The Logical Design of CG24

G. P. DINNEEN  I. L. LEBOW  I. S. REED

THIS PAPER discusses some features of the logical design of the CG24 computer with emphasis on the techniques used in the logical design rather than on the particular characteristics of the computer itself. Often the phrase logical design of a digital system is used to describe the detailed configuration of logical circuits, either schematically in block diagram form or algebraically in the form of logical equations. Here the term, logical design, is used to describe the system operation in terms of the flow of information or the transfer of information from one register to another. Such a description has two significant advantages. First, it provides a rather concise picture by which one may characterize the over-all structure of a system without the confusion introduced by detailed considerations and second, while relatively independent of detailed circuit configurations, it can be translated easily into equipment by a circuit engineer.

CG24

CG24 is a general-purpose real-time digital computer. It is general purpose in that it is a stored program machine with a fairly extensive list of single address instructions and conventional terminal equipment. In addition, it has a data input from a radar receiver and a data output to the radar antenna. Hence it may be used to process radar data in real time and also to direct the position of the radar antenna. The computer is entirely solid state with ferrite core storage and transistor and crystal diode logical and memory drive circuits. Its principal characteristics are shown in Table I. From this table it is readily observed that it is in the class of current commercial scientific machines of the International Business Machines Corporation (IBM) 704 type. It has, however, achieved its operational speed with slower arithmetic circuits than most other machines of this class. This was accomplished by introducing a good deal of parallel operation. Nevertheless, the over-all power requirement of 4.5 kw, about 2 kw of which is for the cathode-ray tube display units, is modest and the physical size is small as seen in Fig. 1.

The computer has been installed and has been operating at a Massachusetts Institute of Technology (M.I.T.) Lincoln Laboratory field station since May 1958.

Registers and Transfers

A digital machine is composed of a number of registers with certain allowed interactions between them. Basically, a register \( R \) is a set of \( n \) bistable elements \( R_i, i=0, 1, \ldots, n-1 \). Such a register may contain \( 2^n \) different numbers. Let \( (R) \) designate the contents of register \( R \). If \( S \) is another \( n \) bit register in the machine, then the simplest interaction between registers \( R \) and \( S \) is the transfer of the number in register \( R \) to register \( S \) or in symbols \( (R) \rightarrow S \), which signifies that after the interaction (transfer) has occurred, the contents of register \( S \) are identical to those of register \( R \) before the interaction. Register \( R \) is unaffected by the transfer.

The class of registers in the machine is enlarged by considering as registers parts of registers, functions of registers, and functions of several registers. Thus, designate part of register \( R \) as its address part, \( Ad[R] \), and consider this as a register. The complement of a register \( \bar{R} \) is considered to be a register. The algebraic sum of two registers \( R \) and \( S \), \( (R)+S \), is considered to be a register. Thus the class of generalized registers of a machine contains not only the registers themselves but also all desired functions of these types.

The class of transfers is enlarged by admitting conditional as well as unconditional transfers. Thus if \( R, S \) and \( T \) are registers and \( \lambda \) is a two valued function, then

\[ \lambda(R)+\lambda(S)\rightarrow T \]

is a conditional transfer stating that if the function \( \lambda \) is true then the contents of \( R \) are transferred to \( T \) while if \( \lambda \) is false the contents of \( S \) are transferred to \( T \).

Description of a Computer by Generalized Transfers

The operation of a digital system may be completely described by sequences of generalized transfers of the type defined in the previous section. For the description of a general purpose computer one must have sequences of generalized transfers which provide the following:

1. An algorithm for the performance of each operation or instruction in the repertoire of the machine,
2. A method of executing a sequence of such instructions; i.e., a program, and
3. A method of operating a control unit to implement steps 1 and 2.

To design a digital computer to meet a certain set of operational requirements, one must implement the necessary generalized transfers for these three functions subject to the constraints imposed by the available hardware.

