The mini-computer—A new approach to computer design

by D. C. HITT, G. H. OTTAWAY and R. W. SHIRK

International Business Machines Corporation
Research Triangle Park, North Carolina

INTRODUCTION

When memories become many times faster than they are today at less cost, how will computers be organized? For those installations which require many times their current performance, present methods may suffice. However, there will be installations which don't need or can't use this much additional performance. How can they benefit from such improved memory design?

In an attempt to answer these questions, IBM built an experimental "Mini-computer" which incorporates new techniques and a unique organization. These methods could be used to take advantage of increased memory speed to reduce data flow and control hardware, lowering system cost. Although the memory actually used was neither unusually fast (2 µsec) nor inexpensive, it was adequate to demonstrate principles.

The experimental computer (dubbed Mini) is a general-purpose, stored-program digital computer (Figure 1). However, it differs from such computers in two major respects: it has no arithmetic or logic unit and only a very elementary instruction set; and, it has only 512 bytes of storage for data and control. Mini's other specifications also reflect its name:

- Size—18 X 15 X 3 inches;
- Weight—24 pounds, excluding power supply and I/O;
- Cards—one SLT board (79 cards). If communication line control is desired, three additional cards on an external board are needed.

Although the Mini-computer is contained in one small tabletop "box," its capabilities are surprising to many people. Using a bare minimum of hardware and instructions, Mini can perform the arithmetic and logic of a central processor, plus those functions normally executed by I/O control units. Its attachments include bulk storage, a graphic display (including animation!), a keyboard/printer, and communication lines. Little additional circuitry was needed to attach these I/O units. In addition to the computer, the only hardware needed to control these devices is the analog circuitry required to electrically and mechanically interface with them (magnet drivers, sense amplifiers, voltage converters, etc.).

Finally, the Mini-computer uses only two types of instructions. Functions such as add, subtract, shift, index, and translate are all programmed with these basic instructions.

Organization of the Mini-Computer

The Mini concept starts with a small core storage device (512 8-bit bytes) around which the rest of the computer is built. (See Figure 2.) To this have been added a Storage Address Register and a Storage Data Register. The 8-bit Storage Data Register is funneled down to a 4-bit path going into the Data Register. This was done because of the extensive use of tables and the need to keep addresses compact to conserve storage.

The Data Register is the single "work" register, all data being shuttled back and forth between it and memory. It is used as an input/output register for I/O operations, as a test register for branch instructions, and as a temporary holding place for storage-to-storage transfers.

The Instruction Counter holds the address of the next instruction to be executed. The I/O Reg-
Figure 1a—The Mini-Computer—Without I/O

Figure 1b—The Mini-Computer system

Figure 1c—Mini system block diagram

Figure 2—Mini data flow

The instructions, which have a single address, can be classified into two main categories: move and branch. The hardware-addressed or "implied" Data Register is associated with all instructions. Move operations transfer data into or out of the register, branch operations test data in it. The complete instruction set is shown below.

**Move Instructions:**

- FETCH—Data at the addressed storage location is moved to the Data Register;
- STORE—Same operation as above, except in the reverse direction;
- INPUT—Data from the addressed input device is moved to the Data Register;
- OUTPUT—Data is moved from the Data Register to the output device.

**Branch Instructions:**

- BRANCH ON ZERO
- BRANCH ON NOT ZERO
- UNCONDITIONAL BRANCH
- BRANCH AND SAVE—The address of the next sequential instruction is saved, and a branch occurs to the specified storage location;
- RETURN—A branch occurs to the storage location saved by the last BRANCH AND SAVE operation.

One of the two conditional branches is obviously redundant. However, having both is a convenience which aids programming and frequently conserves core space. The BRANCH AND SAVE instruction and associated RETURN permit subroutining to be performed with a minimum of programming overhead. As will be seen, subroutines play a very important role in the Mini-computer.

