ADD, {1,2}, {2}.
5.1, ADD, {2,4}, {8}.
8.1, SUB, {7,3}, {3}.

Version 3
(2,10,14)
(2,3,10,14,16,17,19)

1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {5}.
4.1, ADD, {1,2}, {2}.
5.1, ADD, {2,4}, {8,9}.
8.1, SUB, {5,3}, {3}.

Version 4
(2,10,12)
(2,3,10,12,16,17,19)

1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {5}.
4.1, ADD, {1,2}, {2}.
5.1, ADD, {2,4}, {8}.
8.1, SUB, {5,3}, {3}.

Version 5
(2,9,14)
(2,9,14,16)

1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {7}.
4.1, ADD, {1,2}, {2}.
5.1, ADD, {2,4}, {8,9}.
8.1, SUB, {7,3}, {3}.

Version 6
(2,9,12)
(2,9,12,16)

1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {7}.
4.1, ADD, {1,2}, {2}.
5.1, ADD, {2,4}, {8}.
8.1, SUB, {7,3}, {3}.

Version 7
(2,8,11)
(2,8,11,16,18)

1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {7,5}.
4.1, ADD, {1,2}, {2}.
7.1, ADD, {2,4}, {9,8}.
8.1, SUB, {7,3}, {3}.

Version 8
(2,8,10)

1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {5}.

(2,3,8,10,16,17,18,19)

```
4.1, ADD, {1,2}, {2}.
7.1, ADD, {2,4}, {9,8}.
8.1, SUB, {5,3}, {3}.
```

Version 9
(2,8,9)
(2,8,9,16,18)

```
1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {7}.
4.1, ADD, {1,2}, {2}.
7.1, ADD, {2,4}, {9,8}.
8.1, SUB, {7,3}, {3}.
```

Version 10
(2,7,11)
(2,7,11,12,16,18)

```
1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {7,5}.
4.1, ADD, {1,2}, {2}.
7.1, ADD, {2,4}, {8}.
8.1, SUB, {7,3}, {3}.
```

Version 11
(2,7,10)
(2,3,7,10,12,16,17,18,19)

```
1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {5}.
4.1, ADD, {1,2}, {2}.
7.1, ADD, {2,4}, {8}.
8.1, SUB, {5,3}, {3}.
```

Version 12
(2,7,9)
(2,7,9,12,16,18)

```
1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {7}.
4.1, ADD, {1,2}, {2}.
7.1, ADD, {2,4}, {8}.
8.1, SUB, {7,3}, {3}.
```

Version 13
(2,3,14)
(2,3,10,14,16,17,19)

```
1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {5}.
4.1, ADD, {1,2}, {2}.
5.1, ADD, {2,4}, {8,9}.
8.1, SUB, {5,3}, {3}.
```

Version 14
(2,3,14)
(2,3,11,14,16,17,19)

```
1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {5}.
4.1, ADD, {1,2}, {2}.
5.1, ADD, {2,4}, {8,9}.
8.1, SUB, {5,3}, {3}.
```

Version 15
(2,3,12)
(2,3,10,12,16,17,19)

```
1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {5}.
4.1, ADD, {1,2}, {2}.
5.1, ADD, {2,4}, {8}.
8.1, SUB, {5,3}, {3}.
```

Version 16
(2,3,12)
(2,3,11,12,16,17,19)

```
1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {5}.
4.1, ADD, {1,2}, {2}.
5.1, ADD, {2,4}, {8}.
8.1, SUB, {5,3}, {3}.
```

Version 17
(2,3,8)
(2,3,8,10,16,17,18,19)

```
1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {5}.
4.1, ADD, {1,2}, {2}.
7.1, ADD, {2,4}, {9,8}.
8.1, SUB, {5,3}, {3}.
```

Version 18
(2,3,8)
(2,3,8,11,16,17,18,19)

```
1.1, GATE, {2}, {3}.
3.1, ADD, {3,4}, {5}.
4.1, ADD, {1,2}, {2}.
7.1, ADD, {2,4}, {9,8}.
8.1, SUB, {5,3}, {3}.
```

OPERATION HASH TABLE IMAGES

HASH CODE IMAGE .....

