The Performance of the 8089 Integrated I/O Processor in iAPX 86 Microcomputer Systems

1983

Jeffrey A. Lohman
University of Central Florida

Find similar works at: http://stars.library.ucf.edu/rtd

University of Central Florida Libraries http://library.ucf.edu

Part of the Engineering Commons

STARS Citation

http://stars.library.ucf.edu/rtd/699

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
THE PERFORMANCE OF THE 8089 INTEGRATED
I/O PROCESSOR IN iAPX 86 MICROCOMPUTER SYSTEMS

BY

JEFFREY ALAN LOHMAN
B.S.E., University of Central Florida, 1982

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Engineering
in the Graduate Studies Program of the
College of Engineering
University of Central Florida
Orlando, Florida

Spring Term
1983
ABSTRACT

This thesis examines the performance of the Intel 8089 integrated I/O processor through a predictive performance model for the I/O subsystem architectures available to the designer of an iAPX 86 system. The model provides system throughput estimates and is intended to be used prior to any detailed design. The derivation of the model is followed by a description of a prototype system which is used to provide actual throughput measurements. These measurements are compared with the model predictions to evaluate the model error and its utility. The model estimates are then combined with subsystem cost data to gauge the cost-effectiveness of the 8089.
ACKNOWLEDGEMENT

The author is especially grateful for the advice and guidance of Dr. Herbert C. Towle and also wishes to thank Dr. Brian E. Petrasko and Mr. Albert H. Marshall for their assistance in the preparation of this thesis.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
</tbody>
</table>

## Chapter

1. INTRODUCTION                  | 1                      |
2. THE PERFORMANCE MODEL         | 4                      |
3. THE DEMONSTRATION SYSTEM      | 36                     |
4. RESULTS                       | 53                     |
5. CONCLUSION                    | 61                     |

## Appendices

A. 8089 OVERVIEW                 | 69                     |
B. MODEL INSTRUCTION SEQUENCES   | 71                     |
C. CPU IDLE TIME DUE TO IOP BUS ACCESS | 81                   |
D. DEMONSTRATION SOFTWARE LISTINGS | 88                    |

REFERENCES                      | 148                    |
LIST OF TABLES

1. Measured Throughput for Tj = 0 and Variable P.  . 54
2. Measured Throughput for P = 0.44 and Variable Tj 54
3. Predicted Throughput for Tj = 0 and Variable P . 56
4. Interrupt Tio and Ri for P = 0.44 and Variable Tj 56
5. Predicted Throughput for P = 0.44 and Variable Tj 57
6. Model Error for Tj = 0 and Variable P.  . 57
7. Model Error for P = 0.44 and Variable Tj .  . 59
8. Relative CPU Overheads as a Function of F.  . 59
9. Relative CPU Overheads as a Function of Tj .  . 63
10. Relative CPU Overheads as a Function of N.  . 63
11. Component Cost of Architectures.  .  . 65
12. Task Step Execution Times for Polled I/O . . . 78
13. Task Step Execution Times for Local IOP. . . . 78
14. Task Step Execution Times for Interrupt I/O. . 79
15. Task Step Execution Times for Remote IOP . . . 80
LIST OF FIGURES

1. Polled I/O Architecture .......................... 5
2. Interrupt I/O Architecture ......................... 5
3. Local IOP Architecture ............................. 7
4. Remote IOP Architecture ............................ 7
5. A Real-Time Task Model ............................. 12
6. Formal Description of Task Model ................... 12
7. Polled I/O Task for Low Transfer Rates .......... 16
8. Revised Input Procedure for Polled I/O Task With High Transfer Rates ........................... 18
9. Interrupt I/O Task for Low Transfer Rates ........ 20
10. Revised Interrupt Procedure for Interrupt I/O Task With High Transfer Rates ..................... 21
11. Local IOP Task .......................... 25
12. Device-Dependent Steps for No I/O Processing .... 27
13. Device-Dependent Steps for Low Transfer Rates With I/O Processing .......................... 27
14. Device-Dependent Steps for High Transfer Rates With I/O Processing .......................... 28
15. Revised Device-Dependent Steps for Low Transfer Rates With I/O Processing .................... 32
16. Revised Channel Program for Remote IOP With High Transfer Rates and I/O Processing ........... 33
17. The Demonstration System ......................... 37
18. Simplified 86/12A Configuration ........ 39
19. Dual-Port RAM Mapping ................ 40
20. 86/12A Jumpers and Switch-Settings. .... 42
21. Remote I/O Processor Board Schematic. ... 43
22. Demonstration System Software Modules ... 51
CHAPTER I
INTRODUCTION

The concept of an intelligent processor dedicated to performing low-level, device-dependent I/O activity originated in the 1960s with the peripheral processors in the CDC-6600 (1) and the I/O channels in the IBM System/360 (2). Today, I/O processors are available as standard equipment on nearly all mainframes and minicomputers. In general, the I/O processor interfaces to either a few high-speed devices, such as disk drives, or many low-speed devices, such as terminals. It also has access to main memory, where it fetches its instructions from a channel program. Because the I/O processor and the central processing unit can operate concurrently, systems with I/O processors generally have higher throughputs. Intel Corporation markets an integrated I/O processor which makes higher throughput available to microcomputer systems.

Intel's iAPX 86 family consists of three major components: the 8086 central processing unit (CPU), the 8087 numeric data processor (NDP), and the 8089
I/O processor (IOP). A performance model for the 8089 is obtained which enables the designer of an 8086 system to quickly estimate the cost-effectiveness of the 8089 and employ either an 8089, interrupt, or polled I/O subsystem. This entailed analyzing the iAPX 86 system to derive the performance model and constructing a prototype system to verify the model.

The performance of systems with CPU and IOP overlap has been extensively analyzed elsewhere (3,4). However, these analyses have concentrated on systems with a multitasking CPU in a batch environment to determine the impact of the IOP on job throughput and response time. Since batch-oriented multitasking microcomputer systems are rare, the results of these analyses are of limited utility; consequently, the performance model is oriented toward the real-time microcomputer system with one job.

The model allows the throughput of 8086-based microcomputers with polled, interrupt, and 8089 I/O subsystems to be estimated. The model can be used prior to any detailed hardware or software design. It is not intended to yield exact measures of throughput which will be realized in practice. Instead, it is anticipated that the designer of an 8086-based system will use the throughput estimates and rough cost data to select a particular I/O subsystem
architecture which meets system performance requirements.

Once the performance model was derived, a prototype 8086 system was constructed which can measure the throughput of polled, interrupt, and 8089 I/O subsystems. The system was constrained to use available hardware wherever possible and a high-level, structured language. It was not tailored to provide an exact fit to the model; in fact, some of the design constraints conflict sharply with assumptions made in deriving the model. Throughput measurements are compared to model predictions to gauge the error in the model and its overall utility.
CHAPTER II

THE PERFORMANCE MODEL

This chapter traces the development of the performance model. The I/O subsystem architectures available to the designer are examined, and a relationship between the throughputs of a real-time task with two different subsystems is developed. A model of a real-time task is then presented from which expressions are derived that allow the task throughput of any I/O subsystem architecture to be estimated from a few, simple task parameters.

