TABLE OF CONTENTS

1. INTRODUCTION

Flight simulators are based on the theory that the behavior of an aircraft in flight can be represented mathematically by a system of equations in which certain "output" variables, such as position coordinates and orientation angles, change appropriately with time in accordance with certain "input" variables, such as throttle position and stick position. In order to build a flight simulator, it is necessary first to develop a system of equations that will accurately represent the characteristics of a particular aircraft, and then to design and assemble a computer that will solve these equations in accordance with a variety of inputs representing the physical situation and provide accurate output signals with which to drive cockpit instruments, generate control forces, and activate all of the peripheral equipment (e. g. , motion system, visual display) required for a realistic training environment.

Until a short time ago, flight simulators were based exclusively upon analog computation techniques, in which aircraft flight quantities (altitude, airspeed) are analogously represented by electrical and mechanical quantities (voltages, shaft positions) in an electromechanical network that duplicates the interrelationships between the various subsystems that make up a particular aircraft. In recent years, however, in spite of the many advances made in analog computing technology, the flight simulator, along with the aircraft it simulates, has increased in complexity to the point where designers have had to supplement the capabilities of conventional analog equipment by incorporating digital equipment in ever-increasing amounts.

The fundamental difficulty with analog simulators is that the organization of the elements of these machines must be very similar to the organization of the mathematical equations describing the operational system. That is, for every element in an equation there must be a corresponding element in the computer which, as a rule, serves only this one function. Thus, as the complexity of the equations increases, so does the complexity of the analog hardware. Furthermore, once an analog simulator has been built, any changes in the operational system, and hence in the equations of the system, can be implemented only by making corresponding changes in the hardware of the analog computer, with the attendant penalty of lost computer time.

Contrast this with the great flexibility of the digital machine, whose elements are organized to carry out the step-by-step instructions of a written program. Any changes in the equations being solved affect only the written program and do not require changes in the hardware of the computer itself. That is, there is no downtime involved when making the actual changeover except the time required to load the new program (generally only minutes). Add to these qualifications the inherent accuracy

of the digital computer, plus its reduced overall size and power consumption, and it becomes apparent that a digital approach to real-time aircraft simulation is much to be desired.

The first serious attempt to apply digital computing techniques to the problem of real-time flight simulation was initiated in 1950, when a joint Air Force-Navy program was established at the Moore School of Electrical Engineering, University of Pennsylvania. The Moore School's long-time association with the development and application of digital computing techniques made it a natural choice to head the program, which was named UDOFT (Universal Digital Operational Flight Trainer). The intent of the UDOFT project was to develop a suitable digital computer and program it to solve a set of flight equations, so that the feasibility of digital simulation in real time could be established.

The equations used by the Moore School were taken from two existing analog flight simulators, representing the F9F and the F100 fighter aircraft. This was done so that a direct comparison of flying qualities of the digital and analog simulators could be made. No attempt was made to derive new equations directly from the aircraft data because it was intended that the degree of simulation should remain the same.

By 1958 the development of the mathematical techniques had been completed and the logical design of the UDOFT computer was well defined. Actual construction of the digital computer was accomplished by the Sylvania Corporation and was completed in 1960. The results of the program clearly established that a digital computer could successfully be used for the real-time simulation of aircraft flight.

In spite of the fact that it performed successfully, however, UDOFT was (and is) far from being the final answer to the problem of all-digital flight simulation. In the first place, it was never intended to be more than a feasibility tool, preprogrammed to solve a particular set of aerodynamic equations, and, as a result, its application to simulation problems in general (function generation, Boolean operations) was severely limited. What was needed was a new species of special-purpose computer designed, like UDOFT, exclusively for flight simulation purposes, but with adequate capability to accommodate the total range of computational requirements associated with the modern flight simulator. The Mark I is such a computer.

Link began developing the Mark I with the idea that the computer should be economically justifiable for use in flight simulators under substantially the present cost structure and should capitalize on the undeniable advantages of digital computation: high accuracy, extreme flexibility, high reliability, small size, and reduced heat dissipation. The main objective, however, was to obtain a digital computer suitable for use in simulators

that customers, both commercial and military, could afford. As a result, the Mark I is a rather unusual computer in that it is designed for maximum usability in the simulation and training equipment domain.

The Mark I computer is a highly specialized combination of proven techniques that completely eliminates the risk inherent in the use of developmental circuits and components, while at the same time reducing the overall cost and complexity of the computer to the economic level mandatory for flight simulators. With the exception of some minor innovations, Link makes no claim that the Mark I computer advances the state of the digital computer art in general. It is simply a special combination of existing digital computing equipment that uniquely satisfies the computation requirements for simulation devices.

The computations handled by the Mark I can, if carefully simplified, be handled in marginal fashion by other large-scale computers (the CDC 1604, the IBM 7090, the fastest of the Philco Transac series, the PDP3, and perhaps others). However, because the Mark I is designed specifically to accommodate the particular problems of real-time simulation, it is substantially less expensive and will be much easier to program than any commercially available computer would be in this application.

2. ADVANTAGES OF MARK I APPROACH TO FLIGHT SIMULATION

The tasks required of a computer used for flight simulation are highly specialized. To be completely useful, the computer must be capable of:

1) Performing all required mathematical operations peculiar to simulation problems. These operations are addition, subtraction, multiplication, division, squaring, square root, integration, level and polarity detection, and scaling.

2) Performing all required Boolean (switching logic) operations peculiar to simulation problems. These operations are LOGICAL SUM (OR), LOGICAL MULTIPLY (AND), and INVERT. The Boolean operations should lend themselves to the generation of pseudo-Boolean functions (for example, radio call letter generation).

3) Storing large amounts of permanent data, such as curves of arbitrary nonlinear functions, constants, and parameters defining navigational facilities.

4) Accepting large amounts of rapidly changing input data from the cockpit and instructor's controls. These input data should be accepted in raw form, with a minimum need for intermediate modification or translation before the computing process commences.

5) Processing all the current input data and permanent data in accordance with a program of instructions that is completely flexible. The program should be versatile enough so that certain parameters having critical dynamic characteristics (e.g., Euler angle rates) and critical accuracy and resolution requirements (e.g., altitude, latitude, longitude) can be given special consideration without reducing the computing efficiency or useful memory capacity.

6) Providing a large amount of new, smoothly changing, accurate answers. These answers should be provided in a form that requires a minimum amount of conversion or translation in order to be useful.

7) Accepting new, modified, or corrected program instructions and permanent data changes rapidly and with ease.

8) Minor program revision or updating by personnel with a basic analog background.

In specifying the characteristics of the Mark I, Link carefully studied the pertinent features of available digital computers in relation to the requirements of flight simulation. In examining these requirements, they were divided into categories so that areas of computation with similar

problems could be taken together. These categories are:

1) Flight equations

2) Engines and aerodynamic coefficients

3) Accessory systems and instructor inputs

4) Radio navigation

The solution of flight equations requires that many calculations be performed at high speed, over and over again. A high repetition rate is necessary in order to avoid lags in the computations, which would cause the simulation to depart from realism. The experience of the Moore School with the UDOFT program had shown that an iteration rate of 20 times per second was required for an adequately responsive solution of the equations of motion. This requirement demands a high-speed, parallel-arithmetic computer with a random-access core memory.

The Mark I is such a computer. It performs all of the required arithmetic operations (add, subtract, multiply, divide, square, square-root, shift, absolute value) under program control, and with great rapidity. If no other operations were considered, the Mark I could perform as many as 163,840 additions, or 27,273 multiplications, or 23,405 division operations per second. These times include all instruction and data access times as well as the time required to perform the operations. From this point of view, the Mark I ranks with the faster digital computers available at the present time.

The calculation of engine parameters and aerodynamic coefficients involves the generation of a large number of arbitrary functions of one, two, and three variables representing empirical aircraft data. In conventional digital computers, it is necessary to develop a polynomial expression for each arbitrary function and then program the computer to solve the expression for all values of all variables. These operations are very time-consuming and also require a high degree of skill in the mathematical operations of data reduction and curve-fitting to achieve a satisfactory polynomial expression. In addition, the functions can change quite rapidly, so that a high iteration rate is necessary in the solution of the polynomials, and, since many program steps are involved, the burden on the computer is quite high.

In the Mark I, function generation is handled by straightforward linear interpolation, using straight-line-segment function curves which in most cases are taken directly from aircraft data curves. The function interpolations in the Mark I take place sequentially, are repeated 10 times per second, and are done independently of the main program — i.e., they

require no instructions from the main program, and therefore no attention from the programmer. The stored function curves are entered directly as numerical data along with the core memory address of the interpolated result. Because of this simplicity of organization, it is quite easy to change individual function curves as aircraft data change, without the necessity for extensive mathematical operations.

The simulation of aircraft accessory systems is primarily a problem in switching logic, requiring only a small amount of arithmetic computation. This means that the computer has to be able to handle large numbers of Boolean (one-bit) words representing switching (on/off) functions. Many computers have the capability to incorporate two or more independent bits of Boolean information in one computer word; however, they usually require several extra program steps to extract a desired item from the various items contained in a given word.

The Mark I has a separate arithmetic unit that can perform programmed Boolean operations on as many as 2048 independently addressable, single-bit words stored in a functionally separate section of the random-access core memory. Since each single-bit word can be individually addressed, no extra instructions need be written or executed to obtain access to the desired word. This arrangement conserves the expensive core memory, and thus helps to keep the Mark I economically priced. Furthermore, by minimizing the number of instructions that must be performed, it also reduces the labor of initial programming and reprogramming. In other words, because the Mark I has been designed to operate on single-bit words directly, it is ideally suited to the computations associated with accessory systems. Some relays may still be required in cases where large amounts of power are switched, but the racks of relays traditionally associated with logic circuits will be entirely absent from the Mark I flight simulator.

Radio-navigation problems, important as they are from a training standpoint, have always been extremely difficult to simulate with a high degree of realism. The maximum incorporable repertory of radio facilities has always been severely limited, and the accuracy of simulation has often been poor owing to the nature of the computing equipment. Now, with the advent of digital simulation, there is promise of tremendous improvement in the simulation of radio facilities. Not only is greatly increased accuracy possible, but the number of facilities that can be represented is increased by a factor of nearly a hundred.

The Mark I computer was designed with the problem of radio facility representation very much in mind, especially the problem of selecting eligible transmitters for reception by the aircraft receivers. The Mark I provides a repertory of 350 separate and independent navigation transmitters,

and, on the basis of receiver tuning and aircraft geographic position, selects for each receiver the one best transmitter to be received. This selection process is carried out for each navigation receiver aboard the simulated aircraft, and is accomplished automatically, without operator control and without using a single program instruction. Programmed operations are required only for the calculations associated with the nature of the transmitter and receiver and with the simulated physical situation; the actual selection of a transmitter for a given receiver and the transfer of the stored data that define the location and characteristics of that transmitter are accomplished in parallel with the main program, like function generation, and do not subtract from the time available for executing programmed arithmetic operations.

A digital simulation computer must be able to handle large numbers of inputs and outputs, both analog and Boolean. Conventional machines can do this only at the expense of precious program space. The Mark I input/output system, however, operates automatically. No instructions are required in order to accomplish the input/output memory transfers or conversions to or from digital form, and no time is subtracted from the time available for the execution of the general program. The input system automatically scans analog inputs, converts them to digital form, and loads them into preassigned core memory locations. Similarly, the output system obtains variables from preassigned core memory locations, converts them to analog quantities, and makes them available on separate output lines for use outside the computer. In addition, a large number of Boolean (single-bit) input functions are scanned and inserted into preassigned core memory locations, and a large number of Boolean outputs are available from preassigned core locations. In all, a total of 126 analog inputs, 192 analog outputs, 1024 Boolean inputs, and 256 Boolean outputs can be accommodated by the Mark I.

A-D

D-A

Boolean

Because the Mark I program is changed infrequently (in contrast to the typical scientific computer application), it can be stored more or less permanently on a magnetic storage drum. Conventional digital computers utilize the core memory for instruction storage, and, because core memories are so expensive, have a rather limited capacity for the storage of program. In order to utilize the available storage space efficiently, elaborate programming techniques, such as looping, branching, and instruction modification, are necessary. In conventional digital computers, a time penalty is also paid, because after the execution of each instruction the computer must wait for the next instruction to be obtained from the core memory.

This is not so in the Mark I, because the instructions are stored on the drum in the exact order in which they are performed, and as one instruction is being performed by the arithmetic element, the next one to be performed is moving into position to be read. When it is in position, it is read and performed without waiting. Most instructions require only one access to the core memory, others require no access at all, and no instructions require more than one core memory access.

Since program instructions are never modified intentionally by the Mark I computer, errors cannot be made in the process and the instructions will therefore not be altered accidentally — as could happen in a conventional digital computer. Storage of the program on a magnetic drum is also far less expensive than an equivalent core memory. The Mark I contains program storage space for 45,056 instructions. Of these, 4096 are performed 20 times per second, 8192 are performed five times per second, and 32,588 are performed every 0.8 seconds. This, of course, does not include the capability represented by the automatic radio system, the function generator, and the automatic input/output system — none of which use any of the 45,056 instructions.

Link studies have indicated that 45,000-instruction storage is more than adequate to accomplish a degree of simulation which is equivalent to, and in many cases better than, that of present-day simulators. Estimates of required program space were based on a mathematically rigorous axis system and uncompromising accuracy throughout all computations. The use of equations of this degree of rigor in an analog simulator would be prohibitively expensive, and in many cases would be intolerable from the standpoint of dynamic accuracy and stability because of the inherent limitations of analog equipment. For purposes of comparison, it is estimated that the Mark I has a computing capacity equivalent to that of a highly flexible analog system containing 150 servos, 1200 operational amplifiers, 1000 potentiometers, and 20,000 precision resistors. In exploiting this capability, the designer of the Mark I flight simulator has the option of incorporating many additional features not found in present-day analog flight simulators.

3. FUNCTIONAL DESCRIPTION OF COMPUTER

3.1 GENERAL DESCRIPTION

A simplified block diagram of the Mark I flight simulator is shown in Figure 1. The major elements of this system are:

1) The Mark I digital computer

2) The cockpit (flight crew location)

3) The instructor's station (near flight crew)

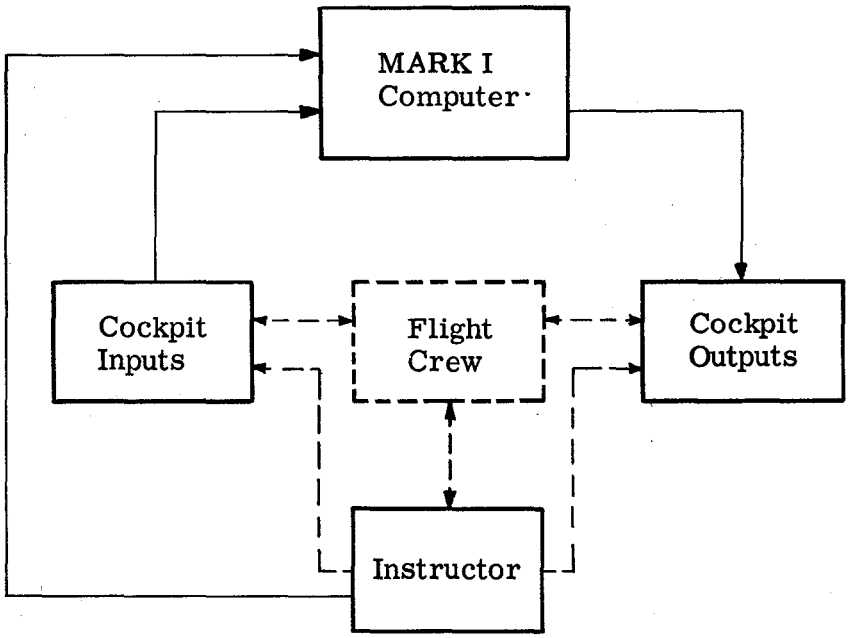


Figure 1 GENERAL DATA FLOW – MARK I FLIGHT SIMULATOR

The block diagram treats the cockpit and the instructor station as the input/output elements for the Mark I computer. Thus, the major input/output loop is closed through the flight crew personnel, while the instructor functions as an arbitrary input device.

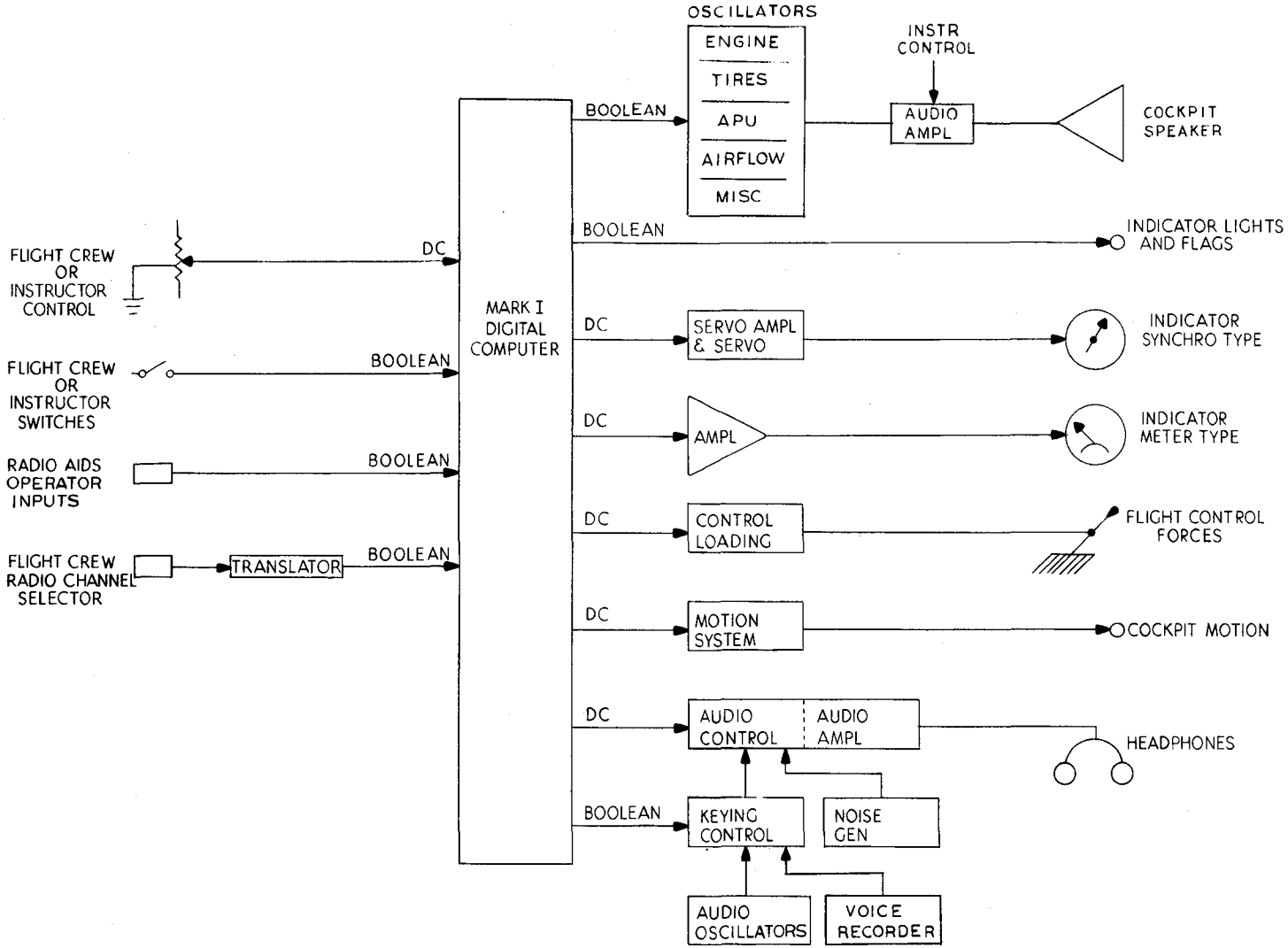


Figure 2 BLOCK DIAGRAM OF MARK I FLIGHT SIMULATOR

Figure 2 is a more detailed block diagram of the flight simulator. From this diagram, it can be seen that the major problems to be solved by the computer are:

1) Accept all cockpit and instructor inputs, such as flight controls, engine controls, accessory system switching functions, radio-navigation controls, and malfunction controls.

2) Compute solutions to all aircraft equations of motion throughout the flight envelope of the aircraft for normal and emergency flight situations.

3) Compute solutions to all engine equations for normal and emergency operation of the engines.

4) Compute solutions to all switching logic equations used to define aircraft accessory systems operation, for normal and emergency operation.

5) Compute all navigation equation solutions.

6) Provide appropriate outputs to cockpit and instructor station indicators, lights, audio devices, recorders, and control force generating equipment.

The manner in which each of these problem areas is handled is described in detail in subsequent sections. In general, the Mark I digital computer will perform all of the arithmetic and Boolean operations required in the modern flight simulator. It will solve the equations which represent the characteristics of the airframe, the onboard systems (including engines), and the complete radio-navigation problem.

As shown in Figure 3, the Mark I computer operates in accordance with a written program stored on a magnetic drum. Using a special tape reader and high-speed drum loader, the many thousands of instructions constituting a typical flight simulator program are stored on the drum in the order in which they are to be performed. During the operation of the computer, these instructions are read and performed in the order in which

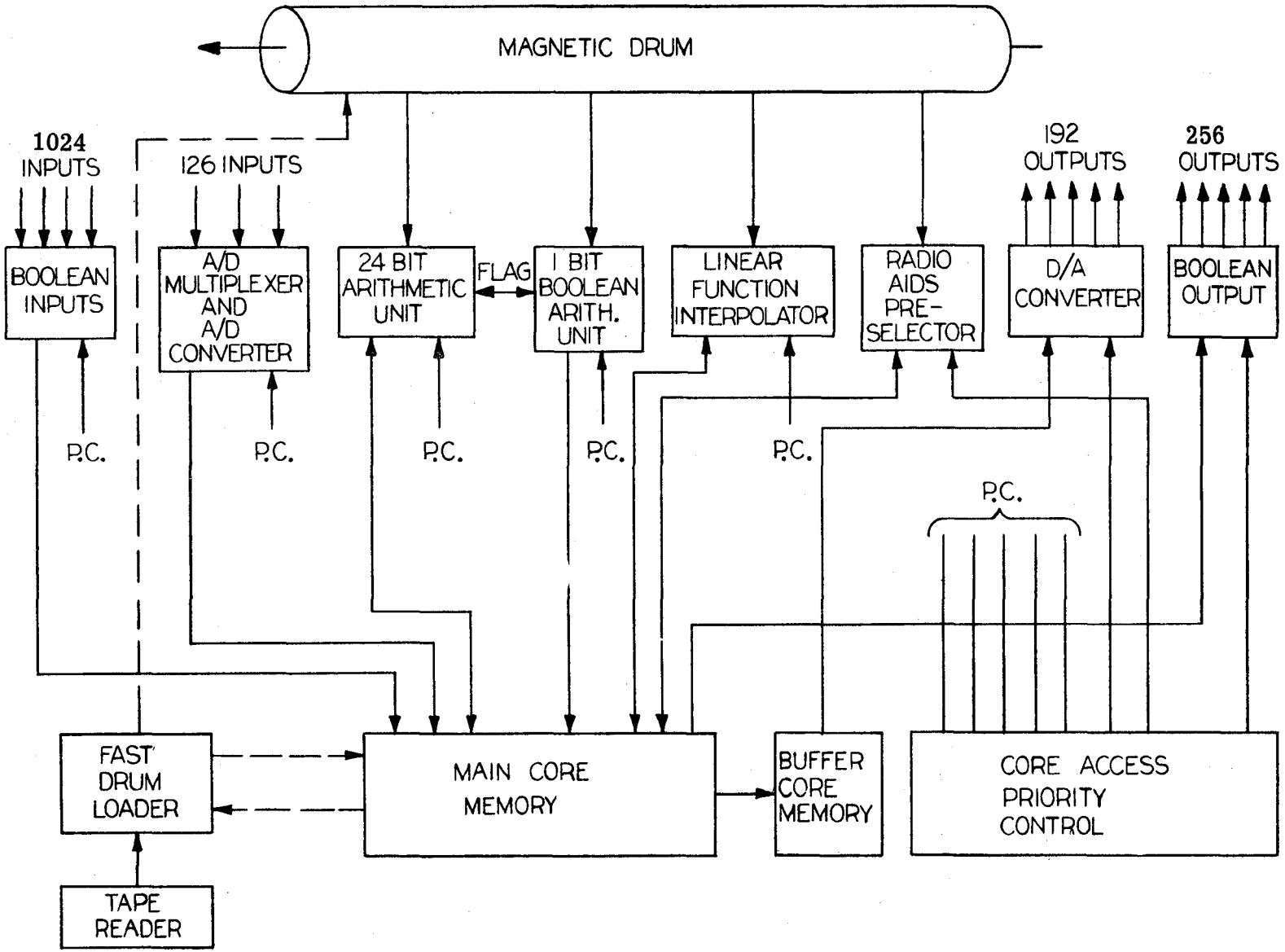


Figure 3 MARK I COMPUTER (SIMPLIFIED)

they were written, without waiting and without the necessity of addressing the location of the next instruction.

If an instruction is read demanding an arithmetic operation, this operation is performed in the main arithmetic unit. This unit contains the registers and logic circuitry for performing all arithmetic operations. Similarly, instructions indicating Boolean operations are performed in the Boolean arithmetic unit. The source of all numerical and Boolean data words for these operations is the main core memory, a fast, random-access storage unit that serves as the Mark I's working memory.

ALU

The Mark I also includes a large-capacity linear interpolator that operates in parallel with the main program to generate instantaneous values of the many complex functions encountered in aircraft flight and engine computations. These functions are constantly being calculated and recalculated and stored in preassigned memory locations in the main core memory for use by the main program. Since this is a parallel operation, there is no time lost in the main program calculating these quantities.

LFI

The Mark I also includes considerable input and output equipment (see Figure 3). In the input system, analog and Boolean inputs from the outside world (for example, simulator cockpit controls) are written into preassigned core storage locations where they may be used by the main program. The output system reads calculated analog and Boolean data out of the core memories in the form of analog voltages with which to activate the simulator equipment (for example, the cockpit instruments). This system, too, operates in parallel with the main program, thus using no separate allotment of program time to achieve these operations.

Also operating in parallel with the main program is a large-capacity radio preselection unit. This unit compares the location of the aircraft and the frequency to which each of its receivers is tuned with the locations and frequencies of some 350 radio transmitters. It then selects the best possible transmitter, if any, that each receiver should be picking up and stores numerical data concerning the selected transmitter in preassigned core memory locations for use by the main program. Again, since this is a parallel operation, there is no time spent in the main program achieving this end.

Because all of its major operations take place in parallel — i.e., without waiting for other operations to be completed — the Mark I is an extremely fast computer. This computational speed is essential in order to ensure adequate dynamic performance of the Mark I flight simulator under real-time conditions.