277). ADD,(1,0.0),(2,0.0)
278). ADD,(2,0.0),(6,2.1)
279). ADD,(1,0.0),(3,1.1)
280). ADD,(2,0.0),(4,0.0)
281). ADD,(3,1.1),(6,2.1)
283). ADD,(3,1.1),(4,0.0)
285). ADD,(2,4.1),(4,0.0)
290). ADD,(2,4.1),(6,2.1)
483). SUB,(7,3.1),(3,1.1)
484). SUB,(5,6.1),(3,1.1)
485). SUB,(7,3.1),(2,0.0)
486). SUB,(5,6.1),(2,0.0)
708). SHIFT,(1,0.0)
709). SHIFT,(2,0.0)
710). SHIFT,(3,0.0)
711). SHIFT,(4,0.0)
712). SHIFT,(5,0.0)
713). SHIFT,(6,0.0)
714). SHIFT,(7,0.0)
715). SHIFT,(8,0.0)
716). SHIFT,(9,0.0)
811). NOP,(1,0.0)
812). NOP,(2,0.0)
813). NOP,(3,0.0)
814). NOP,(4,0.0)
815). NOP,(5,0.0)
816). NOP,(6,0.0)
817). NOP,(7,0.0)
818). NOP,(8,0.0)
819). NOP,(9,0.0)
821). NOP,(6,2.1)
822). NOP,(3,1.1)
823). NOP,(7,3.1)
824). NOP,(2,4.1)
826). NOP,(8,5.1)
827). NOP,(9,7.1)
829). NOP,(5,6.1)
832). GATE,(2,0.0)
834). GATE,(4,0.0)
839). NOP,(3,8.1)
840). GATE,(7,3.1)

Example from "Optimization Strategies for Micro-programs," by L. L. Klier and C. V. Ramamoorthy, IEEE Transactions on Computers, Vol. C-20, No. 7, 1971, pp. 789.

ORIGINAL MICROPROGRAM

MICRO-INST. MICRO-OP., OP. CODE, SOURCE SET, SINK SET, CONTROL SET, OPERATION UNIT SET, TIME VALIDITY SET.

1.1, ADD, {1,2}, {3}.
2.1, SUB, {3,4}, {4}.
3.1, GATE, {3}, {5}.
4.1, GATE, {5}, {6}.
5.1, AND, {6,1}, {5}.
6.1, AND, {3,1}, {7}.

REGION PARAMETERS

FIRST MICRO-INS. = 1
FIRST MICRO-OP. = 1
LAST MICRO-INS. = 6
LAST MICRO-OP. = 1

(WITH NO FAN-OUT)

UNIT ASSIGNMENT TABLE

INDEX	MEMORY SUB-UNITS						
	1	2	3	4	5	6	7
MO(1.1)	1	1	2	0	0	0	0
MO(2.1)	0	0	1	3	0	0	0
MO(3.1)	0	0	1	0	2	0	0
MO(4.1)	0	0	0	0	1	2	0
MO(5.1)	1	0	0	0	2	1	0
MO(6.1)	1	0	1	0	0	0	2
EXIT	2	2	2	1	1	2	1

ALTERNATE UNIT ASSIGNMENT TABLE.

	SOURCE 1	SOURCE 2	SINK 1
MO(1.1)	(1, 0.0)	(2, 0.0)	(3, 1.1) (S, 4.1) (S, 3.1)
MO(2.1)	(3, 1.1) (A, 6, 1.1, 4.1) (A, 5, 1.1, 3.1)	(4, 0.0)	(4, 2.1)
MO(3.1)	(3, 1.1) (A, 6, 1.1, 4.1) (A, 5, 1.1, 3.1)		(5, 3.1) (S, 4.1) (P, 1.1)
MO(4.1)	(5, 3.1) (A, 6, 3.1, 4.1) (A, 6, 1.1, 4.1) (A, 5, 1.1, 3.1) (A, 3, 3.1, 1.1) (C, 3, 1.1)		(6, 4.1) (P, 1.1) (P, 3.1)
MO(5.1)	(6, 4.1) (A, 6, 3.1, 4.1) (A, 6, 1.1, 4.1) (A, 5, 1.1, 3.1) (A, 5, 4.1, 3.1) (A, 3, 3.1, 1.1) (A, 3, 4.1, 1.1) (C, 3, 1.1) (C, 5, 3.1)	(1, 0.0)	(5, 5.1)
MO(6.1)	(3, 1.1) (A, 6, 3.1, 4.1) (A, 6, 1.1, 4.1) (A, 5, 1.1, 3.1) (A, 5, 4.1, 3.1) (A, 3, 3.1, 1.1) (A, 3, 4.1, 1.1) (B, 5, 3.1) (C, 6, 4.1)	(1, 0.0)	(7, 6.1)

DELETION CANDIDATES

NEGATED-FORWARD DELETION CANDIDATES

1). MO(1.1) WITH RESPECT TO MC(N+1)

2). MO(3.1) WITH RESPECT TO MC(5.1)

3). MO(4.1) WITH RESPECT TO MC(N+1)

PARALLEL-FORWARD DELETION CANDIDATES

1). MO(1.1) WITH RESPECT TO MC(4.1)

2). MO(1.1) WITH RESPECT TO MC(3.1)

