A META-COMPILER AS A DESIGN AUTOMATION TOOL

R. MANDELL
G. ESTRIN
Dept. of Engineering
U.C.L.A.
Los Angeles, Calif.

A META-COMPILER AS A DESIGN AUTOMATION TOOL

R. Mandell
G. Estrin
Department of Engineering
University of California
Los Angeles

*This research was sponsored in part by the Information Systems Branch, Office of Naval Research (Nonr-233(52)) and the Atomic Energy Commission (AT (11-1) - Gen 10, Project 14).

A META-COMPILER AS A DESIGN AUTOMATION TOOL

by R. Mandell

G. Estrin

Introduction

In the field of design automation, in general and in the automatic design of digital computers in particular, it is frequently necessary to translate machine descriptions and designer requests from the input language to the language of the design automation system. This process is very similar to the translation of artificial computer programming languages into machine language. Hence, it is reasonable to attempt to apply the techniques developed for writing compilers to the task of translation required within a design automation system. In this paper we will describe the use of one such tool, the meta-compiler or syntax directed compiler.[1]

A meta-compiler is a compiler specifically tailored for the generation of other compilers. The input to a meta-compiler consists of a description of the input and output languages of the compiler to be generated and the algorithmic relationship between them. The use of a meta-compiler reduces the generation of compilers from an extremely formidable task of assembly language programming to a more routine task of higher language programming.

The Meta-Compiler in Design Automation

Let us now shift our attention to the design automation process. The first phase of a design automation system [2] must execute the process normally called system design which involves:

(1) Initial selection of a set of memory elements, transformation elements, gated transmission paths, counters, and input-output interface elements.

(2) Tailoring the given computational algorithms to meet the constraints imposed by the selected set of processing elements.

The initial selection permits evaluation of some measure of cost. The assignment and sequencing of processing elements permits evaluation of some measure of performance. The first approximation to the system design may then be perturbed in an attempt to improve the above measures.

We take note of the similarity between the system design process and traditional program compilation. The input to a compiler is a computational algorithm expressed in a language other than the language of the machine on which it is to be executed. The machine is defined by a set of basic operations, an extended operation set from the library of routines and a set of memory elements. The compiler is then forced to follow a set of rules for assignment and sequencing of the operations subject to a memory constraint such that an equivalent algorithm would be executed on the machine.

In our design automation system, generating the system block diagram is equivalent to defining a new machine. For each new machine a "compiler" must be generated in order to do the assignment and sequencing of micro-operations on the new machine. A meta-compiler is a tool for generating compilers and therefore an essential ingredient of an effective design automation system.

We must also consider the fact that the input language to the design automation system is likely to undergo modification with time and application. We therefore choose to define an intermediate language (I.L.) which we seek to hold fixed. Up to this point in our development we have experimented with a simple intermediate language in which each statement has a structure scanned from right to left as follows:

RESULT | OPERATION | OPERAND 1 | OPERAND 2 |

A compiler generated by a meta-compiler has been used to translate from Iverson [5] operations to the intermediate language. The intermediate language serves as a basis for assignment and sequencing of processing elements, which is performed by a second meta-compiler generated compiler.

Now before we move into an example of the system design process, we must introduce the nature of the design automation system library. Most work in computer design automation [3,4] assumes that the designer provides the system block diagram. We allow this but we also establish a system library in which the classes of processing elements described in (1) above are filed. Associated with this system library, we generate a set of conventions which serve to uniquely select the first trial configuration from the intermediate language list. These conventions may be overridden by the system designer and do not act as absolute constraints on the design. The first part of this paper discusses a simple design example to illustrate the above concepts. The second section discusses the structure of a meta-compiler. In order to see the emergence of a system design we are forced to think like a machine. The resulting system design is then compared with a design which was generated from the original algorithm by use of judgement and experience.

A Simple Design Example

We seek to design a machine which contains a table of numbers (M). When the machine is activated it looks at an index or table entry and uses it to find the first positive number following that entry.

Figure 1 gives a modified Iverson programming description [5] of the computational algorithm. A word description follows below with reference to the numbered lines in Figure 1.

Lines 1-5 declare that: both I, the start bit, and the error flag, D, reside in static registers, one bit long which may be manipulated by an external signal; B is a vector of length G bits stored in primary memory; and M is a matrix of 1000 vectors, each of length, G bits, stored in primary memory.

Line 5 Specifies that the global dimension, G, is 36 bits.

Line 7 Is labeled L1 and is a conditional transfer that will cause the system to dwell at L1 until I is set to 1 by some external control.

Line 8 Specifies the transfer of the last 10 bits of B to A. This implicitly defines A as a 10 bit number.

Line 9 Is a conditional branch which compares the first bit of the Ath row of M with 0. If the bit is zero the next line to be executed is the one labeled L2 (line 13). Otherwise line 10 is executed.

Line 10 Diminish A by 1.

Line 11 If A is zero go to L4 otherwise go to the next line.

Line 12 Transfer control to the line labeled L3.

Line 13-15 Store the first 26 bits of the Ath row of M and all of A in B, set the start flip flop to zero and then go to L1.

Line 16-18 Set the start flip flop to zero and set the error flip flop to 1.

Figure 2 shows the intermediate language (I.L.) program which would be generated. The discussion below, along with the syntax in Appendix 1 should enable the reader to follow this step in the process.

```
.DECLARATION        I                F/F          1      EXTERN    1
                    D                F/F          1      EXTERN    2
                    B                MVECTOR G                     3
                    M                MMATRIX G    1000             4
                    GLOBAL DIMENSION              36               5
.END DECLARATION                                                    6
    L1              I:1, ≠ → L1                                     7
                    A ← ω10/B                                       8
    L3              α1/M⊥A : 0, = → L2                              9
                    A ← τ⊥A-1                                      10
                    A:ε̄ (10), = → L4                               11
                    TO TO L3                                       12
    L2              B ← α26/M⊥A, A                                 13
                    I ← 0                                          14
                    GO TO L1                                       15
    L4              I ← 0                                          16
                    D ← 1                                          17
                    GO TO L1                                       18
```

FIGURE 1

Initial Description of the Computer

LINE	RESULT	OPERATION	OPERAND	OPERAND
		BEGIN	EXAMPLE	
1		BDEC		
2	I	F/F	EXT	
3	D	F/F	EXT	
4	M	MEM		
5		ARRAY	G	1000
6	B	MEM		
7		VECTOR	G	

FIGURE 2

Intermediate Language Description of the Machine

LINE	RESULT	OPERATION	OPERAND	OPERAND
8	////	GLOBAL	36	
9		EDEC		
10	L1	LABEL		
11		WAIT	I	TRUE
12	L3	LABEL		
13	$2	FIELD*	.OMEGA (10)	B
14		LOD	A	$2
15	$3	RREAD	M	A
16	$4	FIELD*	.ALPHA (1)	$3
17	$5	COMPR	$4	FALSE
18		EQ	$5	L2
19	$7	COUNTD	A	
20		LOD	A	$7
21	$8	VECON	.NEPSILON(10)	
22	$9	COMPR	A	$8
23		EQ	$9	L4
24		GOTO	L3	
25	L2	LABEL		
26	$10	RREAD	M	A
27	$11	FIELD	.ALPHA (26)	$10
28	$12	CAT	$11	A
29		LOD	B	$12
30		LOD	I	FALSE
31		GOTO	L1	
32	L4	LABEL		
33		LOD	I	FALSE
34		LOD	D	TRUE
35		GOTO	L1	

FIGURE 2 - Intermediate Language Description of the Machine (Cont.)

Lines 1-9 result from the declaration section of the input description; line 1 and 9 mark the limits of this section. Line 8 defines the global dimension, G, as 36. Lines 2 & 3 declare I and D to be flip flops which are available to external controls. Lines 4 and 5 declare M to be a matrix stored in memory having a dimension of 36 x 1000. B is defined by lines 6 & 7 to be a 36 bit vector stored in memory.

Lines 10 & 11 together have the same meaning as line 7 of Fig. 1. The instruction L1 LABEL means that whenever control is transferred to L1, the line immediately following the LABEL instruction (Line 11) will be executed.

Lines 12-14 are the I.L. translation of line 8 of the source code. A FIELD* instruction gives a name, in this case $2, to a subfield of the vector specified by the second operand. .OMEGA (10) is a translation of ω^{10}.

