RAKSHIT, ANANDA

MICROCOMPUTER BASED TRUCK DISPATCHING SYSTEM - OVERALL SYSTEM MANAGEMENT

THE UNIVERSITY OF ARIZONA M.S. 1984

MICROCOMPUTER BASED TRUCK DISPATCHING SYSTEM - OVERALL SYSTEM MANAGEMENT

by

Ananda Rakshit

A Thesis Submitted to the Faculty of the

DEPARTMENT OF MINING AND GEOLOGICAL ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE
WITH A MAJOR IN MINING ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 8 4

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his judgement the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: _____

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Young C. Kim — Aug 20, 1984

YOUNG C. KIM
Professor of Mining Engineering

Date

ACKNOWLEDGEMENTS

At the outset, I would like to express my sincere thanks to Dr. Young C. Kim, my thesis director, for his guidance and support through all phases of the research.

I thank William E. Kolb, my co-researcher for this research project, for his cooperation and valued suggestions.

I would like to thank Dr. Charles E. Glass, Head, Department of Mining and Geological Engineering and Dr. Jaak J.K. Daemen for spending their valuable time in evaluating this thesis.

Finally, my thanks to Wanda Edwards, Secretary of the Department of Mining and Geological Engineering, for her assistance with prompt paperwork during acquisition of computer hardware and software.

TABLE OF CONTENTS

TABLE OF CONTENTS--Continued

TABLE OF CONTENTS--Continued

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS--Continued

LIST OF TABLES

ABSTRACT

Computer aided truck dispatching for an open-pit mine based on a microcomputer is viable due to the advances made in the field of microprocessors and microcomputers in recent years. Principal benefit of such a system is the low initial expenditure.

Research to develop the framework for a microcomputer based system has been completed. This system, based on an IBM Personal Computer, has been successfully tested with simulated data.

This thesis is concerned with the research done in developing the support software and hardware configuration for the system. Areas of work include time management, sound effects, handling of video and printer output, setting up the testing system and overall integration with the code implementing dispatching algorithm.

Exploring and surveying IBM Personal Computer hardware and software to determine suitability for the dispatching system constituted a major portion of the research.

CHAPTER 1

GENERAL INTRODUCTION

1.1 Introduction

The benefits of a computer based automatic truck dispatching system has been known for quite some time. Very sophisticated systems have been developed and installed in openpit mines. Considerable gains in productivity, safety and maintenance of operating statistics have resulted from the use of successful automatic truck dispatching systems. One of the noteworthy systems is based on a Digital Equipment Corporation (DEC) VAX 11/780, other associated peripheral equipment and about 50,000 lines of documented source code (White, Arnold and Clevenger 1982). The initial investment for installing such a system is, however, upwards of one million dollar and is usually prohibitive for small to medium scale mining operations.

With advances in the field of microcomputer hardware and software, it is now possible to develop a dispatching system based on a microcomputer. A microcomputer based dispatching system would cost only a fraction of that of a full fledged system. This can be

done without a considerable loss of functional capability and more importantly, would be affordable by a small to medium scale mining operation.

The objectives of this research are to develop the basic configuration of a microcomputer based truck dispatching system and clearly demonstrate the feasibility of such a system. The research is divided into two parts :

1. Development of dispatching philosophy and algorithm and implementation them through source code in FORTRAN.

2. Selection and installation of computer hardware, peripherals, marketed software and development of software in assembly language and FORTRAN to accomplish special tasks.

The first part of the research has been done by William E. Kolb. The second part of the research has been my responsibility and is the subject matter of this thesis. Concepts of microcomputer and microprocessor architechture will constitute a part of this thesis because these concepts have been very important in developing the dispatching system.

1.2 Basics of proposed truck dispatching system

In a typical open pit mine there are shovel sites, crusher sites, and dump sites during a period of

operation. Material is loaded into trucks by shovels and hauled to either the crusher or dump sites depending upon whether it is ore or waste. The dispatching problem is to route the trucks to the right crusher, right dump or right shovel such that certain objectives are optimized or satisfied. These objectives can be one or more of the following :

1. Minimize shovel idle time.
2. Minimize truck idle time.
3. Minimize crusher idle time.
4. Minimize the variance of grade of ore fed to the crusher.

In the proposed dispatching system, signposts located at strategic points around the pit will monitor truck movements as they pass by and will send the information to the computer. Each truck will be identified by a unique identification number. Signposts will be equipped with microprocessors and transmitters. Designing the signposts and developing the software for them is outside the scope of current work. At present signpost signals will be replaced by signals from another computer. A schematic diagram of the proposed system is shown in figure 1.

Upon receiving data from signposts, the computer will process them and will assign trucks to their proper

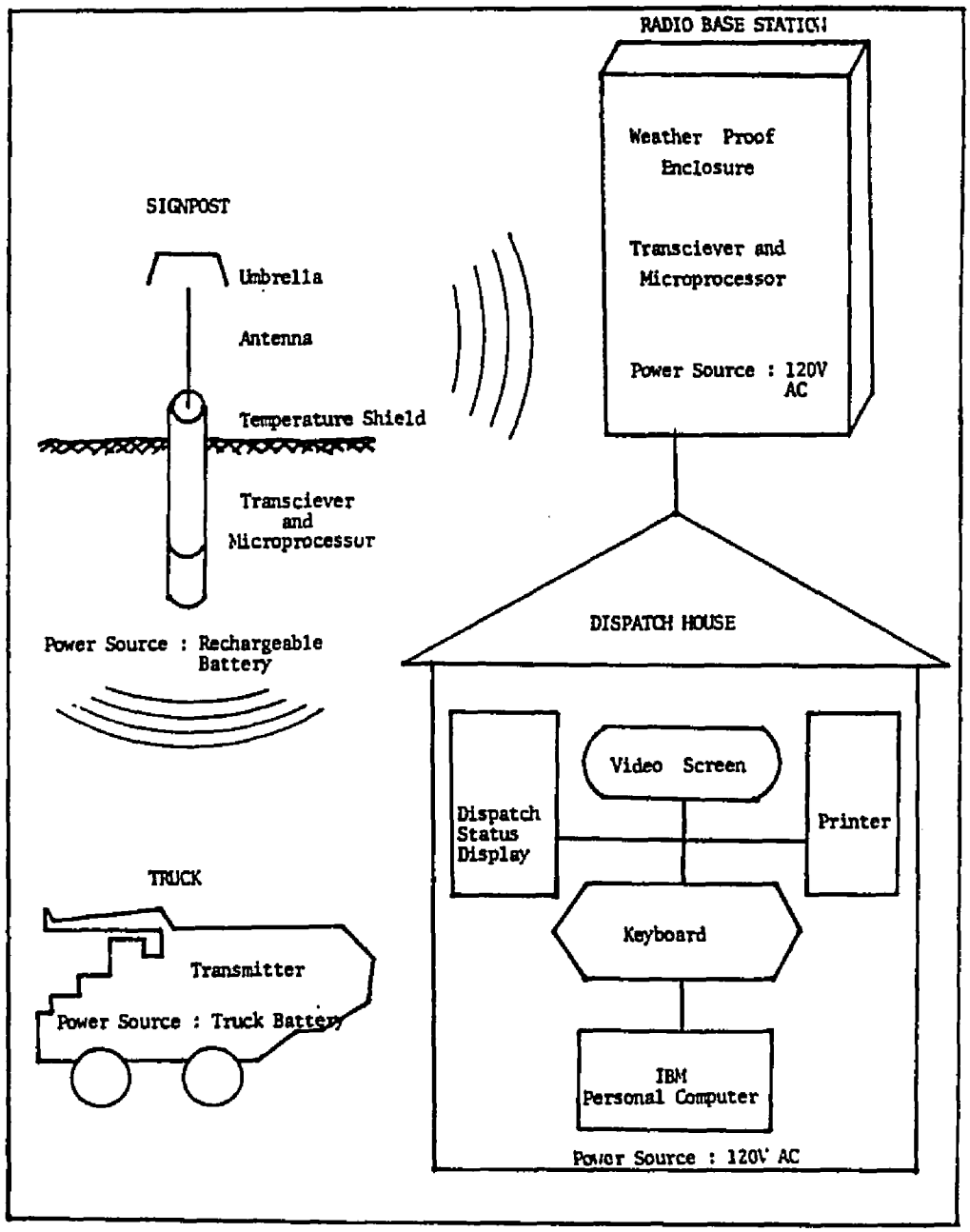

Figure 1. Basic Configuration of Dispatch System (Kim 1983)

destinations. This assignment will be based on a heuristic algorithm. Assignments will appear on the main video monitor as well as on an auxiliary monitor showing the status of trucks and shovels in the form of a dispatch board. The computer will interact with the dispatcher wherever necessary. The computer assignment can be overridden by the dispatcher in case of special circumstances. In such a case, the dispatcher will have to enter his assignment into the computer. Production and operating statistics will be automatically recorded by the computer and will be printed out in the form of reports. The dispatching system will thus achieve the following :

1. Dynamically dispatch each truck after each load, thereby increasing truck productivity.

2. Automatically record and store accurate production statistics.

1.3 State-of-the-art in microprocessors and microcomputers

A microprocessor is the central or peripheral processing unit of a computer built ona single tiny semiconductor chip. It is a LSI (large scale integration) or VLSI (very large scale integration) circuit fabricated on a piece of silicon about 5 mm square and .5 mm thick (Ralston and Relly 1983).

The first commercially available microprocessor was the Intel 4004. It was marketed in 1971 and contained about 1000 transistors. Each register was four bit wide. From 1972 through 1977, microprocessors with eight bit wide registers came into the market. The most widely used of these are Intel 8080A, Motorola 6800 and 6502, and Zilog Z80 (Ralston and Relly 1983). Many microcomputers are based on these microprocessors. For example, Apple II series computers are based on 6502 microprocessors and Tandy TRS-80 series computers are based on Z80 microprocessor (Sargent and Shoemaker 1982).

More complex microprocessors, like Intel 8086, Zilog Z8000, and Motorola MC68000, appeared in the market from 1978 onwards. Intel 8086 has 16-bit registers and Motorola 68000 has 32-bit registers. The Zilog Z8000 has 16-bit registers, but, it can perform 32-bit arithmetic by grouping two 16-bit registers (Morgan and Waite 1982). Microcomputers of today are based on these microprocessors. The IBM-PC, PC-XT and PC-jr. are based on 8088 which is an eight bit version of 8086. Apple LISA and McINTOSH are based on MC68000 microprocessor (Williams 1984). The term microcomputer will be used interchangeably with the term personal computer from this point onwards in this thesis.

1.4 Microcomputer selection for dispatching system

The task of selecting a suitable microcomputer was quite difficult . The first criterion in such a selection was the ability to use FORTRAN. It was decided to use FORTRAN as the high level language for truck dispatching system because of its continuing popularity as a scientific language. The IBM Personal Computer XT appeared to be the most logical choice because of various reasons.

The IBM-XT comes with a built-in 10 megabyte hard disk to serve the need of storing large programs and data during the development stage of the system in particular. A good FORTRAN compiler is available to do the necessary programming. IBM-PC being the most popular personal computer, dispatching programs written for it are more likely to be easily relocatable. Furthermore, many personal computers in the market are IBM compatibles or look alikes. Besides hard disk storage, primary storage in Random Access Memory can go upto 640 Kilobytes. Another very important reason is the large amount of technical literature written on the IBM-PC. This makes the computer easier to understand and work with, especially at a level lower than that of the high level language. Finally, different kinds of expansion boards and software are available which can dramatically improve the performance of the basic unit.

1.5 Outline of overall system design requirements

Designing the dispatch system involved working in the areas listed below. Research done in each of these areas are discussed in detail in the following chapters of this thesis.

1. Overall management of the system including system flow and input/output through IBM Disk Operating System Batch Commands. A major part of this constituted developing the logic for transfer of control between different programs.

2. Displaying and updating the status of trucks and shovels at regular intervals on a dispatch board.

3. Developing communications software in assembly language for simulated system testing.

4. Maintaining time-of-the-day and date and making them available to Fortran programs through assembly language routines.

5. Developing report generator program to produce reports of production and operating statistics at the end of a shift.

6. Finding ways and means to improve speed of operation to maintain a real time status.

7. Producing sound effects with error messages and to alert the dispatcher at the occurence of important events.

CHAPTER 2

PREVIOUS WORK ON TRUCK DISPATCHING SYSTEMS

2.1 Introduction

Over the past ten years or so, different types of dispatching systems have been developed and implemented. Operating experiences with these systems have been published in a number of papers and articles (Mueller 1977; Hempenstall and Hill 1980; White et. al. 1982). All these systems are based on dynamic assignments. Trucks are assigned to a shovel at the end of each load. Assignments can be based on a goal to minimize truck, shovel, or crusher idle time, or to minimize the variance in the grade of ore fed to the crusher. Dynamic assignments are made using one of the three methods given below.

1. Simple radio dispatching: The dispatcher has radio communication with the trucks and is located at a strategic point in the pit such that the entire pit is visible. He follows traffic patterns and assigns trucks to shovel via radio. Assignments can be highly suboptimal, and for a large pit finding a suitable

observation point might be impossible. This system was, in fact, used more than ten years back.

2. Dispatch board aided radio dispatching: The dispatcher is kept informed about truck locations through radio calls from truck operators. Actual pit positions are matched with the dispatch board, an analog computer. Assignments are manual and heuristic and are relayed back to the truck operator over radio.

3. Computer based dispatching: Truck and shovel assignments are determined by the computer. In this category there are many variations. In some, only the dispatch board and assignment method is computerized. Other operations are manual and over radio. In another type, computer assumes a greater role and receives truck location information directly from the operator without going through the dispatcher. In the third type, everything is automatic and even truck locations are detected by special purpose devices and relayed to the computer. In this case truck operator input is completely eliminated.

Simple radio dispatching gives the poorest results and will not be discussed further. Two systems, viz., the dispatch board system at Cyprus Pima and the Modular Mining Systems Inc. dispatch system will be discussed in succession. These systems are good

representations of the work performed on truck dispatching systems to date.

2.2 Cyprus Pima dispatch board system

This dispatch system is based on an analog computer or the dispatch board. It is used to aid the dispatcher in making truck assignments (Mueller 1977). The dispatch board consists of four components; truck markers, shovel blocks, plexiglass slides, and the board itself. The board is a 30" x 24" x 3" corrugated table shown in figure 2.