The basic unit of data operated on by the instruction set is the 4-bit half byte. Instructions themselves are either 1 or 2 bytes in length, with the 1-byte format accounting for roughly 70% of the instructions in a typical program.

Instruction formats

Two instruction formats are employed in an effort to reduce storage requirements. The long format uses a 6-bit operation code and a 10-bit address capable of addressing any of the 1,024 half bytes of core storage. Short-format instructions use a 3-bit op code and a 5-bit address, with the remaining five bits implicit in the operation code.

Thus, because a large percentage of STORE instructions is used to modify other instructions in the immediate vicinity of the store instruction, short-format store addressing can be relative to the Instruction Counter. That is, the remaining five bits of address are supplied from the high-order positions of the Instruction Counter. The same is true of short-format branches.

For short-format FETCH and BRANCH AND SAVE instructions, the required address bits are hard-wired into the computer, causing fixed storage areas to be accessed. These areas contain tables in the case of the FETCH instruction, or subroutine linkages or short, self-contained routines in the case of BRANCH AND SAVE. The short-format instructions make possible significant savings in core because, as stated earlier, they account for about 70% of all of the instructions used in a typical Mini program.

Circuits

The Mini model was constructed from readily available System/360 Model 50 circuit cards employing the Solid Logic Technology (SLT) 30-nanosecond circuit family. The memory unit chosen was a 2-microsecond, read-write core buffer used in the System/360 Model 20. No functional packaging or large-scale integration was attempted. The machine is packaged on a single SLT board and contains approximately the following 650 circuits:

<table>
<thead>
<tr>
<th>Function</th>
<th>No. of Circuits</th>
<th>No. of Cards</th>
<th>Percentage of Total Cards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Flow</td>
<td>250</td>
<td>36</td>
<td>42</td>
</tr>
<tr>
<td>Storage</td>
<td>160</td>
<td>20</td>
<td>31</td>
</tr>
<tr>
<td>I/O</td>
<td>240</td>
<td>23</td>
<td>27</td>
</tr>
</tbody>
</table>

Calculation and data manipulation

Because Mini has no arithmetic or logical instructions, these functions are performed by subroutines. Core requirements for these are moderate, as shown in the following table of some of the subroutines which have been written. Of course, many more could be added.

<table>
<thead>
<tr>
<th>Function</th>
<th>Bytes in Subroutines</th>
<th>Bytes in Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Counter (initialize,</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>increment/decrement)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control binary trigger (set,</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>reset, flip)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control status switch (initialize,</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>set, reset)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Bit</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Translate Code</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Delay</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Increment Address</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Add/Subtract (decimal digit)</td>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>Compare (binary digit)</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Shift (binary digit)</td>
<td>9</td>
<td>16</td>
</tr>
</tbody>
</table>

Two programs have been written to show the arithmetic capability inherent in the Mini data flow and instruction set: the desk calculator and triangle calculator program. The desk calculator program accepts data from the 16-key keyboard and produces two lines of 14-digit output on the display (described below.) The first line displays
entered data, the second displays the results of arithmetic operations on the data. The functions provided are add, subtract, multiply, load, and no-op.

The triangle calculation program accepts data from the keyboard and produces up to six lines of 3-digit output on the display. The program starts by displaying a labeled triangle (Figure 3).

![Figure 3](https://example.com/figure3.png)

**FIGURE 3**—The triangle calculation program

The user keys in the specifications for the known angles and sides and these values are displayed on the appropriate lines. The program then computes the remaining values and displays them to complete the picture.

Since the triangle program is larger than the storage capacity of Mini, it requires successive loads from a bulk-storage device. Each load does a particular type of processing, such as calculating a sine, cosine, arcsine, or arc-cosine by a power series expansion or by a solution of the law of sines or cosines.