The first two of these requirements define the arithmetic and data processing capabilities of the computer. The third

---

Dinneen, Lebow, Reed—The Logical Design of CG24

Table I. CG24 Computer Characteristics

<table>
<thead>
<tr>
<th>General</th>
<th>Application: General-purpose plus real-time control.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerical System</td>
<td>Internal number system: 27-bit binary words, including two parity bits, with provision for double accuracy computation.</td>
</tr>
<tr>
<td></td>
<td>Single-address instructions.</td>
</tr>
<tr>
<td></td>
<td>Fixed-point arithmetic system, with a programmed floating point subroutine.</td>
</tr>
<tr>
<td>Arithmetic Unit</td>
<td>Addition time: 24 ( \mu )sec (including memory access)</td>
</tr>
<tr>
<td></td>
<td>Multiplication time: 84 ( \mu )sec (including memory access)</td>
</tr>
<tr>
<td></td>
<td>Division time: 84 ( \mu )sec (including memory access)</td>
</tr>
<tr>
<td></td>
<td>Square root time: 300 ( \mu )sec (including memory access)</td>
</tr>
<tr>
<td></td>
<td>Pulse repetition frequency: 330 kilocycles per second.</td>
</tr>
<tr>
<td>Storage System</td>
<td>8,192 words, coincident current magnetic cores, 12-( \mu )s cycle time.</td>
</tr>
<tr>
<td>Terminal Equipment</td>
<td>Flexowriter providing 2-way communication between operator and machine.</td>
</tr>
<tr>
<td></td>
<td>Cathode-ray tube displays: Alpha-numeric display of the contents of up to 192 memory registers.</td>
</tr>
<tr>
<td></td>
<td>High-speed photoelectric tape reader.</td>
</tr>
<tr>
<td></td>
<td>Three input registers for transfer of real-time data to the internal memory.</td>
</tr>
<tr>
<td></td>
<td>Two servosystems for controlling two independent shaft positions.</td>
</tr>
</tbody>
</table>

---

Dinneen, Lebow, Reed are with the Massachusetts Institute of Technology (M.I.T.) Lincoln Laboratory, Lexington, Mass. The work reported here was performed at Lincoln Laboratory, a technical center operated by M.I.T. with the joint support of the United States Army, Navy, and Air Force.
requirement; i.e. control, is solely to implement the other requirements. Look first at the control and how it affects the rest of the computer. The control unit of any machine (regardless of physical implementation) may be thought of as having a finite number of configurations or states. Each of these states defines a certain set of generalized transfers to be performed in the machine. A machine is made to perform a given instruction by constraining the control unit to cycle through a definite sequence of states which specify uniquely the sequences of generalized transfers necessary for the execution of that instruction. More precisely, let $f_i$ be a Boolean function which is true if and only if the control is in its $i$th state. The relationship $[f_i](A) \rightarrow B, (C) \rightarrow D$

signifies that the contents of register $A$ at the beginning of the time interval during which $f_i$ is true are to appear in register $B$ by the end of that interval. Similarly the contents of register $C$ are transferred to register $D$ during the same interval. It is easily seen that in this way a sequence of control states $[f_i]$ causes a series of generalized transfers to occur which can provide the means of executing any instruction or a sequence of such instructions. Furthermore, the necessary transfers for implementation of the control function itself may be described in this way.

Note even at this point that this method of description need not presuppose detailed knowledge of the circuits to be employed in the construction of a system. The sequences of states $[f_i]$ may be set up either synchronously or asynchronously. The indicated transfers ($A) \rightarrow B$ and $C) \rightarrow D$ may occur at the same time or at different times; they may be performed serially or in parallel. All that is demanded is that they be concluded by the end of the indicated time interval.

**Application to CG24**

The basic notions of the previous sections were used in the design of CG24. The hardware upon which the design was to be based consisted of:

1. A coincident current magnetic core memory driven by transistors with a 12-microsecond (μsec) cycle time, this time being divided roughly into a 3-μsec read interval, a 3-μsec write interval and two 3-μsec post-write-disturb intervals (the post-write-disturb intervals were later eliminated but the cycle time remained at 12 μsec).

2. Transistor and diode logical circuitry capable of operation under load conditions at clock rates up to about 0.5 megacycle (mc).

The operational requirements indicated that the following computer characteristics were acceptable:

1. Addition time of 24 μsec (including memory access).

2. 8,000 words of core memory for storage of instructions and constants.

3. Single address structure for instructions.

4. Fixed point arithmetic with a 25-bit word.

5. 5 index registers.

On the basis of the circuit characteristics and operational requirements the following timing was established:

1. The entire machine (except perhaps for some terminal equipment) was to be synchronous with the core memory. The 12-μsec memory cycle interval was divided into four 3-μsec pulse intervals labelled $P_1, P_2, P_3, P_4$.