Each truck has a corresponding marker which is color coded according to truck speeds. Black numbers represent slow trucks, red numbers represent medium speed trucks, and yellow numbers represent fast trucks. Each shovel has a block of wood with its number on it. These blocks are marked off in segments which represent five minute time intervals. Black, red and yellow stickers are used with shovel blocks. The stickers are placed on each shovel block so that their location indicates the number of five minute periods normally required for a truck to complete one full haul cycle. The stickers have to be changed only when a shovel moves to a new location. The plexiglass slides are marked off in segments representing

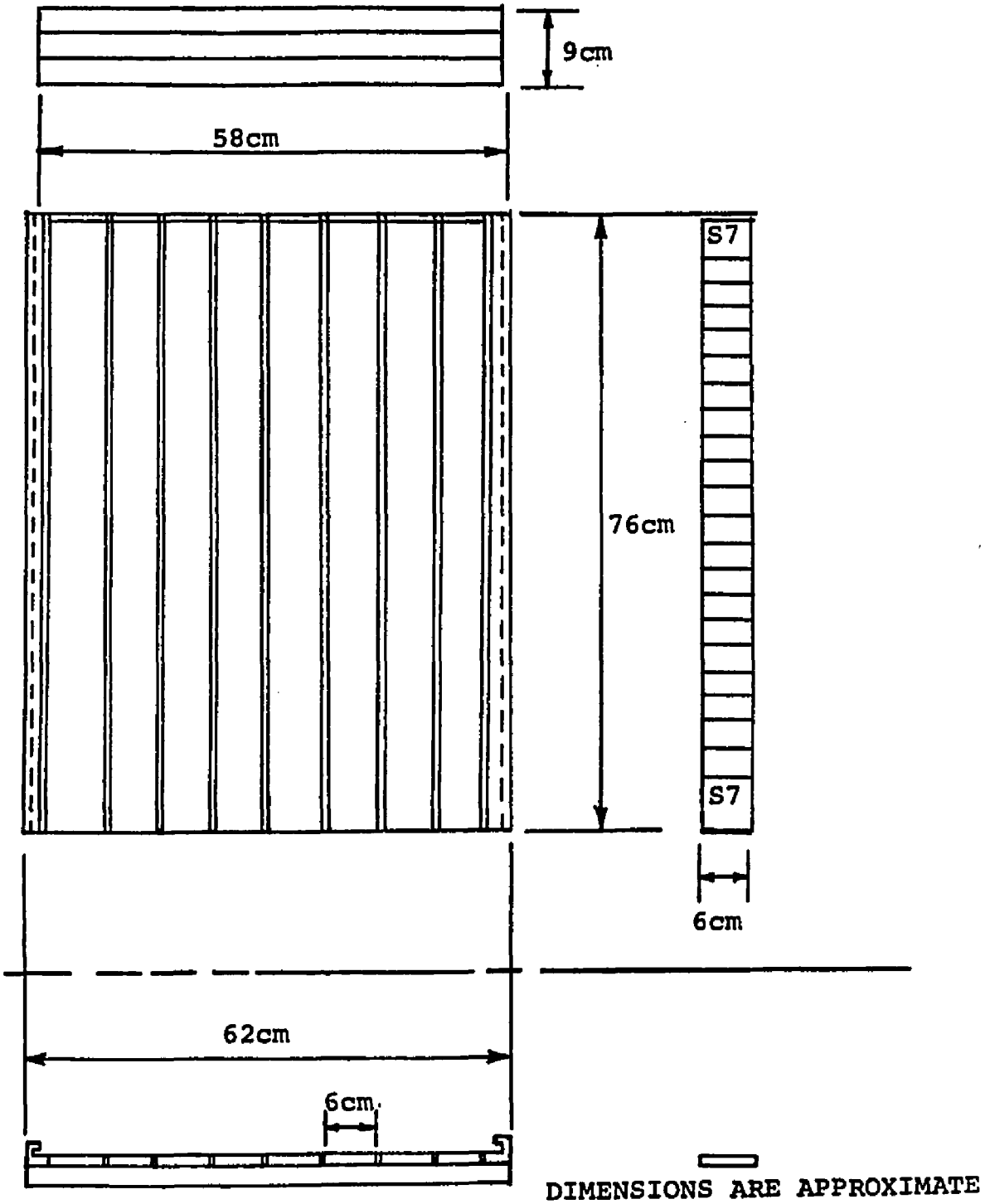

Figure 2. The Cyprus Pima Dispatch Board Configuration
(From Mueller 1977)

five minute time intervals. The numbers on the board down the sides of these slides refer to the time in minutes.

The dispatch board operates as follows. First, blocks are slipped into the board, one for each operating shovel. When a truck calls in for assignment, the dispatcher finds its corresponding marker. A foreman is notified if a truck calls in much later than expected. The dispatcher assigns the truck to a shovel and places the truck marker over the appropriate location on the shovel block, corresponding to the cycle time from that shovel. When time advances, the truck marker is slid up the board every five minutes. As the truck reaches a call point, the operator calls the dispatcher and the process repeats itself. The dispatch board helps the dispatcher make good assignments because it shows how many five minute increments exist between present time and the time that the last truck was assigned to a shovel. This is shown in figure 3. A truck will be assigned to the shovel which has the most increments between the last truck assigned to it and the present time.

This system is easy to operate and it boosted the productivity at the Cyprus Pima mine. However, it requires too much dispatcher attention and competence. It does not record any production statistics and cannot detect misrouted trucks. Also, the dispatcher becomes a

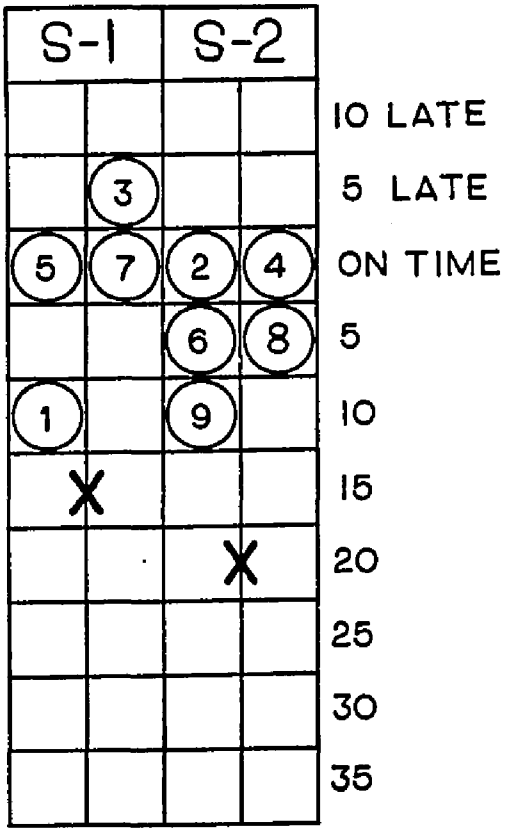

Figure 3. Assignments on the Cyprus Pima Dispatch Board (From Mueller 1977)

S-1 & S-2 represent two shovels

Circles represent trucks

X marks are cycle time for shovels

very busy person, especially when more than one truck operator calls in one after the other. Assignments are heuristic and need not satisfy the broad objectives in a mine.

2.3 Modular Mining Systems Inc. dispatch system

This computer based dispatching system has been in operation in Tyrone mine in New Mexico since May, 1980 (White et. al. 1982) The system is based on a Digital Equipment Corporation (DEC) VAX 11/780 computer with a virtual memory system. This central processing unit is located in the mine general office building and linked, via underground cables, to a dispatch tower some two miles away. The hardware configuration is shown in figure 4. The following devices are located in the dispatch tower:

1. Two 9600 baud video terminals. The first outputs computer generated transactions and error messages to the dispatcher. The second terminal is used by the dispatcher to access various system utilities and for status displays.

2. One diablo printer that keeps a current log of the shift. A production report is automatically printed at the end of a shift.

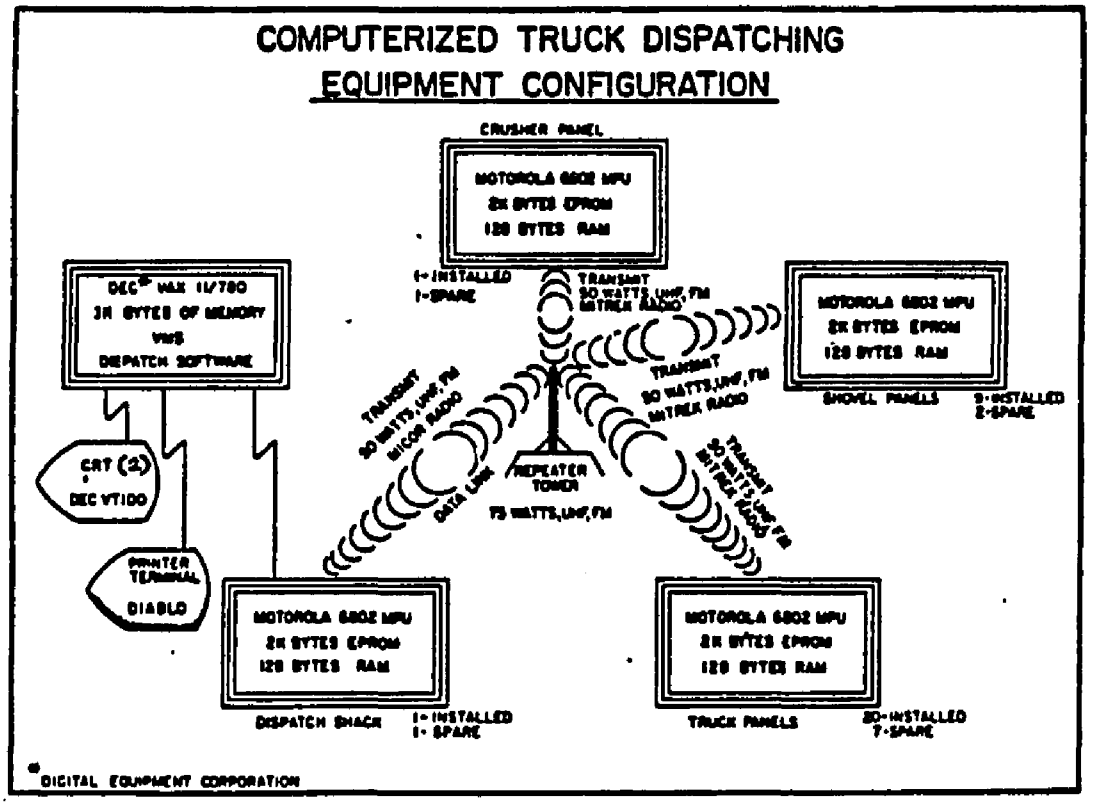

Figure 4. Modular Mining System Inc. Dispatch Hardware Configuration (From White, Arnold and Clevenger 1982)

3. One motorola 6802 microprocessor based master panel that links all field communications with the central processing unit.

Each truck, shovel and crusher is equipped with a 6802 based control console and a Motorola UHF-FM radio. They communicate with the master panel through a repeater tower. The console consists of a series of pushbuttons corresponding to some state of operation such as loading, waiting etc. and certain locations such as shovels, crushers, dumps etc. The right button must be pressed by the operator at the correct time. The consoles can also receive message from the computer. These messages are 8-bytes long and transmitted at 600 baud (White et. al. 1982).

The system is modular and structured in design. The system software consists of about 50,000 lines of commented source code and system command procedures written in Fortran V. The system is divided into PITSET and UTILITY sub-systems. The PITSET subsystem is responsible for the following principal functions:

1. Initial pit layout and pit reconfiguration.

2. Set up of a proposed pit layout for simulation and production forecasting.

3. Restoration of a shared global data base from disk.

4. Modification of shovel data base parameters.

PITSET allows engineers to alter the data base for simulation studies to aid in mine planning, dump design etc. without affecting the actual data base. The UTILITY subsystem is composed of 18 subprograms. They provide status reports about productivity and system state. They also provide production reports and short reports for the manager.

The dispatching system uses both the subsystems to select the optimal route using a linear programming model. The objective is to obtain maximum production from the system (White et. al. 1982).

The current pit situation is monitored by the computer using input provided by operators from truck control panels. An operator requests for new assignment after a load is dumped at a crusher or dump by pressing a designated button on the console. The computer sends in the next assignment on the LED panel of the console where it remains until the truck reaches its destination. This helps avoid misrouting of trucks. After loading a truck the shovel operator reports the material type by pressing the correct button. The proper dump or crusher

destination then appears on the LED console of the truck. Other data or events which are input from control panels of equipment and which are essential for the operation of the system includes arrival at destinations, loading at shovels, delay time, equipment breakdown, back to service after going down and gallons of fuel received by trucks.

The system manages all truck assignments, including those to maintenance shops, fuel docks and tie down areas. Routine functions of dispatching being carried out by the system, the dispatcher is able to better attend to equipment and other non routine problems during the shift (White et. al. 1982).

The operating statistics of the dispatch system at Tyrone mine shows significant gains. It fostered truck operating time per shift from about 350 minutes to about 400 minutes or approximately 14 percent. Productivity gains were almost directly proportional to the increase in truck operating time. The benefits of the dispatching system include the following (White et. al. 1982):

1. Reduction of empty miles traveled due to optimal route selection.

2. Reduction of shovel and truck idle time.

3. Better response to disturbances in the pit like shovel breakdowns, delays and moves.

4. Reduction of shift start and tie-down time windows.

5. Elimination of production differences resulting from various manual dispatching techniques.

CHAPTER 3

CONCEPTS OF 8088 BASED MICROCOMPUTER

3.1 General

The IBM personal computer is based on the Intel 8088 microprocessor. The 8088 forms the central processing unit of the microcomputer. A microcomputer based on the 8088 has a bus oriented architecture. A main bus leads out from the 8088 through intermediate support microprocessors to which all the external devices are attached. Figure 5 shows the typical configuration of a 8088 based computer. All the peripheral devices like the keyboard, diskdrive and even the memory are connected to the main bus which is made up of subbuses. The main bus connects to the 8088 through several other special purpose microprocessors. The peripheral devices are connected to the bus via respective controllers which are sometimes called interface boards (Morgan and Waite 1982).

3.2 Subbuses

The main bus can be divided into 1) power, 2) control, 3) address, and 4) data subbuses. The power subbus carries power to various devices and controllers

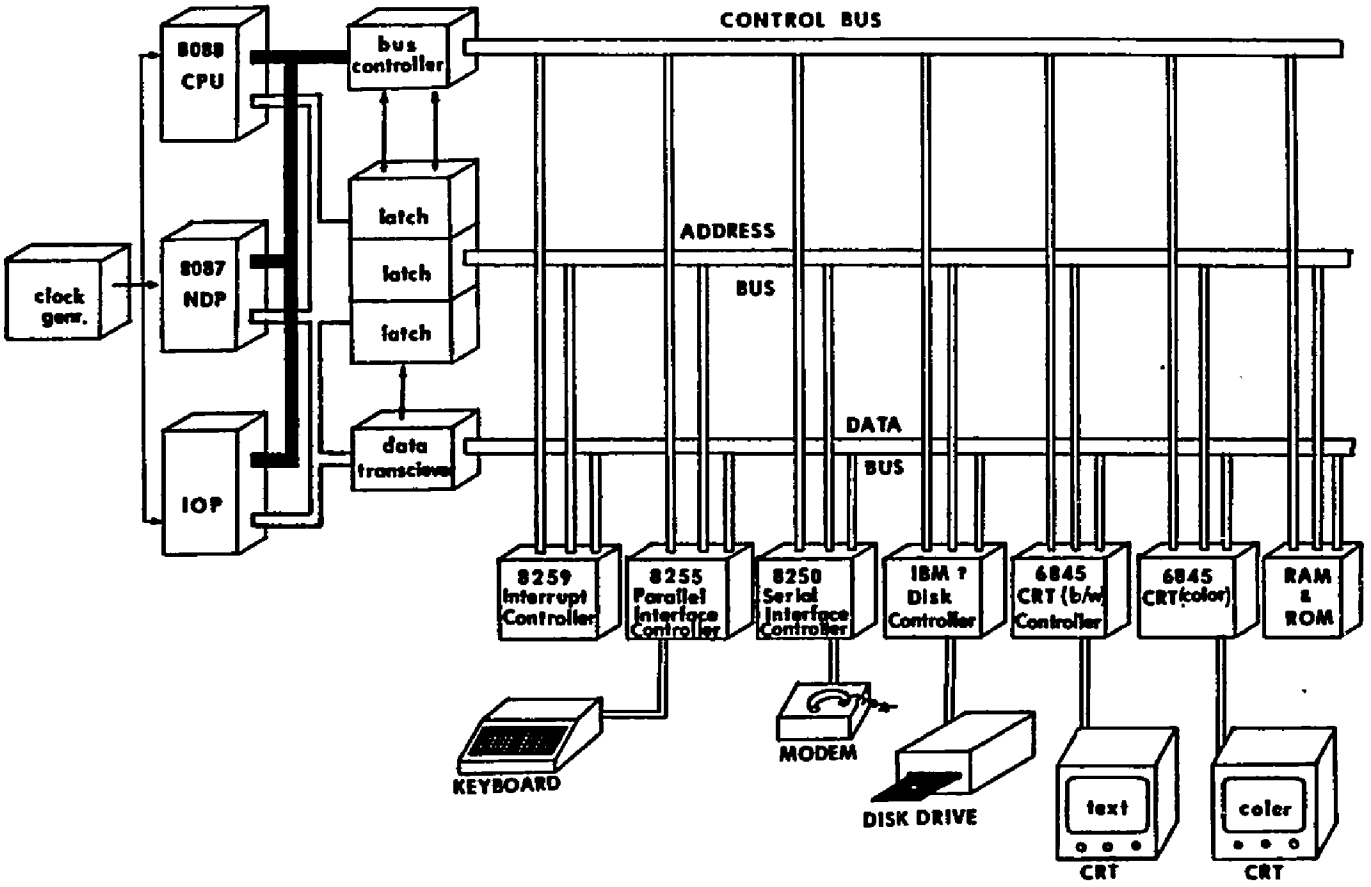

Figure 5. Schematic Diagram of 8088 Based Computer
(From Morgan and Waite 1982)

in the computer from the power supply. The control subbus carries information like signals from the clock chip, system interrupts etc. The address subbus carries information necessary to locate a particular memory cell. An address is a binary number composed of binary digits. The 8088 can support an address bus with 20 signal lines. Each signal line can carry a single binary digit and therefore, a binary number composed of 20 binary digits can be carried by the address bus at a time. With 20 digits there can be 2^{20} unique binary numbers and 2^{20} unique memory locations can be addressed.