Lines 15-18 have the same meaning as line 9 of the source code. Line 15 (row read) means obtain the Ath row of M and call it $3.

Lines 19-20 have the same meaning as line 10 of the source code. COUNTD A means count A down by 1.

Lines 21-24 have the same meaning as lines 11 and 12 of the source code. Line 21 defines vector constant $8 as $\bar{\epsilon}(10)$. .NEPSILON (10) is the coded form $\bar{\epsilon}(10)$.

Lines 24-28 are translated from line 13. RREAD obtains the Ath row of M and gives it the name $10. Line 27 gives the name, $11, to the first 26 bits of $10 and line 27 gives the name $12 to the catenation $11 and A.

Lines 29-35 are a direct translation at lines 14-18.

We now consider the generation of the configuration from the intermediate language. The flow of the process shown in Figure 3, has been completely specified, but not programmed, and in order to test it before we generate the actual computer code, we have gone through the operations by hand for the simple

design example presented in the previous sections. In particular this has helped us in formulation of conventions necessary to access a library and produce an initial design.

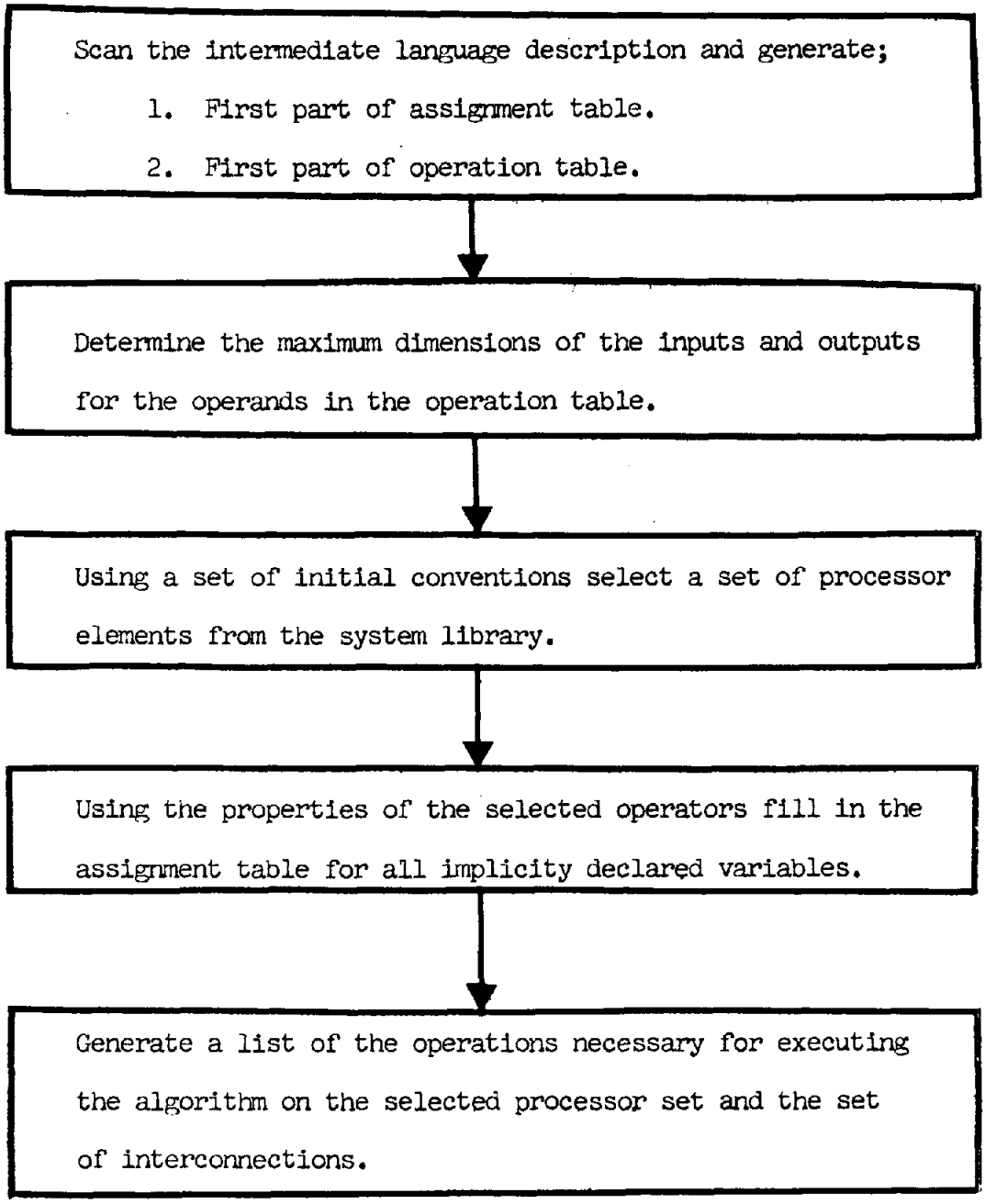

FIGURE 3

The first block of the flow diagram builds up the assignment table Fig. 4 showing the size of all operands and the location of all declared operands. The size of all undeclared operands is deduced from the nature of the operations which produce them. For instance, line 13 of Fig. 2 specifies that the result called $2 will have 10 bits. The next operation LOD A $2 specifies that the operand A will be loaded from result $2. Hence, A is assigned a length of 10. The length of the remainder of the operands can be deduced in the same way.

FIGURE 4
ASSIGNMENT TABLE

1 NAME	2 DECLR. TYPE E EXPLICIT I IMPLICIT VC CONSTANT	3 FIRST DIMENSION	4 SECOND DIMENSION	5* STORAGE TYPE	6* LOCATION AFTER READ	7* SOURCE FOR STORE	8* POINTER REGISTER	9* MEMORY BLOCK	10* ADDRESS
I	E	1		F/F					
D	E	1		F/F					
B	E	36		MEM	MR	MR	MA		1001
M	E	36	1000	MEM	MR	MR	MA	0	
$ 2	I	10		REG	MR(1,10)	MR(1,10)			
A	I	10		COUNTR	CA	CA			
$ 3	I	36		REG	MR	MR			
$ 4	I	1		REG	MR(1)	MR			
$ 5	I	1		BZD	BZD				
$ 7	I	10		COUNTR	CA	CA			
$ 8	VC			WZD					
$ 9	I	1		WZD	WZD				
$10	I	10		REG	MR				
$11	I	26		REG	MR(1,26)				
$12	I	36		CAT	$11	1,26			
				CAT	A	27,36			

* Filled in by Block #4, Figure 4

While the assignment table is being filled with the list of operands, a second list of operations is being built along with each set of operands used by it. Since RREAD is used twice in the I.L. description RREAD would appear in the list with two sets of operands. In the case of the example, they happen to be the same. However, this will not generally be the case. The completed list is shown in Figure 5 although the last column is not filled until block 3 of Figure 3.

OPERATION	OPERAND SET	MODIFIER	ASSOCIATED WITH
RREAD	M A		MEMORY
	M A		
COUNTD	A		COUNTER
COMPR	MR(1) 0	.EQ	BZD
	A 0	.EQ	WZD
MEMORY	B		MEMORY
	M		

FIGURE 5

We now use a series of conventions in order to determine what hardware processors to employ in the first trial design. Any of these conventions may be overridden by explicit declaration. These conventions are merely default starting points and are not constraints. The conventions used in the example were:

1) There shall only be one of each type of processor in the initial system unless more are declared.
2) 1 memory is assumed to be in the system.

3) Exclusive of memory (specified by convention 2) the only storage elements in the system will be those which are explicitly declared or are integral parts of selected processors.

4) Processing units will have their own input and output registers as integral parts of their design.

5) The system library specifies the preferred processor corresponding to each intermediate language operation depending on the dimensions of inputs and outputs of the operation. For example, a different counter may be preferred for a 3 bit count operation than is preferred for a 20 bit count operation.

6) Matrices will be assigned to memory in blocks of predefined size in order to save an index adder.

7) The system will have a wired program so that no instruction fetch cycles are necessary.

Using this set of conventions, the list of dimensions in the assignment table, and the system library, the collection of processing elements in Figure 6 is arrived at.