2. No flip-flop was to be required to change its state at a rate faster than 333 kc (once every 3 μsec).

3. The full 25-bit add time (exclusive of memory access) was to be less than 12 μsec.

4. The control unit was to change its state once every memory cycle. Hence the control state $f_i$ was subdivided into the four substates, $f_i P_j, j = 1, 2, 3, 4$, which would determine the machine transfers.

**Some Typical Instructions**

To describe some of the computer operation, consider first the instruction "add $X$" which signifies that the contents of register $X$ in memory are to be added to the contents of $A$, the result being stored in $A$. The first memory cycle of operation, designated by $af_2$, is the one in which the instruction is obtained from memory. Then the control substates will be designated by $af_3 P_j$ or $a'f_3 P_j$. The machine as so far defined is shown in block diagram form in Fig. 2.

Having established the basic properties of the computer together with its elementary timing, the design techniques may now be applied to the actual machine. First the functions of a few registers of the machine will be detailed. Let $M$ designate the memory of the computer and let $C$ designate the memory address register. The symbol $M < C >$ means the particular memory register determined by the address in the $C$ register. Let $N$ and $L$ designate respectively the memory output and input buffer registers. Let $R$ and $A$ designate two arithmetic registers, the operand storage register and the accumulator respectively. Addition is performed between numbers in $R$ and $A$. Finally let $D$ represent the program counter. For the present, the operation of the control unit will not be specified. Assume that it changes state (from $f_1$ to $f_j$, etc.) in accordance with the previous description. For convenience, the control state $C$ will be designated by an additional function. Let a be a Boolean function which when true indicates that the current memory cycle is one in which a new instruction is being obtained from memory. Then the control substates will be designated by $af_3 P_j$ or $a'f_3 P_j$. The machine as so far defined is shown in block diagram form in Fig. 2.

**Dinneen, Lebow, Reed—The Logical Design of CG24**

From the collection of the Computer History Museum (www.computerhistory.org)
read and write periods. During \( P_1 \) the contents of the selected memory register are transferred to the output buffer \( N \) and at \( P_2 \) this same word is rewritten in memory. This, of course, is necessitated by the destructive character of the read process. During \( P_3 \) the instruction code section of the word is transferred to the control unit. During \( P_4 \), the current address in \( C \) is stored in \( D \) and is counted up by 1 during \( P_5 \). This augmented address is the address of the next instruction in the absence of a jump instruction. Finally during \( P_6 \) the address part of the word just obtained (the address \( X \)) is transferred to \( C \) in preparation for the next cycle.

Assume that the control changes to state \( a'f_1 \) (the control state representing addition) which sets up the following sequence of transfers.

\[
\begin{align*}
|a'f_2P_1|: & (M<C>) \rightarrow N \\
|a'f_3P_1|: & (N) \rightarrow M<C>, (N) \rightarrow R \\
|a'f_4P_1|: & \text{reserved}
\end{align*}
\]

The memory operation as before occurs during \( P_5 \) and \( P_6 \) and the word so obtained is transferred to \( R \) during \( P_6 \). During \( P_7 \) the indicated sum is taken and stored in \( A \). (The addition time of the computer is actually 3 \( \mu \)sec.) During this final interval the address of the next instruction stored in \( D \) is transferred to \( C \) and the computer is prepared to interpret the next instruction.

As another example, consider the instruction "store in \( X \)" which requires that the contents of the accumulator \( A \) be stored in register \( X \). The first cycle of operation during which the instruction is obtained is designated by \( af_3 \) as for addition.

The second cycle is as follows:

\[
\begin{align*}
|a'f_2P_1|: & (M<C>) \rightarrow N, (A) \rightarrow L \\
|a'f_3P_1|: & (L) \rightarrow M<C> \\
|a'f_4P_1|: & \text{reserved} \\
|a'f_5P_1|: & (D) \rightarrow C
\end{align*}
\]

During \( P_6 \), the word in \( A \) is transferred to \( L \) and is stored in memory during \( P_7 \). As previously, the address for the next instruction in \( D \) is transferred to \( C \) during \( P_8 \).

As a final example consider the instruction "transfer negative to \( X \)" which indicates that if \( A \) contains a negative number, the next instruction is to be taken from register \( X \), otherwise from the next register in sequence. Again the instruction read-in cycle is the same as before. It is followed by:

\[
\begin{align*}
|a'f_2P_1|: & \text{reserved} \\
|a'f_3P_1|: & \text{reserved} \\
|a'f_4P_1|: & \text{reserved}
\end{align*}
\]

The conditional transfer indicated at \( P_4 \) states that if \( A_0 \) is true (sign of the accumulator is negative), \( C \) remains unchanged whereas if \( A_0 \) is not true (positive sign), the address in \( D \) is transferred into \( C \).