3.2 MAGNETIC DRUM

The Mark I drum contains ~~16~~ 13 bands of instructions and constants. Once this information has been written onto the drum, the "write" equipment is disconnected and no further information can get into the drum. The design of the Mark I deliberately prevents any further writing on the drum or modification of information contained on the drum.

Once the computer selects a band of instructions to read, every word on that band is read in order, at the rate of one word every 6.105 microseconds, until all the words on that band have been read. At this time, another band is selected and all its words are read. As each instruction is read, the operation indicated is performed. If the operation requires more than 6.105 microseconds to complete, then it is necessary to make the succeeding instructions NO OP's until enough time has been allowed for completion of that operation.

The 16 bands on the drum are made up from 240 data tracks — each track being one bit wide and 4096 bits in length around the circumference of the drum. Thus, there is a total storage capacity on the drum of over 983, 040 bits. The drum rotates at 2400 rpm, therefore requiring 25 milliseconds to complete one revolution All bits comprising a word are stored and read in parallel — that is, if the first word in a band of the main program were being read, then the first bit in each of the 16 data tracks making up that band would be read simultaneously. Since all the words in a band are read in one revolution of the drum, the read rate is approximately 164 kilocycles, or 6.105 microseconds per word.

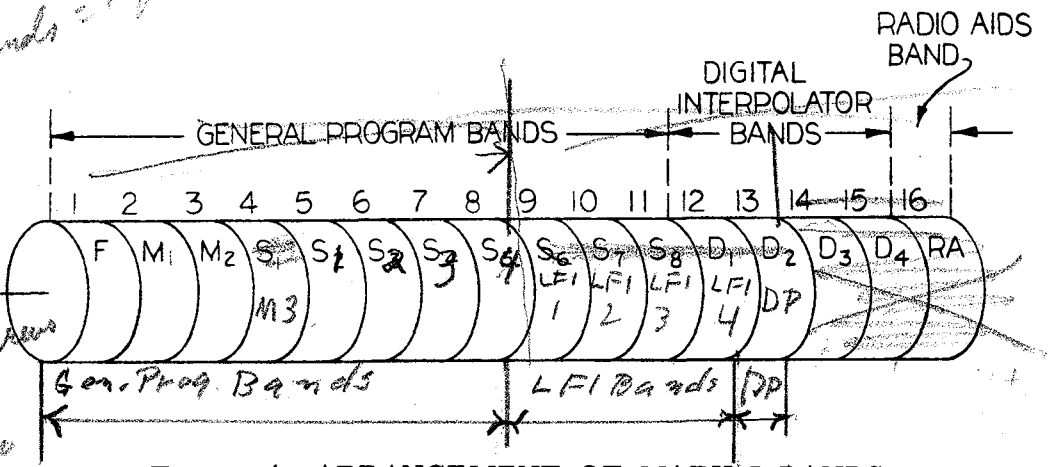


Figure 4 ARRANGEMENT OF MARK I BANDS

Eleven of the 16 bands of words written around the drum (Figure 4) are for the general program, four contain instructions and constants for the linear interpolator, and one is for radio aids. Of the 11 bands reserved for the general program, one is called a fast band, two are called medium bands, and eight are called slow bands.

All words in the general program portion of the drum are 16-bit words. There are 4096 words per band and a total of 11 bands. This gives a total of over 45,000 words for this portion of the drum. The four bands of the digital interpolator portion are made up of 11-bit words at 4096 words per band. This represents a total of over 16,000 words in this section. The 4096 words on the radio-aids band are all 20-bit words.

Each time the drum makes one revolution, three bands of instructions and constants are read simultaneously: one band is read and performed by the general program section of the computer, one band by the digital interpolator, and the single band belonging to radio aids is read and performed by that section (this single band is read and performed on every revolution).

The 11 bands comprising the general program portion of the drum are not simply read in order, 1 through 11. The reason is that in simulator work some quantities change much more rapidly than others; consequently, they require a higher frequency response of the computer (that is, it is essential to recalculate these quantities much more frequently that is necessary for many other computations in the simulation program).

To handle this situation, one band of the general program portion of the drum is designated as a fast band, and all the calculations on this band are performed on every other cycle of drum rotation. The instructions for the calculation of those quantities requiring frequent updating are all placed on this band. Two of the remaining ten bands are designated as medium-fast bands (M_1 and M_2), and the instructions on these bands are performed alternately on every fourth revolution of the drum (first M_1; then, four revolutions later, M_2; then, four revolutions later, M_1; and so forth).

The remaining eight bands are designated as slow bands. These bands are taken one at a time on every fourth revolution of the drum until all eight have been read, and then, of course, the process is repeated. Because of this read order, instructions on the fast band are performed every 50 milliseconds, those on the medium-fast bands every 200 milliseconds, and those on the slow bands every 0.8 seconds.

As stated previously, two other bands on the drum are being read simultaneously each time a band of the general program is read (that is, on each drum revolution). One of these is the radio-aids band, which is read on every drum revolution, and the other is one of the four linear interpolator bands. The four interpolator bands are read one after the other at the rate of one band per drum revolution. Therefore, any given interpolator band is read once on every fourth revolution of the drum. Figure 5 illustrates the final order in which all the bands on the drum are read. The first line of this illustration represents the general program bands, the second line represents the linear interpolator bands, and the third line represents the radio-aids band. Each vertical column represents one drum revolution (25 milliseconds). After 32 revolutions (0.8 seconds), all of the bands have been read in the order indicated.

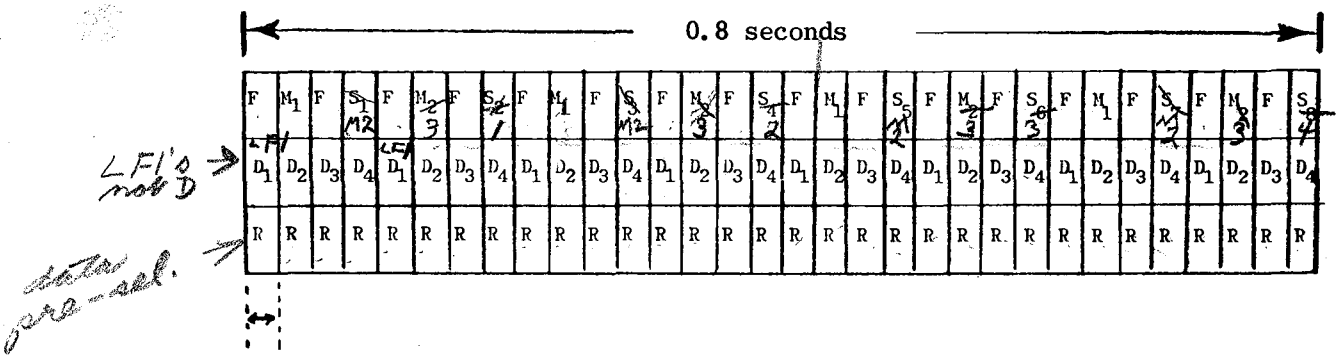


Figure 5 READ ORDER OF MARK I BANDS

3.3 CORE MEMORY

The core memory of the Mark I is a 2048-word, random-access memory, each of whose 2048 words is an addressable location. Numerical information may be stored in any word location, or, conversely, the information contained in any word location may be read out on command. Only one word location in the core may be read out of or written into during any given machine operation cycle (6 microseconds). Each word of the main core is 24 bits long, of which the most significant bit is the sign bit.

It is the function of the main core memory to act as the working storage of the computer. That is, all quantities stored in the main core can be changed, updated, and erased.

The variables in the simulator equations are each assigned to a specific location in the core storage. As each of these variables is recalculated or changed, the new value is inserted into the proper core location, thus replacing the previous value. All inputs to the Mark I computer from the outside world (for example, cockpit, instructor's station) come to preassigned storage locations in the main core. All outputs from the Mark I to the rest of the simulator system are read out from their preassigned storage locations in the main core.

Independent variables to be used by the linear interpolator for the purpose of function generation are also read from their assigned location in the core memory. Similarly, computed functional values are stored in assigned core locations by the linear interpolator. In short, all mathematical quantities needed for a simulator program are stored in the core memory (with the exception of constants, which may be stored on the drum).

As stated before, only one word of the entire core memory may be interrogated or written into during any one 6-microsecond period. However, it is apparent that the many parallel processes of the computer (that is, function generation, radio preselection, input-output reading, main program arithmetic) all require memory access. Hence, it is necessary that these various processes be given a priority rating. In the Mark I, any operation of the main program requiring memory access takes priority. Any operations or instructions in the main program that do not require memory access are considered to be "holes" in the main program, and it is during these "holes" that the auxiliary processes of the computer gain access to the core memory. Hence, instructions in the main program such as SCALE, SHIFT, TAKE ABSOLUTE VALUE, INVERT, ZERO SLICE, FLAG and NO OPERATION, which do not require memory access, act as "holes" to auxiliary sections of the computer.

Priority for all of the parallel processes of the Mark I is as follows:

1 Keyong function Generator
~~2 1)~~ Main program GP (General Program)
~~3 2)~~ Digital function interpolation LFI (Linear Function Interp.)
DP ~~4 3)~~ Radio aids preselection (Data Preselect)
5 ~~4)~~ Analog input scanning A/D
6 ~~5)~~ Analog output reading D/A
P 7 ~~6)~~ Boolean input scanning DSO
8 ~~7)~~ Boolean output reading DSI

The reasons for this particular order of priority will become apparent as the various operations are discussed in detail.

Of the 2048 words of memory in the main core, the first 128 words are reserved for use as Boolean storage locations. Only the first 16 bits in each of the 128 words are used for this purpose. This results in a total of 2048 bits of Boolean storage, since a Boolean word is only one bit long. The core memory, then, is set up to appear as if there were two separate core memory blocks — one being a 2048-word arithmetic core and the other being a 2048-word Boolean core.

Like its counterpart, the arithmetic core memory, the Boolean core memory acts as the working storage for all Boolean operations. Here, Boolean variables are assigned storage locations, Boolean inputs from the rest of the simulator system are read into preassigned storage locations in this memory, and Boolean outputs from the Mark I to the rest of the simulator system are read out from their assigned storage locations in the Boolean memory.

Since the Boolean core memory is actually made up of 128 words of the main core memory, then any instruction requiring access to the Boolean memory is, in truth, accessing the main core memory. Therefore, any instruction of the main program that requires memory access, whether arithmetic or Boolean, represents a "highest priority" operation.

Because there is no circuitry in the Mark I to prevent him from doing so, it is possible for the programmer to address the entire contents of one of these "blocks" of Boolean storage (the entire 16 bits of one of the

128 main-core words) by using an ordinary (non-Boolean) instruction. It is conceivable that this sort of thing might be deliberately performed in order to provide a direct 16-bit binary output of an ordinary arithmetic quantity without going through the D-A converter. In this case, the main word location containing the 16 bits concerned would be "stolen" permanently from the Boolean section for use by the main arithmetic unit. It should be noted that Boolean storage locations may also be "stolen" (in blocks of 16) for the purpose of storing an externally coded, 16-bit binary word as an input, thus bypassing the A-D converter.

It is important, if these practices are employed for the purpose of providing direct binary inputs and outputs of ordinary arithmetic quantities, that the programmer look upon the entire Boolean core as being reduced in size, and never address the bits concerned for any Boolean purpose.

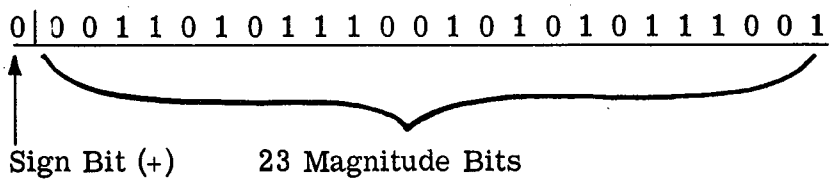
3.4 MAIN ARITHMETIC UNIT

The main arithmetic unit of the Mark I acts as the operating center of the computer. This unit consists of 1) a 24-bit accumulator register for holding the numerical results of arithmetic operations, 2) all of the necessary logic circuitry for performing arithmetic operations and transferring numerical data, and 3) a salvage register that salvages the old contents of the accumulator when a new word is loaded into it.

3.4.1 General Organization

All numerical operations in the main arithmetic unit are of the fixed-point, binary form. All numbers handled by the main arithmetic unit, either as inputs to or results of arithmetic operations, are in the form of an absolute value and a sign. Sign processes are performed in all arithmetic operations, and signs are preserved in the results. All number words are 24 bits in length, the first bit being the algebraic sign, and the remaining 23 bits being the absolute value of the binary number.

If the sign bit is zero, the number is positive. If the sign bit is one, the number is negative. Thus, a numerical word in the Mark I might be represented as follows:



In the Mark I's fixed-point arithmetic, the decimal point is assumed to be to the left of the most significant bit; therefore, the magnitude of a number is always less than one. The largest magnitude that may be represented is .99999. . . . in decimal notation, or .1111111 . . . . in binary notation. All numbers handled by the Mark I must be scaled with this fixed-point notation in mind.

If the result of any arithmetic operation in the Mark I is a number whose magnitude is greater than one (that is, greater than |.11111111. . .|), an automatic overflow process sets each bit of the accumulation to "1" but preserves the sign of the result. Thus, after an overflow caused by a positive result greater than one, the contents of the accumulator will be:

0| 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

An automatic overflow is possible in the execution of any of the following instructions: ADD, SUBTRACT, DIVIDE, SCALE.

The rules for addition in the binary number system are as follows:

0 + 0 = 0

1 + 0 = 1

1 + 1 = 10 (zero with 1 carried)

As an example of binary addition, the sum of the two numbers 1011001 and 1001010 is:

```
Carries      1011000
               1011001
           +   1001010
             10100011
```

Addition in the Mark I is accomplished by a parallel adder with fast carry propagation. Consider the two binary numbers A and B:

$A_n \ldots A_2\ A_1\ A_0$

A = 0 . . . 1 1 1

$B_n \ldots B_2\ B_1\ B_0$

B = 0 . . . 0 1 1

In a parallel adder, the addition of corresponding bits between two numbers is done simultaneously. That is, in the example shown, the A_0 digit is being added to the B_0 digit at the same time that the A_1 and B_1 digits are being added. Carries, of course, must be considered. When adding the two least significant digits ($A_0 + B_0$), there is obviously no carry from the right, and the results of this addition may be described as follows:

A_0	B_0	C_{out}	Sum
1	0	0	1
1	1	1	0
0	0	0	0
0	1	0	1

This table (often called a truth table) shows that the sum of A_0 and B_0 is one if A_0 and B_0 are complementary. C_{out} denotes carries to the left — that is, to the next pair of digits, A_1 and B_1. If A_0 and B_0 are both zero, then the sum is zero and there is no carry. If A_0 and B_0 are both one, the sum is zero and a "1" is carried.

The sum of any succeeding pairs of digits of A and B is made more complicated by the fact that carries from the right must be considered. Examine the following truth table and consider the conditions that make the sum one:

A	B	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
1	1	0	1	0
1	1	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0

This table is perfectly general in that it represents the sum of any two corresponding bits of any two binary numbers. It will be seen that S and C_{out} are now determined by the binary addition of three quantities: A, B, and C_{in}. Using Boolean notation, S and C_{out} can each be defined by an equation in A, B, and C_{in}. If these two equations are implemented for each pair of corresponding digits of the binary numbers A and B, the resultant system represents a complete parallel binary adder.

3.4.2 The 24-Bit Arithmetic Accumulator

Almost 70% of the instructions for the Mark I cause some operation to be performed on the numerical data contained in the arithmetic accumulator (for example, add, subtract, multiply). It should be stated that only numerical data can ever be inserted into the accumulator (that is, instructions are found only on the drum and can never be operated on or modified in any way).

The first bit (most significant bit) of the accumulator is reserved as a sign bit. This bit gives the algebraic sign of the number defined by the remaining 23 bits — "0" for a positive number and "1" for a negative.

Numerical data may be loaded into the accumulator from the core memory and, by virtue of a special instruction, from the drum. Data contained in the accumulator may be stored only in the core memory.

Most operations performed on data in the accumulator require only a very small amount of time and may be initiated and finished in the amount of time equal to one machine operation cycle (6.105 microseconds). A few of these operations (multiply, divide) require several machine cycles to complete; hence, once they are initiated, extreme care must be taken that new instructions do not arrive requiring operations on data in the accumulator until the previous, more lengthy, operation has been completed. This point is treated more fully in a subsequent section.

3.4.3 The 24-Bit Salvage Register

Associated with the arithmetic accumulator is a salvage register, also 24 bits in length. When an instruction is read directing that a word of data from some source be loaded into the accumulator, and there is no salvage register, then whatever word happens to be in the accumulator at that time is lost when the new word is loaded. It is the function of the salvage register, just as the name implies, to salvage the data word in the accumulator just before a LOAD instruction. Hence, if the number X is in the accumulator at the time that an instruction is read directing that Y be loaded into the accumulator, then one machine

cycle later, Y will appear in the accumulator and X will appear in the salvage register. The previous contents of the salvage register will be lost.

The salvage register is an addressable location. That is, it may act as a source of data with which to perform arithmetic operations on the contents of the accumulator. The contents of the salvage register, however, may not be loaded into the accumulator.

3.5 BOOLEAN ARITHMETIC UNIT

The Boolean arithmetic unit is the Boolean counterpart of the main arithmetic unit. This unit performs all Boolean operations indicated by instructions in the general program. Thus, the logic circuitry of the Boolean arithmetic unit is arranged to perform the functions of AND, OR, COMPLEMENT, LOAD, and STORE. Like its counterpart, the main arithmetic unit, the Boolean arithmetic unit has an accumulator and a salvage register. Since Boolean words are one bit in length, the Boolean accumulator, which is used to hold the results of all Boolean operations, is a one-bit register. Boolean words may be loaded into the Boolean accumulator from the core memory, and information in the Boolean accumulator may be stored in memory locations in the Boolean portion of the main core. All Boolean operations, indicated by instructions in the general program, are performed on the contents of the Boolean accumulator by the contents of the address specified in the instruction.

The Boolean salvage register performs the same function as the salvage register of the main arithmetic unit (that is, it salvages the old contents of the accumulator when a new word is loaded). However, unlike the main arithmetic salvage register, which is only a one-word register, the Boolean salvage register can hold four one-bit words, all of which are addressable. Because of this multiword capacity, the Boolean salvage register may act as an intermediate, temporary storage unit, thus reducing the core memory access requirements of the Boolean arithmetic unit. The operation of the salvage register is shown in Figure 6, assuming a hypothetical series of LOAD instructions.

Since the four words of the Boolean salvage register are addressable locations, the contents of any of these word locations may be used to perform a logical AND or OR with the contents of the Boolean accumulator. However, the contents of any of these salvage register locations may not be loaded into the Boolean accumulator. Neither may the contents of these locations be stored in the Boolean core. Only the contents of the accumulator may be stored in the core memory.

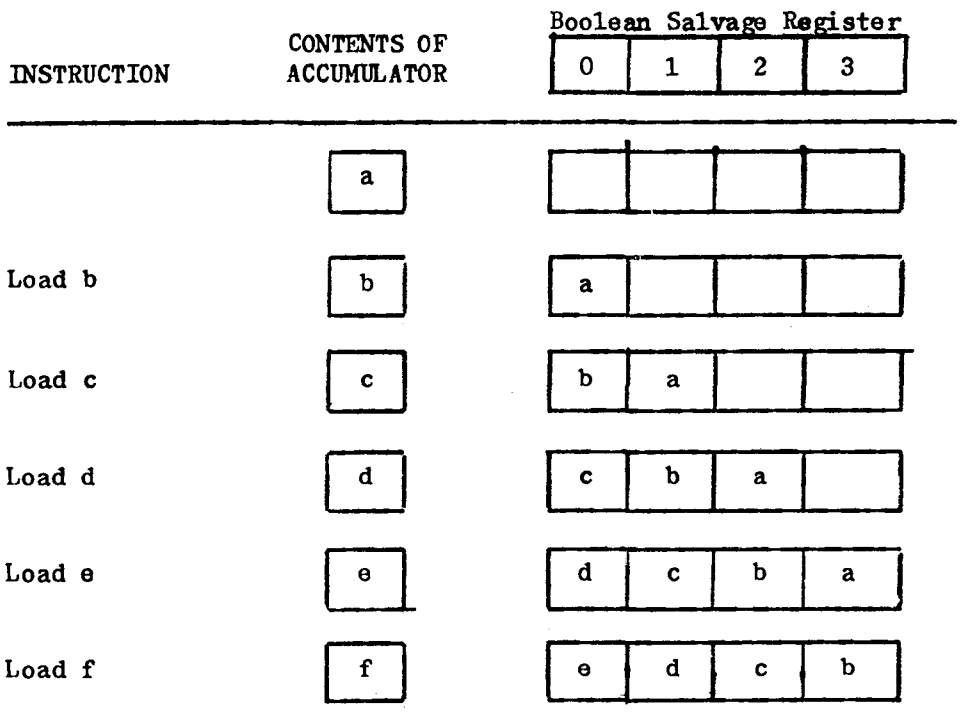


Figure 6 OPERATION OF BOOLEAN SALVAGE REGISTER

3.6 LINEAR FUNCTION INTERPOLATOR

Function generation in the Mark I is done continuously in parallel with the main program of the machine, and thus, because of computation time saved in the main program, contributes largely to the real-time dynamic response of the computer. Function generation is accomplished by means of linear interpolation between the ordinates of fixed breakpoints. Figure 7 illustrates an arbitrary function of X whose X axis has been divided into eight equal segments. These eight segments are defined by nine breakpoints: 0, 1/8, 1/4, 3/8, 1/2, 5/8, 3/4, 7/8, and 1.

If the ordinates of these nine breakpoints are known [f(0), f(1/8), f(1/4), f(3/8), and so forth] and if the independent variable X is known, then it is possible to perform a linear interpolation to determine f(X). For instance, if X lies between 1/8 and 1/4, then it is possible to interpolate between f(1/8) and f(1/4) to obtain a very close approximation of f(X). The linear interpolator has its own arithmetic unit with a parallel binary adder and appropriate registers and logic circuitry to solve the linear interpolation formula for f(X).

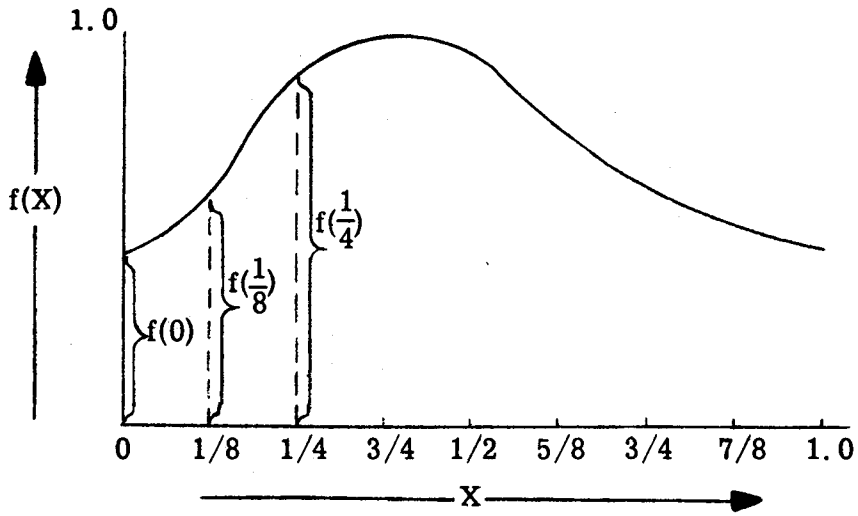


Figure 7 MARK I FUNCTION INTERPOLATION

As stated previously, all numbers handled by the Mark I must be scaled so that their magnitude is not greater than one. This also applies to the function interpolator. Therefore, both the independent variable, X, and the ordinate value, f(X), must be scaled so that their magnitudes are not greater than one. It should also be noted that all numbers handled by the interpolator are assumed to be positive.

The four bands on the magnetic drum used for function interpolation store the breakpoint ordinates of all the function curves. Also, the drum contains the core memory location of the independent variable and the core memory location in which the calculated value of the function will be stored. Each different function is represented by its own block of information listed on one of the bands, all of the blocks being listed in order around the bands. Figure 8 shows the flow of information for function generation.

It is unnecessary to store the X values of the breakpoints, since these breakpoints are fixed at 0, 1/8, 1/4, and so forth. It is only necessary to build a logic of the interpolator in such a way that it can inspect the value of X and recognize which two breakpoints it lies between. Consider the binary representation of the fixed breakpoints:

0 = .00000000. . . . .

1/8 = .00100000. . . . .

1/4 = .01000000. . . . .

3/8 = .01100000. . . . .

1/2 = .1000000 . . . . .

5/8 = .1010000 . . . . .

3/4 = .1100000 . . . . .

7/8 = .1110000 . . . . .

1 = .11111111111111111111.....

It is apparent from these representations that the first three digits of X will determine which two breakpoints X lies between, in accordance with the following scheme:

| 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |

0 1/8 1/4 3/8 1/2 5/8 3/4 7/8 1

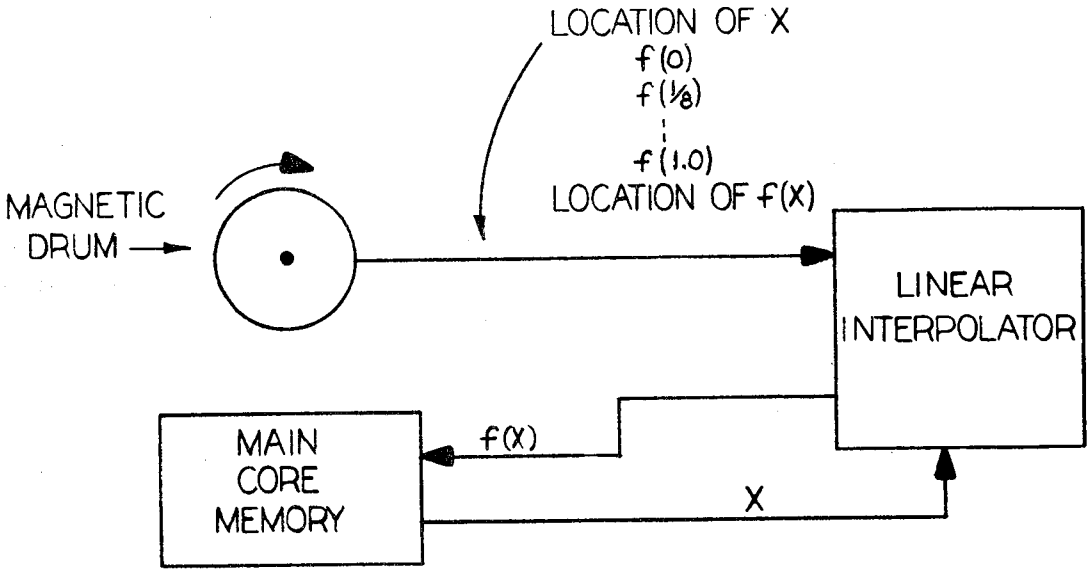


Figure 8 FUNCTION GENERATION FLOW DIAGRAM

Looking at a block of words on the drum concerning a given function (assume a function of a single variable), the first word listed is a control word that serves to identify a new function and tell whether it is a function of one, two, or three variables. The second word listed is the memory location of the independent variable. Following this will be the ordinates of the nine breakpoints in order, beginning with f(0). Notice that the memory location of X, the independent variable, is listed before the ordinate information. It will be remembered that memory access by the main program has priority over access requirements by the interpolator. Therefore, the interpolator may only have access to memory during "holes" in the main program. In order to ensure that the current value of the independent variable is obtained from its memory location, its address must be repeated several times. Thus, the first time the address of X is read from the drum, the interpolator will attempt to interrogate the core memory. If it fails to do so during the read cycle (6.105 microseconds), another word will be read directing it to interrogate the memory again. By repeating this process a sufficient number of times, the probability of encountering a "hole" in the main program, and thereby gaining access to the core memory, can be increased to the point where access is practically ensured. In most cases, repeating the location of the independent variable four or five times will prove sufficient.

