Chapter 18

The IBM 1401

The second-generation transistor-technology IBM 1401 has been included both because a large number[1] have been produced and because it differs from common fixed word length binary and decimal computers. IBM 1401s are used in business data-processing applications requiring variable-length character strings or fields and rather limited calculating ability. Two specific applications are as a card processor in making a transition from plugboard programmed calculators to full-scale automatic computations and for converting data from one medium to another, for example, from card to tape. The 1401 was little used by the scientific, engineering, and scientific business data-processing communities, probably because of the limited Mp size, the low overall processing speed, and the lack of concurrent I/O operation in the smaller configurations. However, it did achieve considerable use as a stand-alone Cio in C('7090) installations, perhaps because of the speed and quality of the T('1403; line; printer).

Although undoubtedly influenced by machines outside the IBM organization, the IBM 1401 is derived primarily from the IBM 702 and 705, which are variable word length decimal machines. The relationship of the various IBM decimal computers to one another is shown in Fig. 1. (RCA's early computers[2] also use a combination of fixed-length and variable-length 7-bit character strings and may have influenced the 1401.)

The IBM 1401's ISP was the first to be adopted by another company. Honeywell defined its H-200 ISP to be a superset of the IBM 1401 ISP. The ISP of the H-200 is more complex and increases performance by organizing Mp by both characters and words.

The IBM 1401, 1440, and 1460 are the only IBM computers to be completely character-string oriented. That is, both instructions and data are stored in variable-length character strings; these strings are addressed by a pointer register to the string. The address integer is fixed at three characters. The encoding process for addresses is given in Appendix 1 of this chapter. The 3-character address (3×6 bits) is assigned as 3×4 bcd characters for encoding addresses 0:999; 2×2 bits for selecting $16 \times 1{,}000$ addresses; and 2 bits for selecting one of the three index registers.

The IBM 1620 processes variable-length data strings, although the instruction length is a fixed 12-digit string corresponding to a word in Mp. The 1620, though not identical to the 1401, is almost a member of the same family.

The 1401 evolved. Figure 1 shows the evolution of "features" which have created new computers. The 1401's optional features are mainly design afterthoughts; they sometimes increase performance, sometimes make certain operations possible, and sometimes provide substantive change. There are approximately 19 features in the 1401: memory expansion beyond the anticipated 4,000 characters and index registers required encoding the field bits of the A and B addresses; store A-Address and store B-Address register

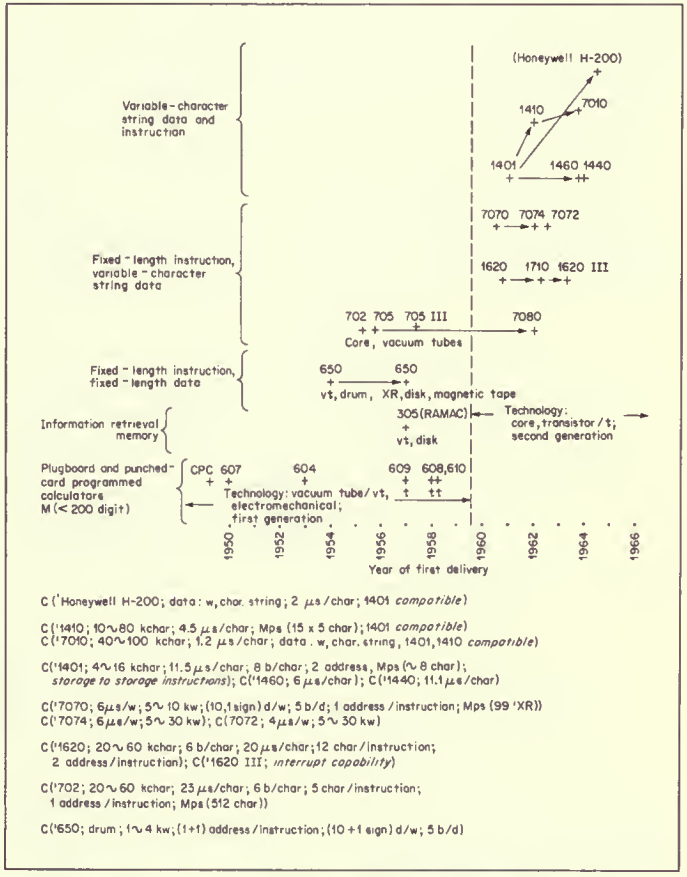

Fig. 1. IBM decimal and character-string computer relationships.

[1] Up to 1966, more 1401s were produced than any other model. An estimated 7,500 1401s, 1,500 1401 G's (card-only system), 3,600 1440s, and 1,500 1460s were produced. About 1,800 1620s were produced.