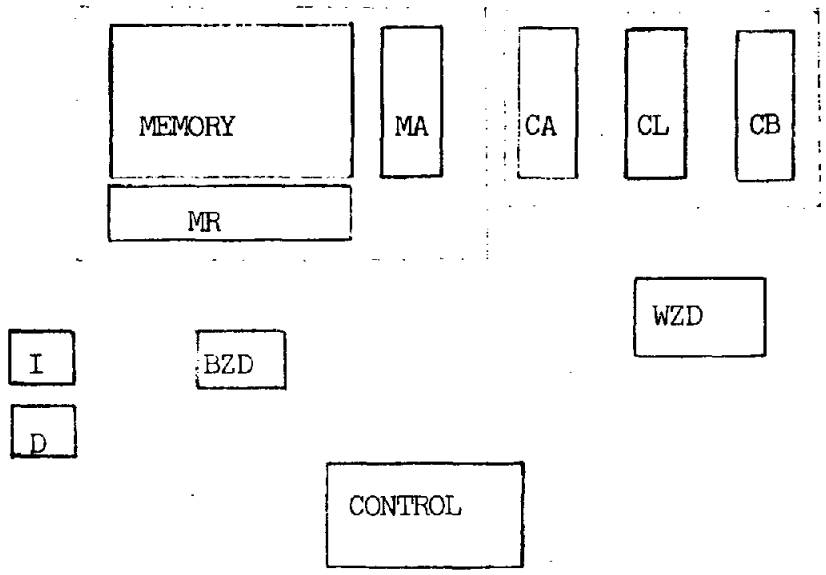

FIGURE 6
INITIAL SYSTEM DIAGRAM

The fact that a memory, to be used in the system, has an input/output register called MR and a pointer register called MA is obtained from the system library. Similarly the fact that a counter consists of an input/output register (CA), a logic network (CL) and a secondary rank register (CB) comes from the library. BZC is a single bit zero detector and WZC is a 10 bit word zero detector.

While selecting the set of processing elements, the final column of the operation table (Figure 4) was filled in. RREAD did not require any unique processing element since it only occurs in association with the array M which may be stored for row reading. Had both row and column reading been specified for the array M, additional equipment or an additional reordered copy of M would have been necessary.

The library also specifies the sequence of operations necessary for loading, reading and operating each element. For example for the counter the read sequence is empty since the information is always available in CA. The load sequence is

GATE <SOURCE> TO CA delay 20 μsec.

and the operate sequence is

GATE CL TO CB delay 40 μsec.
GATE CB TO CA delay 20 μsec.

This information will be employed in generating the sequence of operations necessary to execute the algorithm on the selected hardware set.

Before the new sequence of operations is specified the remaining columns of the assignment table (Figure 4) must be filled in. It should be noted that the variable A (Fig. 5) was assigned to the register (CA) because the counter is

used only for manipulating the variable A (Fig. 5).

All operands that are assigned to the memory are available at the memory data register after a read sequence. Hence, all subfields of words stored in memory are assigned to subfields of the memory data register. If there were other undeclared named variables besides A, these would be assigned to storages in the memory.

The next task of the configuration generation program is to generate a sequence of operations which will be performed in order to execute the desired algorithm on the selected set of processors. This sequence will be specified in a form very similar to Gorman's design table [3]. Figure 7 shows the translation between the intermediate language and this form.

In performing the translation the translator uses the assignment table to find out what device holds the operands. It then consults the library to determine what read or load sequence must be included in the new version of the algorithm. Suppose for example, the command LOD A B appears in the I.L. description of the algorithm. The LOD command has the following meaning: execute the read sequence for the second operand (B) followed by the load sequence for the first operand (A). From the assignment table we would find that the variable B is in memory at address 1001 and that after the read sequence for memory is finished, B will be in MR. The read sequence for the memory that is in the system configuration is

```
GATE  <ADDRESS>  MA    DELAY  40
GATE  TRUE       MEM1  DELAY  2000  REPLY MEM3
```

The constant 1001 is substituted for <ADDRESS> in the read sequence and the sequence is inserted in the sequence of instructions. The location of A is then found in the assignment table and its load sequence is looked up. MR is substituted for <SOURCE> in the load sequence for A, and the resulting sequence is inserted in the code being generated.

The operation GATE in column 3 of Fig. 7 means gate the operand in column 4 to the location in column 5. BRC is a conditional branch instruction. It means transfer control to the instruction whose label is in column 4 if the bit whose name is in column 5 is true. Otherwise execute the next sequential instruction. SET means set the flip flop named in column 4 to the value given in column 5. Column 6 is used to indicate that two or more operations must occur in parallel, and column 7 gives the time required for each operation. If an operation generates a completion signal, its name is placed in column 8.

FIGURE 7

Sequence of operations for performing the algorithm shown in Figure 1 on the system shown in Figure 5.

1 STEP	2 LABEL	3 OPERATION	4 OPERAND	5 OPERAND	6 PARALLEL WITH	7 DELAY	8 REPLY
1	L1	LABEL					
2		WAIT	I	1			
3	L3	LABEL					
4	L3	GATE	1001_{10}	MA		20	
5		GATE	TRUE	MEM1		2000	MEM3
6		GATE	MR(27,36)	CA		20	
7		GATE	CA	MA		20	
8		GATE	TRUE	MEM1		2000	MEM3
9		GATE	MR(1)	BZDI		20	
10		BRC	L2	BZD		40	
11		GATE	CL	CB		20	
12		GATE	CB	CA		20	
13		GATE	CA	WZDI		30	
14		BRC	L4	WZD		20	
15		BRU	L3			40	
16	L2	LABEL					
17		GATE	CA	MA(1,10)		20	
18		GATE	TRUE	MEM1		2000	MEM3
19		GATE	CA	MR(27,36)		20	
20		GATE	1001_{10}	MA	19	20	
21		GATE	TRUE	MEM3		2000	
22		SET	I	FALSE		20	
23		BRU	L1			40	
24	L4	LABEL				20	
25		SET	I	FALSE		20	
26		SET	D	TRUE			
27		BRU	L1			20	

The completed design including transfer paths deduced from Figure 7 is shown in Fig. 8a. Fig. 8b shows the system generated by a designer using traditional methods and judgement.

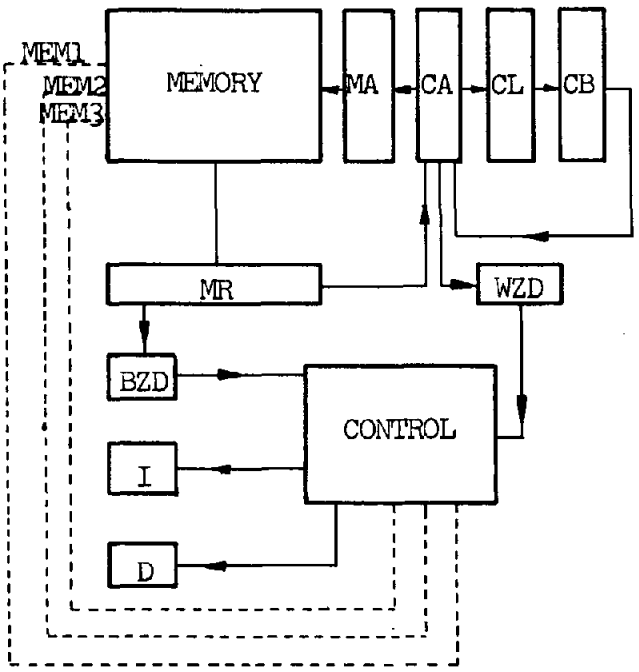

FIGURE 8a

The System Designed According to the Design Algorithm

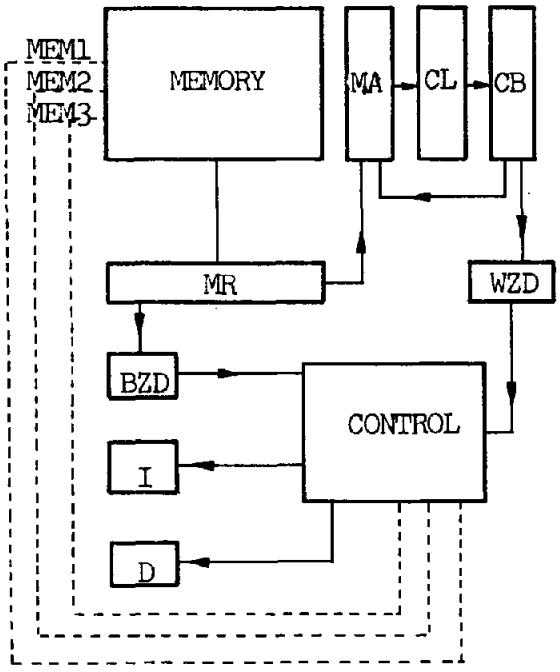

FIGURE 8b

The System Designed by a Designer

The designer recognized that MA could be used as the primary rank of the counter, in place of CA, and that faster operation could be achieved by comparing for zero at the secondary rank of the counter. The designer also recognized that the memory access at lines 16 to 18 of Fig. 7 is unnecessary because the information is already in MR.

