k TRANSFORMATION PACKAGE FOR THE BEHAVIORAL LEVEL OF THE CMU-DA SYSTEM

by

Robert A. Walker

Decambsr, 19)2

DRC-01-14-32

A Transformation Package for the Behavioral Level of the CMU-DA System

Robert A. Walker

11 October 1982

Department of Electrical Engineering
Carnegie-Mellon University
Pittsburgh, Pennsylvania

This research project was funded in part by Digital Equipment Corporation, and by die National Science Foundation under grant ENG 78-25755.

Table of Contents

List of Figures

List of Tables

Acknowledgements

1 would like to thank my advisor, Professor Don Thomas, for his patient encouragement and guidance of this project Were it not for him, this work would not have been possible. Thanks also go to Professors Steve Director and John Shen for serving on my committee and reviewing my work.

I also owe a great deal to Drs. Edward Snow and Michael McFarland for developing the Value Trace, and to all those others in the CMU-DA group whose work I have built upon.

I would also like to thank die members of the CMU-DA group with which I have beeen associated, especially Ted Kowalski, Jayanth Vasantharajan, and John Nestor, whose comments and suggestions were invaluable during the implementation phase of this project I also owe a great deal to Charlie Hitchcock, not only for the above, but also for his friendship and his teriffic enthusiasm for our work.

A very special vote of thanks goes to my fiancte, Ellen Lowenfeld for giving me the moral support to make this work possible, for being there when I needed her, and for truly caring about the results of this work.

Finally, I would like to thank my family, my friends in both die Electrical Engineering and Computer Sciences Departments here in Pittsburgh, and anyone else that I may have forgotten to mention.

Chapter 1
Introduction

Two dry Sticks will burn one green One.

-BenFranklin,*PoorRichard'sAlmanac*

The Carnegie-Mellon University Design Automation system (CMU-DA) [Director 81] consists of a set of computer programs whose goal is to produce a complete design in a user-specified device technology, given as input a behavioral description of die piece of hardware to be designed and a set of constraints. The programs themselves are organized in a hierarchical fashion, with individual programs or groups of programs working on the design at different levels of abstraction. The highest level, the *ISP Behavioral Level* represents die behavior of the design in a formal, structured language called ISPS [Barbacci 80]. This level is completely independent of the target technology in which die design will be implemented The ISP level is translated into die *VT Behavioral Level* which represents the behavior of the design as a set of directed acyclic graphs called the *Value Trace (VT)* [Snow 78]. This level is also independent of the implementation technology, and represents the design in a form well suited for optimization and hardware design. The feature of the CMU-DA system which distinguishes it from other design automation systems is die level of abstraction of the input

There are a number of advantages to die Value Trace format Because it uses data-flow principles to represent the behavior of die design, parallel execution of operations may be realized where the designer had specified sequential operations. By using *carriers* through which *values* flow, rattier than by using *registers* for holding *variables** the number of storage elements at the VT level is reduced. The graphical representation of the Value Trace is well suited to optimization, and transformations similar to inline expansion of subroutines (as well as formation of subroutines) may be performed to eliminate possible arbitrary divisions in the design at die ISP level Other transformations may be done to improve the performance of the final design, making die VT a powerful tool on which to base a high level design aid.

This Master's project explored the use of a set of transformations based on the VT. A software package was implemented to perform transformations at die Value Trace level, and methods were determined to measure the changes caused by these transformations. Design examples were chosen and evaluated, and general conclusions were drawn about die transformations where possible. The feasibility of automatically guiding the transformation process was also explored.

1.1. Motivations

As conceived by Snow [Snow 78], one of die major criteria of the Value Trace representation was that it be easy to manipulate by optimizing transformations, and another that die feasibility and suitabflity of these transformations be easily evaluated. These criteria imply die existence of both a useful set of transformations and a useful set of metrics to evaluate the results of applying these transformations. These transformations and metrics comprise the basis of this Master's project

1.1.1. Motivations for the Transformations

One of the major goals of die current generation of the CMU-DA system is the production of optimal designs. A problem involved with this production of designs is die elimination (or, mem realistically, die reduction) of die biases inherent in die ISP description. It is felt that the ISP behavioral description of a piece of hardware should make absolutely no assumptions about the implementation technology of die finished design. If several different people are to write different ISP behavioral descriptions of die same piece of hardware, all of these ISP descriptions might eventually be transformed into die *same* Value Trace. While die VTs resulting from die ISP-to-VT translator are already *equivalent* in behavior, each is organized according to the coding style of the designer.

One major artifact of the designer's coding style is die breaking up of die design into vtbodies at procedural boundaries as specified in die ISP behavioral description. By using vtbody[1] inline expansion to remove these boundaries, and vtbody formation to form others at win, these boundaries may be moved to other, more desirable places. Another artifact of the designer's coding style is die use of temporary variables. Code in which there are duplicated calculations in successive statements may be rewritten to perform die calculations once and store die results in a temporary variable. The behavior of the two representations is the same, but the first form requires additional computation

[1] Tlic vtbody ŝ t Vahic Trace construct similar to t subroutine or procedure

when represented as a Value Trace. Using redundant operator elimination (sec Section 2.2*2), the first may be transformed into die representation of the second.

Further motivation for the transformations lies in their use for improving the performance of the design while maintaining its original behavior. For instance, inline expansion might be performed on all vtbodics which arc only referenced once, eliminating the operators and control steps involved in calling and returning from that vtbody. In the example in the preceding paragraph, elimination of the duplicated calculations not only eliminates the operators and control steps involved with these calculation, but might eliminate a redundant piece of hardware. Perfonnance factors which might be increased by transformations include: die cost of die design, the execution speed of the design, the power consumption of die design, the amount of microcode needed in the design's microsequencer, and the complexity of the design.

1.1.2. Motivations for the Metrics

Since there are necessarily tradeoffs involved in any design process, there must be some means for evaluating the results of any specific transformation. Once a given transformation has been applied, any or all of the perfonnance factors listed above may have changed, and it would be desirable to know which ones have changed, and by how much. In evaluating the results of these transformations, it is appropriate to vary the external constraints on die die system, and see how these results change. A set of *metrics* can be developed to aid in this evaluation of the results of the transformations. These metrics are measurements that can be performed on the Value Trace and that can be interpreted in such a manner as to give reasonable information on die dianges effected by the transformations.

1.2. Previous Work

The original idea for die Value Trace was first proposed by Snow in his Ph.D. diesis [Snow 78]. This Value Trace was a Directed Acyclic Graph (DAG) similar in nature to those used in optimizing compilers. The nodes of the graph, called *activities* or *operators,* represented an operation to be performed. The arcs of the graph, called *carriers,* represented die *flow* of data from one operator to another. The primary difference between die DAGs created by Snow and those used in optimizing compilers was the addition of control constructs to allow conditionals and subroutines, something not present when DAGs are used by optimizing compilers. Major criteria for the new representation were that it should be easy to manipulate, and it should lend itself well to optimization. Once this representation was established, Snow identified a set of transformations that could be performed on the VT to optimize its representation while leaving its behavior unchanged. These transformations were the basis of the transformations implemented as part of this Master's project.

An ISP-to-VT translator and structures for the Value Trace were implemented by McFarland as a Master's project [McFarland 78]. The VT originally envisioned by Snow had abstracted away all control information except the required ordering of operator execution resulting from data dependencies. McFarland reevaluated this Value Trace and added structures for maintaining control information. These additions made it possible to assign control steps to the operators of the Value Trace, and changed the role of the SELECT operator and the transformations which operated on it. While the SELECT operator had previously served only as a data multiplexor, McFarland's changes expanded the SELECT's role to include control branching. These changes in the role of the SELECT forced the reevaluation of the SELECT transformations developed by Snow.

With the advent of McFarland's ISP-to-VT translator, the third generation of the CMU-DA system came into existence in mid-1981. The overall framework for the upper levels of the system was developed as a joint effort, and is described in [Gatenby 81] and [Lertola 81]. This framework included a graphics system, basic measurements package, automatic control step allocator, and interactive data allocator, and it was into this framework that the programs written as part of this Master's project were placed.

While not yet finished, the third generation of the CMU-DA system contains many elements which did not exist in the previous generations. Rather than existing as a collection of separate programs, the present "core" of the behavioral level portion of the system exists as a single program with separate "modes" for each function. Hitchcock [Hitchcock 82] is currently developing an automatic data-path allocator based on an algorithm developed by McFarland, Tseng [Tseng 82] is developing an automatic allocator based on a bus-style approach, and Kowalski [Kowalski 81] is developing a knowledge-based allocator using an expert-system approach. Pieces still lacking and needed to complete the system are a control allocator and a module binder. The system will soon be capable of producing a design with control-step allocation and data-path allocation, but is not yet capable of producing a finished design.

1.3. Conclusion

In the course of this Master's project, a number of areas were explored A software package was built to implement many of the transformations developed by Snow, and these transfonnations were changed where necessary due to die addition of control information by McFarland To measure the changes brought about by these transformations, metrics were chosen and implemented Design examples were selected, optimized, and evaluated using these metrics. The design space was characterized for some of these examples, and new information was derived Finally, (he feasibility of using information derived in this project to guide the transformation process was explored

The remainder of this report is organized as follows:

- Chapter 2 provides a brief overview of the structures of the Value Trace and the transformations implemented in this project Where McFarland's changes in the Value Trace have necessitated changes in the transformations, these changes are described

- Chapter 3 discusses the choosing of tools to evaluate die results of die transfonnations. This chapter also discusses the role of external constraints on the system, and how these constraints can be varied to determine their effect on the results of die transformations. Previous work by Lei ve on estimating the performance of the final design is explained, as well as how this work can be adapted to the present CMU-DA system. Methods of varying the control style of the design are also discussed

- Chapter 4 provides a number of examples of the major classes of transformations. Each of these designs are evaluated using die tools developed in Chapter 3, and general conclusions are drawn where possible.

- Chapter 5 provides further analysis of some of the results from Chapter 4. Design space plots are drawn, and further characterization of the effects of some of the transfonnations is determined The possibility of using the information developed in this and the previous chapter is explored, and a description is given of the issues involved

- Chapter 6 provides a synopsis of the work done in this Master's project, and some of the information derived from it A proposal for a new organization of die upper level of the CMU-DA system is also given.

Chapter 2
The Transformations

Where we cannot invent, we may at least improve; we may give somewhat of a novelty to that which was old, condensation to that which was diffuse, and currency to that which was recondite.

-Cotton

As described in Snow's thesis [Snow 78], there are three major groups of transformations: operator transformations, SELECT transformations, and vtbody transformations. The operator transformations are transformations on individual operators or groups of operators. The SELECT transformations are transformations to act on the SELECT construct, individually or in groups. The vtbody transformations are transformations for woricing with whole vtbodies or groups of vtbodies at a time. Since these transformations are described in detail in Snow's thesis, they are only briefly described here. Where die additions to the structure of die Value Trace by McFarland have changed the transformations, die additions and changes are discussed.

2.1. Structures of the Value Trace

Before discussing die results of the transformations, it is necessary to understand the structures on which these transformations will be performed At the highest level, die Value Trace is divided into individual groups of operations called *vtbodies*. This division of vtbodies is determined from the ISP behavioral level, and occurs at procedural boundaries and at labels. Each of these vtbodies is further subdivided into individual operations called *operators* or *activities*. The operators are of three types: ISP arithmetic and logical operators, ISP control operators, and VT-specific operators. The ISP arithmetic and logical operators, as the name implies, are the operators defined in the ISPS language which perform such activities as addition and testing for equivalence. The ISP control operators are those operators which invoke other vtbodies, restart the current vtbody, and control the conditional choice of data multiplexing or control branching. VT-specific operators are operators specific to the Value Trace and its needs as a representation for hardware synthesis. Examples of VT-specific operators are those required for reading a specific field of a carrier, or performing sign extension when reading from a smaller carrier into a larger one.

2.2. Transformations of Operators

2.2.1. Constant Folding

Constant folding is a classical example of an optimizing transformation, and one of the simplest Shown in Figure 2-1, constant folding consists of replacing an operator which acts on one or more constants with a new constant It may be applied to any activity, and is always beneficial, possibly resulting in the elimination of a hardware operator. Other benefits include a possible reduction in total number of control steps and a reduction in the complexity of the Value Trace. While it is doubtful that constant folding may be applied'to a newly generated Value Trace, it might be useful after other transformations have been applied. An example of repeated applications of constant folding in this context is show in Section 4J.

Constant folding may also be applied to a SELECT (see Section 13 for a description of the SELECT operator), in which case the appropriate branch of die SELECT is retained, and the others eliminated. Again, this will probably not occur in a new VT, but may come up in other circumstances. Section 43 shows an example of constant folding applied to a SELECT.

2.2.2. Redundant Operator Elimination

Redundant operator elimination is another classical transformation, similar to common subexpression elimination in optimizing compilers. It may be used when two or more operators of die same type have the same inputs, and consists of eliminating one of the operators and replacing all references to its outputs with references to the outputs of the operator to be retained. Figure 2-2 shows an example of this transformation. like constant folding, redundant operator elimination is always beneficial. Although die amount of hardware required to implement die final design may not change if a good allocator is used, die total number of control steps may still be reduced. In all cases, die complexity of die Value Trace will be reduced. Examples of redundant operator elimination may be seen in Section 4.4.

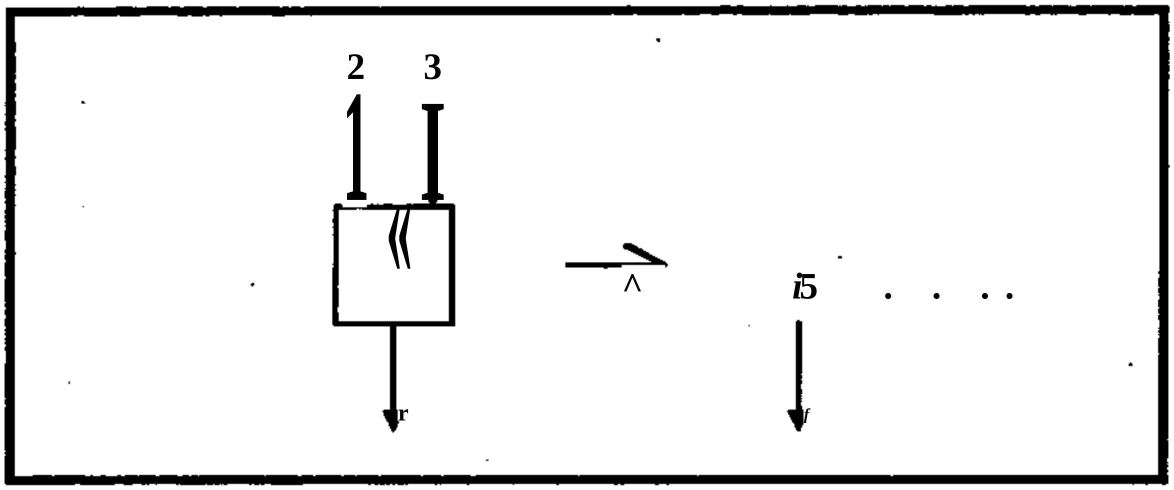

Figure 2-1: Constant Folding

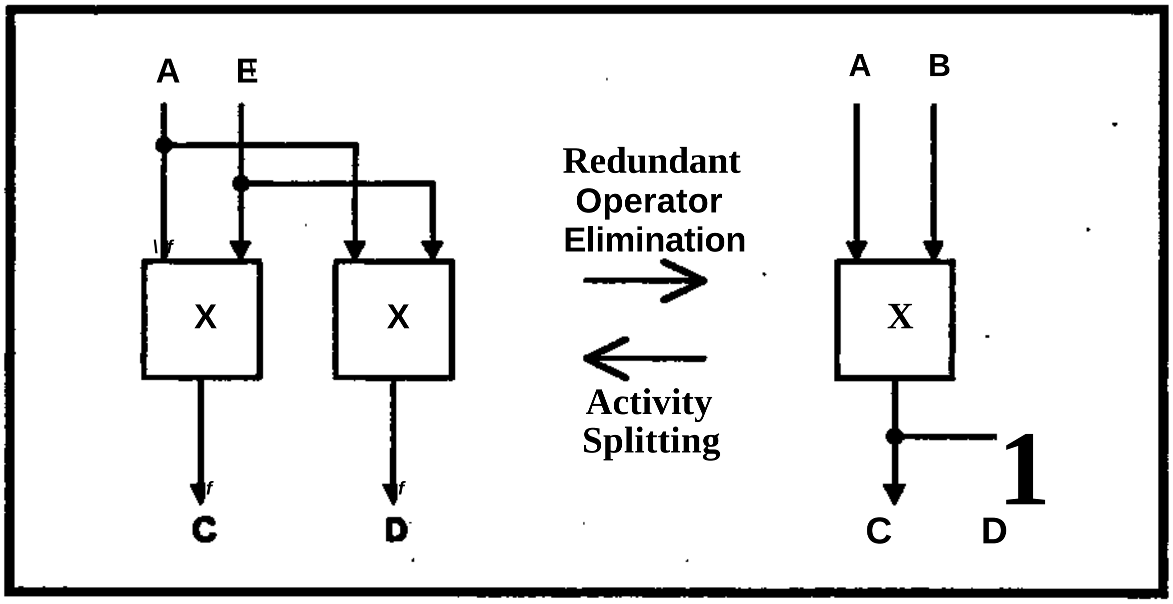

Figure 2-2: Redundant Operator Elimination and Activity Splitting

2.2.3. Activity Splitting

Activity splitting is the inverse of redundant operator elimination, and may be used to make other transformations possible. In applying it, the operator is duplicated and its outputs split between the two resulting operators. An example of activity splitting is shown in Figure 2-1 By itself, activity splitting is never beneficial. The total number of control steps may be increased, the complexity of the Value Trace is always increased, and the amount of hardware might be increased if the data path allocator is not very efficient

2.2.4. Identity Insertion

Identity insertion is another transformation which, used by itself, will have a negative effect Identity insertion consists of inserting a null activity into a data path, often an arithmetic or logical operator whose other input is the identity element for that operator. The total number of control steps may be increased, the complexity of the Value Trace will always be increased, and the amount of hardware might be increased. The only positive effect of this transformation is to make other transformations possible.