[2] RCA 301, 501, and 601.

instructions are necessary for subroutines—the Store Address Register Feature; Indexing Feature; Multiply-Divide Feature; High-Low-Equal Compare Feature; Read Release and Punch Release Feature; the Column Binary Feature; Early-Card-Read Feature; Processing Overlap Feature, etc.

PMS structure

The 1401 PMS structure (Fig. 2) is an early 1 Pc structure. The diagram does not show the S(fixed) Pc interconnection structure with the Ms and T. The Pc-(Ms|T) interconnection restricts the concurrency of T and Ms. The optional processing overlap feature provides a link to Mp to allow the T(card; read, punch) to be run concurrently with Pc processing. When any of the peripheral devices are operating without the processing overlap feature, the Pc is dedicated to be a data transmission link or K (as in earlier computers). The device K is connected directly to Pc. For example, Ms(disk, magnetic tape) data transfers use the main registers of the Pc and can tie it up full time during data transmission. By careful programming, several devices can be synchronized and thus run concurrently for communicating with Pc from a K. The Pc does not have an interrupt system. Thus the peripherals have no way of communicating with Pc. Subsequent models, the 1440 and 1460, added interrupt capability and made it easier to control multiple simultaneous data transfers among the peripheral K's and Pc.

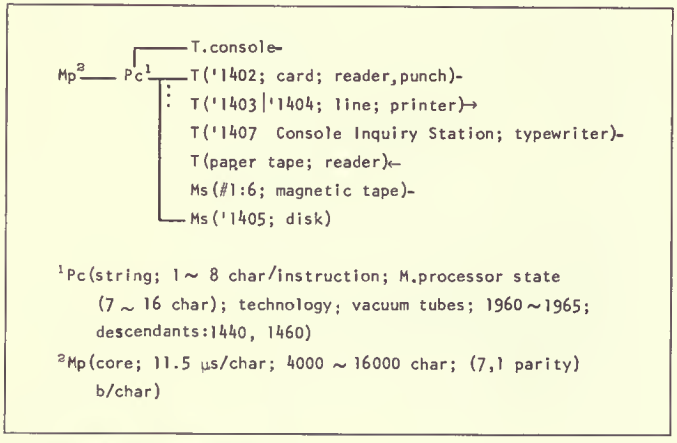

Fig. 2. IBM 1401 PMS diagram.

ISP structure

The IBM 1401 ISP is given in Appendix 1 of this chapter. Instruction strings and data strings are delimited by the special F bit in a character. A character in Mp is of the form[1]

C⟨check,F,B′,A′, 8, 4, 2, 1⟩
An n-character string is C[0], C[1], . . . C[n − 1]
and would be stored in Mp[j:j + n − 1]

The first character (or head) of an instruction must contain the word-mark flag or F bit. The head of the instruction, which is to be interpreted next, is held at Mp[I], and succeeding characters of the instruction are at Mp[I + 1], Mp[I + 2], etc. Correctly defined instructions are 1, 2, 4, 5, 7, and 8 characters long. Undefined instruction lengths of up to 8 characters are also interpreted without an error condition. The interpretation algorithm presented in the ISP description does not explain the action of instructions which have an incorrect length. Actually, the 1401 Reference Manual does not go into details of general instruction interpretation but dwells on "correct" operation. Table 1 presents the correct instruction lengths and formats. If we take the instructions in the table, the set is not variable in length but is fixed at these six sizes. The instruction set (not including the input/output instructions) is presented in Table 2. This table also provides a hint of the implementation, since the execution times are given in terms of memory cycles.

The ISP state, unlike that of more conventional processors, has no temporary operand storage (e.g., accumulators). The ISP state has registers which point to operands. The state of the machine (see Appendix 1) is basically: Mp, the Instruction Location Counter, Indicators or miscellaneous bits, three 3-character blocks of Mp reserved for Index registers, and the two registers A_address and B_address which point to data operands.

Instruction interpretation

There are three principal state types in processing an instruction: o.q., when the instruction is being formed; o.v., when the operands are being accessed or the results are being stored in Mp; and o, when the operation specified by the instruction is being carried out. Each state transition corresponds essentially to a memory access. The three instruction types of Fig. 3 each have their own particular states. Only types 1 and 2 process the variable-length

[1]See Appendix 1 of this chapter for the meaning of the bits in a character. We have renamed the A and B bits A′ and B′ to avoid confusion with the registers.

