J. Cocke

REPORT ON THE SIMULATION PROGRAM FOR THE BASIC COMPUTER SYSTEM

Peter Perkins

Any simulation program may run and even produce an acceptable print out without truly representing the given system. It is the primary purpose of this paper to convince the reader that the Basic Computer, defined as of this date, has actually been simulated on the 704. It will also be seen that, as the result of employing Fortran Assembly, the simulation program is extremely flexible and can readily be adapted to many changes in the computer system. We shall first discuss on an elementary level the basic concepts involved in representing a machine of the future on an already existing computer; they are the transfer of information, the passing of time, and interlock in general. Finally, we shall consider in detail the simulation of the Indexing Arithmetic Unit, the Decoder and Main Arithmetic Unit, and the Buss system.

Transfer of Information

Probably the most basic idea is that of representing, in 704 language, the flow of information throughout the computer system. For purposes of illustrating these general concepts we shall use mainly the buss system portion of the program, but of course these methods are also used in simulating the Indexing and Arithmetic Units. Consider the distinct (general) locations in which a piece of information can exist. In the basic system they are as follows: (1) two sending register (2) a fast memory-in-buss (3) any one of four fast memory boxes (4) a fast memory out buss (5) a slow memory-in-buss (6) any one of 16 slow memory boxes

(7) slow memory-out buss (8) two return register (9) decoder and arithmetic unit (10) indexing arithmetic unit. Each of these general locations has a block of 704 memory associated with it. (Note that each of the 20 memory boxes has a block.) The first eight general locations above comprise the buss system and their 704 blocks each contain a sub-block of four addresses in which are stored the same type of information. That is, each sub-block will contain:

(2) The address in main memory to which the data now in this location is (was) trying to go.

(3) The return or destination address to which the information wants (will want) to return after its memory cycle.

(4) The instruction number of that instruction which is (tying up) this location.

(5) An address which distinguishes a store order from a bring order.

In Fortran language it is now very easy to simulate the transfer or information around the system. For example, in order to get a memory cycle started from the sending register we first must find out if the desired memory box is free. This is accomplished by interrogating the 704 memory cell representing the instruction number tying up this particular Basic Computer memory box. If there is nothing in this 704 address then we know the box is available and we should send it the data if we can catch a place on the memory in-buss (fast or slow.) The buss

availability is determined in precisely the same manner. If we now find the buss free we store the contents of the 704 sub-block assigned to the register into the 704 sub-block assigned to the memory-buss and erase the register sub-block. (It is this type of step which makes up about 50% of the program and which is carried out comprehensively and economically by Fortran with its subscripted variables.) We also set the buss clock (discussed below.) On a succeeding pass through the program the clock will read zero and we can move the information into the appropriate memory box without again checking its availability. After a sufficient number of time units have been subtracted from the memory box clock, it will read zero and we can then move to the next general location (ie memory out buss) etc., etc.

Passing of Time

Every general location 704 storage block (except the sending and returning registers) has either one or two "clock" addresses in it. When information is moved into a new general location, for instance from the indexing A.R. into the decoder, the clock concept is of dual significance. First, the only reason we moved the information on this "pass" through the program was because the clock address of the previous location had been interrogated and found to be zero. Secondly, we must now store in the clock address of the new location that constant equal to the number of time units the information will be present in this location. Every time we make a complete pass through the program, that is after we have moved things

from every location that has information in it AND whose clock address contents is zero AND whose succeeding location is free, then we subtract one from the contents of all clock addresses that do not already contain zero. We then again ask what locations are free and what information wants to be moved.

Interlock

The interlock features of the simulation are rather simple. One example is the use of a "block-bit" address in the 704 storage block associated with the decoder and arithmetic unit. Interrogation of and conditional transfer on the contents of this address will prevent the indexing arithmetic unit from processing the J^{th} instruction before the $(J-1)^{th}$ instruction is through the A. U.