2.2.5. Bit Read Insertion

*Bit read insertion** conceived by Lertola [Lertola 81], consists of die insertion of a bit read operator[2] into a data path. Because die bit read is not an allocatable operator, the amount of hardware is not increased, nor is the total number of control steps, but the complexity of the Value Trace is increased. The major use of this transformation is to aid in allocation, where a bit read may be needed for constant storage, multiple storage of the same value, value storage, or elimination of implicit storage. Lertola's thesis shows several pages of examples of these uses.

[2]AWt read operator is a VT construct for reading a given subfield of one carrier into another orrier. Its inputs consist o f t fkld to be read from, and a constant o n ^ from the right of this field to specify the appropriate subfield. Bit reads often occur after arithmetic operators to eliminate the cany bk ami n ^ i h e results the openukm into a lumacd carrier.

2.3. Transformations of SELECTS

Since its original conception by Snow, the role of the SELECT operator was changed by die addition of control information to the Value Trace by McFarland. As envisioned by Snow, die SELECT was simply a data multiplexor. Figure 2-5 shows an example of this use, with a two bit selector, S, choosing the value in carrier A if the value of S is 0, the value in carrier B if the value of S is 1, and so on. Along with the addition of control information to the Value Trace by McFarland, die role of the SELECT was expanded to include the capability of choosing one of a number of different execution paths (i.e., it performs a "branch" operation). Figure 2-3 shows an example of this use, with the SELECT operator choosing of one of three branches based on the value of the selector. When the SELECT was thought of as a data multiplexor (referred to as a *data SELECT),* Snow's transformations were still valid in all cases. When die SELECT was used for branching of control (referred to as a *control SELECT),* many of Snow's SELECT transformations had to be reconsidered. This reevaluation was due to die control SELECTS having both a beginning and end to each branch, with only one branch being executed, while data SELECTS may be though of as acting only on die ends of branches, with all branches being executed. The SELECT motion transformations were especially affected by this change.

2.3.1. SELECT Motion

SELECT motion consists of replacing similar activities within the branches of a SELECT with a single activity outside the SELECT, and vice-versa. When moving activities out of a SELECT, the activities must be of the same type and must occur in all brandies. SELECT motion may be applied to either control SELECTS or data SELECTS, and an example of downward SELECT motion with a control select is shown in Figures 2-3 and 2-4. Snow had just considered motion at die bottom of die brandies, but die brandies of control SELECTS now have both a beginning and an end, so SELECT motion for control SELECTS has to be extended to aDow for motion at both the top and bottom of the branch. In Figure 2-3, if die three bit reads have die same constant input, they could be moved above the SELECT. Of all die transformations, the results of SELECT motion are the hardest to evaluate (see Section 4.1). With a "smart" data path allocator, the amount of hardware required remains approximately die same before and after SELECT motion, as the allocator effectively perfonns the transformation before the SELECT motion. The number of control steps varies, and is affected by the contents of the branches, control step allocation, and the type of microsequencer used. The only improvement which always occur is a reduction in die complexity of the Value Trace due to die elimination of repeated operators.

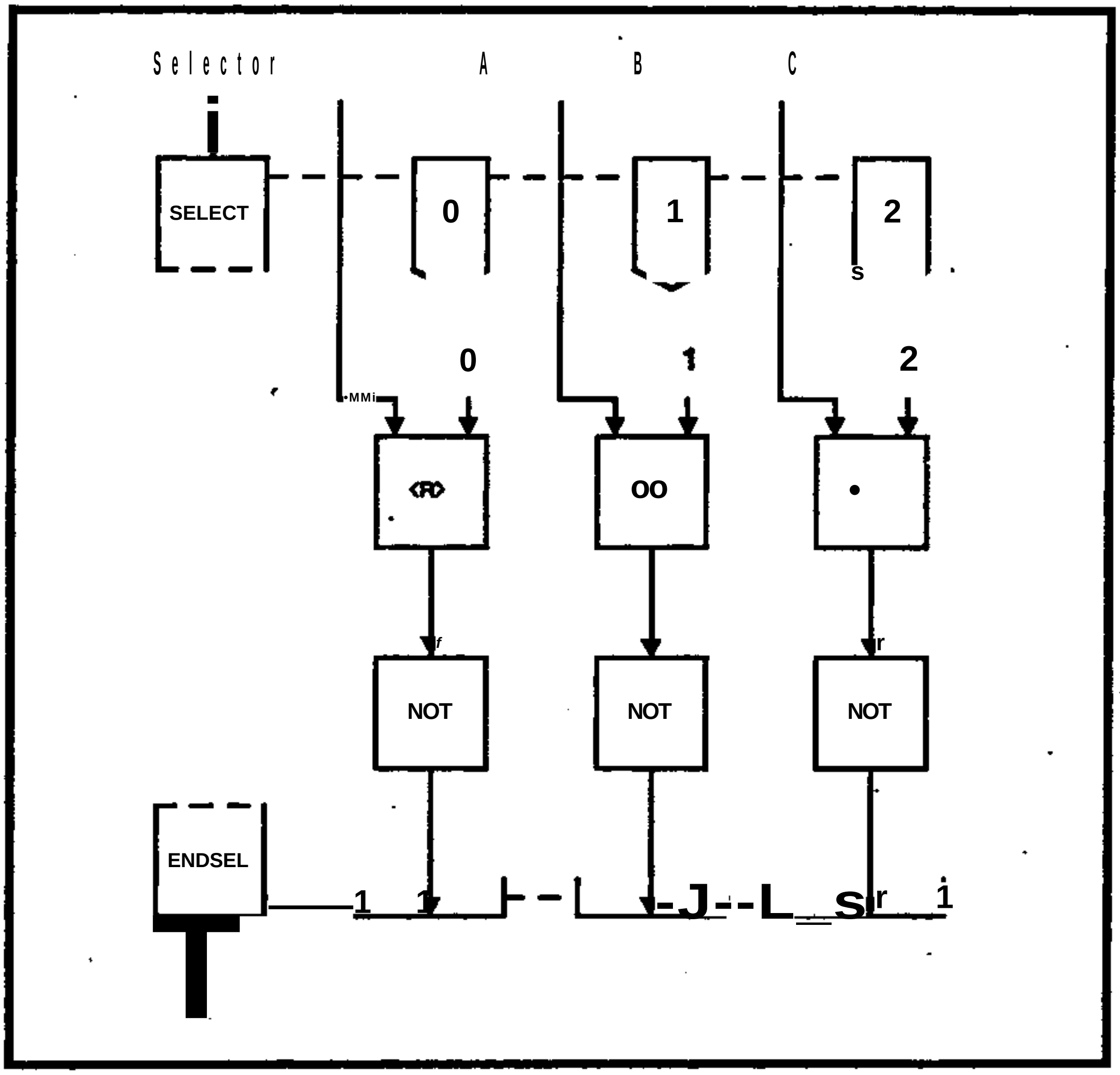

Figure 2-3: Control SELECT Before Downward SELECT Motion

2.3.2. SELECT Factoring/Combination, Selector Reduction

SELECT factoring consists of removing brandies from a given SELECT and using these branches to form a new SELECT which is cascaded with the old one. *SELECT combination* is the inverse of SELECT factoring. Figures 2-5 and 2-6 show an example of SELECT factoring applied twice to a data SELECT. When factoring a SELECT, the same selector is maintained for both levels of SELECTS, resulting in multiple activation values for each branch. Using bit reads to reduce the selector as necessary, the example can be reduced to Figure 2-7. Had the first and third Jbranch of the

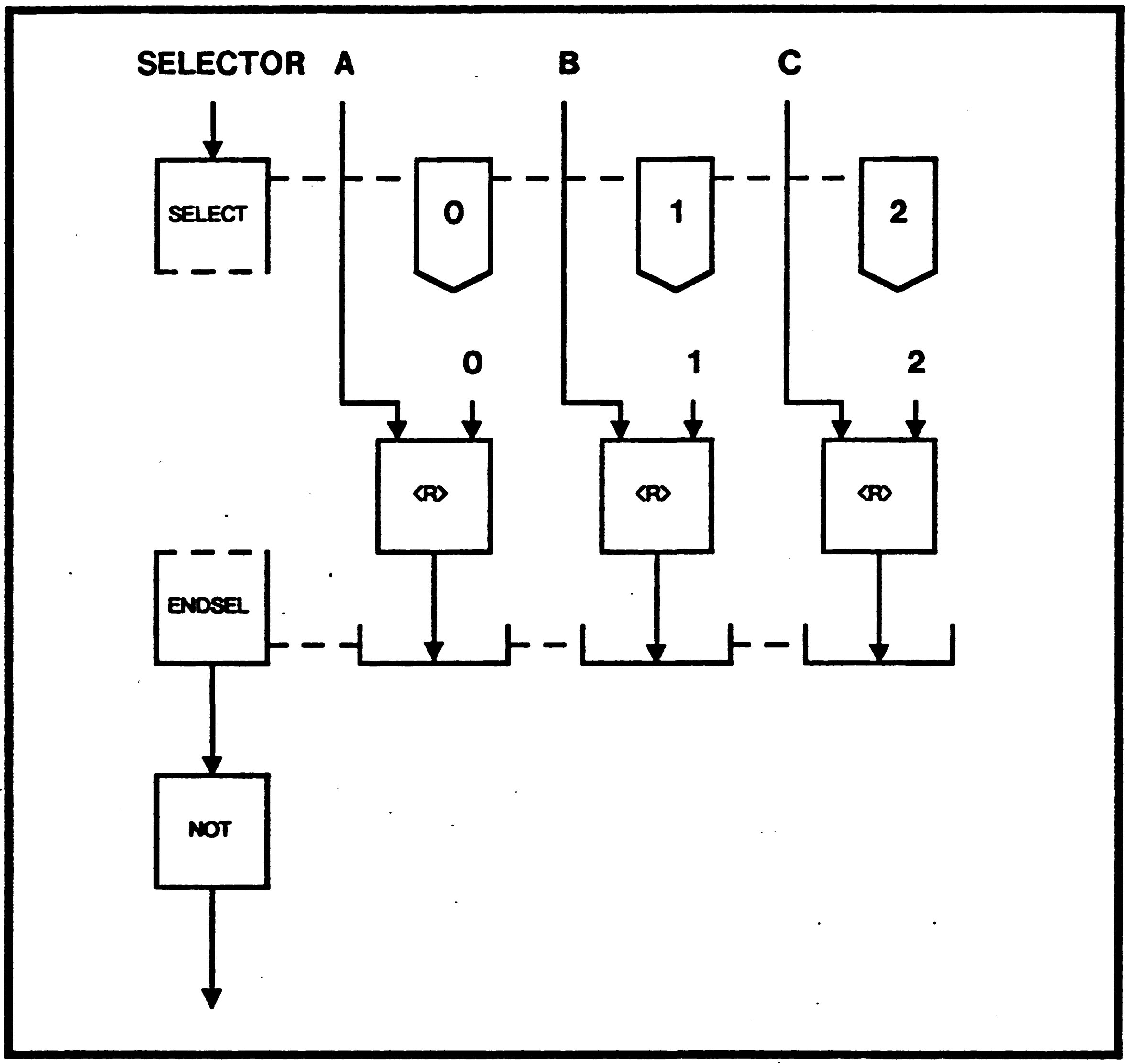

Figure 2-4: Control SELECT After Downward SELECT Motion

original SELECT been grouped together, and the second and fourth as well, the fields of the selector inputs would have been reversed in the final figure. For this reason, SELECT combination may be used in conjunction with SELECT factoring to rearrange trees of SELECTs. Section 4.1 gives another example of SELECT factoring.

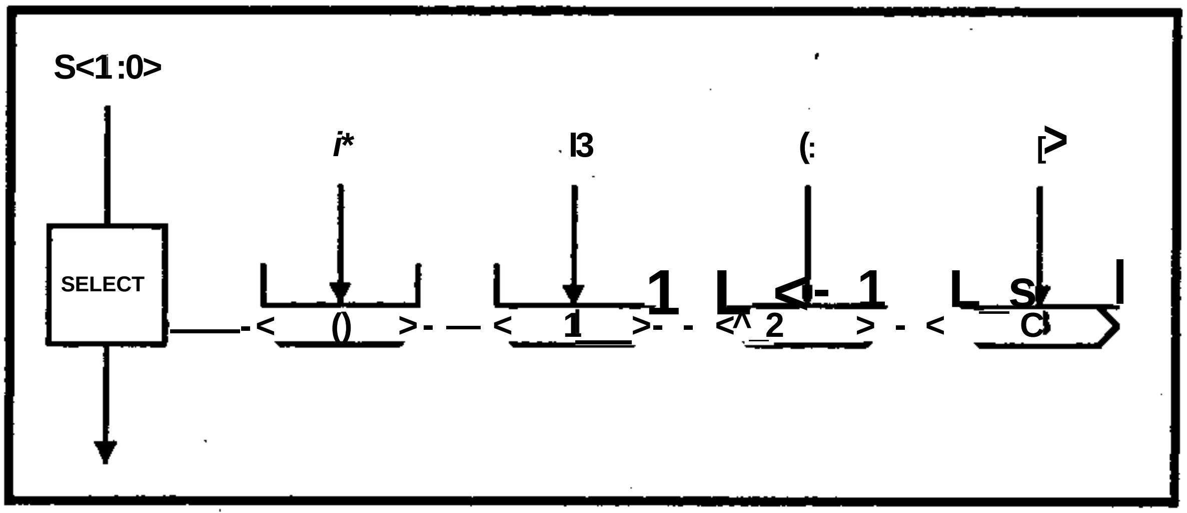

Figure 2-5:" Data SELECT Before SELECT Factoring

2.4. Transformations of Vtbodies

2.4.1. Vtbody Inline Expansion, Vtbody Formation

Vibody inline expansion is analogous to the inline expansion of a subroutine, and consists of replacing a call to a vtbody with a copy of that vtbody. Almost any vtbody may be expanded inline, except those which are RESTARTed[3]; these may not be expanded because the RESTART construct only restarts vtbodies at the beginning, and there is no provision in the CMU-DA environment to restart vtbodies at any other place. When vtbody inline expansion is applied to a vtbody which is called only once, that vtbody is subsequently a candidate for dead vtbody elimination (see Section 142), and the speed and complexity of the Value Trace are improved by the elimination of the operators which call and return from the eliminated vtbody. An example of inline expansion as applied to the Intel 8080 may be found in Section *42. Vtbodyformation,* the inverse of vtbody inline expansion, consists of removing a section of code from one vtbody, forming a new vtbody consisting of that code, and adding a call to this new vtbody in the original one.

[3]The RESTART operator is a VT flow control operator which restam the airrent vtbody. or another vtbody.

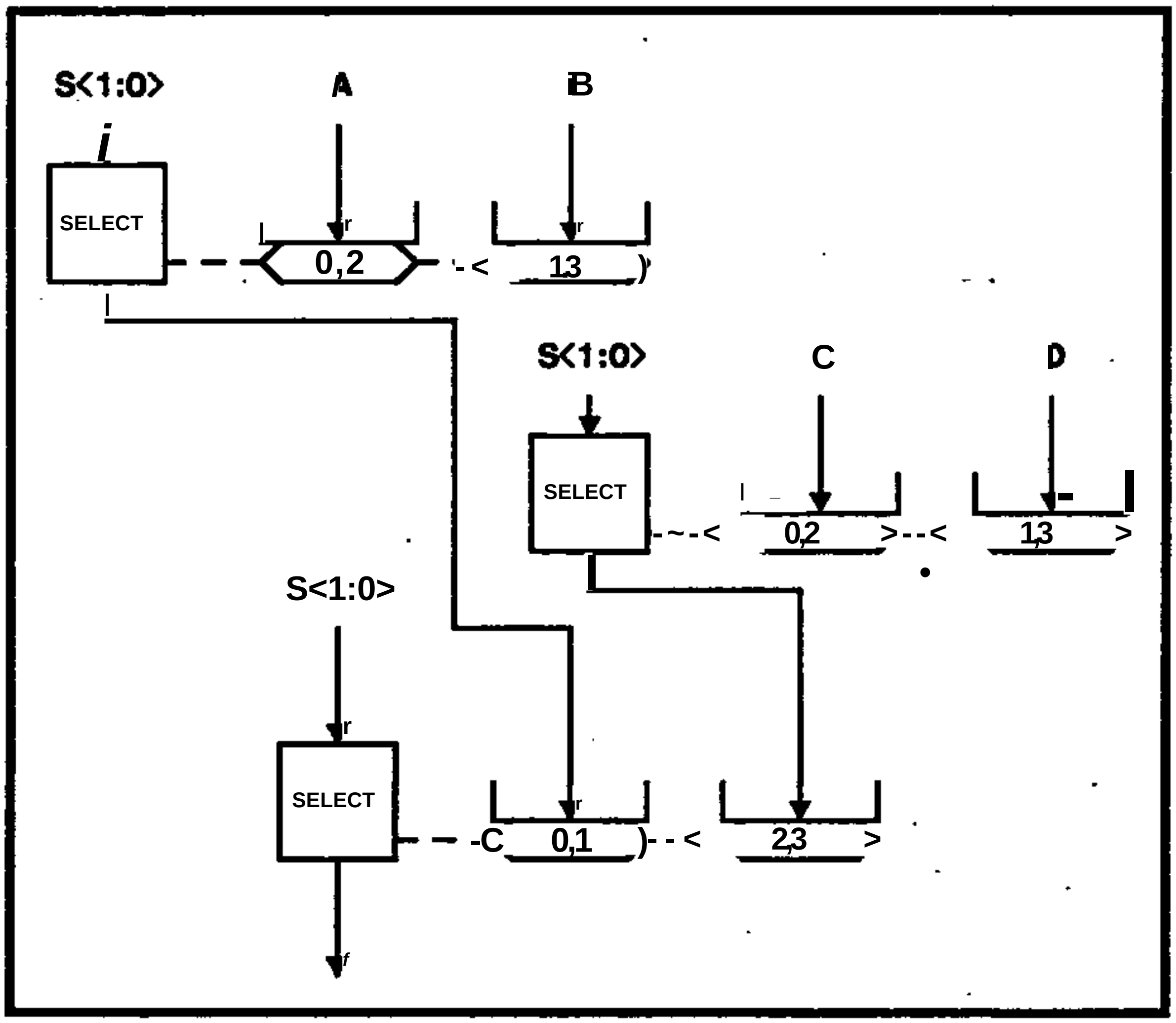

Figure 2-6: Data SELECT After SELECT Factoring

2.4.2. Dead Vtbody Elimination

Dead vtbody elimination is analogous to dead code elimination in optimizing compilers, and consists of removing from the Value Trace those vtbodies which are no longer referenced. These dead vtbodies will come about primarily as a result of inline expansion, and their elimination is always performed when applicable. Section *A2* shows the use of dead vtbody elimination after vtbody inline expansion, and Section 43 shows the use of dead vtbody elimination after loop unwinding.

