U. S. AIR FORCE
PROJECT RAND

SOME THOUGHTS ON THE IBM MILITARY COMPUTER AS A DATA PROCESSOR

C. L. Baker
J. P. Haverty

B-100

February 6, 1959

Assigned to ______________________

The RAND Corporation
1700 MAIN ST. • SANTA MONICA • CALIFORNIA

SOME THOUGHTS ON THE IBM MILITARY COMPUTER AS A DATA PROCESSOR

I. INTRODUCTION

The IBM Military Products Computer is described in the Kingston Military Products Division Document SAC-11, dated 17 November 1958, entitled, "Program Instructions Specifications for the SAC Data Processing System." A somewhat revised description has appeared as Lincoln Laboratory "Advanced Sage Computer Memo #23," dated 18 November 1958. The general characteristics of the machine are as follows: It is a solid state machine, utilizing 10 megapulse circuitry, with a high-speed core storage capability of up to one-quarter million (in units of 16,000) words. A 48-bit complement word form is used; memory access time is to be 2.4 microseconds, with instruction overlap permitting a maximum instruction rate of approximately 400,000 per second.

The computing power of such a machine is limited by two factors: first, the hardware design, and second, the order code or internal organization of the machine. The hardware is given and is presumably a state-of-the-art development. The order code and the machine structure should also represent an advanced development. The following is a discussion of some of the current thought on machine structure. In this discussion, cost constraints were not considered.

II. SUBSYSTEMS OF THE MACHINE STRUCTURE

In an operational environment, a data processing machine such as the IBM Military Products Computer, has four general subsystems: (1) central processor, (2) auxiliary input-output equipment, (3) data input-output equipment including displays, (4) operating equipment, such as control and maintenance consoles. Each of these subsystems must be properly designed for use in the system as a whole. Considerations here include high-speed data processing capability, ease of program updating and program correction, coding convenience, and ease of operating the machine. In the design process, it is easy to fall into a trap of the following nature: Any machine which has been in use for some time leads to techniques for handling data and performing computations which are a result of the limitations of the machine and which are not relevant to the over-all data processing problem. These methods are soon presumed to be an inherent part of the problem environment, and a new machine is designed to perform these techniques rapidly and cleverly. Clearly, we should be searching for more appropriate methods of data processing rather than better ways of implementing outmoded methods.

III. CENTRAL PROCESSOR

The following remarks are primarily directed towards the central data processor. The core memory is the heart of the processor, since data and instructions are stored here and the limiting speed of the machine is usually dependent on the core cycle time. This implies that the memory addressing scheme employed to refer to data and instructions is of the utmost importance. Although the data stored within the memory is variable in length, a fixed word size is necessary so that the storage may be interrogated in parallel to gain an obvious speed advantage. Much expensive memory space is wasted if information is not packed into these fixed length words. Input-output considerations also lead to the desirability of packing data within a word. Hence, some means must be supplied in order to obtain variable length data from the storage.

A. Addressing (Bit vs. Byte)

There are two ways to obtain data which is packed within the core storage: (1) unpack using logical operations or, (2) the hardware itself does the unpacking, via some addressing scheme. The first way confuses the operations necessary to operate on the data with those operations necessary to obtain the data itself. The programmer must continually be aware of the fixed word length of the machine, and these unpacking operations may actually consume a significant portion of the available computation time. To date, most machines have been constructed using this method for obtaining data which is packed in storage.

In considering how the hardware may be made to do the packing and unpacking, an obvious way seems to be to specify the location of the first bit of the information in memory and the number of bits to be read from storage. Both of these parameters must be variable, preferably without re-programming as the nature of the data changes. This specification should therefore change with the data and not be fixed in the instructions. Indexing and indirect addressing features will accomplish this, as suggested below. The length of data obtainable should be variable up to many bits, the maximum number being set by the word length of the machine and cost considerations. If two consecutive words are fetched at once, from two storage boxes (by inter-weaving core addresses between the two boxes) word boundaries need not be significant.