3). MO(3.1) WITH RESPECT TO MC(4.1)

PARALLEL-BACKWARD DELETION CANDIDATES

1). MO(3.1) WITH RESPECT TO MC(1.1)

2). MO(4.1) WITH RESPECT TO MC(1.1)

3). MO(4.1) WITH RESPECT TO MC(3.1)

STRATEGY OPTIONS AND STRATEGY PCINTS

OPTIONS

INDEX	ACTION	OPTION OR ALTERNATE	ACTION

PCINTS

INDEX	POINT TYPE		
1).	MO(1.1)	NF	MO(N+1)
	(40). MO(6.1)	SC	1
	(34). MO(3.1)	SC	1
	(33). MO(2.1)	SC	1
2).	MO(1.1)	PF	MO(4.1)
	(34). MO(3.1)	SC	1
	(33). MO(2.1)	SC	1
	(37). MO(4.1)	SK	ALL
3).	MO(1.1)	PF	MO(3.1)
	(34). MO(3.1)	SC	1
	(33). MO(2.1)	SC	1
	(35). MO(3.1)	SK	ALL
4).	MO(3.1)	NF	MO(5.1)
	(36). MO(4.1)	SC	1
5).	MO(3.1)	PF	MO(4.1)
	(36). MO(4.1)	SC	1
	(37). MO(4.1)	SK	ALL
6).	MO(3.1)	PB	MO(1.1)
	(32). MO(1.1)	SK	ALL
7).	MO(4.1)	NF	MO(N+1)
	(38). MO(5.1)	SC	1
8).	MO(4.1)	PB	MO(1.1)
	(32). MO(1.1)	SK	ALL
9).	MO(4.1)	PB	MO(3.1)
	(35). MO(3.1)	SK	ALL
10).	(A,6,1.1,4.1)	SC	1 MO(2.1)
	(37). MO(4.1)	SK	ALL
11).	(A,5,1.1,3.1)	SC	1 MO(2.1)

```
      (35). MO(3.1)    SK  ALL

12). (A,6,1.1,4.1)     SC  1    MO(3.1)
     (37). MO(4.1)     SK  ALL

13). (A,5,1.1,3.1)     SC  1    MO(3.1)
     (35). MO(3.1)     SK  ALL

14). (A,6,3.1,4.1)     SC  1    MO(4.1)
     (37). MO(4.1)     SK  ALL

15). (A,6,1.1,4.1)     SC  1    MO(4.1)
     (37). MO(4.1)     SK  ALL

16). (A,5,1.1,3.1)     SC  1    MO(4.1)
     (35). MO(3.1)     SK  ALL
     (35). MO(3.1)     SK  ALL

17). (A,3,3.1,1.1)     SC  1    MO(4.1)
     (32). MO(1.1)     SK  ALL

18). (C,3,1.1)    SC  1    MO(4.1)

19). (A,6,3.1,4.1)     SC  1    MO(5.1)
     (37). MO(4.1)     SK  ALL
     (37). MO(4.1)     SK  ALL

20). (A,6,1.1,4.1)     SC  1    MO(5.1)
     (37). MO(4.1)     SK  ALL
     (37). MO(4.1)     SK  ALL

21). (A,5,1.1,3.1)     SC  1    MO(5.1)
     (35). MO(3.1)     SK  ALL
     (35). MO(3.1)     SK  ALL

22). (A,5,4.1,3.1)     SC  1    MO(5.1)
     (35). MO(3.1)     SK  ALL

23). (A,3,3.1,1.1)     SC  1    MO(5.1)
     (32). MO(1.1)     SK  ALL

24). (A,3,4.1,1.1)     SC  1    MO(5.1)
     (32). MO(1.1)     SK  ALL

25). (C,3,1.1)    SC  1    MO(5.1)

26). (C,5,3.1)    SC  1    MO(5.1)

27). (A,6,3.1,4.1)     SC  1    MO(6.1)
     (37). MO(4.1)     SK  ALL
     (37). MO(4.1)     SK  ALL
```

28). (A,6,1.1,4.1) SC 1 MO(6.1)
(37). MO(4.1) SK ALL
(37). MO(4.1) SK ALL

29). (A,3,3.1,1.1) SC 1 MO(6.1)
(32). MO(1.1) SK ALL

30). (A,3,4.1,1.1) SC 1 MO(6.1)
(32). MO(1.1) SK ALL

31). (C,6,4.1) SC 1 MO(6.1)

STRATEGY POINTS AND STRATEGY OPTIONS

POINTS
INDEX POINT TYPE

OPTIONS
INDEX ACTION OPTION ACTION
OR
ALTERNATE

32). MO(1.1) SK ALL
(1). MO(1.1) NF MO(N+1)
(3). MO(1.1) PF MO(3.1)
(2). MO(1) PF MO(4.1)

