Programming support for a small research computer

Pulfer, J. K.; Wein, M.

Report (National Research Council of Canada. Radio and Electrical Engineering Division : ERB), 1968-11

ERB-794

UNCLASSIFIED

NATIONAL RESEARCH COUNCIL OF CANADA

RADIO AND ELECTRICAL ENGINEERING DIVISION

PROGRAMMING SUPPORT FOR A SMALL RESEARCH COMPUTER

J. K. PULFER AND M. WEIN

OTTAWA

NOVEMBER 1968

ABSTRACT

A set of programs is described which convert a small process control computer into an instrument for research in computer usage problems. The authors attempt to give an over-all view of the programming support which it is hoped will provide a background to aid the understanding of the more complex programs used in man-machine communication research.

PREFACE

This report is one of a series describing programming support for a digital computer used by the Radio and Electrical Engineering Division. The computer, Model 840A, manufactured by Systems Engineering Laboratories Inc., is being employed as a tool for research in computer usage problems. It is not used for computation or for scientific or business data processing. Specific areas of interest include: man-machine communications using graphical CRT displays, time sharing and multiprogramming problems on small computers, analysis and synthesis of sound including speech and music, three-dimensional graphic construction and manipulation, and the production of animated films.

The purpose of this report is to provide background to readers not familiar with system software, to enable them to make the most use of the detailed program descriptions available in other reports.

The programs described in this report were, for the most part, supplied by the computer manufacturer.

CONTENTS

FIGURES

PROGRAMMING SUPPORT FOR A SMALL RESEARCH COMPUTER

– J.K. Pulfer and M. Wein –

Introduction

A general purpose digital computer has often been called a universal machine because its behaviour, or response to input stimuli, depends almost completely on the set of instructions or program stored in its memory. It follows that without a stored program the computer can do virtually nothing. Before a data processor can be used as a research tool as described above, a number of programs must be available which will enable it to perform the routine jobs of input and output, code translation, editing, and translation from a user language into binary machine instructions. Programs of this type are often referred to as SYSTEM programs and the total set of support programs is given the name SYSTEM SOFTWARE.

As an example, a common type of system program, when stored in the memory of the processor and executed, makes the computer behave like a data transmission machine. Data may be transferred from some external storage medium such as paper tape into a selected area of the central processing unit (CPU) core memory. Such a program is usually called a LOADER.

A complementary program which makes the computer into a machine for transmitting data from an area in internal core memory to an external storage medium is often called a DUMP program.

Before going into a more thorough description of system software it will be instructive to see what significant operations the computer can perform without a stored program. These aspects of the performance of a data processor are often classified under the heading of SYSTEM HARDWARE. We shall discuss only the hardware of the central processor and the specific pieces of peripheral equipment used in this installation.

Hardware

Figure 1 is a block diagrm of the computer with its peripheral equipment. Communication between a particular peripheral unit and the central processor takes place via three interconnecting cables. Because each cable is common to all peripheral devices, it is called a BUS. The three busses are named the I/O or INPUT–OUTPUT BUS, the BTC or BLOCK TRANSFER CONTROL BUS and the INTERRUPT BUS.

I/O BUS

Data for all input and output operations are transferred in parallel, 24 bits at a time, by the I/O bus. Control signals for a single word data transfer which takes place as the result of executing a stored instruction are also transmitted by the I/O BUS.

BTC BUS

Control signals are transmitted by the BTC BUS for the transfer of a number of words or block of data for which individual transfers are controlled by hardware rather than by the execution of instructions. The hardware which controls the transfer is part of the central processor and is called the BTC or BLOCK TRANSFER CONTROL UNIT.* A set of locations in the CPU magnetic core memory (CORE) are permanently assigned to the storage of quantities which define the beginning address and number of words in a block transfer. For the machine described above, the addresses of the BTC dedicated locations are 48 and 49. Addresses are more often given in octal numbers or radix 8 notation. Thus 48 and 49 are usually written in octal as 60_8 and 61_8. Sometimes the $_8$ is omitted and octal arithmetic is implied by the context.

INTERRUPT BUS

Another important aspect of computer hardware is the INTERRUPT BUS. The interrupt bus is used to transfer signals which are called INTERRUPT REQUESTS from peripheral devices to the central processor. On receipt of an interrupt request, the central processor will stop processing the current program (providing certain initialization procedures have been performed) and transfer control to a second or interrupt program. The process of recognizing an interrupt request, establishing its order of priority relative to other requests, and transferring control from one program to another according to priorities is performed

---

*Up to eight BTC units can be installed in the processor.

by the INTERRUPT SYSTEM hardware in the central processing unit. Memory locations from 100_8 to 137_8 are assigned to the interrupt system as storage locations for the instruction which will be executed by hardware when a corresponding request is recognized. The instruction for the highest priority interrupt is stored at 100_8 and the instruction for the lowest priority is stored at 137_8. A subroutine call or a store-place-and-branch instruction to the desired interrupt processing routine is the most common type to be stored at the interrupt location.