Individual bit addressing implies an address field equal in length to the power of 2 required to address the number of bits in the store plus the power of 2 required to address the number of bits in the word length. Each word should be a-power-of-2 bits long, capable of retaining adequate arithmetic significance and all necessary operation, address, and modifier bits. A 64-bit word machine with a quarter million words store therefore requires ~~24~~ 18 plus 6, or ~~30~~ 24 bits for each address. Indexing is essential only for the starting bit of a field; the length of the field, however, should be at least specifiable indirectly, if not indexed. Combined with indexing and indirect specification, this scheme of addressing permits complete flexibility in data, location and format without changing any instruction addresses.

Because of cost or hardware considerations, the above scheme may be impractical or undesirable. As a possible compromise, we need not refer directly to each bit in memory, but rather refer to individual "bytes" of memory, each byte, for example, being a 6-bit length of storage. Although complete flexibility of bit addressing has been discarded, a large capability still remains, since the greater portion of the data may well consist of either 6-bit characters or of units of information which can be stored in 6 bits or less. It is sufficient to address the first byte of a string of bytes plus the number of bytes. We must also have a capability for referring to bits within a single byte. For a 48-bit word machine (2^3 bytes per word) with a quarter million words, an address field of 18 plus 3 plus plus 3, or 24 bits would be necessary. To refer to bits within a byte, a mask 6 bits in length could be used. This mask should be obtained from a specified register, and not be placed in the instruction.

The least desirable way of addressing memory is to refer to a word only and to specify the field within a word by using a mask. Here many masks are needed for flexibility, and it is usually more economical to unpack and repack by means of serial operations.

The Military Products Computer is really of the third type. Only words are addressable, and special modifiers are specified in each instruction dealing directly with data. These modifiers are not part of the address, since there are no modifiers or indirect specification ability of these. Actually, the modifiers are really instructions to perform a highly specialized set of serial operations. Also, word boundaries become significant barriers in the addressing scheme.

In order to be most useful in an instruction, an address must be modifiable so that, at a minimum, the same instruction can operate on related data from various sections of the core storage. Indexing provides the fastest way to do this and has the advantage of being dynamic, i.e., not interpreted until the actual instruction execution time. Indexing also leads to nonchanging instructions within a program, which is extremely valuable for debugging and for permitting recursive routines.* No exact recommendations can be made for the number of index registers required. If there are too few, many instructions must be executed in order to load and store the data in the index registers. If there are too many, the problem of keeping track of and storing the information in the registers during an interrupt is of significance. The number should be one less than a power of 2, however, and either 7 or 15 index registers would seem to be adequate.

*A routine which in the process of execution can call upon itself to be executed.

There should be a bit external to the address to indicate 0-level addressing, i.e., the quantity in the address field is the data, not its address.

To further increase the power of the addressing of the machine, indirect addressing should be supplied. This permits changing data references of a large number of instructions by a single operation. There are two cases of indirect addressing: (1) the address plus the contents of a specified index register "points" to the cell containing the real address; (2) the address "points" to the cell whose contents are then modified by the contents of the index registers to become the effective address. A single bit within the instruction is needed to indicate indirect addressing and a single bit to indicate which type of indirect addressing is to be used. It is also desirable to have the word "pointed to" interpreted in the same fashion. It is highly desirable to have the ability to relocate routines within core storage, without modification. An indicator bit to modify the final address by the contents of the location counter is therefore extremely useful. This again permits recursive routines and permits some assembly features to be delayed until actual execution time of the program.

If indexing is not effective on the field size portion of the address, it should be possible to specify the field size indirectly. In a single address machine, a full address field is not available for this, so a special register might be referred to (by means of a single bit) to obtain this information.

The addressing scheme, once determined, sets the general flavor of the machine as it appears to the programmer. The context of this paper from now on will be that of the second type of addressing; i.e., byte addressing, as this seems to be a good compromise. Most remarks, however, apply equally well to bit addressing.