33). MO(2.1) SC 1
(10). (A,6,1.1,4.1) SC 1 MO(2.1)
(11). (A,5,1.1,3.1) SC 1 MO(2.1)

34). MO(3.1) SC 1
(12). (A,6,1.1,4.1) SC 1 MO(3.1)
(13). (A,5,1.1,3.1) SC 1 MO(3.1)
(4). MO(3.1) NF MO(5.1)
(5). MO(3.1) PF MO(4.1)
(6). MO(3.1) PB MO(1.1)

35). MO(3.1) SK ALL
(6). MO(3.1) PB MO(1.1)
(4). MO(3.1) NF MO(5.1)
(5). MO(3.1) PF MO(4.1)

36). MO(4.1) SC 1
(14). (A,6,3.1,4.1) SC 1 MO(4.1)
(15). (A,6,1.1,4.1) SC 1 MO(4.1)
(16). (A,5,1.1,3.1) SC 1 MO(4.1)
(17). (A,3,3.1,1.1) SC 1 MO(4.1)
(18). (C,3,1.1) SC 1 MO(4.1)
(7). MO(4.1) NF MO(N+1)
(8). MO(4.1) PB MO(1.1)
(9). MO(4.1) PB MO(3.1)

37). MO(4.1) SK ALL
(8). MO(4.1) PB MO(1.1)
(9). MO(4.1) PB MO(3.1)
(7). MO(4.1) NF MO(N+1)

38). MO(5.1) SC 1
(19). (A,6,3.1,4.1) SC 1 MO(5.1)

```
      (20). (A,6,1.1,4.1)    SC  1     MO(5.1)
      (21). (A,5,1.1,3.1)    SC  1     MO(5.1)
      (22). (A,5,4.1,3.1)    SC  1     MO(5.1)
      (23). (A,3,3.1,1.1)    SC  1     MO(5.1)
      (24). (A,3,4.1,1.1)    SC  1     MO(5.1)
      (25). (C,3,1.1)    SC  1     MO(5.1)
      (26). (C,5,3.1)    SC  1     MO(5.1)

39). MO(5.1)    SK  ALL

40). MO(6.1)    SC  1
      (27). (A,6,3.1,4.1)    SC  1     MO(6.1)
      (28). (A,6,1.1,4.1)    SC  1     MO(6.1)
      (29). (A,3,3.1,1.1)    SC  1     MO(6.1)
      (30). (A,3,4.1,1.1)    SC  1     MO(6.1)
      (31). (C,6,4.1)    SC  1     MO(6.1)
```

STRATEGY CONFLICTS AND EXPEDIENTS.

POINT(S)

	ACTION OR ELEMENTS	OPTION	ACTION
1).	BO(1,1)	EF	BO(3+1)
2).	BO(1,1)	PF	BO(4,1)
3).	BO(1,1)	PF	BO(3,1)
4).	BO(3,1)	EF	BO(5,1)
5).	BO(3,1)	PF	BO(4,1)
6).	BO(3,1)	PB	BO(1,1)
7).	BO(4,1)	EF	BO(5+1)
8).	BO(4,1)	PB	BO(1,1)
9).	BO(4,1)	PB	BO(3,1)
10).	(3, 6, 1,1, 4,1)	BC 1	BO(2,1)
11).	(3, 5) 1,1, 3,1)	BC 1	BO(2,1)
12).	(3, 6, 1,1, 4,1)	BC 1	BO(3,1)
13).	(3, 5) 1,1, 3,1)	BC 1	BO(3,1)
14).	(3, 6, 3,1, 4,1)	BC 1	BO(4,1)
15).	(3, 6, 1,1, 4,1)	BC 1	BO(4,1)
16).	(3, 5, 1,1, 3,1)	BC 1	BO(4,1)
17).	(3, 3, 3,1, 1,1)	BC 1	BO(4,1)
18).	(C, 3, 1,1)	BC 1	BO(4,1)
19).	(3, 6, 3,1, 4,1)	BC 1	BO(5,1)
20).	(3, 6, 1,1, 4,1)	BC 1	BO(5,1)
21).	(3, 5, 1,1, 3,1)	BC 1	BO(5,1)
22).	(3, 5, 3,1, 3,1)	BC 1	BO(5,1)
23).	(3, 3, 3,1, 1,1)	BC 1	BO(5,1)
24).	(3, 3, 3,1, 1,1)	BC 1	BO(5,1)
25).	(C, 3, 1,1)	BC 1	BO(5,1)
26).	(C, 5, 3,1)	BC 1	BO(5,1)
27).	(3, 6, 3,1, 4,1)	BC 1	BO(6,1)
28).	(3, 6, 1,1, 4,1)	BC 1	BO(6,1)
29).	(3, 3, 3,1, 1,1)	BC 1	BO(6,1)
30).	(3, 3, 3,1, 1,1)	BC 1	BO(6,1)
31).	(C, 6, 4,1)	BC 1	BO(6,1)
32).	BO(1,1)	SE ALL	
33).	BO(2,1)	BC 1	
34).	BO(3,1)	BC 1	
35).	BO(3,1)	SE ALL	
36).	BO(4,1)	BC 1	
37).	BO(4,1)	SE ALL	
38).	BO(5,1)	BC 1	
39).	BO(5,1)	SE ALL	
40).	BO(6,1)	BC 1	