I/O Subsystem Architectures

This section outlines the I/O subsystem architectures which can be realized with iAPX 86 components. For those readers not familiar with the 8089, a brief description is presented in Appendix A, while more detailed information can be found in the iAPX 86,88 User's Manual (5).

Polled I/O

The architecture of a polled I/O subsystem is illustrated in Figure 1 using PMS notation (6). In this architecture, the CPU is responsible for preparing
Fig. 1. Polled I/O architecture.

Fig. 2. Interrupt I/O architecture.
the device controllers for data transfers, and actually performing the transfer by repeatedly checking device status information.

**Interrupt I/O**

The architecture of an interrupt I/O subsystem is shown in Figure 2. Each device controller has an interrupt request line connected to the interrupt controller. The interrupt controller multiplexes the interrupt requests onto the CPU's maskable interrupt request line. The interrupt controller maintains individual masks for the interrupt request lines and the priority of the requests. The CPU is responsible for programming the interrupt controller, initializing the device controllers, and error recovery. However, it does not check device status information when performing a data transfer. Instead, the device controller interrupts the CPU when it is ready to transfer the next byte of data.

**Local IOP**

The architecture of a local IOP subsystem is shown in Figure 3. In the local configuration, the IOP shares the Multibus (7) and the I/O bus with the CPU. The CPU and the IOP arbitrate the possession of these buses through a protocol of pulses on the request/grant line.
Fig. 3. Local IOP architecture.

Fig. 4. Remote IOP architecture.
The IOP performs all data transfers to and from the device controllers. Its operation is governed by a channel program of IOP instructions resident in main memory. When a CPU task requires I/O, it signals the IOP to start executing the appropriate channel program by preparing a channel control block and parameter block, and then toggling the channel attention line. The IOP acquires possession of the buses through the request/grant line and reads the channel control and parameter blocks which control the execution of the channel program. When the channel program is completed, the buses are returned to the CPU.

In this configuration, the IOP is used as an intelligent DMA controller to perform all device transfers. It also performs device controller initialization and error recovery.

Remote IOP

The architecture of a remote IOP subsystem is shown in Figure 4. In the remote configuration, the IOP shares only the Multibus with the CPU. The IOP has a private I/O bus which provides a path to all device controllers and a private IOP memory. As above, the IOP is controlled by a channel program which can reside either in main memory or in the private IOP memory. The IOP performs the same functions as in the local configuration; however, the CPU and the IOP
arbitrate possession of the Multibus through 8289 bus arbiters (8).

The Throughput Relation

To compare the performance of a microcomputer with two different I/O subsystem architectures, a relationship between the throughput of a reference I/O subsystem and a new I/O subsystem is needed.

If a real-time task in the reference system has an average throughput $T$ when I/O consumes a fraction $P$ of the available CPU time, then with a constant of proportionality $K$:

$$T = K(1 - P) \quad \text{for} \quad 0 \leq P \leq 1 \quad (1)$$

In this context, throughput refers to the average rate at which the CPU provides or receives data for an input or an output stream of the task. The quantity $P$, referred to as the fractional CPU overhead, includes the effects of device initialization time, data transfer time, and formatting. In other words, $P$ accounts for the CPU time required to transfer elements of a task output stream to the destination of the output stream in the format required by the destination, or the CPU time required to transfer elements of a device input stream from the format of the source to the format required by the task input stream.
If a new system is designed in which an I/O subsystem is employed that results in a fractional CPU overhead of \( P' \), then:

\[
T' = K(1 - P') \quad \text{for} \quad 0 \leq P' \leq 1 \quad (2)
\]

\[
R = \frac{P'}{P} \quad (3)
\]

\[
T' = K(1 - RP) \quad (4)
\]

\[
\frac{T'}{T} = \frac{(1 - RP)}{(1 - P)} \quad (5)
\]

Here \( R \) is referred to as the relative CPU overhead. Having \( R \) and \( P \), the fractional CPU overhead of a task with the reference I/O subsystem, the designer of a real-time system can obtain an estimate of the throughput gain for a new I/O subsystem using equation 5.

It should be noted that the throughput gain is limited by the amount of I/O performed in the reference system. Considering a hypothetical I/O subsystem which requires no CPU overhead, the maximum throughput gain can be found by taking the limit of equation 5 as \( R \) approaches zero:

\[
\lim_{R \to 0} \left( \frac{T'}{T} \right) = \frac{1}{1 - P} \quad (6)
\]

Thus the maximum throughput gain is determined by the fractional CPU overhead of the original system. For typical systems with \( P \) in the range 0.1 to 0.3, the maximum throughput gain is roughly 10% to 30%.
The Real-Time Task Model

It is assumed that the fractional CPU overhead of the reference subsystem, \( P \), can be estimated or calculated from an existing system. The relative CPU overhead, \( R \), of a task with a new I/O subsystem can only be calculated by comparing the CPU overhead of a common task with each I/O subsystem. This section presents a real-time task model which facilitates this comparison.

Any real-time task can be decomposed into a series of nested, smaller tasks, where a task includes data input, data processing, and data output functions. A task model is shown schematically in Figure 5, while the BNF notation for this model is shown in Figure 6. In this model, a task is viewed as a module of code which takes data from input streams and produces one or more output streams. Accordingly, the task consists of two parts. The device initialization block is concerned with preparing the I/O devices and their controllers for the needs of the second part of the task, the cycle. The cycle consists of three functional blocks which may be executed any number of times. The data input block captures the data input streams, while the data processing block produces the output streams, and the data output block transmits the output streams to their destination. At least one of these blocks must be present in the cycle.
Fig. 5. A real-time task model.

<real-time task> := [<initialization>] <cycle>
<cycle> := [<input>] [<processing>] [<output>]
<initialization> := <instructions for device initialization> ...
<input> := <instructions for data input from devices> ...
<output> := <instructions for data output to devices> ...
<processing> := <instructions for generating data output streams from data input streams> ...
<task> ...

Fig. 6. Formal description of task model.
The data processing block may also contain tasks, so the model is recursive, which allows the model to encompass all real-time tasks.

From the previous section, the throughput gain for any subsystem architecture is a function of the relative CPU overhead, \( R \), for the subsystem, and the fractional CPU overhead, \( P \), for the reference subsystem. Furthermore, \( R \) is the ratio of the fractional CPU overheads. The execution time of a real-time task is fixed; therefore, the ratio of the fractional CPU overheads is identical to the ratio of the absolute CPU overheads, or the ratio of the total amount of CPU time spent in the data input, output, and initialization blocks of the model. Given a specific task, expressions for the absolute CPU overhead, \( T_{io} \), can be derived for each I/O subsystem architecture.

The purpose of the performance model is to provide estimates of the throughput gain without detailed hardware and software design. Consequently, the \( T_{io} \) expressions may only be a function of parameters which are known or can be accurately estimated prior to design.