A second example is our application of a "wait-bit" address for the arithmetic unit storage block. Discovery of a zero in the A. V. clock address, in itself, will not justify an attempt at putting the contents of the A. V. into the sending register and trying to get a memory cycle started. Actually there are two possible situations when something is in the A. V. and the A. U. clock reads zero. One is when the data has finished being processed and the A. V. clock has run down. The other is when the A. V. is waiting for data from memory and the time constant has not yet been stored in the clock address. Each requires a different subsequent course of action. Therefore it is necessary to interrogate and conditional transfer on an

auxilliary 704 address that tells us which of the two situations is the case. In general, if we do not want to commence routine X until routine Y is in state Q and if routine Y may be in state Q under more than one condition, then we introduce a supplementary 704 address, the contents of which will distinguish between conditions.

Miscellaneous

Other addresses within the 704 storage blocks are used for a variety of purposes. For instance, in each 704 storage block associated with a memory box we have an address containing the "key" of that box. The "key" 704 address merely holds the highest address of the Basic Computer memory box that this 704 block is representing. It is necessary in order to make the proper box "busy" with any given instruction. Every block associated with one of the 4 registers (^{2}sending and ^{2} returning) includes an address for the "state bit" of that register. If the contents of the "state bit" address of a sending register is a one, then the program knows it wants to start a memory cycle. If a return register "state bit" is a one then the A. V. knows that data has arrived from memory.

The indexing A. V. is always in one of five states. The state of existence when interrogated will determine the subsequent course of action.

Detailed Description Using Fortran Symbols

For the sake of simplicity let us define all constants, basic machine units, and programing devices, in the Fortran language. The following are the 704 symbolic locations which will be assigned absolute locations by Fortran

assembly.

Symbols connected with indexing A.V. IXAUT contains some constant equal to the time for indexing IXAU (IXS):

When IXS = 1, IXAU (1) contains the address of the instruction
When IXS = 2, IXAU (2) contains the address of the index 1
When IXS = 3, IXAU (3) contains the address of index 2
When IXS = 4, IXAU (4) contains the address of index 3
When IXS = 5, IXAU (5) contains the actual address of data
When IXS = 6, IXAU (6) contains the instruction number tying up indexing unit
When IXS = 7, IXAU (7) contains the instruction type tying up indexing unit
When IXS = 8, IXAU (8) contains the time left for indexing unit to run.

IXBB contains value of indexing A. V. "block bit" (1 or 0.)

Symbols Connected with Decoder and Arithmetic Unit

IDK contains a constant which determine the type of instruction
Type No. ((IDK) ⟶ store
Type No. ((IDK) ⟶ bring

IT(I) where I takes on values 1 through 29. In each of the 29 addresses is a different constant used to set the arithmetic unit clock according to the type of operation to be performed.

JAUT is set equal to IT (I) after I has been determined and the A. V. has started operation. One time unit is subtracted from this address on each pass through the program.

IAUW contains the value of the A. V. "waiting bit" (1 or 0.)

Decoder

IDR (I):

When I = 1, IDR (1) contains time left for decoder to run
I = 2, IDR (2) contains memory address to which we must go.
I = 3, IDR (3) contains value of decoder block bit (1 or 0.)
I = 4, IDR (4) contains instruction number tying up decoder.
I = 5, IDR (5) contains instruction type tying up decoder.

Symbols Connected with the Buss System

IBUXT contains that constant equal to the length of time information will tie up the fast memory out buss.
JBUXT same for fast memory in buss
IBUMT same for slow memory out buss
JBUMT same for slow memory in buss

IBUX (I)	When I=1,	time left on this buss section
JBUX (I)	When I=2,	memory address to which we are going
IBUM (I)	When I=3,	return register address to which we are coming back
JBUM (I)	When I=4,	instruction number tying up this buss section
	When I=5,	distinguishes a store from a bring

Symbols Connected with the Sending and returning Registers

IR (I,J)		concerns the J^{th} register (we deal here only with registers (3, 4, 13, 14)
	When I=2-5,	this is the same sub block as we defined for the buss sections
	When I=6,	IR (6,J) contains the value of the sending "state bit" (1 or 0)
	When I=7,	IR (7,J) contains the value of the returning "state bit" (1 or 0)

Symbols Connected with the Memory Boxes

IMT1(I)	contains the constant equal to read out time of the I^{th} memory box
IMT2(I)	contains a constant equal to the read out time plus the read in time of the I^{th} memory box. This method allows us to set both clocks at once.
IK	contains a constant equal to the highest address in the fast memory. This enables us to determine onto which buss (to fast or slow memory) information should be placed.