Memory in a computer is generally organized in bytes, i.e., one byte occupies a single memory location. The 8088, therefore, has the capability to address 2^{20} bytes or 1 megabyte. The data subbus carries the actual data within the computer. There are eight signal lines in the data bus. Therefore, each unit of data can range from 0 to 2^8 in value. Since the data subbus can carry eight bits of data at a time , it is sometimes called a 8-bit microprocessor.

3.3 The memory

The main memory, as mentioned earlier is organized in bytes. This organization is logical and has nothing to do with how the memory is physically organized. It is tied

to the system bus just like any other device. The memory is supported by some logic chips which ensure proper data transfer between the memory and other devices including the 8088 (Willen and Krantz 1983). There are two logically different types of memory used in personal computers. Random Access Memory (RAM) or Read/Write memory and Read Only Memory (ROM). Data and programs can be written to and read from RAM memory, whereas, ROM memory generally stores important programs which can only be read in. ROM memory is written by a special process called burning and cannot be overwritten normally. RAM is logically mapped or located at the lowest address (i.e. zero) and ROM is mapped to the highest address within the possible 1 megabyte address space. This is shown in figure 6.

3.4 Memory units and data types :

The smallest unit of memory is a bit , which stands for a binary digit. A byte consists of eight bits and half a byte or four bits is called a nibble. A word in 8088 consists of two bytes or sixteen bits. The double word is four bytes and the quadruple or quad word is eight bytes. Finally, there is a 80-bit word made up of 80 bits. The larger storage units are used for storing floating point numbers. Floating point numbers are stored

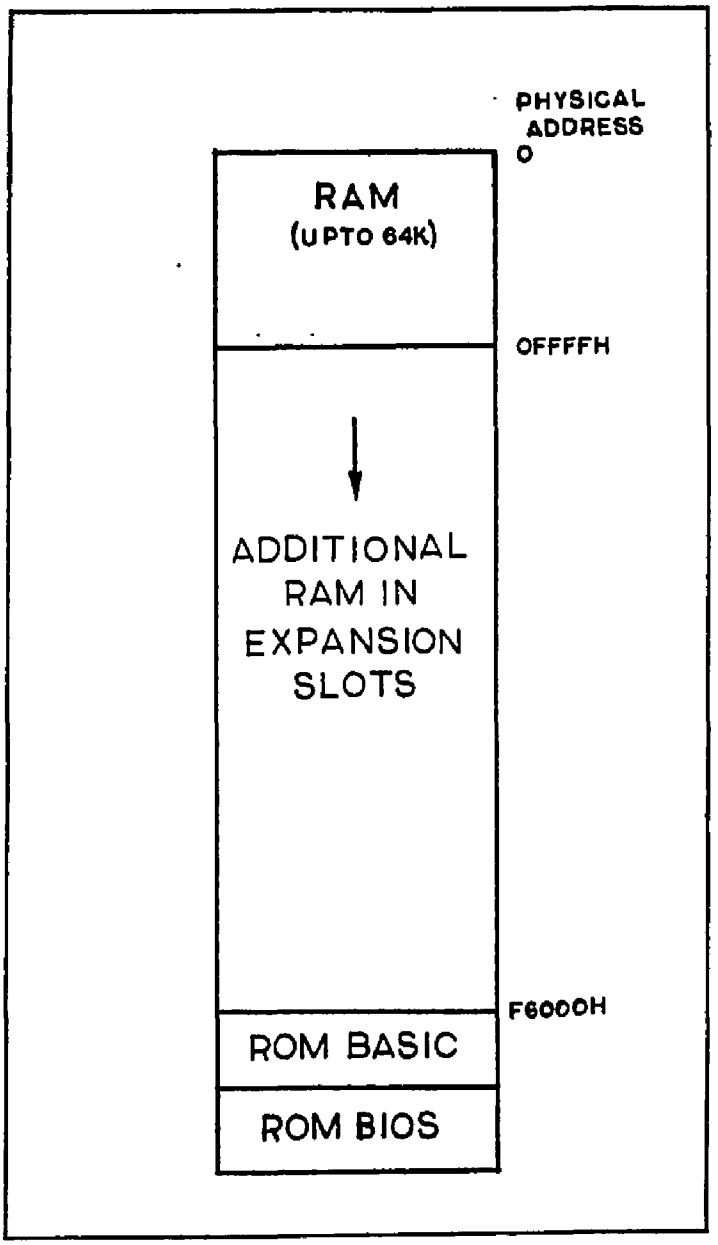

Figure 6. System Memory Map (From Willen and Krantz 1983)

in IEEE notation (Morgan and Waite 1982). which is one of the standard notations available for storage of floating point numbers.Integers in 8088 are stored in two's complement notation (Willen and Krantz 1983). In a two's complement notation the leftmost bit is called a sign bit. It is zero for a nonnegative number and 1 for a negative number. The largest and smallest numbers, floating point and integer, are determined by the unit of storage used for that particular number. For example, if a byte is used to store an integer, it can range in value from -2^7 to $+2^7-1$ or from -128 to +127. The storage units are shown in figure 7 and figure 8 shows how various data types are stored in these storage units.

3.5 The system controllers

Certain functions inside the personal computer are carried out by support microprocessors. These support chips either relieve the CPU of time consuming overheads or are essential for the proper functioning of the 8088 central processor itself.

The memory of the computer is controlled by a mechanism known as Direct Memory Access or DMA. The DMA mechanism in IBM personal computer is implemented through the 8237 DMA controller chip. This chip takes charge of the task of transferring data from the main memory to an

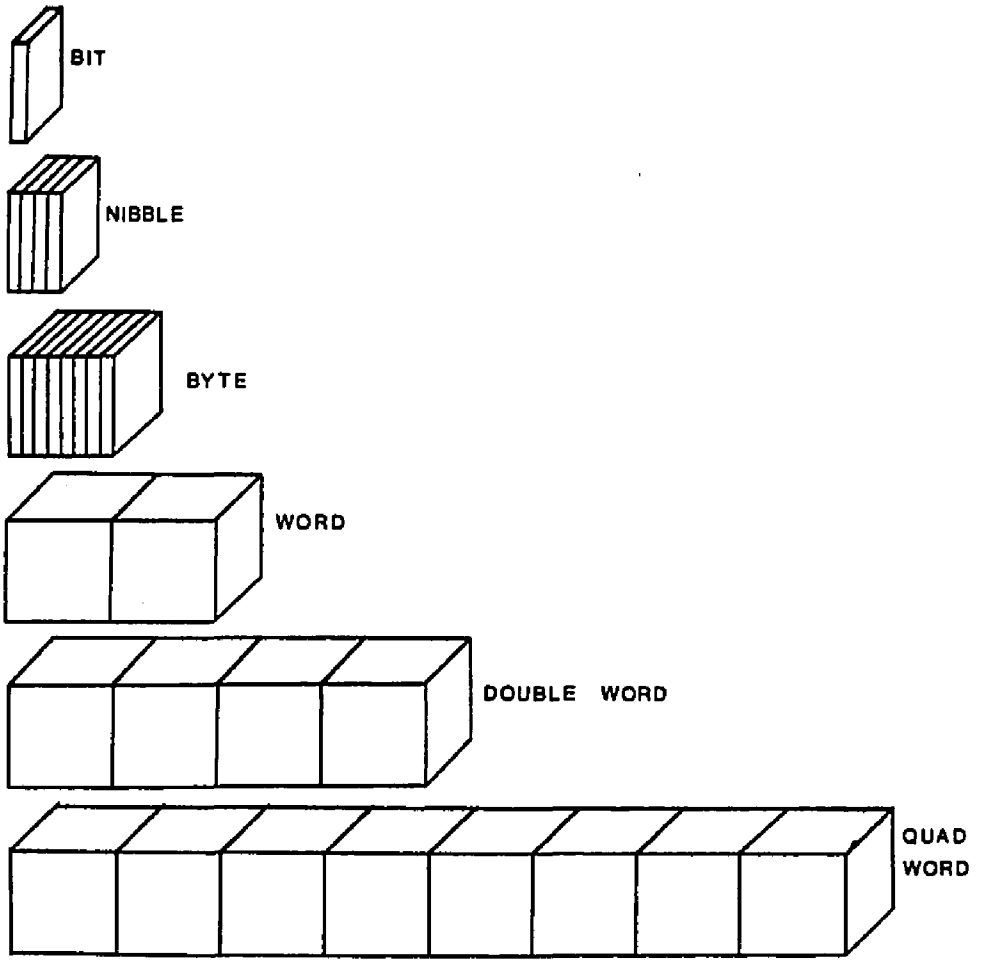

Figure 7. Units of Storage (From Morgan and Waite 1982)

DATA TYPES	UNITS OF STORAGE: BIT	NIBBLE	BYTE	WORD	DOUBLE WORD	QUAD WORD
LOGICAL	*		*			
ORDINAL			*	*	*	*
INTEGER			*	*	*	*
FLOATING POINT					*	*
CHARACTER			*			
POINTER					*	

Figure 8. Data Types and how they are stored
(From Morgan and Waite 1982)

input/output device, like the floppy disk, and vice versa, thus relieving the 8088 CPU of this time consuming operation.

The 8259 interrupt controller (figure5) handles all interrupts before they reach the 8088. Interrupts are signals of external events originating in devices like the floppy disk, keyboard etc. The 8088, however, has only one interrupt input line, and therefore, can receive only one interrupt at a time. The 8259 manages the interrupts and holds on to secondary interrupts while the 8088 processes the first. 8259 itself is capable of receiving up to eight interrupt signals at a time.

The 8253 timer performs important timing and counting functions in a PC. Some of these include issuing memory refresh pulses, maintaining the time-of-day clock and controlling the timing of the disk motor. The 8288 bus controller is used between the 8088 and system bus for functions like time multiplexing and encoding (Willen and Krantz 1983)

3.6 The device controllers

External devices like the keyboard, disk drives, printer and the video monitor communicate with the 8088 and other microprocessors via the system bus. However,

they are not tied to the system bus directly. They are connected to device controller chips which, in turn, are tied to the system bus. There are two general purpose device controllers, viz, the 8250 asynchronous communications element or serial interface controller and the 8255 parallel interface controller (figure 5). The 6845 CRT controller is a special purpose controller (figure 5).

The 8250 chip is the Universal Asynchronous Receiver Transmitter or in short UART. This microprocessor performs all the serial protocol conversion as data is sent and received through the RS-232 serial interface. In serial communications all data and control signals are sent through a single line as opposed to parallel communications where there are separate lines for each data and control signals. A serial protocol is nothing but a set of rules regarding transmission rate in bits per sec also known as baud rate, parity, number of bits per character etc., such that signals are sent and received properly without getting mixed up. The 8250 is a dedicated microprocessor that handles all the details of serial communication. Programming in a serial communications environment without this chip would be extremely complex. The 8255 parallel interface controller is used in the personal

computer to control a variety of devices including the printer. It controls the switches indicating the number of different devices attached to the system and the amount of available memory.

The 6845 CRT controller is a special purpose microprocessor used in the monochrome and color/graphics adapter which controls the monochrome and color video monitors respectively. The characters in a video screen are made up of dots and the process of character formation is quite complex. Furthermore, each character can have its own display attributes like reverse video, blinking, high intensity etc. All these complex mechanisms are handled by the 6845. In case of a color video screen, the 6845 also handles the mechanism of displaying a color image. In addition, there are three integrated circuits on the disk controller board. These IC's encode and decode digital data flow to and from the diskettes and hard disk. These are the only three chips made by IBM itself in the personal computer (Sargent and Shoemaker 1984).

3.7 Time Multiplexing and Encoding

The 8088 has forty pins through which it communicates with the bus. However, same pins are used to emit address information and to send and receive data.

This is achieved through the technique of time multiplexing. When memory needs to be accessed, the address is sent during the first clock cycle and data in later clock cycles. This timing mechanism is employed by means of a clock signal generated by the 8284 clock generator. The 8284 uses an attached crystal to determine the clock frequency. A clock frequency of 4.77 MHz is used in the IBM personal computer. Control signals from 8088 are encoded into numbers from 0 through 7 such that eight signals can be sent using only three pins. This process is known as encoding. The encoded signals are decoded by the 8288 bus controller and buffered into the control bus. Time multiplexing and encoding are shown in figure 9.