**I/O unit control**

**Bulk storage**

A standard IBM Executary dictating unit is controlled to perform functionally as a disk file. Labeled records are recorded on the belt in a fixed format, serial by bit. Synchronization, bit detection, deserialization, parity checking, label comparison, and data movement are all programmed, using 120 bytes of core. The special hardware required to attach the Executary was five cards.

When the Executary is used as a dictating machine, the user controls the machine through switches on the microphone. For use with Mini, the microphone was removed from its cable, and the wires that had gone to the microphone switches were connected to reed relays in the computer. This enables Mini to perform the same three functions by issuing I/O commands:

- Start moving the belt.
- Backspace one track.
- Turn on the write circuits.

Because the random-access, bulk storage used (Executary) is designed to record in the normal audio range, bits are recorded as audible beeps. Each “one” bit consists of three cycles of a 2K Hz tone; each “zero” bit three cycle times of silence.

Mini generates these audible tones with a program to eliminate the need for a separate 2K Hz oscillator. The oscillator program activates the output line and enters a subroutine which provides a delay of 250 usec, then it discontinues the output for 250 usec, activates it for another 250 usec, etc., for three oscillations. The recording format on tape is similar to standard start-stop communications format. Records have labels and markers to enable Mini to locate the beginning of the desired record, and a check character for validating data. The recording rate used with Mini is 40 bytes per second. Speed is limited by the Executary belt and amplifier characteristics, not by the computer. Data rates of 500 bytes per second have been achieved using a different tape transport and amplifier.

**Display**

The objective of the display investigation on Mini was to demonstrate the maximum versatility for drawing pictures on a simple CRT display with the least amount of additional circuitry. (See Figure 4.) The circuitry, packaged on only three cards, provides for two 8-bit digital-to-analog converters, two 8-bit deflection registers, and standard output commands for loading these registers from the data flow. In addition, there is control for blanking the beam.

The 8-bit horizontal deflection register specifies on which of 256 horizontal positions the beam is to come to rest. An identical vertical-deflection register similarly selects one of 256 vertical positions. For example, if both registers contained the value of zero, there would be a spot at the lower left corner of the tube. If both registers were then set to 256 simultaneously, the beam would travel to the upper-right-hand corner tracing a line along the way. Thus, both horizontal and vertical registers must be changed simultaneously if other than horizontal and vertical lines are to be drawn. The hardware-function tradeoff selected allows the upper four bits of both horizontal and vertical
FIGURE 4—Demonstration of display versatility

registers to be set together, and the lower four bits to be set together. This provides the capability of connecting points on either a large or small 16-by-16 point grid.

A program was written to generate an alphanumeric display. The most challenging problem associated with the program was developing a subroutine for tracing each of the individual characters—a function which is generally provided by special-purpose hardware dedicated to generating a specific set of characters. For each character to be drawn by the program, a string of elements is provided. Each element defines a new point on the display. As the points of the string are successively connected, the desired character is drawn. To provide the most general character generator, each element of the string requires 8 bits, four bits for one of 16 vertical positions and four bits for one of 16 horizontal positions.

An alternate arrangement is to provide a more restricted array of end points so that each element can be specified with fewer bits. Figure 5 illustrates one such restricted set of points. Figure 6 shows the character set obtained. This set of 15 points can be specified in four bits, which reduces the core requirements of the character generator by a factor of two.

A program was written to accept characters from a keyboard and display them onto the face of the tube. The character set consisted of the 26 capital letters and 10 digits. A cursor was provided to indicate where the next key to be depressed will be displayed. The space key moves the cursor incrementally along the line and the carriage-return key returns the cursor to the left and down one line. A reset returns the cursor to
FIGURE 7—Mini keyboard interpreted for drawing program

the upper-left-hand position of the tube. Only 352 bytes were required for the programs and tables for this program.

