MICRO-PROGRAMMING THE SPECTRA 70/35

with elementary operations

by C. R. CAMPBELL and D. A. NEILSON

In recent months several computer manufacturers, domestic and foreign, have announced new product lines that offer processors using read-only memories for control. These read-only memories have a particular number of fixed length words which contain control instructions, instructions that are on a lower machine level than the computing instructions. The systems they are controlling have consistent data paths, so common repetitive programming is applicable. The control instructions transfer data from register to register, trigger main memory, perform the arithmetic and logical choices, test flip-flops and conditionally or unconditionally branch to other read-only memory instructions. These instructions, which are bit configurations of various lengths depending upon machine design, are known as Elementary Operations or, commonly, EO's. A group of Elementary Operations make up a microprogram. The microprogram performs the operating instruction.

Most microprogramming is done in some sort of symbolic coding language, an example of which might be "A + (2) B ⟶ B." This could be translated as, "Add the A register to the two's complement of the B register and store the results in the B register." Along with this particular operation, there could also be a test and branch or an unconditional branch. Depending upon the size of the EO, both the conditional and the alternate branch address could be carried in each instruction.

The bit configuration that results from the symbolic coding causes the correct sequence of operations to take place at the correct time. Now if, in addition, testable indicators such as "carry" or "zero result" are automatically set, the microprogrammer can check the result of his manipulations.

In order to give the microprogrammer sufficient flexibility in his control of the internal operations, several basic functions, such as "add binary," "add decimal," "shift," and "cycle memory" must be provided. These functions are planned, along with the tests and indicators, and the control problems are solved before function definition. When definition is finally made, internal operation can be considered from the symbolic level, free from hardware constraints. If the Elementary Operation is straightforward, the microprogramming of an instruction algorithm is similar to the programming of any application, and the job can be done by a programmer, not necessarily an engineer. This, of course, assumes that the programmer working at the symbolic level cannot create any problems for himself other than programming problems.

As an illustrative example, staticizing an instruction exercises most of the basic functions available to the microprogrammer. A program counter retains the address of the next operating instruction. Using the address stored in the P counter, the microprogram cycles the memory and reads the first two bytes of the instruction to be performed. These bytes are stored in registers for testing. The first byte contains the operation code of the instruction to be performed, and by analyzing certain bit positions the micro-

Before becoming supervisor of microprogramming control of the RCA Spectra 70/35 processor, Mr. Campbell was senior product planner on the 70/15 and /25. A former systems analyst at Thiokol Chemical Corp. and General Motors Ternstedt Div., he holds a BS from Wayne State Univ.

program can distinguish between different instruction formats. The operand addresses are staticized according to the format type, and then the eight-bit operation code is transferred to the read-only-memory address register. This operation affects a branch to the correct microprogram that executes that particular instruction. If the operation code is not a valid one, the branch is to the microprogram for op-code trap interrupt.

Writing microprograms at the symbolic level for internal processor control requires more than just precise EO definition. A few of these requirements and a method to satisfy them can be seen by an examination of the RCA 70/35.

The 70/35 is RCA's latest member of the compatible SPECTRA 70 family. A general system description can be divided into three categories:

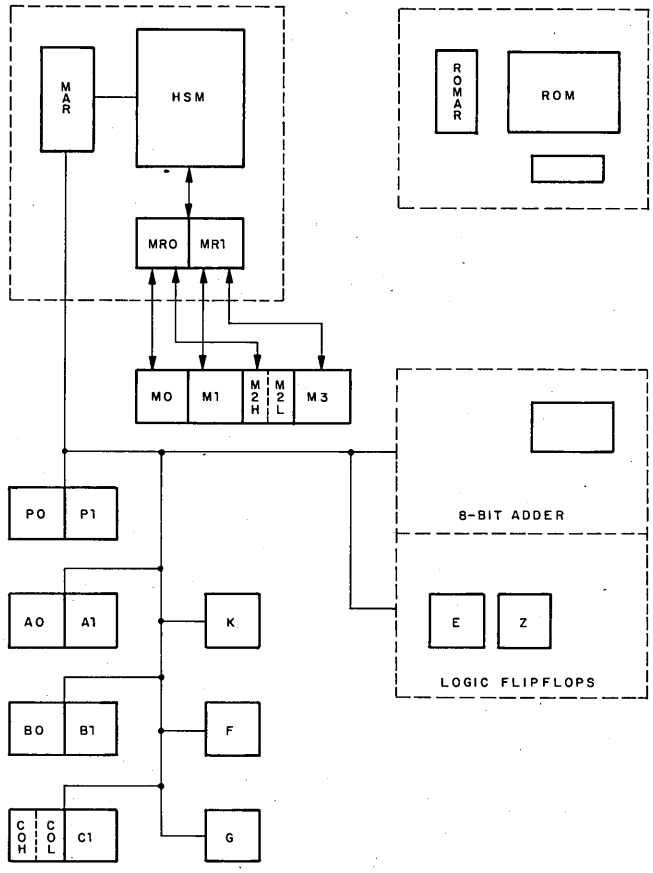
1. Control of the data structure for instruction staticizing and execution is by Elementary Operations (EO's) contained in a nondestructive read-only memory (ROM).
2. All the instructions that are featured in the larger SPECTRA 70 processors are available in the 70/35.
3. All I/O devices in the SPECTRA 70 line communicate with the 70/35 via a multiplexor or selector channel and the RCA standard interface.