Table 1 IBM 1401 instruction formats

Length (char)	*Location:* M[l]	M[(l + 1):(l + 3)]	M[(l + 4):(l + 6)]	M[l + 7]	*Types*
1	C[0]				no-op, halt, or single character to specify a chained instruction
2	C[0]	C[1]			the d_character is used to specify additional instruction information (e.g., select, card stacker)
4	C[0]	C[1, 2, 3]			unconditional branch instruction or single address arithmetic; M[A] ← f(M[A])
5	C[0]	C[1, 2, 3]	C[4]		conditional branch instruction; C[4] selects a specific test
7	C[0]	C[1, 2, 3]	C[4, 5, 6]		two address instruction; M[B] ← M[B] b M[A]; (e.g., add, subtract)
8	C[0]	C[1, 2, 3]	C[4, 5, 6]	C[7]	conditional branch based on Mp[B] character; d_character is test character; (e.g., branch if character equal)

Function of instruction characters:
C[0] op code; always contains a word-mark flag or F bit.
C[1, 2, 3] = branch address for I_Address register or first operand address for the A_Address register.
C[1] or C[4] or C[7] d_character; used as a single character for additional operation code information or a character for comparison, or to select a test.
C[4, 5, 6] primary operand (B_Address register specification).

character strings, {char.string}, and the state diagram accounts for strings on a character-at-a-time basis. For an add instruction Fig. 3 oversimplifies the execution because it implies that each character of the A and B operand is accessed, the addition is performed, and the result is restored according to the B_address register. A more complex description must account for A and B strings of unequal length, and the case of getting a number which must be recomplemented because it is the wrong sign. The recomplementation process requires a reverse scan to find the end of the B string and then a forward scan to recomplement each character of B. Figure 4 is a detailed state diagram of the add execution process.

The states in the ISP description (Appendix 1) within the instruction-interpretation process correspond to the three state types just described: the single-instruction character-fetch operation, the fetch-operand-addresses for the remainder of the instruction, and Instruction_execution. Instruction_execution is not given in any detail. For example, the execution of add is defined as "A"(:= op = 110001) → Ov□M[B] ← M[B] + M[A] {char.string};. The state diagram (Fig. 4) presents this execution in detail. Note that in the ISP description we omit telling the reader that the A and B address registers point to the next lowest variable-length string in M after an operation is performed. We allow the definition of a variable-string operation, for example, + {char.string}, to imply the action on the processor state.

Some instructions can be defined with a single character, and these are called chained instructions. Chained instructions take the previous values of the pointer registers, the A and B address registers, as the operand addresses. The add instruction, for example, can be either 1 (chained), 4, or 7 characters; the forms of all instructions appear in Table 1. The 4-character add instruction places the A address field in both the A and B address registers; thus the effect is an instruction to double a string (add it to itself).

Data

An n-decimal-digit numeric data string is represented as

$$\underline{C[n-1]}, C[n-2], \ldots, C[1], C[0], \underline{C[M]}$$

The underlined characters, $\underline{C[n-1]}$ and $\underline{C[M]}$, have the flag bit present, that is, $(C[n-1]\langle F\rangle = 1)$ and $(C[M]\langle F\rangle = 1)$. The n characters are stored in locations Mp[j], Mp[j + 1], . . . , Mp[j +

Table 2 IBM 1401 instruction set (excluding input, output)

Instruction	*Op Code*†	*Execution time in memory cycles*‡	*Length (char.)*	*Data type*
Add (no recomplement)	A	$L_I + 3 + L_A + L_B$	1, 4, 7	char. string
Add (recomplement)	A	$L_I + 3 + L_A + 4L_B$	1, 4, 7	char. string
Branch	B	$L_I + 1$	4	3 char
Branch if Bit Equal§	W	$L_I + 2$	8	1, 3 char
Branch if Character Equal	B	$L_I + 2$	8	1, 3 char
Branch if Indicator On	B	$L_I + 1$	5	1, 3 char
Branch if Word Mark and/or Zone	V	$L_I + 2$	8	1, 3 char
Clear Storage	/	$L_I + 1 + L_x$	1, 4, 7	char. string
Clear Word Mark	¤	$L_I + 3$	1, 4, 7	1 char
Compare	C	$L_I + 1 + L_A + L_B$	1, 7	char. string
Divide (aver.)§	%	$L_I + 2 + 7L_RL_Q + 8L_Q$	7	char. string
Halt	.	$L_I + 1$	1	
Load Characters to A Word Mark	L	$L_I + 1 + 2L_A$	4, 7	char. string
Modify Address§	#	$L_I + 9$	4, 7	3 char
Move Characters to A or B Word Mark	M	$L_I + 1 + 2L_w$	4, 7	char. string
Move Characters and Edit	E	$L_I + 1 + L_A + L_B + L_y$	7	char. string
Move Characters to Record or Word Mark§	P	$L_I + 1 + 2L_A$	7	char. string
Move Characters and Suppress Zeros	Z	$L_I + 1 + 3L_A$	7	char. string
Move and Insert Zeros§	X	$L_I + 1 + 2\Sigma L_A + \Sigma L_z$	7	char. string
Move Numeric	D	$L_I + 3$	1, 7	1 char
Move Zone	Y	$L_I + 3$	1, 7	1 char
Multiply (aver.)§	@	$L_I + 3 + 2L_C + 5L_CL_M + 7L_M$	7	char. string
No operation	N	$L_I + 1$	1	
Set Word Mark	,	$L_I + 3$	4, 7	1 char
Store A-Address Register§	Q	$L_I + 5$	4	3 char
Store B-Address Register§	H	$L_I + 4$	4	3 char
Subtract (no recomplement)	S	$L_I + 3 + L_A + L_B$	1, 4, 7	char. string
Subtract (recomplement)	S	$L_I + 3 + L_A + 4L_B$	1, 4, 7	char. string
Zero and Add	?	$L_I + 1 + L_A + L_B$	1, 4, 7	char. string
Zero and Subtract	!	$L_I + 1 + L_A + L_B$	1, 4, 7	char. string