The interrupt system hardware is initialized by executing instructions which enable or disable any or all interrupt levels in each group of sixteen. Interrupt request lines corresponding to the assigned memory addresses 116_8 and 117_8 are carried by the I/O bus. These two interrupt lines are available to all I/O devices. Other interrupts to which functions have been permanently assigned are: 100_8 for which a request is raised when power fails, 114_8 for which a request is raised when a block transfer is complete, and 102_8 for which a request is raised when a program protect violation has occurred (as explained below).

The central processor contains a feature which allows each core memory location to be set to either a protected or unprotected state by setting or clearing the 25th bit (protect bit) in the word. Special circuits check the status of the protect bit in memory location before executing each instruction and make a decision as to whether or not the instruction is to be allowed. If the action required by the instruction is not acceptable (depending on the initialization of the system) a memory-protect violation interrupt is requested. The hardware which performs these functions is called the PROGRAM PROTECT SYSTEM. The program protect system is put into operation by switching a key on the computer console to a position referred to as the PROTECTED MODE. When the central processor is in the protected mode, an unprotected instruction will cause a violation if it attempts either to modify a protected memory location, or to branch to a protected memory area.

The 26th (and last) bit in each word is automatically set to either a *one* or a *zero,* so that the total number of *ones* in the word is even. This additional bit is the PARITY BIT and the memory word is said to have EVEN PARITY. The parity is checked on reading from memory, and if an error results in odd parity in a word, a parity-error interrupt is raised. More details of the PROGRAM PROTECT SYSTEM, the INTERRUPT SYSTEM and the I/O SYSTEM can be obtained from the manufacturer's manuals.

Discussion of the hardware will be concluded with a brief description of the central processing unit. Data within the machine are manipulated in the following forms:

1) Words – which are 24 bits in length
2) Addresses – which are 15 bits in length
3) Operation codes – which are normally 6 bits in length.

Two 24-bit registers, called the A and B accumulators are available to the programmer for arithmetic operations and three 15-bit registers called index registers 1, 2, and 3 are available for address modification. In the machine being described, core memory consists of 8192 twenty-four bit words.*

All registers and any memory location may be observed using lights, or modified by switches, at the control console when the computer is halted. The contents of any combination of 24 console switches may be transferred to the A accumulator under program control. The switches are called CONTROL SWITCHES. Four of these switches may also be used to define conditions for skip instructions. When used for the latter purpose, they are referred to as SENSE SWITCHES.**