In this article we will concern ourselves with the first category. The most significant feature of the 70/35, and the one that guided the general design concept that developed, was the use of a 27-bit EO word.

Several secondary features are unique to the 70/35. However, these are a direct result of the basic EO format decision. Two of the features are: 1) the easy development of a comprehensive EO simulator and assembly system, and 2) the feature that allows performing Elementary Operations obtained from main memory as well as read-only memory.

In December, 1964, RCA announced four new processors, one of which, the 70/45, utilized a read-only-memory type control structure. A read-only memory had been designed for this processor, and it used a 53-bit word. The 70/35 design group decided to add one bit and to use the 54 bits as two 27-bit EO words. Each 54-bit group is accessed in 960 nanoseconds, providing an effective read-only-memory cycle time of 480 nanoseconds. The decoding network for the 70/35 is unique, but the basic read-only memory is the same employed on the 70/45. With the 54-bit word, the read-only memory contains 1024 words. With the 27-bit format this, of course, doubles.

This type of control, therefore, uses 2048 words of lesser complexity as opposed to 1024 larger, more powerful EO's. This, in effect, halves the control memory cost if a one-for-one relationship can be retained. The design group concluded that with a relaxation of performance specifications (in comparison with the 70/45), a simple EO structure could more easily utilize sophisticated programming techniques and make up for the loss in EO power. The relaxed performance requirements allow the technique of modular programming with many common routines, a method that pays a time penalty but saves EO steps. Another factor in the EO word design was that in the final judgment, an EO is still function bound. No matter what the power of any step, whether it be a status level or elementary operation, it still must perform its basic function-branching steps sequentially. The adoption of this EO word enabled all 144 operating instructions to be contained in 2048 EO words. Although 144 instructions seems to be a large amount, the greater number require an extremely simple algorithm. In addition, many instructions are similar in their operation and can be combined into common routines that still provide tolerable timing specifications. When the shared routine method is employed, a sufficient number of the smaller steps are left to handle the more complicated instructions.



70/35 BLOCK DIAGRAM

register communication

Register communication in the 70/35 is implemented through a modified one-byte input bus and a modified one-byte output bus. The microprogrammer can address all the registers in the main data structure, with the exception of the memory address register (MAR) and the memory data register (MR). There are fifteen addressable 8-bit registers in the 70/35 that can be used for data manipulation. Some of these registers are further subdivided into two 4-bit registers which also require separate addresses. The 4-bit registers are used to perform such instructions as Pack, Unpack, Move Numeric, etc. Any

Mr. Neilson is the project manager directing design of the Spectra 70/35 processor. Formerly the product planning manager at RCA's Palm Beach Gardens facility aiding in the development of Spectra 70/15 and /25, he has also been associated with Burroughs and IBM. He received a B.S. in electrical engineering from the Univ. of California.

two 8-bit registers can be used as the operands in any of the register-to-register operations. For the decimal functions the 4-bit registers can be used as a destination with any 4-bit or 8-bit source. This feature aids in the emulation of 6-bit machines. The 70/35 uses a 2-byte memory access with configurations of 16, 32 and 65K. Each of these memory configurations has a section of unaddressable or shaded memory.

In order to apply programming techniques to machine control, functions had to be designed that would provide the programmer with the ability to share routines. One general method, of course, was to make the individual steps very basic, thereby making them common to many algorithms. This was inherent in the extremely basic 27-bit Elementary Operation. Another general method was to provide testing capability in each EO to simplify linkage problems.

The 70/35 Elementary Operations fall into two categories; register-to-register operations, and variable operations (i.e., cycle memory, offset address, etc.). Of the 27 bits, five are always used to describe the function type. Fourteen of the functions are the register-to-register operations. These are described as follows:

EO LAYOUT

W	X	Y	T	J
5	5	5	6	6

W—This is the five-bit function field. This is the EO op code and directs the operation to be performed.

X — This five-bit field describes the source and destination register for the operation as described by W.

Y — This five-bit field describes the other source register for the operation described by W.

T — The six-bit field describes a test to be performed. With six bits, 64 tests are available to the 70/35.