Once access to the core memory has been gained, the current value of the independent variable is read into a register in the interpolator. Here, as stated previously, the first three digits of X (the independent variable) are examined to bracket X between two breakpoints. This process is finished before the data field is read; therefore, before the first ordinate is read, the interpolator already knows which two breakpoints bracket X. As the ordinates are read from the drum, only the two ordinates concerned are held for interpolation. The other ordinate words are ignored by the interpolator.

The result of the interpolation is, of course, f(X). This value is held in the linear interpolator until a word is read from the drum directing that f(X) be stored and giving the location in the core memory in which it is to be stored. As before, when the location of X was repeated several times in order to ensure memory access, the location of f(X) must be repeated several times. Again, four or five repetitions of the location of f(X) are sufficient in most cases.

In the event that X were located between 7/8 and 1.0, the interpolator would have to wait until the eighth and ninth ordinates were read from the drum before it could begin its calculations. Thus, this means that some time must be allowed after the last ordinate is listed before the memory location of f(X) is listed. This time is to allow the interpolator to finish its calculations before directing it to store the results. In the case of a single-variable function, two blank words (12 microseconds) are sufficient.

In the case of a function of two variables, four blanks are required, and for the three-variable function, six blanks are required.

The final arrangement of the data and instructions for some single variable function, f(X), might be as shown in Figure 9.

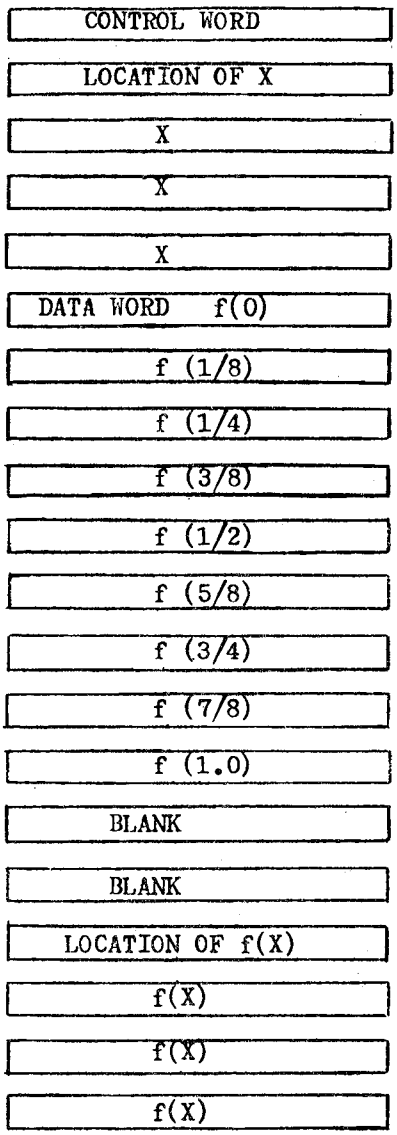


Figure 9 TYPICAL INSTRUCTION SEQUENCE FOR FUNCTION GENERATION

In order to handle a function of two variables in the Mark I, the function must first be represented in the form of a "family" of nine single-variable functions, as shown in Figure 10.

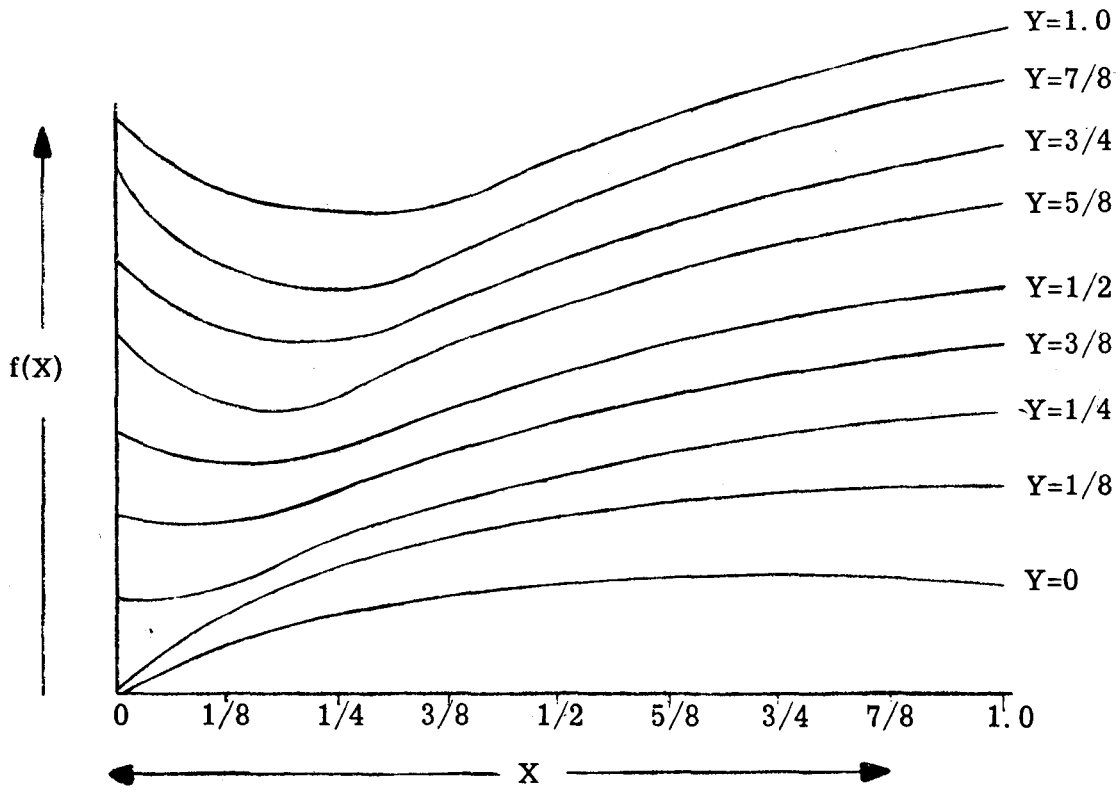


Figure 10 REPRESENTATION OF FUNCTION OF TWO VARIABLES

Examination of this figure shows that 81 ordinates are required to describe the function. The address of Y is listed after listing the address of X, and this address, also, must be repeated several times for memory access. Then the 81 data points are listed in order, beginning with the f(X, Y=0) curve and ending with the f(X, Y=1.0) curve.

Going a step further, a function of three variables would be composed of nine "sheets" such as the one shown in Figure 10. In this case, there would be a total of 729 (9 x 9 x 9) words of data to be listed for the function interpolation. The location of the Z independent variable would be listed after the Y location, and,like the X and Y variables, would have to be listed several times to ensure memory access.

Since it is possible to list 4096 words around a single band of the drum, and since there are four bands reserved for the linear interpolator, there is a total of 16, 384 words of storage capacity on the drum to be used for function generation alone. Referring back to Figure 9, it can be seen that approximately 22 words are required in order to program the generation of one function of a single variable. Hence, if only single-variable functions were stored on the drum, there would be sufficient capacity to generate over 740 different functions. If functions of two or three variables are stored, of course, the capacity is affected accordingly.

To provide for maximum use of the capacity available, the linear interpolator of the Mark I is constructed so that, by using certain indexing bits in the control word, it is possible to use the same function data or curve with four different independent variables, with the results stored in four different locations. The programmer can, for example, instruct the interpolator to index through the same set of engine function curves with four different sets of variables representing four different engines on an aircraft. The purpose of this is to reduce the necessity for having to repeat the data for the same curve several times on the drum, thereby saving storage capacity on the drum. The penalty is that the functions are calculated at only one-fourth of the normal rate — that is, non-indexed functions are recalculated four times as frequently as indexed functions. Since the four bands on the drum read at the rate of one band per drum revolution, and since one drum revolution requires 25 milliseconds, all four bands are read and every function listed is calculated every 100 milliseconds. In other words, the normal rate of recalculation for non-indexed functions is 10 times per second, and that for indexed functions is 2. 5 times per second.

All of the words on the four bands of drum storage for the linear interpolator are 11 bits long. One of these 11 bits (the most significant) is reserved as a control bit and the remaining ten are for data. Therefore, numerical data (ordinates of the breakpoints) are listed with only ten-binary-digit resolution. Arithmetic in the interpolator, however, is carried to 14 places and rounded off.

3.7 RADIO AIDS PRESELECTOR

3.7.1 Introduction

Each transmitter located anywhere in the world causes a signal to be introduced into the antenna of every receiver in the world. Certainly, the signals induced by very distant transmitters are extremely weak and cannot be heard because of the noise level. Others are rejected by the frequency-selection circuits of the receiver. This state-

ment applies particularly well to radio navigation facilities, because navigation transmitters which occupy the same frequency band have deliberately been separated by a large distance, or made to operate with a low power output, or confined by a narrow radiation pattern, in order to prevent interference. These facts make it possible to devise an automatic radio navigation simulation system which, on a basis of receiver frequency and aircraft location, can select for each receiver the one best facility to receive.

It is the function of the Mark I radio aids preselector to examine a total of 350 different radio transmitters and select the one transmitter, if any, that each of the simulated aircraft receivers should be picking up. This process operates in parallel with the general program of the Mark I and is completely automatic, without need for programmer attention.

In the Mark I radio-aids system, the information required for the simulation of 350 navigation transmitters is contained in a band of 20 data tracks of the Mark I's program drum. These tracks are used independently of the main program and linear interpolator tracks, and are used in a read-only mode. The method for loading data into these 20 tracks is identical to that used for loading the program and interpolator data into the magnetic drum. A new set of facilities can be entered by means of the photoelectric tape reader in about a minute's time, although it is necessary to interrupt the simulated flight during the loading operation.

The signal feature of the Mark I's automatic radio-aids system is that it treats each transmitter as a separate entity. Under this concept, an ILS facility would consist of a localizer transmitter, two 75-megacycle marker transmitters, and two low-frequency compass-marker transmitters, for a total of five separate units. Each simulated transmitter is counted separately, with the following exceptions:

1) The glideslope facility is provided as a component of the localizer system and does not require a separate transmitter in the total facilities count.

2) A DMET system installed at a VOR station, or a tacan DME system, is, from a computational standpoint, an integral part of the azimuth transmission facility and does not count in the total.

With the exception of these two types of facilities, each individual transmitter counts as a separate unit. Thus, Z markers and fan markers associated with an A/N range station are totaled individually.

3.7.2 Classification of Facilities

The 350 available transmitters are divided into five groups, according to the type of facility. The maximum number of facilities in each of the groups may not be exceeded, although it is not necessary that all facility channels be employed if a smaller amount is desirable. The groups are:

1) Low-Frequency Transmitters — This group includes low-frequency beacons, low-frequency compass locator facilities, and A/N range stations. A total of 127 such facilities can be represented, of which as many as 32 may be A/N range stations — although it is not necessary that all 32 be so assigned.

2) VHF/UHF Transmission Facilities — These include VOR transmitters, tacan transmitters, Navy UHF direction-finder transmitters, and ILS transmitters. A total of 127 independent VHF/UHF transmitters can be represented.

3) Outer ILS Markers — The system can represent 32 outer ILS markers.

4) Middle ILS Markers — The system can represent 32 middle ILS markers.

5) Fan and Z Markers — The system is capable of representing 32 fan or Z markers, which can be intermixed in any desired proportion.

These facilities can be received when within range and if the appropriate receivers are operative and tuned.

It should be noted that the Mark I radio-aids system does not automatically provide the voice signals associated with certain facilities — that is, it is left to the instructor to provide this in the usual manner. The prime reason for omitting this feature is that of excessive cost in relation to training value. Call letter identification is, of course, provided for the facilities and is deemed to be quite adequate for training purposes.

3.7.3 Facility Selection

3.7.3.1 General Considerations

Facility selection is based upon electronic inspection of the data words provided for each of the facilities. The preselector system employs specialized electronic circuitry to scan the drum data concerning the 350 individual navigational transmitters and select the one eligible transmitter facility for each navigation receiver in the aircraft. Since all 350 facilities must be scanned every drum revolution (40 times per

second), it is apparent that a fast and relatively simple system must be employed to choose the one facility from each group that best merits transfer to the core memory for the programmed computation associated with each receiver.

While a criterion such as slant range to the station might appear to be highly desirable for preselection purposes, the computation of the quantity (involving several subtractions, additions, and multiplications, and one square-root operation) is too complicated to be accomplished in the time available for the preselection operation. To permit the use of simple electronic circuitry, preselection is limited to inspection of station frequency, station X coordinate, and station Y coordinate. Before considering the method of preselection based upon the navigational facility's geographical position, consideration must be given to certain aspects of the use of transmitter frequency in station preselection.

3.7.3.2 Frequency Inspection

Fan, Z, middle, and outer markers all operate on 75 megacycles and are received by a fixed-frequency, untunable receiver. Accordingly, any preselection operation associated with 75-megacycle marker facilities must be based solely upon geographical location of the facility with respect to the aircraft's position.

Low-frequency navigation receivers, such as automatic direction finders, commonly employ continuous tuning similar to that employed in conventional broadcast receivers. Accordingly, the use of frequency in the preselection of low-frequency facilities must allow a band of frequencies somewhat wider than the bandpass of the receiver itself, with a final determination of the degree of tune being performed by programmed computation after a station is selected.

VHF/UHF navigational facilities are ordinarily tuned by receivers employing numerical switching for frequency selection, with the result that preselection may employ the exact frequency of the facility rather than a band of frequencies, as is required for low-frequency transmitters. Thus, the final decision whether a VHF navigational facility is tuned can be based exclusively upon the action of the preselector system alone.

Inspection of facility frequency, then, must be consistent with the receiver tuning and facility frequency characteristics. The criterion against which facility frequency acceptability is judged is the degree of match between the frequency of the receiver and the assigned frequency of the facility. In order for a facility to be "heard", it is necessary that its frequency match the receiver frequency within certain limits. These limits actually correspond to the bandwidth of the receiver,

but it is more convenient to design an electronic inspection system which considers that the limits are assigned to the transmitter facility. The end result is the same, and the same degree of match is determined, but it is determined more easily.

Therefore, for each transmitter represented in the Mark I, an upper and lower frequency limit is assigned. The facility preselector system will find a particular facility acceptable for a particular receiver from a frequency standpoint if, and only if, the frequency of the receiver falls between the lower and upper frequency limits assigned to that facility.

Because all marker transmitters operate on one frequency (75 megacycles) frequency inspection is not actually required to determine whether any marker transmitter can be received. However, it is more convenient from a design point of view to actually conduct a frequency inspection of the marker transmitters, making the limits sufficiently wide so that every transmitter will certainly pass regardless of the frequency used for inspection.

VHF/UHF receivers employ digital tuning with discrete frequency assignments; hence, the limits assigned to the transmitter may be quite close together,ensuring that only those transmitters which exactly match the receiver's frequency will pass the frequency test for a given receiver.

LF transmitter facilities are assigned frequency limits by subtracting and adding, to the assigned operating frequency, a number that is somewhat greater than half the receiver bandwidth. The resultant two numbers are then used as the lower and upper frequency limits. LF transmitters will pass the frequency test only if the current receiver frequency falls between the lower and upper frequency limits assigned to that transmitter.

3.7.3.3 Geographic Position Inspection

In addition to the consideration of frequency assignment, it is necessary to consider the location of the aircraft with respect to the station. It has been mentioned that slant range is too complicated to compute for each of the 350 simulated facilities, and that a simpler criterion must be utilized. The chosen scheme has the necessary simplicity and proves to be entirely adequate in representing the physical situation.

The method employed for geographic inspection assigns to each of the 350 facilities two pairs of coordinates. These coordinate pairs represent the upper, lower, left, and right boundaries of a rectangle that contains a given facility. These rectangles are arbitrarily assigned so that the left and right boundaries (limits) are the east-west direction

(X axis) and the upper and lower boundaries are in the north-south direction (Y axis). In general, each rectangle is made as large as possible, using caution that there are no overlaps of rectangles assigned to different facilities, either operating on the same frequency (in the case of markers and VHF/UHF facilities) or operating on adjacent frequencies that could be within the bandpass of the receiver (in the case of LF facilities).

The assignment of rectangles to each facility is made easier by the facility grouping in the computer memory. For example, there are three groups of markers, outer ILS, middle ILS, and fan or Z markers. Although all three groups operate on the same frequency, only one marker transmitter from each group can be selected at one time. Therefore, it is only necessary to assign the rectangles in such a manner that for a given group, such as outer ILS markers, overlapping rectangles are not assigned. The rectangles assigned to facilities of different groups can be permitted to overlap without causing interference problems. Because of the programmed calculations involving the range and radiation pattern, no interference will result unless two or more markers of different types are represented that in reality do interfere and can be simultaneously received.

3.7.3.4 Summary

Preselection of one of the radio facilities within a group is thus accomplished by determining whether the aircraft is within a pair of left and right X coordinate bounds and a pair of upper and lower Y coordinate bounds, and whether the receiver is tuned to a frequency within a pair of upper and lower frequency bounds. Each transmitter is effectively placed within a rectangular box having north-south and east-west boundaries. The box cannot be placed diagonally on a map.

The third dimension of the box is, of course, frequency rather than altitude, but the three-dimensional concept is convenient for visualizing the physical domain within which the individual transmitters are eligible for reception. If the boxes of a given facility type are all independent of one another (that is, if none of them share a common space), it is apparent that no more than one transmitter of a given group will be considered eligible for reception. It is also apparent that it will be possible to operate in a space that is free of any of the individual boxes, in which case no station may be received. This situation is, of course, immediately altered if the receiver tuning is changed, since this constitutes a change in the vertical dimension of the imaginary three-dimensional space, which may result in the preselection of a station whose box has been entered.

When any station of a particular group simultaneously

meets all three preselected criteria, the stored data describing that station are transferred to a specific group of core memory locations associated with the receiver capable of receiving that station. The transferred data are used by the programmer to calculate signal strength, range leg orientation, beam pattern, and call letters, according to the type of facility.

3.8 ANALOG-TO-DIGITAL INPUT CONVERTER

It is the function of the A-D converter to take all analog inputs to the Mark I (from cockpit, instructor's station, and so forth) and convert them into 14-bit binary numbers, storing these numbers in preassigned (fixed) core memory locations. There are 64 analog inputs, with an expansion capability to 126. Each input is converted to binary form and stored in its individual core memory location. This operation is carried on in parallel with the main program and is fully automatic. To the programmer, there is a block of core memory locations containing the digital equivalent of the various analog input quantities necessary for his equations, and he has only to address the appropriate core location to employ any of these quantities in computations.

All analog input quantities are scaled in the range of -10 volts to +10 volts, these reference voltages being provided by the Mark I. The A-D converter converts these quantities to binary numbers scaled from -1 to +1.

The 64 multiplexed inputs of the A-D converter are sampled and converted into properly signed 14-bit binary words during every two revolutions of the drum. Inputs are sampled only during the first and third quarters of each drum revolution (the second and fourth quarters are reserved for output sampling), and the 64 channels are spread over the four allowed quarters of the two drum revolutions, so that 16 channels are sampled, converted, and stored in each allowed quarter. A counter provides the core memory addresses in which each quantity is stored.

Memory access, of course, is required for storage, and the A-D converter is under control of the priority circuitry. When an input has been sampled, the binary equivalent is held until access to the memory is obtained. At that time, it is stored in the address dictated by the counter. The counter is advanced, the next output is sampled, and the process continues until all of the inputs have been sampled. The process repeats endlessly, and occurs without programming attention or instructions.

3.9 BOOLEAN INPUT SYSTEM

Internal Mark I circuitry will provide interrogation signals for

a maximum of 64 groups of 16 Type A contacts, which may take the form of toggle switches, and static punchcard readers, located externally. This will provide responses at 1024 one-bit inputs. Internal circuitry will generate core memory addresses 001 through 032 (decimal), and each 16-bit group of Boolean words generated by the interrogation of switches will be transferred sequentially to these assigned memory locations.

This circuitry is also under control of the priority circuits. It will operate, throughout all drum revolutions, whenever a NO OP or "no-memory-access" instruction appears in the main program and the "hole" is not utilized by the linear interpolator, the radio aids preselector, or either the A-D or D-A converter. Because of long lead requirements, the interrogation of a set of 16 switches may precede the allowable transfer time of the response by a few hundred microseconds, or longer.

3.10 DIGITAL-TO-ANALOG OUTPUT CONVERTER

Digital-to-analog conversion in the Mark I is another process that operates in parallel with the main program. The Mark I has a total of 128 independent analog outputs, with an expansion capability to 192 analog outputs. These outputs are used to drive indicators, motion systems, and recorders in the external simulator equipment (for example, cockpit, instructor's station).

The equations representing each of the output quantities must, of course, be implemented in the general program to be computed by the Mark I. A fixed word location, in the main core memory, is reserved for each of these quantities, and as each quantity is recalculated in the main program, its new value is stored in its respective memory location. That block of core memory locations containing the 128 output quantities is periodically interrogated, and all of those 128 words are sequentially transferred to a buffer core memory. Thus, the 128 words in the buffer core are fed to 128 individual D-A converters whose outputs are analog voltages representing the digital quantities.

The storage locations in the main core reserved for output quantities are all sampled by the buffer core at a rate of 80 times per second. The buffer core, in turn, is sampled by the D-A converter 80 times per second. This number, then, represents the sampling rate of the entire D-A output system.

In order that a maximum of 192 words in the main core memory be read into the buffer core memory, 192 separate memory accesses are required; however, it should be remembered that only one word location of the main core may be interrogated during any 6.1-microsecond machine cycle. These words will all be accessed once during the second quarter of a drum rotation, and then accessed again during the fourth

quarter of a drum rotation. Since there are 40 drum revolutions per second, and since each word is read into the buffer core twice during each drum revolution, the sampling rate is 80 times per second for all outputs. It should be recognized that the drum plays no actual part in this process, but that its rotation merely provides a convenient time base for examining the D-A conversion.

Since there are 4096 words written around the circumference of the drum on each band, 1024 words are read during the time that the drum takes to make one quarter of a revolution. The time required to read each word represents the basic machine cycle of 6.1 microseconds; therefore, one quarter revolution of the drum represents enough time for 1024 possible memory accesses. The D-A converter is, of course, under the control of the priority control circuitry, and it can have access to the core memory only when it is not being accessed by either the main program, the digital interpolator, or the radio-aids preselector. Obviously, then, there must be at least 192 "holes" in the total access requirements on the core memory during both the second and fourth quarters of a drum revolution, or else all of the output words would not be sampled.

A counter in the Mark I generates the core memory addresses of the output quantities that are to be read into the buffer memory. Upon access of the first address, the counter advances to the next address and holds that address until an opportunity occurs to interrogate that word. The word is read into the buffer memory and the counter advances again. This process continues until all 192 addresses have been interrogated.

The buffer memory itself is a 16 x 192-bit core matrix composed of twelve 16 x 16-bit memory boards. The matrix arrangement is equivalent to a column of 192 words — each word essentially a 16-bit word. The word length in the main core is, of course, 23 bits plus one sign bit; however, only the sign bit and the ten next most significant bits are read into the buffer core, thus utilizing only 11 of the available 16 bits per word.

As words are read from the main core, they are read into an 11-bit register. Each digit of this register, in turn, drives the vertical wire strung through the corresponding bit position of all 192 words of the buffer core (see Figure 11). A write amplifier drives a horizontal wire running through all the bits constituting a word. After the first output quantity has been read from the main core and into the output register, the horizontal drive amplifier for Word 1, only, is turned on. This horizontal driver provides half the total current required to switch the cores. A "1" in any of the 11 bits of the output register causes the corresponding vertical driver to also provide half the necessary switching current. A "0" causes no current to flow. Hence, since the fields generated by the two wires strung through a core are additive, the presence of a "1" in the register causes the corresponding bit in the word concerned to switch to

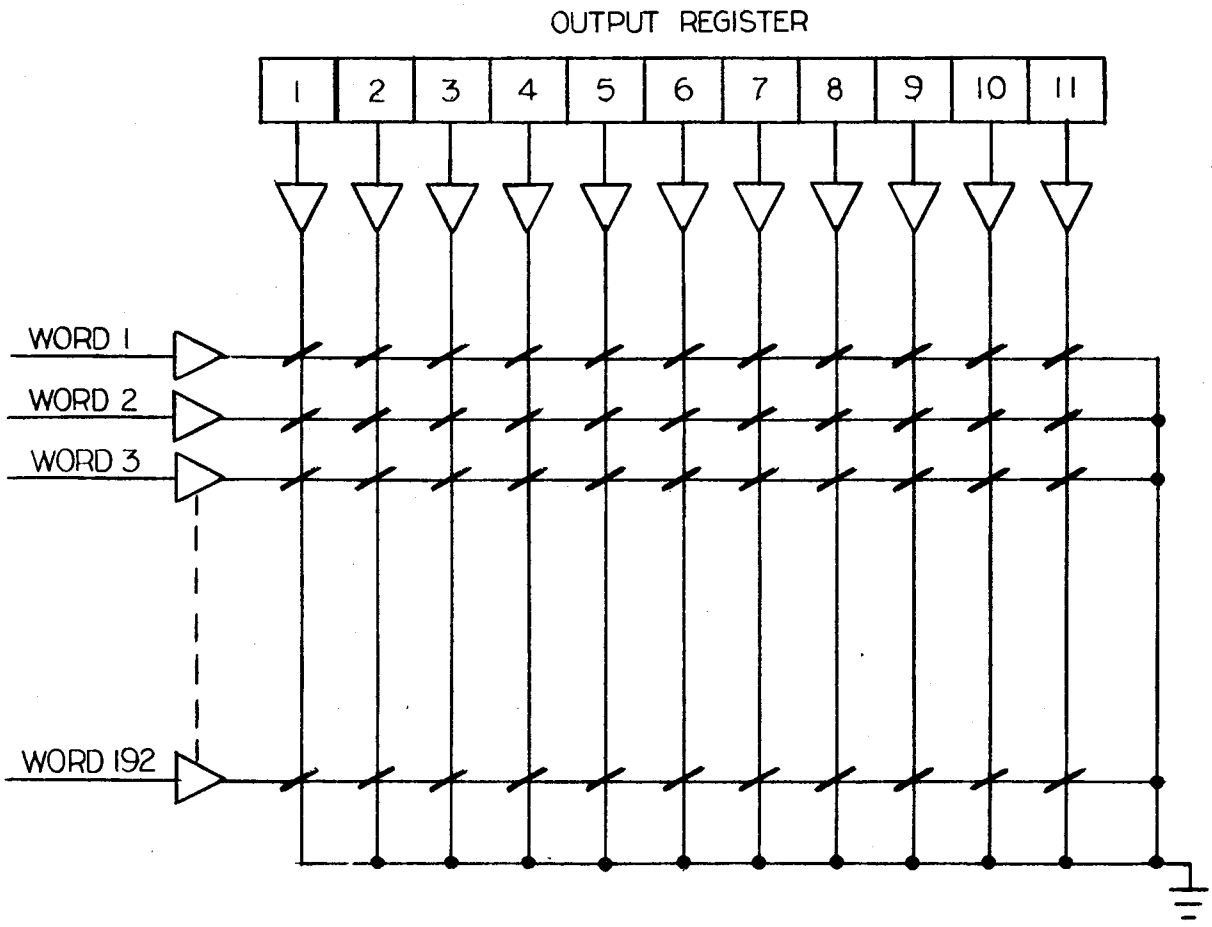


Figure 11 192-WORD BUFFER CORE AND OUTPUT REGISTER

the "1" state, and the presence of a "0" in the register leaves the corresponding core in the "0" state, since insufficient current was provided to cause it to switch. A simplified block diagram of the complete output conversion system is shown in Figure 12.