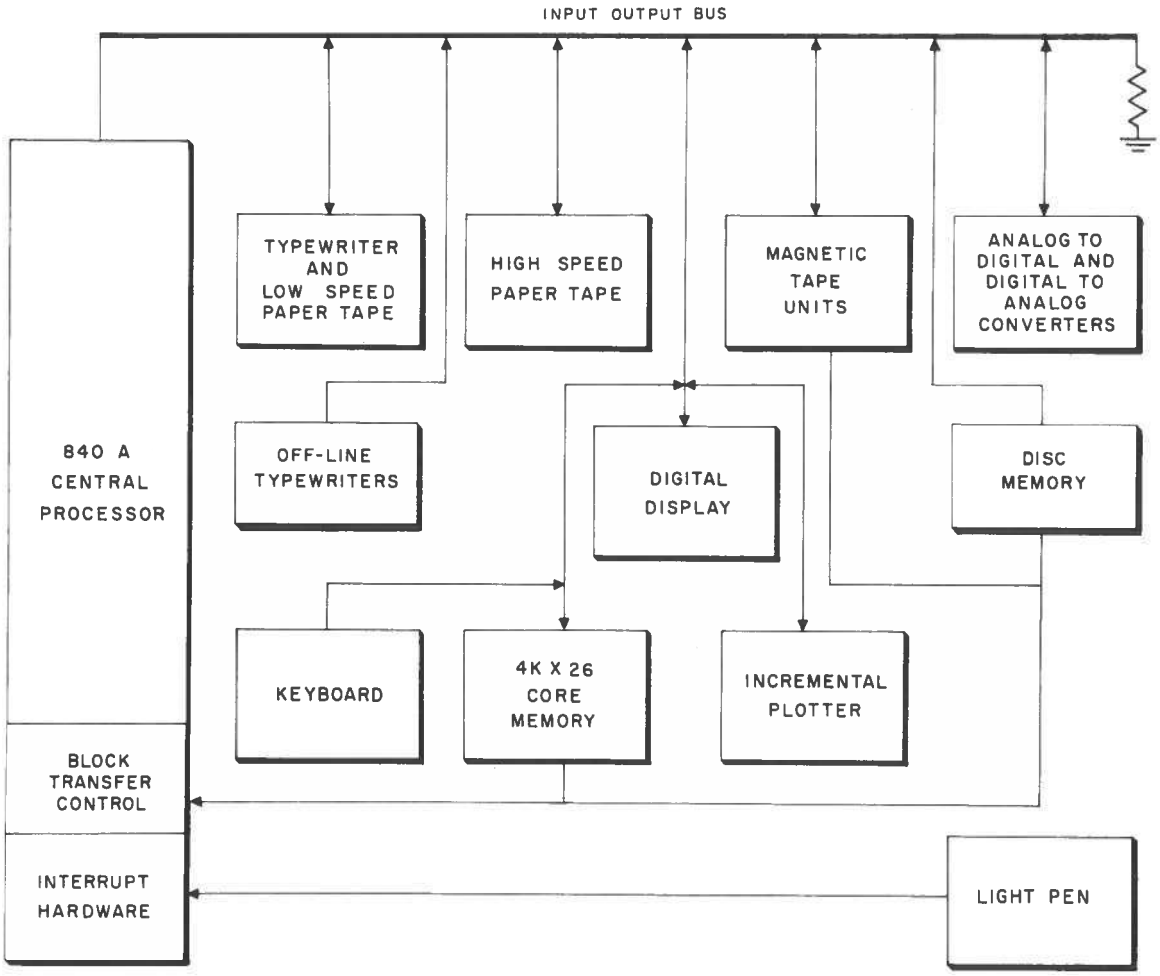

Fig. 1 Simplified block diagram of the 840A computer with its peripheral equipment

*Additional memory blocks can be added to expand the capacity to 32,768 words.

**Circuits can be added to make all 24 console switches into sense switches.

The various items of peripheral equipment which are shown in Fig. 1 are self-explanatory. Typewriters and paper-tape punches and readers are character-oriented devices which communicate with the central processor by means of 8-bit ASCII characters. The magnetic tape control unit converts the 6-bit-plus-parity characters stored on the 7-track magnetic tape to 24-bit words for transmission to the CPU. The digital display, disc memory system, and external core memory operate at the full 24-bit level. More detailed information about individual peripheral units will be given if necessary when describing programs designed to service them.

System Software

LOAD and DUMP Programs

In order for the computer to perform useful functions a program must be resident in its memory. A simple loader program which will bring other programs into core memory from external storage media such as paper tape, magnetic tape, or a disc storage file, is often called a BOOTSTRAP LOADER. To use the 840A computer when no programs exist in core, it is necessary to load the bootstrap loader instructions into consecutive memory locations using the console switches. The program is, therefore, as short as practicable. We will begin by describing the paper tape bootstrap loader which makes the computer behave like a simple machine for transmitting data from paper tape via the paper-tape reader into preassigned locations in core memory. The functions performed by the loader are:

1) Commands the paper-tape reader to read characters and transmit them to the computer on request.
2) Reads characters and checks if they are all zero. Continues until a non-zero character is found, thereby advancing the paper tape through the leader to the first character.
3) Assembles three successive 8-bit characters into a 24-bit word and stores it in memory. Continues to read, assemble, and store words until three successive zero characters are detected.
4) Either halts or branches to the first instruction of the program just loaded.

Notice that the format of the characters on paper tape consisted of three characters for each word in memory. In other words the data on tape are an exact replica of the data in core. Such a format is often called ABSOLUTE or CORE IMAGE format. A system program which loads and dumps in core image format is called an ABSOLUTE LOADER or an ABSOLUTE DUMP (Fig. 2). It is customary to distinguish between ABSOLUTE PAPER TAPE LOADER program and a BOOTSTRAP PAPER TAPE LOADER even though both

