Copy # 26

Division 6 - Lincoln Laboratory
Massachusetts Institute of Technology
Lexington 73, Massachusetts

SUBJECT: DESIGN CONSIDERATIONS FOR AN EXPERIMENTAL COMPUTER

To: R. R. Everett

From: W. A. Clark

Date:

Approved: _____

Abstract: A preliminary logical design for a real-time-control computer, with system capabilities approximating AN/FSQ-7, eliminates buffer storage and centralizes control of input-output transfers. High-speed transistor circuits and a random-access storage system of 2.5 million to 5 million bits make possible significant simplifications in system logical design. Breakpoint operation similar to that of DYSEAC, but using many program counters, promises great flexibility in the handling of terminal equipment. This memorandum deals primarily with the features of the multiple-program-counter system.

Introduction

The design proposed here (TX-1) is the first member of a family of parallel, single-address computers for real-time control now under study in Group 63. This family has the following principal characteristics:

1.) Surface-barrier transistor circuitry
2.) Large core memory (2.5-5 million bits)
3.) Multiple program counter logic.

The TX-1 has been designed with the system capabilities of the AN/FSQ-7 in mind but with no attempt to meet the detailed specifications of the SAGE system. It is essentially a "bufferless" machine as opposed to the FSQ-7 which provides buffer drums between the central machine and various terminal devices. In the TX-1, information passes more directly into and out of the memory unit of the central computer. All transfers of information are programmed by means of minor sequences of read-in or read-out operations which are executed on a "demand" basis during interruptions of the major program or of one another. This method of getting information in and out of the central computer, which requires the use of an additional program counter for each minor sequence, is the distinctive feature in which the system logical design of the TX-1

The research reported in this document was supported jointly by the Department of the Army, the Department of the Navy, and the Department of the Air Force under Air Force Contract No. AF 19(122)-458.

differs significantly from that of the FSQ-7.

This note deals primarily with the features of multiple program counter operation of the TX-1 in a control application with roughly the input-output requirements of the SAGE system. Where possible, correspondence with equivalent features of the FSQ-7 will be pointed out. However, not enough is known yet about the details of the transistor circuit "building blocks" to permit the drawing of block diagrams of the TX-1 in anything approaching final form. It is hoped that this summarization of some of the early thinking of the Group 63 Logical Design Section will serve to stimulate further thought and comment.

Influence of Large Core Memory on Design

The fact that large core-arrays (256 x 256 x 36) are now feasible for use in the central memory of the computer makes possible two major simplifications in a system of FSQ-7 proportions which will be realized in the TX-1:

1.) Consolidation of Auxiliary Storage

In the FSQ-7, the storage of the program and associated tables is distributed over 8 physically-independent yet logically-equivalent drums. Consequently an obvious logical simplification results from the lumping together of this storage into fewer large core-arrays. One 256 x 256 array is the equivalent of about 5 or 6 drums.

2.) Elimination of Separate Intermediate Buffer Storage

In the FSQ-7, one drum serves as a buffer of input information; another 3 drums, output information. Of these output buffer drums, 2 hold information for the display system and enable it to run at a higher rate by redisplaying old information several times between changes of data. All of this buffer storage can be eliminated if provision is made to address the central memory on an "in-out break" basis. In the case of the display system, the central memory must give up additional cycles for the redisplay of old information, but this requirement is not a limiting one at FSQ-7 rates.

The total storage capacity of the FSQ-7 including drums and its internal core memory is about 4.5 million bits, almost entirely in 32 bit words. This is roughly the equivalent of two 256 x 256 x 36 core arrays.

The TX-1 will be so designed that it can operate 2 independent memory units concurrently, getting an instruction from one unit while the other unit is referring to the operand of the previous instruction.

Thus, by arranging to store the program in one unit and the operating data in the other, a significant increase in speed of operation is possible. This increase has been estimated to be about 70 percent. The design will, of course, also permit the program and its operanda to be stored in the same memory unit with a corresponding reduction in operating speed in this mode.