All 11 bits of the register are read into the buffer core simultaneously. The next output quantity in the main core is then read into the output register (as soon as memory access is available), and the process is continued until all 192 output quantities have been stored in the buffer core.

In actuality, words stored in the main core memory are not read into the output register in the straight binary form in which they are held in the main memory. Instead, they are coded in the following fashion:

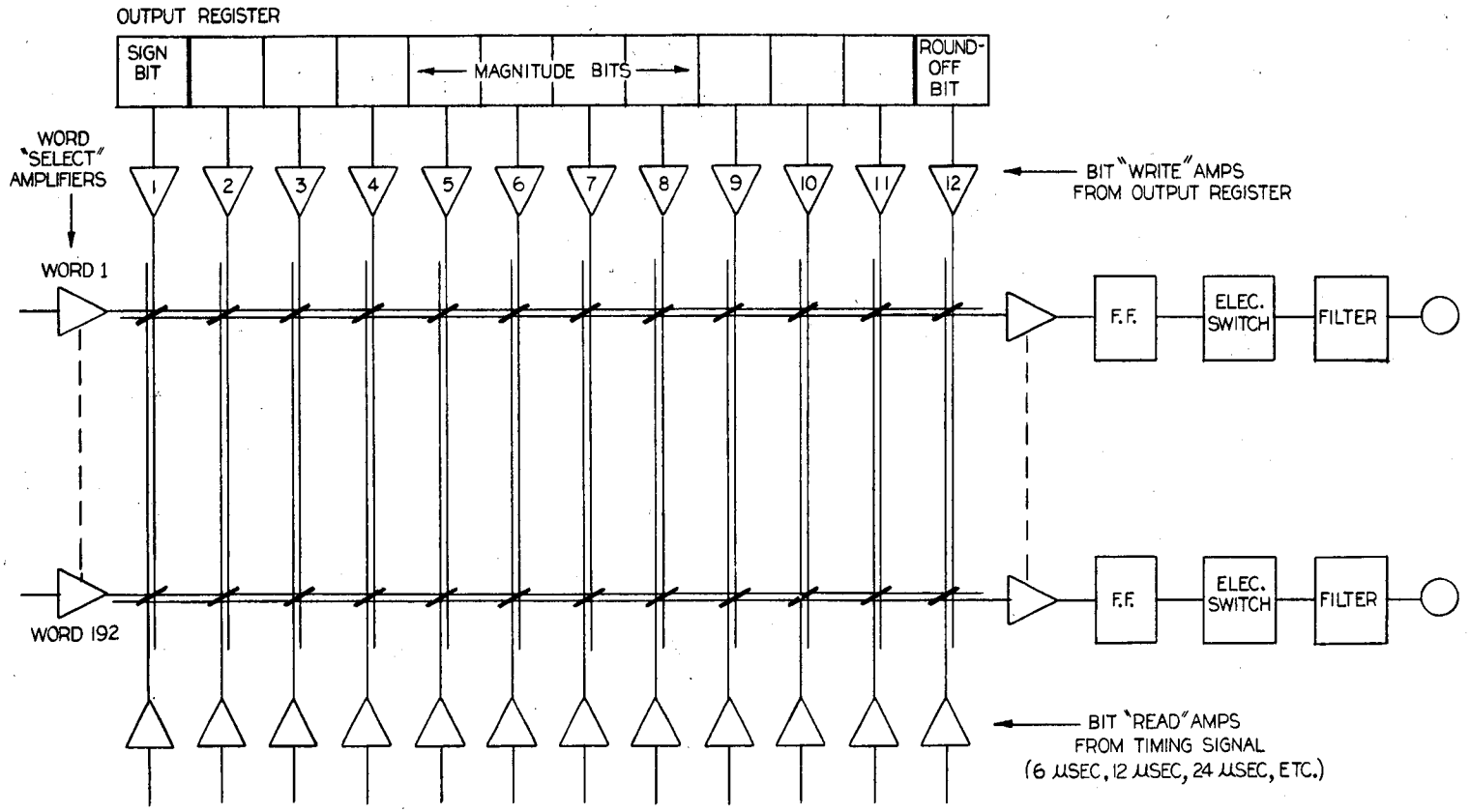


Figure 12 MARK I D-A OUTPUT SYSTEM

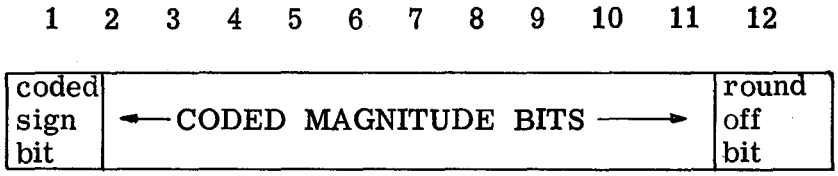
1) The sign bit is always inverted. (A "1" in the memory goes into the register as a "0", and a "0" in the memory goes into the register as a "1".)

2) If the sign bit in the memory is negative (1), all the magnitude bits are inverted. If the sign bit in memory is positive (0), all the magnitude bits are read into the register without inverting.

The following examples will demonstrate this transformation:

Memory Word	Output Register
0 1 1 0 1 0 0 0 1 1 1 1 0	1 1 1 0 1 0 0 0 1 1 1
1 1 1 0 1 0 0 0 1 1 1	0 0 0 1 0 1 1 1 0 0 0
1 0 0 0 0 0 0 0 0 0 0	0 1 1 1 1 1 1 1 1 1 1

These examples indicate that the 24-bit word in memory is merely truncated after the 11th bit with no attempt at round-off. Again, this is not the entire truth. Actually, the output register is 12 bits long, and all the words stored in the buffer core are 12 bits long. This 12th bit is generated electronically to compensate for round-off error and is used to set the 12th bit of the register:



This 12th bit is generated by forming the one's complement of the 12th bit on the word in main core. That is, if the sign of the word in the main core is positive, the 12th bit transfers to the output register unchanged. If the sign of the word in the main core is negative, the complement of the 12th bit is transferred to the output register.

Words stored in the buffer core must now be converted to analog voltages. This process is accomplished by sampling all 192 words, one bit at a time, beginning with all the least significant bits (12th bits) and working backwards to all the most significant bits (sign bits). The 12th and 11th bits are both "looked at" by the D-A converter for 6 microseconds each, the 10th bit for 12 microseconds, the 9th bit for 24 micro-

seconds, the 8th bit for 48 microseconds, and so forth. Each succeeding bit is looked at for twice the amount of time devoted to the previous bit.

Each of the 192 D-A converters consists of a drive amplifier, a flip-flop, an electronic switch, and a three-section RC filter, as shown in Figure 13. If the core being interrogated is in the "1" state, then the flip-flop goes to the "1" state and the electronic switch goes to +10 volts. If the core is in the "0" state, then the flip-flop remains in the "0" state and the electronic switch goes to -10 volts. The filter smooths the output of the switch. Once a core has been interrogated it returns to the "0" state.

Note that once an entire word has been sampled by the D-A converter, all the cores making up that word location in the buffer core are back in the "0" state and are ready to be written into again. Since the cores go to zero when interrogated, it is the function of the flip-flop to retain the information regarding the original state of the core. The amount of time that the flip-flop remains in any state is determined by the significance of the bit that set it — 6 microseconds for the LSB and LSB +1, 12 microseconds for the LSB +2, 24 microseconds for the LSB +3, ..., and finally, 6144 microseconds for the sign bit. The electronic switch, of course, follows the flip-flop.

When writing into the buffer core, all 12 bits of a word are written in parallel, one word after another. When reading from the buffer core, all 192 words are read out at the same time, one bit at a time.

If the word being read from the buffer core is all zeros, the output of the transistor switch is a constant -10 volts. If the word being read is all ones, the switch output is a constant +10 volts. Figure 14 shows the switch output for a word consisting of alternate ones and zeros.

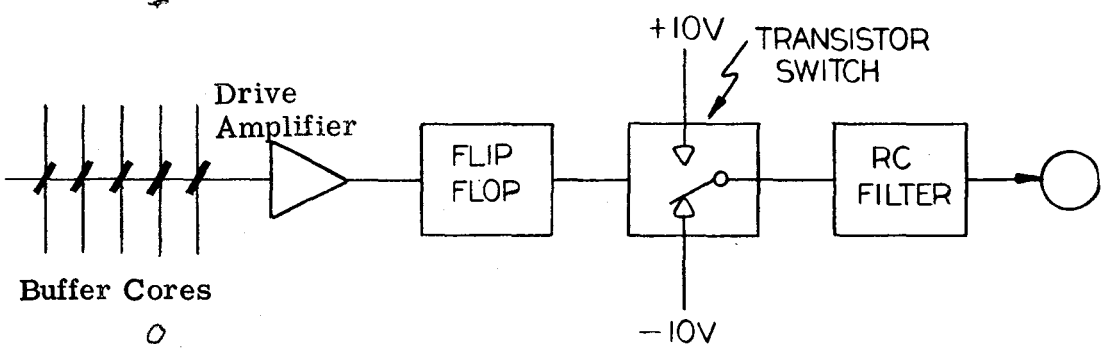
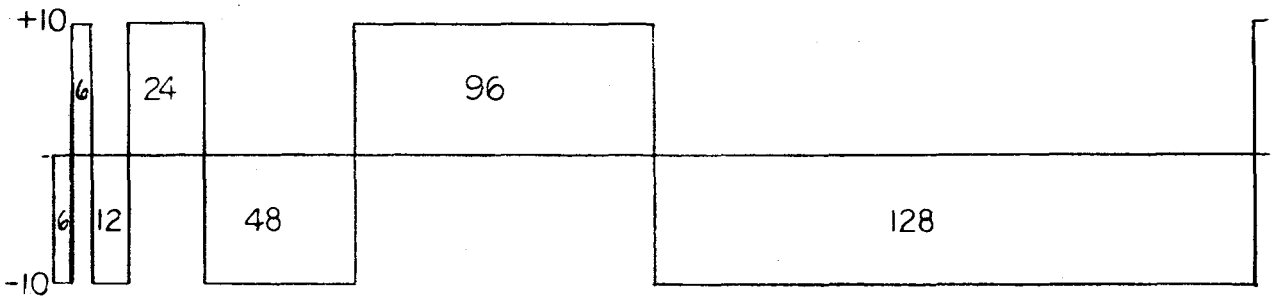


Figure 13 SIMPLIFIED DIAGRAM OF D-A CONVERTER

HYPOTHETICAL WORD: 1 0 1 0 1 0 1 0 1 0 1 0



Time in microseconds

Figure 14 D-A SWITCH OUTPUT

3.11 BOOLEAN OUTPUT SYSTEM

The Mark I is designed so that outputs from preselected core memory addresses for four 16-bit Boolean data words will be transferred by internal circuitry to the 64 register-bit locations provided. Each bit's output, for these registers, incorporates a driver with sufficient power to operate a relay that provides a "C" contact for external use. The four core memory word addresses, 124 through 127 (decimal), are permanently allocated for Boolean output words.

4. PROGRAMMING THE MARK I

4.1 EASE OF PROGRAMMING

Unlike the general-purpose digital computer, on which hundreds of problems might be programmed and solved per day, the digital simulation computer need never be reprogrammed except to accommodate changes in the aircraft or in the radio-aids facilities. It appears highly unlikely, therefore, that the purchaser of a digital flight simulator would write a complete instruction list for the machine, since a completely checked-out program will be supplied by the simulator manufacturer. In other words, modification of an existing program is a much more likely occurrence than the generation of a completely new program for the Mark I flight simulator.

In discussing the relative ease or difficulty of programming any digital computer, it is necessary to consider the skill level of the programmer and the equipment at his disposal to aid in formulating the computer program. One of the design goals of the Mark I computer has been to permit initial programming of the computer and program modifications by individuals who are not skilled digital computer programmers. This is felt to be an absolute necessity in a digital flight simulator, so that modification of the computer program can be performed in the field by essentially the same personnel who operate and maintain present analog flight simulators.

The instruction complement of the Mark I has deliberately been restricted to the minimum number of simple instructions necessary for flight simulation problems. There are 26 different instructions in all, and the function of each is explained later on in this section. With one exception, all instructions are performed in the computer in exactly the order in which they were written. This exception is the CONDITIONAL SKIP instruction, which permits the skipping of a number of instructions based on the existence of a certain condition (such as the insertion of a malfunction from the instructor's console). This continuous-flow pattern of instructions makes programming easier and also facilitates the checking of a program through one-step operation in a straightforward manner, since each step that the computer performs follows the instruction list exactly.

The branching operation of the Mark I differs from that of a general-purpose computer in that the various branches are arranged serially rather than in parallel, and the one branch is performed while the other branches are skipped. During the time that instructions are skipped, the various parallel processes of the Mark I are permitted access to the core memory, thus effectively utilizing this program time for computer operation.

Another factor in the programming ease of the Mark I is the built-in priority control circuitry. No program control, input-output subroutines, or program-interrupt features are required to enable the linear interpolator, radio aids preselector, and input-output converters to gain access to the core memory. From the programmer's point of view, there are a number of core memory locations containing the data derived from the various parallel processes, and he has only to address the appropriate core location in order to employ any of these quantities in computations.

Another aid to programming and program modification is the fact that the particular location of an instruction on the drum has no significance. Obviously, for purposes of fault detection and one-step operation, it is necessary that the location of the instruction be known, but in normal operation this knowledge is not required. In the event that it is desired to add a term to an equation for which the program has already been prepared, it is only necessary to insert the new portion of the program in the appropriate position and effectively "slide back" the remaining instructions in the program. It should be emphasized that the addition of one or more instructions in a band on the drum will never affect all of the remaining instructions on the instruction list, and in most cases will affect only those instructions in the particular sector of the drum involved. The smallest program segment that can be loaded into the Mark I is one sector of 1024 words, which is equivalent to one-fourth of one program band.

In order to more readily accommodate the alteration of particular instructions in the Mark I computer, all of the computer instructions will be distributed relatively evenly around the drum. For example, if all of the required slow-band instructions could be readily accommodated on five of the available eight slow bands, these instructions would be distributed uniformly upon all of the eight slow bands. This would permit a large number of NO OP instructions to be inserted in each sector of each slow band, thus helping to ensure that any additional program steps in one sector would affect the instruction location in that sector only. In the event that more than one sector is affected by a modification of the program, or even that more than one band is affected, no problems arise in the programming; however, some additional bookkeeping is required to update the list of instructions.

Another feature that makes the Mark I easy to program and easy to modify is its specially-designed linear interpolator for the generation of arbitrary functions. Both in initial design and during programming modifications, the only information that has to be provided is the ordinates of the breakpoints of an eight-segment straight-line approximation of the data curves (this is virtually an irreducible minimum level of information for arbitrary functions). In the event that aircraft data indicate that a curve should be modified, the affected breakpoints can be inserted in lieu of the previous data.

4.2 FORMAT OF INSTRUCTION WORDS

There are 26 machine instructions which the programmer may use to direct the Mark I in the step-by-step solution of equations. It should be kept in mind that a long list of instructions is all that is written on the 11 general-program bands of the drum and that these instructions are read and performed in the order in which they are written — one instruction during each 6.1-microsecond machine cycle. Each instruction describes an operation which the Mark I performs and gives the core memory location of the data word involved in the operation.

All the instructions fall into three general categories — control, transfer, and arithmetic — and each instruction has an identifying code number recognizable by the machine. Each instruction word is made up of two sections: a two-octal-digit section that identifies the operation (identifying code), and a four-octal-digit section that specifies a core memory address. For example:

01	1500

= ADD the contents of memory cell 1500 to the contents of the accumulator.

(where 01 and 1500 are octal numbers)

When this six-digit word is punched up on the tape preparation unit, this unit automatically converts the octal word into a 16-binary-bit word. The first five bits are the binary code for the operation, and the following 11 bits are the binary code for the core memory location (the 11 general-program bands on the drum are all 16 bits wide). The example given above would be coded as follows:

01	1500

0	1	1	5	0	0
00	001	01	101	000	000

Notice that each digit of the octal instruction word is coded separately. This format is binary-coded octal. Each octal digit of the instruction is separately converted to binary (three binary bits are required to count from zero to seven) and the binary equivalents are written in the same order as the octal numbers. The first octal digit of the operation code is never larger than three, thus requiring only two binary digits to describe it. This is also true of the address section's first octal digit. All other octal digits can take on values from zero to seven; thus, each requires three binary digits. As another example of the binary-coded-octal conversion, consider the instruction word 23 | 1777. This is a

STORE instruction, directing the arithmetic unit to store the contents of the accumulator in core memory location 1777. The binary equivalent, as written on the drum, is:

2		3			1		7			7			7		
1	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1

4.3 LIST OF INSTRUCTIONS

The instructions which can be performed by the main arithmetic unit are:

NO OP	- Do nothing during this machine cycle (used to allow time for completion of arithmetic operations and to test read heads).
LOAD	- Insert the contents of a specified core memory location into the accumulator.
INDEX LOAD	- Insert into the main accumulator the contents of either of two consecutively numbered core memory locations, depending on the state of the Boolean accumulator.
ADD	- Add the contents of a specified core memory location to the contents of the accumulator.
SUBTRACT	- Subtract the contents of a specified core memory location from the accumulator contents.
MULTIPLY	- Multiply the accumulator contents by the contents of a specified core memory location.
DIVIDE	- Divide the accumulator contents by the contents of a specified core memory location.
NEGATIVE MULTIPLY	- Same as MULTIPLY, but reverse algebraic sign of the product.

SQUARE - Multiply the number in the accumulator by itself (does not require memory access).

SQUARE ROOT - Perform one iteration of the Newton-Raphson approximation:

$$\text{for } X = \sqrt{Y}$$

$$X_{n+1} = \frac{1}{2}\left[\frac{Y_n}{X_n} + X_n\right]$$

SCALE - Shift the accumulator contents to the left or right. The direction and number of places are specified by the address portion of the instruction, leaving the sign bit unchanged. (Equivalent to multiplying or dividing by integral powers of two.)

SHIFT - Logical shift (including the sign bit) to the right or left. The direction and number of places are specified by the address portion of the instruction. (This instruction effectively "rubber-stamps" the number a specified number of places to the right or left of the existing location, filling the margins with zeros.)

INVERT - Reverse the sign of the accumulator contents.

ABSOLUTE VALUE	-	Make the sign of the accumulator contents positive.
ZERO SLICE	-	If the accumulator contents are negative, set the accumulator to zero. If positive, do nothing.
STORE	-	Store the accumulator contents in the core memory location specified by the address portion of the instruction.
INDEX STORE	-	Store the contents of the main accumulator in either of two consecutively numbered core memory locations, depending on the state of the Boolean accumulator.
STORE CONSTANT	-	This instruction-address combination causes the computer to read the next 15-bit word appearing in sequence from the drum into the accumulator and into the core memory location specified by the address portion of the instruction.
CONDITIONAL SKIP	-	If the Boolean accumulator contains a "1", skip the number of instructions specified by the address portion of the instruction.
CONDITIONAL STOP	-	Stop performing instructions if an external control signal (console switch) is not present. Resume program with the next instruction if a signal is present. (Used for diagnostic routines and problem freeze.)
FLAG NEGATIVE	-	If the algebraic sign of the main accumulator contents is negative, set the contents of the Boolean accumulator to "1."

The instructions performed by the Boolean arithmetic unit are:

LOAD BOOLEAN ACCUMULATOR	-	Insert into the Boolean accumulator the contents of the memory location specified by the address portion of the instruction, salvaging the previous contents of the Boolean accumulator.
INVERT BOOLEAN ACCUMULATOR	-	Complement the contents of the Boolean accumulator.
STORE BOOLEAN ACCUMULATOR	-	The Boolean accumulator contents are stored in the memory location specified by the address portion of the instruction, leaving the Boolean accumulator contents unchanged.
BOOLEAN SUM (OR) (A + B)	-	Perform the Boolean sum operation with the contents of the Boolean accumulator and the Boolean data word in the memory location specified by the address portion of the instruction, leaving the sum in the Boolean accumulator.
BOOLEAN PRODUCT (AND) (A·B)	-	Perform the Boolean product operation with the Boolean accumulator contents and the Boolean data word in the core memory location specified by the address portion of the instruction, leaving the product in the Boolean accumulator.

An examination of the above list of instructions shows that while some are quite common and may be found in almost any digital computer, others are peculiar to the Mark I and are to a large degree responsible for the high degree of flexibility found in the Mark I. These instructions allow the programmer greater freedom in writing the program, and reduce the requirement for optimization of the program. In fact, ease of programming has been a goal of the logical design of the Mark I, so that elaborate programming techniques such as looping, branching, address arithmetic, and the like will not be required.

4.4 SCALING OF VARIABLES

As stated previously, all constants and variables handled by the Mark I must be scaled to lie within the range from -1 to +1. The situation is analogous to that encountered when scaling quantities for analog computers in the range from -100 volts to +100 volts. A major difference does exist, however, in the choice of a scaling factor. Ordinarily, in an analog computer, the choice of scale factor is completely free, as long as it constrains the quantity concerned to lie in the proper range. In the Mark I, however, the choice of scale factors is limited to integral powers of two in order to minimize time-consuming multiplication operations.

Consider, as an example, some variable X that ranges from -80 to +500. Obviously, if given complete freedom in the choice of scale factor, the number .002 is a good choice, since $.002X_{max} = 1.0$. However, under the restriction imposed above, the choice is limited to the smallest integral power of two which, when divided into X_{max}, results in a number less than or equal to one — that is, we must find the smallest positive integer α for which $X_{max}2^{-\alpha} \leq 1.0$. For the example given, $X_{max} \cdot 2^{-9} < 1.0$, and 2^{-9} (1/512) is the required scale factor.

If the choice of constants were completely free, most resulting numbers would be awkward to accommodate on the Mark I. That is, there would be an extremely large number of time-consuming multiplications and divisions caused by the use of these numbers. On the other hand, recall that a SCALE, or arithmetic shift, is equivalent to multiplying by integral powers of two — positive powers for left scales and negative powers for right scales. The Mark I is capable of shifting as many as five places in one basic machine cycle. Thus, a left scale of three places is equivalent to multiplying by 2^3, and a right scale of three places is the same as multiplying by 2^{-3}. The operation, of course, requires only one instruction word, whereas a multiplication requires five words. The saving in program time and the benefits derived therefrom are apparent.

4.5 PROGRAMMING TECHNIQUES

4.5.1 Preparation and Programming of Data for the Linear Interpolator

Flight simulator computations necessarily employ empirical

data describing the characteristics of particular aircraft. These data generally cannot be expressed analytically and must be generated in a literal manner. In the Mark I, this job is accomplished by the linear function interpolator, the operation of which has already been described in Section 3.

In the reduction and scaling of empirical aircraft data for use in the Mark I, there are two major considerations:

1) The value of any number handled by the Mark I must lie between -1. 0 and +1. 0.

2) All function data utilized by the linear interpolator are assumed to lie within the first quadrant — that is,all values of the function and of the independent variable must be greater than or equal to zero.

The implication of the second consideration is that any function curve defined for negative values of the independent variable and/or having negative function values will require an axis transformation of the type illustrated in Figure 15. A new independent variable, X', must be generated to conform to the axis transformation, and the function data generated from this curve, f(X'), must be operated on by an inverse transformation before being used, to result in a true f(X) curve.

The implication of both considerations taken together is that the entire range of the independent variable must be scaled to lie between zero and +1. 0. Similarly, the entire range of the function data must lie between zero and +1. 0 (see Figure 15).

Once all the function data for a given system have been reduced, scaled, and prepared for programming, it is then necessary to compile a listing of the words to be written on the interpolator bands of the magnetic drum. Each function will be represented by a block of words. Complete blocks are listed one after the other in the program. Figure 16 illustrates such a sequence.