The differences in the designs shown, Figure 8, point up optimizations that will be achieved during a timing sequence analysis of the code in Figure 7. This phase of the design will not be discussed in this paper.

THE META III Meta-Compiler

The second section of this paper will discuss a meta-compiler which is in essence of the META III meta-compiler written by Fred Schneider and Glen Johnsen [1]. Several instructions were added to the META III language in order to make the meta-compiler a more useful tool in design automation. The extended meta-compiler has been used to construct a compiler which translates a subset of the Iverson language into the intermediate language described above. Additional table manipulation instructions are planned in order to facilitate the construction of the configuration generator described in the first part of this paper. Figure 9 is a general diagram of the META III program for the IBM 7094.

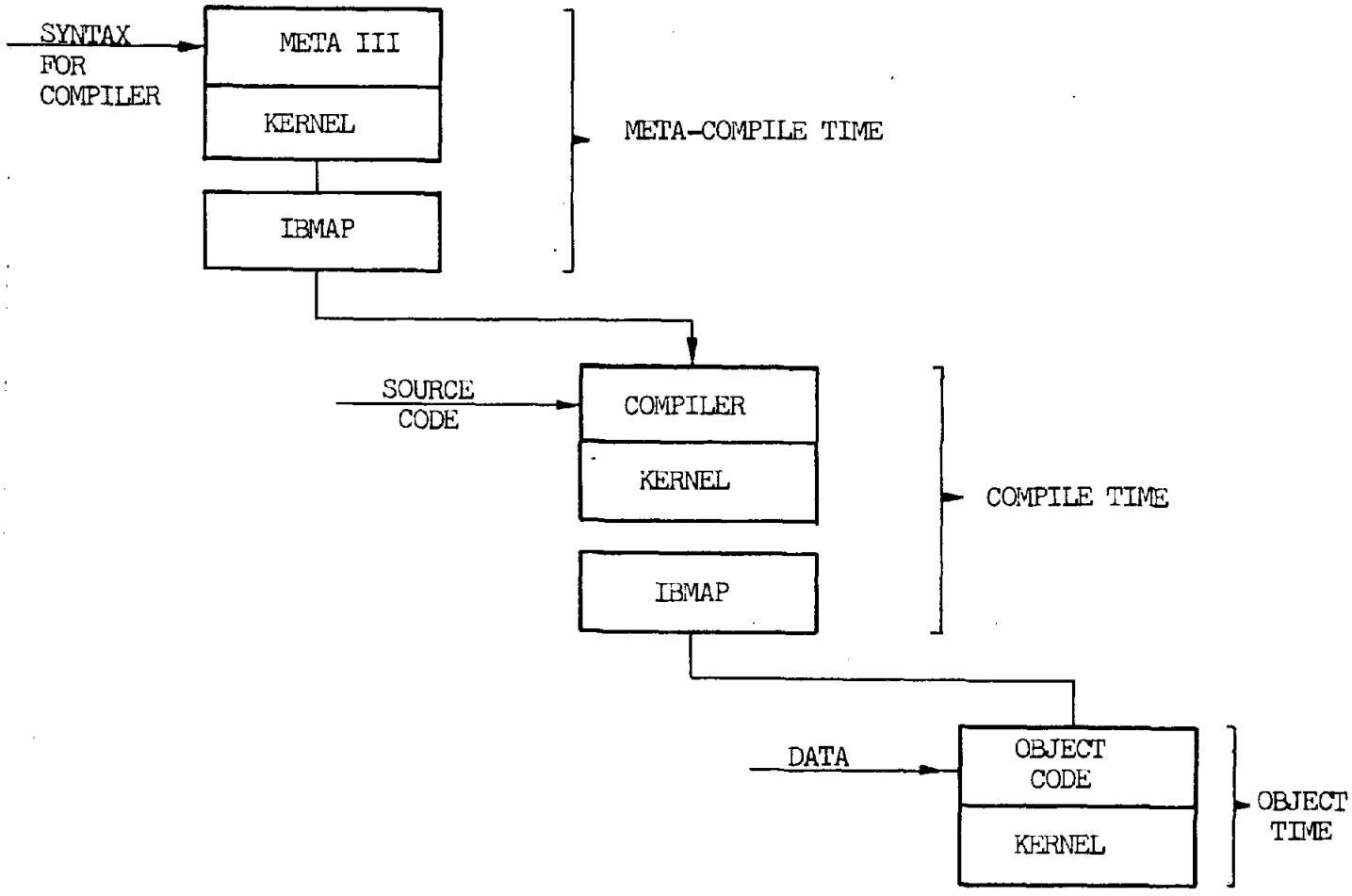

FIGURE 9 The METAIII System

The compiler to be generated is described by a group of syntax equations, coded in the meta language described below. These are translated into MAP by META III. The MAP code is then assembled by IBMAP. The resulting program is a compiler which accepts source code and translates it into MAP. When this is assembled by IBMAP it becomes the object program.

The kernal of the meta-compiler is a set of service routines, written in MAP, which may be used by the meta-compiler, the compiler and the generated object program.

Fortunately, the meta-compiler itself has been described in meta-language. Hence, the meta-language itself may be modified by a process of boot-strapping in which the syntax description of the modified meta-compiler is produced. This often entails the hand generation of a new service routine to be added to the kernel.

The meta-compiler may be viewed as a simulated computer along with a program. The simulated computer will be called the meta-machine. The meta-machine is simulated by means of the subroutines in the kernel. Any attempt to transplant the meta-compiler to a new machine would involve constructing a new simulator for the meta-machine.

As shown in the block diagram of Figure 10, the meta-machine consists of a number of arrays of registers which function as push down stacks. Each array has a pointer register which points to the cell which is currently functioning as the top of the stack. The pointer is moved up or down by incrementing or decrementing the contents of the pointer register. When a piece of data is added to the stack, the pointer is moved up by 1.