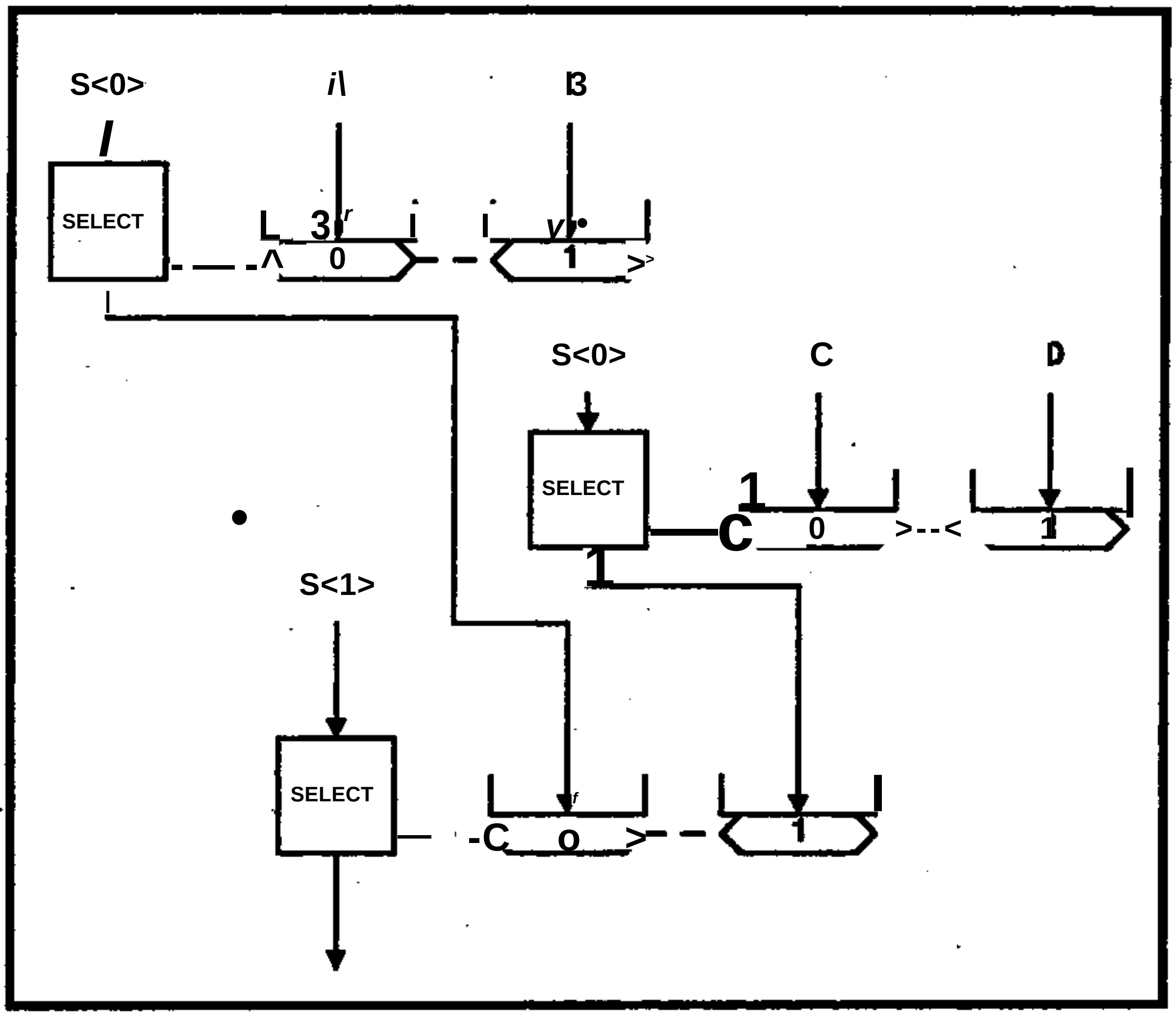

Figure 2-7: Data SELECT After Selector Reduction

2.4.3. Loop Unwinding

Loop unwinding consists of inline expanding instantiations of a looping vtbody so that constant folding and other transformations might be applied. In this manner, a loop counter may be eliminated and replaced with a finite number of calls to a given subroutine. By eliminating or reducing these loop counting operators, the amount of hardware needed is decreased, as are the number of control steps and the execution time. Section 43 gives an example of loop unwinding.

2.5. Conclusion

As part of this Master's project, the transformations described in this chapter were implemented. These transformations comprise most of those defined in Snow's thesis; others were not implemented due to time constraints, but will be added later. The transformations that were not implemented were those for moving operators into and out of vtbodies (vtbody motion), and the composite operator of pipelining. This transformation package was written in the C programming language [Kernighan 78], and comprised the major portion of the code written for this project

Existing as a separate function in the framework described in Section *IX* the transformation commands are manual in nature, rather than automatic This is in keeping with the current view of the "core" of the behavioral level portion of the system existing as a set of tools which are intended for use by either a human researcher, or, more likely, an automated *design assistant.* To avoid improper use, all commands check for applicability before beginning the actual transformation. The commands are manual in nature, and are used in conjunction with a set of metrics and initial assumptions to explore the design space in such a way that data may be obtained which will one day allow automation of the system. It is these metrics and initial assumptions which are discussed in the next chapter.

Chapter 3
Tools for Evaluation

I do love to note and to observe.

- Ben Johnson, *Volpone*

I am greedy of getting information.

- Callimachus, *Iambi*

Many of the transformations catalogued in Chapter 2 involve a tradeoff in the performance of die final design. The application of these transformations may improve some performance factors while hindering others, making it necessary to have a set of metrics for measuring the effects of these transformations on the final design. With these metrics, not only can Ac effects of a particular transformation be measured, but information might be found which can predict these effects and aid the designer in choosing which transformations to apply. In order to further characterize the results of the transformations, it may be useful to explore their results while varying die external constraints on the system. It is the development of these metrics and the varying of these constraints that this chapter is concerned with.

3.1. Choosing Tools for Evaluation

In evaluating the results of die transformations described in the previous chapter, many levels of the CMU-DA system must be considered. Shown in Figure 3-1, the upper levels of this system consist of a number of program modules, each of which is constrained by external factors. These factors include:

- Design style (control and data) - This information reflects a global decision affecting the design as a whole. Data styles identified in [Thomas 77] include 8-bit microprocessor style, bit-slice microprocessor style, distributed style, pipeline style, bus style, and parallel style. Control styles include the type of microcontroller used and the method of organizing the microwords. This information is *top-down* information; it involves die selection of an overall approach for the design process, which in turn affects die lower levels.

- Module data base information - This information reflects the chosen target technology. It is a summary of all the logical and physical parameters (function, cost, power consumption, etc.) of the given devices in that technology, and is essentially *bottom-up* information. The operator delay information from this data base is further summarized and presented to die upper levels of the system.

- The current Value Trace - This represents the current "state" of the design in progress. It must be considered since not all vtbodies are candidates for all transformations.

- Designer specified constraints - This information reflects the physical constraints placed on the design by the designer. For example, he may choose to maximize speed at the cost of chip area, or minimize power consumption at die cost of all eke.

In order to characterize die transformations, it is appropriate to vary as many of these constraints as possible in order to determine their effect on die results. While it is not be possible to vary all of these constraints, it is still possible to vafy a number of them.

In considering the different data design styles, the major variable involved is the data path allocator. In die current generation of die CMU-DA system, there are two allocators under construction: a distributed data path allocator being developed by Hitchcock [Hitchcock 82], and a knowledge-based allocator being developed by Kowalski [Kowalski 81] which currently designs in a stylized bus-oriented mode, but is capable of being expanded to allow designing with other styles. Neither of these allocators is yet available, so it is not possible to vary the design style at this time. In the course of this project, the preliminary information from [Hitchcock 82] was used when allocator-specific information was required.

It is possible to vary the control design style somewhat The two major steps in die implementation of die control portion of die design are *control step allocation* and *control allocation.* The control step allocator has the responsibility of partitioning die design into sets of operations (called *control steps)* which arc invoked together and require a fixed amount of time to execute. The control allocator is responsible for allocating hardware to control the data paths of the design. While there is no control allocator in the current generation of die CMU-DA system, a control step allocator does exist Altering the control style affects the control step allocation, which in turn affects the

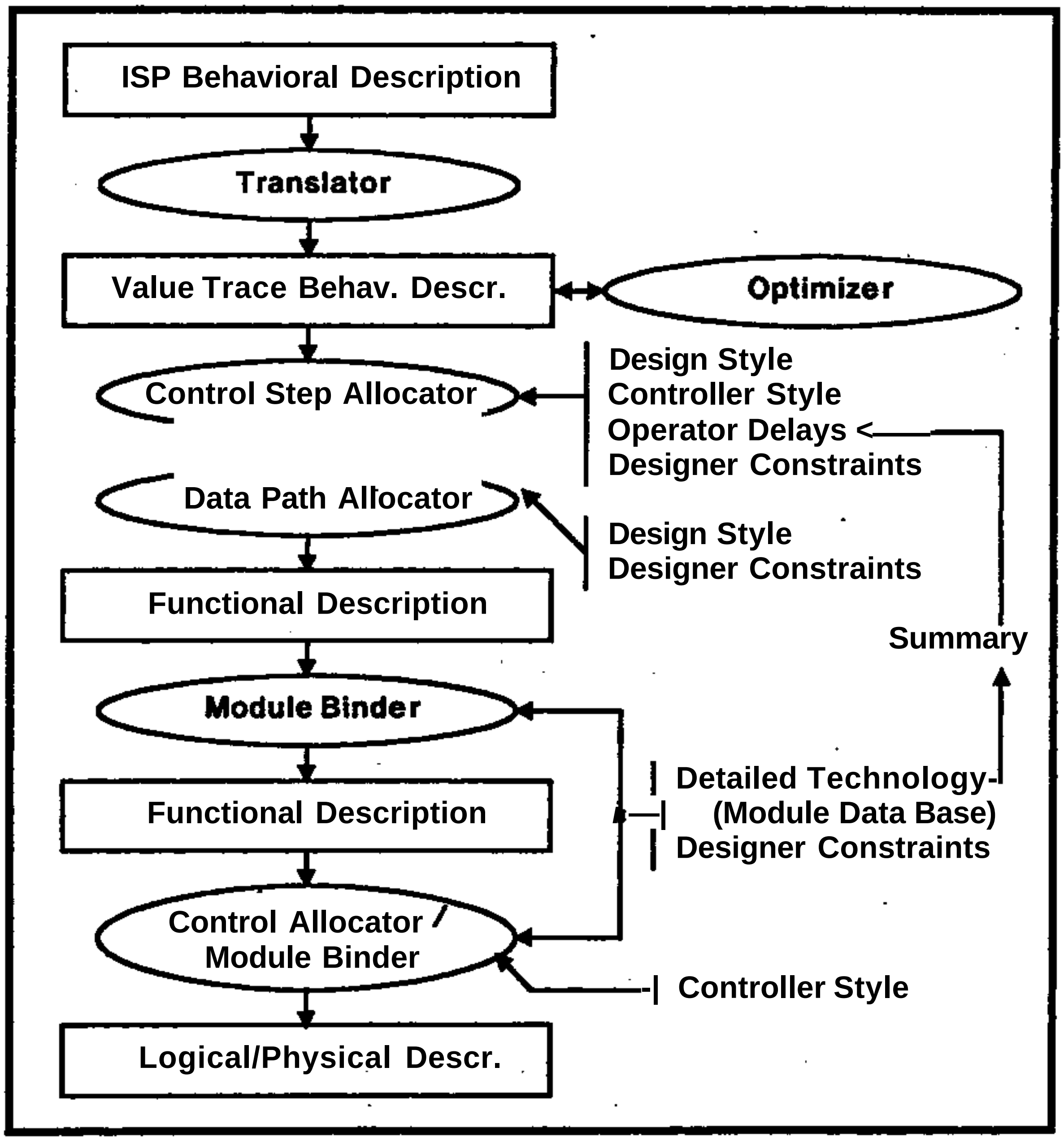

Figure 3-1: The Upper Levels of the CM U-D A System with External Constraints

results of the transformations. This altering of the control style is accomplished by varying the type of microcontroller used with die design, and is discussed later in this chapter.

There is no module binder or control allocator in the current generation of the CMU-DA system, so the module database information can not be varied at those levels. However, the operator delays for a particular implementation technology are summarized and passed to the control step allocator, and are varied at that level. The operator delay of the flow control instructions (CALLs, SELECTs, etc.) is modified at this level while varying the control style of the design. Changing the target technology does not change the effect of the transformations, and is not done as part of this project.

In considering the current VT, different Value Traces must be examined, because not all Value Traces are candidates for all transformations. A number of example Value Traces are shown in Chapter 4.

The designer specified constraints can be varied at some levels but not at others. These constraints can not be varied at the module binder level or at the control allocator levels because these levels do not exist in the current system. The constraints can be varied at the upper levels of the system by varying the data and control style of the design.

3.2. Metrics for Measuring the Characteristics of the Final Design

In evaluating the final design, the most important considerations are the style of the design, and methods for evaluating the results of the transformations for the different styles. One way to evaluate these results would be to examine both the data part and the control part of the design, and find ways of measuring the changes in each part. The rest of this chapter is concerned with the measurement of these changes in the design.

3.2.1. The Data Part

Three important performance factors of the final design are its cost, its power consumption, and its execution time. Since the current generation of the CMU-DA system has no means for producing a final design, these factors must be predicted from either the ISP behavioral level or the VT behavioral level. Drawing on information developed in the second generation of the CMU-DA system, an estimation of this data may be produced.

3.2.1.1. Leive's Work on Predictors

As part of the second generation of the CMU-DA system, a module binder was developed [Leive 81] which took as input the functional level description produced by the current data path allocator [Hafer 81], and produced as output a description of the design at the logical/physical level. This module binder was capable of producing final designs in one of two target technologies: SN74xx TTL modules, and Sandia CMOS Standard Cell modules. Choosing three relatively small designs, Lcive performed 64 separate runs of the module binder, varying the constraints and transformations for each run. From the output of the module binder, he plotted three graphs of cost, power consumption, and delay for the different runs of each design, and developed a set of equations (called *predictors)* which characterized the design space of these three performance factors. The input parameters to these predictors were obtained from the bound data path graph, and consisted of the following:

- Number of bindable nodes
- Length of longest control path .
- Width of major data path

The actual equations, which may be found in Leive's thesis, predict die basic performance factors from the parameters listed. The performance factors are as follows:

- Cost in dollars - This varies directly with the number of bindable nodes and die width of the major data path, and is the lump sum of all the manufacturing costs associated with a particular module. In his thesis, Lcive also argues that the area of the chip is directly related to the cost
- Power consumption - This also varies directly with the number of bindable nodes and the width of the major data path.
- Delay - This varies directly with the length of the longest control path.

3.11.1 Estimating Leive's Parameters

When he obtained the necessary parameters for his equations, Leive had available a fully bound data path graph produced by Hafer's allocator. Because the current generation of the CMU-DA system did not have a working allocator, it was necessary in this project to estimate the results of the allocation process in advance, assuming a generic distributed-style allocator. This estimation process did not always yield the precise, quantitative results that could have been obtained from a working allocator, but it is felt that the results obtained nevertheless provided a qualitative ^{N}feel^{M} for the results of the transformations, and in some cases produced quantitative values.

The number ofbindable nodes is the most difficult parameter to estimate. This quantity is the sum of the number of operators to be bound, the number of registers needed, the number of multiplexors needed, and the number of memories in die system. The number of operators to be bound is heavily dependent on the data path allocator to be used. It is estimated by determining the number of operators that might possibly require allocation, and modified as necessary using the preliminary information from [Hitchcock 82]. The number of registers needed is also highly allocator dependent, and is currently estimated by routines provided by Hitchcock. The number of multiplexors needed is estimated by determining the number of individual SELECT outputs, since each of these will require the multiplexing of the outputs of each branch of the SELECT. The number of memories in the system can be determined directly from the Value Trace. Estimating the number of bindable nodes is a difficult proposition at best, and seldom yields quantitative results, but its intelligent use should still provide qualitative information on the results of particular transformations. The metrics developed s o far are thus:

- Maximum number of aDocatable operators - The maximum number of operators that might require allocation.

- Register estimate * The estimated number of registers needed, as determined by Hitchcock's routines.

- Multiplexor estimate - The* estimated number of multiplexors needed, as determined from the outputs of the SELECTS.