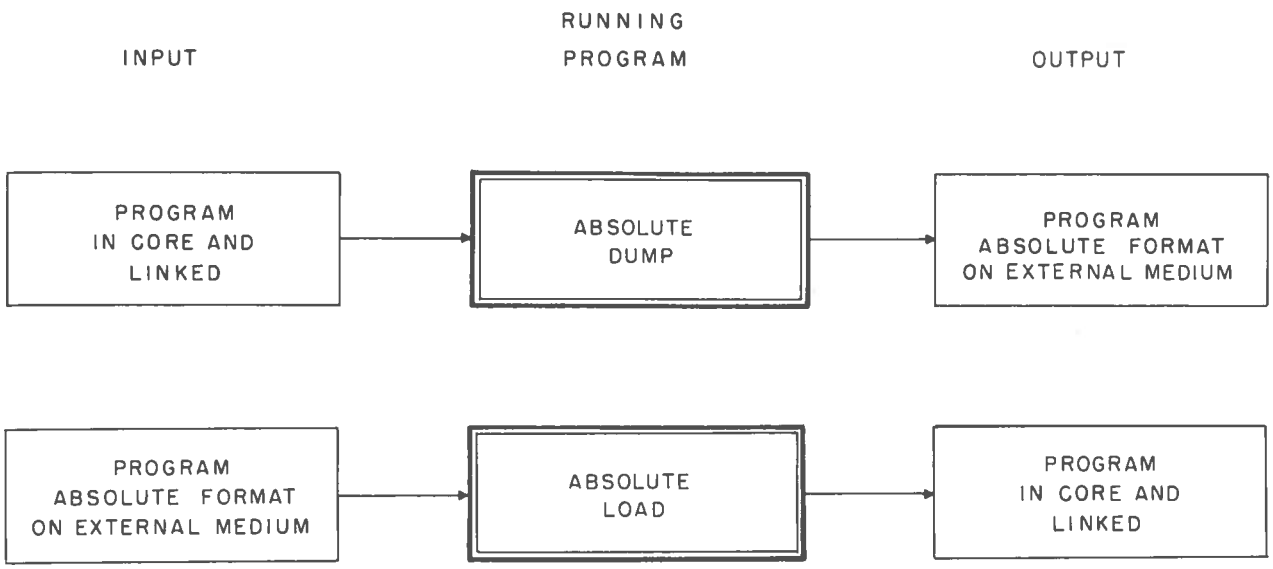

Fig. 2 Transfer of a program between the internal core memory and an external storage medium

use the same absolute format. The term ABSOLUTE LOADER is reserved for a program which, in addition to performing the functions of the bootstrap loader, also

1) Reads an initial block which contains the starting address and the word count indicating the size of the core area into which the transfer is to be made, and performs the storage function accordingly.
2) Reads paper-tape blocks of a fixed number of characters. For purposes of verification, each block contains the arithmetic sum (ignoring overflow) of all the words in the block. This CHECKSUM word is recomputed on loading and compared with the one read from the paper tape.
3) Performs some preassigned action such as typing an error message if a CHECKSUM error is found.

An ABSOLUTE PAPER TAPE DUMP program performs all the functions necessary to generate a paper tape which can be reloaded by the absolute loader.

Bootstrap loaders, absolute loaders, and absolute dump programs are also available for transfers between core and disc and for transfers between core and magnetic tape. They differ from the paper-tape programs basically only in the instructions which are concerned with the details of operating the particular device, and with the storage format on the external medium. The programs must also have the capability of locating the desired program on disc or

magnetic tape. Because of the high rate of transfer, both disc and magnetic tape programs make use of the BTC (block transfer control) in transferring core image blocks between the central processor and the peripheral unit.

The Unit Record Programs

The absolute load and dump programs described above define one format for storing data on an external medium – the absolute or core-image format. A second format which is also important is one which is suitable for storing alphanumeric characters. Examples are the ASCII format on paper tape which is produced by a standard ASR-33 teletypewriter, or the Hollerith format on punched cards produced by keying in a string of alphanumeric characters on a key punch. A set of programs which transmit data in this form is called the UNIT RECORD SYSTEM. Records which contain textual data are called HOLLERITH RECORDS, while those that contain a sequence of quarter-words in machine language form are called BINARY RECORDS.

The simplest example of a UNIT HOLLERITH RECORD is the record produced by typing a sequence of characters on a teletype machine (TTY) followed by a carriage return and line feed (CR/LF). The length of the record is not defined and may be one or more characters. Of course, on a typewriter, a record is limited to one line or approximately 72 characters.

On a punched card, a unit record is one card containing up to 80 columns of which each column represents one character.

On magnetic tape (7-track), a unit record is a block of characters stored in the IBM-BCD code. Each character is 6 bits plus parity. The length of a record is not restricted by hardware.

On disc, a unit record is one sector or 256 truncated ASCII characters. (A truncated ASCII character is one consisting of the 6 least significant bits of an 8-bit full ASCII character.)

In core memory as used by the 840A systems programs, a Hollerith record is a block of one or more 24-bit words in which each word contains four truncated ASCII characters.

A UNIT BINARY RECORD is a block of 24-bit words with no restriction on number and content. The code format of the characters, and the means used to terminate a block, of course depend on the storage medium as is the case for Hollerith records.

In core, a binary record consists of one or more computer words in sequential locations.

On paper tape, a binary record consists of 6-bit quarter words expanded to 8 bits so as to be acceptable ASCII characters. The only limitation on length is the requirement that each record must consist of a multiple of 4 characters so as to make up an integral number of 24-bit words.

On magnetic tape, the record consists of a block of 6-bit characters in binary format.

On disc, a binary record consists of one or more sectors. Each sector contains 64 24-bit words. The record consists of one or more words, and the remaining words necessary to complete the last sector are zeros.