J — This six-bit field has a branch address that is taken as the test is true. Because of the six-bit limitation this branch address is limited to modulo 64.

FUNCTIONS

Transfer —

When the function (W Field) describes a transfer, the bit configuration held in the register described by Y is transferred to the register described by X.

Add —

This function transfers the quantity in the register described by X to a holding register. It then takes the quantity in the register described by Y, adds it to the quantity in the holding register, and gates the result back to the register described by X.

Add One's Complement —

This function is similar to the Add except that the X quantity is complemented before the Add.

The balance of the functions used in register-to-register operations provide for decimal adding and subtracting, adding with a signed binary, and for performing the logical operations (And, Or, Exclusive Or). The register-to-register operations can also be used with an eight-bit literal operand held in the EO word.

Along with the register-to-register operations, several other functions are available to the 70/35 microprogrammer. There is a function to cycle the main memory, a function to quickly create memory addresses from a 4-bit register number, a function to permit internal two-byte transfers register-to-register, and a two-way branching function. All the functions that are not register-to-register allow a choice of either a test and branch or an unconditional branch. The test and branch will still allow only modulo 64 branching. The unconditional branch, however, allows branching to any location in the read-only memory.

Two functions greatly aided routine sharing in the 70/35, the first of which was an "indirect function." This function utilized as its decoding base (function bit configuration), the low order four bits of the op code register. Routines then could be written for a general approach with the arithmetic or logical operation determined by the particular instruction to be performed. By using the four low order bits of the op register, 12 different indirect functions can be performed in the 70/35. Assuming staticizing is concluded, the example is a general illustration of the indirect function:

1. Read operand 2; test
2. Read operand 1
3. Perform indirect functions; test
4. Write operand (result of function)
5. Decrease length and if not zero branch to #1

With initializing and the proper tests assigned, this routine can handle Add Decimal, Subtract Decimal, And, Or, Exclusive Or and with tests that permit skipping one step or another, the routine can also do Zero and Add, Compare Logical or Compare Decimal.

Another function added for programming versatility was the indirect branch, a method by which the branch address is previously prepared in a particular register. Upon conclusion of a routine, the register is transferred to the ROM address register and the proper return address is effected. This is, of course, a common programming technique. However, the important thing is that it was implemented in the 70/35 specifically as a microprogramming aid.

The 70/35 presently has 26 defined functions, with several having multiple variations. This leaves six functions for future unique applications.

The elementary operations available to the 70/35 microprogrammer are as follows:

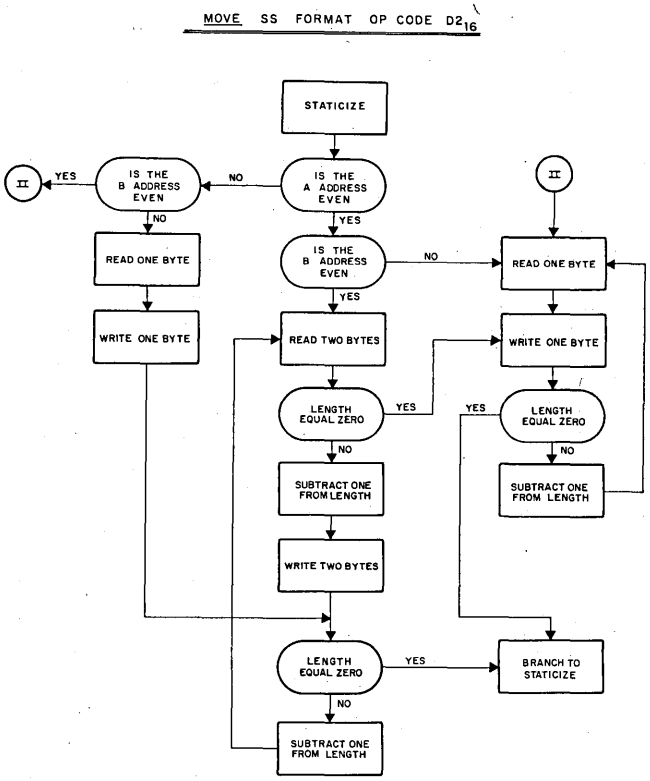
Register-to-Register	Variation
Add	Cycle Memory (3 functions)
Add With Sign	Offset Transfer (address) (2 functions)
Add One's Complement	Two Byte Transfer
Add One's Complement With Sign	Shift
Add With Reset Carry	Literal Increment-Decrement
Add Two's Complement	I/O Control
Add Decimal	Special Branch (2 functions)
Add Nine's Complement	Special Control
And	
Or	
Exclusive Or	
Compare (binary)	
Transfer	
Indirect Function	

Early in the 70/35 design study a simulation system was designed that would be the main tool for testing algorithm logic on the engineering prototype. Microprograms were written in a simple coding language and assembled in an edit phase. When the microprogrammer was ready to test his program, he entered it along with any required initializing data into the simulator phase. This program traced the microprogram and provided a step-by-step internal picture of the machine for the programmer to analyze. After the complete microprogram file was considered logically true, read-only-memory allocation data and logic definition were combined with the file to produce the Elementary Operation bit configurations and the read-only-memory wiring list. In addition to the edit, simulation

and read-only-memory assembly, the system provides many statistical runs analyzing test and branch data.