In general, a task will include input and output streams of \( N \) bytes each. There will be \( M \) task cycles with device controller transfer rates of \( F \) bytes per second. The amount of I/O related processing performed on each byte of the input and output streams can be
denoted $T_j$. For the calculation of CPU overhead, $N$, $M$, $F$, and $T_j$ are sufficient to characterize a task. Furthermore, only $N$, $M$, $F$, $T_j$, and $P$ must be specified to estimate the throughput gain of an I/O subsystem. These are variables which are either known or can be accurately estimated prior to design.

Because a task may use different I/O devices for input and output, the expressions for $T_{io}$ will be very complicated if both input and output streams are considered. Consequently, the $T_{io}$ expressions will be derived for a task with one input stream only. However, the same $T_{io}$ expressions can be used for a task with only one output stream, since the absolute overhead for this task will also be a function of $N$, $M$, $F$, and $T_j$, although these may have different values. In fact, the total overhead for a task with multiple input streams and multiple output streams can be calculated by summing the overheads of each individual stream using the $N$, $M$, $F$, and $T_j$ applicable to each stream. In other words, the total overhead can be found by applying superposition using the same $T_{io}$ expression.

**Absolute CPU Overhead Relations**

In this section, expressions for $T_{io}$ as a function of task characteristics $M$, $N$, $F$, and $T_j$ are derived. For each architecture, the software required to
implement a task with these characteristics is presented in flowchart form. The time required to execute each step of the task is initially variable, and Tio is expressed as a function of these variables and the task characteristics. The task steps are then realized with instruction sequences that are typical of actual practice. The execution times of these sequences are used to evaluate the variables, resulting in Tio expressions that are a function only of the task characteristics.

Polled I/O

For devices with low transfer rates, any I/O processing can be executed while the I/O device is busy. This results in the flowchart of Figure 7. Here the task data input function is implemented with a call to a procedure which serves the needs of several identical device controllers. Consequently, the procedure parameters are stack-based and include N, the device controller data and status port numbers, and a pointer to a buffer for the N input bytes. The remainder of the task is straightforward, with the time required to execute each step listed below the step. For devices with higher transfer rates, I/O processing cannot be performed while the I/O device is busy; consequently, the procedure is modified to include two loops. The first loop simply inputs the N bytes
Fig. 7. Polled I/O task for low transfer rates.
and stores them in a buffer. The second loop steps through the buffer processing each byte. This flowchart appears in Figure 8. Using Figure 7, \( T_{io} \) for the low transfer rate case is:

\[
T_{io} = \left( \frac{T_{sd}}{M} \right) + T_{pc} + T_{save} + T_{load} + T_{rest} + T_{ret} + \frac{(N/F)}{7} \tag{7}
\]

for

\[
1/F \geq T + T_{j} + T_{si} + T_{lc} + T_{w} \tag{8}
\]

Tw denotes the minimum time required to test device status. The device initialization time, \( T_{sd} \), is distributed evenly over \( M \) cycles to include its contribution to \( T_{io} \). Using Figures 7 and 8, \( T_{io} \) for the high transfer rate case is:

\[
T_{io} = \left( \frac{T_{sd}}{M} \right) + T_{pc} + T_{save} + T_{load} + \frac{(N/F)}{10} + T_{load} + (T_{mov} + T_{j} + T_{si} + T_{lc})N + T_{rest} + T_{ret} \tag{9}
\]

for

\[
1/F \geq T_{w} + T + T_{si} + T_{lc} \tag{10}
\]

The execution times of the task steps in Figures 7 and 8 are evaluated in Appendix B using typical instruction sequences for each step. Using the results of Appendix B, equations 7 through 10 become:

\[
T_{io} = \left( \frac{42}{M} \right) + \frac{(N/F)}{202} \text{ clocks} \tag{11}
\]

for
Fig. 8. Revised input procedure for polled I/O task with high transfer rates.
\[ F \leq \frac{1}{T_j - \left(\frac{27}{N} + 107\right)} \text{ bytes/clock} \quad (12) \]

and,

\[ T_{\text{io}} = \left(\frac{42}{M}\right) + \left(\frac{N}{F}\right) + (59 + T_j)N + 209 \text{ clocks} \quad (13) \]

for

\[ F \leq \frac{1}{107 - \left(\frac{27}{N}\right)} \text{ bytes/clock} \quad (14) \]

where,

- \text{clocks} = \text{the number of CPU clock periods}
- \text{Tj} = \text{I/O processing time per byte of input in clocks}

**Interrupt I/O**

For devices with low transfer rates, any I/O processing can be performed as each byte is input. For higher transfer rates, the interrupt frequency is too great, and only the actual transfer can be performed after an interrupt. In this case, I/O processing is accomplished by calling a procedure to process the entire string after the last byte is input. Figure 9 shows the flowchart for the task with low transfer rates, while Figure 10 shows the modified interrupt procedure for higher transfer rates.

The task first sends a new mask to the 8259 interrupt controller (8), where the 8259 is assumed to have been initialized earlier. The task then loads a buffer pointer, N, and the device controller data and status port numbers into an area of memory known
Task Start

New Mask to 8259A (Tseli)

Cycle Start

Fill Int. Proc. Parm Block (Tstore1) or (Tstore2)

Enable Device Interrupts (Ten)

Data Processing

M Cycles?

N

Y

Task End

Save CPU Registers (Tsave)

Remove Interrupt (Trem)

EOI Command (Teoi)

Load Parameters (Tload)

Input a Byte (T)

Process a Byte (Tj)

Store the Byte (Tsi)

N

Bytes? (Tic)

Y

Restore CPU Registers (Trest)

Disable Interrupts (Tdis)

Return (Tiret)

Interrupt Procedure

Fig. 9. Interrupt I/O task for low transfer rates.
Fig. 10. Revised interrupt procedure for interrupt I/O task with high transfer rates.
to the interrupt procedure. In the case of Figure 10, it additionally copies the buffer pointer and N into another area because the interrupt procedure destroys the first copies. The device interrupts are then enabled, where it is assumed that the CPU interrupt system is already enabled.

The interrupt procedure in Figure 9 first saves the CPU registers and commands the device controller to remove its interrupt request. It then sends an end of interrupt (EOI) command to the 8259 to cause it to reset the in-service flag for the interrupt request. The remainder of Figure 9 is similar to Figure 7. In Figure 10, after N bytes have been input, the device interrupts are disabled, and a processing procedure is called with the buffer pointer and N as parameters. Upon return, the interrupt procedure is completed.