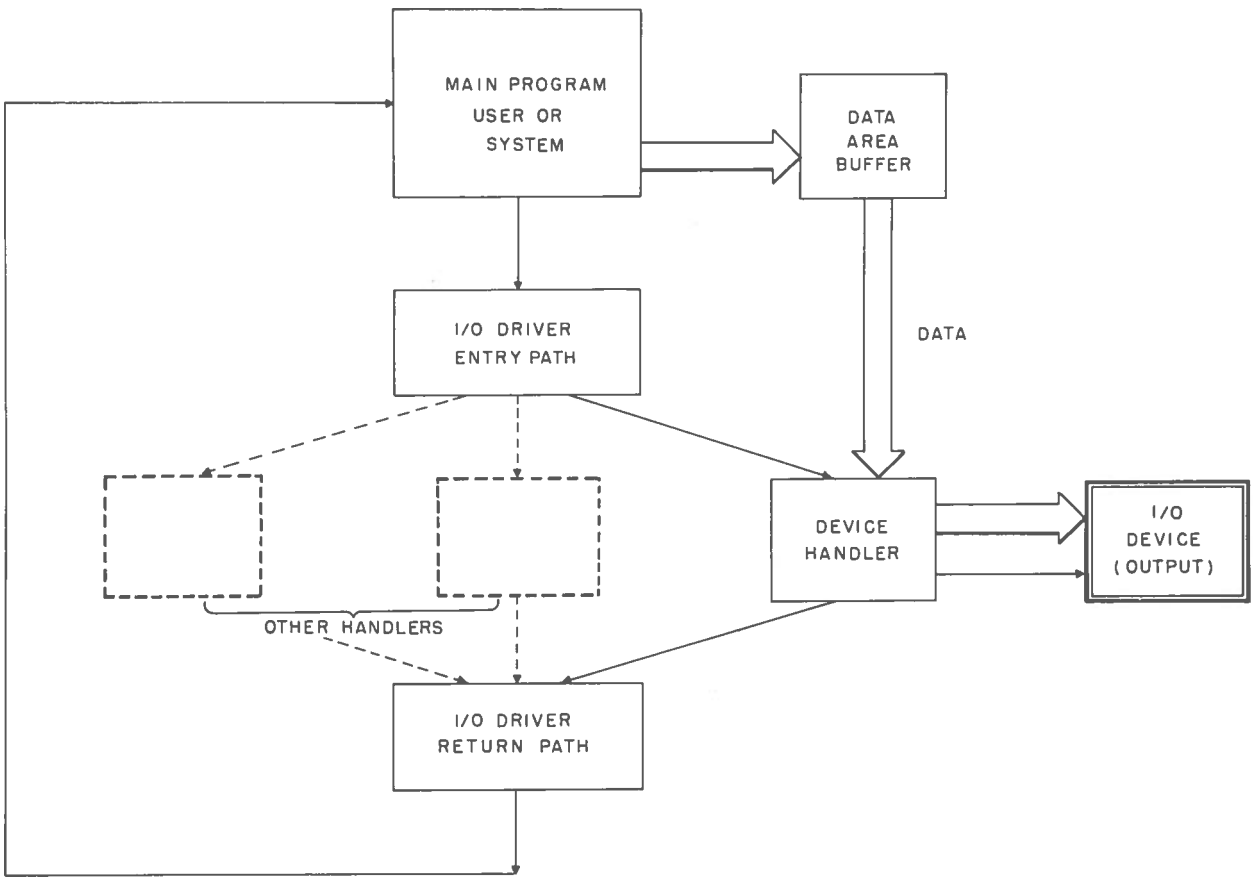

Fig. 3 The paths along which control and data are transferred from the main program to the desired device handlers

The unit record system is organized into a set of HANDLER subroutines and a pointer or a DRIVER program. The use of the unit record system removes the need for other programs to contain the detailed instructions necessary to transfer a unit record to an I/O device. Other programs can initiate the transfer of a record by executing a call to the driver which then selects the appropriate handler servicing the desired device (as shown schematically in Fig. 3). The detailed instructions for operating the devices and for code conversion (if necessary) are all contained in their respective handlers, thus making other programs independent of these devices.

In general each peripheral device has its own handler subroutine although devices with many functions in common such as high-speed paper-tape punch and reader and a teletype paper-tape punch and reader may be serviced by a single handler. The following functions are common to all handlers:

1) The ability to execute certain control tasks pertaining to the device which it services. Examples are: to make leader on paper tape, to rewind magnetic tape, to position heads on the disc, and to advance one or two lines at the top of a page on the typewriter.

2) The ability to inform the calling program of the status of a device, e.g. 'end-of-file' found on magnetic tape.

3) The ability to transfer, on call, a UNIT HOLLERITH RECORD between an area in core (specified by its beginning address and word count) and the storage medium used by the device serviced.

4) The ability to transfer on call a UNIT BINARY RECORD between an area in core (specified by its beginning address and word count) and the storage medium used by the device serviced.