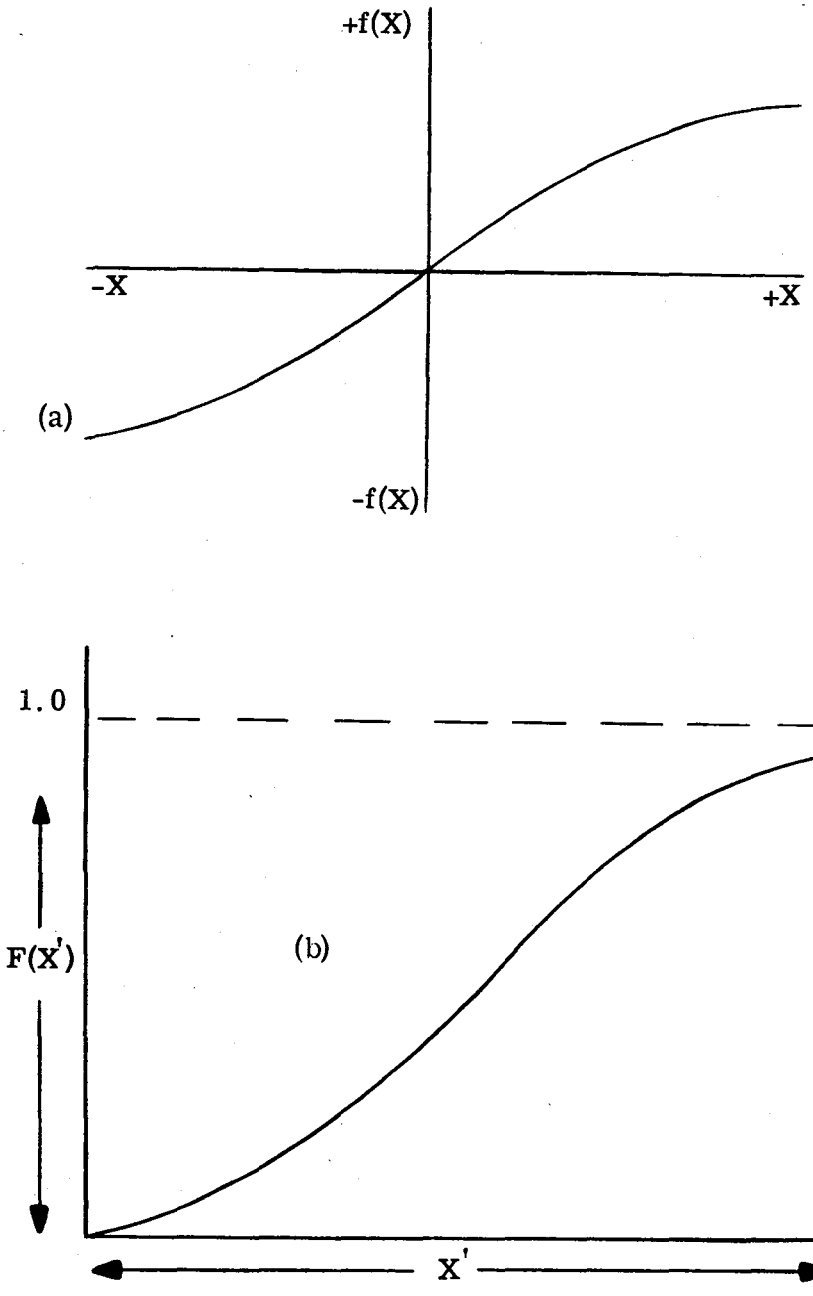


Figure 15 FUNCTION DATA TRANSFORMATION AND SCALING

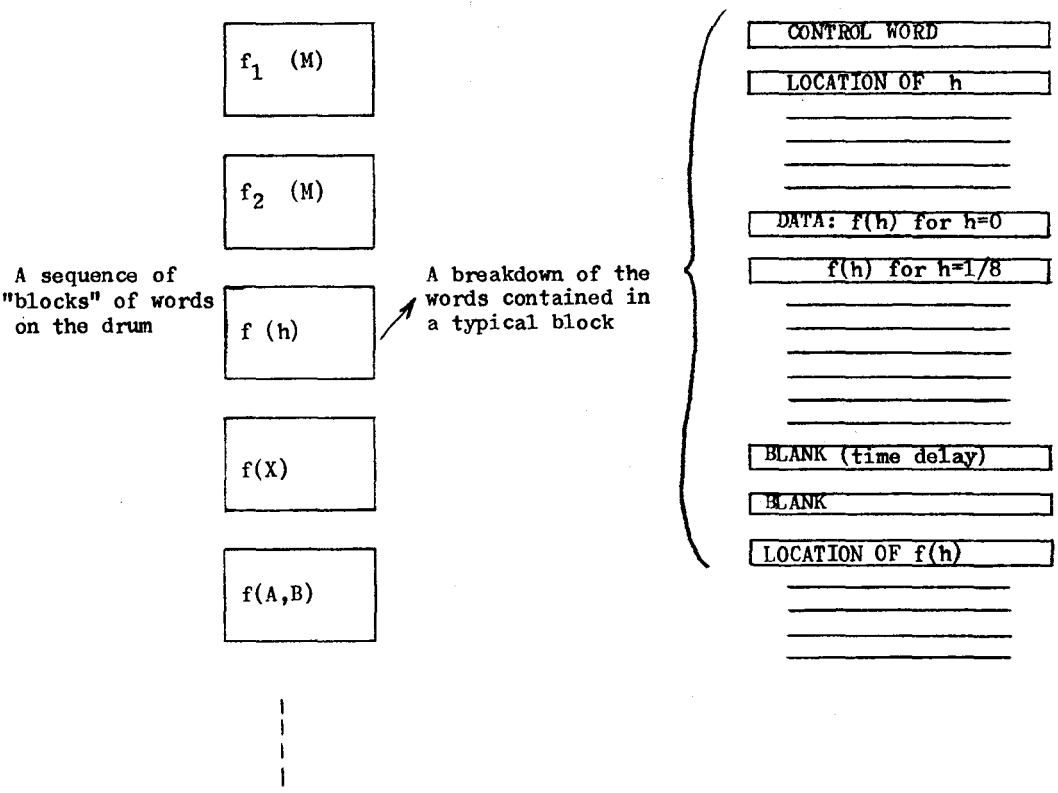


Figure 16 DRUM REPRESENTATION OF FUNCTION DATA

The control word is a coded word that relates information to the interpolator concerning the function being described by the succeeding list of words. A sample control word is shown in Figure 17.

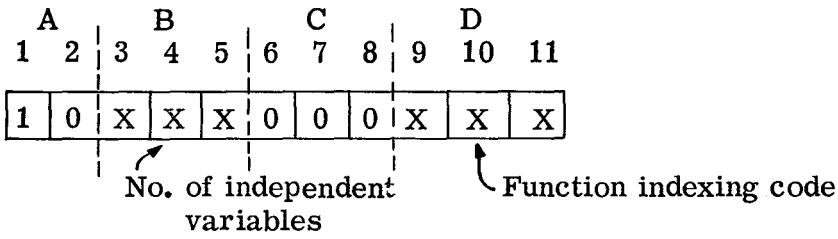


Figure 17 SAMPLE CONTROL WORD

The first bit (most significant bit) of the control word is always one, and the second bit is always zero. Bits 9, 10, and 11 represent a small coded word that tells which variables are to be indexed. The third, fourth, and fifth bits of the control word tell whether the function is of one, two, or three independent variables. Bits 6, 7, and 8 are not used.

The three-bit code that tells the interpolator whether the function is of one, two, or three variables is as follows:

001 - Function of one variable

010 - Function of two variables

011 - Function of three variables

The index code (bits 9, 10, and 11) is as follows:

000 - None of the variables are indexed

001 - Index on X

010 - Index on Y

011 - Index on X and Y

100 - Index on Z

101 - Index on X and Z

110 - Index on Y and Z

111 - Index on X, Y, and Z

If a function is indexed, then it is indexed through four different variables. The core memory addresses of the independent variable and the calculated function, as listed on the drum, must have zeros in the last two digits (least significant digits). A counter in the interpolator will automatically modify these last two digits to create four different addresses in which the four different independent variables can be found and the four different calculated answers stored. These addresses will be:

	ADDRESSES OF X	ADDRESSES OF f(X)
1)	X X X X X X X 0 0	X X X X X X X X 0 0
2)	X X X X X X X 0 1	X X X X X X X X 0 1
3)	X X X X X X X 1 0	X X X X X X X X 1 0
4)	X X X X X X X 1 1	X X X X X X X X 1 1

The initial pair of addresses are automatically used the first time calculation is performed. Four drum revolutions later, when that block of words is read again, the second pair of addresses will be used, and so forth. Note that the control word may be looked upon as being made up of four binary-coded octal digits ABCD (see Figure 17). A is always two, B may be one, two, or three, C is always zero, and D may take on values from zero to seven. Thus, the complete 11-bit control word describing a function of three variables with indexing on the third variable only could be written as the octal number 2304.

The order of information contained in a block of words describing a function is as follows:

1) Control word

2) X address

3) Y address

4) Z address

5) Numerical data

6) Computational time

7) Answer address

This list of seven items may be described as seven "zones" of words. Every word written on the interpolator bands is an 11-bit word, of which the most significant bit is a control bit.

The first word in each zone stored on the drum will have a "1" in its control bit. All the other words will have a "0" in their control bit. In this manner, the beginning of each zone will be marked by the appearance of a "1" in the control bit of the first word of that zone. Zones 3 and/or 4 will be ignored automatically if the control word identifies the function as being a function of only one or two variables, as the case may be. Zone 1 contains only the control word discussed previously. Zones 2, 3, 4, and 7, when used, are all address zones, and each of these zones consists of four or five repetitions of the core memory address concerned.

Zone 6 represents extra time allowed for the interpolator to finish its calculations. This time is accounted for by having two blank words (except for the control bit in the first word) for a single variable function, four blank words for a function of two variables, and six blank words for a function of three variables. Actually, anything at all may be written in these words as long as the control bit convention is observed.

Zone 5 is the zone containing all numerical data. The control bit of the first data word must, of course, contain a "1", while all the others are zero. This ordinate information is stored sequentially — nine successive words specifying a single curve, nine groups of nine words specifying a family of curves (for a function of two variables), and nine blocks or sheets of nine groups of nine words each specifying a function of three variables.

Of the nine words describing a single curve, the first word is f(0), the second is f(1/8), and so forth (see Section 3). For a function of two variables, the first curve, or group of nine words, is associated with Y = 0, the second curve is associated with Y = 1/8, and so forth. Similarly, for a function of three variables, the first sheet listed is associated with Z = 0, the second sheet with Z = 1/8, and so forth.

Sometimes it is not necessary to include the entire numerical data zone. If, for instance, the programmer were certain that Z would always lie between Z = 0 and Z = 1/2, there would be no point in including data sheets for Z = 5/8, 3/4, 7/8, and 1.0. Therefore, after listing all the data words making up the sheets for Z = 0, Z = 1/8, Z = 1/4, Z = 3/8 and Z = 1/2, the data field could be truncated by merely listing the first

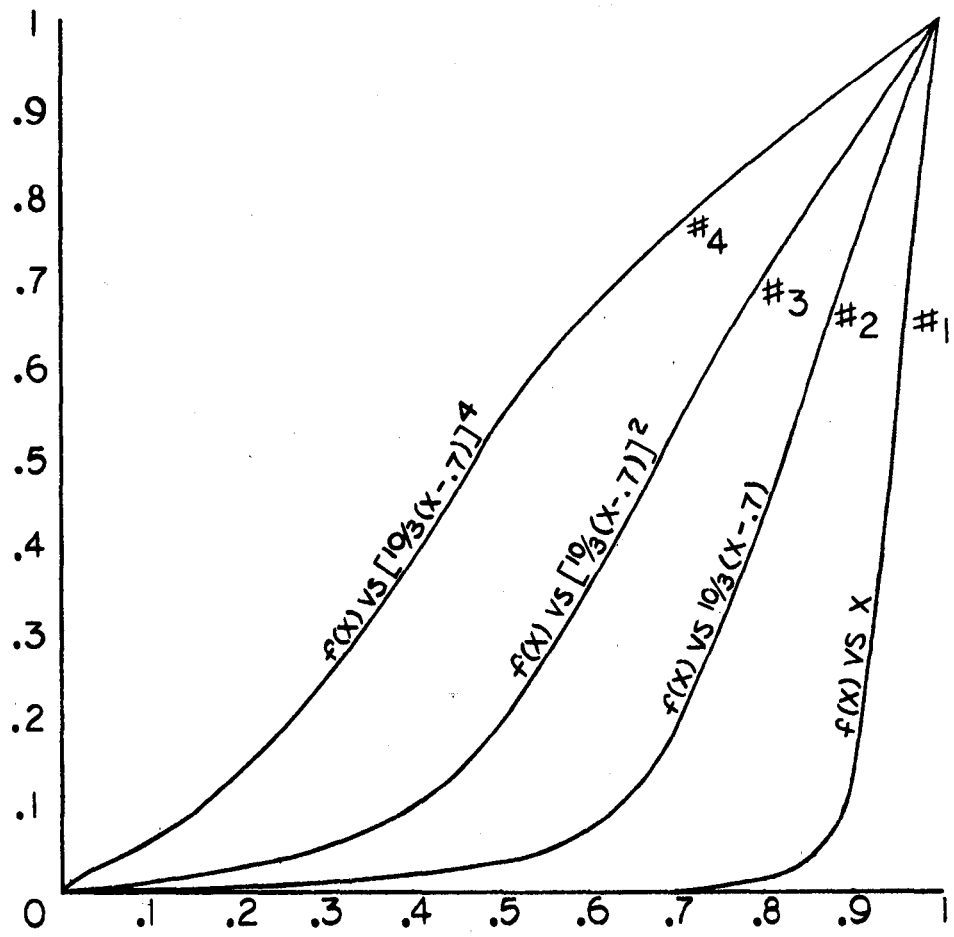


Figure 18 CURVE WARPING TECHNIQUE

word of the next zone (the interpolator will recognize the change by the appearance of a "1" in the control bit of that word).

Similarly, if one is dealing with a function of two variables and the second variable, Y, will always be limited to some value less than one, then it is permissible to reduce the number of curves required to describe that function.

4.5.2 Programming Ill-Behaved Functions for the Linear Interpolator

When a function cannot be closely approximated by straight-line segments between fixed breakpoints, special methods must be used to program the function for the digital interpolator. This will happen when the derivative of the function curve is large in magnitude, or when the curve has points of inflection whose ordinates do not coincide with any of the fixed breakpoints. There are many methods of handling this problem; the method chosen is determined by the nature of the function curve.

Probably the most promising method for use on the Mark I is that of "warping" the curve into a more reasonable shape. This is accomplished by replotting the function data in terms of a new variable that is a function of the original independent variable. That is, instead of plotting f(X) versus X, we can plot f(X) versus X^2, or $\sqrt{X}$, or X^3-1, or any other deliberately chosen function of X that will warp the function data in the desired manner. Consider the four curves shown in Figure 18. Curve No. 1 is the original curve of f(X) versus X. Curve No. 2 represents a linear "stretching" of the independent variable: f(X) versus 10/3 (X-0.7). Curve No. 3 represents the function data plotted versus $[10/3\ (X-0.7)]^2$. Any degree of warping may be obtained by choosing the proper function of X as an independent variable. The function chosen must, of course, be generated as part of the general program and stored for use by the digital interpolator.

Not all ill-behaved functions will lend themselves to warping techniques. We may find for example, that there is no function of the independent variable capable of producing the desiged degree of warping, or, more probably, that such a function exists but is too difficult and time-consuming to derive. Furthermore, even when a suitable function can be derived, we may find that an excessive number of program steps must be utilized in order to generate the new independent variable. In these cases a new approach to the problem must be taken.

Consider the function illustrated in Figure 19 . Since the breakpoints are fixed as shown by the logic of the interpolator, the function constructed by the interpolator will resemble the dashed line shown in Figure 19a. Obviously, f(X) is badly distorted by the removal of the peak in the vicinity of X = 0.41. Thus, in order to con-

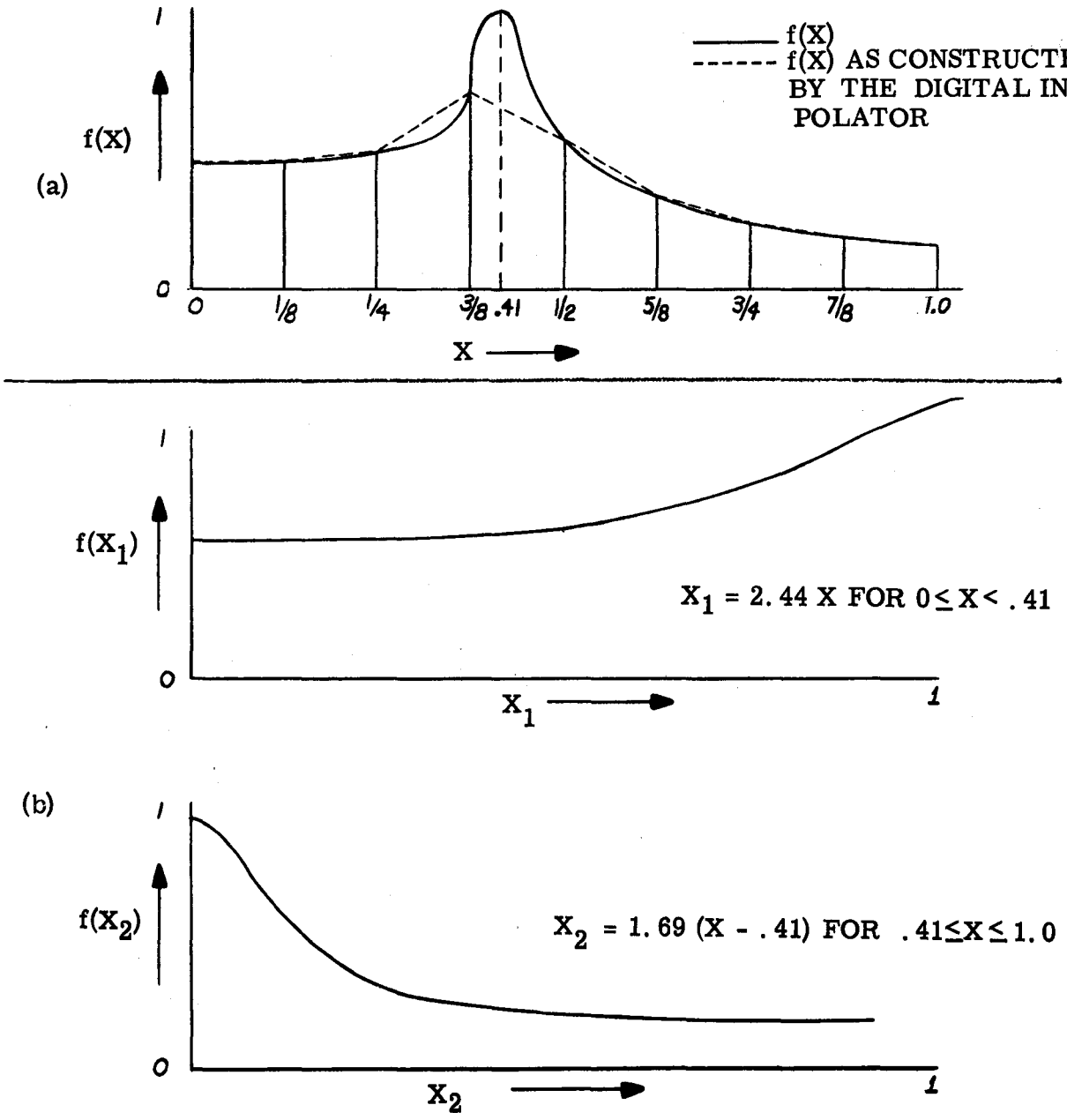


Figure 19 FUNCTION SPLITTING TECHNIQUE

struct f(X) with accuracy, one approach might be to split f(X) into two functions: $f(X_1)$ for $0 \leq X < 0.41$ and $f(X_2)$ for $0.41 \leq X \leq 1.0$ (see Figure 19b). By rescaling X in the region between X=0 and X=0.41, a new quantity X_1 may be defined as:

$$X_1 = \frac{1.0}{0.41} X = 2.44 X \qquad \text{for } 0 \leq X \leq 0.41$$

Similarly, by rescaling X in the region between X = 0.41 and X = 1.0, a new quantity X_2 may be defined:

$$X_2 = \frac{1.0}{1.0 - 0.41} (X - 0.41) = 1.69 (X - 0.41) \quad \text{for } 0.41 \leq X \leq 1.0$$

Two new and entirely separate functions are now constructed: $f(X_1)$ and $f(X_2)$. The function $f(X_1)$ will be an ordinary eight-segment function programmed on the interpolator portion of the drum. It will describe the original f(X) for $0 \leq X \leq 0.41$. Similarly, $f(X_2)$ will be a separate eight-segment function on the drum describing f(X) for $0.41 \leq X \leq 1.0$. This means, of course, that all three independent variables (X, X_1, and X_2) must be assigned storage locations in core memory. Similarly, core locations must be reserved for $f(X_1)$ and $f(X_2)$.

Each time the Mark I calculates a new X, it is necessary to test X to see if it lies in the region of X_1 or in that of X_2. Simultaneously, a Boolean flag must be set and stored in order to later identify which region X lies in. This is necessary because later, when it is desired to use f(X) in some calculation, there must be some simple way to identify the correct functional value for the computer to select — that is, $f(X_1)$ or $f(X_2)$. Having tested X and set the flag, X_1 and X_2 may now be calculated and stored in their assigned locations. It does not matter if, for instance, a value of X_2 is calculated and stored for some value of X lying in the region of X_1. It is true that this X_2 will be an incorrect number and that the corresponding value of $f(X_2)$ will also be incorrect, but it must be remembered that because of the flag previously set, the Mark I will ignore $f(X_2)$ and select $f(X_1)$.

It should be noted that this particular method of splitting an ill-behaved function is not a recommended method because it requires a large amount of program space to implement. However, the basic idea of splitting a function may be extrapolated into several methods or variations, many of which require less program space and are better suited to use on the Mark I.

4. 5. 3 Numerical Integration Methods

The high-speed portion of the computation cycle of the Mark I computer operates at a repetition rate of 20 cycles per second. Table I illustrates the periods and degrees of damping of a number of computer solutions for two different integration formulas. This table has been prepared employing Z-transform techniques and has been calculated for damped sinusoidal solutions, since these are considered to be representative of the types of dynamic response most critical in the simulator flying qualities. For longer-period oscillations than those indicated on the table, the computer solution is invariably vastly superior to the tabulated information and may be considered to be, for practical purposes, perfect. In every instance, it should be understood that the dynamic response of the simulator (at least for a computer employing word length as great as the 24-bit calculation specified for the Mark I) is, even with the degree of deterioration indicated in Table I, vastly superior to that now obtainable in electronic computers which employ electromechanical elements for multiplication or integration.

Columns 1 and 2 of Table I indicate the period and damping characteristics of a number of equation solutions using "pencil and paper" methods. Columns 3 to 6 indicate the results obtained when the computation is performed by a digital computer. Columns 3 and 4 represent the computer solution using parabolic integration, while columns 5 and 6 represent the computer solution using an integration formula commonly called Gurk O33. The relative-amplitude columns represent the percentage relationship between successive peaks of the sinusoidal oscillation. A factor of 100% indicates a neutrally damped system continuing to oscillate at a fixed magnitude, 95% indicates that each successive cycle has a magnitude equal to 95% of that of its immediate predecessor, and so on. It should be understood that these percentages pertain to relative magnitudes per cycle of oscillation and not per cycle of the machine computation. The table has been prepared for a basic cyclic repetition of 20 calculations per second.

It should be observed that the parabolic integration formulas give materially superior accuracy with respect to damping ratio for reasonable periods in comparison with the Gurk O33 integration formula. Furthermore, the errors which are present in the case of parabolic integration result in a solution which is slightly overdamped. The opposite effect is observed for the errors employing Gurk O33, an unfortunate consideration in view of the fact that marginally damped oscillations, such as Dutch roll at high altitude, may become objectionable if subjected to a reduction in natural

TABLE I

INTEGRATION DYNAMIC ERROR FOR DAMPED SINUSOIDS

True Solution		Computer Solution Using Parabolic Integration		Computer Solution Using Gurk O33 Integration	
Period In Seconds	Relative Amplitude (%)	Period In Seconds	Relative Amplitude (%)	Period In Seconds	Relative Amplitude (%)
31.4	10	31.4	10.00	31.5	10.02
31.4	50	31.4	50.00	31.5	50.07
31.4	90	31.4	90.00	31.5	90.13
31.4	100	31.4	100.00	31.5	100.14
3.14	10	3.14	9.98	3.14	10.17
3.14	50	3.14	49.88	3.14	50.11
3.14	90	3.14	89.79	3.14	91.96
3.14	100	3.14	99.73	3.14	102.2
1.57	10	1.58	9.81	1.56	10.11
1.57	50	1.57	49.01	1.56	51.49
1.57	90	1.57	88.32	1.56	93.41
1.57	100	1.57	98.18	1.56	103.94
0.628	10	0.662	6.50	0.617	12.41
0.628	50	0.623	36.78	0.600	47.35
0.628	90	0.614	69.45	0.592	91.84
0.628	100	0.612	78.38	0.592	103.42

damping owing to machine error. The use of Gurk O33 does, however, permit the attainment of reasonably satisfactory dynamic solutions at oscillatory frequencies higher than those which can be handled by parabolic integration (unless overdamped solutions can be tolerated).

It should be recognized that the Mark I computer is not wired or designed to accommodate any particular integration formula; the particular choice of formula is determined by programming. Parabolic integration is moderately more economical of computer time than the more complex Gurk O33 formula. However, the actual percentage of increase in computation time required through the use of complex integration formulas is relatively small, since an enormous amount of additional computation unrelated to integration must be done in any case.

4.5.4 Extra-Precision Arithmetic

In a digital computer, integration is accomplished by calculating an incremental change in the quantity concerned and adding that increment to the old value of the quantity. In the equation $X = \int \dot{X}\, dt$, if $\dot{X}$ is extremely large in magnitude, then the LSB (least significant bit) of X is correspondingly large. Since the magnitude of this LSB determines the smallest rate of change of X that can be integrated with accuracy, it may be necessary in some cases to magnify the LSB to obtain a smaller increment. This situation is comparable to that encountered in analog integration, where a wire-wound potentiometer provides a rate signal to an integrator, and the ΔV between the centertap of the potentiometer and the next wire over determines the smallest rate of change that can be fed to the integrator. In every analog system, a smaller ΔV can be obtained by using a vernier potentiometer to interpolate between two adjacent wires of the main potentiometer. In the Mark I, the programmer can accomplish this by means of "extra-precision arithmetic."

To illustrate this technique, let's consider the hypothetical case of an aircraft flying almost due north (Y axis). Assume there is a very small drift velocity to the east (X axis). Let ±1 in the Mark I correspond to ± (half-way around the world at the equator) ≅ ±12,500 miles. With a 23-bit resolution in the Mark I, the pilot can know his X position anywhere in the world within 8 feet — certainly an excellent resolution for position. However, X is calculated by integrating $\dot{X}$, and, since the LSB

of X in the Mark I corresponds to approximately 8 feet, and X is integrated at the rate of 20 calculations per second, then the smallest drift velocity recognized is 8 feet x 20/second = 100 feet/second. A drift velocity of this magnitude can cause appreciable error in heading and position and would quickly be noticed. Therefore, in order to calculate X accurately for small rates, there must be an increase in the resolution of X.

A suitable increase in resolution can be achieved by treating X as the sum of two numbers:

$$X = X_{Coarse} + X_{Fine}$$

X_C will have the same scale factor as X ordinarily would, and X_C really is X, all by itself. However, instead of integrating X to generate X, $\dot{X}$ will be integrated to generate X_F, where X_F is $2^{-10}X_C$. This represents a ten-bit increase in resolution. Each time X_F is calculated by integrating $\dot{X}$, it is tested to see whether it is large enough to add a significant digit to X_C. If it is large enough, then X_F is scaled ten places to the right and added to X_C. Obviously, whatever amount is added to X_C must also be subtracted from X_F.

Extra-precision arithmetic can, of course, be utilized for any computation in which increased resolution must be achieved; however, in view of the 23-bit inherent resolution of the Mark I computer, it is highly unlikely that extra-precision methods will be required except to deal with the type of problem we have illustrated.

4.5.5 Simulation of Accessory Systems

4.5.5.1 General Approach

One of the unique features of the Mark I computer is its integral Boolean capability. This capability has been incorporated specifically to permit economical simulation of the complex switching systems found in current aircraft designs. It represents an economical approach, eliminating the large quantities of wired relay systems commonly required for this type of simulation. Furthermore, aircraft switching systems are particularly vulnerable to modification, and the Boolean capability of the Mark I is a flexible capability that can easily be reprogrammed to accommodate modification.