This complete operation was viewed from the symbolic level by the microprogrammer. In the original planning, hardware aids, function requirements and many other significant design decisions were mutually discussed, but subsequently all algorithms were written and tested using the simulator.

In order to demonstrate the operation of a 70/35 microprogram and the symbolic language used to program it, the sequence for Move Numeric and Move Zone is shown with a detailed explanation. The operation code for Move Numeric is $D1_{16}$ (11010001); the operation code for Move Zone is $D3_{16}$ (11010011). During staticizing, the operation code is placed in the F register, the length is placed in the G register, a fully staticized second operand address is placed in the B register and a fully staticized second operand address is placed in the A register. With the operation code in the F register, the microprogram can distinguish between Move Numeric and Move Zone by



FUNCTIONAL FLOWCHART OF THE MICROPROGRAM FOR THE SS FORMAT MOVE INSTRUCTION

examining the one bit where the two op codes differ. This example assumes the instruction has been staticized.

1. A → S, RR, +1 (the symbolic expression for a cycle memory read)
2. M_2 → C_0
3. B → S, RR, +0 F_1=0 Branch to 7
4. C_{0H} → M_{2H}
5. B → S, WRS +1 G=(00) Branch to Staticize
6. G–(1) → G Branch to 1
7. C_{0L} → M_{2L} Branch to 5

1. The second operand is read from main memory and put into M_2. The A address is incremented by one.
2. Register M_2 is transferred to C_0. Both M_2 and C_0 are sub-divided into two four-bit registers.
3. The first operand is read from main memory and put into M_2. In addition the bit F1 is tested to see whether the branch is taken. If not, the next sequential step is performed.
4. The zone from the second operand previously stored in C_0 replaces the zone of the first operand.
5. The modified first operand is written to main memory and the B address is incremented by one. The G register, which holds the length count, is checked for zero. A length of zero indicates the instruction is complete. If the test is true, the branch to staticizing is taken. If the test is not true, the next sequential instruction is performed.
6. One is subtracted from the G register and the microprogram unconditionally branches to Step 1.
7. In Step 3, a test was made for instruction type by checking bit F1. If the instruction was Move Numeric, Step 3 is followed by Step 7 which replaces the numeric portion of the first operand and unconditionally branches to Step 5.

special features

The 70/35 can contain an optional second read-only-memory bank. Assuming one is used to perform the 144 SPECTRA 70 instructions (this is not a systems requirement), this provides another 2048 27-bit elementary operations for some other purpose. With a control structure that is "data oriented" rather than "instruction oriented," it is possible to emulate almost any machine organization. In fact, with the hardware improvements available, the 70/35 emulates older 6-bit machines with a substantial improvement factor.

Another unique feature on the 70/35 is the implementation of the Diagnose instruction. The Diagnose instruction is primarily a test and maintenance feature which aids troubleshooting in the field and initial checkout in manufacturing. However, it can also test future read-only-memory bit configurations for accuracy.

The object of any maintenance aid is to identify the problem area. To do this effectively, it was determined that the test and maintenance programs should be able to control the internal structure of the machine. The adoption of the small EO word structure provided the 70/35 with the ability to easily perform microprogramming steps obtained from main memory. This method allows the test and maintenance programs to make internal checks never before possible. It also permits the programmer to simulate foreign instructions at the microprogram level.

The Diagnose instruction will perform any function that can be performed from the ROM except input/output instructions. For I/O, the test and maintenance routine must jump back to the program level. However, because they are able to control internal machine operation this presents no problem. Termination interrupts can be handled at the program level or at the microprogram level. The preparation of test and maintenance microprograms can use a variation of the same simulation system used by the microprograms in ROM. As in the regular microprograms, coding is done in the simple assembly language and all work is done at the symbolic level. The Stimulation System provides the listings, the testing, and the object bit configuration.

The techniques developed in the current processors using read-only-memory control offer a high degree of flexibility for future system design and marketing. If the Elementary Operations are basic and easy to understand, a user could program his own internal data structure. The specialized customer has selected system configurations that approximate his real need, but at the same time has long wished for a system tailored to his unique application. The design method used with read-only-memory controlled processors make this unique system practical for both user and manufacturer. ■