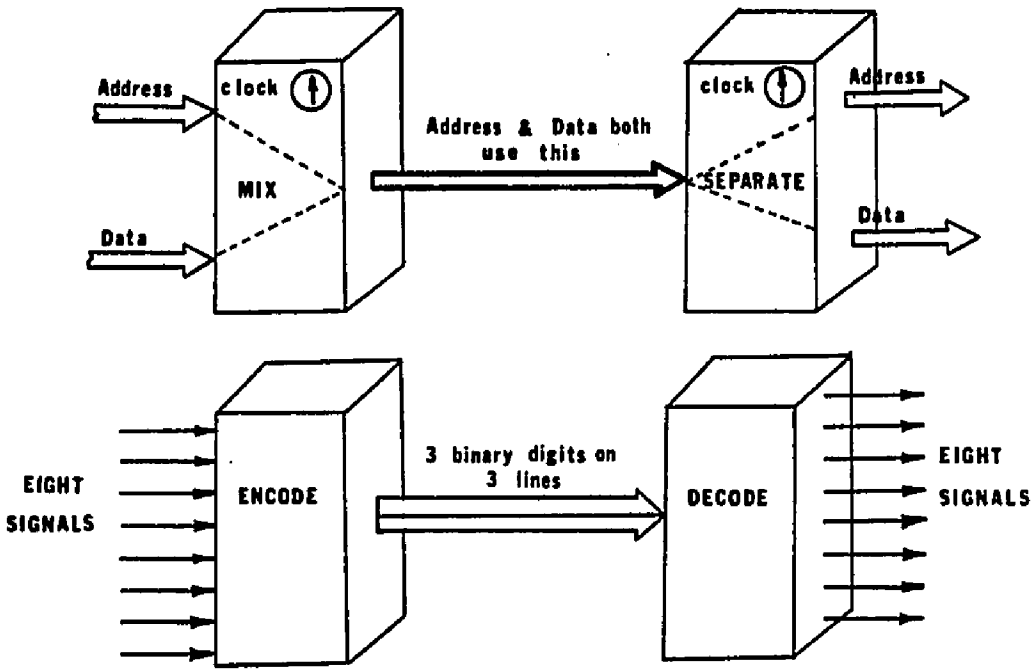

Figure 9. Time Multiplexing and Encoding
(From Morgan and Waite 1982)

CHAPTER 4

8088 ARCHITECTURE AND ASSEMBLY LANGUAGE PROGRAMMING

4.1 General

The 8088 is a general purpose microprocessor and is an 8-bit version of the more general Intel 8086 16-bit microprocessor. Internally, the 8088 has 16-bit registers. Outside connections are 8-bit for 8088 and 16-bit for 8086. A functional diagram of 8088 is shown in figure 10. The microprocessor can be divided into two parts, the execution unit or EU and the bus interface unit or BIU. The EU of 8088 is identical to 8086, whereas, the BIU of 8088 is 8-bit compared to 16-bit for 8086. The immediate quetion arises why the 8088, and not 8086, is used in IBM personal computer. The reason is to maintain upward compatibility with Intel's older 8-bit microprocessors like the 8080 and 8085 (Morgan and Waite 1982). The 8088 is said to have pipelined design because of the instruction stream byte queue. Instructions fetched from memory are placed in this queue. The execution unit control system then takes an instruction and sends it to the EU for execution. As execution is taking place, the bus interface unit fills up the

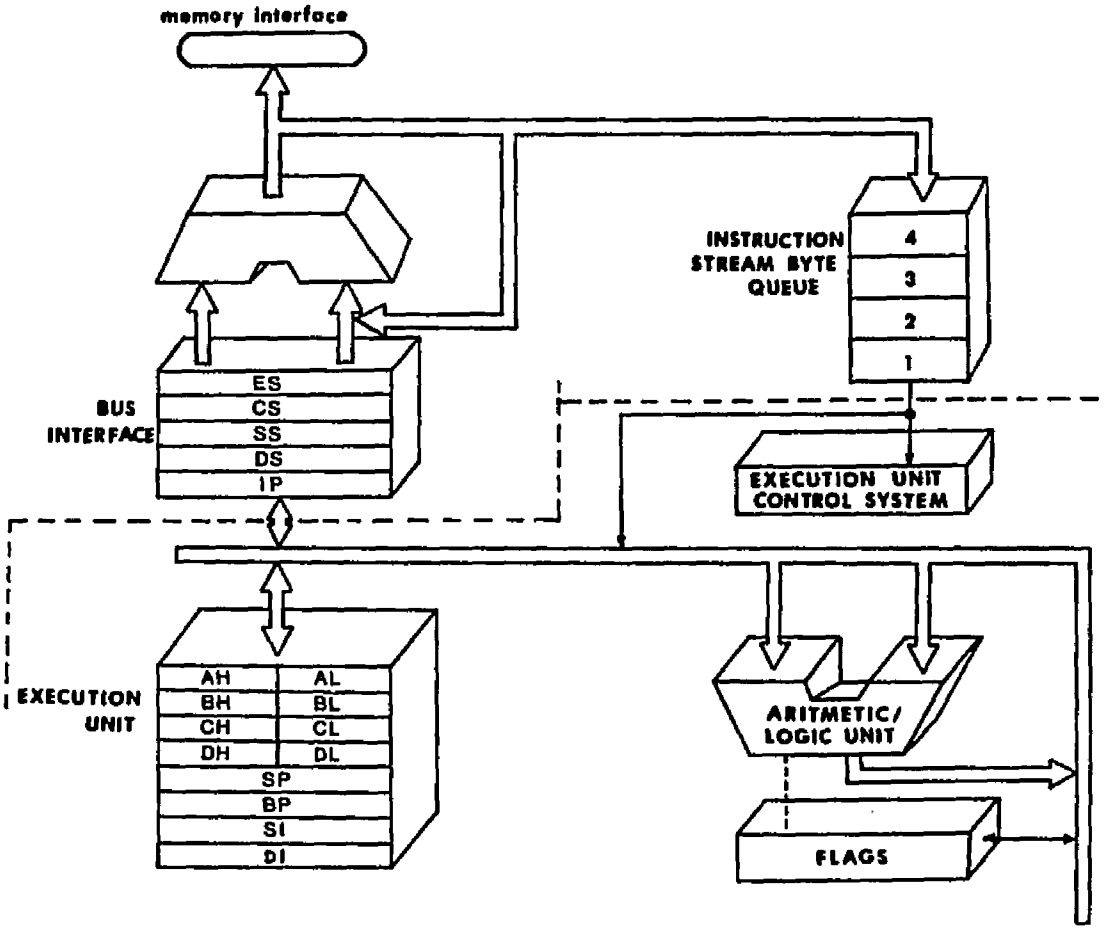

Figure 10. Functional Diagram of 8088
(From Morgan and Waite 1982)

instruction stream byte queue. The EU, therefore, never has to wait for an instruction to be fetched from memory. This results in increased speed of the 8088. In 8086, this queue is 6 bytes long and instructions are sent to the EU two bytes at a time (Willen and Krantz 1983).

4.2 8088 Register set

The 8088 register set is shown in figure 11. Registers are the most important components of a microprocessor as far as programmers are concerned. All operations like addition, subtraction, multiplication, division, string manipulation, and input/output addressing are done through the registers. Registers are like variables in a high level language program. The difference is that the number of registers are fixed. For example, to add 15 to 13, 15 is moved to register AX, 13 is moved to BX and then BX is added to AX. The result 28 is stored in AX. Some registers of 8088 are in the execution unit and others are in the bus interface unit as shown in figure 10.

There are four general purpose registers in 8088 each 16 bit wide. The four registers are accumulator register AX, base register BX, count register CX, and data register DX. All these register can be optionally broken down under program control into two eight bit

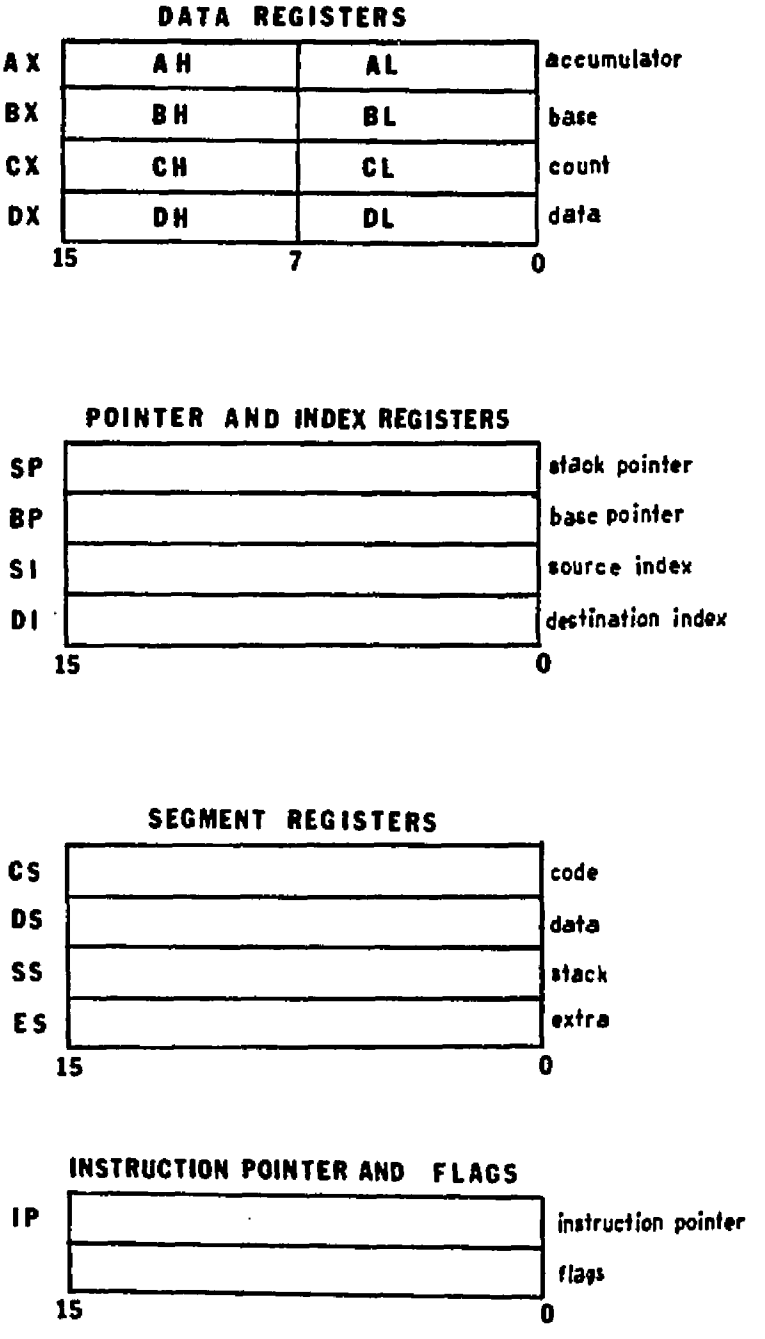

Figure 11. The 8088 Register Set
(From Sargent and
Shoemaker 1984)

parts. The lefthand eight bits is called the high byte or the most significant byte, and the righthand eight bits is called the low byte or the least significant byte.

The next group of registers are the pointer and index registers. Both are 16-bit registers and cannot be further subdivided. This group includes the stack pointer SP, the base pointer BP, the source index SI, and the destination index DI. Registers SP and BP are used for stack operations. A stack is a last-in-first-out structure used in programs to save intermediate results and then retrieve them when necessary. Particular use of stacks is in saving return addresses when subroutine or functions are called. Registers SI and DI are used for string manipulation instructions. SI points to the location of a source string and DI points to the location of the destination string when a string is to be moved. BP, SI, and DI can also be used as general purpose registers.

There are two special purpose registers, the instruction pointer IP and the flags register. The instruction pointer points to the current instruction to be fetched from memory. The flags register has several one bit flags describing the current status of the microprocessor. For example, if the results of an arithmetic operation is zero, the zero flag is set. A

flag is set when a value of one is moved into the flag. Figure 12 shows the flags register.

Finally, there are the four segment registers. These are, code segment register CS, data segment register DS, stack segment register SS and extra segment register ES. These registers are explicitly or implicitly used when memory is accessed and are needed due to the segmentation scheme used in 8088 to access memory.

4.3 8088 segmentation scheme

The data subbus, as noted earlier, has 20 signal lines and as a result the addressing capability of 8088 is 1 megabyte. This 1 megabyte memory space is logically divided into segments shown in figure 13. A memory address is a binary number and therefore, with a 16-bit register of the 8088, 2^{16} or 64K bytes of memory can be accessed. To access the one megabyte memory a special scheme known as segmentation is employed (Morgan and Waite 1982). In this scheme, a 16-bit register addresses memory within a 64K segment. Another register, known as a segment register, helps point to the starting address of a segment. When memory needs to be accessed, a segment register is multiplied by 16 to form a 20-bit number. This 20-bit number is then added to a 16-bit number taken from one of the other registers. The second number points

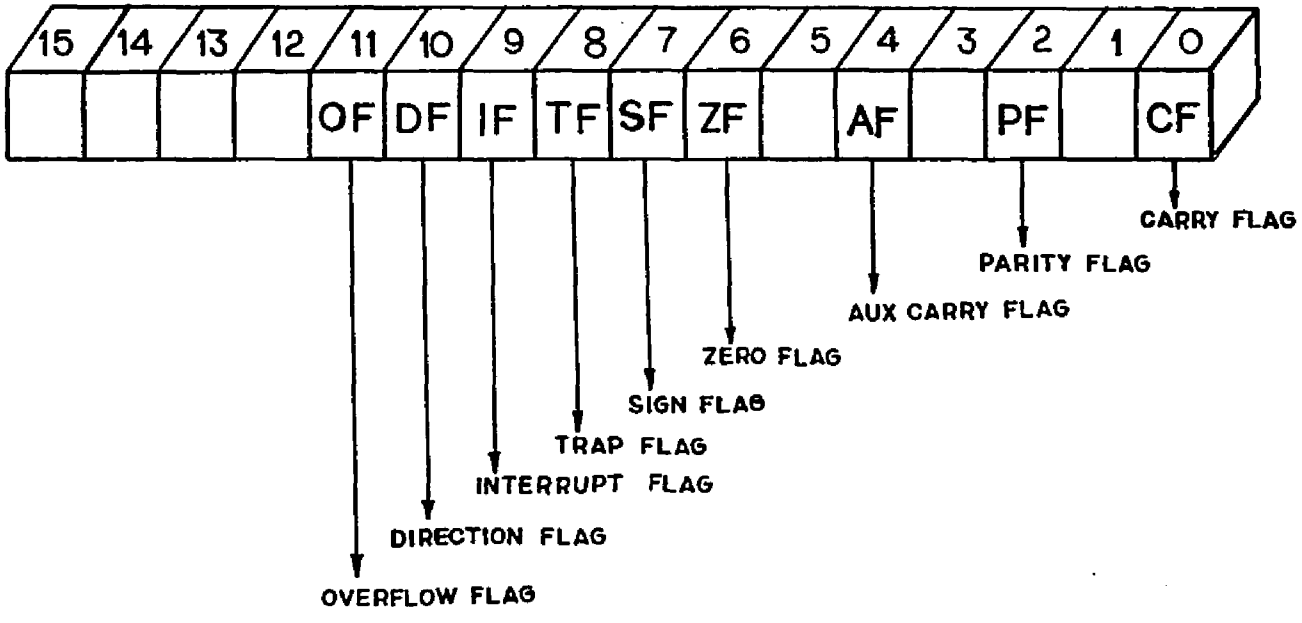

Figure 12. The Flags Register
(From Willen and Krantz 1983)

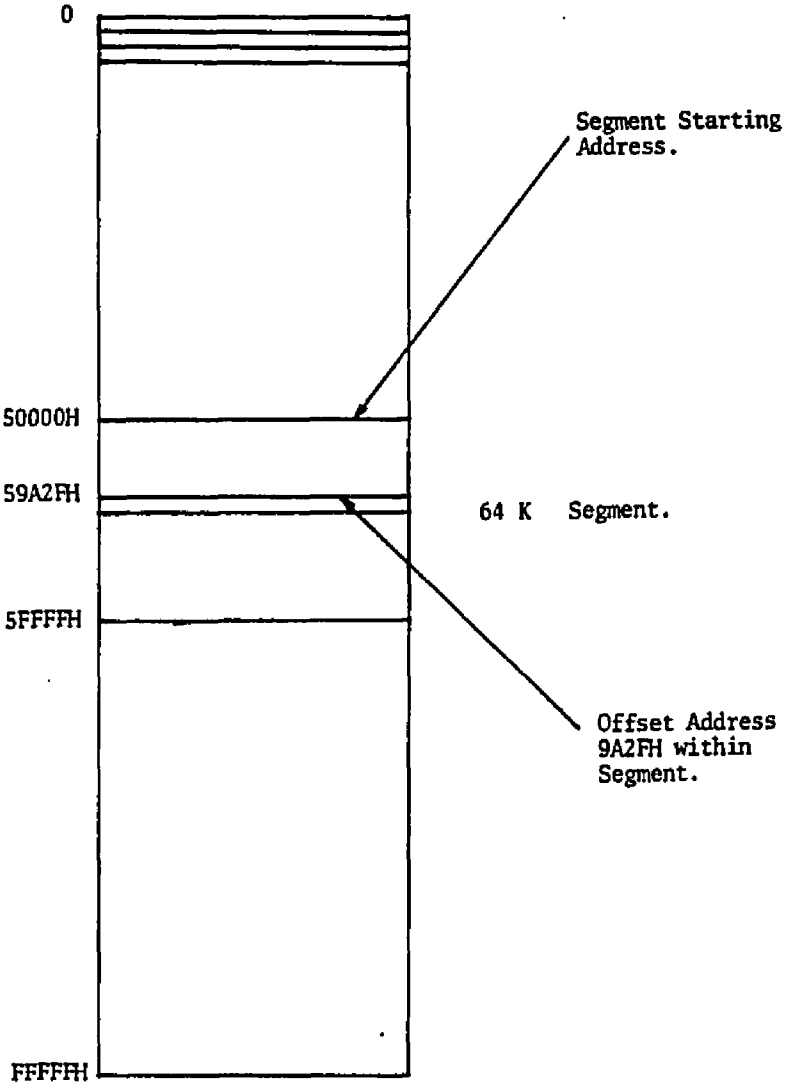

Figure 13. A 64K Segment within 1M Address Space (From Willen and Krantz 1983)

to a location within a segment. The resultant 20-bit number is then able to point at any place within the 1 megabyte address space and is known as the physical address. Physical address computation is shown in figure 14. Though the size of a segment is limited to 64K, a segment may start at different locations which can be controlled from an assembly language program. The address inside a segment is known as offset address and a complete address is often referred to as segment:offset.