The fundamentals of Boolean logic are illustrated in Figure 20. The Boolean (two-valued) function E is defined by the two

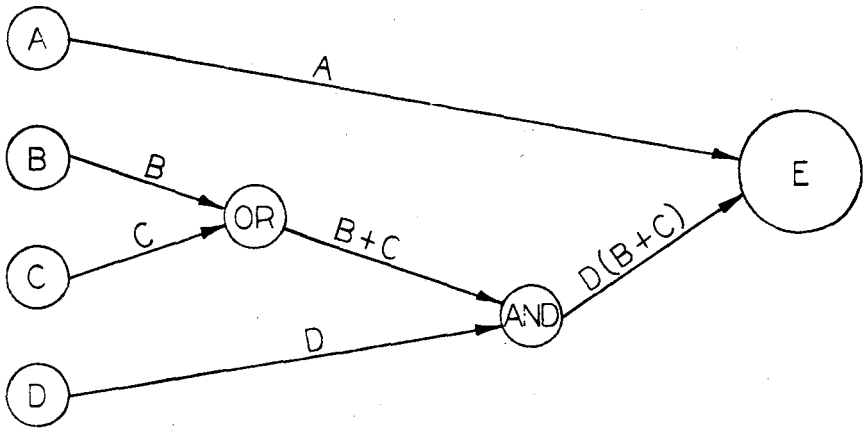


Figure 20 FUNDAMENTALS OF BOOLEAN LOGIC P. 65

lines directed into it. One of these is the Boolean function A; the other is the indicated Boolean function of B, C, and D. In Boolean notation, E=A+D (B+C). If the elements of the diagram represented the Boolean elements (switches, lights, circuit breakers, relays) of a particular aircraft subsystem, then that subsystem could be simulated by programming the Mark I to solve the Boolean equation for E. In practice, all aircraft systems are reduced to a simplified Boolean flow diagram containing only those inputs, outputs, and variables which need to be stored.

4.5.5.2 Typical Example

The simplicity of simulating accessory system switching operations in the Mark I computer may best be seen by the following example. Figure 21 represents a typical aircraft electrical subsystem. Load sensing and balancing components have been omitted for clarity. In this example, the load ammeter will read the current from either generator, depending on the position of the ammeter switching relay. In addition, the ammeter will read one-half of the total load when both "Bus Tie" relays are energized. The load ammeter must reflect actual bus loads that are computed from the corresponding circuit breaker and switch positions. For simulation purposes, two distinct computational steps are required. First, Boolean switching logic must be performed to determine the number of loads that should be reflected by the ammeter; second, the load current to be indicated by the load ammeter must be computed.

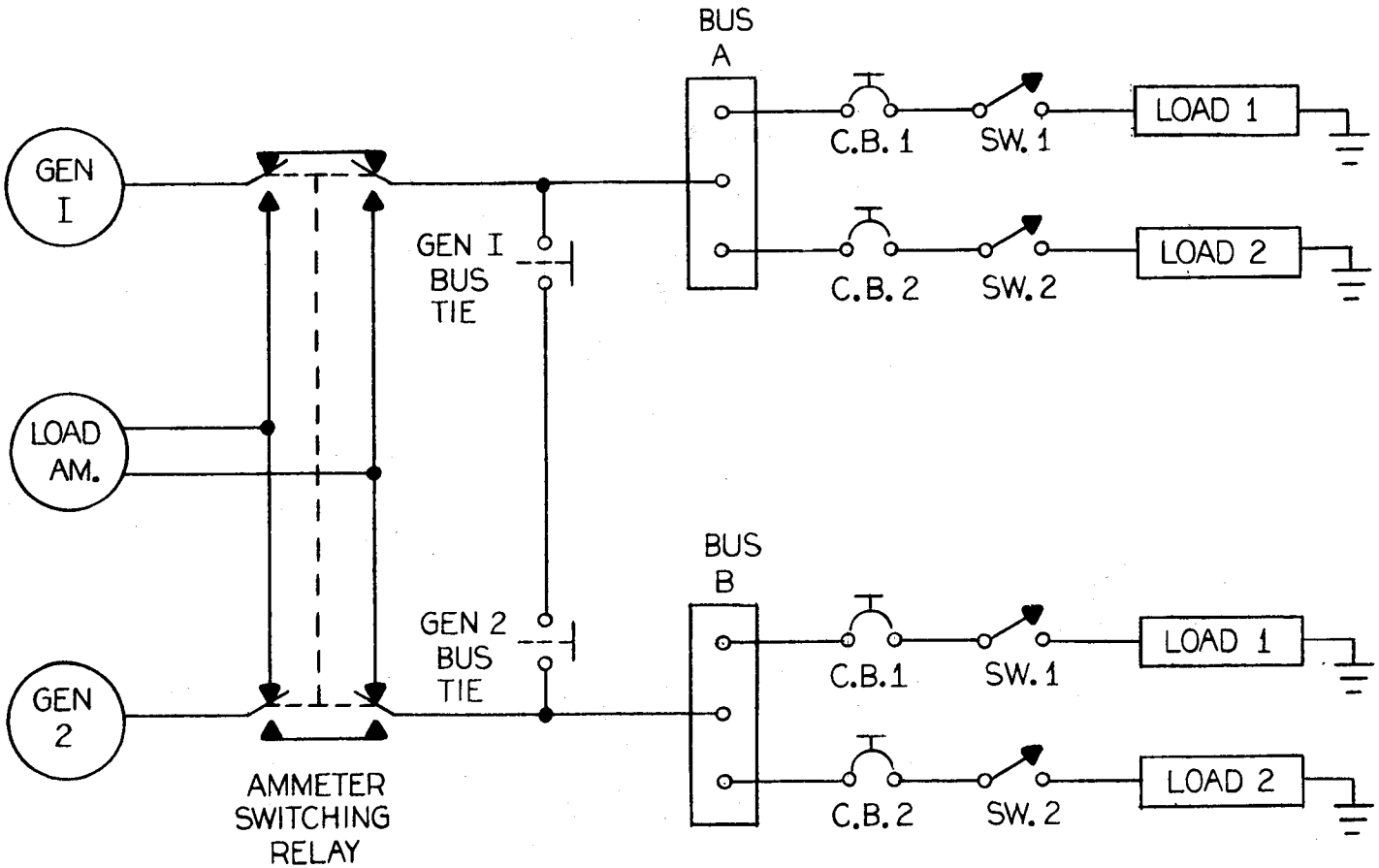


Figure 21 TYPICAL ELECTRICAL SUBSYSTEM

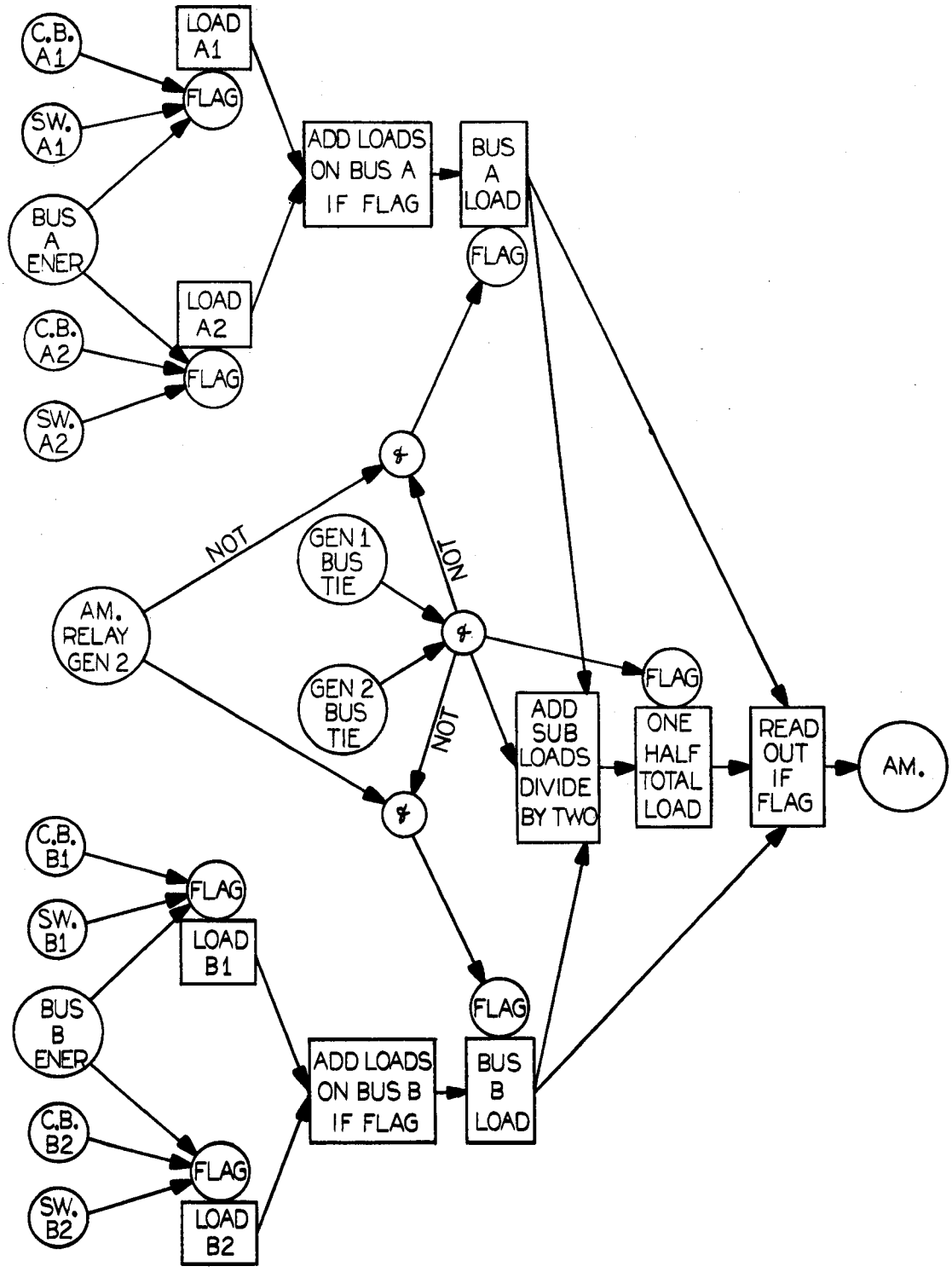


Figure 22 LOGICAL AND ARITHMETIC FLOW DIAGRAM FOR SYSTEM SHOWN IN FIGURE 21

Figure 22 is a Boolean flow diagram based on the system shown in Figure 21. The Boolean functions are designated by circles and the arithmetic operations by rectangles. Arithmetic operations controlled by Boolean functions are designated as FLAG operations. This is shown by a circle labeled FLAG adjacent to the rectangle that represents the arithmetic operation. In practice there are more than two individual loads on a bus. From Figure 22, the following equations can be written for the Bus A load:

$$\text{FLAG } A_1 \ (F_{A1}) = SW_1 \ CB_1 \text{ (Boolean)}$$

$$\text{FLAG } A_2 \ (F_{A_2}) = SW_2 \ CB_2 \text{ (Boolean)}$$

$$\text{BUS A Load} = \text{Load } A_1 + \text{Load } A_2 \text{ (Arithmetic)}$$

where: Load A_1, Load A_2 = O for F_{A1}, $F_{A_2} = 0$

Load A_1, Load A_2 = K for F_{A1}, $F_{A_2} = 1$

Individual loads on Bus A and Bus B will be arithmetic numbers stored in the drum memory. The program for these loads will be repetitive. The following program describes the Bus A load computation:

	Instruction	Data
FLAG A_1	LOAD B (load Boolean accumulator)	Bus A (energized)
	BOOLEAN PRODUCT (multiply by)	Switch A_1 ("ON")
	BOOLEAN PRODUCT	Circuit Breaker A_1 ("ON")
	COND SK (conditionally skip the next instruction if above result is not true)	
	LOAD CONSTANT (main accumulator)	Bus A_1 Load

FLAG A_2	LOAD B	Bus A (energized)
	BOOLEAN PRODUCT	SW A_2
	BOOLEAN PRODUCT	CB A_2
	COND SK (continually skip the next two instructions if above result is not true)	
	LOAD CONSTANT	Bus A_2 Load
	ADD	Bus A_1 Load + Bus A_2 Load

In a similar manner, the Bus B load (and any subsequent loads necessary) would be computed and added to the Bus A load to provide the required loadmeter readout.

4.5.6 Elapsed Time Computation

Any desired elapsed time can be simulated in the Mark I by successively subtracting a small decrement from a stored constant during each cycle of operation until the original number is reduced to zero. The total time, t, is determined by dividing the stored number (M) by the decrement (N) and multiplying by the time required for one cycle: $t = M/N\ T_c$.

A diagram of the required operation is shown in Figure 23.

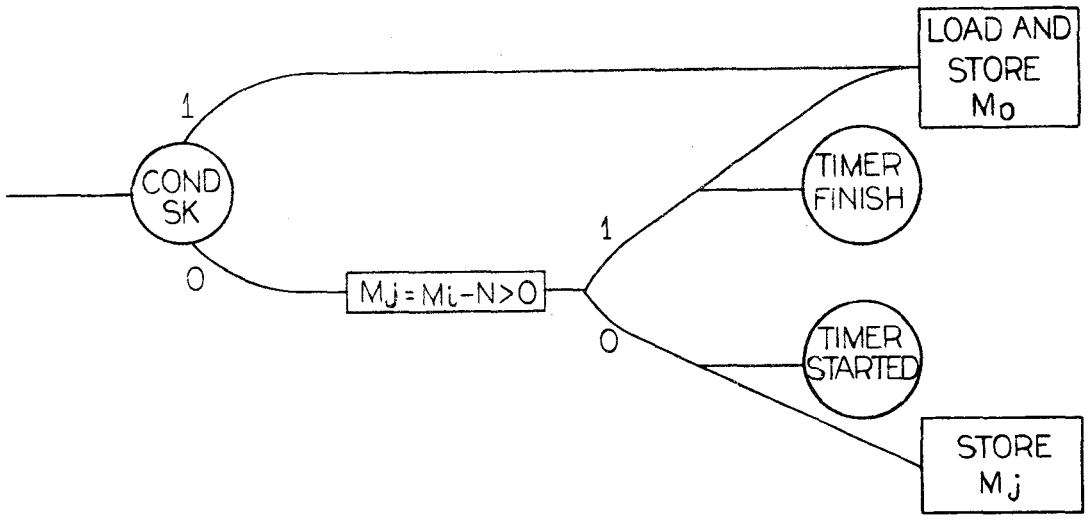


Figure 23 ELAPSED TIME OPERATION

Assuming N is a constant already stored, perhaps for some other purpose, the program would be:

Instruction	Data
COND SK	
LOAD	M_i
SUB	N
STORE	M_j
FL NEG	
B INV	
COND SK	
LD CONS	M
STORE	M

Referring to the logic diagram in Figure 23, it can be seen that if the timer (A) has not been started (A = 0), the time instructions are skipped. When the timer has been started (A = 1), then the repetitive subtraction process is begun and continues until $M_j < 0$. At this point, a flag is set indicating the specified elapsed time has passed, and the instructions that manifest the end of the elapsed time are executed.

4.5.7 Motor Drive Computation

There are many applications involving a motor-driven unit, usually found at one extreme or the other of the allowable travel. Typical of these are motor-driven valves, flaps, trim and wheels. Figure 24 shows a typical unit of this type and diagrams of the computer operations to accomplish it.

The stored value (M) represents the position of the simulated device at all times, with K = .9998 at the upper limit and zero at the lower limit. The upper and lower limits are stored Boolean functions, and as such, can be used in Boolean operations. The position (M) is a stored arithmetic number and can be used in any arithmetic operations. The constant (N) which is added or subtracted each machine cycle can be two different values, one for increase and the other for decrease.

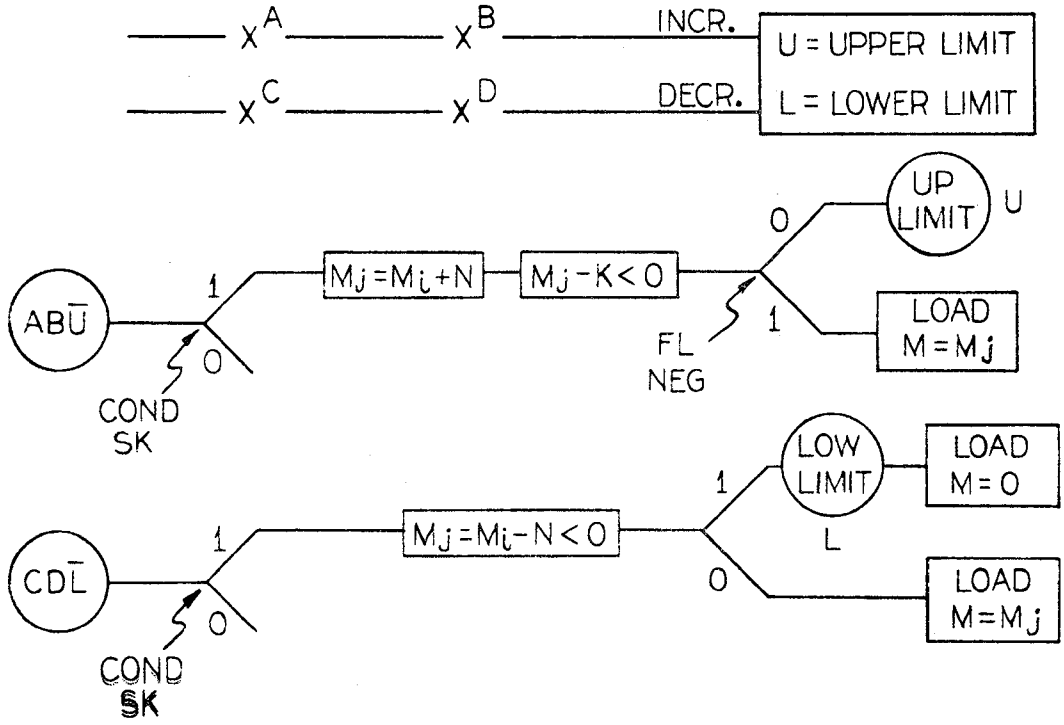


Figure 24 MOTOR DRIVE OPERATION

The programs for computer routines indicated in Figure 24 are as follows:

Instruction	Data	Contents of Boolean Accumulator
B LOAD	A	A
B MUL	B	AB
B INV		$\overline{AB}$
B SUM		$U+\overline{AB}$
COND SK		
LOAD	M	
ADD	N	M+N

Instruction	Data	Contents of Boolean Accumulator
STORE	M	
SUB	K	M+N-K
FL NEG		
B INV		
B STORE	U	
B LOAD	C	
B MUL	D	CD
B INV		$\overline{CD}$
S SUM	L	CD+L
COND SK		
LOAD	M	
SUB	N	M-N
STORE	M	
FL NEG		
B STORE	L	
COND SK		
ZER SL		
STORE	M	

4.6 PROGRAM LOADING

All instructions, numerical data for the generation of arbitrary functions, and descriptive data for radio navigation facilities are stored on the drum memory of the Mark I computer. For clerical convenience, the useful storage area of the drum is divided into 16 bands, each containing four 1024-word sectors. The four bands containing function generation information employ 11-bit words (44 tracks), the 11 bands containing general program instructions employ 16-bit words (176 tracks), and the single radio-navigation facility band employs 20-bit words (20 tracks). All three data bands are loaded from punched paper tape, employing one standardized loading procedure. Any of the 64 sectors may be loaded independently, employing a relatively short segment of punched tape, or the entire program (or any desired sequential fragments thereof) may be automatically loaded in sequence, employing a single, full-length tape.

The drum loading system is an integral part of the Mark I computer. The loading operation is controlled through manipulation of console switches rather than through program instructions. Accordingly, there are no instructions related to program or memory loading in the Mark I instruction repertory, nor is it necessary to employ loading routines or special computer programs to accomplish computer loading. The tape loading method is felt to offer certain advantages over other possible loading techniques; however, if program loading directly from punched cards is desired, this

can be readily incorporated into the Mark I with no modification of the basic computer. The various items of equipment utilized in the drum loading sequence and their characteristics are as follows:

1) Photoelectric punched tape reader — 500 characters/second

2) Tape handler — 500 characters/second forward speed; 100 inches/second rewind speed

3) Input register

4) Main core memory (used as loading buffer)

5) Core memory address counter

6) Drum memory address counter

The tape feed controls are extremely simple; no computer knowledge or sophisticated control manipulations are required to accomplish tape loading. Furthermore, the starting and stopping of normal computer operation, preceding or following tape loading, is accomplished by simple switch manipulation; it is not necessary to set up complex console switches or to introduce instructions through manual console entry to initiate simulator operation after tape loading. In view of the simplicity of the tape loading control system, the fact that computer operation can be initiated without recourse to complicated control manipulations, and the fact that a single 1024-word sector of the drum can be loaded in a few seconds time, it is believed that Mark I program alterations will require considerably less effort than the experimental modification of analog simulators.

The data words are read from eight-hole punched tape. General program instruction words occupy two lines each on the tape, linear interpolator words occupy two lines, radio-aids data words occupy three lines, and core memory load words occupy four lines. Core memory load words will seldom be used in program loading of the Mark I; however, they have been included in order to reflect the maximum time that is required to completely load the drum memory and the core memory. As each word is read from the punched tape, it is compiled in a 24-bit input register. After each word is compiled, it is transferred for storage to one-half of the main core memory (1024 words), which is used as a loading buffer in this process. An address counter is advanced each time a data word is stored in the core memory to provide the memory address input for the following word. When a block (sector) of 1024 data words has been loaded into the core memory, the tape reader is stopped and this block of data is written on the program memory drum in the proper block address location. The block address at which the first block of information on input punched tape is to be written is set manually into control switches and this address is automatically increased by one after each block of data is loaded.

After the process of writing each block of data onto the drum is complete, a bit-by-bit comparison is made between the recorded drum information and the data contained in the core memory. If there are any discrepancies, the loading process will be halted pending manual instruction to attempt reloading. For each block, the writing and comparison process requires two drum revolutions — that is, 0.05 seconds. Manual controls are provided to permit complete rerunning of the punched tape or a duplicate tape in a verification mode to double-check the entire loading process if desired.

The tape length required to fully load the drum memory and the main core memory is as follows:

Type of Data	No. of Bands	No. of Characters per Word	Characters	Tape Length (ft)
General Program	11	2	89,980	750
Interpolator	4	2	32,720	273
Radio Aids	1	3	12,288	103
Core Memory Load	2048 words	4	8,192	68.5
			143,180	1194.5
		Totals (Maximum)		

The time estimate given below is based on the use of two reels of 3.5 mil Mylar tape. It should be noted that no time has been included for the changing of reels (probably less than one minute) or for rewinding the second reel (this can be performed after the computer is in operation). The loading time estimate is as follows:

OPERATION	TIME REQUIRED
Load Program Instruction Data	180 sec. (Reel 1)
Rewind Tape	90 sec.
Load Interpolator, Radio Aids, and Core Memory Data	106 sec. (Reel 2)
Total	376 sec. or 6.25 min.

After the complete simulation program has been loaded into the Mark I, any modification of this program due to aircraft changes or radio aids facility changes will normally require reloading only a small segment of the total simulation program. The smallest segment of the program that can be loaded into the computer is one block, or sector, of 1024 words. Neglecting the time required to insert the punched tape in the drum loader and to rewind the tape after the loading has been completed, which does not represent simulator downtime (that is, can be accomplished while the simulator is in operation), the time required to load one sector of either the general program or the linear interpolator program is approximately four seconds. For the radio-aids data band, the time required to load one sector is approximately six seconds. It may be noted, therefore, that loading a completely new set of radio facilities into the Mark I computer requires only 24 seconds of simulator downtime.

4.7 PROGRAM MODIFICATION

4.7.1 General-Purpose Instructions

The general-purpose program will invariably consist of a very large number of relatively simple program fragments, each fragment accomplishing a specific computational task (for example, an individual fragment might be the computation of indicated airspeed or the calculation of direction cosines from quaternions). In the basic program these individual fragments will be arranged in sequence and assembled in program sectors of 1024 words each. It should be recognized that the entire 1024-word space will not, in general, be employed unless the computer is being utilized perilously close to 100% capacity. Furthermore, in view of the fact that persons unfamiliar with numerical real-time calculations may understandably be led to attach undue significance to the sequence in which the individual calculations are performed, it should be recognized that the actual sequence of calculations is of very little importance. Except for obvious precautions such as resisting the temptation to program the extraction of the square root of the sum of $X^2 + Y^2$ before the quantities X and Y are squared and summed (the actual sequential position of the individual program fragments within a complete program is almost entirely meaningless.) Furthermore, the address location of the individual instructions in the sequential memory of the Mark I is immaterial, and is recorded only for clerical purposes to permit identification of the location of the individual instructions in the event that program alteration is considered desirable.

The procedure for modifying the general-purpose instructions begins with writing a new program to replace the original and undesired program steps. This will, in general, entail replacement of a group of instructions with a new group which may be either larger or smaller in

total number than their predecessor group. These new instructions are then keypunched in any desired groupings from one through eight per card. The instructions to be eliminated from the original program are then identified in the individual card decks from which the original program tape was prepared. The obsolete instruction cards are removed and the new cards inserted in their place. An entirely new and unrelated group of instructions might be placed at the end of the program group or might be inserted between any program fragments in the deck, although it is unwise to break an individual computation group without scanning the original program to discern the probable consequences (for example, it is undesirable to split a program at a point immediately prior to storage of the answer, since the computed result would be abandoned in the accumulator immediately prior to storage and would never be available for subsequent computations or output use).

The corrected deck is then fed through the tape preparation unit to generate a new 1024-word tape fragment which may be individually loaded for test purposes into its corresponding sector of the drum. After testing the modified program, the individual tape sector may be inserted, through splicing, into a complete 64-sector master program tape. If desired, the new spliced master program tape can, at some convenient time, be duplicated as a continuous unspliced tape. This duplication operation can be accomplished without interference with the operation of the flight simulator since the tape preparation unit is completely independent of the simulator.

If desired, after preparing a new 1024-word program fragment, the tape may be fed through the Hewlett-Packard printer and associated circuitry to obtain a printed listing of the new program steps for future reference. Alternatively, the revised IBM deck from which the new tape was prepared can be listed on a standard IBM tabulating printer.

4.7.2 Interpolator Data

Simple numerical alterations to existing interpolator data can be accomplished through removal of the card or cards containing obsolete numerical data and replacement of these cards by cards containing the updated information, keypunched in decimal form. This new deck is then converted into a 1024-word replacement tape and introduced into the computer or into the master tape following a procedure similar to that employed for the general-purpose instructions. In the event that it is necessary to make a major alteration of the interpolator data (as, for example, the substitution of a function of two variables for a function of a single variable), it is readily apparent that insufficient space may be available in the original loading sequence for the new interpolator curve information.