Using Figures 9 and 10, $T_{io}$ is given by:

$$T_{io} = (T_{ip} + T_{save} + T_{rem} + T_{eoi} + T_{load} + T + T_{j} + T_{si} + T_{rest} + T_{ret} + T_{ic})N + T_{dis} + T_{store1} + T_{en} + (T_{sdi}/M)$$

for

$$\frac{1}{F} \geq T_{ip} + T_{id} + T_{save} + T_{rem} + T_{eoi} + T_{load} + T + T_{j} + T_{si} + T_{ic} + T_{rest} + T_{ret}$$

and,

$$1/F \geq (T_{ip} + T_{id} + T_{save} + T_{rem} + T_{eoi} + T_{load} + T + T_{j} + T_{si} + T_{ic} + T_{rest} + T_{ret})$$
$$T_{io} = (T_{ip} + T_{save} + T_{rem} + T_{eoi} + T +$$
$$T_{si} + T_{ic} + T_{rest} + T_{iret})N + T_{dis} +$$
$$T_{pc} + T_{loadp} + (T_{mov} + T_{j} + T_{si} +$$
$$T_{lc})N + T_{ret} + T_{store2} + T_{en} +$$
$$(T_{sdi}/M)$$  \hspace{1cm} (17)

for

$$1/F \geq T_{ip} + T_{id} + T_{save} + T_{rem} + T_{eoi} +$$
$$T_{load} + T + T_{si} + T_{ic} + T_{rest} +$$
$$T_{iret}$$  \hspace{1cm} (18)

where,

$$T_{ip} = \text{time required by 8086 to jump to}$$
$$\text{the interrupt procedure}$$

$$T_{id} = \text{interrupt request delay from 8259}$$
$$\text{request input to 8086 request input}$$

The execution times of the steps in Figures 9 and 10
are derived in Appendix B, which results in the $T_{io}$
expressions:

$$T_{io} = (305 + T_{j})N + 92 + (56/M) \text{ clocks}$$  \hspace{1cm} (19)

for

$$F \leq 1/(307 + T_{j}) \text{ bytes/clock}$$  \hspace{1cm} (20)

and,

$$T_{io} = (364 + T_{j})N + 225 + (56/M) \text{ clocks}$$  \hspace{1cm} (21)

for

$$F \leq 1/307 \text{ bytes/clock}$$  \hspace{1cm} (22)
Local IOP

When an 8089 is employed in an I/O subsystem, Tio is the result of two effects: actual 8086 I/O related software, and CPU time lost when the 8086 is idled by the loss of bus possession. Accordingly, the flowchart for this configuration in Figure 11 considers both the task software on the 8086, and the 8089 channel program.

The 8086 is used to initialize the device controller, since a second channel program would be necessary otherwise, and this would result in greater CPU overhead. The CPU then loads the IOP parameter block. After the cycle starts, the channel control block is filled and a channel attention is issued. While the channel control block could be filled before the cycle starts in this particular case, in general the IOP may be used several times during a task. Because the location of the channel control block is fixed, it must be shared by each invocation of a channel program. This assumption maintains the validity of the resulting Tio expression when it is applied to a system with both input and output streams. Furthermore, in this example, the IOP is assumed to have been previously initialized.

The steps outlined above account for Tio due to CPU I/O related software. The IOP channel program first
Fig. 11. Local IOP task.
executes the channel command word (CCW) in the channel control block. It then loads the necessary registers. The actual transfer and processing instructions which follow are dependent upon the device transfer rate. Finally, the channel program executes a halt instruction. In this case, the channel program is assumed to complete before the input stream is needed, so the CPU does not need to check the state of the IOP busy flag before accessing the input data.

The device-dependent steps are outlined in Figures 12, 13, and 14. For input with no I/O processing, these steps consist of initializing the appropriate IOP registers for a DMA transfer, then executing the XFER and WID instructions to start the transfer. Following this, the entire string is input using DMA. After the last byte is acquired, the IOP executes the DMA termination sequence, as in Figure 12. For I/O processing with slow devices, single-cycle DMA is used to transfer a single byte, which is then retrieved, processed and stored, with this sequence being executed N times, as shown in Figure 13. Finally, in Figure 14, for fast devices, the entire string is input using DMA, and after the last byte, the IOP steps through the array processing each byte in turn.
Fig. 12. Device-dependent steps for no I/O processing.

Fig. 13. Device-dependent steps for low transfer rates with I/O processing.
Fig. 14. Device-dependent steps for high transfer rates with I/O processing.
The impact of IOP instructions on Tio is through bus contention. Because the CPU has a small queue, or instruction look-ahead buffer, it can continue execution when it has lost bus possession until the queue is empty or it requires memory access for data. In this situation, the CPU time lost, \( f(x) \), is a function of the number of consecutive clock periods, \( x \), during which the bus was unavailable to the CPU. An expression for \( f(x) \) is derived in Appendix C. Using Figures 11 and 12 along with Appendix C:

\[
Tio = \frac{(Tds + Tpb)}{M} + Tcb + Tca + 
\begin{align*}
&f(Tccw + Tload + Txfer) + \frac{N}{2}f(4) + \\
&(\frac{N}{2})f(8) + f(Tterm + Thlt) 
\end{align*}  \tag{23}
\]

for

\[ F \leq \frac{1}{20} \text{ bytes/clock} \quad \text{(see Appendix C)} \tag{24} \]

Here \( f(4) \) is the number of idle CPU clock periods due to the four consecutive clocks that the IOP needs to obtain the first byte of a pair of input bytes, and \( f(8) \) is the number of idle clocks due to the eight clocks needed to obtain the second byte of a pair and store the resulting word. Similarly, for Figure 13:

\[
Tio = \frac{(Tds + Tpb)}{M} + Tcb + Tca + 
\begin{align*}
&f(Tccw + Tload) + Nf(8 + Tterm + \\
&Tmov + Tj + Tst + Tlc + Txfer) + \\
&f(Thlt) 
\end{align*}  \tag{25}
\]
for 
\[ 1/F \geq 8 + T_{term} + T_{mov} + T_j + T_{st} + T_{lc} + T_{xfer} \] (26)

Using Figure 14, Tio is given by:

\[
Tio = ((T_{ds} + T_{pb})/M) + T_{cb} + T_{ca} + \frac{f(T_{ccw} + T_{load} + T_{xfer})}{2} + (N/2)f(4) + \frac{(N/2)f(8) + f(T_{term} + T_{reload})}{2} + Nf(T_{mov} + T_j + T_{si} + T_{dec} + T_{lc}) + f(T_{hlt})
\] (27)

for
\[ F \leq 1/20 \text{ bytes/clock} \] (28)

The execution times of these steps are evaluated in Appendix B, with the resulting Tio expressions:

\[
Tio = (145/M) + 325 + 4.1N \text{ clocks}
\] (29)

for
\[ F \leq 1/20 \text{ bytes/clock} \text{ and } T_j = 0 \] (30)

and,

\[
Tio = (145/M) + 283 + (116 + T_j)N \text{ clocks}
\] (31)

for
\[ F \leq 1/(118 + T_j) \text{ bytes/clock} \] (32)

and,

\[
Tio = (145/M) + 376 + (77 + T_j)N \text{ clocks}
\] (33)

for
\[ F \leq 1/20 \text{ bytes/clock} \] (34)
Remote IOP

The same flowcharts can be used for the remote configuration that were used for the local configuration. There are, however, some exceptions. First the IOP must initialize the device controller because the CPU does not have access to it. The initialization will be performed each time the channel program is invoked, because a separate channel program for initialization would require more CPU overhead. With the remote configuration, the CPU is idle only when the IOP accesses the Multibus. Thus, CPU overhead can be reduced in processing input from slow devices by using single-cycle DMA to input to a fixed IOP memory location. This input is then processed and stored in main memory. These revised device-dependent steps are shown in Figure 15, which supersedes Figure 13. For fast devices, Figure 14 is still used, except that the buffer pointer and N, passed to the IOP in the parameter block are copied into the IOP memory at the load step. These revisions are shown in Figure 16.

