A small computer for the direct processing of FORTRAN statements

By Alan J. Melbourne* and John M. Pugmire†

A small computer with a microprogrammed FORTRAN compiler is described. A comparison is made with a conventional computer of similar speed which uses a software compiler.

Scientists and engineers in university and industry frequently encounter problems too lengthy or complex to be conveniently handled by a desk-calculator, but which are small by computer standards. Tedious though a hand-calculation may be, it may nevertheless be preferable to a computer solution unless several conditions can be satisfied.

(i) The computer must be easy to program. The user, although qualified technically, may be but an inexperienced programmer unable to undergo special training for the occasional usage envisaged. High-level languages such as ALGOL or FORTRAN have been expressly created for ease of use, and have the additional advantage that programs so written are easily readable.

(ii) The computer must be readily available. The nature of scientific work makes accurate forecasting of computer time difficult. For this reason, renting time on a large installation is not entirely satisfactory because of the tight job-scheduling involved. A small computer locally installed seems preferable if provided with a compiler. Compilation, however, is a time-consuming process and may take longer than running the final compiled program. It should be reduced to a minimum.

(iii) Quick and comprehensive correcting facilities must be provided. An inexperienced and infrequent programmer must be expected to make mistakes. If each mistake should involve a complete or partial recompilation, considerable time will be wasted, particularly if the object program is only to be run once.

(iv) The installation (including compiler facilities) must be low-cost. It is envisaged that the computer be shared between several groups; due to the unpredictability of demand there must be periods of idleness to provide reasonable availability.

A machine-design philosophy is here outlined which aims at providing immediate problem-solving facilities at the disposal of the FORTRAN programmer. The internal language of the machine has essentially the same structure as FORTRAN, which means that the "compiler" is small: small enough to be incorporated economically in the control unit. Embodied in control circuitry rather than core-storage, the compiler is thus extremely fast, and "compilation time" is easily absorbed within the mechnical delays inherent in the program input procedure.

The only input-output device needed is a typewriter, and programs can be debugged and corrected directly at the typewriter console, without re-compilation.

Machine language

FORTRAN statements vary considerably in length and structure, and consist of delimiters and identifiers which are themselves of variable length. Some delimiters correspond to a defined computer operation; others serve as identification of statement types.

The stored form of the FORTRAN statement differs from the user's form in the following respects:

(i) all identifiers (including array names) are replaced by storage addresses;

(ii) delimiters initiating a chain of control commands are replaced by the control-unit address of that command sequence;

(iii) the identification delimiters are not stored;

(iv) arithmetic statements are converted into Reverse Polish notation; this could equally well be effected on execution, but would be more time-consuming.

Core storage is organized in 16-bit bytes, the mode of each being determined by the four high-order bits, thus:

(*a*) operator, the remaining 12 bits form the control sequence address;

(*b*) numeric, 4 decimal (4-bits) digits, representing either data or a storage address;

(*c*) alphabetic, 2 alpha-numeric characters, each of 8 bits.

The general instruction form is therefore an operator byte followed by a string of numeric bytes, and bears a close structural resemblance to the original FORTRAN statement. The arithmetic statement is treated as a

* *IBM World Trade Laboratories (Great Britain) Ltd., Hursley Park, Nr. Winchester, Hants.*
† *Compagnie IBM France, 5, Place Vendome, Paris 1er, France.*

string of independent substatements terminated by "=", the "store result" operator. Subscript parentheses and commas are replaced by array operators: other parentheses are removed.

Implementation

Control is by microprogram, held in fixed-store. Fixed and floating-point arithmetic are inherent. The data flow is unremarkable, comprising five 16-bit registers. Any of 100 reserved storage locations (RSL) may be directly addressed by the microprogram, and act as control registers and pointers.

The microprogram creates a push-down stack in storage by assigning a RSL to control allocation of space to stack entries. The RSL contains the address of the current stack extreme; this address can only be modified by the microprogram by an amount equal to the size of one stack entry.

The word-length is fixed at m bytes throughout a given program. The value of m is selected at the operators' console: $2 \leqslant m \leqslant 9$.

Program acceptance

In the basic machine with typewriter input, each FORTRAN delimiter is assigned one keyboard position. Depressing the key initiates automatic printout of the delimiter name, whilst an associated 1-byte operator is inserted in Instruction Storage in the location immediately following the previous (translated) statement.

This operator is the address of the translation microprogram for the current statement type. Translation is split into two phases: Primary and Secondary.

Primary translation

The principal activities during the first phase are:

(*a*) checking the incoming statement for errors;

(*b*) normalization, packing, and justification of incoming quantities into a standardized form, viz.: m bytes for constants, 3 bytes for identifiers, 1 byte for statement numbers;

(*c*) recognition of delimiters and replacement of each by the address of the corresponding section of the Secondary Translation microprogram.