In addition to these standard functions, handlers which service more than one peripheral unit are capable of distinguishing between these units by means of an identifying device number supplied by the calling program. This number is called the LOGICAL DEVICE NUMBER, or LOGICAL DIRECTORY NUMBER. It is also used by the DRIVER program to route calls to the appropriate handler as outlined below.

The main functions of the I/O DRIVER subroutine are to respond to calls for input–output transfer of unit records and, depending on the name by which it is called and logical device number supplied with the call, to route the request for a binary or Hollerith transfer to the appropriate handler subroutine.

The driver contains a table called the I/O DRIVER CALL TABLE which defines the logical device number for each peripheral unit. Assignment of logical device numbers is, therefore, merely a matter of making entries in the call table.

The type of record transferred is determined by the name by which the driver is called.

CALL H$WR requests a Hollerith record transfer
CALL B$WR requests a binary record transfer
CALL H$WL is a special peripheral positioning call which prepares the external device for the receipt or transmission of the first record.

The direction of transfer is determined by the sign of the logical device number. A negative number specifies core-to-peripheral or OUTPUT transfer, while a positive number specifies peripheral-to-core or INPUT transfer.

For some of the more complex peripheral devices such as magnetic tape or disc, a fourth calling method is necessary to provide a means of commanding and testing special functions of the device or its handler. A CALL H$WR, but with a negative integer in place of the beginning address will perform a special function determined by the integer supplied.

It is worthwhile at this point to discuss briefly mechanisms used to transfer information from a calling program to a subroutine. One method is to store the data in the arithmetic and/or index registers before calling the subroutine. A second method is to store the data in memory locations which are immediately before or immediately after the call instruction. The subroutine can then compute the addresses of the data with reference to its return address, and therefore can obtain it as required. A third method is to store all information to be transferred to subroutines in a preassigned common area in core with addresses known to both calling program and subroutine.

Only the first two methods are used in the unit record system. As an example, suppose it is desired to write on the high-speed paper-tape punch the contents of memory locations 1000 to 1002. The logical device number of the punch is 3. The sequence of instructions to call the driver subroutine (CALLING SEQUENCE) would be as follows:

```
LOAD A Accumulator with -3 (output logical device number)
CALL H$WR
1000        (beginning address)
3           (word count)
Next instruction
```

The I/O driver H$WR in turn generates a call to the paper-tape system handler and transmits the two parameters. The paper-tape handler turns on the punch, fetches each of the three

words, breaks each word into four characters and expands each character to 8 bits before causing it to be punched. After the 12 characters have been punched the carriage return and the line-feed characters are punched. The punch is left turned on, because further calls might follow and turning the punch on and off would cause unnecessary delays.

Programs to Convert from User Language to Machine Language

We have now described the absolute load-dump programs and the unit record programs. We have still not shown how a large program may have been generated and stored in core or on an external storage medium. A program may be loaded directly into core memory, one word at a time, by using the console switches. This method is not convenient for preparation of programs consisting of more than a few instructions. Apart from the clumsiness of entering sequences of machine words through switches, preparation of programs is difficult. All memory addresses must be explicitly specified by the programmer. Thus an address in a part of the program that has not yet been written must be specified explicitly, when any reference to that address is made (FORWARD REFERENCE).

A program written directly in machine language is ABSOLUTE. The program is consistent only if it is loaded in core into the addresses specified by the programmer. In order to relocate the program in core, it is necessary to modify the address portions of instructions to maintain consistency.

The basic assembler program relieves the programmer of the need to specify explicitly absolute addresses of core locations. Instructions written in the assembly language are essentially symbolic representations of machine instructions.

The conversion of a program written in assembly language, to in-core instructions is divided into two steps. In the first step, Hollerith records prepared manually by the programmer, and describing the desired sequence of operations, are translated by the symbolic assembler into unit binary records which contain the information on how the program should be built up in core memory (Fig. 4, upper). At this stage the program is RELOCATABLE, in that it can be loaded into any part of core.

In the second step the unit binary records (OBJECT form of the program) are accepted by a second program, called the RELOCATING LOADER. The relocating loader interprets the records as commands for building up instructions in core memory and generates absolute addresses which are internally consistent (Fig. 4, center). The program is now ready for execution (Fig. 4, lower).

Three basic facilities are provided by the symbolic assembler program:

1) The programmer may assign symbolic names to core locations and refer to these locations elsewhere in the program by their symbolic names. During