The TX-1 System Design

The "in-out break" mentioned above is somewhat like that of the FSQ-7. However, because of the absence of large-scale buffering, all terminal devices with critical timing requirements (for example, phone line receivers) must have a guarantee of access within certain time limits. The fact that all devices time-share the same memory unit will mean that one device will, in general, have to tolerate interruptions by other devices with higher priority. This fact greatly devalues "block transfers" of information and furthermore makes it necessary for each terminal device to "remember" what it was doing at the time of the interruption. In the TX-1, this function of storage of memory addresses during interruptions is handled partly by the use of index registers and partly by means of additional program counters in a manner later described.

Figure 1 is a simplified block diagram of the TX-1 showing only the principal information paths. The various terminal devices, labeled T_1, T_2, etc., are shown as being connected to the central computer by busses for simplicity. Each device includes enough storage to enable it to function between central memory accesses. The Memory consists of one or more 256 x 256 arrays, depending on the intended application of the computer and also includes a small toggle switch test memory. The Arithmetic Element will be assumed to be essentially like that of the FSQ-7 in logical structure for the purposes of this note.

The Terminal Selector is, as the name implies, a device for selecting one and only one terminal device and connecting it up to the central machine. This selection is made at designated interruption points in the program sequences. Some sort of priority system is implied such that terminal devices with critical timing requirements can be handled along with devices of less urgency. A "demand chain" similar to those appearing in the terminal system of the FSQ-7 satisfies the general priority requirement. In addition, an "in-out switch" is required to permit the central machine to select terminal devices directly, for example, to switch "off" units into the demand chain.

The Program Element consists of a set of program counters (one for each terminal device), a set of index registers, and an adder (see Fig. 2). To run a system of the size of SAGE, about 100 index registers and somewhat fewer program counters might be required. These might all be consolidated into one 256 register core-array with an access time of about 1-microsecond and a cycle time of half that of the central memory. Such a consolidation would have the advantage that the proportion of program counters to index registers is not fixed.

The primary function of the Program Element is to supply addresses to the Memory according to the requirements of the stored program. These addresses come from either a program counter or the adder register. The particular program counter in operation during any given machine cycle depends on which terminal device has been selected by the Terminal Selector, and the address it contains is indexed by one each time it is used, in the usual way. The index register in use during a given cycle depends, as in FSQ-7, on an index register number stored with the instruction being executed. The contents of the designated index register are added into an adder with the address-half of the instruction being executed, and the sum is then sent to Memory as the address of the operand designated by the instruction. Except for the fact that the TX-1 has several program counters and must select one of them at a time for any given cycle, the Program Element operates in essentially the same way as in FSQ-7.

To summarize: the word structure of a typical instruction, add, which requires its operand base-address, K, to be modified by the contents of index register j is

j add K

If index register j contains the number J, then the address of the operand (addend) is J + K.

We can now focus attention on the multiple program counter logic:

Multi-Sequence Operation

The TX-1 is a multi-sequence computer employing a set of program counters, each one of which marks the progress in its own sequence of instructions and is brought into operation at the request of an external device with which it is associated. In this respect, the TX-1 design is an extension of the 2-sequence DYSEAC of the National Bureau of Standards.

The program operating in the TX-1 can be thought of as consisting of a major program sequence and several essentially independent minor sequences. The major sequence is the large real-time control program corresponding to the single program of a one-sequence computer like Whirlwind. Each minor sequence is devoted primarily to transferring information between the computer and a corresponding terminal device.

These sequences are interleaved in time in an arbitrary fashion. The timing will depend largely on the requirements of the terminal devices themselves. However, interruptions in any sequence can occur only at points designated by the sequence itself. These

---

See 6M-3144 "The Multi-Sequence Program Concept" for a general description of multiple program counter operation.

breakpoints are marked by the presence of a "one" in a breakpoint bit stored with each instruction. A sequence may thus retain control of the computer by preventing interruption for as long as necessary to carry out required transfers, communicate with other sequences, etc. At each breakpoint the Terminal Selector selects the terminal device with the highest priority and connects it to the computer for the next computer cycle. Of course, the peak rate load of the entire terminal system will determine the maximum intervals between breakpoints in the sequences. The average input and output rates will limit the complexity and length of the sequences. The overall average number of transfers might require 10 to 15 percent of computer time at anticipated FSQ-7 rates.

The illustrated instruction must then include an indication of the status of its breakpoint bit:

*j add K

where the asterisk will be taken to mean that interruption is not permissible and that the next instruction to be executed will come from this same sequence.