†Alphanumeric code used to specify instruction.
‡M(t.cycle: 11.5 μs/char)
§Optional-feature instructions.

Abbreviations for symbols used in timing:
L_A = length of the A-field (in characters)
L_B = length of the B-field
L_C = length of multiplicand field
L_I = length of instruction
L_M = length of multiplier field
L_Q = length of quotient field
L_R = length of divisor field
L_S = number of significant digits in divisor (excludes highorder 0s and blanks)
L_W = length of A- or B-field, whichever is shorter
L_X = number of characters to be cleaned
L_Y = number of characters back to rightmost 0 in control field
L_Z = number of 0s inserted in a field
Σ = number of fields included in an operation

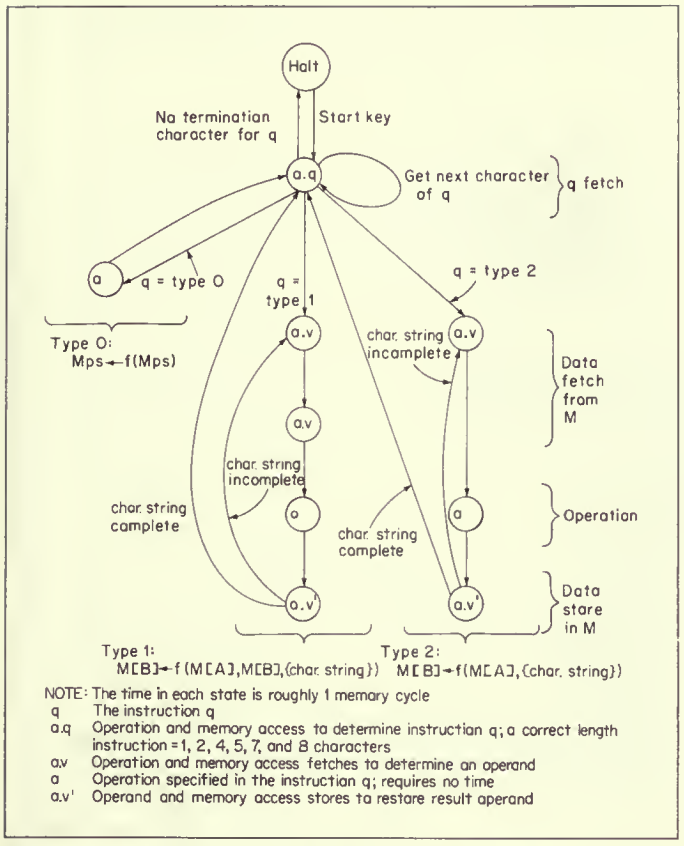

Fig. 3. IBM 1401 instruction-interpretation state diagram.

n − 1]. The values of the string are based on the bcd value of the 8, 4, 2, 1 bits of each digit. The magnitude of the integer is

$$C[n-1] \times 10^{n-1} + C[n-2] \times 10^{n-2} + \cdots + C[0] \times 10^{0}$$

and the sign is

$$\text{Sign} := ((\neg C[0]\langle A'\rangle \wedge C[0]\langle B'\rangle) \rightarrow -;\ \neg(\neg C[0]\langle A'\rangle \wedge C[0]\langle B'\rangle) \rightarrow +)$$