A program was also written to permit an operator to draw pictures on the tube. This program interpreted the general purpose, 16-key keyboard as shown in Figure 7. By depressing one of the arrowed buttons, the operator can move the last end-point of his picture one increment in the direction indicated by the arrow. When the operator is satisfied with the position of the last point, he can freeze that point by depressing the load key. This stores the coordinates of the last point and creates another moveable point at the same location.

The forward and reverse buttons allow the operator to modify previously fixed points. Up to 16 different pictures containing a total of 128 end-points can be stored in the available buffer space. Animation can also be obtained by rapidly flashing the 16 pictures. This program requires only 256 bytes of program and 128 bytes of buffer.

Printer/Keyboard (IBM 1052/1053)

The Mini-keyboard interface consists of six Binary Coded Decimal data lines, a “key depressed” signal line, and a “keyboard reset” line to reset the keyboard between key depressions.

To accept data from the keyboard, the Mini program tests the “key depressed” line. When the “key depressed” line is up, the program enters a delay subroutine to allow the data lines to settle to a steady state. Then, the program transfers the information on the data lines to storage and issues a reset signal to prepare for the next key depression. This sequence requires only nine bytes of storage in addition to the delay subroutine.

The Mini-printer interface has six data lines which position the print element; seven control lines corresponding to the functions of print, shift up, shift down, tab, carriage return, space, and backspace; and a “carriage in motion” line. The positioning information specified by the program is held in a special output register for the duration of a delay subroutine.

To use the printer, Mini raises a control line and holds it active for the duration of the delay subroutine. For the print function, the program first transfers the proper combination of bits for the character to be printed from storage to the output register. For tab or carriage return, the program tests the “carriage in motion” line and waits until it is no longer active before proceeding.

The Keyboard/Printer requires five cards.

A program to couple the 1052/1053 keyboard and printer together for normal typewriter use has been written. This program accepts data from the keyboard and immediately outputs it to the printer so that the data keys print and the function keys perform as in a normal typewriter. As an intermediate function, the program translates the BCD input code to the required output code before setting the output register. The program requires 54 bytes in addition to 45 bytes for subroutines and tables.

Sixteen-Key Keyboard

The keyboard used on Mini contains 16 general-purpose keys. Four bit-line outputs are available to the program, as well as an output directly from the zero key. The separate zero-key output, coupled with some additional programming, eliminates both extra keyboard hardware (no second contact needed on each key to indicate a key is depressed) and extra circuitry (no keyboard strobe).

To control this keyboard, Mini first tests the four data-line outputs. If no information is available, the program tests the zero-key output. If all lines are inactive, the program then repeats the above tests. If any data line or the zero-key line is active a delay is taken to insure steady-state values on the lines. The information is then trans-
ferred to storage, or if the zero key is active, a zero is inserted into storage.

The program requires 15 bytes in addition to the delay subroutine.

Communications

A standard common-carrier data set was attached with three cards. A program was written to enable communication between Mini and an IBM 1050 terminal.

Programming

Address modification is used extensively in programming Mini. The following example illustrates its usefulness:

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>FETCH</td>
<td>Increment Table</td>
</tr>
<tr>
<td></td>
<td>STORE</td>
<td>x + 1</td>
</tr>
</tbody>
</table>

The first instruction is a short-format FETCH with op code at half-byte location x and address at half-byte location x + 1. This instruction addresses an 8-byte increment table at a fixed location in storage. The position in the table is determined by the address at location x + 1. The table is as follows:

<table>
<thead>
<tr>
<th>0123456789ABCDEF</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0123456789ABCDE</td>
<td>Data</td>
</tr>
</tbody>
</table>

The second instruction is a short-format STORE. It inserts the half-byte fetched by the previous instruction into the instruction stream at location x + 1, thereby modifying the address portion of the FETCH operation.