IM(I,J)	concerns the J^{th} memory box.	
	When I=2-5	this is the same sub-block as we defined for buss sections and registers
	When I=6, Im (6,J)	contains the "key" of the J^{th} box that is it will allow us to pick out this box when we have information headed for it.
	When I=7, IM (7,J)	contain the amount of read in plus read out time left for this J^{th} memory box.
	When I=1, IM (1,J)	same for read in time above

An intuitive grasp of the program, obtained by following through the flow diagram, (Appendix II), may be desirable before a detailed study of the actual program step by step, is made. Then, with the definition of a few Fortran statement the reader should be able to make his way through the

whole situation.

Three types of Fortran statements make up about 90% of our program. They are "If" statements, "Do" statements, and statements of the form A = B. An "If" statement, which is of the form "If (KOW) A, B, C" means that if the contents of location"KOW" is negative transfer to A, if zero transfer to B, if positive transfer to C.

A "Do" statement, which is of the form "Do N I = a_1, a_m" is a looping instruction. It says to perform the succeesing instructions as far as, and including instruction numbered N and then begin the loop again. Each time the program loops back the value of the subscript will be increased by one, until the instructions have been carried out with the value of I = a_m. The statement of the form A = B simply sets the content of A equal to the contents of B.

Let us consider the following portion of our program:

```
 10     Do 101 I = 2, 5
        IM (I, J) = JBUX (I)
        IM (I, J) = IMTI (J)
        IM (7, J) = IMT2 (J)
101     JBUX (I) = 0
 12     Do 162 J = 1, 4
        IF (IR(6, J)) 500, 162, 122
```

When we transfer into statement 10 the value (J) has already been determined as the number of the memory box we are looking for. Also, the clock address of the memory in buss must have been zero. The "Do loop" now sets the 704 sub-block associated with this particular J^{th} memory box equal to the sub-block associated with the in buss, JBUX (I). It also clears the in buss during the fifth instruction. The "do" loop does not effect the third or fourth instructions because their only subscript is a

J which has been fixed. These two instructions, however, start the read-in and read-out time clocks by setting the contents of the clock addresses equal to the cycle times for this particular box. When this "Do" loop is satisfied control proceeds to statement 12, another "Do" loop. The first statement in this loop asks if sending registers 1, 2, 3, or 4 want to start a memory cycle (In this program only 3 or 4 could possibly want to.) That is, if the contents of the state bit address is negative transfer is made to 500, a stop instruction, for this address should never be negative. If it is zero transfer is made to 162 the last statement of the loop, which will in turn send control to the beginning of the loop with the value of the subscript increased by one (unless it is already 4, in which case it goes to the next instruction outside the loop. If the contents of the "state bit" address is a one and thus positive, transfer is made to 122 and steps to instigate a memory cycle are taken, etc., etc. As the reader can undoubtedly observe, the use of Fortran greatly clarifies and facilitates simulation programing especially through the self-explanatory nature of its instructions.

PRINT OUT

The print out (see appendix IV) is read as follows: Each of 26 columns represents one of 26 general locations (20 memory boxes, 4 buss sections, indexing A. V., decoder). Each row represents the passing of .1 microsecond. The number printed at the intersection of column L and row T is the number of the instruction which is tying up location L at time T. Thus, glancing across any row we can immediately see which units are tied up with what instructions at any given time. Also, we may skim down a column and see how often a given unit is busy. The one remaining column gives the "state" of the indexing unit at .1 microsecond intervals.