Probably the most straightforward method of handling this problem is to introduce the new two-variable function data at the end of the interpolation sequence. As in the case of the general-purpose programs, space will be available following the interpolation sequence unless the computer is being utilized to maximum capacity. Information keypunched for the interpolator consists of addresses of the X, Y, and Z variables (depending upon whether the function is a function of one, two, or three variables), numerical data for the breakpoints of the curves, and the address of the memory location into which the answer is to be inserted. This information, suitably keypunched, is simply added to the last IBM card deck and a new tape sector punched.

In the example given herein, the original function of one variable remains in the program and is still calculated, since we have not replaced it in the procedure described. Unless the space occupied by the original erroneous obsolete function is needed for generation of another function, the most convenient means of obliterating the original calculation would be to alter the address into which the obsolete single-variable function interpolated answer is to be stored. Accordingly, the card containing the answer address for the obsolete function should be removed and replaced with instructions to store this computation in an unused core memory address. With this procedure the original interpolation is performed, but is stored in a memory location from which it is never used. Here again it should be stressed that the sequence in which the interpolation calculations are performed is completely immaterial. Aside from the proviso that the individual groups of numerical words specifying the ordinates of the curves comprising a single function of one variable (81 words for a function of two variables, and 729 words for a function of three variables, unless the data field for the last variable is truncated as described in Section 4.5) be stored in continuous sequence without break, the sequence of word assignments in the interpolator is entirely at the discretion of the programmer.

4.7.3 Introduction of New Radio Information

New radio information is introduced in a manner exactly analogous to the introduction of general-purpose program instructions, with the following exceptions:

1) The data must be composed of pairs of pseudo instruction words, as discussed in Section 5.6.

2) The various types of radio facilities are grouped in families with respect to their drum storage location.

There are groups of data words for 32 middle markers, 32 outer markers, 32 fan or Z markers, 128 VHF navigation facilities, and 128 low-frequency navigation facilities. The position of an individual radio facility within the group is immaterial — that is, there is no significance to the position of 128 VHF navigation facilities with respect to one another. However, data for a VHF facility cannot be placed in the drum area belonging to a low-frequency facility because the computer will then process the attendant data as a low-frequency facility, with resultant computational error. Since the various categories of navigational facilities are grouped in continuous blocks rather than intermixed, it is not anticipated that any difficulty will be encountered in meeting this programming requirement.

The individual data words within the block of information pertaining to an individual station (such information as latitude, longitude, airport elevation, call letters, frequency) are assigned stereotyped locations within a block and cannot be indiscriminately intermixed without machine misinterpretation of the resultant data. Since reassignment of navigational facilities will, in general, consist of simple substitution of one facility for another, the word-by-word replacement of data in the appropriate cells does not appear to constitute a serious problem.

Accordingly, the pseudo words generated according to the instruction book are simply keypunched as general-purpose instructions, the new cards replace the old keypunched pseudo words pertaining to the obsolete facility, and the deck is translated into a new 1024-word tape sector. The procedure for loading this modified tape is identical to that employed for all tape loading throughout the computer.

It should be recognized, then, that minor Mark I updating projects, such as the introduction of new radio information, can easily be accomplished in the field by essentially the same personnel who now operate and maintain conventional analog flight simulators. Indeed, this has been one of the primary design objectives of the Mark I program, and it is one of the special features that make the Mark I an ideal computer for flight simulation.

5. PROVISIONS FOR EASE OF OPERATION AND MAINTENANCE

5.1 GENERAL CONSIDERATIONS

In an analog simulator, failures of individual components are clearly identified with single elements of the computation. These elements bear identification labels which are meaningful to a person familiar with the mechanics and the functioning of the equipment that is simulated. The individual elements in the Mark I are singularly associated with individual steps in arithmetic operations or the transfer of numbers from one portion of the computer to another. Just as in the case of an analog simulator, the defective element can be localized through observation of the nature of the operational fault. However, it is readily apparent in the case of the digital simulator that the malfunctions have little or no relation to individual parameters or instrumentation characteristics, since the components of the digital computer are time-shared throughout all calculations. For this reason, the operation of the instruments and displays affords little, if any, insight into the possible identity of a defective components in the digital system.

This might be a serious disadvantage for digital simulation if it were not for the fact that the internal program of the digital computer (and, accordingly, the operation of the individual elements) can be readily altered to facilitate diagnostic operations. A corresponding capability in an analog computer would be almost intolerably complex, since it would be necessary to physically transfer thousands of connections, alter resistance values, reset servo positions, to accomplish diagnostic checks automatically. In the digital computer, such alteration in basic machine operation can be accomplished without the incorporation of a single additional component in the design of the computer.

Diagnostic routines function by "exercising" the computer, employing standardized test programs. It is apparent that an unlimited variety of ingenious programs can be devised to operate the computer in such a manner as to facilitate identification of a single defective element. Because of the extremely high computational capability of the computer, it is also common practice to process the results of the individual test operations in the diagnostic routines to simplify display of the results. As an extremely oversimplified example, if the machine were being tested to determine whether or not it is capable of dividing 3 by 5, the correct answer (0.6) could be stored in the diagnostic routine and subtracted from the computed answer, with the result that a correct solution would produce an output of zero. Since an answer of zero would clearly be easier to interpret (when one of thousands of such results obtained rapidly in sequence), the facility for arithmetically processing the test answers is clearly advantageous.

The Mark I computer has a relatively small number of inputs and outputs compared to a complete analog system. Furthermore, these are arranged in such a manner that removal of the connectors can be accomplished with little effort. Accordingly, it is also possible to disconnect the normal input-output information (the majority of which is in analog form) and close the input-output circuitry through adapter plugs. If this is done, a diagnostic routine can also cross-compare the various inputs and outputs to check the operation of the analog-to-digital and digital-to-analog conversion equipment.

In addition, the Mark I will contain provisions for component test by virtue of a static card tester, a dynamic computer test by virtue of individual rack-mounted test point panels, and a static test by virtue of controls provided to permit operation of the computer one instruction at a time. Individual switches will be provided to enable the operator to stop the computer on a specific single instruction word and to temporarily alter this instruction word without affecting the permanently stored program. Indicator lights will be provided to permit readout of the instruction location and the instruction word contained therein. In addition, indicator lights will be included to permit observation of the content of important registers in the computer for specific use during this one-step operation of the Mark I computer.

Many of the components of the Mark I computer are interchangeable. This should materially facilitate troubleshooting, since large numbers of elements can be interchanged to simplify failure analysis. Although not directly associated with the Mark I computer itself, the servos employed to operate instruments in a simulator employing the Mark I computer have a much higher degree of standardization than corresponding servos in an analog computer, since the servos are employed only for the relatively straightforward and simple task of rotating synchro transmitters or similar data transmission devices. Because of the high order of standardization among the simple instrument drive servos, the opportunity to interchange duplicate servos to facilitate maintenance of the instrument drive portion of this system is considerably greater in the digital system than in a corresponding analog simulator.

5.2 COMPUTER CONTROL PANELS

The Mark I computer will contain numerous control and display panels whose functional design is predicated on ease of computer operation and maintenance. Three of these panels — the computer operator's panel, the flight computer panel, and the interpolator/radio aids

computer panel — will be externally available to the computer operator for monitoring and control of all major functions of the computer operation.

The computer operator panel (Figure 25) will contain the major computer operating controls and the computer operating mode controls. Computer operating modes will consist of:

1) Normal: Under clock control.

2) Single-Shot Repeat: Under manual control. Repeat one addressed instruction for each operation of control.

3) Single-Shot Advance: Under manual control. Execute addressed instruction, and increase address in auxiliary counters by one for each operation of control.

4) Instruction Replace: Manual controls provided for insertion of content and address of a single instruction which will replace drum instruction at addressed location, under manual control, in either normal or single-shot modes.

5) Overflow Check: Under manual control. The history of program overflows between successive STORE instructions is placed in the 24th bit location of the data memory.

6) Auxiliary Process Execution Failure: Under manual control. For either linear interpolator or radio aids process, the occurrence of an execution failure will cause computer to stop, with drum address of failure point contained in auxiliary counters, and core address of affected item in appropriate register.

These special operating modes have been provided for convenience in checking machine malfunctions and will also facilitate program debugging.

The single-shot operation, under manual control, is provided with switches for the insertion of an address at which the manual process is to start, and provides for repeats of the addressed instruction or automatic advance. These switches may be used in an additional control mode which will permit any one instruction to be replaced during either normal or single-shot operation by the use of an instruction set up in these switches. Also provided is an overflow-check process, which will permit a programmer to use the 24th bit of the core memory words to retain information regarding any overflows that have occurred during the processing of a program. Provision to print out this information, as well as the complete contents of the core memory and main arithmetic accumulator contents, is provided. An execution-failure test for each of the auxiliary processes

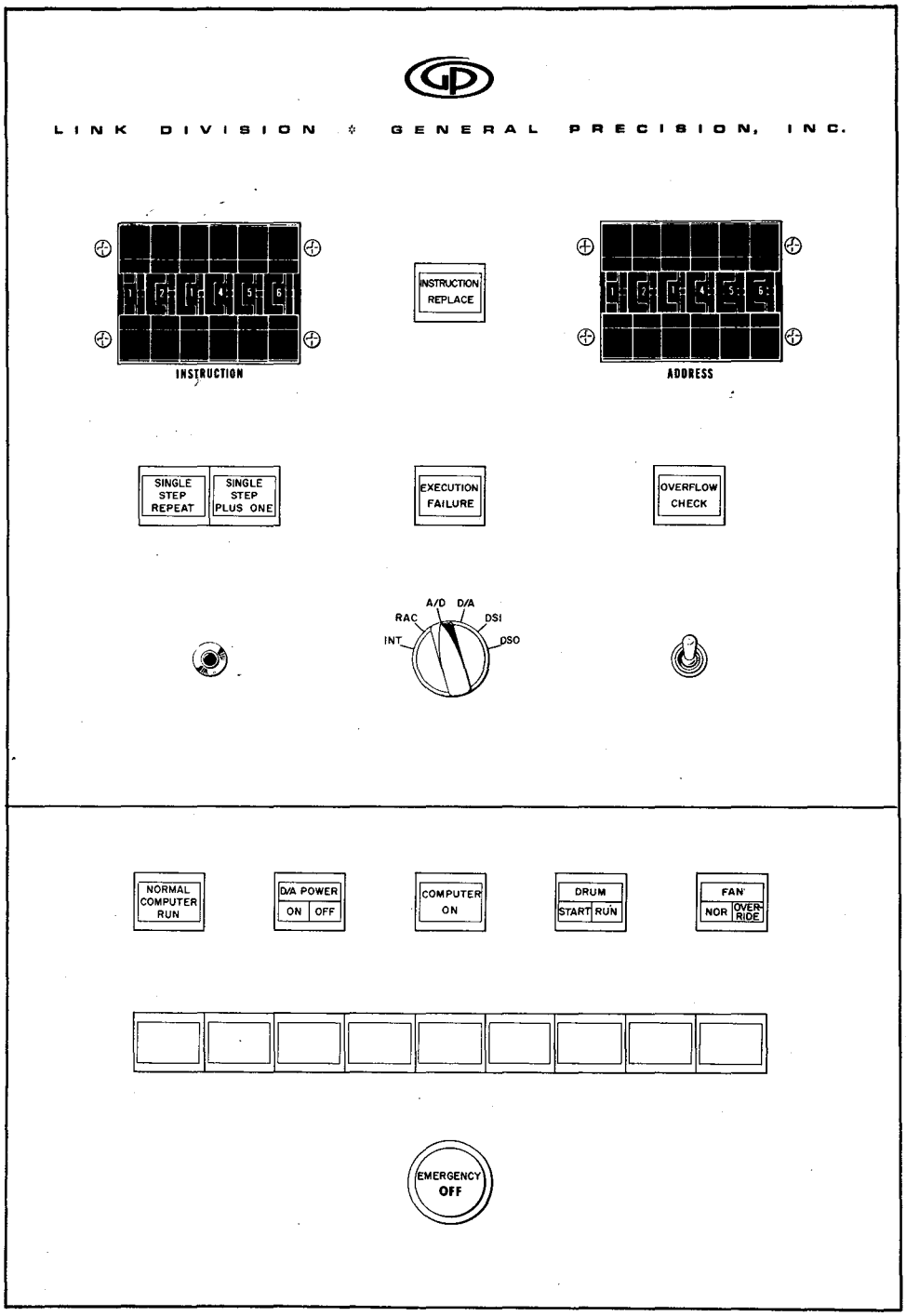


FIGURE 25 MARK I OPERATOR'S PANEL

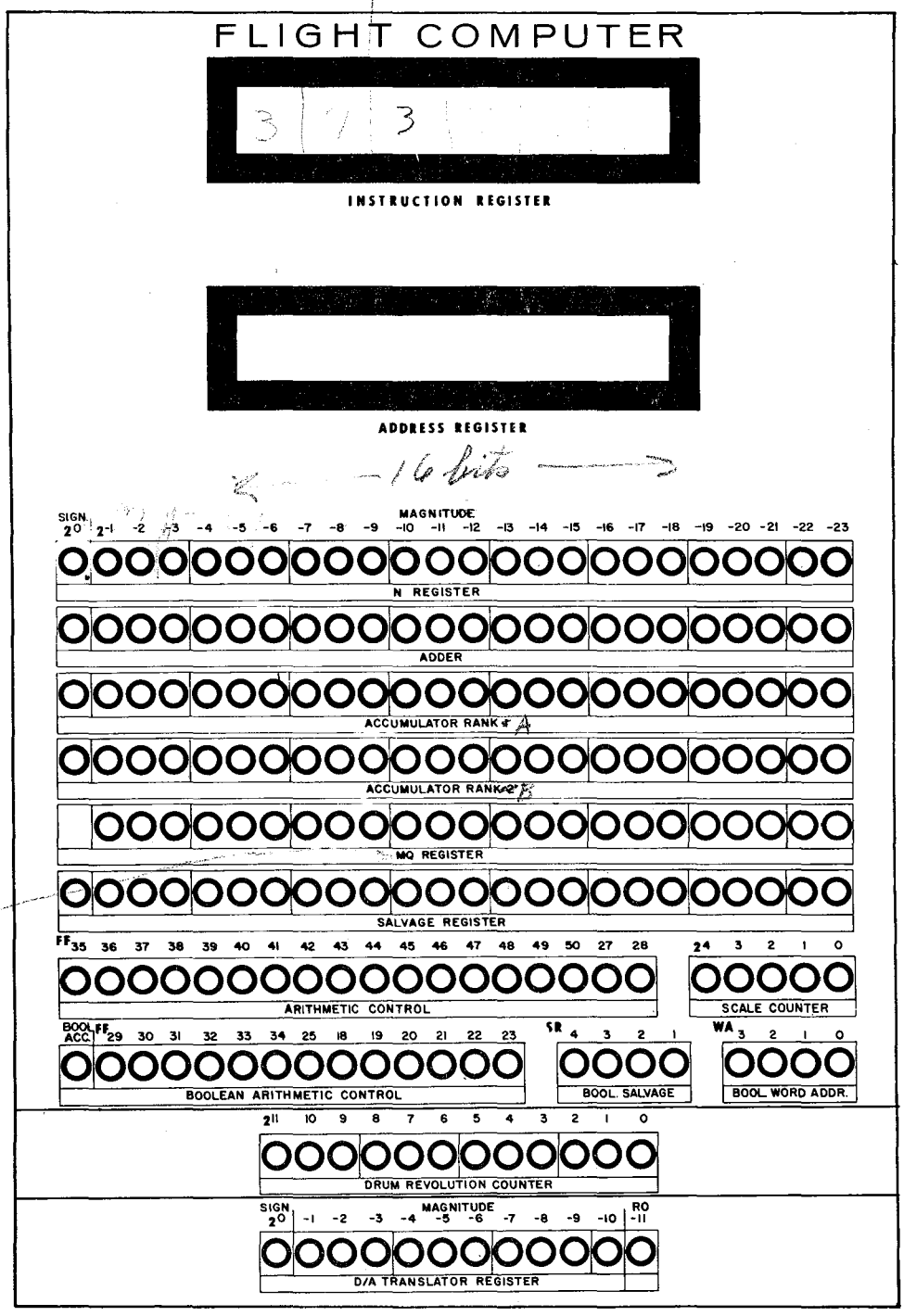


FIGURE 26 FLIGHT COMPUTER PANEL

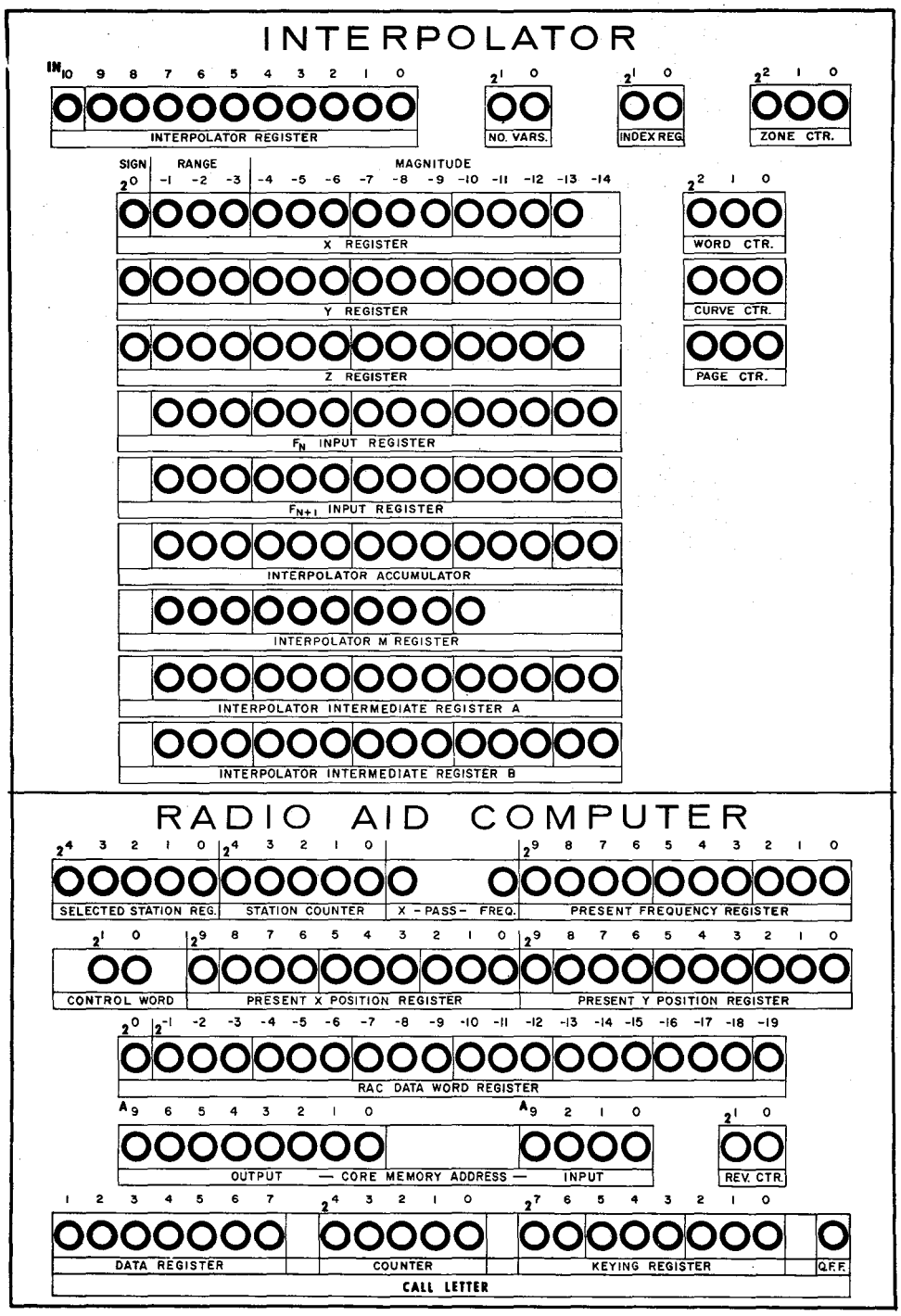


FIGURE 27 INTERPOLATOR/RADIO-AIDS PANEL

under priority control is provided. These are 1) interpolator core memory accesses, 2) A-D converter transfers to core memory, 3) D-A converter transfers out of cores, and 4) digital switch transfers in and out of cores.

The flight computer indicator panel (Figure 26) and the interpolator/radio aids computer indicator panel (Figure 27) will provide visual observation of the functioning of individual circuit elements during the one-step operating mode. Nixie indicators will be located in the upper portion of one panel. These indicators are being provided for operator convenience, displaying the registers indicating contents of instruction being executed and drum address of this instruction. These, together with the other operational register indicators, operate only during a manual single-shot process or after a conditional stop has occurred (any display during normal operation would not produce intelligible data because of the extreme speed of the Mark I computer).

In addition to the three panels illustrated, rack-mounted panels are provided containing indicators and control circuits incorporated into the computer for program loading operations, individual power supply control, and for such malfunctions as power supply failures, overtemperature alarms, and other machine failures outside the logic area.

5.3 DIAGNOSTIC PROGRAMS

Before the Mark I digital computer is operated with the input and output equipment of the digital flight simulator, a diagnostic program will be loaded into the computer drum storage by the computer operator. This diagnostic program, designed to detect errors in all computer-dependent operations, will consist of two logical parts:

1) Computer diagnostic program

2) Input/output diagnostic program

The computer diagnostic program will completely check all possible internal operations of the Mark I, such as arithmetic, transfer, and Boolean operations. The radio aids preselection, digital function interpolation, and other internal data transfer operations will also be thoroughly tested. The input/output diagnostic program will determine whether the data transfers from the simulator equipment to the computer and from the computer to the simulator equipment are correct.

The diagnostic program will be punched on paper tape and will be loaded into the computer drum storage in exactly the same manner as the operational program. Duplicate copies of the diagnostic program tape will be provided because of the possibility of loss or mutilation of a paper tape.

Written and pictorial instructions concerning the diagnostic program will be included in the computer operator's manual. Using these instructions, the computer operator can easily load the diagnostic program, initiate the program, and determine whether the program has detected an error.

5.3.1 Computer Diagnostic Program

After the computer diagnostic program is loaded and initiated, the program will execute all possible internal operations, and compare the results against preplanned results loaded with the program. If the results agree, the computer will stop, and the computer operator will check the neon light configuration on the computer control panel to determine whether it agrees with a pictorial display in the computer operator's manual that depicts the error-free condition. Agreement between the panel display and the pictorial display will indicate that the program has operated successfully.

If an error occurs, the neon light indicators on the computer control panel will notify the operator of the type of error — that is, the operator's manual will contain pictorial displays of the light indicators for the various possible errors. Each type of error will have a unique light-indicator configuration.

5.3.2 Input/Output Diagnostic Program

The computer operator's manual will contain written and pictorial instructions pertaining to the proper setting of the various input devices. The diagnostic program will determine whether the input values agree with the preplanned values, and will indicate error or the error-free condition by illumination of the light indicators on the computer control panel.

The output portion of the input/output diagnostic program will set all output quantities to certain predetermined values. The computer operator or maintenance personnel can then read all the various indicators and determine whether the equipment is functioning properly.

5.4 ELECTRONIC CARD TESTER

A static card tester will be provided as part of the Mark I computer. This card tester will enable the maintenance personnel to determine whether an electronic card is functioning properly.

Suitable card mounts will be positioned on the card tester module to allow testing the various types of cards in the system. These card mounts, used in conjunction with rotary-type switches on the control panel, will be used for malfunction detection, debugging, and preventive maintenance.

A detailed instruction book will be provided with the card tester, illustrating the card mounts and rotary switch settings used for each type of card. The instruction manual will also show the indicator readings for a card that is operating normally, and also the indicator readings for the type of malfunction.

5.5 AUTOMATIC READ HEAD CHECKING

In order to utilize all available time in the Mark I computer, a dynamic read head checking circuit is incorporated to make use of time during the performance of arithmetic operations requiring more than one machine cycle. Multiplication and division operations, for example, require several machine cycles to complete. The computer program makes allowance for the time requirements of these arithmetic operations by the insertion of the NO-OP instruction for each segment of machine cycle time required. In effect, this NO-OP instruction inhibits any further transfer of data from the main drum memory; however, the data contained in the NO-OP instruction consists of a series of alternate 1's and 0's. This data is compared in the associated read head checking circuitry to allow verification of the ability of each individual read head to distinguish between a "1" state and a "0" state. The instruction data word is randomly alternated to continuously check both states of operation. That is, the first NO-OP data word would consist of the binary number 101010------, while the next NO-OP data word would consist of the binary number 01010---. In this manner, all read heads will be continuously checked for reading accuracy. The failure of any read head to read properly will illuminate an indicator light on the Mark I control panel.

5.6 TAPE PREPARATION EQUIPMENT

The designer of a special-purpose digital computer to be employed for real-time flight simulation is faced with a number of compromises in the selection of a method of introduction of data into the computer:

1) Since,for the overwhelming portion of its usable life, the computer will be employed in purely routine training operations, it is essential that the standard loading procedure be simple and relatively rapid.

2) Means must be provided to permit expeditious introduction of minor modifications to the program without extensive control manipulation or without requiring excessive amounts of time for individual changes.

3) Means must be provided to permit, on infrequent occasions, the keypunching or typing of an entire program by unskilled clerical personnel in an orderly and economical manner.

4) Means must also be provided to permit simulator engineers or maintenance personnel to produce small program fragments without recourse to elaborate programmed translation routines, awkward keypunch manipulation, or use of sophisticated data translation techniques which might be tolerable for wholesale production of entire programs but would be intolerable for the introduction of a small number of modifications into an existing program.

5) Consideration must be given to the reliability of the associated equipment and to the vulnerability of the physical program data to damage, improper loading sequence, loss of individual elements (in the case of punched cards), or wear due to repeated passage through mechanical reading devices.

From the viewpoint of rapid loading, the use of magnetic tape to store the basic program is most attractive. Unfortunately, magnetic tape readers generally are considered to be less reliable than photoelectric punched paper tape readers. Furthermore, magnetic tape is rather awkward to use since it cannot be edited conveniently and cannot be interpreted through visual inspection. Since the entire drum loading operation for the Mark I can be accomplished in less than 6. 5 minutes employing a photoelectric punched paper tape reader, the potential loading speed attainable with magnetic tape is believed to be unjustified.

From the standpoint of convenience in modifying programs, experimenting, and introducing minor variations in the sequence of compu-

tations, the use of punched cards for program information is highly desirable. The advantage of the use of punched cards completely disappears, however, if the data contained in the cards must be punched in a language other than direct program language or if it must be machine-translated before loading, since the only real advantage of punched cards lies in the fact that the individual fragments of a program can be physically manipulated individually to facilitate minor changes in programming. Unfortunately, the required to load an entire program using punched cards may be excessive unless an unusually fast and expensive card reader is employed. For example, the conventional card readers employed in present IBM machines read 200 cards per minute. The large number of instructions required in a computer for flight simulator operation might entail as much as an hour's reading time at this relatively slow rate, unless the information content of each card is condensed to such a degree that the flexibility advantage of the punched cards is seriously compromised.