Example of an In-Out Word Transfer

To illustrate a typical machine cycle, consider the example shown in Fig. 3. Terminal device #2 requires its next word from Memory and has been selected by the Terminal Selector as the subscriber with the highest priority.

The cycle begins (a) with the selection of the program counter to be used during the cycle, in this case PC#2. The address 101, subsequently indexed to 102, designates the next instruction to be executed in sequence number 2 and is sent to Memory as shown in (b). This instruction, 13 rdo 4000, indicates that index register 13 is to be used to modify the address of the operand, and the address-base 4000 and index register number are transferred to the Program Element (c). Inasmuch as index register 13 contains the number 7, the address of the operand becomes 4007 and this register is selected in Memory (d). The Control decodes the instruction rdo (read out) and transfers the contents of register 4007, which happens to be 666, to the in-out bus to which T_2 is connected (e), and the operation is complete. No asterisk appears with the instruction in this example, and hence a breakpoint is indicated. Thus the next instruction in sequence 2, stored in register 102, will not be executed until T_2 next requests, and is granted, control of the computer. Eventually the sequence folds back on itself after a branch instruction which resets program counter #2. Also, at some point in the sequence index register 13 is indexed and sensed for overflow, and reset as required.

Notice that reset values used in such a minor sequence may be supplied by the major sequence. This permits the major sequence, for example, to apportion internal storage among several input devices according to need. Also, one sequence can determine the progress of

another sequence by examining its program counter.

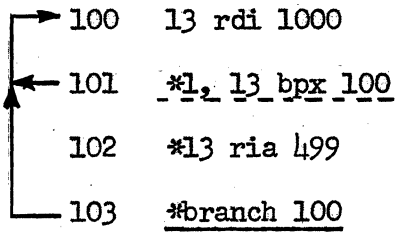
Examples of Minor Sequences

The following examples will help to illustrate some of the ways in which minor sequences might be used. First, it is necessary to describe the functions of the orders appearing in the examples: Most of these are found in the FSQ-7 order code. (K is the address of a register in the central memory.)

cad K: Clear accumulator and add in contents of K
aor K: Add one to contents of K (performed in AE)
fst K: Store Acc contents in K
branch K: Reset selected program counter to value K
k rdx K: Read contents of index reg. k into reg. K
k ria K: Reset index reg. k to the value K
j, k bpx K: If index reg. k is positive, reduce it by amount j and branch to K. If index reg. negative, ignore instruction.
bsn K: Branch to K if indicator in selected terminal device is set.
rdi K: Read word from selected terminal device to reg. K
rdo K: Read word from reg. K to selected terminal device

Except for the three orders which make special use of index registers, any of the above may be prefixed by an index register number.

Example 1: Input device assigned registers 1000 to 1499 for storing input data. Recycles if this assigned zone exceeded. Uses index register 13; program sequence stored in regs. 100-103.



The program counter for this sequence starts at 101 when request for machine cycle is granted. If assigned zone is not exceeded, index reg. 13 is counted down by 1 and sequence branches to 100, reads in word, and gives up control (no asterisk on instruction in 100 indicates breakpoint). Program counter ends up at 101. If zone was exceeded, index reg. 13 is reset to 499 before word is read in.

This example represents a minimum program-storage sequence. Note that each read-in requires the execution of 2 instructions (except when the zone is exceeded at which time 4 instructions are momentarily required). By expanding the sequence, the number of instructions per read-in can be reduced to nearly 1 as the following example shows:

Example 1a: (Same requirements as Example 1.)

```
100   13 rdi 1003
101   13 rdi 1002
102   13 rdi 1001
103   13 rdi 1000
104   *4, 13 bpx 100
105   *13 ria 496
106   *branch 100
```

Except for end-of-zone reset, this sequence uses 1.25 instructions per read-in. The feature of expanding sequences in this manner can, of course, generally be employed to reduce instruction time.