It should be noted that in the remote configuration, the channel and parameter blocks are located in main memory, while the IOP task block is located in the IOP memory. Access to the IOP memory does not cause contention on the Multibus. Using this information with Figures 11, 12, 15, and 16, Tio is given by:
Fig. 15. Revised device-dependent steps for low transfer rates with I/O processing.
Fig. 16. Revised channel program for remote IOP with high transfer rates and I/O processing.
\[ T_{io} = \frac{T_{pb}}{M} + T_{cb} + T_{ca} + g(T_{ccw}) + \\
g(T_{load}) + \frac{N}{2}f(4) + f(4) \]  (35)

for
\[ F \leq \frac{1}{20} \text{ bytes/clock and } T_j = 0 \]  (36)

and,
\[ T_{io} = \frac{T_{pb}}{M} + T_{cb} + T_{ca} + g(T_{ccw}) + \\
g(T_{load}) + Nf(4) + f(4) \]  (37)

for
\[ \frac{1}{F} \geq 8 + T_{term} + T_{iomov} + T_j + T_{st} + \\
T_{lc} + T_{xfer} \]  (38)

and,
\[ T_{io} = \frac{T_{pb}}{M} + T_{cb} + T_{ca} + g(T_{ccw}) + \\
g(T_{load}) + \frac{N}{2}f(4) + Nf(4) + \\
Nf(4) + f(4) \]  (39)

for
\[ F \leq \frac{1}{20} \text{ bytes/clock} \]  (40)

Here \( g(x) \) denotes the time that the CPU is idle due to
the number of clocks included in \( x \) that are devoted to
Multibus access. The execution times for the steps in
these equations are calculated in Appendix B, with the
resulting \( T_{io} \) expressions:

\[ T_{io} = \frac{103}{M} + 98 + 1.1N \text{ clocks} \]  (41)

for
\[ F \leq 1/20 \text{ bytes/clock and } T_j = 0 \]  (42)

and,
\[ T_{io} = \frac{103}{M} + 98 + 2.2N \text{ clocks} \]  (43)

for

\[ F \leq \frac{1}{(124 + T_j)} \text{ bytes/clock} \]  \hspace{1cm} (44)

and,

\[ T_{io} = \frac{103}{M} + 98 + 5.6N \text{ clocks} \]  \hspace{1cm} (45)

for

\[ F \leq \frac{1}{20} \text{ bytes/clock} \]  \hspace{1cm} (46)
CHAPTER III
THE DEMONSTRATION SYSTEM

A system was constructed to evaluate the accuracy of the performance model. The system was constrained to use readily available parts. Consequently, as shown in Figure 17, an iSBC 86/12A single-board computer (9) was used to implement the CPU and provide main memory. A remote I/O processor board was constructed using an 8089. These boards are plugged into a Multibus card cage, and both are provided an RS232C connection to an ADM-3A terminal (10). The 86/12 can communicate with the terminal directly through polled or interrupt I/O, and indirectly through the remote IOP. A local IOP is not supported by the 86/12 board; therefore, the model can only be verified for polled, interrupt, and remote IOP subsystems.

The 86/12A Single-Board Computer

The 86/12A consists of a 5 MHz 8086 CPU with 16K bytes of EPROM, 32K bytes of RAM, an 8259A Programmable Interrupt Controller (PIC), 8253 Programmable Interval Timer (PIT), 8255 Programmable Peripheral Interface (PPI), and an 8251A USART. The 86/12A is
Fig. 17. The demonstration system.
provided with a large number of wire-wrap jumpers and switches to allow it to be configured to meet application requirements. A simplified diagram of the configuration used in this system is shown in Figure 18.

The RAM is dual-ported, that is the RAM can be accessed by the CPU and other bus-masters over the Multibus. An 8K block of RAM is available to the remote I/O processor board starting at location 0 in the system address space and location 6000H (Hexadecimal) in the on-board CPU's address space. This is clarified in Figure 19.

The USART drives the serial connector to the ADM-3A, with one of the three 16-bit down counters in the PIT generating the baud rate. The other two counters are cascaded to provide a CPU time out interrupt. The first counter divides the input frequency to produce a 1kHz input to the next counter. This counter is initially loaded with the time out period in milliseconds. When this count reaches zero, the counter asserts an interrupt request input of the PIC, which vectors the CPU to an interrupt service routine. The PIC has eight interrupt request inputs which can be masked and prioritized dynamically. The USART also drives an interrupt request line, TxRDY, which can be programmed to interrupt when the USART is ready to transmit a byte. This is used to implement interrupt I/O. The Multibus interface employs
Fig. 18. Simplified 86/12A configuration.
Fig. 19. Dual-port RAM mapping.
the serial-priority resolution technique (11) for bus arbitration, with the 86/12 as highest priority master. The demonstration software resides in the EPROM. The complete list of non-default jumpers and switch-settings is shown in Figure 20.

Remote IOP Board

The schematic of the remote IOP board is shown in Figure 21. The 8089 interfaces to the Multibus and a private I/O bus. The 8289 bus arbiter regulates IOP access to the Multibus, while an 8288 bus controller provides the command outputs for both of the buses. Because only 8K of main memory is available to the IOP over the Multibus, only the lower 13 address lines of the Multibus are driven by the IOP board. The remaining 7 address lines are pulled up by the Multibus card cage. Thus, only two 8212 latches are needed to capture the Multibus address. These latches are loaded with the inverse of the 8089 address lines at the negative edge of the address latch enable (ALE) signal. This is because the Multibus address lines are active-low. Two 8287 inverting bus transceivers interface the 8089 data bus with the active-low Multibus data lines. The private I/O bus is eight bits wide. In this mode, the 8089 does not multiplex the eight high-order address bits. Therefore, only one 8212 is necessary to latch the lower
1. Use 2732A-3 EPROMs
   a. Jumper 94 to 96 and 97 to 99
   b. Set switch S1 8 to 9 open and 7 to 10 closed

2. Connect bus clock to Multibus
   a. Jumper 105 to 106

3. Connect bus priority out to Multibus
   a. Jumper 151 to 152

4. Connect common bus request to Multibus and ground any request
   a. Jumper 144 to 145 and 131 to 130

5. Setup timers 0 and 1 in series with PPI port C bit 7 connected to the gates
   a. Jumper 59 to 61, 57 to 56, 8 to 13, and 10 to 13

6. Connect interrupt sources to PIC
   a. Jumper 88 to 81, 91 to 80, 83 to 79, 90 to 78, 82 to 77, 72 to 76, and 71 to 75

7. Implement dual-port RAM mapping
   a. Jumper 127 to 128
   b. Set switch S1 6 to 11 closed, 5 to 12 closed, 1 to 16 closed, 2 to 15 closed, 3 to 14 closed, and 4 to 13 closed

Fig. 20. 86/12A jumpers and switch-settings.
address bits. An 8286 non-inverting bus transceiver buffers the data bus.