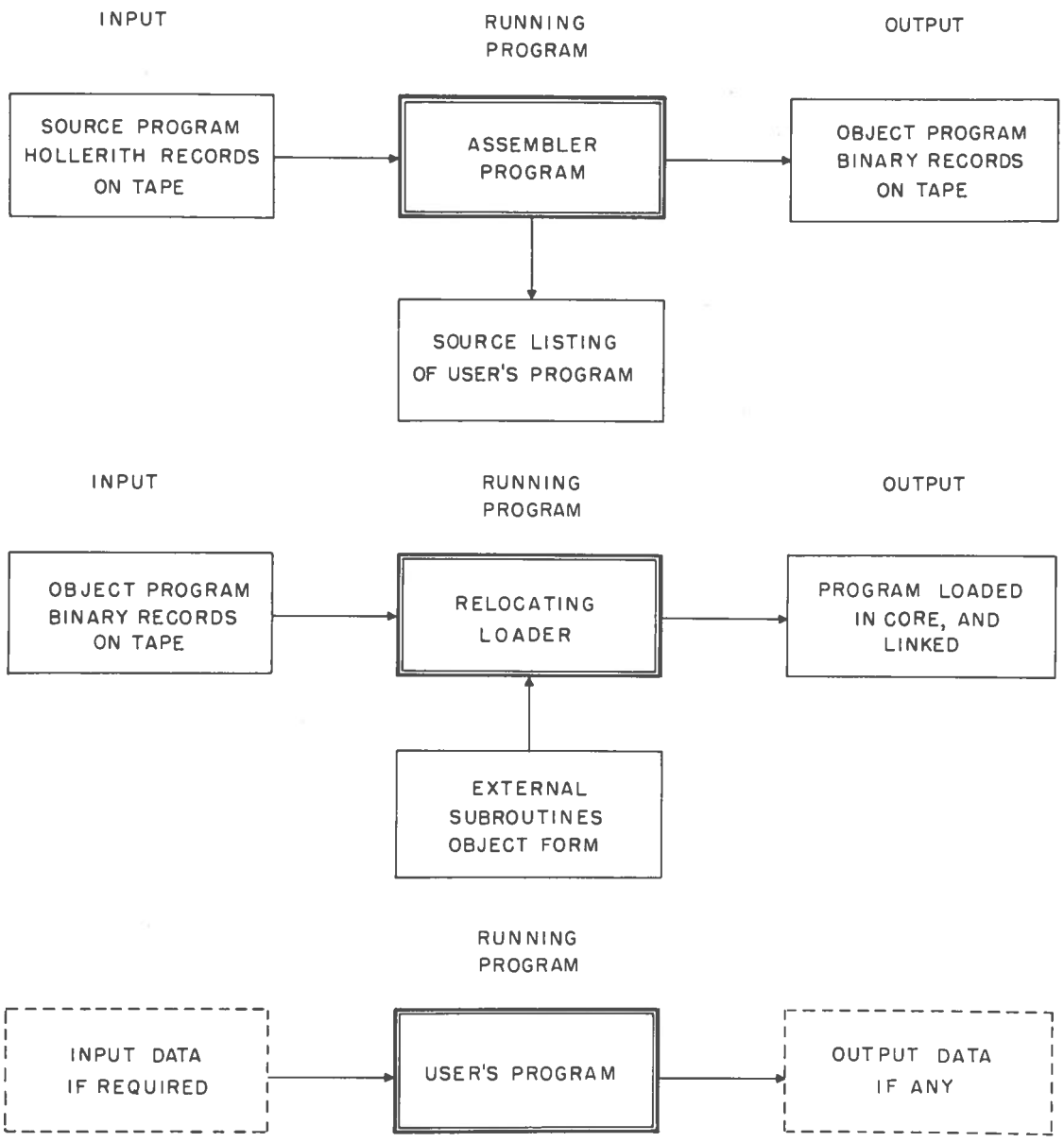

Fig. 4 Successive steps required to translate a program on paper tape into a running program in the core memory

translation the assembler program assigns addresses to the symbolic names. The addresses are made relative to the start of the program. The relocating loader makes the addresses absolute by adding the RELOCATION CONSTANT which is the starting address of the program being loaded.

2) The programmer can refer to an EXTERNAL address, that is, one which is not contained in the program being written but which, instead, is contained in a subroutine which is written and assembled separately.

The relocating loader program loads both the first program and the subroutine in which the external address is defined, computes the absolute value of the external address and inserts the correct value in the instruction in which the external reference was made. This function of the loader is called LINKING to subroutines. The reference to the external address may be either a subroutine call or an access of a data word. As a special case of the function of linking, the relocating loader links requests for input–output transfers of unit records, to the I/O driver which is already resident in core.

3) In the assembly language the programmer can request the assignment of core locations for data, and he can specify in several useful forms the initial values to be stored in the data locations.

The desired contents of the reserved locations can be specified as octal integers, decimal integers, fixed- or floating-point numbers, or sequences of alphanumeric characters, stored in the core locations as four characters per word, in the truncated ASCII code.

In addition to the object output, the assembler program generates and prints a SOURCE LISTING of the program which consists of the instructions written by the programmer together with the octal equivalents of the machine instructions that the loader will generate in core. Error messages are also shown on the listing. The addresses shown in the machine instructions on the source listing are relative to the beginning of the program just as they are on the object output.