OPTIMIZED MICROPROGRAM VERSIONS

MICRO-INST. MICRO-OP., OP. CODE, SOURCE SET, SINK SET, CONTROL SET, OPERATION UNITS SET, TIME VALIDITY SET.

Version 1
(4,8)
(1,4,8,10,12,15,28)

1.1, ADD, {1,2}, {6}.
2.1, SUB, {6,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

Version 2
(4,8)
(1, 8,10,15,28)

1.1, ADD, {1,2}, {6}.
2.1, SUB, {6,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

Version 3
(4,8)
(1,4,8,10,12,28)

1.1, ADD, {1,2}, {6}.
2.1, SUB, {6,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

Version 4
(4,8)
(1,4,8,10,28)

1.1, ADD, {1,2}, {6}.
2.1, SUB, {6,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

Version 5
(4,7)
(4,7,18,25)

1.1, ADD, {1,2}, {3}.
2.1, SUB, {3,4}, {4}.
5.1, AND, {3,1}, {5}.
6.1, AND, {3,1}, {7}.

Version 6
(4,7)
(4,7,25)

1.1, ADD, {1,2}, {3}.
2.1, SUB, {3,4}, {4}.
5.1, AND, {3,1}, {5}.
6.1, AND, {3,1}, {7}.

Version 7
(1,4)
(1,4,8,10,12,15,28)

1.1, ADD, {1,2}, {6}.
2. UB, {6,4}, {4}.
5. , {6,1}, {5}.
{6,1}, {7}.

Version 8
(1,4)
(1,4,8,10,15,28)

, {1,2}, {6}.
2.1, SUB, {6,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

STRATEGY OPTIONS AND STRATEGY POINTS

(WITH FAN-OUT)

OPTIONS

INDEX	ACTION	OPTION OR ALTERNATE	ACTION
1).	MO(1.1)	NF	MO(N+1)
	(46). MO(6.1)		SC 1
	(40). MO(3.1)		SC 1
	(39). MO(2.1)		SC 1
2).	MO(1.1)	PF	MO(4.1)
	(40). MO(3.1)		SC 1
	(39). MO(2.1)		SC 1
	(43). MO(4.1)		SK ALL
3).	MO(1.1)	PF F	MO(4.1)
	(40). MO(3.1)		SC 1
	(39). MO(2.1)		SC 1
4).	MO(1.1)	PF	MO(3.1)
	(40). MO(3.1)		SC 1
	(39). MO(2.1)		SC 1
	(41). MO(3.1)		SK ALL
5).	MO(1.1)	PF F	MO(3.1)
	(40). MO(3.1)		SC 1
	(39). MO(2.1)		SC 1
6).	MO(3.1)	NF	MO(5.1)
	(42). MO(4.1)		SC 1
7).	MO(3.1)	PF	MO(4.1)
	(42). MO(4.1)		SC 1
	(43). MO(4.1)		SK ALL
8).	MO(3.1)	PF F	MO(4.1)
	(42). MO(4.1)		SC 1
9).	MO(3.1)	PB	MO(1.1)
	(38). MO(1.1)		SK ALL

```
10) . MO(3.1)   PB  F   MC(1.1)

11) . MO(4.1)   NF      MC(N+1)
      (44) . MO(5.1)    SC  1

12) . MO(4.1)   PB      MO(1.1)
      (38) . MO(1.1)    SK  ALL

13) . MO(4.1)   PB  F   MC(1.1)

14) . MO(4.1)   PB      MO(3.1)
      (41) . MO(3.1)    SK  ALL

15) . MO(4.1)   PB  F   MO(3.1)

16) . (A,6,1.1,4.1)     SC  1   MO(2.1)
      (43) . MO(4.1)    SK  ALL

17) . (A,5,1.1,3.1)     SC  1   MO(2.1)
      (41) . MO(3.1)    SK  ALL

18) . (A,6,1.1,4.1)     SC  1   MO(3.1)
      (43) . MO(4.1)    SK  ALL

19) . (A,5,1.1,3.1)     SC  1   MO(3.1)
      (41) . MO(3.1)    SK  ALL

20) . (A,6,3.1,4.1)     SC  1   MO(4.1)
      (43) . MO(4.1)    SK  ALL

21) . (A,6,1.1,4.1)     SC  1   MO(4.1)
      (43) . MO(4.1)    SK  ALL

22) . (A,5,1.1,3.1)     SC  1   MO(4.1)
      (41) . MO(3.1)    SK  ALL
      (41) . MO(3.1)    SK  ALL

23) . (A,3,3.1,1.1)     SC  1   MO(4.1)
      (38) . MO(1.1)    SK  ALL

24) . (C,3,1.1)   SC  1   MO(4.1)

25) . (A,6,3.1,4.1)     SC  1   MO(5.1)
      (43) . MO(4.1)    SK  ALL
      (43) . MO(4.1)    SK  ALL

26) . (A,6,1.1,4.1)     SC  1   MO(5.1)
      (43) . MO(4.1)    SK  ALL
      (43) . MO(4.1)    SK  ALL

27) . (A,5,1.1,3.1)     SC  1   MO(5.1)
```

```
      (41). MO(3.1)    SK   ALL
      (41). MO(3.1)    SK   ALL

28). (A,5,4.1,3.1)     SC   1     MO(5.1)
     (41). MO(3.1)     SK   ALL

29). (A,3,3.1,1.1)     SC   1     MO(5.1)
     (38). MO(1.1)     SK   ALL

30). (A,3,4.1,1.1)     SC   1     MO(5.1)
     (38). MO(1.1)     SK   ALL

31). (C,3,1.1)    SC   1     MO(5.1)

32)  (C,5,3.1)    SC   1     MO(5.1)

     (A,6,3.1,4.1)     SC        MO(6.1)
     (43). MO(4.1)     SK
     (43). MO(4.1)     SK   ALL

34). (A,6,1.1,4.1)     SC   1     MO(6.1)
     (43). MO(4.1)     SK   ALL
     (43). MO(4.1)     SK   ALL

35). (A,3,3.1,1.1)     SC   1     MO(6.1)
     (38). MO(1.1)     SK   ALL

36). (A,3,4.1,1.1)     SC   1     MO(6.1)
     (38). MO(1.1)     SK   ALL

37). (C,6,4.1)    SC   1     MO(6.1)
```