B. Processing the Data

Once able to get at the data stored in memory, it must be processed. There are several required types of processing ability: (1) arithmetic computation, (2) arithmetic and logical comparison of data, setting and testing of logical conditions, (3) control operations, including looping, subroutine execution, etc., (4) transfer of information.

With complete addressing capability available, only a relatively few operations are needed for each type. Within the arithmetic class, add, multiply and divide will suffice, with modifiers to take absolute and complement forms of each operand if desired. With a fixed arithmetic register length, complement machine, a bit is needed to indicate sign extension.

Split word arithmetic capability is not necessary if serial processing is adequate. However, parallel processing, by means of split words, might significantly increase operating speed depending upon the nature of the job; the ability to process vectors directly, for instance, may be a large enough percentage of the over-all job to justify the extra cost. Nevertheless, this type of specialized processing should appear as special cases of the machine structure rather than the normal form of the machine.

Specifically, we believe that the Military Products Computer should be structured as a full-word machine with a small class of special instructions to handle split-word processing. The proposed configuration is "natural" for only a small percentage of the SAGE routines and may not be "natural" for <u>any</u> of the SACCS routines. Moreover, how sure are you that the future data processing tasks for this machine will justify this specialized structure?

Floating point operations should be considered if a significant amount of scientific computation is contemplated.

Among the various conditional test operations required, comparison is the most important. The ability for testing equality or inequality, relative magnitude, and testing for data lying within a given range is necessary. It is not clear whether it is more useful to set an indicator to indicate successful or unsuccessful comparison, or to actually transfer control to a different sequence of instructions. Either is probably sufficient. Within the logical operations, various types of shifts and masking ability are not so important, given the addressing flexibility proposed, but sufficient power should be provided to handle unusual cases.

Various other operations are necessary for controlling the sequence of instructions within the data processor. Some of these functions overlap with the comparing instructions, and the two sets should be made compatible. These operations are of two types: alteration of the instruction flow, and break-out-of-flow of instructions with subsequent resumption of this flow. Indexing instructions are most important in altering flow of instructions, since a high frequency of execution is indicated. For general indexing, three parameters generally are needed: an increment, a test value and an alternate instruction address. In a single address machine, compromises are usually made to get speed. For example, the decrement and the test value are set equal and specified directly within an instruction, while the transfer address is given the full addressing capability. Much more desirable for working on varying amounts of data (and to make instructions independent of data format), is the ability to refer to the increment and the test value as we do data; i.e., indirectly. Skip or no-skip can provide alternate flow paths. For very "tight" loops, where the extra instruction time required here becomes important, a "repeat the next n orders" instruction, where n is small, is desirable; the address would "point" to the increment number of cycles. The usual conditional transfers may be employed for making two-valued decisions. Here, testing the condition of data is conventionally implied, and the alternate instruction address is spelled out; however, the reverse case is equally important. For example, "point" to a bit or byte to test and skip conditionally with, perhaps, setting or resetting ability at the same time.

Special hardware features should be built into the machine to permit a breakout from a sequence of instructions, as is necessary for execution of a subroutine. There are several ways of accomplishing this, all of which save the current location and transfer control to another location. This should be accomplished so as to permit many levels of subroutine being used automatically (and recursively) by a routine. Each instruction should have the ability to return to the previously executed sequence. (It would be effective only when a conditional transfer, if called for, is unsuccessful). A similar scheme is necessary to handle interrupts due to machine errors or to external signals to the computer.

Finally, the usual instructions should be available for transferring data between memory and high-speed registers, indicators, etc., and between registers.

It should be noted that the possibilities above represent a minimal set of instructions and are all concerned with the basic operations desirable on data, not operations necessary to get at the data. Special purpose instructions are desirable only if they fulfill the need for performing often repeated operations--binary-decimal conversions, for example. In any event, the final instruction list should be constructed on a general need basis, augmented when necessary for speed. In particular, the instruction list should not be comprised of instructions which were found useful on another machine with different design philosophy. A total of less than 64 distinct orders, with a resultant 6-bit operation field, seems sufficient.