A second programming language available on the 840A is the FORTRAN language. Fortran is suitable mainly for scientific and engineering computation, because a mathematical equation can be written with little change as one instruction or statement in Fortran. Typically, a complex mathematical problem can be written in relatively few statements. On translation by the COMPILER program, the few source statements result in many machine instructions in object form. Thus, Fortran is an efficient language for describing mathematical problems, without being concerned with the actual details of the computer operation.

Because system programs are tightly coupled to the computer operation and its structure, these programs are usually not written in Fortran, but rather in assembly language.

Supervisor Programs

We now have described the means of generating programs and of loading them into the core memory of the computer. Each step in the process has required the execution of a SYSTEM program, which itself may have been loaded into core before execution.

In this section we describe part of the system software which is designed to help the operator manipulate other programs. This software may be called a SUPERVISOR, or an EXECUTIVE system. It may also be spoken of as a set of HIGHER LEVEL system programs, because it is used to provide communication and control between the user and other elements of the system software.

In the following paragraphs we will discuss the need for executive programs and their important functions. It is desirable that some system programs which are used regularly should remain in core at all times in order to avoid unnecessary loading (and by implication requiring that the loader stay in core). Such programs are described as CORE RESIDENT SYSTEM PROGRAMS. Other programs which may not be as important or which may be too large to keep in core, may be stored on disc in absolute core-image form, so that they can be transferred rapidly to core when needed. These programs are called DISC RESIDENT. Still other programs of lower priority may be stored on paper tape or magnetic tape.

The decision as to how the storage of system programs should be allocated between core, disc, magnetic tape, and paper tape depends on many factors.

Obviously, the more core memory available to the system, the more programs should remain in core. The choice of core resident programs depends on how the system is being used. If the primary use is program development, then assemblers, compilers, load and dump, and debugging programs should be resident. If the primary use is computation, loaders and diagnostic programs should be resident. If the use is industrial process control, business data processing, or scientific data processing, then the most important programs are the processing programs and the associated input–output data handlers.

In a research environment such as ours, the program development and input/output programs are important.

At this point it is worthwhile to list briefly the steps which must be taken by a user in a non-executive system in order to have his program executed. We will assume all necessary system programs are core-resident. The user must:

1) Prepare SOURCE PROGRAMS or sequences of SOURCE STATEMENTS as Hollerith records on cards or paper tape.
2) Place the source program medium (e.g., paper tape) in the input device.
3) Put the starting address of the assembler or compiler program in the program counter of the CPU.
4) Supply (through control switches on the computer console) the logical numbers of the devices to be used for source input, object output, and source listing output.
5) Supply (through control switches) the operating parameters of the assembler or compiler program (e.g., number of passes).

6) Start the computer running to execute the assembler or compiler program.

7) Transfer the object programs produced by the above procedure to desired input devices.

8) Load the starting address of the relocating loader into the CPU program counter.

9) Supply through console control switches, the logical device numbers of the input devices, and the starting address at which the programs are to be loaded.

10) Start the computer running to execute the relocating loader.

11) Set the program counter of the CPU to the starting address of the program just loaded.

12) Supply the user program with any parameters it requires by presetting registers and by setting control and/or sense switches.

13) Start the computer running to execute the user program.

Notice that the computer was halted between each processing operation to allow the operator to prepare initial conditions and running conditions for the next phase. It is often desirable that the computer run continuously so that it will be able to respond at any time to 'interrupt' requests. This is particularly true if the computer is being shared by several users whose programs are being interleaved by the priority interrupt hardware.

In the environment in which this computer operates, executive or supervisory programs are used to accomplish the following tasks.

1) To keep the computer running between each phase in the processing operation

2) To supply initial conditions and running conditions to each phase of the operation without halting

3) To keep track of the core addresses and disc addresses of system programs so that they may be loaded and executed without user intervention

4) To accept operator commands to transfer records or files between peripheral devices or between peripheral devices and core

5) To accept operator commands to initialize or position peripheral devices

6) To 'control' automatically some peripheral devices after use (e.g., to turn off a paper tape punch)

7) To accept operator commands defining the status of the interrupt system hardware and the program protect hardware

8) To manage tables in the I/O driver which allocate peripheral devices to specific tasks.

These functions are not all performed by a single program. Some are allocated to the I/O driver because it is core resident, some are allocated to disc resident programs which are only brought into core to control specific devices when required.

Conclusion

There are many types of system programs which are not discussed in this report. Software associated with disc management, graphical display management, diagnostics, debugging, and even details of the various executive programs, are included in a series of reports which are being published separately. It has been our intention to provide background to readers not familiar with system software, to enable them to make the most use of the detailed program descriptions available in other reports.

Acknowledgment

The authors would like to point out that, for the most part, the software described in this report was supplied by the computer manufacturer – Systems Engineering Laboratories Inc., of Fort Lauderdale, Florida, and that minor changes suggested by the company's representatives have been made in the report.