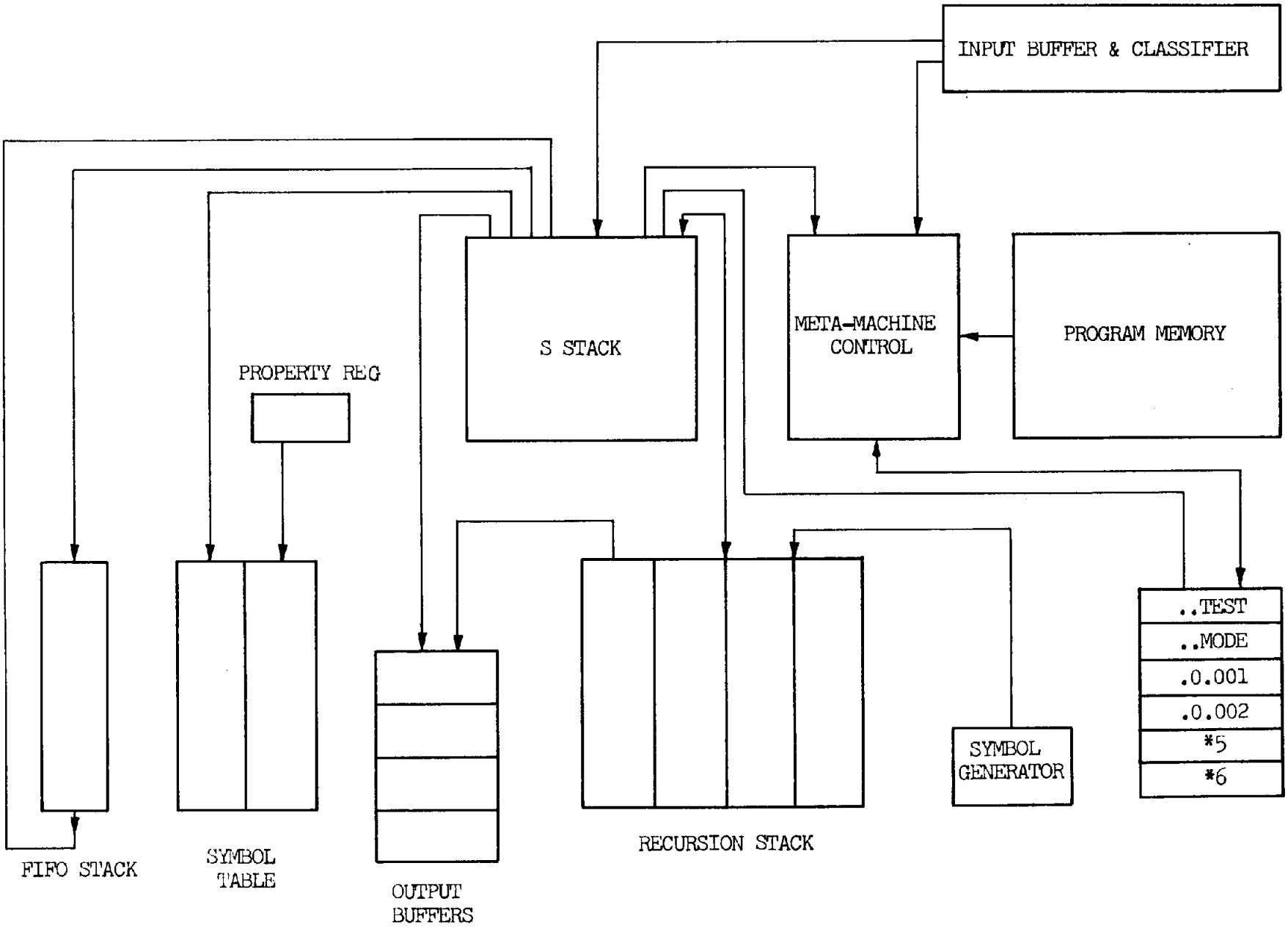

FIGURE 10 The Meta-Machine

Programs (syntactic equations at meta-compile time, source code at compile time or, data at object time) enter the machine through the input buffer and classifier, (IBC) where they are treated as a continuous string of characters. In the course of the analysis of these programs, groups of characters at the head of this string are classified and moved from the IBC to the various arrays in the meta-machine. After a group of characters is removed from the IBC, all blanks following the group are deleted and the input string is advanced until a nonblank character is at the front of the buffer.

The Meta Language

The META III language is built around a special coding of the Backus Normal Form (BNF) [6], which is a language for describing the syntax of programming languages. To this syntactic language is added a set of imperative instructions which manipulate or test the information in the various parts of the meta-machine. For example, consider the BNF equation (1) which defines a <STATEMENT> in terms of other syntactic entities.

<STATEMENT>=<LABEL>..<STATEMENT>|<ARITHMETIC STATEMENT>|<CONTROL STATEMENT> (1)

This equation, called a syntactic equation, has the following meaning: A <STATEMENT> is equivalent to a <LABEL> followed by two periods, followed by a <STATEMENT> or a <STATEMENT> is an <ARITHMETIC STATEMENT> or a <CONTROL STATEMENT>. Equation (1) translates to the META III language equation

```
STATEMENT = LABEL'..'STATEMENT
          / ARITHMETICSTATEMENT                    (2)
          / CONTROLSTATEMENT.,
```

As shown in Equation (2) BNF quantities which are enclosed in brackets are translated to strings of capital letters or numbers with no imbedded blanks.

Such a string must be headed by a letter and followed by a blank or a symbol which is neither a letter or a number. This type of string is called an identifier. A sequence of symbols in the BNF equation which is not enclosed in brackets is translated to the identical string enclosed in apostrophes (')* and the BNF symbol | (meaning exclusive or) is translated to the right-leaning slash (/). An equation in the meta-language is always terminated by the symbol (.,). Identifiers such as LABEL, STATEMENT, and CONTROLSTATEMENT must be defined either by a syntactic equation (i.e. appear on the left of an equal sign) or must be an operation in the meta-language (Tables 1, 2 & 3). Though an identifier may be made up of any number of characters, only the first six characters (if there are more than 6 characters are truncated to six characters for internal representation). Thus, the identifiers ARITHM and ARITHMETICSTATEMENT are considered to be identical.

When an identifier appears in a syntactic equation it has the following meaning. At compile time examine the state of the meta-machine and see if it matches the state of the machine given in the definition of the identifier. Consider, for example, the definition for LABEL given in equation 3.

$$\text{LABEL} = \text{'.L'}\ (\text{NUMBER/ .ID}).\text{,} \qquad (3)$$

Equation (3) means,determine if the first two characters at the head of the IBC are .L. If they are, remove them and see if the next group of characters is either a number or an identifier. If the first characters are not .L, the machine is not in the state required in order to meet the definition of LABEL. If the string .L is found, but is not followed by a number or an identifier, processing cannot continue because the state of the machine has been changed by removing .L from the IBC. In this case an error message is printed and compilation

---

*The pair of parentheses is not a part of the symbol

is resumed at the end of the statement that was in error. Notice that there are two alternatives for the second part of the definition of LABEL in equation (3) and that the equation could have been written

```
LABEL = '.L' LAB1 .,
LAB1  = NUMBER/ .ID .,                    (4)
```

The use of parentheses in Eq. (3) saves the necessity for defining the additional syntactic variable LAB1.

The strings '.L' and .ID are examples of a test imperative instruction. There are two types of imperative instructions, tests and actions. A test imperative tests the status of the meta-machine at compile time and may cause data to be transfered from one register to another if the test condition is met. An action imperative merely moves data between registers or produces output. The test imperatives are defined in Table 1 of the Appendix and the action imperatives are defined in Tables 2 and 3 of the Appendix. The third column of Table 1 indicates what action is taken if the question in column 2 is answered in the affirmative.

The action imperatives include a pair of imperatives which move information to the output buffers and cause output to be generated. These imperatives are .OUT (LIST) and .W(LIST). LIST may be any sequence of the imperatives in Table 3. After the last imperative in LIST is acted upon,the contents of the buffers are output and the buffers are cleared. .OUT causes MAP code to be generated; the LABEL field comes from buffer 1, the OPERATION field comes from buffer 2, and the VARIABLE field comes from buffers 3 and 4. .W causes the contents of all four buffers to be written out as one line of text, 120 characters wide, on a separate tape at compile time.

The vectors and matricies used in defining the imperatives are the names of the parts of the META III machine depicted in Figure 10. Additional imperatives which are not necessary for the understanding of the example in the Appendix

may be found in [1].

Another extremely useful element of the META III language is the symbol $. Equation 5 is an example of how this symbol is employed

```
PROGRAM = $(STATEMENTS) .,                    (5)
```

Equation 5 means: A program is a sequence of statements. The sequence may have no members. This is exactly equivalent to equation (6).

```
PROGRAM = STATEMENT PROGRAM/. EMPTY .,        (6)
```

Using the $ notation equation 2 becomes

```
STATEMENT = $(LABEL '..') (ARITHM/CONTRO) .,
```

CONCLUSION

Meta-compilers can play a decisive role in building a design automation system. The first part of this paper showed how a compiler forms the communication link between the designer and the design automation system. A compiler for translating a large subset of a design automation language is shown in the Appendix in order to illustrate how a compiler is expressed in META III language. The first part of the paper also showed how a compiler can be employed to perform actual design work in selecting sets of elementary processors, and translating an input algorithm into a form that is executable on this processor set.

A meta-compiler can also be used as a tool for system integration since it can accomplish the translation which is often necessary between the input-output languages of existing programs. For example, it is a relatively easy task to write a meta-language program for translating the intermediate language described in this paper into the PAT (Personal Array Translator) language [12]. This translated version of the initial algorithm can then be executed on the PAT system to determine whether it actually achieves the desired computational results

Additional uses for meta-compilers in design automation are for simulating logical equations [7] and for translating logical equations into networks of logic circuit modules.