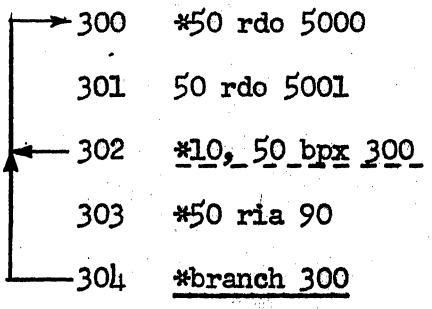
Note that if several input devices use the same program counter in the sequence of example 1, then they all load data indiscriminately into the one zone 1000-1499. A separate sequence for each results in separate zones for each device.

Example 2: Time of arrival to nearest t-seconds of input data of example 1 is required.

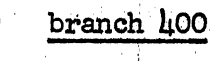
```
200   13 rdx 1500
201   13 rdx 1501
203   13 rdx 1502
204   13 rdx 1503
205   *branch 200
```

Terminal device consists of timer which requests machine cycle every t-seconds. Current intra-zone location read into 1500 on first cycle. t-seconds later, current location read into 1501, etc. These mark boundaries of data in the zone which are within t-seconds of one another. In this case, major sequence uses this information at least once every 4t-seconds.

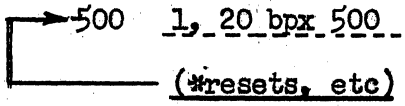
Example 3: Continuous, cyclic read-out (for example, to a display system) of every 10th pair of registers in zone 5000 thru 5091. Index reg. #50 used.



Example 4: Dynamic stop (imposed, for instance, by major sequence to shut off input device momentarily).

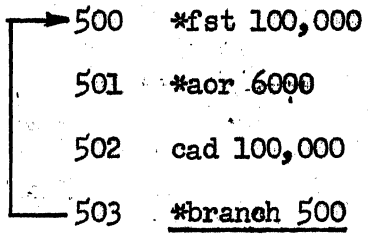


Example 5: Clock time to within t-seconds required in index register 20:



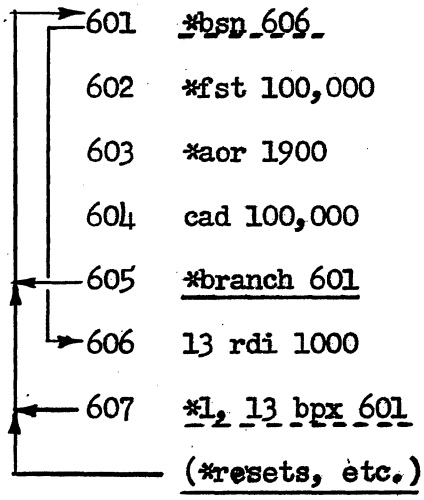
Terminal device consists of timer requesting cycle every t-seconds.

Example 5a: Clock time to appear in register 6000.



Same terminal device as in previous example. Sequence starts in 503 with branch. Next instruction empties accumulator for use by this sequence. Count is made in register 6000, accumulator restored on 502, and sequence gives up control until next timing event.

Example 6: Input device with parity on input word. Word is to be read in only if parity indicator is set. Count of failures to read in to appear in reg. 1900.



Program counter for this sequence starts from 607. If zone not exceeded, executes sense instruction in 601: Branches to 606 if selected parity indicator set, then reads in and gives up control. If indicator not set, increases count in 1900, restores accumulator and gives up control at 605.

These examples illustrate the flexibility of the multiple sequence technique and should suggest other applications not mentioned here. An important application which needs study is the programmed generation of displays. There is some hope that a display system like that of the FSQ-7 might be simplified to a considerable extent by special display sequences operating at different rates.

This flexibility in handling terminal equipment is an extremely important feature. The ultimate application of the machine in a control system may be largely unknown and yet will not appreciably affect the design of the central computer. By stressing the construction of computer programs rather than terminal equipment a considerable gain in system flexibility is achieved.

Signed Wesley A. Clark, Jr.