An 8251A USART, 8253 PIT, 1K bytes of RAM, and 4K bytes of EPROM are connected to the I/O bus. The 8089 channel programs reside in the EPROM, while the RAM provides a private working store. The PIT generates the baud rate for the 8251A from the timer clock which is equal to the crystal frequency divided by 12. The 8251A drives the serial connector through MC1488/89 RS232 drivers and receivers (12). The 8251A outputs TxRDY and RxRDY are connected to the 8089 DMA request lines (DRQ) to allow DMA transfers between the 8251A and the 8089.

The clock (CLK) inputs of the 8253 can accommodate a maximum frequency of 2 MHz. Therefore, the peripheral clock (PCLK) output of the 8284A is divided by two with a flip-flop. The CLK input of the USART is also driven by this source. However, this input must be at least 30 times the frequency of the RXC and TXC inputs, which in turn must be a minimum of 16 times the baud rate. These constraints limit the maximum baud rate to 2400 bits per second with the oscillator used on this board. Consequently, the 86/12 USART and the ADM-3A are also set up for 2400 baud.
Software

The purpose of the demonstration software is to allow verification of the model for polled, interrupt, and remote IOP architectures by supporting the specification of as many task characteristics as possible and performing the demonstration as accurately as possible, providing the throughput upon completion of the demonstration.

Capabilities

The expressions for $T_{io}$ derived from the real-time task model do not depend upon the actual task performed. The only item of interest is the CPU overhead due to task I/O. Therefore, the task in this system is to calculate successive outputs of a difference equation with a unit step input, with the outputs being directed to the terminal. This task was chosen to provide an element of realism, and, more importantly, verify that the model derived assuming a task with one input stream is equally valid for a task with an output stream.

The user of the system can specify which I/O technique should be used to transmit the outputs to the terminal, and the amount of time required to compute each output, $C$. The amount of time required to process each byte of an equation output, $T_j$, is also programmable, as is the total amount of CPU execution time.
Because the maximum baud rate is limited to 2400, F variation would be restricted to lower baud rates. Variable baud rates would require recompilation of the software for each change, and it would provide no additional insight into actual system performance, since baud rates lower than 2400 are much less common than 9600 or 19200. Therefore, F variation is not supported by the demonstration system. Furthermore, 8 bytes are required to represent each output of the difference equation. Thus N is eight, and variable N would require not only recompilation, but also extensive modification of the software for each change to reformat the insertion of line feeds and carriage returns into the output stream. Consequently, N variation is not supported. Because N and F are fixed at 8 and 2400 baud respectively, it takes 0 milliseconds to transmit each output of the equation to the terminal, where 0 is given by:

\[
0 = \frac{1}{((2400 \text{ bits/sec})(1/11 \text{ byte/bits})}
\frac{1}{(1/8 \text{ output/bytes})})
\]

\[
0 = 36.7 \text{ milliseconds}
\]

where,

11 bits/byte = 1 start bit + 8 bits/byte + 2 stop bits

Furthermore, because F is restricted to 2400 baud,
Tj variation only has an impact on interrupt I/O. For reasonable values of Tj below the millisecond range, or approximately 250 8086 instructions, Tj can be performed while the device is busy in the polled and remote IOP cases. Consequently, actual Tj variation is only provided in the interrupt case. As a result, using polled I/O as the reference subsystem, the fractional CPU overhead, P, is given by:

\[
P = \frac{0}{(C + 0)} = \frac{36.7}{(C + 36.7)}
\]  

(49)

where,

\(C = \text{time required to compute each output in milliseconds}\)

Thus C allows the user to control P. The total execution time entered by the user determines M, which is simply the number of outputs sent to the terminal before the CPU is timed out. Therefore, for a transfer rate of 2400 baud, the demonstration system allows the following model equations to be verified:

\[
P = \frac{36.7}{(C + 36.7)}
\]  

(49)

\[
T_{\text{IOP}} = \frac{(42}{M}) + \frac{(N/F)}{202} \text{ clocks}
\]  

(50)

\[
T_{\text{IOI}} = \left(305 + T_j\right)N + 92 + \frac{56}{M} \text{ clocks}
\]  

(51)

\[
T_{\text{IOR}} = \left(103/M\right) + 98 + 2.2N \text{ clocks}
\]  

(52)

where,
Tiop = absolute CPU overhead for polled I/O subsystem

Tioi = absolute CPU overhead for interrupt I/O subsystem

Tior = absolute CPU overhead for remote IOP subsystem

N = 8

F = 0.000043636 bytes/clock

M = total number of equation outputs transmitted to terminal

Tj = 500(Tj in 0.1 millisecond units)

and,

T'/Tp = (1 - RP)/(1 - P)

for

R = Tio'/Tiop

where,

T' = throughput of interrupt or remote IOP subsystem

Tp = throughput of polled I/O subsystem

Tio' = absolute CPU overhead for interrupt or remote IOP subsystem

Implementation

The terminal is the only I/O path to the user. Therefore, it serves two purposes. First, it is used to prompt and receive values for the I/O technique to be used in the demonstration, C, Tj, and the total available CPU time. It is then used as the output
device for the demonstration.

Consequently, the software also functions in two modes. Upon reset, it first determines what I/O technique will be used. Because both boards share the connector to the terminal, only one of the boards may attempt to drive the connector. If polled or interrupt I/O is desired, the remote IOP board must be pulled out of the card cage. If the remote IOP is to be used, the edge connector on the 86/12A must be detached. The system software assumes that the remote IOP is desired initially. It then attempts to initialize the IOP. If the IOP does not respond, it knows that the IOP is not present and the assumption was wrong. It then corrects the assumption and directs all I/O through the 86/12A USART. In this case it asks the user whether polled or interrupt I/O is desired. In either case, it then requests C, Tj, and the available CPU time. The available CPU time is loaded into a counter which provides a time out interrupt. There is a procedural implementation of the task for each I/O technique. The variables C and Tj are adjusted for the known computation and processing times. The interrupt counter is started, and the connection and I/O type information is used to call the appropriate task implementation. These procedures are responsible for initializing their own device controllers and
formatting their own output streams. Each procedure uses the adjusted values of C and Tj to execute time delays which simulate the requested computation and processing times. Finally, when the available CPU time has expired, the counter interrupts the CPU. This vectors the CPU to a service routine which calculates the task throughput and displays this result on the console. The system software then halts.

The system software is composed of five modules. Four of the modules are written in PL/M-86 (13,14), while the fifth is written in ASM-89 (15). The broad functions of each module are shown in Figure 22, while more detailed information can be found in the amply commented listings and cross-references in Appendix D.