The authors wish to acknowledge the cooperation of the Systems Development Corporation in permitting use of their time sharing system, TSS. This work would have taken many times as long to complete without the rapid turnaround achieved at the terminal.

APPENDIX

The syntax shown in Figure 11 is the META III language version of the intermediate language compiler. The first line of a META III program always begins with the word .SYNTAX followed by the name of the first syntactic equation (PROGRAM in this case) to be executed at the beginning of compile time. The syntactic equations are arranged alphabetically by name for ease of reading. Equation PROGRAM begins on line 1430. The imperatives used in the syntax are defined in Tables 1,2,3 of the Appendix. The reader is guided through a few lines of the program in the following in order to ease his understanding.

The first imperative of PROGRAM (.DIAGNOSTIC SYNTAX (RECOVER)) designates RECOVER (line 1520) as the first syntax equation to be executed after an error is discovered in the source code. The next imperative sets the .MODE word in the communication array to T. This causes the meta-compiler to include certain useful diagnostic traces in the compiler while it is being generated. .ID (line 1440) is the first test imperative executed by the compiler. This looks for an identifier which gives the name of the source code being compiled. When the identifier is found, it is moved from the Input Buffer and Classifier (IBC) to the top of the S stack. The next imperative (*4) moves the identifier from S to the top of the fourth column of the recursion array. Next, the compiler will output a line of column titles via the .W instruction. The commas separating the strings 'RESULT', 'OPERATION' etc. indicate that these strings will be sent to separate output buffers. Each of the four output buffers fills one of the four columns into which the output page is divided. After the column titles are output the I.L. command $0 BEGIN <NAME> is output where the name of the program is substituted for <NAME> by the *4 imperative on line 1450. Next the compiler attempts to find a STATEMENT followed by zero

or more statements. A STATEMENT is defined on lines 1990-2050. A statement may have an optional LABEL followed by either a CONTROL, a ST or a DECLARATION. LABEL, CONTROL, ST and DECLARATION are defined on lines 620, 300, 1780 and 420 respectively.

Figure 12 of the Appendix is a coded version of the program given in Figure 1. Coding was necessary because some of the symbols of the original Iverson description were not available on the key punch. For example, since ⊥ (binary value) is not on a key punch, .BV. is substituted for it. Also since subscripting is not possible, subscripts and superscripts are replaced by bracketed quantities. Thus, $M^{\perp A}$ becomes .MR.M(.BV.A,). The symbol .MR. preceeds M in order to indicate that the symbol M represents a matrix whose elements are single bits. As will be seen from the syntax (lines 940 and 860) this tag preceeds the identifier M into the S stack and is carried with M during most of the compilation process to identify the type of quantity represented by the identifier, M.

```
.SYNTAX PROGRAM                                                          0010
ADOUT=MODTE .PUT'ADD'AOUT                                                0020
.,                                                                       0030
AOUT= *1 *2 .W(.C,*1 *2,*,*)*-2 *-5                                      0040
.,                                                                       0050
ARITHE= TERM$('+' TERM MODTE .PUT 'ADD    ' AOUT                         0060
              /'-' TERM MODTE .PUT 'SUB    ' AOUT)                       0070
.,                                                                       0080
BITPAT=                                                                  0090
        '1' .PUT'.TRUE'                                                  0100
        /'0' .PUT '.FALSE'                                               0110
        /(.SV '.EPS'/.SV '.NEPS')('ILON'/.EMPTY)TAIL                     0120
                                  '(' NUMBER')'                          0130
                    .W(.C,'VECON',...1'ILON',*) .R                       0140
                                                *-5                      0150
.,                                                                       0160
COMOP1=VECON'/' COMOP1/RPRIMARY .,                                       0220
CONCAT=.W(.C,'CONCAT',...2,*).R.R *-5 .EMPTY                             0230
.,                                                                       0240
COMOP=VECON('/' .PUT 'ENDFLD' COMOP1 FIELD / .EMPTY .W(.C,'VECON',...1   0250
                                                      ,*)                0260
                                          .R *-5                         0270
                    )                                                    0280
.,                                                                       0290
CONTROL=                                                                 0300
        (                                                                0310
        '.GO' 'TO' LABEL .W(,'GO TO', 'L' *)                             0320
       /'.COMPR' '(' (REXP'..'REXP                                       0330
                             /ARITHE'..'ARITHE                           0340
                      ).W(.C,'.COMPR',...2,*).R .R .R                    0350
                ')' XFRLIST                                              0360
      /'.WAIT''(' REXP '..'REXP ')' .W(,'WAIT',...2,*) .R .R .R          0370
       )(',,'/.EMPTY)                                                    0380
.,                                                                       0390
COUNT=.W(.C,*,*).R .W(,'LOD',*,*5).R                                     0400
.,                                                                       0410
DECLARATION=                                                             0420
        '.BDEC' .W(,'BDEC')$(.ID DECTYPE                                 0430
                             /'.GLOBAL'NUMBER.W('////','GLOB',*)         0440
                             /'.GLOBAL'NUMBER.W('////','GLOB',*)         0440
                             )'.EDEC' .W(,'EDEC')                        0450
.,                                                                       0460
```