Secondary translation

The second phase scans the standardized form produced by Primary Translation and performs the following principal activities:

(i) Creation of the final form of the statement in Instruction Storage. In the case of arithmetic statements and subscript expressions this involves a conversion into a Reverse Polish form.

(ii) Allocation of space in Data Storage for all identifiers, function arguments and arrays.

(iii) Construction of a Statement Number table showing the correspondence between statement addresses in the source program and in core storage.

Instruction storage

The translated form of the statements may be divided into three categories:

(*a*) *Fixed length and format*: e.g. DO n I = m_1, m_2, m_3 which always occupies 6 bytes.

(*b*) *Variable length, fixed format*: e.g. READ, A, B, C. The microprogram detects the end of statement by the initial operator of the following statement.

(*c*) *Variable length and format*: arithmetic statements and function statements, which may be considered as strings of unary and binary operators, terminated by detection of the initial operator of the following statement. All arithmetic statements start with a special machine-inserted operator denoted by +

For example: A = B + FNX(C * D, U. V) will be held as 15 bytes

$$+\ bf_1xcd * f_2uv/f_3 + a =$$

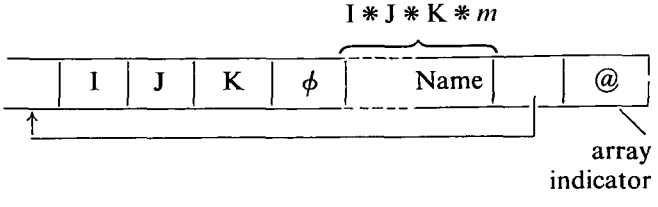
where b, c, d, etc., are machine addresses, x is the address of the Data Storage area allocated to FNX. The operators f_1, f_2, f_3 correspond to (,) respectively and are effectively "begin function", "end argument" and "end function".

Data storage

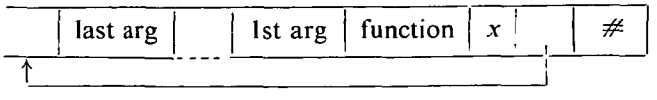
As Instruction Storage is built up from the low-order extreme of core storage, so the Data Storage is allotted from the high extreme.

Simple variables and constants are allocated m bytes. The variable name is inserted in the 3 high-order byte locations (names are restricted to 4 alphanumeric characters if $m = 2$). A constant value is regarded as its name which thus occupies m bytes.

The statement DIMENSION ARRAY (I, J K) is "executed" during Program Acceptance, and allocates space as shown:



Functions may be defined by Function Subprograms or in arithmetic statements. In both cases, a Data Storage area is allocated as shown:



where x is the address of the function subprogram in Instruction Storage. The FUNCTION Statement itself is not translated. The marker $\neq$ takes one of two values according to whether the function is being currently defined, or is already defined.

The special characters @, $\neq$ are used to skip function array areas in a normal identifier scan.

At the end of Program Acceptance, locations 100 to IS_0 contain the machine version of the input program; locations DS_0 to M (M is the maximum machine address) contain data space for variables, arrays and functions. Locations SN_0 to SN_1 contain the Statement Number Table.

If the DEBUG console switch is on, the Statement Number Table is retained and a new Symbol Table created in which each identifier is paired with its machine address; both tables are then retained during the execution phase.

Program execution

The execution phase is supervised by a Master Microprogram which delegates control to an individual statement microprogram.

For the purpose two stacks are created:

(i) Instruction Stack, starting at IS_0 and increasing in the direction of storage address M.

(ii) Data Stack, starting at DS_0 and increasing in the direction of storage address 100.

Master microprogram

Upon detection of an operator symbol in Instruction storage, the Master Microprogram places the address of that instruction in the Instruction Stack. Control is transferred to the individual statement microprogram, which may itself utilize the Data Stack. Upon completion of the current statement, its address is removed from the Instruction Stack, and control is returned to the Master Microprogram.

A DO statement is not regarded as completed until all instructions in its range have been performed the specified number of times. Occurrence of a second DO within the range of the first will increase the Instruction Stack; thus during the execution of a DO nest of depth d the Instruction Stack will contain up to $d + 1$ entries.

To facilitate housekeeping, the last statement in any DO range contains an EDR (End DO Range) flag in the operator byte. After execution of each statement, the Master Microprogram tests for an EDR flag. If it is not present, the next sequential statement is executed.

DO statement execution

A large part of the housekeeping associated with the DO statement is performed by the Master Microprogram as described above. The DO statement microprogram merely initiates the DO index and examines the Instruction Stack. If a reference to this DO statement is already in the stack, the stack is decreased to that point. This is in case a backward jump out of a DO nest had previously occurred.