4.4 Programming the 8088 and support chips

The 8088 and the supporting microprocessors can be programmed with a set of instructions in machine language. Because a machine language is made up of an array of numbers, it is hard to develop and follow. For this reason programs are written in assembly language. In assembly language, machine language numbers representing instructions are replaced by pnemonics. For example, the machine language statement for moving number 7 into the low byte of register AX is given by :

B0 07

Corresponding assembly language instruction is :

MOV AL,7

Where, MOV is the command and AL is the register into which the number 7 is moved.

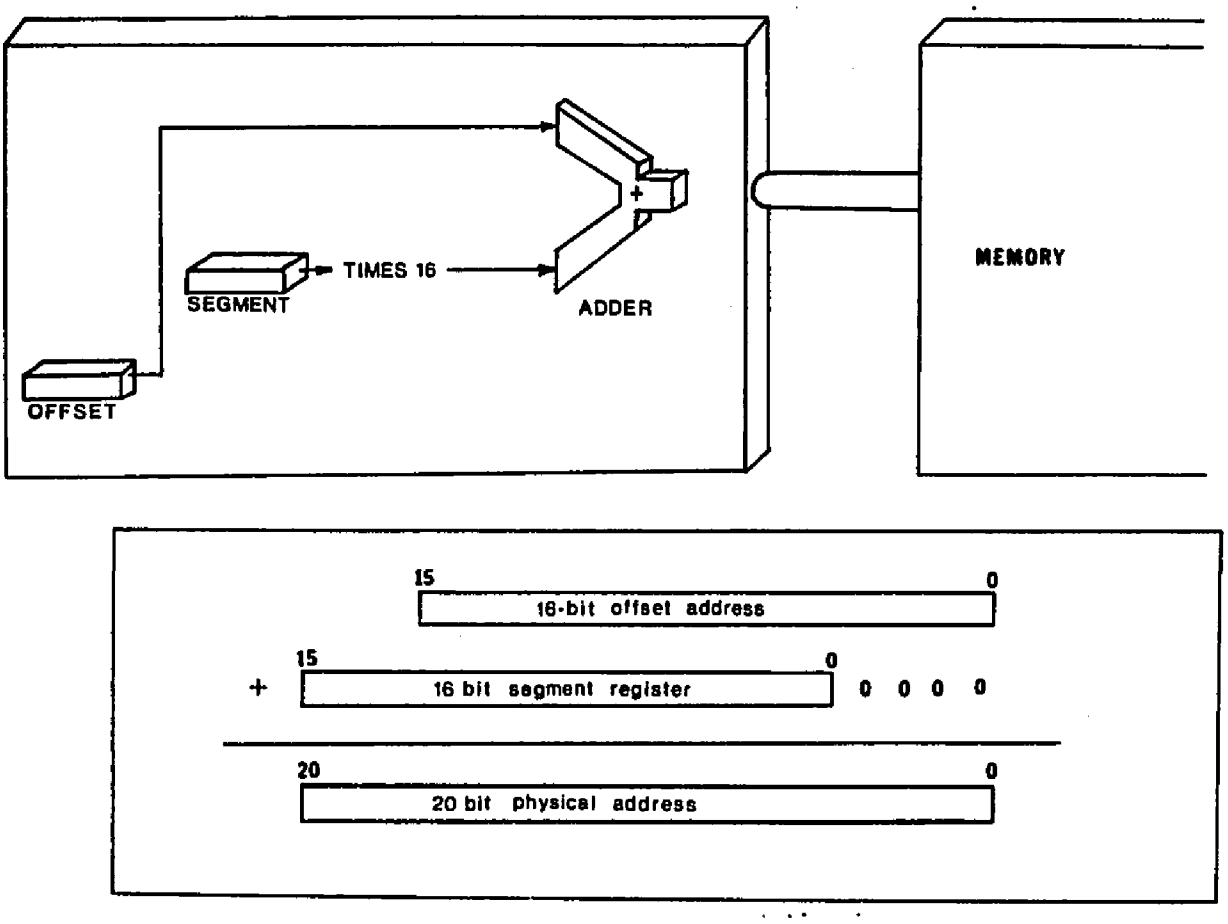

Figure 14. Physical Address Computation (From Morgan and Waite; Willen and Krantz)

There are a large number of machine language instructions for the 8088 and their assembly language pnemonics. Other than machine language pnemonics, an assembly language program consists of psedo-operations. These are directives for the assembler program, and are not translated into machine language instructions by the assembler.

A useful feature of the IBM personal computer is the use of relocatable object modules (Morgan and Waite 1982). Compilers for high level language and assemblers for assembly language translate source programs to object modules. These object modules are called relocatable because absolute memory addresses are not specified in these modules. In other words, the object code may not be loaded directly into the memory for execution. A program called DOS linker creates a load module from one or more object modules. This approach enables high level language routines to be linked together with assembly language routines to form a single load module. For example, one part of a program may be written in a high level language like Fortran and another part in assembly language. The two may then be linked together to form a single executable program. In this manner, one can take advantage of both languages at the same time.

Support microprocessors can be programmed via 8088 input/ouput instructions. All programmable support chips are mapped into logical port addresses of 8088. Therefore, an instruction for a support chip can be sent from 8088 by using the appropriate port.

4.5 Basic Input Output System and Disk Operating System

Basic Input Output System (BIOS) and Disk Operating System (DOS) are sets of programs that manage the computer operations and allow applications to be written for the computer. The Basic Input Output System or BIOS is a set of programs stored in the ROM memory of the computer. When the computer is turned on, these set of programs take control of the system and executes certain tasks needed for initializing the computer. Specific portions of the Disk Operating System or DOS (which is another set of programs stored in a diskette or the hard disk) are then read in and loaded into memory. Programs in DOS manage various operations when the computer is up and running. However, unlike BIOS, DOS is stored in RAM memory, and is erased when the computer is turned off or rebooted. After BIOS loads a part of DOS into memory, it passes control to this part as well, which, in turn, loads rest of the DOS into memory and does some other operations to initialize the computer.

This initialization process, first by BIOS and then by DOS, is known as Bootstrapping. The DOS programs must be in disk drive A or in the hard disk for proper bootstrapping. The system memory map after bootstrapping is shown in figure 15 . DOS and BIOS also provide a convenient way of programming in assembly language in many cases. Knowledge of the system memory map is essential for memory read/write operations in assembly language programming. Programs and data reside in the area marked free memory in figure 15.

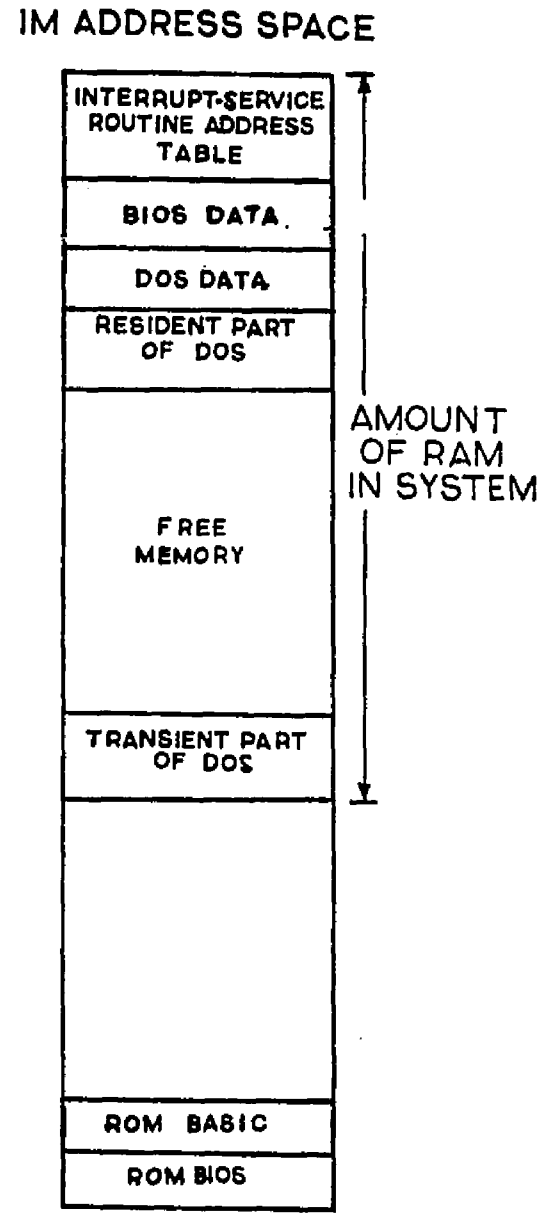

Figure 15. DOS Memory Map after Bootstrap (After Willen and Krantz 1983)

CHAPTER 5

MANAGING TIME AND DATE FOR DISPATCHING SYSTEM

5.1 Time-of-day clock

The dispatching system is designed to operate on a real time basis. Therefore, accurate time keeping is very important. Fortunately, a time-of-day clock is maintained by the IBM personal computer in memory as a 32-bit binary number. However, The clock comes into operation only after the system is bootstrapped (turned on) and ceases to function as soon as the computer is turned off. Current time must be re-entered by the user each time the system is bootstrapped. To avoid this inconvenience as well as possible error, a perpetual clock with battery backup has been installed as a part of a multifunction card in one of the expansion slots. During bootstrapping, the DOS automatically reads in the time and date from this clock.

The 32-bit binary clock is maintained in memory by BIOS in 55 millisecond increments. This means the clock is updated by one unit every 55 millisecond. Time is maintained with help from the 8253 timer chip. In this chip, there are three counters, numbered 0,1, and 2.

These counters are capable of decrementing once they are loaded with an initial value from a latch register. The 8253 communicates with the external environment through the latch registers. There are three latch registers for the three counters in 8253. Each of these counters can be programmed to operate in one of six different modes numbered from 0 to 5. The mode can be set using the 8253 command register at logical port address 43H of the 8088. Of the six modes, only mode 3 is relevant for time keeping purposes. In mode 3, the 8253 generates a square wave signal on the output line. The frequency of the square wave is equal to the frequency of the input clock signal, divided by the value programmed into the latch register.

The clock input to the 8253 in IBM personal computer is tied to a 1.9318 MHz signal. The period of each cycle is 1/1.9318 or 840 nanoseconds. Upon power-up BIOS initializes channel zero to operate in mode 3 with a count value of 00000 Hex. This results in 65536 iterations before zero is reached again. The output of channel zero is a square wave of frequency 1.9318/65536 or 18.2 Hz. Thus, channel zero of 8253 sends an interrupt 18.2 times a second or every 55 milliseconds (Willen and Krantz 1983).

5.2 Calculating time from binary value

Procedure to calculate time in hours, minutes and seconds is shown below.

This calculation is best illustrated by an example. Suppose the binary number representing time at any particular moment is :

0000 0000 0000 0011 1110 1001 0011 0111

The number above is divided into groups of four digits each because it will be converted to hexadecimal. Groups of four digits of a binary number starting from the right forms hexadecimal digits. The corresponding hexadecimal number is shown below :

0003E937 or 3E937

Corresponding decimal number is 256311. Because time-of-day is kept in 55 millisecond intervals, time in seconds represented by the above number is :

256311 x 55 / 1000 = 14097.105

Converting to hours, minutes and seconds, time-of-day represented by the binary number is :

03:54:57.10

Because time is maintained as a 24-hour clock the time above is AM and not PM.

5.3 Programs to read and display the time

Subroutines CLOCK in assembly language and TIME in Fortran are written to make the time available to

other subroutines and programs of the dispatching system as and when needed. Any Fortran program can call TIME which in turn calls CLOCK.

Time kept in BIOS time-of-day clock is accessed through DOS interrupt call given in table 1. Assembly language function CLOCK uses the assembly language instruction INT 1AH to get the time in CX and DX registers. INT 1AH is a standard interrupt provided by the disk operating system. To return this value to the Fortran subroutine TIME, the 32-bit binary number is moved to the register pair DX,AX. DX has the most significant word and AX has the least significant word. A dummy argument is used in the calling statement of TIME as :

```
TIME = CLOCK(I)
```

where, I is the dummy argument. Use of an argument even if it is not used is a requirement of the Fortran compiler.

Subroutine TIME has three arguments. Time-of-day is passed to the calling program as the first argument. The other two arguments are to provide the user with some options. The first argument must be declared (or default) REAL and the latter two must be declared (or default) INTEGER. Integer constants can also be used directly in place of integer variables. Argument 2 provides the user

Table 1. DOS interrupt call for time-of-the-day (Willen and Krantz 1983)

Interrupt Call	Input Requirements	Output / Results
INT 1AH	AH = Ø	Returns current value of time-of-day counter. CX = Most Significant word of count. DX = Least Significant word of count. AL = Ø, if count has not passed 24 hours since the last time it was read.
INT 1AH	AH = 1 CX = MSW of count. DX = LSW of count.	Sets the time-of-day counter to the value specified in the CX and DX registers.

with an option to display the time on screen. If the value is zero, there is no display. If the value is one, there is a display in the usual hh:mm:ss format. If any other value is passed, an error is flagged. Time can be returned in argument 1 in hours, minutes, or seconds. This is controlled by argument three as follows :

Value of Argument 3	Value returned in
1	Hours
2	Minutes
3	Seconds
Any other value	Error flagged

For example, if the call statement is CALL TIME(VAL,1,3) and the time is 54.6 seconds after midnight, then there will be a display on the screen like:

Current time is__ 0: 0:54.6

and a value of 54.6 will be returned in VAL. Dispatch system calls time with arguments as above. The object codes of TIME and CLOCK are linked together with other routines of the dispatch system to obtain a single executable code. This executable code is referred to as the main program of the dispatch system hereafter.

5.4 Assembly language subroutines for Fortran

There are several considerations in writing an assembly language subroutine which is to be called from a

Fortran program (Microsoft Fortran 77 User's Manual 1983). In the Fortran used to program the dispatching system, all subroutines and functions are external by default which means they can be programmed separately from the main program. Also, all subroutine or function calls are long calls and have four byte return addresses. In a long call the subroutine or function is located at a different segment than the main program and four bytes are necessary to store the main program address at which the subroutine or function would return.