Deviation from Model Assumptions

For the polled and interrupt cases, the chief source of error will be the use of PL/M-86 to implement the tasks when the model was derived using assembly language. Because this compiler does not track register usage, the variables are memory-based as opposed to the register-based variables used in the derivation. Also contributing to the error is the code required for carriage return and line feed insertion, as well as compiler code inefficiency in general. The primary source of error in the remote IOP case will be the
MAIN DEMONSTRATOR MODULE (PL/M-86)
  • System Calling Sequence
  • Difference Equation Procedures

DEMONSTRATOR SUPPORT MODULE (PL/M-86)
  • Demonstration Setup Procedures
  • Interrupt Procedures

CPU UTILITIES MODULE (PL/M-86)
  • 86/12A Device Drivers
  • String I/O Procedures
  • ASCII to Hex Conversion Procedures

IOP UTILITIES MODULE (PL/M-86)
  • IOP Initialization Procedures
  • IOP Channel Program Control Procedures

CHANNEL PROGRAMS (ASM-89)
  • USART and PIT Setup
  • Terminal I/O

Fig. 22. Demonstration system software modules.
lower clock rate employed on the remote IOP board. In Figure 21, the 8089 clock (CLK) input is driven by the peripheral clock (PCLK) output of the 8284A. The frequency of this output is one-half that of the normal clock (CLK) output. Furthermore, the crystal frequency is 14.318 MHz instead of the 15 MHz crystal used on the 86/12A board. This results in an IOP clock rate of 2.386 MHz instead of the 5 MHz CPU clock rate. Thus, the IOP bus cycles are longer than those assumed in the derivation of the model, resulting in more CPU idle time. The PCLK output was used because the 8089 did not meet its timing specifications after a short burn-in period.
CHAPTER IV
RESULTS

The throughputs measured in two demonstrations are shown in Tables 1 and 2. As shown in Table 1, the time required to compute each output, hence P, was varied with Tj equal to zero. The available execution time was a minimum of 10 seconds resulting in at least 100 outputs for each demonstration. Therefore, the effect of M on Tio is insignificant, and M was not recorded in either demonstration. In Table 2, the time required to compute each output was held constant at 46 milliseconds, corresponding to a P of 0.44, while Tj was varied between 0.1 and 1 millisecond.

Using equations 49 through 54 and 0.2 microseconds per clock period for a 5 MHz 8086, the predicted overheads for Tj equal to zero are:

\[ T_{iop} = 36,710 \text{ microseconds} \]
\[ T_{ioi} = 506 \text{ microseconds} \]
\[ T_{ior} = 23 \text{ microseconds} \]

with,
\[ R_i = \frac{T_{ioi}}{T_{iop}} = 0.01378 \]
\[ R_r = \frac{T_{ior}}{T_{iop}} = 0.0006265 \]
## TABLE 1
MEASURED THROUGHPUT FOR $T_j = 0$
AND VARIABLE $P$

<table>
<thead>
<tr>
<th>Time to Compute Each Output (Milliseconds)</th>
<th>Throughput (Outputs/Second)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Polled</td>
</tr>
<tr>
<td>46</td>
<td>11.1</td>
</tr>
<tr>
<td>56</td>
<td>9.9</td>
</tr>
<tr>
<td>69</td>
<td>8.8</td>
</tr>
<tr>
<td>85</td>
<td>7.7</td>
</tr>
<tr>
<td>107</td>
<td>6.6</td>
</tr>
<tr>
<td>137</td>
<td>5.4</td>
</tr>
<tr>
<td>183</td>
<td>4.3</td>
</tr>
<tr>
<td>260</td>
<td>3.2</td>
</tr>
<tr>
<td>412</td>
<td>2.1</td>
</tr>
<tr>
<td>870</td>
<td>1.1</td>
</tr>
</tbody>
</table>

## TABLE 2
MEASURED THROUGHPUT FOR $P = 0.44$
AND VARIABLE $T_j$

<table>
<thead>
<tr>
<th>$T_j$ (0.1 Milliseconds)</th>
<th>Throughput (Outputs/Second)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Polled</td>
</tr>
<tr>
<td>1</td>
<td>11.1</td>
</tr>
<tr>
<td>2</td>
<td>11.1</td>
</tr>
<tr>
<td>3</td>
<td>11.1</td>
</tr>
<tr>
<td>4</td>
<td>11.1</td>
</tr>
<tr>
<td>5</td>
<td>11.1</td>
</tr>
<tr>
<td>6</td>
<td>11.1</td>
</tr>
<tr>
<td>7</td>
<td>11.1</td>
</tr>
<tr>
<td>8</td>
<td>11.1</td>
</tr>
<tr>
<td>9</td>
<td>11.1</td>
</tr>
<tr>
<td>10</td>
<td>11.1</td>
</tr>
</tbody>
</table>
\[
\frac{T_{i'}}{T_p} = \frac{(1 - (R_i)P)}{(1 - P)} \tag{55}
\]
\[
\frac{T_{r'}}{T_p} = \frac{(1 - (R_r)P)}{(1 - P)} \tag{56}
\]

where,

\[
T_{i'} = \text{predicted throughput of interrupt subsystem}
\]

\[
T_{r'} = \text{predicted throughput of remote IOP subsystem}
\]

Using these equations, the predicted throughput gains and throughputs for the interrupt and remote IOP cases are shown in Table 3. The predicted throughputs were obtained by multiplying the predicted throughput gain by the measured throughput of the polled I/O case.

For the demonstration of Table 2, \(P\) is fixed at 0.44 while \(T_j\) is variable. In this case the predicted overheads are:

\[
T_{iop} = 36,710 \text{ microseconds}
\]

\[
T_{ioi} = 506 + 800T_j \text{ microseconds}
\]

\[
T_{ior} = 23 \text{ microseconds}
\]

where,

\[
T_j \text{ is in 0.1 millisecond units}
\]

Thus \(R_r\) is still 0.0006265, while \(T_{ioi}\) and \(R_i\) as a function of \(T_j\) are shown in Table 4. Using \(R_r\) and \(R_i\) from Table 4 in equations 55 and 56, the predicted throughput gains and throughputs are shown in Table 5.
### TABLE 3

PREDICTED THROUGHPUT FOR Tj = 0 AND VARIABLE P

<table>
<thead>
<tr>
<th>P</th>
<th>Tp</th>
<th>Ti'/Tp</th>
<th>Ti'</th>
<th>Tr'/Tp</th>
<th>Tr'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4438</td>
<td>11.1</td>
<td>1.787</td>
<td>19.8</td>
<td>1.797</td>
<td>19.9</td>
</tr>
<tr>
<td>0.3959</td>
<td>9.9</td>
<td>1.646</td>
<td>16.3</td>
<td>1.655</td>
<td>16.4</td>
</tr>
<tr>
<td>0.3472</td>
<td>8.8</td>
<td>1.525</td>
<td>13.4</td>
<td>1.532</td>
<td>13.5</td>
</tr>
<tr>
<td>0.3016</td>
<td>7.7</td>
<td>1.426</td>
<td>11.0</td>
<td>1.432</td>
<td>11.0</td>
</tr>
<tr>
<td>0.2554</td>
<td>6.6</td>
<td>1.338</td>
<td>8.8</td>
<td>1.343</td>
<td>8.9</td>
</tr>
<tr>
<td>0.2113</td>
<td>5.4</td>
<td>1.264</td>
<td>6.8</td>
<td>1.268</td>
<td>6.8</td>
</tr>
<tr>
<td>0.1670</td>
<td>4.3</td>
<td>1.198</td>
<td>5.2</td>
<td>1.200</td>
<td>5.2</td>
</tr>
<tr>
<td>0.1237</td>
<td>3.2</td>
<td>1.139</td>
<td>3.6</td>
<td>1.141</td>
<td>3.6</td>
</tr>
<tr>
<td>0.0818</td>
<td>2.1</td>
<td>1.088</td>
<td>2.3</td>
<td>1.089</td>
<td>2.3</td>
</tr>
<tr>
<td>0.0405</td>
<td>1.1</td>
<td>1.042</td>
<td>1.2</td>
<td>1.042</td>
<td>1.2</td>
</tr>
</tbody>
</table>