WAC/md/jg

Drawings Attached:
Figure 1 - SA-62428
Figure 2 - SA-62414
Figure 3 - SA-62458

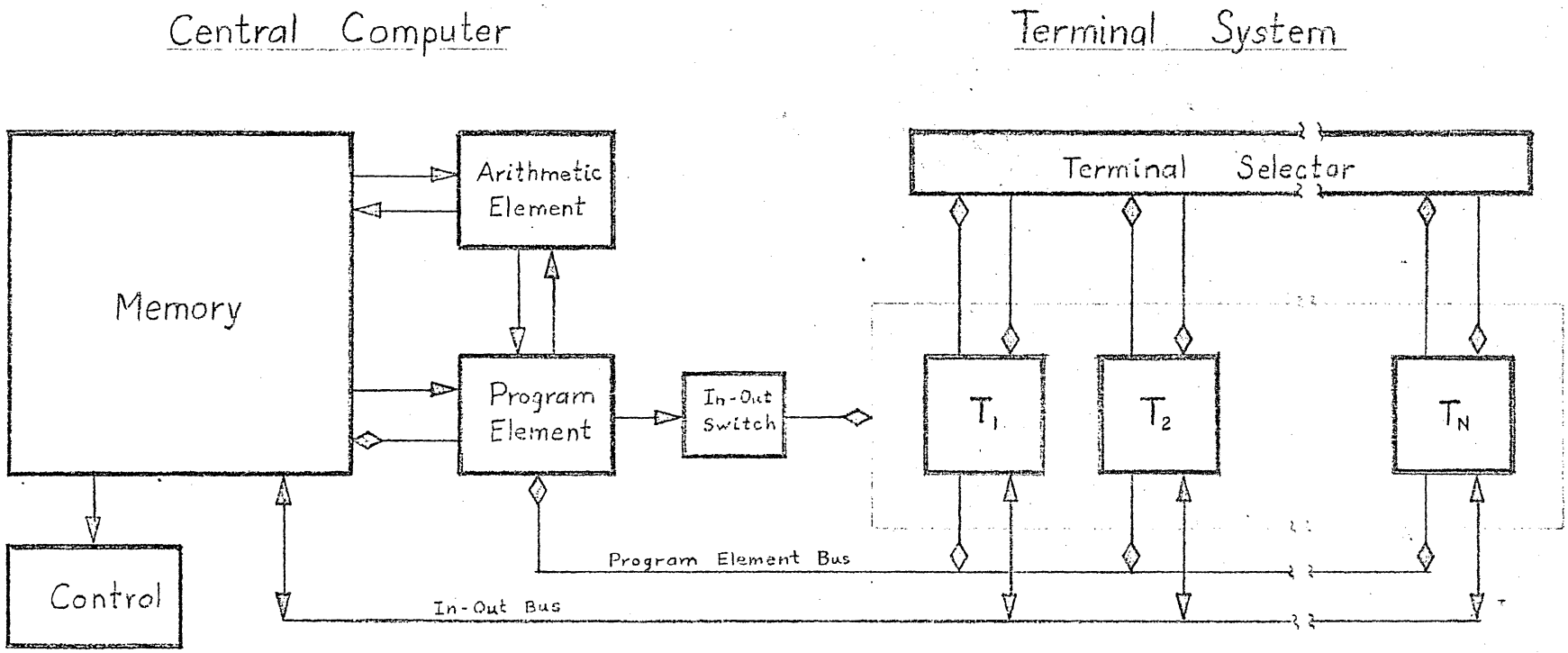


Figure 1