A string is addressed (or accessed) via the A_address or B_address pointer registers. These point to the tail (or least significant digit), that is, C[0], of the string. The instruction-execution state diagram of a variable-string add is shown in Fig. 4. The state diagram assumes that A and B address registers are set up according to Fig. 3. Thus Fig. 4 is a more detailed description of states o.v, o.v, o, and o.v′. Each horizontal pair of states (Fig. 4) corresponds to a single scan of the states of type 1 instruction o.v, o.v, o, o.v′ in Fig. 3. Transitions among states 2 and 3 correspond to the character-by-character scan with string A and B being added together; the result string is placed in B. States 4 and 5 define the string addition, when string A is terminated; i.e., it is considered to be zero. States 7, 8, 9, and 10 define the recomplementation process in which the B string has to be recomplemented. This condition occurs when the operand signs differ, and the A-field result is greater than the B field; the results are in ten's complement form. States 7 and 8 define the B-field scan (to return to find the least digit of B), and states 9 and 10 define the recomplementation of each character. Thus an add operation may require up to three scans of the B string.

The 1401 ISP (Appendix 1 of this chapter) has four parts: State Declaration, Instruction-interpretation process, Instruction-execution process, and Operand address-register calculation process. The Operand address-register calculation process is analogous to the Effective-address calculation in more conventional Pc's and is the most elaborate part of the instruction interpretation. The operand address registers A_address and B_address are part of the Pc state and must be retained between instructions. At the end of an instruction, these registers point to the character of the next lowest data string in Mp, that is, the character at C[n].

Implementation

The 1401 has a small Pc state, and there are only a few registers in the implementations. Figure 5 shows the registers, interregister transfer paths, and data operations that make up the register-

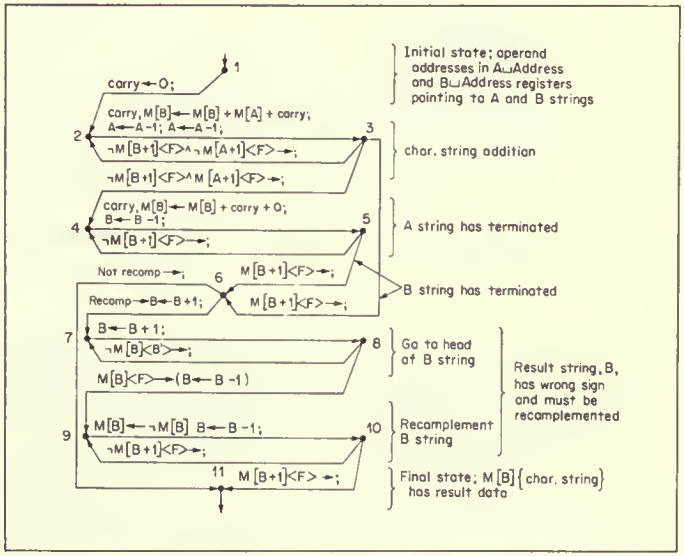

Fig. 4. IBM 1401 add-instruction-execution state diagram.