- Number of bindable nodes - The sum of the three quantities above, plus the number of memories in die system. This is the estimated number of elements that will actually be bound to hardware, and is determined from the Value Trace.

The next of Leive's parameters, the length of die longest control path, is easier to determine, as it depends on the results of the control step allocator. There currently exists an automatic control step allocator in die CMU-DA system, and it is relatively simple to assign control steps to a given vtbody and determine the longest control path. To determine the control path for the *complete* Value Trace, some hand analysis must be performed because no software currently exists for calculating the longest control path through multiple vtbodies. Most of the transformations take place only on a single vtbody or on a pair of vtbodies, so this analysis is usually straightforward. The metric is thus:

- Length of the longest control path - The length of the longest control path, as estimated from the Value Trace.

The third parameter of Leive's, the width of the major data path, is die easiest to compute from the Value Trace. Using Leive's criterion for measurement of this parameter (counting the number of path links of each bit width in the design, multiplying the number of lengths by their bit width, and selecting the width corresponding to the largest product), it can easily be computed. While it is computed from the Value Trace rather than a data path graph, it is felt that the results will usually be die same, because most paths that exist in the Value Trace also exist in die data path graph. The metric is defined as follows:

- Major data path width * The width of the major data path, as estimated from the Value Trace.

3.2.2. The Control Part

In varying the control portion of the design, there are two related items which may be changed: the microcontroller style, and the operator delays. The operator delays, in turn, arc dependent on die microcontroller style, as well as on the implementation technology. Assuming die implementation technology to be fixed, the major variable is the microcontroller style, which may have a definite effect on the results of die transformations, and is well suited to variation. The remainder of this section discusses how this change is accomplished.

3.111. Microsequencer with IFC

As described in [Mano 76] and [Mick 80], a typical microcontroller with implicit flow control (IFC) in all instructions is of the form shown in Figure 3-1 The word to be read from the control memory is selected by the ROM address register, and is maintained at die output of that register until another value is produced. Each of the words read from the control memory is called a *microinstruction,* and contains the following fields:

- Data operations - These bits represent an operation to be performed by the microsequencer.

- Address control - After the current microinstruction has been executed, the ROM address register must be supplied with the address of the next instruction. This instruction may be in one of two places: it may be in the next sequential location, or it may be located elsewhere. If this instruction is in the next sequential location, the address generator can simply increment a counter, but if it is located elsewhere, some means must be provided for specifying this information. With implicit flow control, this next address (or possible next address, in the case of conditional statements) is specified in a predefined field in *each* instruction. These address control bits specify whether or not the next instruction is to be determined sequentially, and also specify control conditions for conditional instructions.

- Next address -These bits specify the next address for the control memory, and are used in conjunction with the address control field as defined above.

Using a sequencer with IFC, CALLs and SELECTS will take *zero* time, or, more precisely, the same amount of time as sequentially accessing the next instructions.

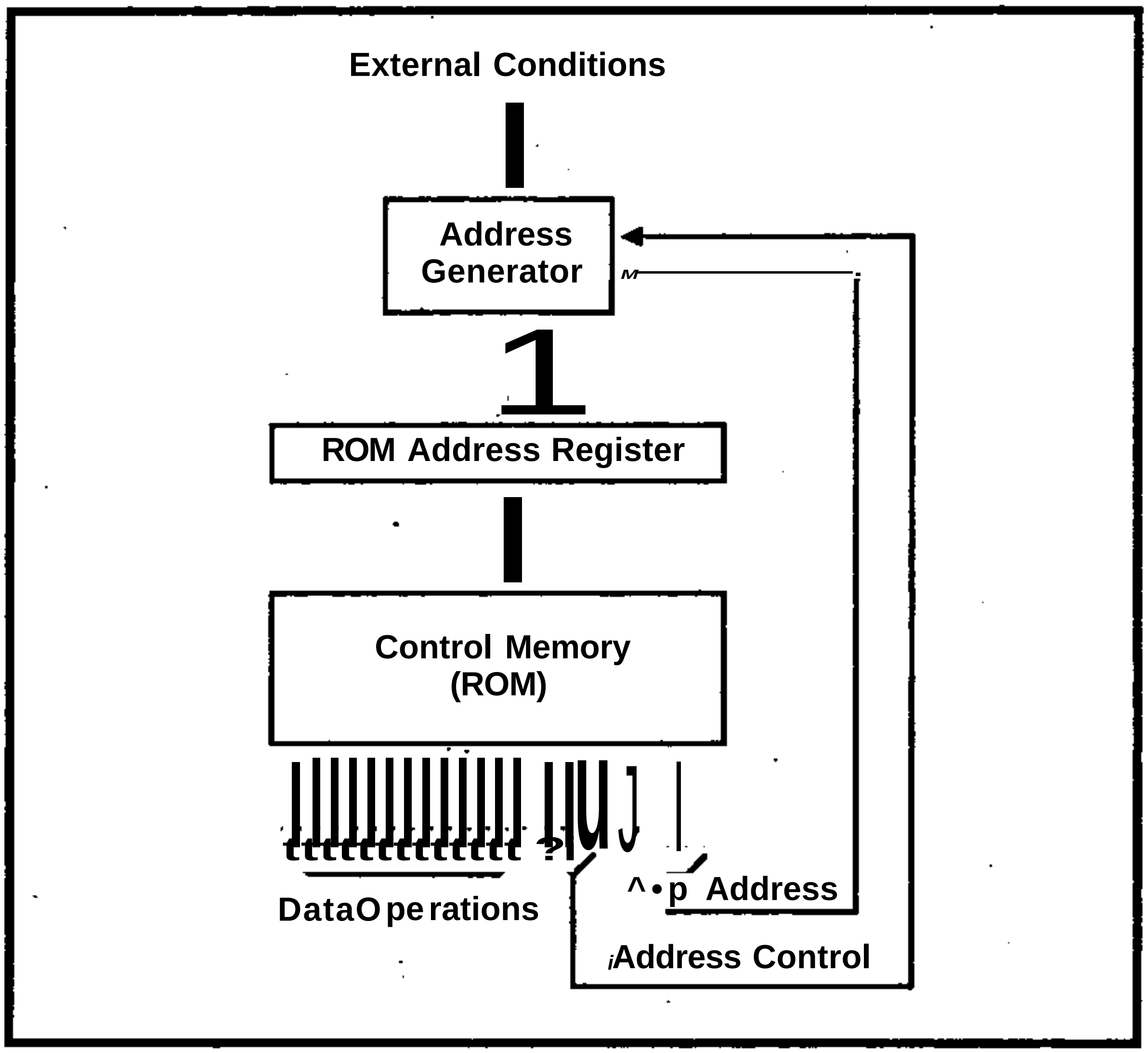

Figure >2: Microsequencer with IFC

322.2. Microsequencer **with** EFC

Another type of microsequencer is that with explicit flow control (EFQ instructions. While die IFC controller encodes the flow control information implicitly in every word, the EFC controller has explicit instructions for flow control. Implementation details may vary, but a method described in [Agrawala 76] uses one field of die microinstruction to control the interpretation of the remainder of the fields. This method allows for a set of instructions analogous to assembly language instruction sets. Using this scheme, the microsequencer assumes sequential execution of all instructions, except on encountering an explicit flow control (EFQ instruction. For example, the Burroughs B1700 has a distinct set of 31 microinstructions, with a variable number of initial bits specifying the instruction, and the remainder used to supply whatever data is needed by the instruction. When used as a controller in conjunction with the Value Trace, a microscqucnccr with EFC will require a *non-zero* amount of time to execute flow control constructs like the CALL and SELECT. Another example of a controller with EFC was the control allocator constructed by Cloutier [Cloutier 80] and Nagle [Naglc 81] in the second generation of the CMU-DA system.

3.2.3. Other Metrics

In addition to the metrics provided by Leive's equations, two other metrics were identified as providing worthwhile information on the results of the transformations. The first of these metrics, the total number of control steps, is essentially the number of words of microcode in the mkrosequenccr. While it is dependent on the control step allocator, this metric is also dependent on the type of microcoding used by the mkrosequencer, as will be shown in the next chapter. It is easily determined from a Value Trace with control step allocation information.

The second of these metrics, the total number of operators, provides a relative measure of die complexity of the Value Trace. This is a measure of complexity because a design with a large number of operators is more complex and takes more time to process than one with a smaller number of operators. While the elimination of a non-allocatable operator like a bit read may not improve die performance of the design as measured by the parameters defined so far, it is important to note that its elimination *is* significant, if for no other reason than that subsequent processing of the vtbody win be faster due to its absence. The additional metrics are thus:

- Number of control steps - The number of control steps in the Value Trace, after control step allocation has been performed.

- Number of operators - The number of operators in the Value Trace.

3.3. Conclusion

As part of this Master's project, most of the metrics described in this chapter have been implemented. The portion not implemented is the Inside knowledge" required to correctly apply Leive's equations to the Value Trace, since a full characterization of this inside knowledge would constitute a working allocator. Nevertheless, the metrics implemented can provide a useful measure of die results of the transformations. This will be shown in the next chapter, which consists of a scries of large examples showing the use of the transformations. The metrics and variations on the microcontroller discussed in this chapter will be used to evaluate the results.

Chapter 4
Design Examples

One example is more valuable... than twenty precepts written in books.

- Robert Ascham, *The Scholemaster*

To evaluate die usefulness of the transformations and metrics developed as part of this Master's project, four examples of different classes of transformations are selected and examined. In assigning control steps to the designs, it is assumed that die design will be implemented as a distributed-style TTL design with a controller based on the AM2910 microsequencer. For each example, the results of the transformation as shown by die metrics of the previous chapter are given, and the design is evaluated for both IFC and EFC microcontrollers. '

4.1. SELECT Factoring and Motion in the AM2903

The ISP description shown in Figure 4-1 and the VT diagram shown in Figure 4-2 is the function calculation section of the Advanced Micro Devices AM2903 bit slice microprocessor, with the first branch simplified[4]. As can be seen in the figure, the second and third branches of the DECODE statement are essentially the same, with the only difference being the computation involving R and S. These two branches may be moved from this SELECT into a new one using SELECT factoring, and the common operations moved below this new SELECT using downward SELECT motion. The final VT diagram is shown in Figure 4-3, and the results of the transformations are summarized in Table 4-L Appendices B.1 and B.2 contain a listing of the Value Trace before and after these operations are performed.

[4]This branch contains the decoding for multiplication, divisk^ and normalization, but the full contents were not shown for reasons of simplicity.

```
!**************************************************************

SELECT :=
    begin

**PC.State**

    R<3:0>,                               ! R inputs to ALU
    S<3:0>                                ! S inputs to ALU

**External.State**

    Cn<>,                                 ! Carry in
    F<3:0>,                               ! Output from ALU
    I<8:0>                                ! Instruction inputs

**Instruction.Execution**{us}

    exec :=
        begin
        DECODE I<4:1> =>
            begin
            "0 := no.op(),
            "1 := F = ((S - R) - 1) + Cn,
            "2 := F = ((R - S) - 1) + Cn,
            "3 := F = (R + S) + Cn,
            "4 := F = S + Cn,
            "5 := F = (not S) + Cn,
            "6 := F = R + Cn,
            "7 := F = (not R) + Cn,
            "8 := F = 0,
            "9 := F = (not R) and S,
            "A := F = R eqv S,
            "B := F = R xor S,
            "C := F = R and S,
            "D := F = not (R or S),
            "E := F = not (R and S),
            "F := F = R or S
            end
        end
    end

!**************************************************************
```

Figure 4-1: Function Calculation in the AM2903 - The ISP Description

4.1.1. Interpreting the Results

As can be seen from the table, the maximum number of allocatable operators will decrease somewhat. With a smart data-path allocator, the actual number of operators that will be allocated hardware will probably not change. While this will depend on the design style and optimizing capability of the allocator, most allocators will realize that operators common to all branches of a SELECT can be implemented with the same piece of hardware, since only one of the branches will be active at a time. In this case, the amount of hardware required will be the same before and after SELECT motion downward.

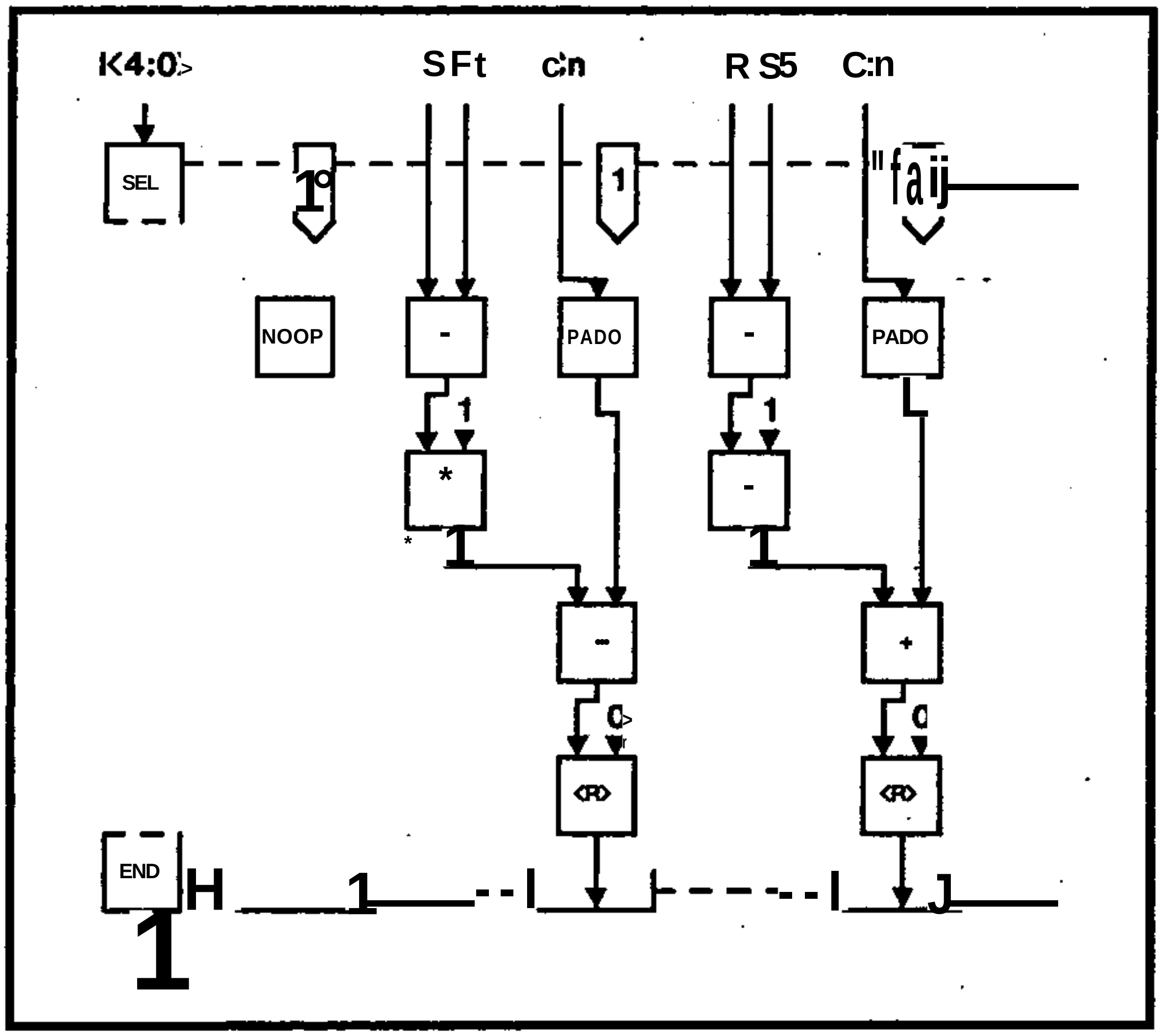

Figure 4-2: Function Calculation Before Transformation - A Partial VT Diagram

Using the criterion established in the last chapter for estimating the number of multiplexors, this number will increase due to the addition of the new SELECT required by the SELECT factoring. Because this new SELECT has only one output, die number of multiplexors will increase by one (the number of estimated multiplexors is the number of outputs of SELECTS that do not go to memories).