STRATEGY POINTS AND STRATEGY OPTIONS

POINTS
INDEX POINT TYPE

OPTIONS
INDEX ACTION OPTION ACTION
OR
ALTERNATE

38). MO(1.1) SK ALL
(1). MO(1.1) NF MO(N+1)
(4). MO(1.1) PF MO(3.1)
(5). MO(1.1) PF F MO(3.1)
(2). MO(1.1) PF MO(4.1)
(3). MO(1.1) PF F MO(4.1)

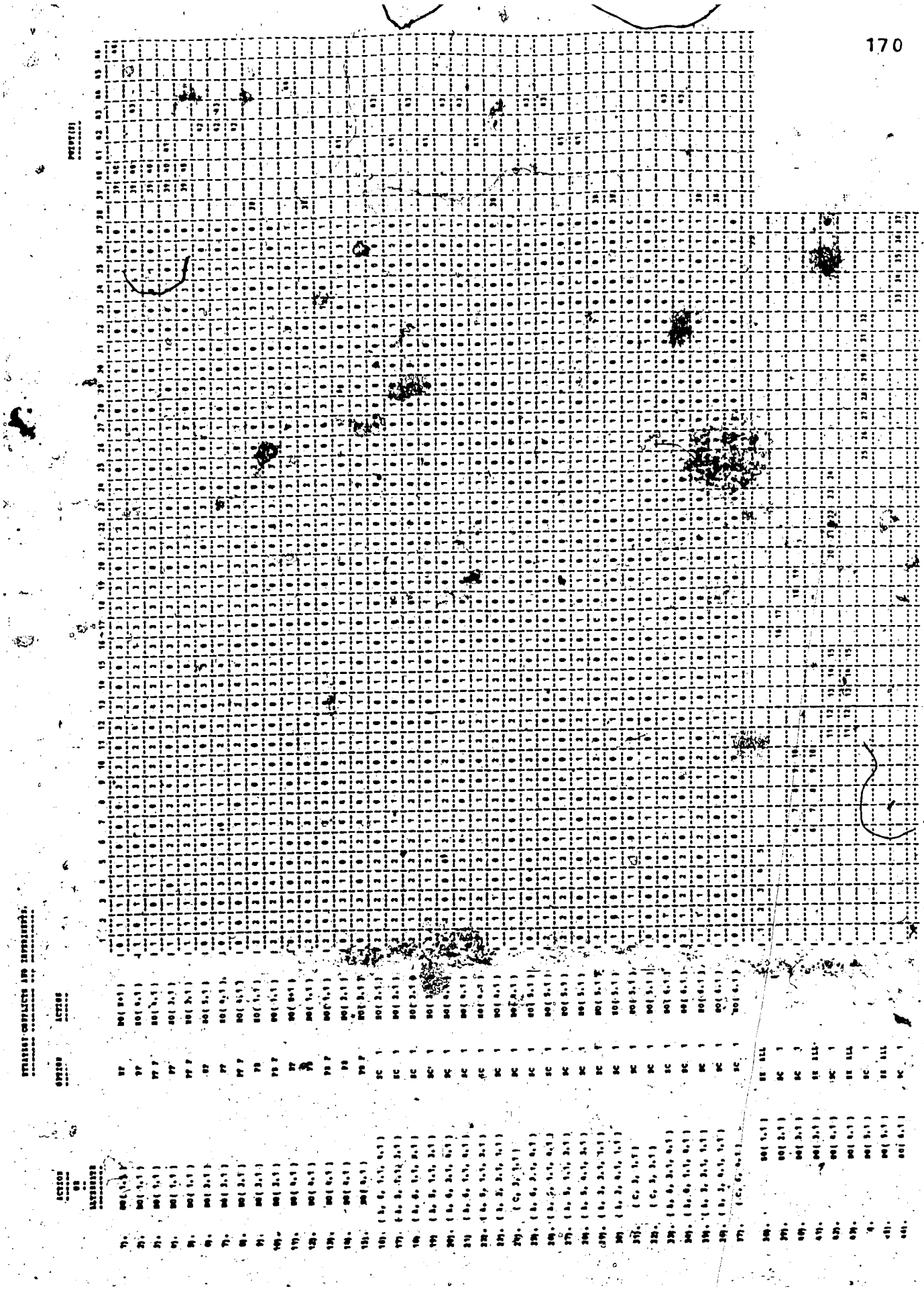
39). MO(2.1) SC 1
(16). (A,6,1.1,4.1) SC 1 MO(2.1)
(17). (A,5,1.1,3.1) SC 1 MO(2.1)

40). MO(3.1) SC 1
(18). (A,6,1.1,4.1) SC 1 MO(3.1)
(19). (A,5,1.1,3.1) SC 1 MO(3.1)
(6). MO(3.1) NF MO(5.1)
(7). MO(3.1) PF MO(4.1)
(8). MO(3.1) PF F MO(4.1)
(9). MO(3.1) PB MO(1.1)
(10). MO(3.1) PB F MO(1.1)

41). MO(3.1) SK ALL
(9). MO(3.1) PB MO(1.1)
(10). MO(3.1) PB F MO(1.1)
(6). MO(3.1) NF MO(5.1)
(7). MO(3.1) PF MO(4.1)
(8). MO(3.1) PF F MO(4.1)

42). MO(3.1) SC 1
(20). (A,6,3.1,4.1) SC 1 MO(4.1)
(21). (A,6,1.1,4.1) SC 1 MO(4.1)
(22). (A,5,1.1,3.1) SC 1 MO(4.1)
(23). (A,3,3.1,1.1) SC 1 MO(4.1)
(24). (C,3,1.1) SC 1 MO(4.1)
(11). MO(4.1) NF MO(N+1)
(12). MO(4.1) PB MO(1.1)
(13). MO(4.1) PB F MO(1.1)
(14). MO(4.1) PB MO(3.1)