When an assembly language function or subroutine is called, Fortran builds up a frame. This frame contains all the necessary information for proper return to the caller. The frame is saved in memory and register BP contains the framepointer. Registers DS and SS also contain information necessary for a proper return. Therefore, the first task in an assembly language function or subroutine is to save registers BP, DS and SS by pushing them into the stack. Before returning to Fortran, these values must be popped off from the stack. However, there is no need to set up a stack in the assembly language routine because it can use the Fortran stack. To ensure that the assembly language routine is accesible from a Fortran program, it must be declared public.

When an assembly language function or subroutine is called, the address of actual parameters are automatically pushed into the stack. Before returning to the calling program, the assembly language program must discard these addresses from the stack by popping them off. Functions return a one-byte value in AL, a two-byte value in AX, and a four byte value in the DX,AX pair. The return value is moved into the DX,AX pair as shown in figure 16.

In case of truck dispatching system and function CLOCK, address of integer I is pushed first. Since I is the only parameter, the return address is pushed next. All addresses are in segment:offset (see chapter 4) form because of 8088 segmentation scheme. Before clock returns to the calling program TIME, address of parameter I is discarded from the stack. Stack diagram just before transfer to CLOCK is shown in figure 17.

5.5 Incorporating the date

To incorporate date in the dispatching system, a Fortran library system called FORLIB.PLUS is used. One of the libraries in this system is called ACSV20 and has the assembly language routine to get the date from the computer. Another program, also in FORLIB.PLUS, makes the date available to any Fortran program. However, ACSV20 is

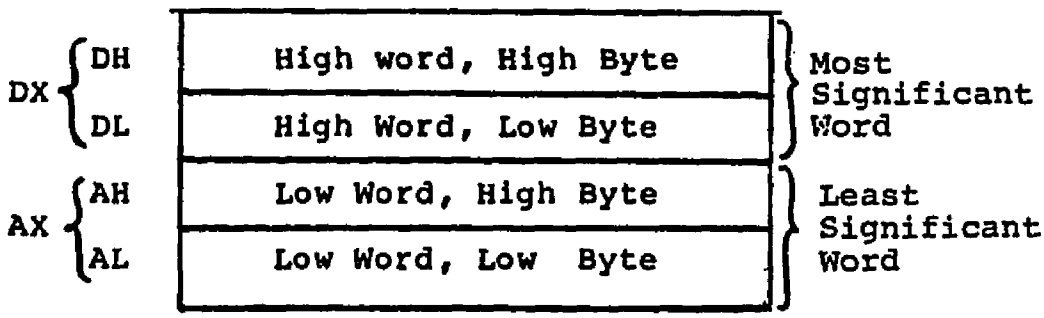

Figure 16. Location of Four-Byte Return Value
(From Microsoft Fortran 77 Manual)

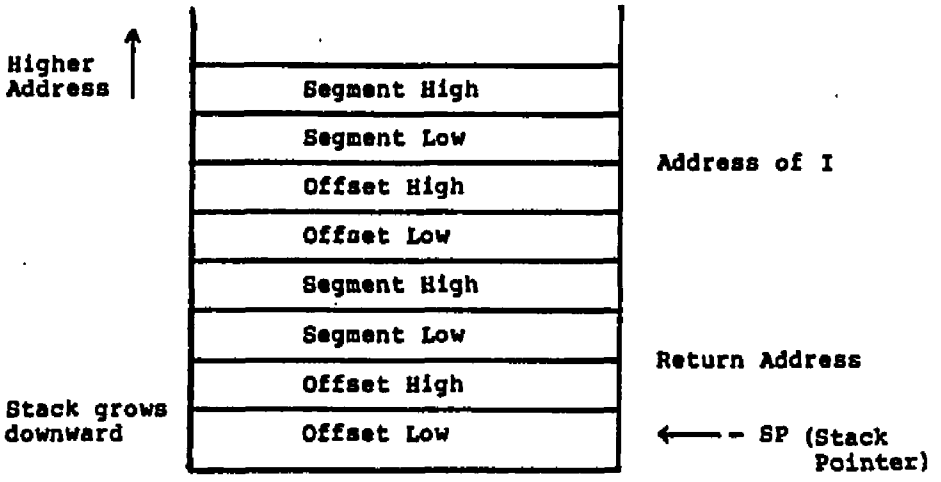

Figure 17. Stack Before Transfer to Function CLOCK

not used as a library when the main dispatch programs are linked. This helps reduce the size of the executable code which otherwise tends to approach the memory limit of the machine. As an alternative, a small program called DATE20.FOR is written which writes the date in a sequential file. When the main dispatch program takes control, it simply reads in the date from this sequential file. The date comes in month, day, year in three separate variables which are then packed into one single variable. When date needs to be displayed, this variable is unpacked to give month, day and year.

CHAPTER 6

SUBSYSTEM TO DISPLAY THE DISPATCH BOARD

6.1 The dispatch board

When the dispatch system is running, the status of trucks and shovels is shown on a dispatch board. The dispatch board is in the form of a video display rather than the board operated with plexiglass slides discussed in chapter two. The board is updated every three minutes and after it is updated, a permanent copy is printed out. This board enables the dispatcher to keep track of truck activities around shovels in the pit. The dispatcher is thus better informed to override the assignment made by the computer. Any unusual occurrences can also be easily spotted. A sample display of the dispatch board is shown in figure 18.

The top row shows the shovels with their two digit identification number. The first information on the left of the board is the time at which it was last updated. Then there is a column of arrival times for trucks. The central part of the board contains three digit truck identification numbers. Each truck is reported under a particular shovel and against a time

SHOVELS ==>		11	12	13	DOWN	SPARE
	10+LATE					
	10 LATE	911				
	5 LATE	912	915			
7:19	ON TIME	901 908	904 910	906		
	5		903	902		
	10		907			
	15					
	20					
	25	914		909		
	30		913			
	35					

Figure 18. Dispatch Board

shown on its left. The shovel represents the one the truck is assigned to and the time represents the number of minutes after which a truck is supposed to report for reassignment. If a truck is shown against ON TIME, it is supposed to report for reassignment within the next three minutes. A reassignment takes place after the truck has dumped its load at the dump or crusher. If a truck is shown against one of the three late rows, it has not reported for reassignment and is late by the amount shown plus approximately three minutes. A truck might be shown also under the down or spare column.

6.2 Displaying the dispatch board

Two programs are responsible for displaying the dispatch board. First, however, the main dispatch program exits after creating a file BOARD.DAT. Then, DOS command MODE transfers control to the second video monitor. At this point, program CURSORCO.ASM written in assembly language takes control to remove the blinking cursor, a source of distraction, from the screen. When the cursor is removed, program DISPLAY.FOR takes control. This program reads in the values from the file BOARD.DAT. BOARD.DAT contains values for various elements of the dispatch board described in the last section. These values are written by the main program in integer format

but read in by DISPLAY.FOR as characters. Zeros are then replaced by blanks through a DO-loop. This is necessary to make the dispatch board legible because most of the values on the board at any particular time are zeros.

6.3 Programming the 6845 color controller

To remove the cursor from the second video monitor, which is a color monitor, assembly language routine CURSORCO.ASM programs the 6845 color adapter. Though a very complex microprocessor, upon bootstrap, BIOS initializes this chip thereby making its control through a program fairly simple. The 6845 register that controls the cursor image can be programmed via 8088 instructions through designated output ports. The 6845 registers as they are on the color adapter, are described in Table 2. There is an address register mapped at 3D4H of the 8088 which is used to select all other registers. All other registers are mapped at i/o address 3D5H of 8088 (Willen and Krantz 1983).

To access a particular register, that register number must be placed in the address register at i/o address 3D4H. Then, the intended value for that register can be sent through i/o address 3D5H. To remove the cursor, cursor start register is accessed by placing a value of 10 in the address register through port 3D4H.

Table 2. 6845 Registers as on the color adapter (From Willen and Krantz 1983)

PC/XT i/o addr.	6845 Register	Register Number	Description
3D4H	Address Register		Used to select all other registers
3D5H	Horizontal Registers	0 - 3	Initialized to control Hor. characteristics
	Vertical Registers	4 - 9	Initialized to control Ver. characteristics
	Cursor Start	10	These registers define cursor image
	Cursor end	11	
	Start add. High	12	These registers define which part of the adapter memory is used
	Start add Low	13	
	Cursor add. High	14	These registers determine in which display position the cursor will appear
	Cursor add. Low	15	

Then a value of 20H is output through port 3D4H. The cursor start register is shown in figure 19. It is clear from the figure that the cursor will not be displayed if bit 6 is set to zero and bit 5 is set to one in the register. A value of 20H in hexadecimal corresponds to 00100000 in binary. A value of one is thus set to bit 5 and all other bits are set to zero. This effectively turns the cursor off.

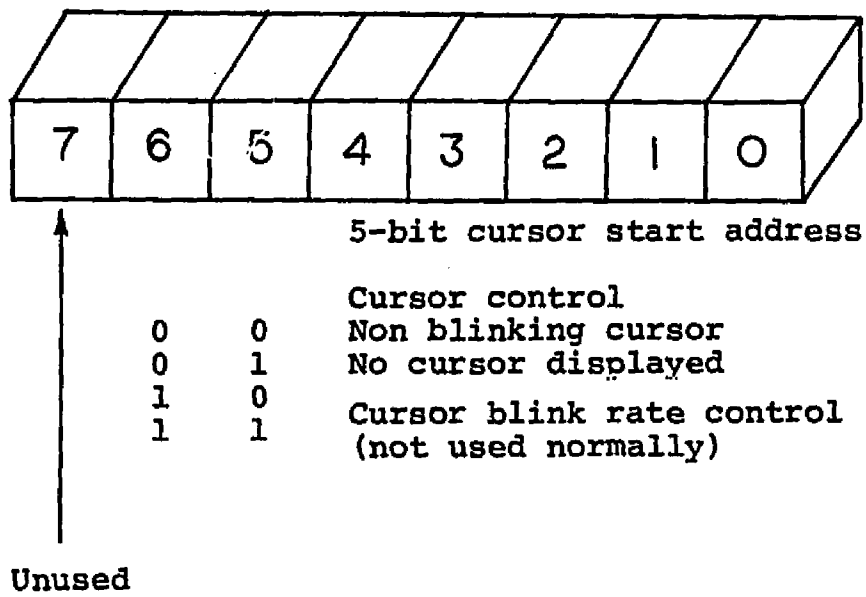

Figure 19. Cursor Start Register
(From Willen and Krantz 1983)

CHAPTER 7

PROGRAM SPEED CONSIDERATIONS AND SOUND EFFECTS

7.1 General

Because the dispatch system is to operate in a real time environment, speed of execution of different programs is a major consideration. The programs must be able to process data coming from the field before the next event occurs. With a mainframe or a state-of-the art minicomputer this poses no problem. In a microcomputer, however, this is a major constraint and several steps are taken to ease it. Truck assignments in the dispatching system are based on a heuristic algorithm because more sophisticated algorithms would take a longer time to process. Another factor in improved speed is the use of the 8087 numerical data processor. This is a coprocessor of 8088 and speeds up program execution remarkably. Finally, major savings in time are achieved using two programs called Superdrive and Superspool. These programs speed up input/output operations between the central processing unit and peripheral devices.

7.2 8087 numerical data processor

The IBM personal computer has a built in provision for using the 8087 numerical data processor, a coprocessor of 8088. In a coprocessing system, two or more processors share the same instruction stream (Morgan and Waite 1982). The 8088 and 8087 follow the same program, but switch turns in executing the instructions. Some operations are best performed by 8087 and other operations are better performed by 8088. Overall, using the 8087 in conjunction with the 8088 improves the speed of program execution by as much as 100 times in some cases. The 8087 is particularly adept at performing floating point operations. To take advantage of the 8087, however, appropriate software is needed. Fortunately, the Fortran compiler used in dispatching system can take advantage of the 8087 numerical data processor. Table 3 gives a speed comparison between 8086 and 8087 and can be considered as a fair indicator of the improved speed that can be attained by using the 8087 with the 8088.

7.3 Superdrive and Superspool

Input/output is an area of slow operation that can be improved by using a software called Superdrive (AST Research Inc. 1983). The total size of executable code is around 310 kilobytes for the dispatch system

Table 3. Speed comparison between 8086 and 8087 (After Morgan and Willen 1983)

Operation	Time in microseconds 8087	8088
Multiply (single precision)	19	1600
Multiply (double precision)	27	2100
Divide	39	3200
Square Root	36	19600
Tangent	90	13000
Exponentiation	100	17100

* both processors running at 5MHz.

programs. Moreover, there are data files read in by the system and output files written back. Overall control of system flow is done through DOS batch commands. As control passes from the main program to support programs, all variables are saved. When the main program takes control again, these variables are read back in to reinitialize the whole system. Also, when the program restarts, the executable code has to be loaded into memory from the disk. These input/output operations to and from the disk are dependent upon the physical speed of the disk drives.

The superdrive software simulates a diskdrive inside RAM memory. The capacity of the drives can be specified when it is created. For dispatching system, a drive of 330 kilobytes is created. Because the drive is a part of RAM memory, speed at which programs and data are retrieved and output written to are not limited by the physical limitations of a mechanical disk drive. These operations are carried out at RAM speeds (AST Research Inc. User's Manual 1983). Dispatching system programs are loaded into superdrive and executed from superdrive. Speed improvement achieved by this method is significant. The superdrive, however, is created by software and therefore, all programs as well as data files are lost if the computer is turned off or if there is a power

failure. For this reason, essential data files created by the program are saved in the hard disk from time to time.

Printing of files in dispatch system is carried out through other software called Superspool (AST Research Inc. 1983). This software creates a print buffer out of RAM memory. When a file is to be printed, it is read in from the disk and placed in this buffer. The 8088 is then relieved and is able to carry out further program execution. Without such a print buffer, the 8088 is tied down to the printing job and the real time nature gets completely jeopardized. Size of the print buffer can be specified at the time of creation of the buffer. A buffer size of 16 kilobytes is created for the dispatching system which is sufficient for holding the largest print file of the system. Superdrive and superspool together occupy 346 kilobytes of RAM memory and leave 230 kilobytes for DOS and program execution.

7.4 Sound generation through program delay loops

The dispatch board is updated every three minutes. After each updating, it is desirable to have some attention getter for the dispatcher. Also, a short beep with error messages help alert the dispatcher. Two programs, SOUND.ASM and BEEP.ASM are written for producing a series of tones and a short beep respectively.

Through SOUND.ASM, seven different tones are played in succesion through the speaker provided with the IBM personal computer. Frequency and duration of tones are changed dynamically with the program. Alternating values of the data bit associated with the speaker between zero and one, a square wave signal is produced. The duration of time this data bit is ON or OFF (one or zero) is controlled using delay loops. In this way, the frequency of the square wave can be controlled. Because the speaker data bit is used, the square wave is automatically gated to the speaker circuit. The speaker and its amplifier cannot, however, respond to the sharp edges of a square wave and the waveform is rounded off at the edges to produce a more pleasant sounding sine wave (Willen and Krantz 1983). This is shown in figure 20.