With the addition of the SELECT, the number of registers needed will also increase. Because the selector input to the old SELECT win now be used as an input for two SELECTS, a register will be required to hold the value from die time it is computed until it is required for die second SELECT. This extra register was not needed before because die value was used immediately after it was

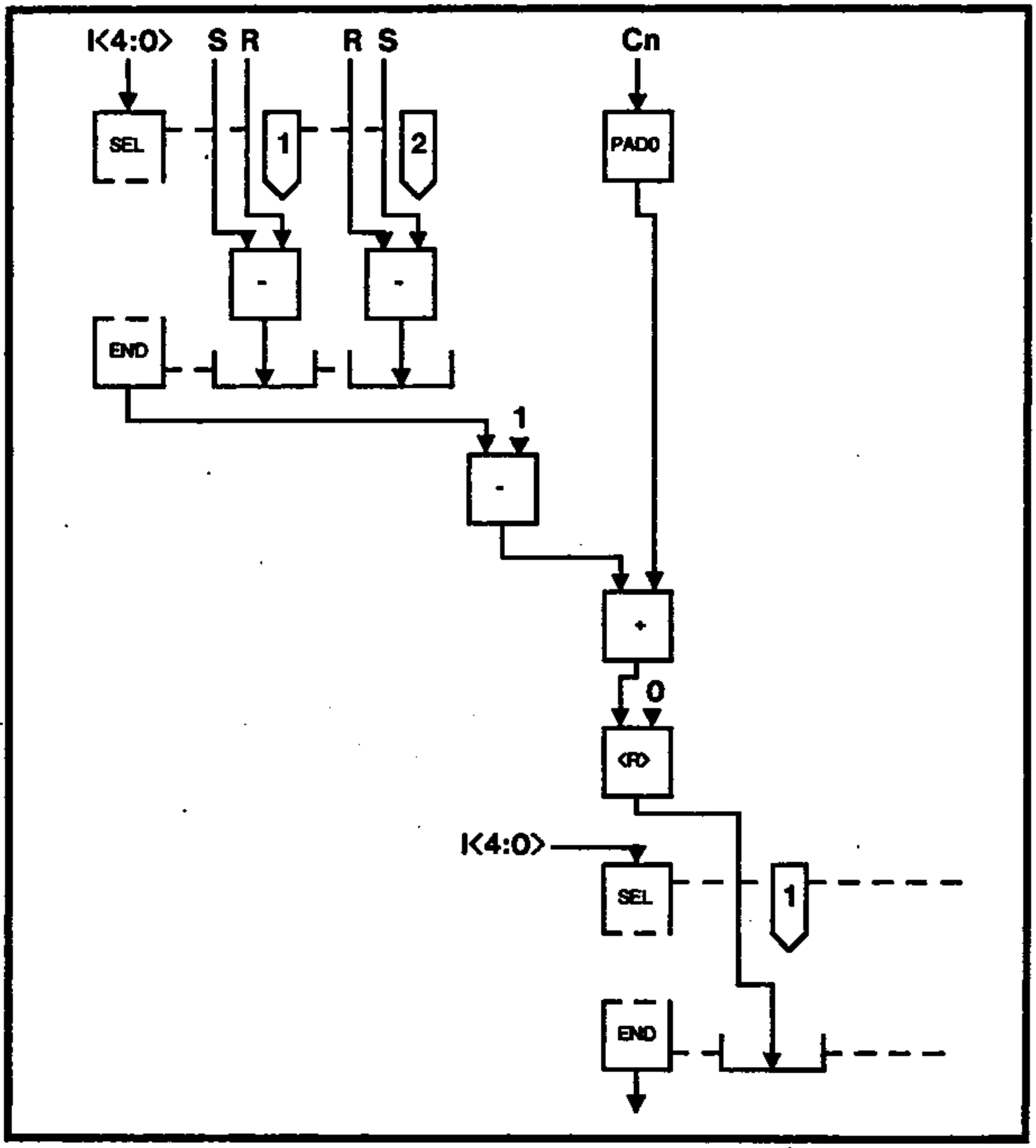

Figure 4-3: Function Calculation After Transformation- A Partial VT Diagram

computed, but win now be required because die value must be retained through die computations in and below the new SELECT.

The number of bindablc nodes will thus increased slightly due to the addition of the new SELECT. This is because die number of multiplexors will increase and because die number of

	ETC Controller				IFCODntroUer			
	Before	After	Change		Before	After	Change	
MaxAlloc.Ops.	24	22	-2	-8.3%	24	22	-2	-8.3%
Reg. Est.	5	7	+ 2	+ 40.0%	5	7	+2	+ 40.0%
Mux.Est.	1	2	+ 1	+ 50.0%	1	2	+ 1	+ 50.0%
Bindabte Nodes	30	31	+ 1	+ 3.3%	30	31	+ 1	+3.3%
Longest Path	5	7	4-2	+40.0%	4	5	+ 1	+ 25.0%
Major Path Width	4	4	+0	+ 0.0%	4	4	+0	+0.0%
Control Steps	35	35	+0	+0.0%	19	18	-1	•5.3%
Operators	42	39	-3	-7.1%	42	39	-3	-7.1%

Table 4-1: Results of Select Factoring and Motion in the AM2903

registers will increase. As mentioned earlier, this assumes that the data allocator is capable of recognizing common operations in adjacent brandies of SELECTS, and uses the same piece of hardware for each.

The length of the longest control path will increase for two reasons. The extra SELECT will be added, and extra control steps will be required for it if an ETC controller is used. The number of control steps will also increase with the shifting of control steps among the SELECTS. Before optimizing, the longest path through the SELECT itself will be two steps in length. After optimizing, the longest path through the old SELECT will be only one step in length, but an additional step will be required through the new SELECT, and another will be required for the operations moved out of the new SELECT.

The number of control steps will remain approximately the same, for similar reasons. An extra control step will now be required by the SELECT if an EFC controller is used, and the total number of control steps for the rest of the design will decrease by one. This is due to the replacement of two control steps per branch prior to downward SELECT modem with one control step per branch and one control step below the SELET after downward SELECT motion. If die branches had been longer, this decrease would have been more pronounced.

The total number of operators will be reduced because of die replacement of multiple operators above the SELECT with a single operator below the SELECT. Due to the addition of the new SELECT, the total number of operators will also increase by one.

4.1.2. Comments on the Results

As envisioned by Snow, SELECT factoring was a transformation for breaking a complex SELECT up into a number of smaller SELECTS, for facilitating SELECT motion, or for making other transformations possible, and it had no negative side effects. In the course of this evaluation, the following negative side effects were discovered:

- The creation of a new SELECT will usually require the addition of an extra register, so that die old selector will now be available to both the old and the new SELECT.
- For each output of the SELECT not going to a memory element, an extra multiplexor may be required.
- The longest control path and the total number of control steps will always increase if an ETC controller is to be used, because the SELECT itself will require a new control step.
- The number of bindable nodes will also be increased by the addition of die new SELECT.

From the results obtained in this section, it appears that the desirability of SELECT factoring and SELECT motion depend both on data and control style of the design. The number of allocatable operators is affected by the data allocator used and by the type of microcontroller used, and the length of the control path and the number of control steps is affected by the microcontroller style. It also appears that SELECT motion by itself has the positive results of reducing the number of operators and possibly the number of control steps, without having the negative side effects associated with SELECT factoring.

4.2. Vtbody Inline Expansion in the Intel 8080

The ISP description shown in Figure 4-4 is the instruction decoding section of the Intel 8080. The only instruction call explicitly shown is that of die RRC instruction; the calls to instructions in die other branches have been replaced by no-ops. Because the RRC subroutine is only called once, it may be expanded inline and the old vtbody eliminated The results of this operation are shown in Table 4-1 The Value Trace before and after these operations are performed is given in Appendices Cl and C2. Since all of die instructions in die Intel 8080 are only called once, they may all be expanded. Table 4-3 shows the results of these expansions. The question marks (??) represent information that is not available at tfcis time, but which should become available when the work in [Hitchcock 82] is completed.

```
|***************************************************************

INLINE :-
    begin

    •• PC.State ••

    psw<7:0>,
        CYo :• ps«<0>,
    A<7:0>

    •• Instruction.Foraat ••

    IR<7:0>,
        group<l:0>   :* IR<7:6>,
        df1e1d<2:0> :• IR<5:3>,
        sf1eld<2:0> :• IR<2:0>

    •• Instruction.Execution ••

    Exec:-
        begin
        DECODE group •>
           . begin
            0 :« DECODE sfieid «>
                begin
                [0:6] :• no.op(),
                7 :- DECODE dfieid ->
                    begin
                    [0,2:7] :- no.op()t
                    1 :- RRC()                1 RRC instruction
                    end
                end.
            [1:3] :• no.op()
            end
        end,

    RRC :•
        begin
        CY • A<0> next
        A - A srr 1
        end

    end
```

Figure 4-4: Instruction Decoding in the Intel 8080

4.2.1. Interpreting the Results

Because the only operators affected by inline expansion are the CALL to die vtbody to, be expanded and the LEAVE from this vtbody* the number of bindable nodes will not change. The operators themselves are not allocatable, being purely flow control instructions, and no multiplexors or registers will be eliminated by their removal.

By eliminating the CALL and the SELECT, two control steps will be eliminated for an EFC

	ETC Controller				IFCOantroUer			
	Before	After	Change		Before	After	Change	
MaxAflocOps.	1	1	+0	4-0.0%	1	1	+0	4-0.0%
Reg. Est.	7?	??	7?	7?	7?	7?	7?	4-039%
MuxEst.	6	6	• 0	-t-0.0%	6	6	4-0	4-0.0%
Bindable Nodes	7?	7?	7?	7?	7?	7?	7?	4-039%
Longest Path	10	8	-2	•20.0%	6	5	-1	-16.6%
Major Path Width	8	8	-i-O	4-0.0%	8	8	4-0	+0.0%
Control Steps	16	14	-2	-12.5%	9	8	-1	-11.1%
Operators	15	13	-2	-13.3%	15	13	-2	-13.3%

Table 4-2: Results of Inline Expansion in the Abbreviated 8080

	EFCController				IFCODntroller			
	Before	After	Change		Before	After	Change	
Control Steps	542	390	•152	•28.0%	361	226	-135	-37.4%
Operators	730	578	-152	•20.8%	730	578	-152	-20.8%

Table 4-3: Results of Inline Expansion in the Full 8080

controller and one for an IFC controller. With the ETC controller, this will occur because each of the two operators has its own distinct control step before transformation, and these two control steps are eliminated. With the IFC controller, although the CALL and LEAVE will not be required to each have an unshared control step simply because they are flow control instructions, the CALL will have an unshared control step in this example because it is the only operator in its branch and can not share a control step with other operators. The length of the bngest control path will be reduced by 2 for the EFC controller and by 1 for the IFC controller for exactly the same reasons.

The total number of operators will always be reduced because of the elimination of die CALL to the vtbody which is to be expanded inline, and die LEAVE at the end of this vtbody.

4.2.2. Comments on the Results

Although Snow listed a couple of beneficial results of vtbody inline expansion, he did not list the reduction in the length of the longest control path, the number of control steps, and the number of operators resulting from the elimination of CALLs and LEAVEs when expanding inline all vtbodies which are only called once. Many ISP descriptions, especially those of processors, have some sort of instruction decoding loop which decodes the instruction codes and calls separate subroutines to perform the appropriate operation, so the inline expansion of these subroutines is not uncommon. While it is simpler to write (and much simpler to understand) a one-page instruction decoding loop which calls four pages of subroutines for the execution of the instructions than it is to write a five page loop which handles all of this, the latter would be represented much more eflkiendy at the VT level. The CMU-DA system is attempting to abstract away such biases imposed on the design by the designer's programming style, and inline expansion may be used to convert the internal representation of the first style into that of the second. In the Intel 8080, this reduction of the biases imposed by the designer's coding style reduces the number of control steps, and thus the amount of microcode required for implementation, by 28 to 37 percent

4.3. Loop Unwinding

The ISP description shown in Figure 4-5 is an example of a looping construct which executes the subroutine ACTION0 three times. A VT diagram of the inner loop (containing the loop count mechanism) is shown in Figure 4-6. As the loop is unwound, the following occurs:

- The CALL to the inner loop from the main loop is expanded inline in the main loop.
- The + operator now has as inputs the constant zero (the initial value of COUNT) and the constant one (the increment), so constant folding may be applied.
- After this constant folding, the bit read following the + operator now has as inputs the constant one (from the 4- operator) and the constant zero (the field offset it is to read from), so constant folding may be applied again.
- After this constant folding, the LSS operator now has as inputs the constant one (from the bit read) and the constant 3 (the maximum loop count), so constant folding may be applied again.
- After this constant folding, the SELECT operator has as its input the constant 1 (signifying TRUE), and constant folding may be applied again.
- The SELECT is now eliminated, and all that remains of the expanded inner loop is a call to ACTION0, and a RESTART operator which restarts the loop with a new value for COUNT.

```
LOOP :-
    begin

    •• Loop.State ••
    CountXLoop.Counter<2:0>

    •• Setup ••
    MIAMain.Loop :•
        begin
        Count * 0 next
        IL()
        end

    •• What.V111.Be.Unwound ••
    ILMnner.Loop :•
        begin
        ACTIOM() next
        Count • Count + 1 next
        1f Count Lss 3 •>
            restart IL
        end

    •• Example.Of.Action.To.Be.Performed ••
    ACTI0N\Act1on :-
        begin
        no.op
        end

    end
|*****************************************************************
```

Figure 4-5: Loop to be Executed Three Times

At this point, the main loop is the same as before, with the addition of a call to ACTIONO and an initial loop value of one. After two more applications of loop unwinding and constant folding like that above, the main loop consists of three calls to ACTIONO. and the unused vtbody representing the inner loop may be eliminated. Because the results of transformation are affected by the parameters of the subroutine ACTIONO, the results are computed for short, medium, and long versions of this subroutine. For the short ACTIONO, the longest control path length is assumed to be 5, the maximum number of bindable nodes 10, the number of operators IS, and the number of control steps 10. For the medium ACTIONO, these numbers are doubled, and for the longest ACTIONO, they are tripled. The Value Trace before and after these transformations are performed is given in Appendices D.I and D.1

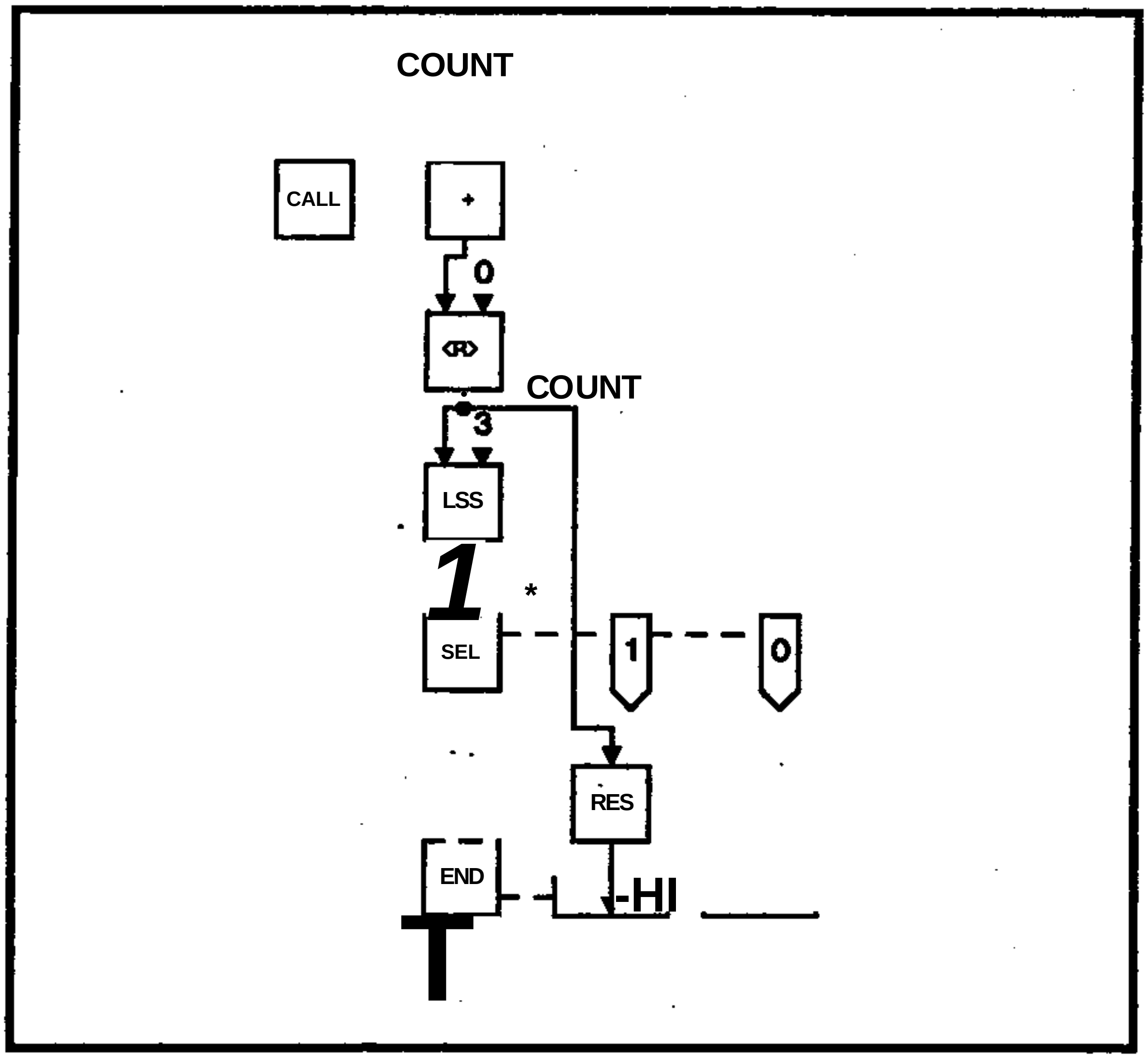

Figure 4-6: A VT Diagram of the Inner Loop Before Transformation

	EFCControfler				IFCDsntroiter			
	Before	After	Change		Before	After	Change	
Bindable Nodes	13	10	-3	-23.1%	13	10	-3	-23.1%
Longest Path	35	19	•16	-45.7%	32	19	-13	-40.6%
Major Path Width	3	3	+0	+0.0%	3	3	+0	+0.0%
Control Steps	22	15	-7	•313%	24	20	-4	•16.7%
Operators	25	20	-5	-20.0%	25	20	•5	•20.0%

Table 4-4: Results of Loop Unwinding - Short ACTION

	EFCController				IFCODntroller			
	Before	After	Change		Before	After	Change	
Bindable Nodes	23	20	-3	-13.0%	23	20	-3	•13.0%
Longest Path	50	34	•16	•32.0%	47	34	-13	-27.7%
Major Path Width	3	3	+0	+0.0%	3	3	+0	+0.0%
Control Steps	32	25	•7	-21.9%	39	35	-4	-10.3%
Operators	40	35	-5	-12.5%	40	35	-5	-12.5%

Table 4-5: Results of Loop Unwinding - Medium ACTION

	EFCController				IFCODotroiler			
	Before	After	Change		Before	After	Change	
Bindable Nodes	33	30	-3	-9.1%	33	30	-3	-9.1%
Longest Path	65	49	-16	•24.6%	62	49	-13	-21.0%
Major Path Width	3	3	+0	-t-0.0%	3	3	•0	•0.0%
Control Steps	42	35	-7	-16.7%	54	50	-4	-7.4%
Operators	55	50	-5	-9.1%	55	50	-5	-9.1%

Table 4-6: Results of Loop Unwinding - Long ACTION

4.3.1. Interpreting the Results

With file elimination of the + and LSS operators, the number of maximum alloeatable operators will decrease. Unlike the previous examples in this chapter, the number of operators in this example that will require binding to hardware will decrease because there are no other alloeatable operators in the vtbody with which the hardware could be shared prior to optimizing. The data path allocator currently under construction by Kowalski [Kowalski 81] does aDow sharing of hardware between vtbodies, so there might be a smaller decrease in the number of alloeatable operators when this allocator is used.