FIGURE 11 The Intermediate Language Translator

```
DECTYPE=                                                                0470
        ('F/F' .W(*,'F/F')                                              0480
        /'BIT' .W(*,'BIT')                                              0490
        /'M' .W(*,'MEM') TYPE                                           0500
        /'MATRIX' NUMB NUMB .W(...2,'MATRIX',...1,*) .R .R              0510
            /'VECTOR' NUMB .W(...1,'VECTOR',*) .R                       0520
     )PROPERTY                                                          0530
.,                                                                      0540
FIELD=*1*2(.TEST'ENDFLD' .W(.C,'FIELD*',...1'('*')',*1)                 0550
            .R *-5                                                      0560
       /.EMPTY .W(.C,'FIELD', ...1'('*')',*1) .R FIELD1)                0570
.,                                                                      0580
FIELD1=.TEST 'ENDFLD' .W(.C,'FIELD*',...1 '('*')',*6).R                 0590
                                                              *-5       0600
         /.EMPTY .W(.C,'FIELD',...1'('*')',*6) .R FIELD1                0610
.,                                                                      0620
LABEL=                                                                  0630
      '.L'(NUMBER/.ID)                                                  0640
.,                                                                      0650
LOD=MODTE.R .W(,'LOD', ...1,*)('.SNAP' SNAP/.EMPTY).R '.,'              0660
.,                                                                      0670
MMODE2=.EQ(...1,...5) .MOVE1 .MOVE 5 .R                                 0680
.,                                                                      0690
MMODE=.EQ(...1,...3) .MOVE 1 .MOVE 3 .R                                 0700
.,                                                                      0710
MOD=.SV'.BV' TAIL                                                       0720
    /.SV'.FV' TAIL                                                      0730
.,                                                                      0740
MOD1= .SV'.BR' TAIL                                                     0750
   /.SV'.FR' TAIL                                                       0760
    .,                                                                  0770
MODT=.EQ(...1,...3) .MOVE 1 .MOVE 3 .R                                  0780
.,                                                                      0790
MODTE=MODT                                                              0800
/...1 NTEST      ...3 NTEST .MOVE 1 .MOVE  3 .R .R .PUT '.BV'           0810
.,                                                                      0820
MPYDIV=$('*'TERM MODTE .PUT 'MPY' AOUT                                  0830
            /'/' TERM MODTE .PUT 'DIV' AOUT)                            0840
.,                                                                      0850
MRMODE=.SV'.MR' TAIL                                                    0860
.,                                                                      0880
```

```
MREXP=                                                                  0890
      MRTERM$('.OR' TAIL MRTERM MMODE *1 .W(.C,'OR',*,*)                0900
                                          *-1 *-5)                      0910
.,                                                                      0920
MRPRIMARY=                                                              0930
         MRMODE .ID                                                     0940
        /'('MREXP')'                                                    0950
            /MRMODE MVPRIMARY .MOVE 2 .R                                0960
        /'.NOT' TAIL MRPRIMARY .W(.C,'.NOTMR',*) *-5                    0970
.,                                                                      0980
MRTERM=                                                                 0990
         MRPRIMARY $('.AND' TAIL MRPRIMARY MMODE *1 .W(.C,'AND',*,*)    1000
                                            *-1 *-5)                    1010
.,                                                                      1020
MVUNSTAK=                                                               1030
*1.W(.C,'D'*1'('...2',',...1')',*,...2).R.R.R*-1 *-5                    1040
.,                                                                      1050
MVEXP=MVTWOP$('+' MVTWOP MMODE *1 .W(.C,'ADD'*1,*,*) *-1*-5             1060
              /'-' MVTWOP MMODE *1 .W(.C,'SUB'*1,*,*) *-1 *-5           1070
              )                                                         1080
.,                                                                      1090
MVMODE= .SV!.MV' TAIL                                                   1100
.,                                                                      1110
MVPRIMARY= MVMODE(MRPRIMARY.MOVE1 .R/.ID)                               1120
         /'(' MVEXP')'                                                  1130
         /'-' MVPRIMARY .W(.C,'.M.MINUS',*) *-5                         1140
.,                                                                      1150
MVSTAK=TWOP MVPRIMARY MVSTAK MMODE2 MVUNSTAK/ .EMPTY                    1160
.,                                                                      1170
MVTWOP=MVPRIMARY MVSTAK                                                 1180
.,                                                                      1190
NTEST=.TEST'N'/.TEST'.BV'/.PUT 'MODERR' RECOVER                         1200
.,                                                                      1210
```

```
NUMB= .SV 'G'                                                                    1220
      / NUMBER                                                                   1230
,                                                                                1240
OP1LST=.SV'.ADD' TAIL                                                            1250
      /.SV'.SUB'TAIL                                                             1260
      /.SV'.ADD' TAIL                                                            1270
      /.SV'.OR' TAIL                                                             1280
,                                                                                1290
OP2LST=.SV'.MPY' TAIL                                                            1300
      /.SV'.AND' TAIL                                                            1310
,                                                                                1320
PLUS=$('+'TERM ADOUT                                                             1330
      /'-'TERM SUOUT                                                             1340
      )                                                                          1350
,                                                                                1360
PRIMARY=MOD RPRIMARY  .MOVE 1 .R                                                 1370
      /NUMBER *4 .PUT 'N' *-4                                                    1380
      /'+' '/' RPRIMARY .W(.C,'+RED',*).R .PUT '.BV' *-5                         1390
      /'-' PRIMARY .W(.C,'MINUS',*) *-5                                          1400
      / '(' ARITHE ')'                                                           1410
,                                                                                1420
PROGRAM=.DIAGNOSTIC SYNTAX(RECOVER)                                              1430
       .MODE T .ID *4 .W(  'RESULT'   ,'OPERATION','OPERAND ONE',                1440
                        'OPERAND TWO')  .W(.C,'BEGIN',*4)                        1450
            STATEMENT $(STATEMENT)                                               1460
,                                                                                1470
PROPERTY=                                                                        1480
      $('EXTERN'('AL'/.EMPTY) .W(,'EXTERN')                                      1490
        )                                                                        1500
,                                                                                1510
RECOVER=SNAP                                                                     1520
      .SEARCH('.,'/'.END' RETURN/.DELETE .FALSE) $(STATEMENT)                    1530
,                                                                                1540
REDEXP=                                                                          1550
            '(' ( ARITHE '$' .W(.C,'RREAD',...2,*)                               1560
           /'$' ARITHE .W(.C,'CREAD',...2,*)                                     1570
         )')' .R .R *-5                                                          1580
,                                                                                1590
REXP=RTERM$('.OR' TAIL RTERM MODT *1.W(.C,'.OR',*,*) *-1 *-5)                    1600
,                                                                                1610
```

```
RMATRIX =MRPRIMARY                                                              1620
.,                                                                              1630
ROP=.SV '.AND' TAIL                                                             1640
   /.SV'.OR' TAIL                                                               1650
.,                                                                              1660
RPRIMARY=VECTOR(',' RPRIMARY CONCAT/.EMPTY)*1 .PUT'.BR' *-1                     1670
.,                                                                              1680
RTERM=RPRIMARY$('.AND' TAIL RPRIMARY MODT *1 .W(.C,'.AND',*,*) *-1 *-5)         1690
.,                                                                              1700
SHIFTOP= .SV'.SL' TAIL                                                          1710
        /.SV'.SR' TAIL                                                          1720
        /.SV'.SCR' TAIL                                                         1730
.,                                                                              1740
SNAP=.EMPTY .W('STACK IS  '...0'*'...1'*'...2'*'...3'*'...4'*'...5'*'...        1750
       6'*'...7'*'...8'*'...9'*')                                               1760
.,                                                                              1770
ST='.SNAP' SNAP ST                                                              1780
    /'.MODE' ALPHABETIC SETT                                                    1790
   /PRIMARY'='.SV'.BV'TAIL.ID('+'('1'('.,'.PUT'COUNTU' COUNT                    1800
                                      /.PUT'N'.PUT'1' ADOUT PLUS LOD            1810
                                 )                                              1820
                            /TERM ADOUT PLUS LOD                                1830
                          )                                                     1840
                        /'-'('1'('.,'.PUT'COUNTD' COUNT                         1850
                              /.PUT'N'.PUT'1' SUOUT PLUS LOD                    1860
                            )                                                   1870
                            /TERM SUOUT PLUS LOD                                1880
                     )                                                          1890
                       /MPYDIV LOD                                              1900
                       )                                                        1910
                  /TERM PLUS LOD                                                1920
     /(RPRIMARY'='REXP MODT .R                                                  1930
        /MRPRIMARY'='MREXP MODT .R                                              1940
         /MVPRIMARY'=' MVEXP MMODE .R                                           1950
    ).W(,'LOD',...1,*)('.SNAP' SNAP/.EMPTY) .R','                               1960
    /'.END'.FALSE                                                               1970
.,                                                                              1980
STATEMENT=                                                                      1990
          (LABEL .W('L'*,'LABEL')/.EMPTY)                                       2000
          (CONTROL                                                              2010
           /ST                                                                  2020
          /DECLARATION                                                          2030
          )                                                                     2040
                                                                                2050
```

```
SUBEXP=NUMBER(',' NUMBER .W(.C,'SUBREG',*','*,*)                    2060
              /.EMPTY .W(.C,'COMPON',*,*)                           2070
              ) *-5                                                 2080
.,                                                                  2090
SUOUT=MODTE .PUT 'SUB' ACUT                                         2100
.,                                                                  2110
TAIL='.'/.EMPTY                                                     2120
.,                                                                  2130
TERM =PRIMARY$('*' PRIMARY MODTE .PUT 'MPY    ' ACUT                2140
               /'/' PRIMARY MODTE .PUT 'DIV    ' ACUT)              2150
.,                                                                  2160
TEST=                                                               2170
     .SV '.EQ' TAIL                                                 2180
     /.SV '.NE' TAIL                                                2190
.,                                                                  2200
TYPE=                                                               2210
'MATRIX' NUMB NUMB .W(,'MATRIX',...1,*) .R                          2220
    /'VECTOR' NUMB .W(,'VECTOR',*)                                  2230
.,                                                                  2240
TWOP='.OP''('OP1LST ',' OP2LST ')'                                  2250
.,                                                                  2260
VECON=.SV '.ALPHA' TAIL '('NUMBER')'                                2270
       /.SV'.OMEGA' TAIL '('NUMBER ')'                              2280
.,                                                                  2290
VECT1=(.ID                                                          2300
      /COMOP                                                        2310
       /MOD1 ARITHE *1 .R .R *-1                                    2320
        /'.NOT' TAIL RPRIMARY .W(.C,'.NOT' , *),R *-5               2330
        /'(' REXP ')' .MOVE 1 .R                                    2340
     /BITPAT                                                        2350
     /(MRPRIMARY/MRMODE MVPRIMARY .MOVE 1 .R)                       2360
        .MOVE 1 .R  REDEXP                                          2370
       )('('SUBEXP')' / .EMPTY)                                     2380
.,                                                                  2390
VECT2= ROP '/' RPRIMARY.W(.C,...2'RED',*) .R .R      *-5           2400
            /NUMBER SHIFTOP RPRIMARY .W(.C,...2,...3,*).R.R.R *-5   2410
.,                                                                  2420
VECTOR=VECT1('//' RMATRIX .W(.C,'CCOMP',...2,*) .R.R *-5            2430
            /'/' RMATRIX .W(.C,'RCOMP',...2,*) .R .R  *-5           2440
            /.EMPTY)                                                2450
       /VECT2                                                       2460
.,                                                                  2470
XFRLIST=                                                            2480
       $(TEST',' LABEL .W(,...1,*5,'L' *).R)                        2490
.,                                                                  2500
.END                                                                2510
```

```
EXAMPLE
.BDEC
I F/F EXTERNAL
D F/F EXTERNAL
M MMATRIX 1000 G
B MVECTOR G
.GLOBAL 6
.EDEC
.L1 .WAIT(I..1).,
A=.OMEGA(10)/8
.,
.L3 .COMPR(.ALPHA(1)/.MR.M(.BV.A$)..0).EQ.,.L2
.,
.BV.A=.BV.A-1
.,
.COMPR(A...MEPS(10)).EQ.,.L4
.,
.GO TO .L3
.,
.L2 B=(.ALPHA(26)/.MR.M(.BV.A$)),A
.,
I=0
.,
.GO TO .L1
.,
.L4 I=0
.,
D=1
.,
.GO TO .L1
.,
.END
```