The necessary transfers for three instructions have been demonstrated. All the remaining instructions are designed in the same systematic way. What remains is to indicate how the control unit operates so as to require the machine to perform the indicated sequences of transfers.

The Control Unit

The control of CG24 may be thought of as a memory unit. It contains a set of registers called \( Cm \), an address register \( G \) and an output register \( F \). There is also an auxiliary register \( T \). The control is shown in block diagram form in Fig. 3. The states of the \( F \) register determine the control states of \( \alpha_f \); one bit in \( F \) is designated \( a \). During \( P_1 \) of every memory cycle an address is inserted in \( G \). The control word selected by \( G \), \( Cm<G> \), is stored in \( F \) during \( P_4 \) of each cycle thereby setting up the proper control state \( a'f_1 \) or \( a'f_3 \) for the succeeding cycle of operation. In CG24, \( Cm \) is a fixed diode memory. Logically speaking it could have been an active memory like a core memory. Each word in \( Cm \) represents a unique control state \( a_f \) or \( a'f \). It may be divided into two parts, one which initiates the transfers in the machine such as those described in the previous section, and a second address part designated \( G[F] \) which refers to operation of control itself.

Each instruction in the computer is designated by a unique sequence of control states, hence by a unique set of control words in \( Cm \). The address part of a control word usually contains the address in \( Cm \) of the next control word. Actually the operation of the control memory is determined by the transfer to \( G \) which was mentioned previously. This is

\[
|f_5P_1|: f(N) \rightarrow f'[G(F)] \rightarrow G
\]

When \( a \) is true, the current cycle is one in which an instruction is obtained from memory. In this case the instruction code part of \( N \) is transferred to \( G \). This transfer was implied in the previous section when the instruction read-in cycle was described. During all other cycles \( a \) is not true and either the address part of \( F \) is transferred to \( G \) or \( G \) remains the same depending upon a Boolean function \( \lambda \) to be described presently. Following this transfer into \( G \) results in the transfer

\[
|f_5P_1|: Cm<G> \rightarrow F
\]

which sets up the proper state for the next cycle. This state is either that describing the beginning of an instruction \( (I[N]) \rightarrow G \), the repetition of the previous cycle \( (G) \rightarrow G \) or the advancing to the next state of the current instruction \( (G[F]) \rightarrow G \).

All that remains is to describe the advancing function \( \lambda \). This function is true if the number in the \( T \) register is 0, 1, 2, or 3 and is false otherwise. Thus to "stick" the control in one state a number larger than 3 is inserted in \( T \) and to advance to a new state \( T \) must be counted down to 3 or lower.

The advantage in this kind of control is its inherent versatility. To change the nature of the machine it is necessary only to change the control memory which in turn represents different sequences of states and hence transfers in the machine. In CG24 such changes can be made at a very slow rate by physically replacing the control memory with a different memory. If the memory were active such changes could be programmed.

Conclusions

The ideas presented here demonstrate a method of designing a digital computer by specifying control states which in turn specify sequences of generalized transfers. All this is done with some rather general ideas about the actual circuit configurations. To translate this description into a real computer is a relatively simple step logically. It involves setting up a system of gated transfers where the gating is derived from the \( F \) register and a counter whose state designates the \( P_1 \) interval.

The versatility of the control memory was noted in the previous section. Indeed this does provide a method of setting up arbitrary sequences of states to execute instructions. This versatility is not put to use particularly in CG24 where the control memory is limited to 64 states and where each of the generalized transfers is wired in. Any new instruction for CG24 must be made up of existing transfers, states for which already exist in the control memory.

However, the notation using transfers between registers and the implementation
of the control memory does permit the design of a much more versatile machine. The next step in designing such a machine is to provide a method of building in all possible transfers between existing generalized registers. Suppose the set of generalized registers is designated by $\mathcal{M}$ and suppose that a control word in $F$ contains three addresses, $(\mathcal{A}_d[F], (\mathcal{A}_d[F]),$ and $(\mathcal{A}_d[F])$ each of which refers to one of the generalized registers in $\mathcal{M}$. Then the general transfer is stated as follows:

$$f_1: \mu(\mathcal{M} < \mathcal{A}_d[F]) + \mu(\mathcal{M} < \mathcal{A}_d[F]) \rightarrow \mathcal{M} < \mathcal{A}_d[F]$$

The function $\mu$ is a Boolean function designated in the machine. If this function is true the generalized register determined by the contents of $\mathcal{A}_d[F]$ are transferred to the generalized register determined by the contents of $\mathcal{A}_d[F]$; otherwise the contents of the register determined by $\mathcal{A}_d[F]$ are transferred.