The number of registers required will also decreased, due to the elimination of the register used to hold the value of COUNT after die bit read operator in the inner loop. This register is required to hold this value after it is produced until it is needed by die RESTART operator, but is eliminated with the elimination of the loop count mechanism.

The number of bindablc nodes will decrease due to the decrease in allocatablc operators and the decrease in registers required. No multiplexors will be eliminated because the SELECT in the inner loop has no outputs. The decrease in the number of bindablc nodes will be due solely to the decrease in allocatablc operators and registers.

With the elimination of all the operators in the loop count mechanism and the amount of repetitions of this loop, the length of the longest control path will be reduced considerably, as will the total number of control steps. The length of the longest control path is also a function of the number of times the loop is repeated, and will be decreased by a larger amount

Because all of the operators in the inner loop are eliminated by the full unwinding process, the number of operators will be decreased by exactly that amount

4.3.2. Comments on the Results

The method of loop unwinding used in this project differs somewhat from that defined by Snow in his thesis. As envisioned by Snow, loop unwinding consists of splitting off a separate vtbody for each instantiation of a loop, while in this Master's project, loop unwinding consists of expanding die instantiations inline. Snow's form may be converted into the form used in this project by inline expansion, and the form used in this project may be converted into Snow's form by vtbody formation. There are advantages and disadvantages to both forms. With the multiple vtbody form, the major disadvantage is that new vtbodies must be created, necessitating additional CALLs and LEAVES and time involved with the procedure call. With this same form, the major advantage occurs when the action performed by the loop does not exist in a separate vtbody, but is reproduced inline. In this case, unrolling with the method used in this project results in multiple copies of die same code reproduced inline, and a tradeoff between the number of operators eliminated by die removal of the loop count mechanism and the amount of operators duplicated by the inline expansion of the loops. Loop unrolling by inline expansion should be used only when die number of operators eliminated is greater than die number of operators in the action to be performed.

4.4. Redundant Operator Elimination and inline Expansion in the AM2901

The ISP description shown in Appendix El is the description of the AM2901 bit slice microprocessor. In five of the eight vtbodies resulting from the ISP-to-VT translation, redundant operator elimination may be performed, eliminating 7 bit read operators, 6 NOT operators, 6 OR operators, 3 AND operators, and 1 EQL operator. Because they are only called once, three of the vtbodies (SOURCE, DEST, and EXEC) may be expanded inline. No figures are shown because these transformations are spread across five separate vtbodies. The results of these operations are summarized in Table 4-7. The question marks (??) represent information that is not available at this time, but which should become availabe when die work in [Hitchcock 82] is completed. Appendices E.2 and E3 contain a listing of the Value Trace before and after these operations are performed.

	EFCControUer				IFCC<xrtroUer			
	Before	After	Change		Before	After	Change	
MaxAllocOps.	78	62	-16	-20.5%	78	62	•16	-20.5%
Reg.Est.	63	r>	7>	7?	63	7?	7?	7?
Mux.Es*.	16	16	' +0	•0.0%	16	16	• 0	•0.0%
Bindable Nodes	157	7?	7>	7?	157	7?	7?	7?
Longest Path •	20	12	•8	-40.0%	15	9	•6	-40.0%
Control Steps	91	83	-8	-8.8%	62	56	-6	-9.7%
Operators	163	134	•29	-17.8%	163	134	-29	-17.8%
• - Only for vtbodies RUN, SOURCE, DEST, and EXEC								

Table 4-7: Results of Optimizing the AM2901

4.4.1. Interpreting the Results

As a result of the redundant operator elimination, the maximum number of allocatable operators will decrease. This reduction does not include the elimination of redundant bit read operators because bit reads are not allocatable operators. The number by which the allocatable operators will decrease will probably be much less, and will depend on how well the data allocator being used shares hardware among redundant operators.

The longest path for the SOURCE, DEST, EXEC and RUN vtbodies will be reduced by die inline expansion of the first three vtbodies into the last, which will result in the elimination of three sets of

CALLs and LEAVEs. As in the example in Section 42, the results will vary with the type of controller used, although the percentage of improvement will remain the same for this example.

The total number of control steps will be reduced. This reduction has two components: part of it is because of the elimination of the arithmetic and logical operators during the redundant operator elimination, and part is due to the elimination of the CALLs and LEAVEs during the inline expansion. In the first case, the reduction will come about due to the reduction of all operators other than bit read operators.

The total number of operators will also be produced, because of both the elimination of the arithmetic and logical operators as well as the elimination of the CALLs and LEAVEs. It will also be reduced by the elimination of the bit reads operators.

4.4.2. Comments on the Results

Redundant operator elimination remains much as originally defined by Snow, with potential for reduction in both the number of control steps as well as the amount of hardware required for implementation. The degree of benefit obtained from its use is heavily affected by the data style of the design, and particularly by the individual data path allocator. It should always be performed, as it will result in a reduction in die complexity of the Value Trace by the elimination of operators.

4.5. Conclusion

As part of this Master's project, design examples were chosen as candidates for transformation, and evaluated before and after these transformations were performed. From this evaluation process, new information was derived about the relationship between the transformations and the data and control style of the design, effects were noted that were not catalogued by Snow, and the metrics developed earlier were shown to be useful in measuring changes in the design. This information will be useful for providing heuristics for aiding the designer in choosing what transformations to apply and when, and provides new questions regarding die role of transformations in the CMU-DA system. These issues and others will be discussed in the next chapter.

Chapter 5
Analysis of Results

In this life we want nothing but facts, Sir, nothing but facts.

- Dickens, *Hard Times*

Facts, or what a man believes to be facts, are delightful... Get your facts first, and then you can distort them as much as you please.

- Mark Twain

After evaluating a number of design examples, die design space resulting from die transformations involved with these examples was explored. As part of this exploration, general conclusions based on the transformations in question were drawn when possible. With the evaluation of this data, the feasibility of automatkally guiding the process of choosing when to apply the various transformations was also explored. These issues are addressed in this chapter.

5.1. Exploring the Design Space

Once the metrics described in Chapter 3 have been used to find the necessary parameters, it is possible to use Leive's equations [Leive 81] as described in Section 3.2.1.1 to characterize the design *space* of the final design. Plots of this design space for the SELECT factoring/motion example of Section 4.1 and the loop unwinding example of Section 4.3 are shown in the following sections.

For these same examples, plots are also shown of the tradeoff between the number of bindable nodes and the number of control steps. The number of bindable nodes provides a measure of die amount of hardware operators required in the design, and the number of control steps provides a measure of the amount of microcode required by die controller, so these plots might be viewed as showing the tradeoffs between implementing a part of the design in microcode firmware and implementing this same part of the design in hardware.

In all of these plots, the symbol "0" represents the original, unoptimized design, and the symbol T the final design. All other symbols represent the intermediate designs; most of these are represented by M+", but intermediate designs of special interest are represented by integers for ease of reference. It is possible for more than one step in the design to be represented by the same point, as multiple steps in the transformation process might have the same performance characteristics.

In order to more fully characterize the design space resulting from the transformations, the results of the work in this Master's project are compared to earlier work by Barbacci and Siewiorek [Barbacci 75]. Although this earlier work was of a somewhat different nature, there are enough parallels and contrasts involved to warrant discussion.

5.1.1. SELECT Factoring and Motion

Figures 5-1 and 5-2 are the Leive design space plots showing the performance changes during the transformation of the SELECT factoring/motion example from Section 4.1 (with EFC controller). Beginning at point "O", cost, power consumption, and delay are increased substantially by the SELECT factoring, resulting in the performance shown by point T\ As the bit read and PLUS operators are moved below the SELECT, power consumption and cost worsen, and delay improves, reaching the performance shown by point "2". SELECT motion is then applied to the PADO and MINUS operators, and power consumption and cost improve while delay remains constant, resulting in the performance shown by point "F*. The points of the design space in Figure 5-1 fall in a straight line because Leive's predictors relate both the cost and the power consumption *directly* to the number of bindable nodes and the width of the major data path. If the cost and power consumption could have been determined from a final design these points would not have fallen in such a straight line, as die cost and power consumption would vary for die individual types of hardware operators. Leive's predictors necessarily rely on an "average" cost and power consumption per operator, and thus force a linear distribution of points in the cost-power design space.

Figure 5-3 is a plot of the tradeoffs during the transformation occurring between the number of bindable nodes in the design and die number of control steps. As in die previous example, point "1" represents die performance of die design after SELECT factoring is applied, and point "2" represents die performance after SELECT motion has been applied to die bit read and PLUS operators. This

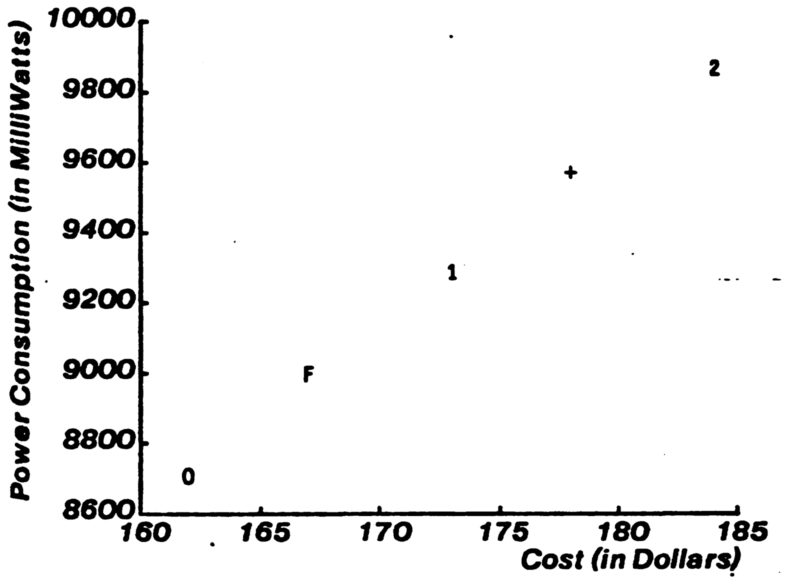

Figure 5-1: Select Factoring and Motion - Cost vs. Power Tradeoffs

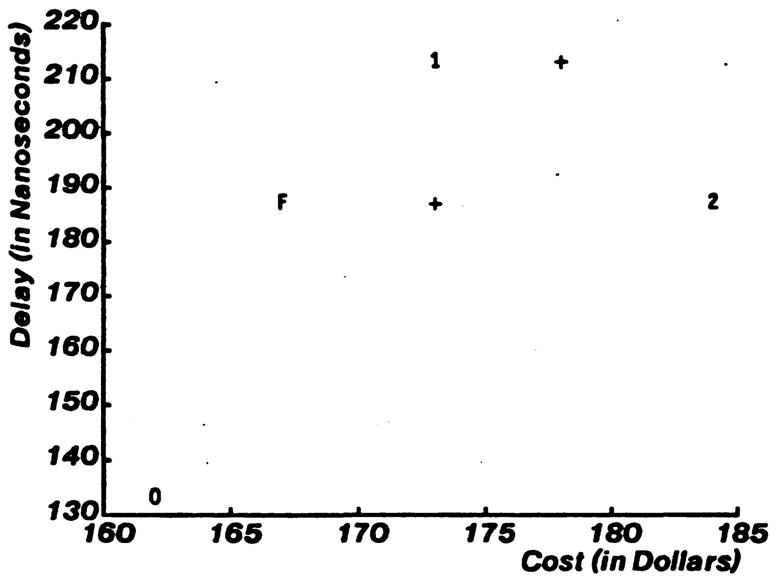

Figure 5-2: Select Factoring and Motion - Cost vs. Delay Tradeoffs

plot is similar in shape to that of Figure 5-2 (Cost vs. Delay), because the values being plotted in each are related. Those values on the abscissas of the graphs, number of bindable nodes and cost, are related by Leive's predictors and are directly proportional to each other. The values on the ordinates, the number of control steps and the delay, are related in the sense that both are measures of time.

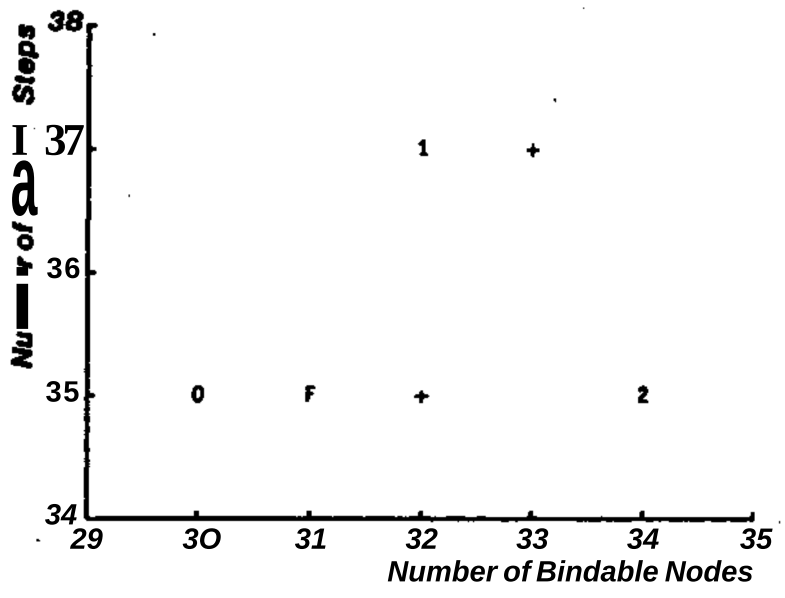

Figure 5-3: Select Factoring and Motion - Bindable Nodes vs. Control Steps

The number of control steps is the total number of discrete time steps through which the controller will pass at some point, and die delay is the actual execution delay of the system. The two are not directly proportional because in execution a single path may be traversed more than once, increasing the delay but not the amount of control steps.

As stated in Section 4.L2, SELECT factoring by itself has almost all negative effects, with the exception of allowing SELECT motion, and SELECT motion coupled with SELECT factoring may not improve the design. In this example, SELECT factoring causes a relatively large decrease in performance (from point "O" to point "1"), and SELECT motion does improve the design (from point ^{M}r to point T"), although not enough to overcome the decrease in performance caused by the SELECT factoring. As seen in the figures, this example of SELECT factoring/motion results in the cost of the design being increased by 3%, the delay by 41%, and the power consumption by 3%, so it would not pay to perform this transformation unless it is part of a larger set of transformations. This view of SELECT factoring/motion is in conflict with Snow's belief that SELECT factoring has no harmful side effects.

5.1.2. Loop Unwinding

The Lcive design space plots shown in Figures 5-4 and 5-5 arc those of the loop unwinding example from Section 4.3 (with ETC controller and short ACTION). The performance of the original design begins at point "O", and immediately falls to that of point "1" as the loop is expanded Constant folding is applied, and performance improves, moving downward and to the left on the plots. It eventually results in the same cost as the original design, but with more delay (at this point, the design is the same as before, with the exception of a different constant going to the next instantiation of the loop and an extra CALL to the action being performed). As the loop is expanded once more, the performance of the design is again at the point shown by "1" in Figure 5-4, and at the point shown by "2" in Figure 5-5. Once more, cost, power consumption, and delay improve as constant folding is performed, although not surpassing that of the original design. The final expansion of the loops results in point "1" in Figure 5-4 and point "3" in Figure 5-5, and constant folding improves the performance as before. When this final unwinding is complete, the vtbody containing the loop count mechanism is no longer needed and may be eliminated, resulting in the dramatic improvement shown by point " P. As seen in the figures, this loop unwinding results in a 23% improvement in cost and power consumption, and a 46% decrease in delay.