Simplified System Block Diagram of the TX-1

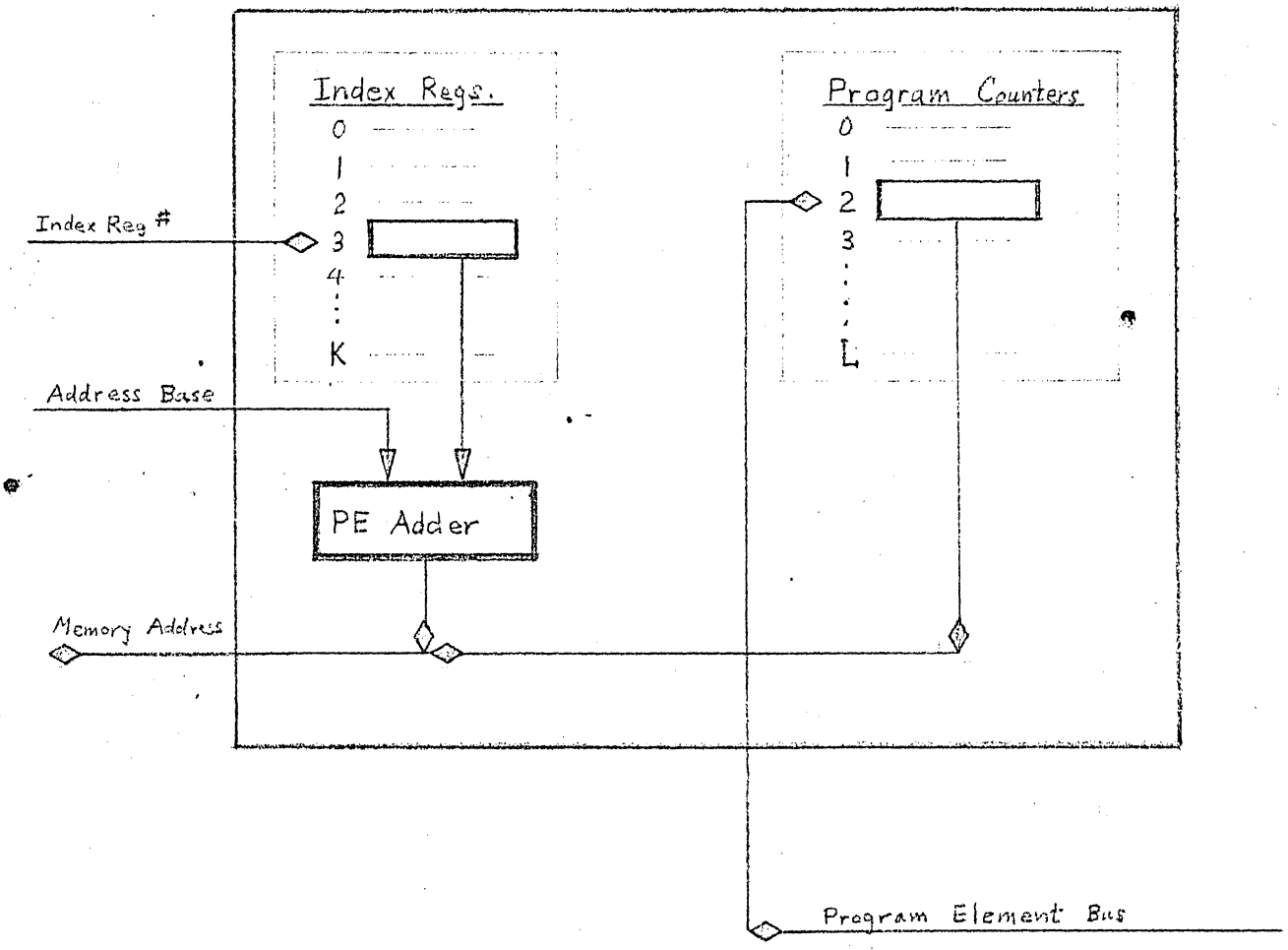
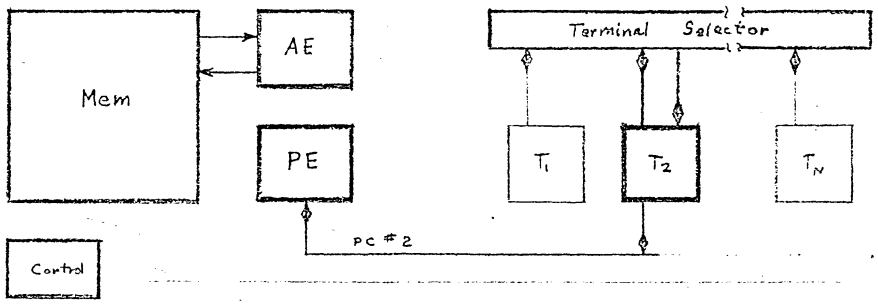
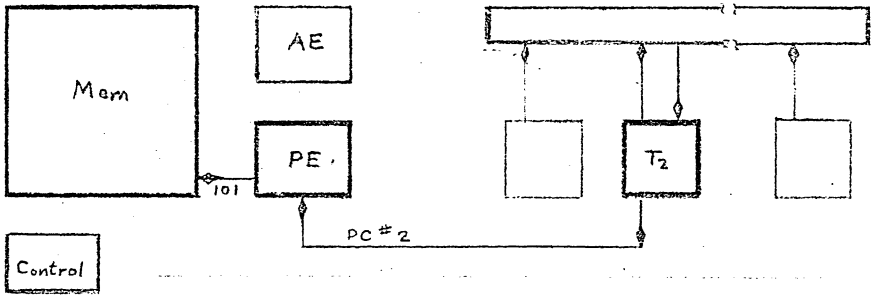