The Military Products Computer apparently will have on the order of 256 instructions since an 8-bit operation is specified. We question whether any programmer can efficiently use an instruction list of this length.

IV. Auxiliary Input-Output Equipment

Auxiliary backup storage for the main memory is required, since the core store is undoubtedly not big enough for all data, and files to be processed may become very large. Programming state-of-the-art is not yet far enough along to make detailed recommendations in this area, and only a few remarks are possible. Operation of the auxiliary store should not slow down the central processor, but must be capable of obtaining control information and signalling the main computing element when the input-output has been completed. The entire memory must also appear homogenous to the programmer; i.e., there should be no special addresses associated with input-output functions. The bit rate should be very high and parallel word transfer is thus desirable rather than a bit or byte mode of transfer. Several types should be considered; tapes for long permanent files, drums for backup when memory overflows, and disc storage for large random files, and these auxiliary storages must be capable of simultaneous operation. The card input-output equipment, including card readers, punches and printers, should also be included in this category because of the possibility of tape buffering. The input or output unit should not look different to the main processor in the two modes of operation.

What little we have heard about this portion of the Military Products Computer has not been encouraging. One impression we have is that under certain type of input-output operations, such as tape-to-tape, the instruction rate of the central processor may be drastically reduced.

V. Data Input-Output Equipment

In communicating with the outer world, data is sent to and received from the computer in forms other than that suited for internal representation in the computer. For example, formatted messages and keyboard inputs are available as inputs, and tabular displays are required as outputs. It is necessary not only to buffer the information being inputted and outputted, in order to achieve speed, but it is also necessary to process this data for translation. In display, for instance, this processing must be done either by the central processor or by a computer on the display end of the link. The processing is apt to slow down the main computer unnecessarily if it is done there, and special editing instructions should be supplied if this is the case. These cannot be specified without detailed knowledge of the particular input or output link under consideration, but it should be done or the central computer will end up doing a job it wasn't designed to do.

Alternatively, one might consider an input-output computer which could perform the editing tasks and control the buffering. There appear to be many advantages to partitioning off the data input-output functions. One immediate advantage is that the development of the two-computer routines are much more independent of each other than if both routines must share the same computer. A possible significant advantage is the employment of the input-output computer to select or screen the input-output data as it is buffered. This "selective" control of input-output data can be a powerful tool in a large-scale data processing system.

We have no comments on this portion of the Military Products Computer since we have no information on its input-output configuration.

VI. Operating Equipment

Operating features include the console and any other direct communication with the machine by the programmer or operator and the maintenance engineers. The functions of operating the machine and maintaining it should be considered as separate functions. The design of the operating equipment should emphasize that the programmer-operator portion be as simple as possible and display only the direct information necessary in the operating scheme that is used, while the maintenance portion should be organized to the needs of the maintenance engineer.

The FSQ-7 is an example of merging into one console the two functions mentioned above. On the basis of our experience and feedback from users of the FSQ-7, we feel this merger should be avoided if possible.

VII. Summary

The above discussion has admittedly glossed over what is the most important part of the data processing system--the input-output system--although this is the area in which proper design can have enormous payoffs in speed, convenience and flexibility.

There are three reasons why we touched only briefly on this area: first, no description of proposed input-output equipment had been available for comment; second, the programming state-of-the-art is not too far advanced in this area; third, input-output is highly interrelated with the system context. Nevertheless, as much analysis as is possible should be attempted before the final specification of this area.

The central computer was stressed because this is the area where we had the most information and where the state-of-the-art is more highly developed--not because this is the most profitable area to examine.

It should be emphasized that in the interest of presenting a spectrum of ideas on machine structure, cost constraints were not considered.