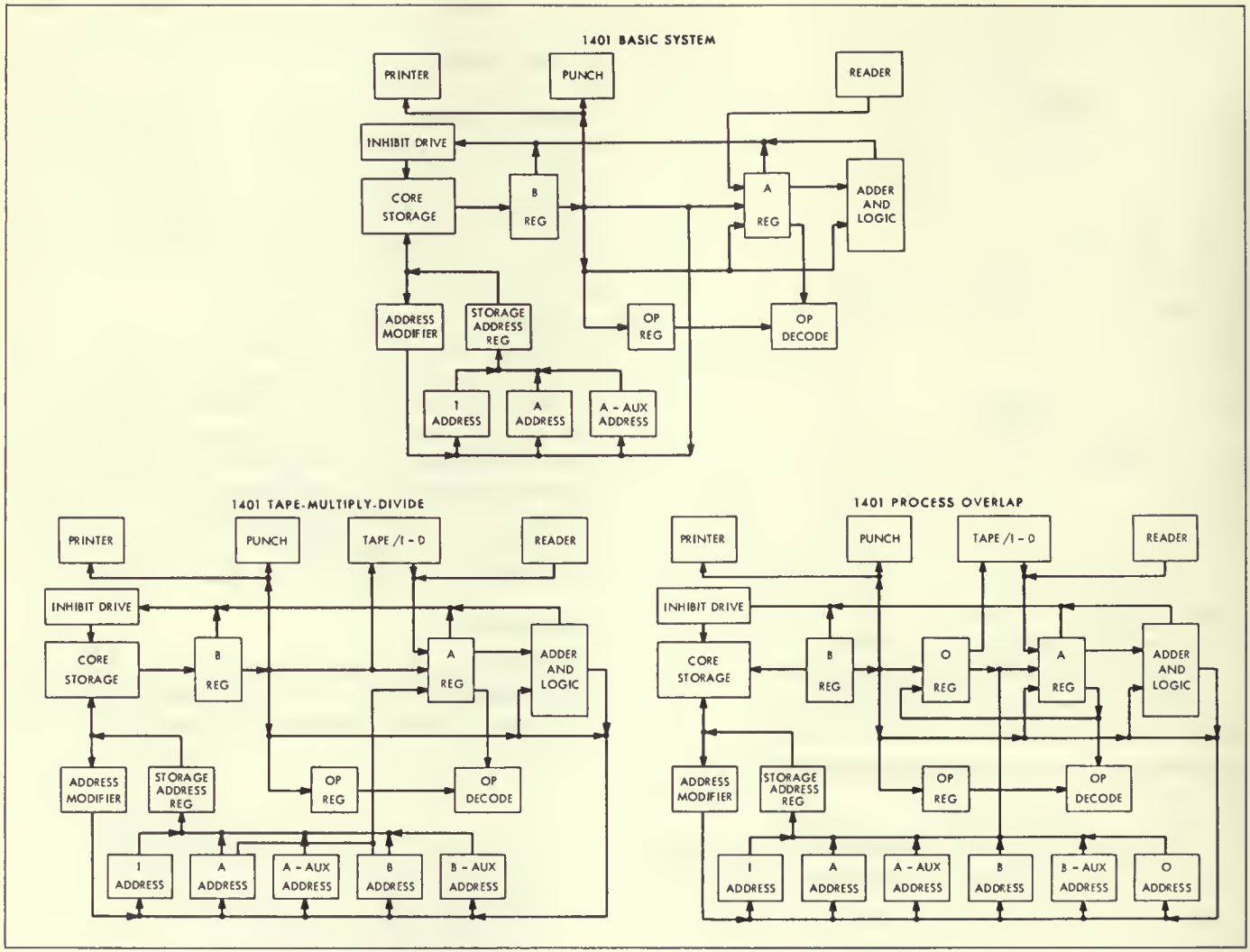

Fig. 5. IBM 1401 system data flow (registers structure). (*Courtesy of International Business Machines Corporation.*)

transfer level primitives of the complete computer together with several options. The options, of course, increase the complexity (and concurrency). Without the overlap feature, for example, all data are accessed in Mp via Pc's address registers.

There are register pairs consisting of a 3-character memory address (access) register, and a 1-character data register. The memory-address, memory-data register pairs are A_address, A_data; B_address, B_data; I_address, Operation/Op; Overlap-_address, Overlap_data/O.

The implementation is straightforward, and the instruction times (Table 2) show the implementation at the register-transfer level. For example, as an instruction is being read by Pc, prior to instruction execution, each new character is taken in and examined for the instruction-terminating flag bit. When the flag bit is present, the instruction is complete and ready to be executed. The character of the next instruction is not saved but is picked up again after the previous instruction has been executed.

APPENDIX 1 IBM 1401 ISP DESCRIPTION

```
                                Appendix 1

                          IBM 1401 ISP Description

Pc, Pc Console, and IO Device Control States
 The following description is a highly simplified description of the IBM 1401.  For example, the edit instruction given below in one
 line corresponds to a three page description in the Reference Manual for the 1401.  It does not include the input-output instruc-
 tions which transfer character strings to fixed blocks of primary memory.  The character strings are denoted as character.string/
 ch.string/ch.s.  For the character.string operations the A␣address/A and B␣address/B registers contain a pointer to the next A and
 B strings at the end of the operations; this aspect of the operation is not described--but implied in the string operations.

   I[1:3]<B',A',8,4,2,1>                          I␣address register, the instruction location pointer

   A[1:3]<B',A',8,4,2,1>                          A␣address register

   B[1:3]<B',A',8,4,2,1>                          B␣address register

   String Data pointer registers A and B point to the least significant digit end of a variable length string in memory (see Mp
   State definition below).  Normally A and B are decreased by one and move to the more significant end for variable length string
   {ch.s} operations.B is normally the result string, and the length is defined by a word mark, F, the last character of the B
   string.  If A string has a word mark, and is shorter than the B string, then the remaining A string is taken to be a zero.  I is
   a pointer to the most significant digit of the instruction.  Although Pc register characters have the B',A',8,4,2,1 bits, the M
   has two additional bits check, and field.  The bits of Mp are:

      Check/Parity␣bit.  The sum (modulo 2) + 1, of the F,B',A',8,4,2,1, bits.

      WM/Word␣Mark/F/Field␣bit.  This bit defines the beginning of each instruction.  The F bit also defines the most significant
      digit (the last digit) of a variable length numeric integer string.

      B',A',8,4,2,1 bits.  A 6 bit character is encoded in these bits.  If numeric data is represented, the 8,4,2,1 bits are used
      as a bcd digit.  The sign is encoded with the least significant digit.  For numeric data, a minus sign, -, is encoded by
      (A' = 0) ∧ (B' = 1).  All other combinations of A',B' represent a plus sign, +.

   XR[1:3][1:3]<B',A',8,4,2,1>1= M[87:89,92:94,97:99]<B',A',8,4,2,1>  3 three character optional index registers stored in Mp


   Indicators[0:63]                               logical bit array encoding Pc State (not including I,A, and B)

   There are a set of 31 status bits of the possible 64.  They can be cleared or set under instruction control.  Some Indicators
   are used by external Pc status or I/O status.  The indicators can be selected for testing by the d character of an instruction.
   The Pc indicators assignment to Pc State is:

      Unconditional := 1                          always a 1

      Sense␣switch<A,B,C,D,E,F,G>                 a set of 7 console keys

      Unequal␣compare                             B ≠ A

      Equal␣compare                               B = A

      Low␣compare                                 B < A

      High␣compare                                B > A

      Overflow                                    set by arithmetic overflow, cleared by a branch instruction if
                                                   it is set

   The indicator array is partially encoded below:

      Indicator[000000] := Unconditional
      Indicator[110001] := Sense␣switch<A>
           ⋮
      Indicator[010001] := Unequal␣compare
           ⋮
      Indicator[011001] := Overflow

Mp State

     M[0:15999]<Check,F,B',A',8,4,2,1>            primary memory

       address[X[1:3]<B',A',8,4,2,1>]<1:5>₁₀ := (  Address encoding for 1 of 16000 from a 3 char value of regis-
                                                    ter X.  Indexing described below.
```