Function execution

A space corresponding to the Function Data Storage area is allocated in the Data Stack. Using the newly-defined stack limits, each argument is evaluated as an arithmetic expression and its value copied into the reserved stack area. When all arguments have been evaluated, the contents of the reserved stack area are copies into the Function Data Storage area and the stack decreased. The Function subprogram is then entered, using the argument values in Data Storage. The final function value is entered in Data Storage and also the Data Stack, in case the same function should be immediately called again.

Input-output execution

In the basic machine, the input-output format is decided by the data content. Complete format control can be provided in an expanded machine, together with the possibility of attaching punched-card and paper-tape equipment.

Program testing

Since the machine is to be usable by inexperienced programmers, a high error rate must be expected during program input, even with the automatic delimiter print-out provided. The microprogram will perform all the functions of a pre-compiler, except that of flow-tracing.

A CORRECT key is provided for those errors detectable while the individual statement is being entered. The internal control registers are then reset to their pre-statement values, the carriage is shifted and returned, and the corrected statement can be entered.

Errors detected after statement entry can be corrected only between numbered statements. The ALTER key is depressed, followed by the last numbered statement preceding the erroneous one. All statements preceding the next correct numbered statement (n_0) are then re-entered in order, including the corrected statement. The UNTIL key is then depressed, followed by the number n_0.

To correct a program during execution, it is necessary to maintain both symbol and statement number tables throughout. This is effected by the DEBUG switch, which also calls the Trace facility. Whenever a statement is executed, its number (or an X if unnumbered) is printed. DO statements are represented by an open bracket, the conclusion of each iteration by a comma, and the termination of the range by a closing bracket.

Evaluation

To assess the effect of the proposed machine philosophy on cost and performance, we compare with machine X, a commercially available machine of similar range.

We note that, for any set of representative problems, there is a common set S of basic operations which both machines will perform: the purely arithmetic storage-to-storage operation (excluding pseudo-arithmetics like subscript manipulation and sign change). We now regard the proposed machine P as comprising: P1, the orthodox register-data flow configuration, together with the microprogram for executing the operations S; P2 the remaining microprogram, which includes the compiler, array manipulation, function execution, and general stack-organization facilities.

Cost

Because the pushdown is microprogrammed, there is no special circuitry in the machine, which may be regarded as typical of machines of the same size. The cost of the proposed philosophy therefore lies wholly within the P2 microprogram, which is in fact almost exactly the same size as that of P1. In a machine in the low-cost scientific field, therefore, the cost is approximately that of doubling the microprogram of a more conventional machine.

Note that the basic FORTRAN processor needs only typewriter input-output.

Performance

Machine X is provided with a single-pass FORTRAN compiler. A set of representative problems was:

(i) hand-coded for machine X,

(ii) written in FORTRAN, and

(iii) compiled on machine X.

After allowing for difference in basic circuit speeds, the execution times for the described machine were, for all problems:

(*a*) greater than the hand-coded solution times, but

(*b*) less than the compiled solution times.

The instruction storage required by machine P was at all times less than that required by either configuration of machine X. The time taken to write the programs in FORTRAN was considerably less than to hand-code them.

Efficiency

For programs which are to be run repeatedly, optimum coding for inner loops becomes increasingly important. If the program is to be restricted to FORTRAN, the P2 microprogram may have to be larger and more sophisticated. Clearly any compiler feature can be implemented in microprogram—at a price. Two factors contribute to the minimization of the P2 microprogram: the close structural resemblance between source and object program, and the fact that the compiler itself can be implemented in a language (chosen by the machine designer) which is itself independent of source and object languages.

The machine is intended for the inexperienced programmer who wishes to express and solve scientific problems, preponderantly "one-off," with a minimum of delay and cost. Time saved in writing and testing programs will more than compensate for any inefficiency in inner-loop processing. Direct communication with the machine will quicken the education process.

Conclusions

There is little doubt as to the usefulness of a machine which provides the facilities described here. Of the several design philosophies conceivable, we have studied the feasibility and practicability of incorporating the FORTRAN processing facilities in the machine circuitry. Some measure has been given of the effect of the proposed philosophy on the cost and performance of the basic computer unit. It is felt that any depreciation here is more than offset by the improved efficiency and operational facilities of the overall system.

Acknowledgements

The design and a complete simulation of the machine described formed part of an exploratory machine design study under Mr. C. E. Owen, IBM World Trade Laboratories (Great Britain) Ltd. The authors wish to thank the laboratory management for permission to publish this paper.

Errata

A simplex method for function minimization by J. Nelder and R. Mead.

The following alterations are required in the appendix to the above paper which was published in Vol. 7, p. 308. The two expressions for the information matrix should have a factor of 2 attached, and that for the variance-covariance matrix a factor of $\frac{1}{2}$. The authors are grateful to Dr. Rodes Trautman for drawing their attention to this slip.