FIGURE 12 The Coded Version of Figure 2- The Example Machine

TABLE 1 TEST IMPERATIVES

TEST IMPERATIVE	TEST	DATA MOVED IF TEST CONDITION IS MET	COMMENT
.ID	Is an identifier in IBC?	Move the identifier from the IBC to the top of S .	
ALPHAB	Is first character in IBC a letter?	Move the letter from IBC to the top of S.	
DIGIT	Is first character in IBC a digit?	Move the digit from IBC to the top of S.	
NUMBER	Is the first string of characters in IBC a number?	Move the number from IBC to the top of S.	A number is a string of 6 or less digits.
*P	Does the identifier on the top of the stack have the property P in its symbol table entry?		P is a letter of the alphabet.
'P'	Is the first sequence of characters in IBC the same as P?	Delete P from IBC.	P is any sequence of characters.
.SV 'P'	Is the first sequence of characters in P in IBC the same as P?	If the test result is true the first six characters of P are put on top of S.	P is a sequence of characters.
.TP	Is the value of the mode cell in the CA the same as P?		P is a letter.
.EQ(...P_1,...P_2)	Execute...P_1 and ...P_2(Table 2) then compare the new top cells of the stack if they are the same the test is met. After the test restore the stack to its original condition.		
.TEST 'P'	Is the top of the stack the same as P?	If the test is met, remove the top of S otherwise leave it unchanged.	P is a string of up to 6 characters.

TABLE 2 ACTION IMPERATIVES

IMPERATIVE	ACTION	COMMENT
.FALSE	..Test ← 0 The conditions required by the syntactic equation in which .FALSE appears are not met	
.EMPTY	The conditions required by the syntactic variable are met, no matter what the state of the meta-machine.	
+P	Put P in the property register along with any letters that may already be there.	P is a letter.
.MODE P	Put P in the cell mode in CA at Meta-compile time.	P is an alphabet character.
...P	Move the contents of the Pth cell from the top of S to the top of S. The top of S is indexed 0.	P is a digit.
.MOVE P	Same as ...P except that the Pth cell from the top of S is squeezed out of the stack.	P is a digit.
*P	Remove the contents of the top cell of S from S and put them in the cell which is the Pth column of the top of the recursion stack.	P is a digit $1 \leq P \leq 4$
*-P	If the Pth column of the top of the recursion stack does not contain zero, move its contents to the top of S. If Pth column of the top row contains zero, get an arbitrary but unique symbol from the symbol generator and move it to the top of S. Also put it in the Pth column. of the recursion stack.	P is a digit $1 \leq P \leq 4$
*P	Put the top of S in *P in the communication array.	$5 \leq P \leq 6$
*-P	Move *P in the communication array to the top of S.	$5 \leq P \leq 6$
.SEARCH(EX)	Repeat the execution of EX until the conditions specified by EX are met.	EX is a set of alternative sequences of tests and actions

TABLE 2 (Continued)

IMPERATIVE	ACTION	COMMENT
.DIAGNOSTIC SYNTAX (P)	P is specified as the name of the syntax equation which specifies the action to be taken when the error occurs.	P is an identifier.
.DELETE	Remove the first character from IBC.	
.S P	.S sets P into the MODE cell in CA. This action occurs at compile time and not at meta compile time as in.MODE	P is alphabetic.
REMOVE and .R	Pop the S stack by moving its pointer down 1.	
.P	Move the pointer of S up.	
INSERT	Moves the contents of ..TEST to top of S stack leaving ..TEST unchanged.	
.PUT 'P'	Places P on top of the S stack.	P is a sequence of up to 6 characters including blanks.
CLEAR	Clear the property register.	
SET	Enter the identifier on top of S in the symbol table and give it the properties currently held in the property register.	
OR	Find the entry in the symbol table for the identifier currently on top of the S stack and replace the contents of the corresponding property field with the logical-or of the property register and the property field.	
FI	Move top of S to the top of the FIFO stack.	
FO	Move the bottom of the FIFO to S.	
=	Push down the recursion stack and fill in the top of the array (all four columns) with zeroes.	The = has meaning only after the first identifier in an equation.
.,	Pop up the recursion stack by incrementing its pointer.	., has meaning only at the end of an equation.

TABLE 3 ACTION IMPERATIVES FOR OUTPUT

IMPERATIVE	ACTION	COMMENT
*	Move the top of S to the current output buffer.	
*P	Nondestructively move contents of the Pth column of the top of the recursion stack to the output buffer.	P is a digit
*S	Nondestructively move the top of S without altering the top of S.	
...P	Move the Pth cell down from the top of S nondestructively to the current output buffer.	P is a digit.
.C	Move the contents of *5 (in communication array) to *6. Then add 1 to the contents of *6 and put the result both in *5 and the output buffer.	
'P'	Place P in the output buffer.	P is any sequence of characters.
,	Advance the buffer pointer to the next buffer. All subsequent data transferred to the output buffers will go to the buffer pointed to.	
/	/ applies only to the .OUT imperative. / causes the current contents to be output as MAP instruction. After the instruction is output the buffers are cleared.	

REFERENCES

1. Schneider, F.W., and G. Johnson, "A Syntax-Directed Compiler Writing Compiler," Proc. of the ACM Convention, 1964.

2. Mandell, R., and G. Estrin, "Specifications for a Design Automation System," Proc. of the SHARE Design Automation Workshop, June 1965.

3. Schorr, H., "Computer-Aided System Design and Analysis Using a Register Transfer Language," IEEE Transactions on Electronic Computers, Vol. EC-13, Dec. 1964, pp. 730-738.

4. Gorman, D.F., and J.P. Anderson, "A Logic Design Translator," Proc. of the FJCC, 1962, pp. 251-261.

5. Iverson, K.E., A Programming Language, John Wiley and Sons, Inc., New York, 1962.

6. Cheatham, T.E., and F. Sattler, "Syntax-Directed Compiling," Proc. of the Spring Joint Computer Conference, 1964, pp. 31-37.

7. Rutman, R.A., "LOGIK, A Syntax-Directed Compiler for Computer Bit-Time Simulation," M.S. Thesis, UCLA Library, August 1964.

OTHER META-COMPILERS

8. Reynolds, John C., Cogent Programming Manual (ANL-7022), Argonne National Laboratory, Clearinghouse for Federal Scientific and Technical Information, National Bureau of Standards, Springfield, Virginia.

9. Feldman, J.A., "A Formal Semantics for Computer Languages and Its Application in a Compiler-Compiler," Communications of the ACM, Vol. 19, No. 1, Jan. 1966, pp. 3-9.

10. Ross, D.T., "AED JR., An Experimental Language Processor," Electronic Systems Laboratory, Dept. of Electrical Engineering, Massachusets Institute of Technology, Cambridge, Mass., ESL-TM-211, Sept. 1964.

11. Oppenheim, D.K., "The META5 Language and System," System Development Corporation, 2500 Colorado Ave., Santa Monica, Calif., TM-2396/000/01, 1/25/66.

AN IVERSON LANGUAGE TRANSLATOR

12. Hellerman, H., "PAT Manual," International Business Machines Corp., Yorktown Heights, New York.