Assuming the half-byte at x + 1 is initially "0", the above sequence fetches a "1" to the Data Register, then modifies itself to fetch a "2" next time. Now, using the BRANCH ON NOT ZERO instruction, a loop can be programmed:

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>FETCH</td>
<td>Increment Table</td>
</tr>
<tr>
<td></td>
<td>STORE</td>
<td>x + 1</td>
</tr>
<tr>
<td></td>
<td>BRANCH</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>NOT ZERO</td>
<td></td>
</tr>
</tbody>
</table>

The BRANCH ON NOT ZERO instruction tests the Data Register and branches back to the start of the sequence as long as the Data Register contains a non-zero quantity. This will be the case as long as the half-byte at location x + 1 is not an "F." Once this becomes "F" the FETCH instruction will load "0" into the Data Register and the branch will fail. Assuming a "0" in this location initially, the loop will be executed 16 times with successively higher numbers being loaded into the Data Register.

The table used in this example is one of two which can be accessed with the short-format FETCH instruction. The other table is organized for decrementing and is as follows:

<table>
<thead>
<tr>
<th>0123456789ABCDEF</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0123456789ABCDE</td>
<td>Data</td>
</tr>
</tbody>
</table>

The counting example could just as easily have been programmed using this decrement table. It would then count through the loop 16 times but would load successively smaller numbers into the Data Register.

The programming techniques used with Mini will not be described in detail. However, an example of the use of stored tables and the basic instruction set will clarify the manner in which processing is accomplished. The following routine performs binary addition on two half-bytes:

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>FETCH</td>
<td>Increment Table</td>
</tr>
<tr>
<td></td>
<td>STORE</td>
<td>x + 1</td>
</tr>
<tr>
<td>y</td>
<td>FETCH</td>
<td>Decrement Table</td>
</tr>
<tr>
<td></td>
<td>STORE</td>
<td>y + 1</td>
</tr>
<tr>
<td></td>
<td>BRANCH</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>NOT ZERO</td>
<td></td>
</tr>
</tbody>
</table>

Let us assume that the two numbers to be added have been loaded into locations x + 1 and y + 1 in the instruction stream. The first number is incremented, the second decremented and tested for zero. This sequence continues, the first number being counted up and the second counted down, until the loop is ended by the second reaching zero. At this point the sum is at location x + 1.

The increment/decrement tables used in the
previous examples are the most commonly used in Mini and are therefore stored at fixed locations addressable by the short format FETCH instruction. Other tables, however, are constructed for special purposes and are accessed by the long-format FETCH. Examples of such tables are those used for code translation, shifting, decimal incrementing and decrementing (or of any other base, for that matter), bit testing, overflow detection, masking, etc. Other examples will no doubt suggest themselves to the reader.

From this discussion, it is obvious that Mini is a table-driven system. Because of this reliance on stored tables for basic data manipulation and because of the very limited storage area available, it was imperative that tables be kept as compact as possible. This explains why half-byte addressing is used with storage. Four-bit table addresses allow the use of 8-byte tables such as the increment and decrement tables described previously. Had storage addressing been at the byte level, these tables would each have required 128 bytes!

CONCLUSIONS

Mini, a small, experimental computer with a surprising amount of capability, has been built around only two classes of instruction—move and branch. Arithmetic and logic functions are performed by table-look-up subroutines. Digit addressing makes it possible to use much smaller tables than ordinarily used, conserving memory.

No additional instructions or hardware are necessary to operate the computer. Mini has thus demonstrated that it is possible to build a computer with no arithmetic or logic hardware! Further hardware savings—in memory—have been made by using one-byte instructions (part of address being hard-wired or implied by operation code).

In the I/O area, Mini has shown that special-purpose hardware can be replaced by general-purpose data flow and that—once you have the data flow—little additional circuitry is needed to control I/O devices. The Mini-computer, because of its inherent simplicity, makes it possible to approach an I/O task—often quite complex—as directly as though it were a CPU operation.

Mini is smaller than most of the terminals and I/O units used with present-day computers, even though it wasn’t built from miniature components. Mini may have indicated the shape of computers to come.