(15). MO(4.1) PB F MO(3.1)

```
43). MO(4.1)    SK  ALL
     (12). MO(4.1)    PB    MO(1.1)
     (13). MO(4.1)   PB F   MO(1.1)
     (14). MO(4.1)    PB    MO(3.1)
     (15). MO(4.1)   PB F   MO(3.1)
     (11). MO(4.1)    NF    MO(N+1)

44). MO(5.1)    SC  1
     (25). (A,6,3.1,4.1)    SC  1    MO(5.1)
     (26). (A,6,1.1,4.1)    SC  1    MO(5.1)
     (27). (A,5,1.1,3.1)    SC  1    MO(5.1)
     (28). (A,5,4.1,3.1)    SC  1    MO(5.1)
     (29). (A,3,3.1,1.1)    SC  1    MO(5.1)
     (30). (A,3,2.1,1.1)    SC  1    MO(5.1)
     (31). (C,3,1.1)    SC  1    MO(5.1)
     (32). (C,5,3.1)    SC  1    MO(5.1)

45). MO(5.1)    SK  ALL

46). MO(6.1)    SC  1
     (33). (A,6,3.1,4.1)    SC  1    MO(6.1)
     (34). (A,6,1.1,4.1)    SC  1    MO(6.1)
     (35). (A,3,3.1,1.1)    SC  1    MO(6.1)
     (36). (A,3,2.1,1.1)    SC  1    MO(6.1)
     (37). (C,6,4.1)    SC  1    MO(6.1)
```



OPTIMIZED MICROPROGRAM VERSIONS

MICRO-INST. MICRO-OP., OP. CODE, SOURCE SET, SINK SET, CONTRCL SET, OPERATICN UNITS SET, TIME VALIDITY SET.