APPENDIX 1 IBM 1401 ISP DESCRIPTION (Continued)

```
            X[3]<B',A'> x 4000₁₀ +
            X[1]<B',A'> x 1000₁₀ +
            X[1:3]<8,4,2,1>{bcd.string})

Instruction Format
   op<F,B',A',8,4,2,1>                          instruction register specifying the operation
   d_char<F, B',A',8,4,2,1>                     additional character used in some instructions
   d_char_present                               indicates a d_char is used in the current instruction
   active                                       indicates an instruction string is still being fetched
   A_address_present                            indicates there is an A address part of an instruction
   B_address_present                            indicates there is a B address part of an instruction
Move, load, and store instruction types control the initialization of A and B.
   move or load or store A or B/mls := ((move characters and edit = op) ∨ (load characters to A word mark = op) ∨ (move characters
   to A or B words mark = op) ∨ (move characters and suppress zeros = op) ∨ (move numerical = op) ∨ (move zone = op) ∨ (store A
   address register = op) ∨ (store B address register = op))

Instruction Interpretation Process
   Run → (op ← M[I]: I ← I + 1; next                fetch operation
          Fetch_operand_addresses; next             fetch addresses for A and B
          Instruction_execution)                    execute

Address Calculation Process
 The 1401 calculates explicit effective addresses by first setting up the A, and B address registers. Operands are not fetched
 in Instruction_execution. There are 1,2,4,5,7 and 8 character instructions which have the op and the following operands
 (respectively): no char, d char, the I or A address, the I or A address and d char, the A and B address, and the I or A address
 and B address and d char. The following process defines the operation for correct length instructions.
   Fetch_operand_addresses := (
   d_char_present ← 0;
   M[I]<F> → (active ← 0);                          1 char instruction
  ¬M[I]<F> → (active ← 1; ¬ mls → B ← 0); next      proceed to get an I or A address
   active → (d_char ← get_char; next A[1] ← d_char;  I or A address set up or d_char
             d_char_present ← 1; next
             ¬ mls → (B[1] ← A[1])); next
   active → (A[2] ← get_char; next ¬ mls → B[2] ← A[2]); next
   active → (A[3] ← get_char; next ¬ mls → B[3] ← A[3]): next
   active → (A_address_present ← 1);                record whether I or A address is present
  ¬ active → (A_address_present ← 0); next
   A_address_present → (d_char_present ← 0;         add index register to I or A
      (A[2]<B',A'> ≠ 0) → (A ← A + XR[A[2]<B',A'>] {3.ch}));
  ¬ M[I]<F> → (B ← 0); next                         B address set up or d_char
   active → (d_char ← get_char; next B[1] ← d_char:
      d_char_present ← 1);
   active → (B[2] ← get_char); next
   active → (B[3] ← get_char); next
   active → (B_address_present ← 1);                record whether B address is present
  ¬ active → (B_address_present ← 0); next
   B_address_present → (                            add index register to B
      d_char_present ← 0;
      (B[2]<B',A'> ≠ 0) → (B ← B + XR[B[2]<B',A'>]{ 3.ch}));
```