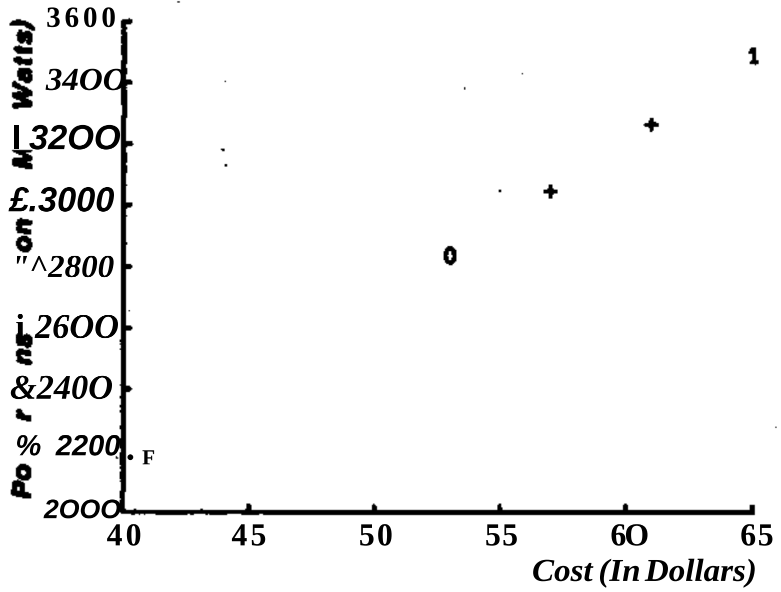

Figure 5-4: Loop Unwinding - Cost vs. Power Tradeoffs

The plot of die number of bindable nodes versus the number of control steps is shown in Figure 5-6, with points "1", "2", and "3" representing the same intermediate designs as in Figure 5-1 As in

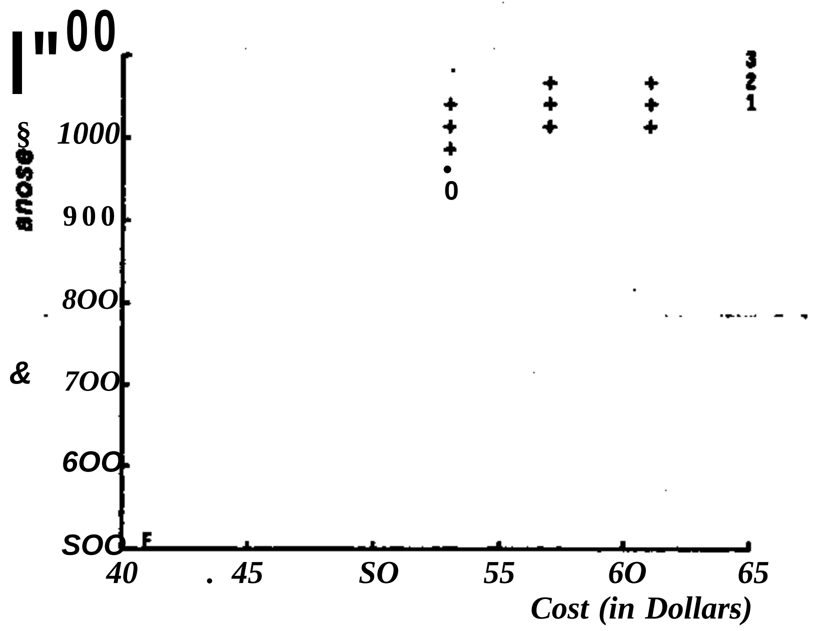

Figure 5-5: Loop Unwinding - Cost vs. Delay Tradeoffs

the previous example, it is similar to the plot of cost versus delay, and also shows the benefits of loop unwinding.

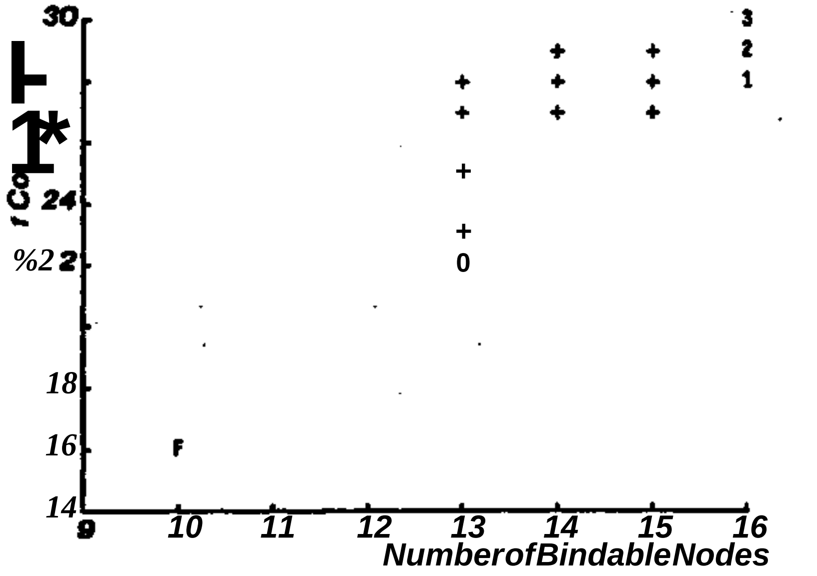

Figure 5-6: Loop Unwinding - Bindable Nodes vs. Control Steps

5.1.3. Comparison with Barbacci and Siewiorek's Work

In the first generation of die CMU-DA system, Barbacci and Sicwiorck [Barbacci 75] explored the tradeoffs involved between serial and parallel implementations of sections of a design. A program called EXPL [Barbacci 73] was written which took the output of the ISPS compiler and a set of user-specified constraints and generated a behavioral graph. It then performed various serial-to-parallel and parallel-to-serial conversions, and allocated hardware for each of the resulting designs. The results of this allocation, along with a simple set of heuristics, were used to guide the next iteration of the procedure. No transformations of the sort used in this Master's project were applied, as it was felt by Barbacci and Sicwiorck that these type of transformations were best left to the individual technology-dependent allocators in use. With this system, Barbacci and Siewiorek achieved time savings ranging between 10 and 40 percent, and a corresponding cost increase between 1 and 160 percent

It is interesting to explore the implications of these findings, and to compare their work to the work done as part of this Master's project Although both of these projects are concerned with altering die implementation of the design while maintaining the behavior, two entirely different methods of approach to the transformation process are used. The work done by Barbacci was intended to explore the tradeoffs involved with making a design more serial or more parallel, and improved one performance factor at the cost of another. The work done in this project was with a different type of transformation; it avoided entirely the serial-parallel tradeoffs[5], and attempted to improve one or more performance factors while maintaining the status quo of the others. Aside from the different views of this transformation process, other differences between Barbacci and Siewiorek's system and the current CMU-DA system include:

- The role of the ISP description was different The current generation of the CMU-DA system treats the ISP description *solely* as a behavioral description of the design. Barbacci and Siewiorek's system did use the ISP description as a behavioral description, but this system also used the ISP description as an initial specification of the hardware elements involved, often mapping ISP variables directly into hardware storage elements.

- The design styles were different The current generation of the CMU-DA system will be capable of generating distributed-style designs and bus-style designs when the allocators currently under construction are completed. Barbacci and Siewiorek's system used a very stylized type of design, with all data operators and operands connected to a bus. With this system, die amount of concurrency in the design was limited by the number of buses.

[5]In the current CMU-DA system, the serial-parallel tradeoffs are decided by the control step and data path allocators.

- The data allocation methods were different The current generation of the CMU-DA system will be capable of allocation in many ways, and is limited only by the allocation algorithms implemented by the designers of the allocators. Barbacci and Siewiorek's system was much more restricted, using a set of *templates* to specify the register-transfer level operations to be performed for each data operation in the ISP language.

- Barbacci and Siewiorek's system was a completed system, whereas the current CMU-DA system is not yet completed. Any evaluation of the results of the transformations given in this Master's project can only be estimated, since the final design can not yet be generated for analysis. Barbacci and Siewiorek not only had finished designs from which to work, but also performed more detailed analysis of the results. As an example of this analysis, their calculations for the execution times of the designs were much more precise, including such factors as the probability of executing a particular path in a multi-way branch.

Although there were differences between these two systems and the differences between the types of transformations used, the two systems produced improvements that were similar in scope. An important difference between the two systems is that Barbacci and Siewiorek's system was truly exploring *tradeoffs** and would improve one factor at the cost of another, while the system implemented for this project seldom worsened die other factors in this manner.

5.2. Guiding the transformation Process

One of die goals of this Master's project was to explore the possibility of automatically or semi-automatically guiding die transformation process, based on information determined in the course of this project as to the nature of the transformations. Some of the problems involved with this guidance were explored, and some work was done in implementing automatic and semi-automatic guides. This section discusses these problems and the resulting guides.

5.2.1. Problems Associated with Iterative Transformation and Allocation

As previously envisioned, die optimizer, control step allocator, and data allocator of the CMU-DA system were to be independent functions of the system, with die optimizer performing die transformations discussed in this report and acting in conjunction with the control step allocator / data allocator pair. It was hoped that there could be an iterative design process, with repeated use of die optimizer and allocators in an iterative manner. If this had been possible die optimizer would have had full access to the control step and data path allocation data, and could have used this data to guide the transformation process. There were a number of previously unforseen problems which hindered this iterative use.

5.11.1. Preservation of Allocation Information while Optimizing

In order to use the optimizer and the two allocators in any sequence, there must be some method of preserving the control step allocation and data allocation information through the application of the optimizing transformations. Because this control step information and the hardware data structures required for data allocation were not part of the Value Trace when it was conceived by Snow, he did not explore these issues. The examples below show the problems involved with the preservation of this information.

In performing constant folding, a single operator, usually arithmetic or logical in nature, is eliminated and is replaced with a constant value (see Section 12.1). If this is the only operator that uses the control step of which it is a part, the control step allocation may be preserved, because the elimination of a control step simply implies decrementing the control step number of all those control steps larger than the current one.* If more than one operator uses the same control step as this operator, die elimination of this operator may decrease the efficiency of the control step "packing.*[9] If hardware is allocated for this operator, the elimination of the operator invalidates the data-path allocation associated with it While it would be possible to simply deallocate this operator, this solution does not consider any decisions made which led to its allocation, and may cause inconsistencies in the allocation. If no hardware is allocated for this operator, the allocation is still valid.

In performing redundant operator elimination, two operators of the same type are replaced with a single operator (see Section *222).* If the control steps for the two operators are the same, the resulting single operator can be assigned these same control steps, and the control step allocation preserved If, as is more likely, the two operators have different control steps, die action to be performed is more difficult The simplest solution would be to assign die new operator the control steps of one of the original operators, and decrement the control step number of other control steps as necessary to fin up die "hole" left by the elimination of those control steps associated with the other original operator. This is not entirely straightforward, because die choice of which control steps to retain and which to eliminate depends on the algorithm used by die control step allocator, and should be made by this allocator. If hardware is allocated for both of the original operators, elimination of one of these operators will invalidate die hardware allocation as in the constant folding example above. If hardware is allocated for only one operator, the resulting single operator could receive this hardware allocation, and the allocation preserved. If no hardware is allocated at all, die allocation is also preserved

The problems above are representative of those involved with trying to preserve allocation information through the transformation process. While it may be possible to preserve this information in some cases, it is not possible to preserve it in others, especially in accordance with the original allocation algorithms. It appears that, with our current views on the role of the optimizer in the CMU-DA system, transformation must be completed *before* allocation is begun, unless the whole allocation process is to be repeated after every transformation. Further discussion on this subject, and a proposal for a new role for the global optimizer, will be given in Section *622.*

5.11.1 Availability of Data for Transformation Guidance

In order to guide the transformation process, certain data must be provided. In terms of the metrics discussed in Chapter 2, the computation of the register estimate, the length of the longest control path, and the number of control steps all depend on the results of the control step allocation. While the number of bindable nodes (and its individual components), the length of the longest control path, and the major data path width are currently being estimated from the Value Trace level, these should be determined from the completely allocated path graph, as discussed in Section *32.12.* In keeping with the views established in the last section, that transformation should take place *before* control step and data path allocation, the derivation of this information from the Value Trace level while transformation is being performed is not possible. Any guidance of the transformation process must be done without Ac benefit of this data. Section *622* also raises these issues.

5.2.2. Current Guides for Transformation

To explore the feasibility of implementing guides for the transformation process, a number of manual or semi-automatic guides and automatic guides were implemented. All of these guides use information directly available at the Value Trace level before control step and data path allocation have been performed. The semi-automatic guides are used to search the entire current vtbody for the applicability of particular transformations, which are then performed manually. The automatic guides are used to automatically perform this search, applying the transformation wherever appropriate.

5.2.2.1. Manual or Semi-Automatic Guides

The manual or semi-automatic commands implemented as part of this Master's project are all commands which look for the applicability of certain transformations on individual vtbodies or groups of vtbodies, and print out the appropriate information so the designer can manually perform the transformation if he wishes. Specific commands implemented are as follows:

- Look for constant folding - Searches the current vtbody for all applications of constant folding, and prints out a list of the operator numbers and operation type for all those activities to which constant folding may be applied.

- Look for redundant operator elimination - Searches the current vtbody for all applications of redundant operator elimination for pairs of operators, and prints out a list of the operators numbers and operation type for all those activities to which redundant operator elimination may be applied.

- Look for dead vtbody elimination - Searches the Value Trace as a whole for all vtbodies which are not referenced by any other vtbodies and which are not explicitly declared as the MAIN vtbody, and prints out a list of those vtbodies as being candidates for dead vtbody elimination.

- Look for downward SELECT motion - Searches the current vtbody for all applications of downward SELECT motion, and prints out a list of the numbers of the SELECTs involved and the output of each SELECT into which the operator will be moved.

5.2.2.2. Automatic

To explore the possibilities of automatically guiding the transformation process, one command was implemented to automatically perform vtbody inline expansion. This command is conditional in that it expands inline all vtbodies which are called only once. The command works as follows:

- All vtbodies in the Value Trace are scanned.

- If the vtbody is called more than once, it is not a candidate for expansion.

- If the vtbody is RESTARTed from any vtbody, it is not a candidate for expansion. This is due to the lack of any sort of labeling mechanism in the Value Trace environment. If this vtbody were expanded inline, there would no longer be any way to refer to the point which was the beginning of the vtbody before expansion, since the only labels inherent in the CMU-DA system are the implied labels at the beginning of each vtbody.

5.3. Conclusion

As part of this Master's project, die design space of some of the examples from the preceding chapter was explored, and the results of the transformations implemented for this project compared with earlier work by Barbacci and Sicwiorek. The feasibility of semi-automatic and automatic guidance of the transformation process was explored, and problems which would hinder this guidance evaluated A small number of semi-automatic and automatic guides were implemented which were not affected by these problems.

Chapter 6
Synopsis and Future Work

Remember this also, and be well persuaded of its truth. The future is not in the hands ofFate, butinours.

- Jules Jusserand

In order to provide a concise summary of the work performed in this Master's project, a synopsis is presented in the first half of this chapter. Since the wort: done in this project has pointed out some of the fallacies in role of the optimizer as originally defined by Snow and McFarland, a new role for the optimizer and a new organization for the upper level of die CMU-D A system is proposed. This new role and organization is discussed in the second half of this chapter.

6.1. Synopsis and Conclusions

In the course of this Master's project, a number of related areas in the high level design process were explored. An software package was implemented to perform transformations at the Value Trace behavioral level. A set of metrics to evaluate the results of these transformations on the performance of die final design were designed and implemented. These transformations and metrics were written in the C programming language [Kernighan 78], and comprise some 6000 lines of code. After these packages were written, a number of design examples were chosen and evaluated in an attempt to more fully characterize the effects of the transformations on the design in progress. The feasibility of automatically guiding the transformation process was explored, and a several semi-automatic and automatic guides implemented

The transformation package which was constructed implemented the transformations described in Chapter 2 of this report, most of which were described earlier by Snow in his Ph.D. thesis [Snow 78]. This transformation package was an interactive, menu-driven system which fits into the framework for the CMU-DA system described in [Gatcnby 81] as a separate function, and comprises the bulk of the code written for this project In some cases, changes which have been made to the basic structure of the Value Trace since it was defined by Snow forced the rc-cvaluation of the transformations.

A set of metrics was developed to perform the evaluations described in Chapter 3. In choosing these metrics, the constraints on the individual elements of the CMU-DA system were examined, and those appropriate for being varied to explore the results of die transformations were determined. In order to measure changes in the data part of die design, metrics were designed and implemented to estimate those parameters required by Leive's predictors [Leive 81] to estimate the speed, cost, and power consumption of the final design. To measure changes in the control part of the design, two different types of microcontrollers were examined, and examples in subsequent chapters were evaluated for each of these controllers. For those metrics which were not already computed by the CMU-DA system, commands were added to the transformation mode package to do so.

Some work was also done in exploring the feasibility of automatically or semi-automatically guiding the transformation process. Since part of this process included the gathering of data from die control step allocation and the data path allocator, the problems involved with an iterative transformation and allocation process were explored, and it was determined that, with the current view of die system, this iterative process did not work. Without this infonnation there was no longer a great deal of infonnation to guide the transformation process, but a small number of commands were implemented based on the infonnation available at the unallocated Value Trace leveL

In addition to the coding and implementation, the following new infonnation was discovered in the course of this project:

- SELECT factoring has many negative side-effects, none of which were catalogued by Snow.
- The desirability of SELECT factoring/motion depends largely on the data and control style of the design, especially the latter.
- Inline expansion can be used to reduce biases imposed on the design by the designer's coding style, and to reduce the amount of delay in the design.
- Loop unwinding can also be done through inline expansion, and can be converted from this form into Snow's form through vtbody formation.
- The results of the types of transformations implemented in this project are similar in range to the earlier work by Barbacci and Siewiorek.

- In many cases, it is impossible to preserve control step and data path allocation data during the transformation process, especially if it is desired to remain consistent with the allocation algorithms.

- If the iterative transformation/allocation process is not allowed because of the preceding point, there is no control step or data path information available at the time of transformation to guide the transformation process.

6.2. Future Work

In die course of this Master's project, new information has been discovered about the relationship between the transformations at the Value Trace level and the other levels in the CMU-DA system, particularly the allocation levels. The lack of information from these other levels has made the guidance of the transformations more difficult that previously imagined, and has raised questions about the role of the Global Optimizer in the CMU-DA environment Previous views on the role of the Global Optimizer are discussed in the next section, and are followed by a proposal for a new view of its role and a new organization of the upper level of the CMU-DA system.

6.2.1. Views on the Global Optimizer

In the past, it has been envisioned that one of the elements of the upper level of die CMU-DA system would be a *Global Optimizer*. This Global Optimizer would be responsible for the ISP-to-VT translation, as well as die execution of any optimizing transformations. It was originally conceived by Snow [Snow 78], and its role was later expanded by McFariand [McFarland 78]. It was this definition of the Global Optimizer which was used as the basis for this project

6.2.1.1. Snow's **Views**

In his Ph.D. thesis [Snow 78], Snow envisioned the upper level of the CMU-DA system existing as shown in Figure 6-1. The major component at this level is die *Global Optimizer,* which he saw consisting of both an ISP-to-VT translator and a package to perform die transformations which he defined. The output of this component was a Value Trace which was optimal with respect to die designer's criteria. It was only *after* this transformation was performed that control steps were assigned, and data path allocation started. With this straight line of execution, it was not possible for the Global Optimizer to properly evaluate die applicability of those transformations which were conditional in nature (Le., those which involve a tradeoff between two performance factors). Snow's solution to this problem was to always perform the transformations, even if the transformation might add extra operators or control steps to the design. He expected the partitioncr and data path allocator to detect this anomaly and inform the global optimizer so that it "may repair the defect and flag it to avoid redoing the damage in the future."