To satisfactorily meet these diverse requirements, the Mark I is designed to employ paper tape as the basic means of introducing information into the computer. To capitalize on the flexibility advantage of punched cards, however, a fast photoelectric punched card reader is provided in the tape preparation unit to prepare a machine-language paper tape from a group of punched cards. Thus, the advantages of the relatively rapid and foolproof paper tape system are retained, while, at the same time, the programmer is permitted the use of punched cards in the assembly of a tape to permit convenient insertion, deletion, and rearrangement of individual program and data words.

The tape preparation unit employs a photoelectric card reader which reads standard IBM cards at a rate of 240 cards per minute. Each card may contain any desired number of instructions from 1 to 8, or any desired number of interpolator data words from 1 through 9. Restriction of the maximum number of instruction words per card permits the retention of conventional IBM Hollerith code for keypunching of the instructions, spaces the individual instructions to such a degree that they may be conveniently read visually (if keypunched on a IBM 026 keypunch with printing attachment) and minimizes the size of a card deck corresponding to one 1024-word sector of the program. (It is considered that most programmed operations, both in initial programming and in modification, will be accomplished in 1024-word groups.) At eight instructions per card, a 1024-word sector may be prepared from a stack of cards less than one inch in thickness.

The tape preparation unit employs a 110-character-per-second Teletype Corporation high-speed tape punch. This punch is widely used for communication purposes and is believed to be highly reliable.

A group of 128 cards, each containing eight instructions per card, can be translated into a machine tape in approximately 20 seconds. As stated earlier, it is not necessary that each card contain eight instruction words. For modification of existing programs, it is often convenient to introduce instructions one word per card. Furthermore, groups of cards containing any number of instructions (not exceeding eight per card) may be intermixed at random with no disadvantageous consequences other than loss of time — that is, the tape preparation unit effectively wastes time when potential data fields of the card are not employed. In other words, a 1024-word sector punched two words per card would take four times as long (or approximately 80 seconds) to translate to tape as a corresponding sector punched eight words per card. After preparation of a tape, the cards need not be reused, unless modification of the program is required.

The preparation of interpolator data in the card-to-tape conversion equipment introduces two problems not encountered in the preparation of general-purpose instructions. These are:

1) As the eight straightline segment approximations employed for linear interpolation require nine data points per curve, it is highly convenient to increase the allowable number of data words per card to nine to permit each card to contain all the numerical data representing a single curve.

2) It is clearly desirable that the numerical information for the interpolator be punched into the cards in decimal language rather than in octal. Accordingly, the card-to-tape equipment provided with the Mark I computer contains internal circuitry to translate decimal keypunch information from IBM cards to binary before activation of the tape punch.

Information for the radio-navigation facilities also presents additional problems arising in the following areas:

1) The basic data words are 20 bits in length.

2) A considerable variety of different types of data words are employed to characterize each navigation facility; these words not only contain numerical information in binary form but also contain information regarding call-letter generation patterns.

To achieve the highest possible order of standardization and simplification of the tape preparation circuitry and drum loading circuitry of the basic Mark I computer, the 20-bit radio words are assembled from two 16-bit pseudo general-purpose instructions. Through this concept the relatively small number of instructions or data words required for the four radio data sectors of the drum may be prepared by the same keypunching philosophy and circuitry techniques employed for the generation

of general-purpose instructions. A detailed instruction manual will be provided with the computer to permit the programmer to assemble the appropriate pseudo instructions from the various types of numerical and call letter information required for each navigational facility. Thus minor modifications in the radio data may be prepared by preparation of a pseudo program, using the translation handbook to prepare the individual instructions.

It should be recognized that the keypunch operations are performed in direct programming language without recourse to indirect statements, oblique language, translation during keypunching,or similar confusing inefficiencies. Punching of large volumes of instruction cards for preparation of an entire program is best handled through the utilization of an IBM 026 keypunch,which is not provided with the basic Mark I computer but is widely available at any facility employing even the simplest of IBM accounting equipment. No modifications to the keypunch are required and the keypunch technique is identical to that employed in standard IBM business operations.

For modest-scale experimentation and alteration of programs, it is entirely satisfactory to employ the small IBM hand keypunch provided with the basic Mark I equipment. Reasonable numbers of cards can be punched with this device at little inconvenience, although it is somewhat inefficient for producing cards in quantities of hundreds or thousands.

It should be recognized that the individual instruction cards prepared for the Mark I tape preparation unit may be processed in various ways through straightforward utilization of unsophisticated IBM tabulating and accounting equipment, if such machines are available. For example, programs written one instruction per card might advantageously be condensed to eight instructions per card through longstanding procedures commonly employed in IBM tabulating equipment. Furthermore, the instruction decks may be listed on simple tabulating machines to provide a highly readable copy of the instructions for record purposes. The extent to which the many advantages of machine manipulation or processing of the individual program cards can be capitalized upon is, of course, dependent upon the extensiveness of IBM facilities readily available to the using activity. Since IBM facilities of the relative simplicity required to handle the problems described above are almost universally obtainable but are not mandatory for operation of the tape preparation unit or the digital simulator itself, it is believed that the proposed procedure for generation of program tapes offers the optimum combination of reliability, simplicity, and low cost, while permitting certain idealistic extensions of the basic concept to be accomplished in the likely event that simple additional IBM equipment is available on a part-time basis for use by the using agency.

With the exception of an 026 keypunch, it is considered improbable that the usefulness of any additional standard IBM equipment would justify rental or puchase of such equipment by an agency interested primarily in operation of a flight simulator for training or research purposes.

If desired, the high-speed paper tape punch may be controlled directly through an electrical keyboard provided with the basic Mark I system. Use of this keyboard, which resembles that of a conventional desk calculator, eliminates the need for punched card input to the tape preparation unit. The actual numerical typing operations employing the electrical keyboard are identical to those employed with the IBM 026 keypunch or the IBM mechanical keypunch (that is, the actual numerical keys punched are the same for all three machines). The numerical keyboard may be employed to enter decimal information for the interpolator data, with automatic translation to binary occurring within the tape preparation unit circuitry. While tapes may be prepared employing the manual keyboard at a typing speed approximately equivalent to that which may be accomplished on the 026 keypunch, it should be recognized that the individual instructions are punched into a continuous paper tape and can be removed, rearranged, or changed only through tape splicing operations.

The tape preparation unit may also be employed to duplicate paper tapes, to generate file copies, or to reproduce tapes which are beginning to show evidence of wear. It should, of course, be recognized that new tapes may also be prepared through utilization of a master file of IBM card instructions as an alternate to the duplication facility provided.

5.7 AUTOMATIC PRINTOUT EQUIPMENT

As in the case of the tape preparation unit and drum loading equipment, the selection of suitable output printing apparatus for a digital simulator is something of a compromise. The use of a Flexowriter (or similar electrically-controlled typewriter) offers the advantage of highly flexible format (including printing of alphanumeric characters and punctuation marks) and the ready availability of parts and service throughout the world. Unfortunately, the use of an electrically controlled typewriter or teletype machine results in seriously limited printout speed: the printing speed of a teletype machine or electric typewriter is exasperatingly slow when relatively large volumes of data must be listed. Furthermore, the reliability of the extremely complex electric typewriters is subject to considerable question.

The problem of printing speed and reliability may be solved through the use of one of the many high-speed line printers now available for use with data processing equipment. Unfortunately, these devices, while highly reliable, are extremely expensive. Furthermore, an inordinate amount of electronic buffering circuitry is required, since the major speed advantage of the less exotic line printers is achieved through parallel printing of large numbers of characters, with resultant extravagant electronic circuitry requirements.

To achieve a compromise between the extravagance of the use of high-speed line printers and the slow operating speed and suspect reliability of electric typewriters, the Mark I printout equipment employs a Hewlett-Packard printing mechanism which is, in effect, a foreshortened version of a full-scale line printer. That is, the device prints all characters of a single data word simultaneously and employs a time-division print control mechanism somewhat similar to that employed on more sophisticated line printers. The control circuitry and activation mechanism of the Hewlett-Packard printer is considerably more simple and is believed to be considerably more reliable than the electronics and mechanisms associated with conventional electric typewriters. The principal disadvantage of the simple Hewlett-Packard printer lies in the fact that it prints individual words in sequence on a tape somewhat similar to adding machine tape. However, fan-fold paper tape will be employed in the Hewlett-Packard printer to eliminate the undesirable coiled paper tape of the conventional adding machine print mechanism. Through this procedure the Hewlett-Packard line printer becomes the equivalent of a full-scale page printer with the exception that it prints on a narrow page having but a single column of figures per line.

The printer may be employed with the Mark I computer or tape preparation unit to accomplish the following purposes:

1) Listing of program tapes. In this mode of operation the circuitry automatically translates interpolator data words from binary to decimal to permit convenient interpretation of functional data in decimal form.

2) Listing of contents of core memory. During certain program debugging operations, it is sometimes desirable to print out the contents of the working memory of the computer. This is particularly useful when the programmer is completely baffled by apparently inconsistent results and would like to obtain a convenient listing of the actual contents of all memory locations at a particular instant. In operations such as this, the relatively slow printing speed of an electric typewriter is particularly exasperating.

3) Printout of accumulator contents during one-step operation of the computer. During program debugging operations or computer maintenance operations, it is sometimes desirable to operate the computer in the one-step mode. The printout unit permits the printing, in either octal or decimal equivalent, of the contents of the accumulator following each step of the program in the one-step mode. Through the use of this printout facility, the necessity of reading register(Nixie)lights and recording the observations in a series of tabulations is eliminated.

4) Printout of memory locations in which computational overflows have occurred. In the Mark I it is possible to select a mode of computer operation in which the least significant bit of core memory is sacrificed for use as a tally to indicate the presence of a computer overflow during the calculations preceding the storage of an answer in the core memory location under discussion. In this mode of operation any overflow, once tallied, remains in the memory until manually cleared. This tabulation of overflows may be accumulated for any desired length of time during normal or one-step operation. After accumulating the historical overflows for the desired time period, the computer may be stopped and the address of the memory locations containing an overflow tally in the least significant bit printed out by means of the Hewlett-Packard printer. The programmer can then scan the list of memory locations for an identification of the individual calculations in which overflows have occurred at least once during the test period. It is believed that this facility will be highly useful in relieving the programmer of the ever-present worry of the consequences of injudicious scaling. This consideration is particularly important in aerodynamic calculations, since it is often difficult, even for a highly experienced aerodynamicist, to place a realistic upper bound on the magnitude of a variable under computation. Since it is relatively easy to correct injudicious scaling if the offending terms can be identified, the advantage of a historical tally of overflows is obvious.

6. PHYSICAL DESCRIPTION OF COMPUTER

The general arrangement of the Mark I computer is shown in Figure 28. The computer is housed in nine standard bays arranged in a single line to minimize cable runs. Each bay is 2 ft. wide, 2 ft. 2 in. deep, and 7 ft. high, requiring approximately 39 square feet of floor area. Each bay is fitted with an integral blower unit for equipment cooling. The air intake is across the lower front and the air exit is through louvers on the upper rear of each bay, providing approximately 5,400 cfm of airflow through the computer.

Casters and lifting eyes are available, the additional heights of which are indicated in the following table. Mounting provisions for these accessories are provided in all bays. Fork lift slots in the flush bases are provided in all bays. The nine bays are arranged in four double sections and one single section, which are separable for shipment, and are bolted together upon installation.

The design of the equipment is such that the use of a computer floor installation is not required. Input power access and connectors for all analog and digital inputs and outputs are located at the rear of the top plates of appropriate bays. A photograph of the Mark I computer cabinets is shown in Figure 29.

Mark I Installation Data:

Length	18 feet
Depth	26 inches
Height	77-5/8 inches
Caster Height	1-7/8 inches additional
Lifting Eye Height	2-5/8 inches additional
Weight	5,640 lbs. (approximate)
Air Intake	5,400 cfm (approximate)

Dimensions of maximum-size module for shipment:
77-5/8 inches by 26 inches by 48 inches.
(Can be shipped in any attitude)

Power Consumption:

117 V, 60 cps A C, Load ~65 amperes ±10%

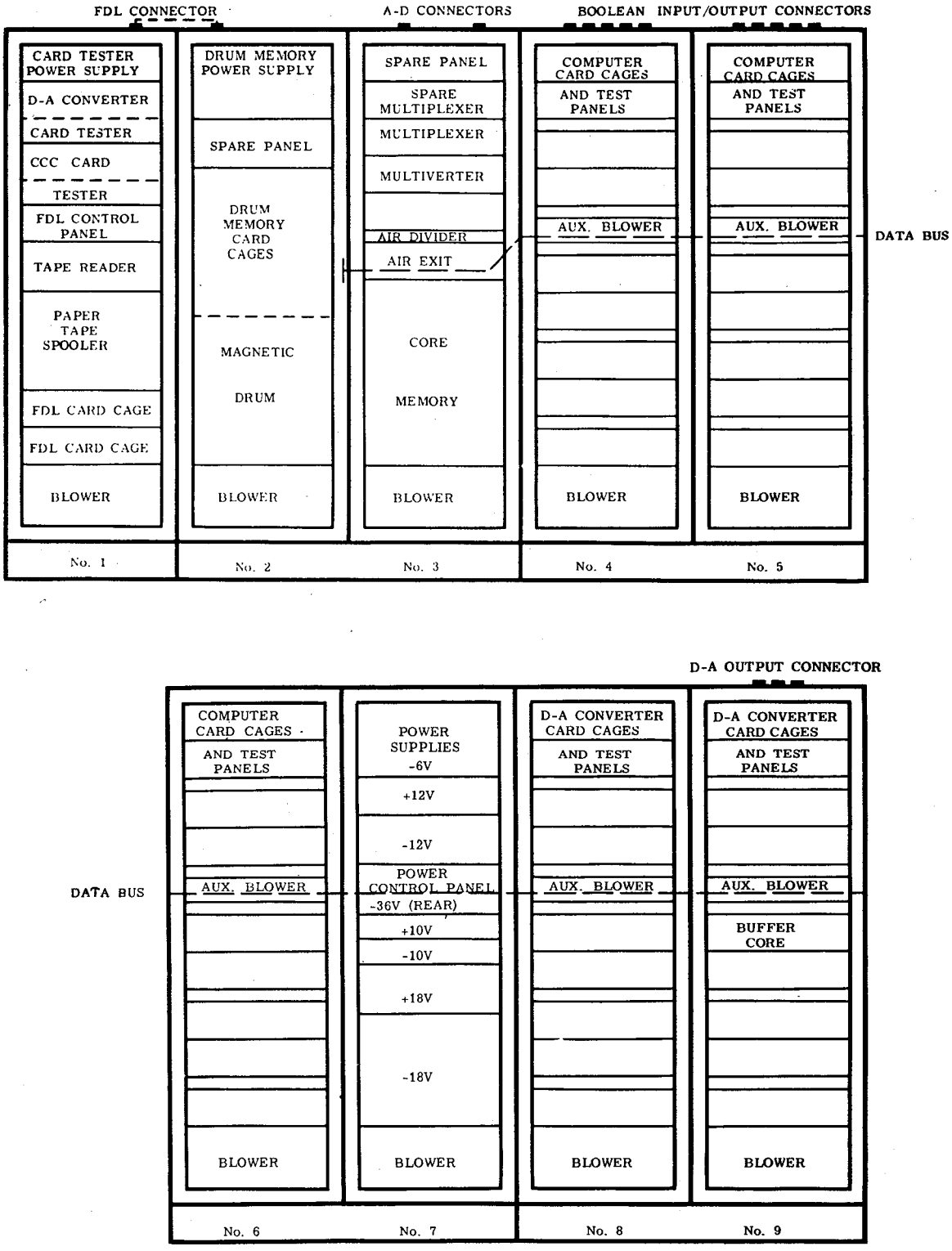


Figure 28 ARRANGEMENT OF MARK I COMPUTER BAYS

FIGURE 29 MARK I COMPUTER CABINETS

7. SUMMARY OF COMPUTER CHARACTERISTICS

7.1 FLIGHT COMPUTER

1) Arithmetic Unit

a) Data Word Size: 24 bits (sign bit plus 23 bits magnitude)

b) Type of Arithmetic: Fixed-point fractional binary, with full sign control. Results expressed in true-value magnitude and sign.

c) Operational Registers: Input (N) Register, Accumulator (A) Register, Multiplier-Quotient (MQ) Register, Salvage (S) Register.

d) Arithmetic Operations: Add, Subtract, Multiply, Divide, Negative Multiply, Square, Square Root Step, Scale, Shift, and other special operations. Multiply and Divide instruction time = 30.5 microseconds. Add instruction time = 6.1 microseconds. Basic Add time (without access) = 1.0 microsecond .

2) Boolean Arithmetic Unit

a) Data Word Size: 1-bit Boolean expressions, stored and accessed in 16-bit words, with 1-bit selection provided through a special bit-alteration mode of the data memory.

b) Arithmetic Type: Calculation of Boolean functions.

c) Operational Registers: Input (N) flip-flop, Boolean Accumulator (BA) flip-flop, four salvage (BS) flip-flops.

d) Arithmetic Operations: Boolean Sum (OR), Boolean Product (AND), Boolean Invert (NOT). All processes performed in basic instruction time of 6.1 microseconds.

3) Data Memory

a) Type: Magnetic core storage for 2,048 24-bit words. Random-access, parallel operation, with 5.0-microsecond cycle time. Memory processes: read-restore, clear-write, and bit alteration.

b) Use: The data memory is shared by the several portions of the computer subsystem on a priority basis, as follows:

(1) Flight Computer instruction processing.
(2) Linear Interpolator function processing.
(3) Radio Aids station preselection processing.
(4) Flight Computer Input-Output transfers.

4) Priority Control

The Flight Computer instruction program normally will be composed of a large proportion of instructions which require more than one 6.1-microsecond period for execution. For such instructions, only the first 6.1-microsecond period of the 30.5 or 36.6 microseconds of processing time is required for data memory access. Therefore, processing of items of 2, 3, or 4 priority order is performed automatically during the remaining free time periods. This permits performance of these auxiliary processes without requirement for program control, input-output subroutines, or program-interrupt features.

5) Program Memory

a) Type: Magnetic drum storage. (BAND) Track length: 4,096 bits. Speed: 40 revolutions/second. Useful storage: 65,382 words, plus clock tracks.

b) Use: The drum memory is shared by several portions of the computer subsystem, with space allocations as follows:

8 bands x 4K x 2 bytes = 64K bytes (1) Flight Computer Instructions: 11 bands, each containing 16 tracks. Useful storage: 40,946 words of 16 bits each.

3 2K bytes (2) Linear Interpolator Instructions and Data: Four bands, each containing 11 tracks. Useful storage: 16,344 11-bit words.

1 band x 9K x 3 bytes = 12K bytes (3) Radio Aids Station Data: One band containing 20 tracks. Useful storage: 4,092 20-bit words.

TOTAL DATA STORAGE = 108K PLUS CLOCK TRACKS

6) Instruction Control

a) Instruction Form: Single-address, dual-process with zero address used as flag to initiate performance of alternate execution process.

b) Instruction Size and Format: Operation Code: 5 bits. Execution Address: 11 bits. Modification of detail of address region exists for certain instructions.

c) Address Modification (Indexing): Not provided.

d) Logical Branching: Not provided.

e) Branching: Provided in the form of a Conditional Skip instruction which permits non-execution of 1 to 127 following program steps.

f) Instruction Processing: The 11 bands of instruction storage in the program memory are divided into groups and processed at differing rates, as follows:

Group	No. Bands	Title	Processing Rate
1	1	Fast Instruction Band	20/second 50ms
2	2	Medium Instruction Bands	5/second each 200ms
3	8	Slow Instruction Bands	1.25 second each 1.25 S

7) Operational Control

a) Basic Clock Rates: Drum Clock rate: 6.1 microseconds. Operational 1.0-microsecond clock phases (1-6) are synchronized with drum clock, and 0.5-microsecond subphases are synchronized with operational clocks.

b) Operational Counters: Drum address counter. Drum band sequencing counter. Drum revolution counter. Auxiliary counters for the above information, but under Operating Mode control.

c) Operating Modes:

(1) Normal: Under clock control.

(2) Single Shot Repeat: Under manual control. Repeat one addressed instruction for each operation of control.

(3) Single Shot Advance: Under manual control. Execute addressed instruction, and increase address in Auxiliary Counters by one for each operation of control.

(4) Instruction Replace: Manual controls provided for insertion of content and address of a single instruction which will replace drum instruction at addressed location, under manual control, in either normal or single-shot modes.

(5) Overflow-Check: Under manual control. The history of program overflows between successive Store instructions is placed in 24th bit location of data memory.

(6) Auxiliary Process Execution Failure: Under manual control. For either Linear Interpolator or Radio Aids process, the occurrence of an execution failure will cause computer to stop, with drum address of failure point contained in auxiliary counters, and core address of affected item in appropriate register.

8) Sampled-Data Inputs and Outputs

a) A/D Multiplexer and Converter

(1) Size: 64 input analog channels, with space provided for extension to 126 channels. Transfers are under priority control.

(2) Precision: 14 bits and sign.

(3) Type: Packard-Bell EM-3 Multiplexer, and M-2 Multiverter.

(4) Sample Rate: 20/second, for total of 126 channels.

(5) Conversion Time: 64 microseconds.

(6) Input Range: ±10 volts. Analog Input Passband approximately 0 to 3 cps.

b) Digital Switch Inputs

(1) Size: 1,024 binary channels, interrogated in 64 groups of 16 bits, with transfers under priority control.

(2) Sample Rate: 20/second, for total of 1,024 channels.

(3) Input: One Form A contact on each switch, etc., to be sampled.

c) Digital Switch Outputs

(1) Size: 256 binary storage elements.

(2) Type: Each element is composed of a mercury-wetted Form D contact, with three leads brought out for external use. Modifications of state of these contacts are under priority control.

(3) Sample Rate: 10/second, for total of 256 channels.

d) D-A Converter

(1) Size: 192 analog output channels. (Modular circuit cards installed only for number of channels required, which is normally 128.)

(2) Precision: 11 bits plus sign.

(3) Output Range: ±10 volts.

(4) Sample Rate: 80/second, for total of 192 channels.

(5) Input: Word-serial from data memory to D-A Core Buffer Memory. A code translator is provided to furnish compatible inputs to D-A unit. Transfers are under priority control.

(6) Output: Word-parallel, bit-serial, pulse-time-modulated analog channels, under timing control from drum address counters. Analog output filter on each channel with passband of approximately 0 to 3 cps.

7.2 LINEAR INTERPOLATOR

1) Arithmetic Unit

a) Data Word Size:

(1) From Program Memory: 10 bits, with sign assumed positive.

(2) Arithmetic Unit, Internal: 14 bits, with roundoff. Sign assumed positive.

(3) Arithmetic Unit, Output: 16 bits, without roundoff. Sign assumed positive.

b) Type of Arithmetic: Fixed-point fractional binary. Signs assumed positive.

c) Arithmetic Operation: Linear interpolation process for equidistant increments of argument.

d) Process Time: 120 microseconds for a function of one variable.

e) Operational Registers: Input Registers, $F(X_n)$ and $F(X_{n+1})$. Accumulator (IA). Multiplier Register (MR). Result Registers (RA) and (RB).

2) Data Memory

The data memory contains argument values, X, Y, and Z. Most significant 14 bits of data word transferred to interpolator input registers under priority control, with argument addresses obtained from program memory as described below. Provides storage for function (answer) values.

3) Program Memory

a) Interpolator Instructions:

(1) Size: 1-bit flag, 10-bit control word or data memory address.

(2) Order: Order of instruction words in drum band indicates significance, obviating need for operation code.

1. Control instruction, giving number of variables and indexing information.
2. X, Y, and Z data memory argument address instructions.
3. Data
4. Answer instruction, data memory address for function.

(3) Indexing: 0-3 indexing counter value may be added to all or any combination of X, Y, and Z argument addresses, and to answer address.

b) Interpolator Data:

(1) Size: 1-bit flag, 10-bit data word, with sign assumed positive.

(2) Order: For a function of one variable, a straight line approximation to the function with equal intervals of the argument will be provided by nine data words. The represented function will lie entirely within the first quadrant. For a function of two variables, nine such curves will be provided, each composed of nine data words. For a function of three variables, nine pages, each composed of nine curves, will be provided.

4) Interpolator Input Registers

a) Size: 14-bits, composed of 1-bit sign, 3-bit argument interval identification, and 10-bits argument increment.

b) Operational Registers: Three input registers, (X), (Y), and (Z).

c) Sign Control: It sign is positive, the sign bit is dropped and transferred value is used with sign assumed positive. If sign is negative, the argument value is made identically zero.

d) Addresses: Data memory addresses of argument values are obtained from interpolator instructions X, Y and Z in program memory.

5) Interpolator Control

The Linear Interpolator processing is under wired-program control, with process sequencing under priority control, and with selection of process form, selection of indexing, and data memory addressing under instruction control.

6) Interpolator Processing

The Linear Interpolator has inputs and outputs only with respect to the Data Memory for arguments and interpolated function values. Since four Program Memory drum bands are assigned to Interpolator function tables, the complete field of 16,344 word-locations will be processed each 0.1 second. Therefore, interpolated function values are automatically and continuously available to the Flight Computer at this rate.

7.3 RADIO AIDS COMPUTER

1) Preselection Unit

a) Data Word Size: 10-bits, stored in the Program Memory in parallel groups of two words, forming 20-bit words of the Radio Aids band.

b) Type of Process: Limit Comparison on Upper and Lower Bounds of ten-bit parallel data words.

c) Operational Registers: Data Memory Acquisition Address Control for X-position, Y-position, and Frequency-set input data. Upper and Lower Limit Comparators. Selected-Station Data Register. Data Memory Transfer Address Generator for selected station data.

d) Process Operations: Limit Comparison of X, Y, F control data, preselection of detailed station data words, and transfer to Data Memory.

NOTE: Comparison performed with respect to X, Y control data only for all Marker Stations.

2) Data Memory

Preaddressed locations provided for following preselected data.

Number of Stations	Station Class	Number of Data Words
1	Middle Marker	4,20-bit
1	Outer Marker	4,20-bit
1	Fan-Z Marker	4,20-bit,and 4,16-bit
2	LF Stations	8,20-bit,and 4,16-bit
4	VHF Stations	4,20-bit,and 4,16-bit

Transfer of preselected station data is under priority control.

3) Program Memory

Radio aids band provides storage capacity for various types of stations, as follows:

Number of Stations	Station Class
32	Middle Marker
32	Outer Marker
32	Fan-Z Marker
95	LF Stations
32	LF/AN Stations
127	VHF Stations
350	Total

4) Keying Function Generator

a) Operational Registers: Data Memory Address Generator for Call Letter Groups for the preselected stations of Fan-Z Marker, LF, and VHF classes. Data Memory Readout Timing Generator. Keying Signal Generator.

b) Operation Mode: The main program is permanently blanked during word times 0-7 on all Program Memory bands. This period is used during each fourth drum revolution to transfer out the selected Call Group bit from appropriate preselected stations in the Data Memory.

c) Timing: The bit timing is 0.1 second. The total Call Group pattern timing is 32 seconds, with a dead band of 6.4 seconds, which is used for operation of external VORTAC signal generator.