APPENDIX 1 IBM 1401 ISP DESCRIPTION (Continued)

```
    (¬ M[I]<F> ∧ active) → (d_char ← get_char;                    final d_char
                           d_char_present ← 1); next
    (¬ M[I]<F> ∧ active) → Run ← 0;                               halt if more than 8 char instruction
                )                                                 end Fetch_operand_addresses
    get_character:
    A sub-process used to fetch each new character in the instruction.  If F is found in a character, the process terminates.
      get_char<B',A',8,4,2,1> := (
        ¬ M[I]<F> ∧ active → (M[I]; I ← I + 1);                   value is present character
          M[I]<F> → active ← 0);                                  no value, terminate
Instruction Set and Instruction Execution Process
    Instruction_execution := (
      character string/ch.s movement and clear memory:
        "M" (:= op = 100100) → (M[B] ← M[A] {ch.s});              move characters to A or B word mark - character string (ch.s)
        "Z" (:= op = 011001) → (M[B] ← M[A] {ch.s}; next          move characters and suppress zeros
                                M[B] ← f(M[B]) {ch.s});
        "L" (:= op = 100011) → (M[B] ← M[A] {ch.s});              load characters to A word mark
        "E" (:= op = 110101) → (M[B] ← f(M[A],M[B], {ch.s}));     move characters and edit
This instruction moves the A field string to the B field string under control of an edit character string in the original B field.
        "/" (:= op = 010001) → (M[B] ← 0 {ch.s.mod.100};          clear storage, ignores the F mark and moves to next modulo
                                                                   100 address
                              ¬ B_address_present → ;             clear storage
                                B_address_present → I ← A);       clear storage and branch
      character string, {ch.s}, arithmetic:
        "A" (:= op = 110001) → (Ov,M[B] ← M[B] + M[A] {ch.s});    add
        "S" (:= op = 010010) → (Ov,M[B] ← M[B] - M[A] {ch.s});    subtract
        "!" (:= op = 101010) → (M[B] ← 0       - M[A] {ch.s});    zero and subtract
        "?" (:= op = 111010) → (M[B] ← 0       + M[A] {ch.s});    zero and add
        "@" (:= op = 001100) → (Ov,M[B] ← M[B] x M[A] {ch.s});    multiply; full length product in M[B], special hardware option
        "%" (:= op = 011100) → (Ov,M[B] ← M[B] / M[A] {ch.s});    divide; quotient and remainder both end up in M[B].
        "#" (:= op = 001011) → (M[B] ← M[B] + M[A] {3.ch};        modify address
                                B ← B - 3; A ← A - 3);
      branches, halt, no-operation:
        "N" (:= op = 100101)→ ;                                   no operation
        "." (:= op = 111011)→ (Run ← 0;
                              ¬ A_address_present → ;             halt
                                A_address_present → I ← A);       halt and branch
        "B" (:= op = 110010) → (
          (¬ B_address_present ∧ ¬ d_char_present) → I ← A;        branch
          (¬ B_address_present ∧   d_char_present) → (             branch if indicator on
            Indicator [f(d_char)] → (I ← A);
            Indicator [f(d_char)] ← 0);
          (B_address_present ∧ d_char_present) → (                 branch if char equal
            B ← B - 1;
            (M[B] = d_char) → I ← A)):
```

APPENDIX 1 IBM 1401 ISP DESCRIPTION (Continued)

```
    "V" (:= op = 010101) → (B ← B - 1;                          branch if word mark and/or zone
                            M[B]<f(d_char)> → (I ← A));
    "C" (:= op = 110011) → (                                    compare
       Indicators ← M[A] = M[B] {ch.s});
subroutine calling:
    "Q" (:= op = 101000) → (                                    store A address register
       M[A - 2:A] ← A[1:3]; A ← A - 3);
    "H" (:= op = 111000) → (                                    store B address register
       M[A - 2:A] ← B[1:3]; A ← A - 3);
single character operations
    "," (:= op = 011011) → (M[A]<F> ← 1; M[B]<F> ← 1;           set word mark
                            A ← A - 1; B ← B - 1);
    "Ø" (:= op = 111100) → (M[A]<F> ← 0; M[B]<F> ← 0;           clear word mark
                            A ← A - 1; B ← B - 1);
    "D" (:= op = 110100) → (M[B]<8,4,2,1> ← M[A]<8,4,2,1>;      move numerical
                            A ← A - 1; B ← B - 1);
    "Y" (:= op = 011000) → (M[B]<8',A'> ← M[A]<8',A'>;          move zone
                            A ← A - 1; B ← B - 1);
          )                                                     end Instruction_execution
```