General approach of writing program SOUND.ASM is taken from Willen and Krantz 1983. Register DX is initialized to the number of square wave cycles to be produced in each tone and determines the duration of the tone. Register BX is initialized to the number of tones to be played. The speaker data bit is turned ON for the first half of the cycle. The duration of time the signal remains ON is determined by a value loaded in register CX. The second half of the cycle is then initiated by turning the speaker data bit OFF. CX is loaded with the

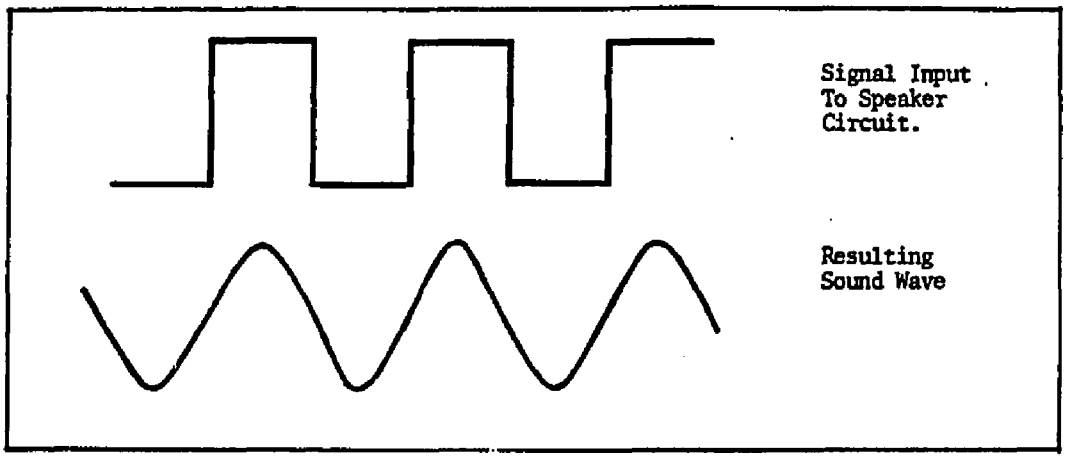

Figure 20. Speaker Data Signal
(From Willen and Krantz 1983)

same value as before. When a cycle is completed, the value in DX is counted down once and another cycle is initiated. The process repeats itself until DX reaches zero. At this point playing of one tone is completed. To play another tone, the value in BX (initialized to the number of tones to be played) is counted down once. Values in DX and CX are changed and as a result, a different tone with a different duration is played. Seven different tones are played in succession after which the program stops, because BX has been counted down to zero.

BEEP.ASM is programmed in the same manner as SOUND.ASM except that a single tone is played for a short duration. Also, BEEP.ASM is written as a public routine and is called from other programs by placing the statement CALL BEEP whereever necessary.

CHAPTER 8

SIMULATED TESTING, REPORT GENERATION AND SYSTEM FLOW

8.1 Simulating signpost data

The dispatching system is tested for validity using a second IBM personal computer. The two computers are connected through RS-232 serial interfaces (discussed later) and a special purpose cable. When a dispatch program needs data from the signposts, it sends a signal to the second PC. The second PC upon receiving the signal, asks for data input on the video screen. Test data is then keyed in to the second PC and a program sends this data back to the dispatching system running on the main computer. End of data input is signaled by keying in the letter 'E' on the second PC. When actual signposts will be designed as an extension of the current research, the second PC will no longer be needed.

8.2 Serial data transmission

As mentioned earlier, the two computers are connected through RS-232 serial interface and a special purpose cable. RS-232 serial interface is a standard hardware configuration used for serial communications. For the IBM personal computer, there can be at most two

RS-232 interfaces. Serial communication and parallel communication are the two methods by which data transfer can take place between two computers. In parallel communication, there is a separate line for each bit that make up the data and control signals. In serial communication, on the other hand, bits making up data and control signals are sent over a single line in succession. The operation is carried out using a timing mechanism. To make sure all the bits are sent and received properly without getting mixed up, a set of rules is used. A commonly used rule is the Asynchronous Serial Protocol. The speed at which serial data transmission takes place is measured in bits per sec and is known as baud rate.

Programming in an asynchronous serial communications environment to send and receive data is made simpler by the use of a dedicated micrprocessor in the IBM personal computer. This microprocessor is the National Semiconductor 8250 Universal Asynchronous Receiver Transmitter or UART in short. To send a byte of data, a program outputs the byte to the UART which is mapped to one of the input/output ports of the 8088. The UART then performs all protocol conversion and the data is transmitted with the appropriate characteristics. The RS-232 interface electronically connects the UART to the

outside world. If two computers to be connected are at long distance from each other, then the cable used for transmission is the phone line. A device known as the modem is needed between the RS-232 interface and the phone line to convert the electronic voltages to different tones that can be transmitted over the phone line (Willen and Krantz 1983).

In case of dispatching system, the two computers are placed side by side and a phone line and modems are not necessary. Instead, the two RS-232 interfaces are connected using a special cable. The RS-232 interface for IBM-PC is a 25-pin male connector. Although all pinouts are defined, only about nine pins are typically used (Sargent and Shoemaker 1984). The definitions of these nine pins are given in table 4. The wires inside the special cable connects pin 2 of the first (computer) RS-232 to pin 3 of the second (computer) RS-232 and vice-versa. Pin 7 of first is connected to pin 7 of second. Other pins are not used because they are for signals to control the modem and are not needed in case of the dispatch system data communication programs.

8.3 Communication programs for the dispatch system

To send signpost data from a second computer, two programs are written in assembly language. One program,

Table 4. RS-232 pin definitions (From Sargent and Shoemaker 1984)

Commonly used pin numbers for RS-232 lines

RS-232 DEFINITION	I/O	TERMINAL PIN NO.	MODEM PIN NO.
Signal Ground		1	1
Transmit Data	O	2	3
Recieve Data	I	3	2
Clear to Send	O	4	5
Data Set Ready	I	5	4
Chassis Ground	I	6	20
Carrier Detect		8	8
Data Terminal Ready	O	20	6

called SIGNALXT.ASM, runs on the main computer and another program, called SIGNALPC.ASM runs on the second computer.

When the main program of the dispatching system needs signpost data, control is transferred to the program SIGNALXT.ASM. This program opens a disk file to store the incoming data, initializes the UART, and then sends the character to the second computer to request signpost data input. To do this, the character 'S' is output to the UART through the 8088 port 3F8H. The register of the 8250 UART which holds a character before it is transmitted is mapped at this port address.

The UART then transmits the character via the RS-232 interface. Program SIGNALPC.ASM runs on the second computer in an infinite loop waiting for the character 'S' from the main computer. Within the loop it checks the status of the receiver data register of its own UART. The receiver data register holds a character after it is received from the RS-232 interface. When the character 'S' is found in the receiver data register, the program falls out of the infinite loop and starts the request for signpost data input on the video screen. Data is received from the keyboard one signpost at a time and is sent immediately back to the main computer through the serial connection.

SIGNALXT.ASM writes these data characters to the previously opened file SINPOST.DAT. A return key pressed on the keyboard ends the data from one signpost. SIGNALXT.ASM, on receiving the return, starts a new line in SINPOST.DAT for a new signpost. When the character 'E' is entered from the keyboard, SIGNALPC.ASM transmits it to the main computer and then falls back to the infinite loop waiting for another request from the main computer. SIGNALXT.ASM, upon receiving the 'E', closes the file SINPOST.DAT using a regular Fortran read statement. Macro-flowcharts explaining program logic for the two programs are shown in figure 21 and figure 22.

There is no limit to the number of signposts that can be used with the testing system. However, the main program of the dispatching system is limited to function with upto 24 signposts. All data transmission take place using ASCII (American Standard Code for Information Interchange) character codes. In ASCII, seven bit combinations of zero and one make up a character. There are 128 characters defined by the ASCII standard. This standard includes all numbers from 0 - 10 , all English alphabets from A - Z, and some special characters and functions. The cable to connect the two computers must be plugged to port number one, code named COM1 in the IBM

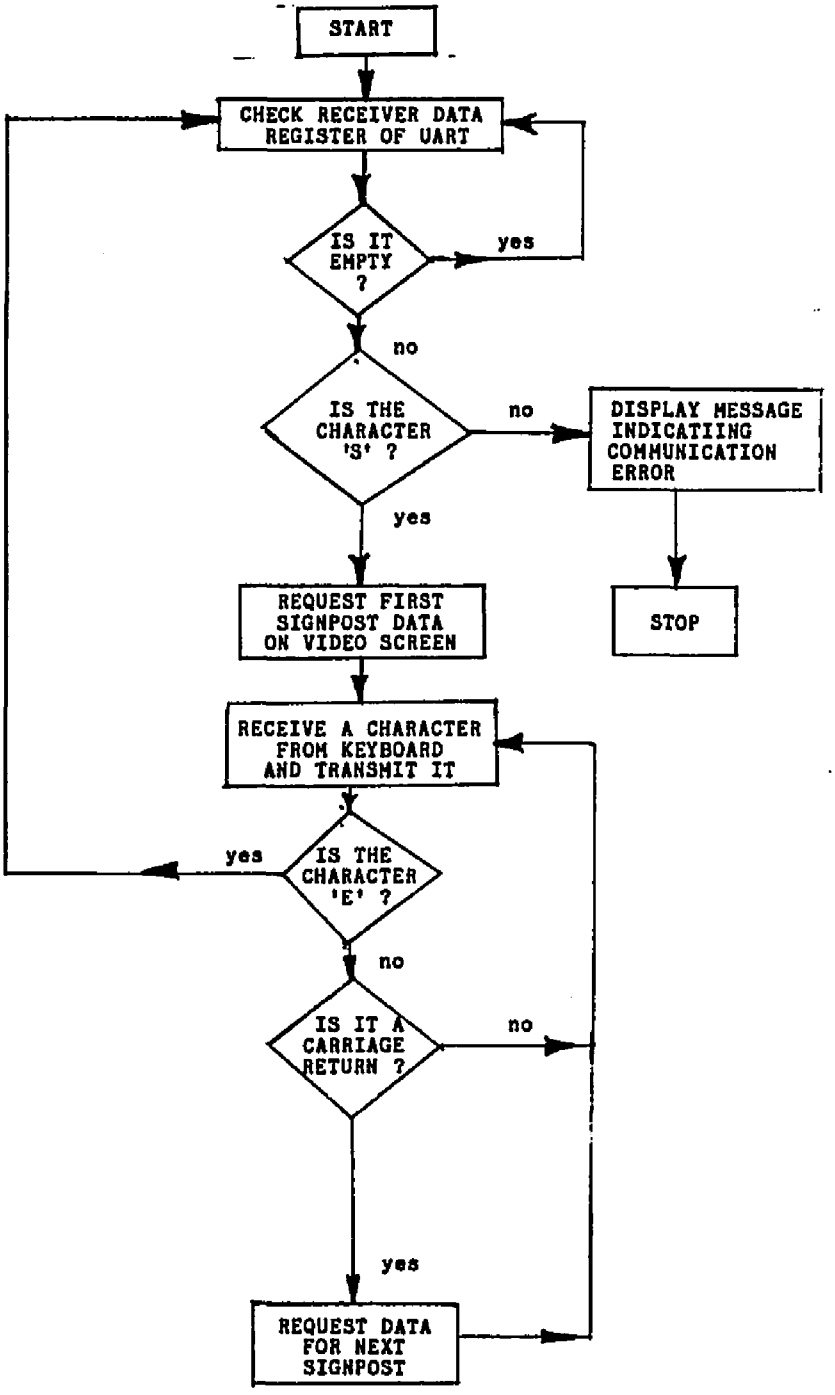

Figure 21. Macroflowchart of SIGNALPC.ASM

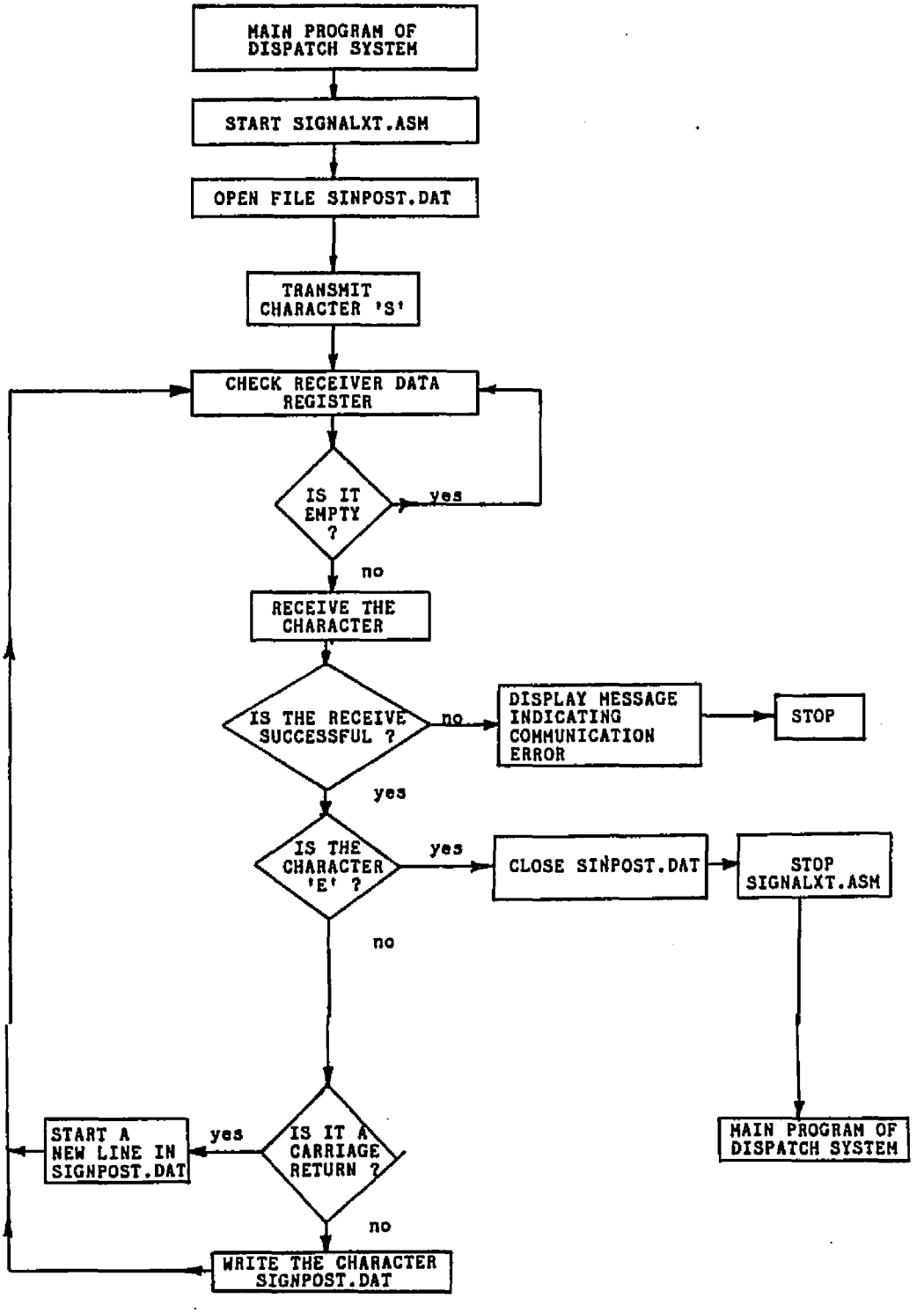

Figure 22. Macroflowchart of SIGNALXT.ASM

Disk Operating System Manual (Version 2.0 1983), of each computer for proper operation.

8.4 Utility subsystem for generating reports

The main program of the dispatching system keeps record of various production statistics as the system is operating on a real time basis. All these statistics are printed out in the form of reports once a shift has ended. It has been mentioned earlier that as control passes from the main program to support programs, variable values are saved in a file. This file is called SYSTEM.BIN and is written in binary. When the main program takes control again it reads back in the values from SYSTEM.BIN to reinitialize the system. Many of the variable values saved in SYSTEM.BIN are records of production statistics.

At the end of a shift, the main program writes out a file called ASYSTEM.BIN or BSYSTEM.BIN or CSYSTEM.BIN depending upon whether it is first, second or third shift. The first shift is assumed to start at seven in the morning and each shift is assumed to be eight hours duration. These end-of-shift files are binary and have the same structure as SYSTEM.BIN. In fact, they are the same SYSTEM.BIN files named differently when a shift ends. Program UTIL which is the driving program

responsible for generating reports, reads in end-of-shift files. It provides the dispatcher with options to print out a short management report or a detailed report or both. Depending upon dispatcher response, UTIL calls subroutines MANGR and DETL. All report generating programs are written in Fortran.