When viewed in this way, all the registers and derived registers are connected together through a large selection switch (or several such switches for simultaneous transfers) which is actuated by the state of control. Changing the nature of the machine is then accomplished by changing the contents of the control memory which essentially sets up a sequence of selection switch positions to perform the desired instructions.

References


Discussion

D. P. Boone (Astronautic Company): You will notice that the feeder is tied to the radar. How do you accomplish the feeder from the radar?

Dr. Lebow: The way it is set up right now is the following: There are three registers which take the data from the radar, and associated with these registers is another bit which tells the computer that some data has come in. Then upon sensing this bit, the data are transferred directly into the computer memory, without going through the arithmetic unit.

W. J. Seiple (Federal Laboratories): In discussing the conversion of the data, how do you get them into the computer so that it is tagged to that correlation?

Dr. Lebow: One of the registers I spoke about is a register that contains the actual real time, the time at which the return was received, and this represents part of the data that are introduced into the machine.

K. L. Deane (Varianmatic): Does this include some analog to digital conversion?

Dr. Lebow: Yes, for the data part of the word, not for the time part, of course.

Question: What consideration, or what are the considered optimizations of these controls?

Dr. Lebow: This kind of optimization was considered by the people who actually deal with circuit design. I, perhaps, did not say enough about that at the beginning of the talk when I explained what we meant by logical design. I think our use of the term is a little different than what most people mean by the term. You can see from the talk that the term logical design does not get down to the detailed logical configuration.

Obviously, a lot of work is necessary in going from the transfer level to the actual circuit details, and this is a function of hardware that the machine will be built of. This was optimized in some sense by the people who actually built this machine.

Design Criteria for Autosynchronous Circuits

J. C. Sims, Jr.  
H. J. Gray

Synopsis: The circuits and organization of present computers are such that possible operating speeds are lower than the capabilities of the components. The speed limitations of such synchronous computers will be described, and design criteria for higher speed operation set forth. Examples will be given for a logic and circuit organization which results in both faster operation and improved performance to cost ratios. In particular, circuits which are free of transient logical malfunctions, sometimes called "spikes," will be developed and a typical autosynchronous system will be shown.

Computing circuitry is generally organized to transfer digital information from a first storage through a logical net into a second storage register. As the information passes through storage, the wave shapes are standardized and the relative and absolute timing of the signals are restored.

This situation is generalized in Fig. 1. Here information stored in two registers is read out on arrival of a timing pulse, $CP_1$, passes through a logical net, and is received by an output register. Upon receipt, it can be read out again to the same or to a further network in response to a timing pulse, $CP_2$. In synchronous machines, the signals $CP_1$ and $CP_2$ are clock pulses, and in the common single-phase systems are the same signal.

In order to establish a point of departure for the present discussion, a brief analysis will be given of synchronous systems of this form. The storage registers are usually flip-flop or shift registers, depending on whether the system is parallel or serial. The logical net is made up of combinations of "and," "or," and "not" elements, each element usually consisting of an amplifier and a group of logical diodes or resistors. The computing device is thus constructed of four basic devices, a store or flip-flop, an "and" circuit, and "or" circuit, and a "not" circuit. In some systems, a stroke element is used to function as an "and," "or," and "not" device. Stroke elements are typified by the "Larc 1c" circuit and the "nor" circuit. 1, 2

Dr. Lebow: This kind of optimization was considered by the people who actually deal with circuit design. I, perhaps, did not say enough about that at the beginning of the talk when I explained what we meant by logical design. I think our use of the term is a little different than what most people mean by the term. You can see from the talk that the term logical design does not get down to the detailed logical configuration.

Obviously, a lot of work is necessary in going from the transfer level to the actual circuit details, and this is a function of hardware that the machine will be built of. This was optimized in some sense by the people who actually built this machine.

Fig. 1. General block diagram

Sims, Gray—Design Criteria for Autosynchronous Circuits

From the collection of the Computer History Museum (www.computerhistory.org)