Figure 2

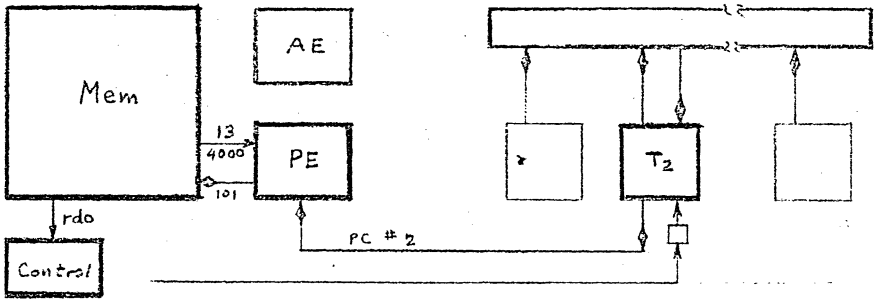
Program Element



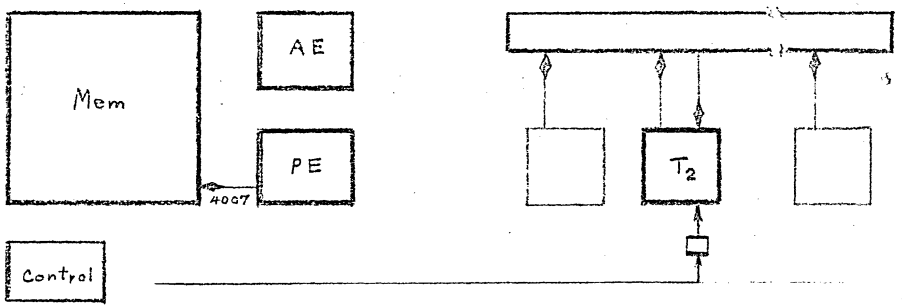
(a) Terminal Device #2 Selected



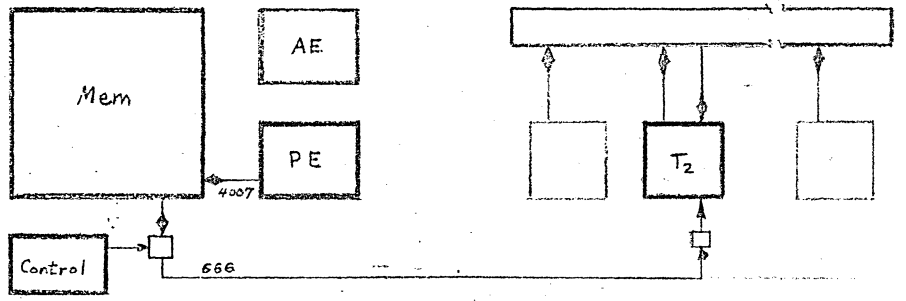
(b) Next Instruction Location to Memory



(c) 13 rdo 4000 Read from Memory



(d) PE Adder Returns Operand Address 4007



(e) Contents of 4007 Read Out to T_2

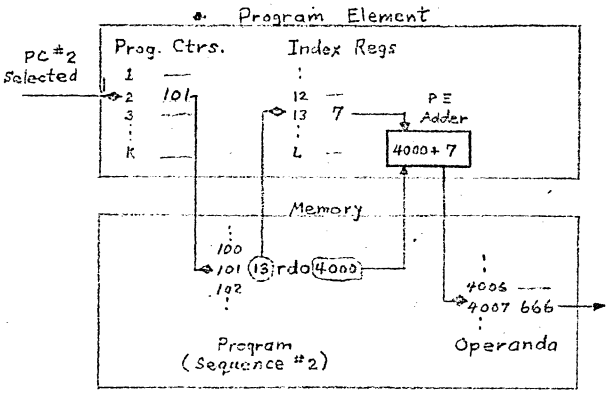


Figure 3

Example of In-Out Word Transfer