8.5 Management report and detailed report

Subroutine MANGR generates a short management report. In this report, there are two columns for each item. A sample report is shown in figure 23. The first column shows statistics for the current shift and the second column shows statistics for the previous shift. For each variable that is written in the binary file by the main program of the dispatching system, two variables are defined. Statistics of current shift is read into one variable and that of last shift read into the second variable. These variables are named identical to those of the main program except that, the current shift variable names end with letter 'C'and last shift variable names end with letter 'L'.

Program UTIL reads in values of the two sets of variables from two binary files. For example, at the end of the second shift, files BSYSTEM.BIN and ASYSTEM.BIN are read in. BSYSTEM.BIN contains data from the current

MINE MANAGER
SHIFT REPORT
SHIFT : B
TIME : 8: 1
DATE : 6/ 9/1984

	THIS SHIFT	LAST SHIFT
Total Tons Ore	2890.00	3120.00
Total Tons Oxide	.00	.00
Total Tons Waste	2040.00	2530.00
Total Tons Mined	4930.00	5650.00
Waste(Oxide + Waste):Ore Ratio	.71	.81
Truck Down Time (Minutes)	.00	.00
Shovel Down Time (Minutes)	.00	.00
Crusher Down Time (Minutes)	.00	.00
Truck Idle Time - %	1.32	1.41
Shovel Idle Time - %	2.01	1.76
Crusher Idle Time - %	1.67	1.43
Truck Late Time (minutes)	18.00	21.00
Average Grade	.38	.36

Figure 23. Management Report

shift and ASYSTEM.BIN contains data from the previous shift. At the end of first shift, however, there is no previous shift of the same day and file CSYSTEM.BIN from the previous day is read in.

Using information stored in a file called SHIFT.DAT, program UTIL is able open the two binary files needed at the end of a particular shift. SHIFT.DAT is a one line formatted file with the letter 'A', 'B' or 'C'. By reading in SHIFT.DAT first, UTIL is able to open the appropriate binary files. The current shift file is always on logical device 3 and the previous shift file is always on logical device 2. After reading in all variables, UTIL asks the dispatcher on the video screen which report(s) need to be printed out. Subroutines MANGR and DETL are called accordingly and the reports printed. This marks the end of program execution for the dispatch system for one shift. For the next shift, the system has to be restarted by the dispatcher without turning the computer off. If the computer is turned off, superdrive and superspool would have to be created again.

Subroutine DETL prints out the detailed report which has information only for the current shift. The first page of this report is identical to the management report but statistics of previous shift are not printed. This page is followed by a series of production and

operating statistics grouped under subheadings for trucks, shovels, crushers, dumps and truck routes. Finally, there is a section showing shovel digging rates. Sample output for all subheadings are shown in figure 24 and figure 25.

8.6 System flow

System flow is controlled by IBM disk operating system batch commands. These commands are placed in a file with a BAT extension to the filename and are executed sequentially when a filename is entered from the keyboard. Three such batch files are used with the dispatching system.

The first batch file is called AUTOEXEC.BAT. and resides on the hard disk. The special feature of this file is that commands in it are executed automatically after the computer is turned on. Superdrive of 330K and Superspool of 16K is created by commands in AUTOEXEC.BAT. Messages indicating that such creations have taken place successfully appear on the screen. Time and date is obtained from the perpetual clock inside the computer by another command.

The second batch file is called START.BAT. The dispatcher has to type in START to begin execution of commands in this file. START.BAT also resides on hard

TRUCK REPORT

TRUCK ID	TOTAL TONS ORE	OXIDE	WASTE	COMB. TOTAL TONS	NO OF LOADS	AVG. QUEUE TIME	TOTAL QUEUE TIME
901	340.00	.00	.00	340.00	2	2	10
902	.00	.00	170.00	170.00	1	0	0
903	170.00	.00	.00	170.00	1	3	10

SHOVEL REPORT

SHOVEL ID	TOTAL TONS ORE	OXIDE	WASTE	AVERAGE IDLE TIME	TOTAL IDLE TIME	DOWN TIME
11	1870	0	0	0	0	0
12	1020	0	850	3	3	0
13	0	0	1190	4	26	0

CRUSHER REPORT

CRUSHER ID	AVERAGE IDLE TIME	TOTAL IDLE TIME	DOWN TIME	TOTAL TONS
3	1	8	0	2720

Figure 24. Detailed Report-Part One

DUMP REPORT

DUMP ID	AVERAGE IDLE TIME	TOTAL IDLE TIME	TOTAL TONS
1	0	0	0
2	7	31	1020

ROUTE REPORT

ROUTE ID	NO. OF SEGMENTS	AVERAGE TRAVEL TIME
1103	3	11
1202	4	15
1203	4	17
1304	5	17

SHOVEL DIGGING RATES

SHOVEL ID	TRUCK TYPE 1	TRUCK TYPE 2	TRUCK TYPE 3	TRUCK TYPE 4	TRUCK TYPE 5
11	180	150	210	0	0
12	120	100	150	0	0
13	240	210	270	0	0

* All shovel digging time factors are in SECONDS.

END OF REPORT

Figure 25. Detailed Report-Part Two

disk. Commands in this file switch the system over to superdrive designated as drive B and requests the dispatcher to insert the dispatch diskette in the floppy disk drive and to hit a return. The dispatch diskette contains most programs of dispatch system and are copied over to the superdrive. Setting up the dispatch system diskette is discussed later. A Fortran program called SIFT is executed next. This program asks the dispatcher whether the current shift is A, B or C and puts the letter in a file called SHIFT.DAT. This is the file used by the program UTIL as mentioned in the previous section. The last command in START.BAT transfers control to the third batch file DISPAT.BAT.

File DISPAT.BAT is a fairly complex batch file. A DOS batch file is similar to a Job Control Language file of a mainframe computer. However, the set of available commands is far from what is needed to achieve control of the system flow. It lacks several necessary features. Program running within a batch file cannot pass parameters from one another and as a result conditional execution becomes difficult. There is an IF command in the command set which is used in a rather roundabout way to accomplish conditional execution. This IF command can execute commands depending upon whether a file exists or not. Programs in dispatch system create temporary blank

files that act as flags for the IF command. For example, the main program at the end of a shift creates a file called ENDSHIFT.TMP. The statement IF EXIST ENDSHIFT.TMP GOTO NEXT3 transfers control out of a loop to the statement label NEXT3 when ENDSHIFT.TMP is found. The utility subsystem begins at label NEXT3. It must be noted here that this method is necessary only because of lack of proper commands in the DOS command set and causes loss of program efficiency due to opening and closing numerous files. These files serve no purpose other than to act as flags for the IF command.

Permanent records of production statistics are kept in floppy disks. At the end of three shifts, files ASYSTEM.BIN, BSYSTEM.BIN and CSYSTEM.BIN are saved in a floppy disk. For each day of operation of the dispatch system, there would be one floppy disk.

8.7 System startup procedures

The dispatching system is in a package of two floppy disks. The first diskette is called the dispatch diskette and the second one is called support diskette. The dispatch diskette contains programs and data needed for running the system and the support diskette contains programs needed for preparing the system before startup. The support diskette contains files AUTOEXEC.BAT

and START.BAT, programs DATABAS.EXE and SIGNALPC.EXE and programs ASTCLOCK.COM, SUPERDRV.COM and SUPERSPL.COM.

The first step in preparing the system would be to insert the support diskette in the disk drive and copy all programs to the hard disk. Then program DATABAS.EXE should be executed from the hard disk to create a file called SYSTEM.BIN. This file contains initialization parameters for the dispatch system like number of shovels, trucks, crushers and their specifications, route profile etc. This information is to be supplied to DATABAS.EXE by creating a file called SYSTEM.DAT using a text editor. Details of DATABAS.EXE and SYSTEM.DAT are outside the scope of this thesis.

In the second step, the support diskette should be taken out of the disk drive and the dispatch diskette should be put in. File SYSTEM.BIN should be copied to the dispatch diskette from the hard disk and then the diskette should be taken out. The communication cable is to be plugged into number one serial port of each of the two computers.

At this point, the dispatch system is ready to begin operation. It must be made sure that the hard disk has the programs of IBM Disk Operating System Version 2.0. If there is a diskette in drive A of the main computer, it should be taken out. Because the system

defaults to and reads in the operating system from drive C when turned ON, a diskette in the drive will cause an error. The first step of starting the dispatch system is to turn both the computers OFF. A DOS 2.0 diskette should be inserted in drive A and the support diskette should be inserted in drive B. Then the second computer should be turned ON. When it shows the A> prompt, control must be transferred to drive B and the command SIGNALPC must be typed in at the B> prompt. After waiting for ten seconds, the main computer should be turned ON. When the C> prompt appears, the command START should be typed in. After this point, messages coming out on the screen would guide the dispatcher through the system. The dispatching system is designed to operate under operating system DOS 2.0 and higher and in no case would work with an earlier version.

CHAPTER 9

CONCLUSION AND RECOMMENDATIONS

9.1 Conclusion

When the dispatch system operates with data sent in from the second personal computer, transfer of control between programs takes place within the disk operating system command file at a satisfactory speed due to the use of Superdrive; the simulated disk drive within random access memory. A status report may be printed out at dispatcher option while the system is working on something else. The Superspool print buffer allows such printing and program execution at the same time. The dispatch board is displayed and updated on an auxiliary video monitor attached to the main computer. Transfer of control to the auxiliary monitor and back to the main monitor each time the board is updated, is achieved through DOS batch commands.

A number of functions for the dispatching system are carried out through assembly language programs. These include serial communications between two computers, making available the time-of-the-day to other programs of the dispatching system, producing sound effects and

removing the cursor from the auxiliary video monitor. Display of the dispatch board and updating it at regular intervals is done through a Fortran program. The utility subsystem to produce various reports is also programmed in Fortran. System management is done through IBM disk operating system commands placed in batch files. These batch files play a very important role in dispatching system. The programs and processes discussed in this thesis are fully integrated with the main program of dispatching system which is developed as a separate part and form the subject matter of another thesis written by William E. Kolb.

Overall benefits from a microcomputer based dispatching system is the following :

1. Operating effeciency is increased due to reduction of truck and shovel idle time and reduction of empty miles traveled due to better route selection.

2. Reduction of the number of truck shifts needed to meet a certain production target.

3. Automatic recording of various operating statistics and generation of reports at the end of a shift and also within the shift at dispatcher option.

4. Low installation and operating cost of a microcomputer and microprocessor based system over a system based on a minicomputer.

5. Control of ore grade at the crusher and the possibility of ore blending at the crusher as an added benefit.

6. Reduction of safety related problems around the pit that comes with the dispatching system due to continious monitoring and alertness of truck and shovel operators.

7. Control of pit traffic at a central location reducing confusion and enabling better attention to equipment problems.

9.2 Future software and hardware improvements

A more powerful operating system should be considered as an alternative to the IBM-PC disk operating system. It was mentioned in chapter 8 that the current operating system lacks certain features of a job control language and ways have been devised to achieve conditional execution. Although it serves the purpose, it undoubtedly results in inefficiency. A multitasking operating system would probably allow more flexible program execution. A number of UNIX (Copyrighted by AT&T) like multitasking operating system for small computers are available. One such is offered by IBM and another by Microsoft Corporation known as XENIX (Barry and Jacobson 1984; Martin 1984).

The Microsoft Fortran 77 compiler used for dispatching system appears to be thoroughly debugged and performs quite well. This Fortran is powerful and flexible and there was no problem in programming including the use of assembly language routines. But the compatibility of this Fortran with a different operating system needs to be explored.

The first hardware improvement may be the use of a larger auxiliary video monitor for display of the dispatch board. The current monitor allows for 80 columns horizontally and 25 lines vertically. This limits the number of shovels that can be displayed to 10 allowing for one SPARE and one DOWN column. A larger monitor will allow display of more shovels.

The second hardware improvement would be the use of a second printer. At present, there is one daisy wheel printer linked to a parallel port. When the status report is printed at dispatcher option, hardcopy of the dispatch board waits to be printed in the print buffer. A second dot-matrix printer may be linked to an alternate parallel port to solve this problem.

9.3 System integration with actual signposts

Designing signposts to be located in the field and writing software for them will be taken up as an

extension of this research. Care has been taken such that the present software would run after slight modifications with actual signposts. Using serial communications for data input from a second computer, a test of communication software that will be needed to handle data input from signposts is partially accomplished. However, because data is keyed in from a second computer, they are received directly by the program and placed in a file. This is possible because typing in on a keyboard is a rather slow process and experience shows that the time between typing of two keys is sufficient for a program to receive a character, write it in a file and then get ready to receive the next character. With actual signposts, data is likely to arrive too fast for the developed software to operate successfully. An alternative is to build a circular buffer within the receiving program that will hold on to incoming data while an earlier piece of data is being processed. Use of an assisting central processing unit in the form of a second microprocessor to handle incoming data may also be necessary. This microprocessor may be housed either within the main computer or may be a part of a whole new unit.

SELECTED BIBLIOGRAPHY

Arnold M.J. " Benefits of Computer Based Dispatching in Tyrone Mine ", World Mining, April 1983.

Barry Steven H. and Jacobson Randall "XENIX for the IBM PC XT ", Byte, July 1984.

IBM Personal Computer DOS Manual, V2.00 IBM Personal Computer Language Series. IBM Corp., Boca Raton, Florida.

IBM Personal Computer Macro Assembler Manual IBM Personal Computer Language Series, IBM Corp., Boca Raton, Florida.

Hempenstall J. and Hill R. " Bougainville Increases Efficiency with Computers ", World Mining, Feb. 1980.

Kim Young C. Research Proposal to Cyprus Bagdad Mine, College of Mines, University of Arizona, Sept. 1983.

Kolb William E. " Development of a Microcomputer Based Truck Dispatching System ", Research Seminar, Dept. of Mining and Geol. Engg., University of Arizona, 1983.

Martin Charles L. " AT & T Enters The Market ", Personal Computing, July 1984.

Microsoft Fortran 77 User's Manual, V3.13, Microsoft Corp., Belleview, Washington.

Morgan C.L and Waite M. 8086/8088 16-bit Microprocessor Primer, Byte/McGraw Hill, 1982.

Mueller Edward R. " Simplified Dispatching Board Boosts Truck Productivity at Pima ", Mining Engineering, August 1977.

Rakshit A. " Using the IBM Personal Computer for Automatic Truck Dispatching ", Research Seminar, Dept. of Mining and Geol. Engg., University of Arizona, 1984.

Ralston A. and Relly Edwin D. Jr. Encyclopedia of Computer Science and Engineering, pp 973-974, Van Nostrand Reinhold, 1983.

Sargent L. and Shoemaker R. Interfacing Microcomputers to the Real World, Byte/McGraw Hill, 1982.

Sargent L. and Shoemaker R. The IBM Personal Computer from the Inside Out, Byte/Mcgraw Hill, 1984.

Six Pack Plus User's Manual AST Research Inc. Santa Clara, California.

Tetewsky Avram " Benchmarking Fortran Compilers ", Byte, Vol. 9, No. 2, February 1984.

White J.W., Arnold M.J. and Clevenger J.G. " Automated Open Pit Mine Dispatching at Tyrone ", Engineering and Mining Journal, June 1982.

Williams Gregg " The Apple Macintosh Computer ", Byte, Vol. 9, No. 2, February 1984.

Willen David C. and Krantz Jeffrey I. 8088 Assembler Language Programming : The IBM PC, Howard Sams, 1983.