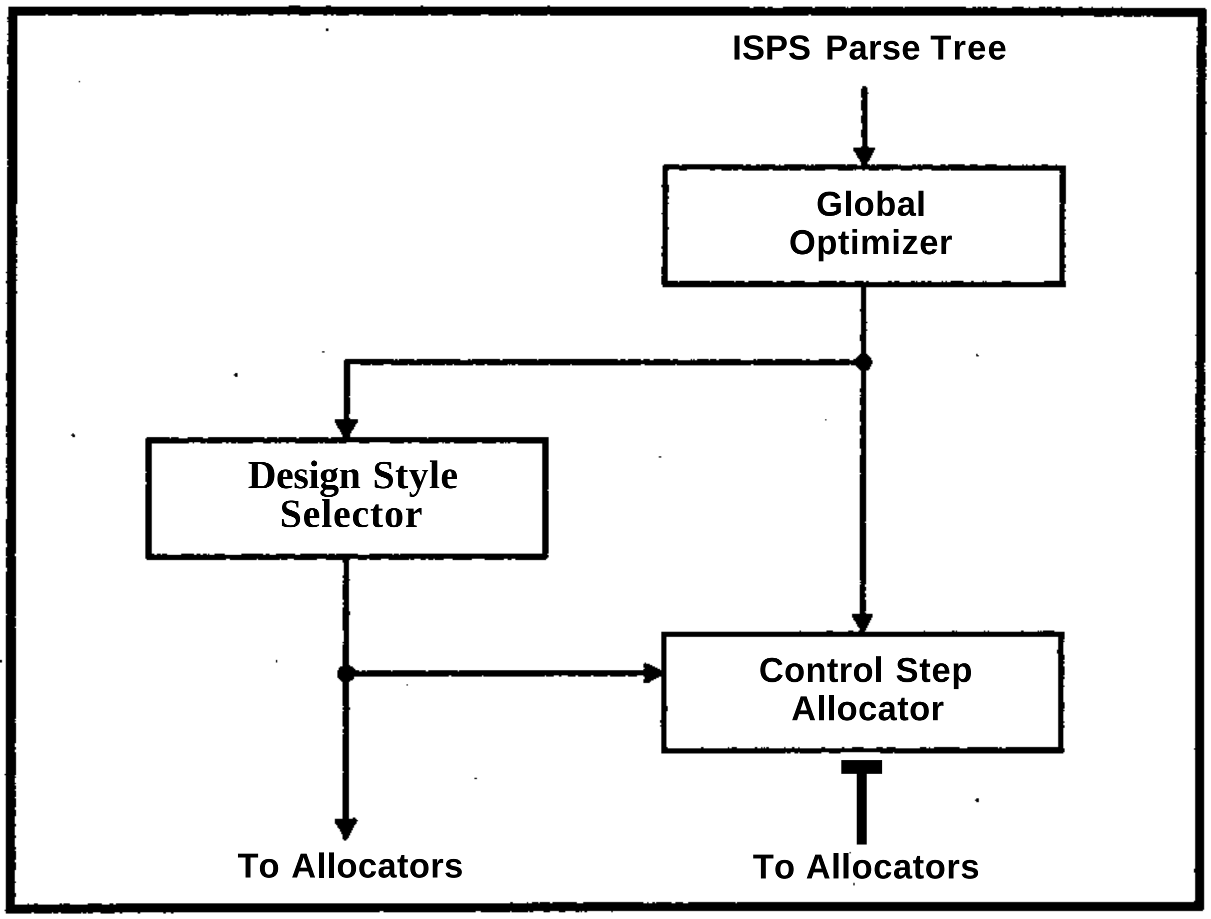

Figure 6-1: The Upper Level of the CMU-DA System as Envisioned by Snow

6.2.1.1 McFariand's Views

In his Master's thesis [McFarland 78], McFarland envisioned the top level of the CMU-DA system existing as shown in Figure 6-1 With his addition of control information to the Value Trace, it became possible to assign control steps directly to the VT, rather than having to do so on some parallel structure. This allowed the control flow shown in the figure, with the unconditional transformations performed, the Value Trace assigned control steps, and the control step information used to provide information to the optimizer and guide it in the application of the conditional transformations[6]. In order to allow this guidance of the conditional transformations, McFarland suggested that the Global Optimizer perform "preliminary" control step allocation, which would be "improved by later stages of the design."

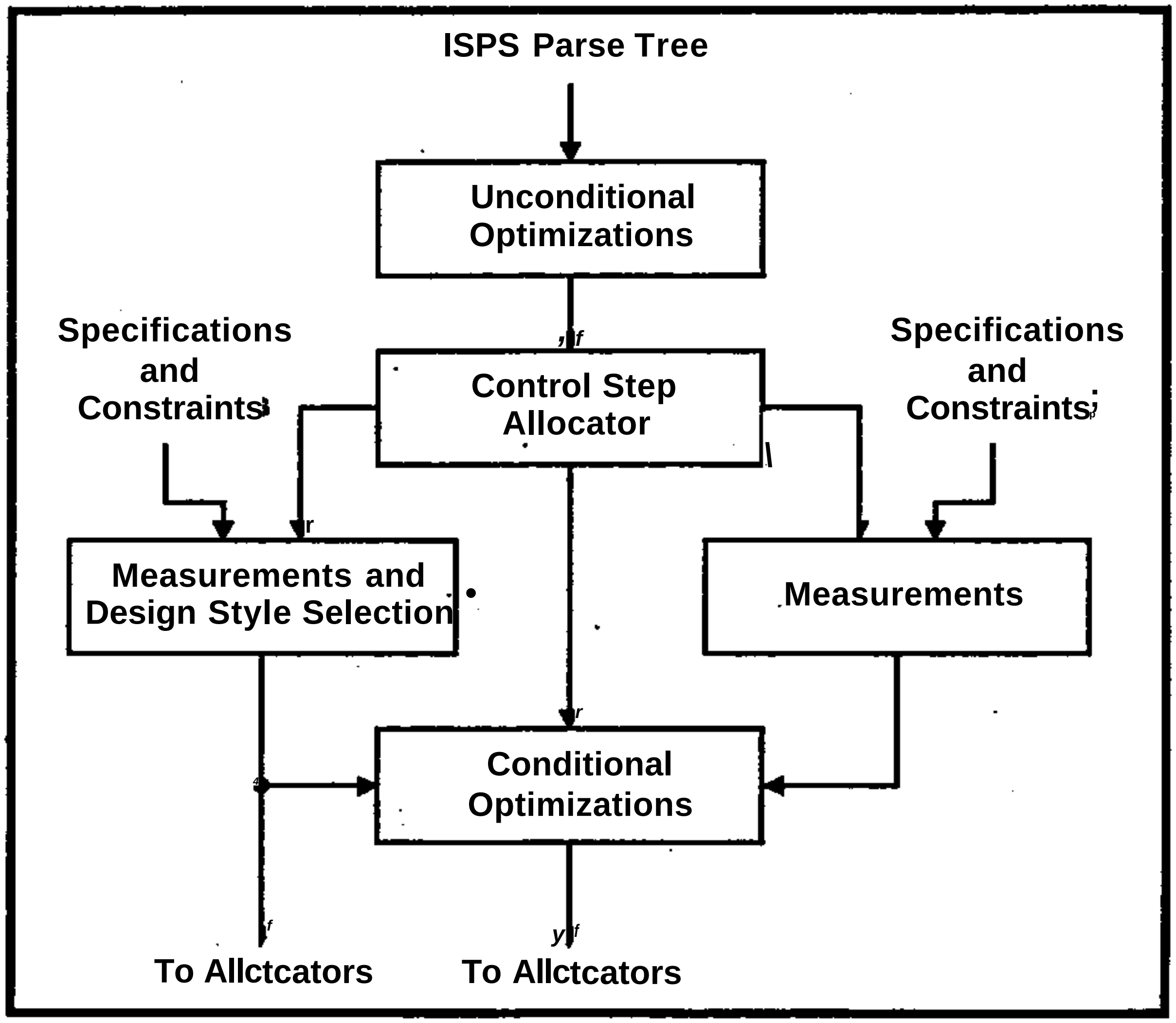
ISPS Parse Tree
Unconditional Optimizations
Specifications and Constraints
Control Step Allocator
Specifications and Constraints
Measurements and Design Style Selection
Measurements
Conditional Optimizations
To Allctcators
To Allctcators

Figure 6-2: The Upper Level of die CMU-DA System as Envisioned **by McFarland**

[6]This is essentially the state of the CMU-DA system at the present, although not all of the guidance of the conditional transformations is mKunatif.

6.2.2. Should our View of the Global Optimizer be Changed?

6.111. The Iterative Design Process

In the CMU-DA system, a desirable characteristic, and one which is currently unavailable, is the ability to perform fully iterative design. With the current system it is not possible to move at will between the five functions of design style selection, optimization, metrics, control step allocation, and data path allocation. The present assumption is that these functions will be used in a mostly predefined order (with the exception of metrics), and there is no feedback mechanism to allow for iterations in the design.

If such a fully iterative design system could be developed, it would have a number of advantages. These advantages would include the sharing of information between levels, and better exploration of the design space.

Each of the functions of die system could possibly benefit from information obtained at other levels. As seen before, it would be very desirable for the transformation package to have access to control step information, as it could much better guide the transformation process if this information was available. It might also be possible for the design style selector to use information from die allocation process to vary the design style, and it might be possible for the control step allocator and data path allocator to work together in an iterative fashion to explore serial-parallel tradeoffs similar to those examined by Barbacci and Siewiorek.

In general, such a fully iterative system should allow for better exploration of the design space. In the current system, decisions made at one level limit those made at lower levels, (e.g., design style selection limits die choices made by the data path allocator). With the fully iterative system, this would no longer true, as there would be a "tree" of decisions, with decisions made at an upper level simply affecting Oat branch, and alternate choices being represented by die other branches. This method of exploration, since it more fully covers the design space, should also allow the better realization of any constraints imposed upon the system by die designer.

6222. A Proposal for the CMU-DA System

A proposal for a new view of die Global Optimizer, and organization of die upper level of die CMU-DA system as a whole, is shown in Figure 6-3. With diis organization, die five packages shown (design style selection, optimization, metrics, control step allocation, data path allocation) are capable of acting in a related but independent manner. Each package may require information provided by another package, as in the case of the optimization/transformation package requiring control step allocation information, and each package should be consistent in its treatment of the effects of the other packages. For example, it is acceptable, and in fact necessary, for the data path allocator to refuse to allocate hardware until control step allocation has been performed. As another example, the optimizer must either refuse to do transformations once control step and data path allocation is performed, or some method must be found to handle the problem of optimizing an allocated design without changing the allocation strategy.

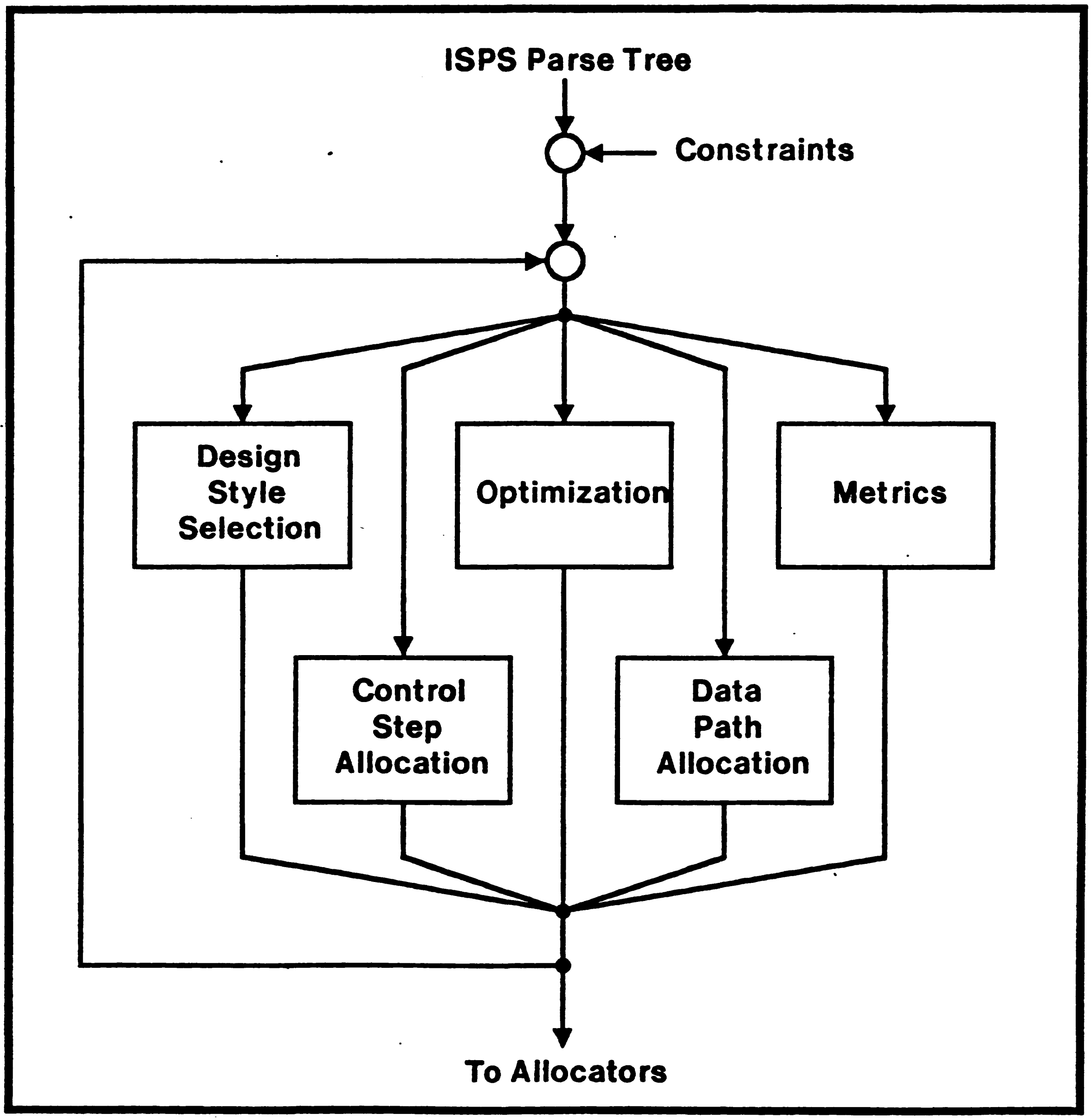

Figure 6-3: Proposed New Organization for the Upper Level of the CMUA-DA System

6.123. Advantages of this Proposal

A major advantage to this proposed organization of the CMU-DA system would be the availability of data for the guidance of the transformations. As was discovered in die course of this Master's project, guides for the optimizaton process need information both from the control step and data path allocators. Since this information (particularly the data path allocation information) is not available with the current organization, the guides are limited to using information from the unallocated Value Trace, and their usefulness is limited. If both the control step and data path information were available, a wider variety of guides could become available. With the data path allocation information, Leive's predictors could be applied, and transformations could be chosen with a particular performance objective in mind, as in increased throughput or lower cost With the control step allocation information, the feasibility of applying such conditional transformations as SELECT factoring/motion could be predicted.

«

Another advantage to this proposal would be the potential for exploring serial-parallel tradeoffs similar to those explored by Barbacci and Siewiorek [Barbacci 75]. While this work was concentrated on one very stylized form of data-path allocation, work might be done exploring the tradeoffs involved with other types of allocators. As a result of this exploration, the different types of allocation might be more fully characterized, and some general conclusions on serial-parallel tradeoffs derived. With both this type of serial-parallel transformation and the transformations implemented in this Master's project, the design space could be more fully explored, and better improvements in performance might result

In the present generation of the CMU-DA system, there does not exist an automatic design style selector, or a data path allocator capable of allocation in a wide variety of styles, but work was done on this problem in an earlier generation of the CMU-DA system. Thomas discussed the concept of the Design Style Selector in his Ph.D. thesis [Thomas 77], and identified methods for choosing the design style. Lawson [Lawson 78] implemented this system as a Master's project, and added an assumed control style for each data style. To implement this sort of Design Style Selector in die CMU-DA system would require an iterative design system, as the information required by the selector must come from a number of different areas. This information includes:

- External constraints imposed on the performance and physical characteristics of the design by the designer.

- Database information on the performance characteristics inherent in each design style.

- Measurements made from the VT level, including such factors as degree of parallelism and intcrconncctivity, number of registers, etc. The first factor requires control step information, and the second requires data path allocation information, so if this information is to be used by the design style selector, a fully iterative design system must exist

6.114. Problems with this Proposal

There still exists die problem discussed in Section 5.11.1, of trying to preserve allocation information during die transformation process. In a fully iterative system, it must be possible to perform transfonnations at any point in the allocation process, so some solution to this problem must be found. This solution might take'on one of the following forms:

- Both control step allocation and data path allocation might be completely redone after every session of optimizing.

- The optimizer might signal each allocator when it is doing something to render that allocator's work invalid, and that allocator can then be called after the transformation session to completely redo its allocation.

- The allocators might be made "smarter", and called as above when their work is rendered invalid, but only redoing the portions of the allocation affected by the transformations.

In order to determine which of these solutions to apply, some work needs to be done exploring die feasibility of the third choice. If this choice is indeed possible to implement, it would be die most efficient of alL If not, the first or second choice could be implemented, but might use substantially more computation time.

6.3. Conclusion

A concise summary of the work performed as part of this Master's project was provided. Since, in die course of this project, views on die rede of die Global Optimizer have changed, the original views on this component were presented, and a new view of die upper level of the CMU-DA system proposed.