Version 1
(10,13)
(10,13)

1.1, ADD, {1,2}, {3,5,6}.
2.1, SUB, {3,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {3,1}, {7}.

Version 2
(10,12)
(1,10,12,16,18,34)

1.1, ADD, {1,2}, {5,6}.
2.1, SUB, {6,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

Version 3
(10,12)
(1,10,12,17,18,34)

1.1, ADD, {1,2}, {5,6}.
2.1, SUB, {5,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

Version 4
(10,12)
(1,10,12,16,19,34)

1.1, ADD, {1,2}, {5,6}.
2.1, SUB, {6,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

Version 5
(10,12)
(1,10,12,17,19,34)

1.1, ADD, {1,2}, {5,6}.
2.1, SUB, {5,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

Version 6
(10,11)
(10,11,27)

1.1, ADD, {1,2}, {3,5}.
2.1, SUB, {3,4}, {4}.
5.1, AND, {5,1}, {5}.
6.1, AND, {3,1}, {7}.

Version 7
(10,11)
(10,11,31)

1.1, ADD, {1,2}, {3,5}.
2.1, SUB, {3,4}, {4}.
5.1, AND, {3,1}, {5}
6.1, AND, {3,1}, {7}

Version 8
(9,13)
(1,9,13,16,18,34)

1.1, ADD, {1,2}, {5,6}.
2.1, SUB, {6,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

Version 9
(9,13)
(1,9,13,17,18,34)

1.1, ADD, {1,2}, {5,6}.
2.1, SUB, {5,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

Version 10
(9,13)
(1,9,13,16,19,34)

1.1, ADD, {1,2}, {5,6}.
2.1, SUB, {6,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

Version 11
(9,13)
(1,9,13,17,19,34)

1.1, ADD, {1,2}, {5,6}.
2.1, SUB, {5,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

Version 12
(6,13)
(6,13,21)

1.1, ADD, {1,2}, {3,6}.
2.1, SUB, {3,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {3,1}, {7}.

Version 13
(6,13)
(6,13,24)

1.1, ADD, {1,2}, {3,6}.
2.1, SUB, {3,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {3,1}, {7}.

Version 14
(6,13)
(6,13)

1.1, ADD, {1,2}, {3,6}.
2.1, SUB, {3,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {3,1}, {7}.

Version 15
(6,12)
(1,6,12,16,18,21,34)

1.1, ADD, {1,2}, {6}.
2.1, SUB, {6,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

Version 16
(6,12)
(1,6,12,16,21,34)

1.1, ADD, {1,2}, {6}.
2.1, SUB, {6,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

Version 17
(6,12)
(1,6,12,16,18,34)

1.1, ADD, {1,2}, {6}.
2.1, SUB, {6,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

Version 18
(6,12)
(1,6,12,16,34)

1.1, ADD, {1,2}, {6}.
2.1, SUB, {6,4}, {4}.
5.1, AND, {6,1}, {5}.
6.1, AND, {6,1}, {7}.

OPERATTION HASH TABLE IMAGES

HASH CODE IMAGE .....

277). ADD,(1,0.0),(2,0.0)
281). ADD,(3,1.1),(1,0.0)
285). ADD,(5,3.1),(1,0.0)
287). ADD,(6,4.1),(1,0.0)
477). SUB,(3,1.1),(4,0.0)
708). SHIFT,(1,0.0)
709). SHIFT,(2,0.0)
710). SHIFT,(3,0.0)
711). SHIFT,(4,0.0)
712). SHIFT,(5,0.0)
713). SHIFT,(6,0.0)
714). SHIFT,(7,0.0)
811). NOP,(1,0.0)
812). NOP,(2,0.0)
813). NOP,(3,0.0)
814). NOP,(4,0.0)
815). NOP,(5,0.0)
816). NOP,(6,0.0)
817). NOP,(7,0.0)
819). NOP,(4,2.1)
821). NOP,(5,3.1)
822). NOP,(3,1.1)
823). NOP,(6,4.1)
824). NOP,(7,6.1)
828). NOP,(5,5.1)
834). GATE,(3,1.1)
838). GATE,(5,3.1)
857). AND,(3,1.1),(1,0.0)
861). AND,(5,3.1),(1,0.0)
863). AND,(6,4.1),(1,0.0)