### TABLE 4

INTERRUPT Tio AND Ri FOR P = 0.44 AND VARIABLE Tj

<table>
<thead>
<tr>
<th>Tj  (0.1 Milliseconds)</th>
<th>Tio  (Microseconds)</th>
<th>Ri</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1306</td>
<td>0.03558</td>
</tr>
<tr>
<td>2</td>
<td>2106</td>
<td>0.05757</td>
</tr>
<tr>
<td>3</td>
<td>2906</td>
<td>0.07916</td>
</tr>
<tr>
<td>4</td>
<td>3706</td>
<td>0.10100</td>
</tr>
<tr>
<td>5</td>
<td>4506</td>
<td>0.12270</td>
</tr>
<tr>
<td>6</td>
<td>5306</td>
<td>0.14450</td>
</tr>
<tr>
<td>7</td>
<td>6106</td>
<td>0.16630</td>
</tr>
<tr>
<td>8</td>
<td>6906</td>
<td>0.18810</td>
</tr>
<tr>
<td>9</td>
<td>7706</td>
<td>0.20990</td>
</tr>
<tr>
<td>10</td>
<td>8506</td>
<td>0.23170</td>
</tr>
</tbody>
</table>
### TABLE 5
**PREDICTED THROUGHPUT FOR P = 0.44 AND VARIABLE Tj**

<table>
<thead>
<tr>
<th>Tj (0.1 Msec.)</th>
<th>Tp</th>
<th>Ti'/Tp</th>
<th>Ti'</th>
<th>Tr'/Tp</th>
<th>Tr'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11.1</td>
<td>1.770</td>
<td>19.6</td>
<td>1.797</td>
<td>19.9</td>
</tr>
<tr>
<td>2</td>
<td>11.1</td>
<td>1.752</td>
<td>19.4</td>
<td>1.797</td>
<td>19.9</td>
</tr>
<tr>
<td>3</td>
<td>11.1</td>
<td>1.735</td>
<td>19.3</td>
<td>1.797</td>
<td>19.9</td>
</tr>
<tr>
<td>4</td>
<td>11.1</td>
<td>1.717</td>
<td>19.1</td>
<td>1.797</td>
<td>19.9</td>
</tr>
<tr>
<td>5</td>
<td>11.1</td>
<td>1.700</td>
<td>18.9</td>
<td>1.797</td>
<td>19.9</td>
</tr>
<tr>
<td>6</td>
<td>11.1</td>
<td>1.683</td>
<td>18.7</td>
<td>1.797</td>
<td>19.9</td>
</tr>
<tr>
<td>7</td>
<td>11.1</td>
<td>1.665</td>
<td>18.5</td>
<td>1.797</td>
<td>19.9</td>
</tr>
<tr>
<td>8</td>
<td>11.1</td>
<td>1.648</td>
<td>18.3</td>
<td>1.797</td>
<td>19.9</td>
</tr>
<tr>
<td>9</td>
<td>11.1</td>
<td>1.630</td>
<td>18.1</td>
<td>1.797</td>
<td>19.9</td>
</tr>
<tr>
<td>10</td>
<td>11.1</td>
<td>1.613</td>
<td>17.9</td>
<td>1.797</td>
<td>19.9</td>
</tr>
</tbody>
</table>

### TABLE 6
**MODEL ERROR FOR Tj = 0 AND VARIABLE P**

<table>
<thead>
<tr>
<th>P</th>
<th>% error Ti'</th>
<th>% error Tr'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4438</td>
<td>-0.50</td>
<td>-3.40</td>
</tr>
<tr>
<td>0.3959</td>
<td>-1.81</td>
<td>-2.96</td>
</tr>
<tr>
<td>0.3472</td>
<td>-0.74</td>
<td>-1.46</td>
</tr>
<tr>
<td>0.3016</td>
<td>-0.90</td>
<td>-1.79</td>
</tr>
<tr>
<td>0.2554</td>
<td>-1.12</td>
<td>0.00</td>
</tr>
<tr>
<td>0.2113</td>
<td>-2.86</td>
<td>-2.86</td>
</tr>
<tr>
<td>0.1670</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.1237</td>
<td>-2.70</td>
<td>-2.70</td>
</tr>
<tr>
<td>0.0818</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.0405</td>
<td>-9.09</td>
<td>-9.09</td>
</tr>
<tr>
<td>mean</td>
<td>-1.97</td>
<td>-2.43</td>
</tr>
<tr>
<td>% error</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The percent error of the predicted throughputs are calculated from the equations:

\[
\% \text{ error } T_i' = \frac{(T_i' - T_i)}{T_i} \quad (57)
\]
\[
\% \text{ error } T_r' = \frac{(T_r' - T_r)}{T_r} \quad (58)
\]

The percent error of the predicted throughputs for the demonstrations are shown in Tables 6 and 7.

The performance model is intended to provide estimates of the throughput of any of the four architectures using one of the architectures as a reference. In this case, polled I/O was used as the reference and the percent error in Tables 6 and 7 indicates that the model provides very good estimates. The case of variable \( T_j \) with \( P \) of 0.44 implies that the worst case error for realistic systems is on the order of 5%. Few systems spend more than 50% of their CPU time performing I/O. However, because the relative throughput equation is more sensitive to \( R \) for large \( P \), the error in the model will be greater for these systems. Furthermore, these errors are not surprising, since the demonstration system sharply deviates from the model by employing a high-level language to implement the I/O software and a much lower clock rate on the IOP. However, this is a true test of the model because real systems will employ software written in a high-level
### TABLE 7

**MODEL ERROR FOR P = 0.44 AND VARIABLE Tj**

<table>
<thead>
<tr>
<th>Tj (0.1 Msec.)</th>
<th>% error Ti'</th>
<th>% error Tr'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.51</td>
<td>-3.40</td>
</tr>
<tr>
<td>2</td>
<td>0.52</td>
<td>-3.40</td>
</tr>
<tr>
<td>3</td>
<td>1.58</td>
<td>-3.40</td>
</tr>
<tr>
<td>4</td>
<td>2.14</td>
<td>-3.40</td>
</tr>
<tr>
<td>5</td>
<td>2.72</td>
<td>-3.40</td>
</tr>
<tr>
<td>6</td>
<td>3.31</td>
<td>-3.40</td>
</tr>
<tr>
<td>7</td>
<td>3.35</td>
<td>-3.40</td>
</tr>
<tr>
<td>8</td>
<td>3.98</td>
<td>-3.40</td>
</tr>
<tr>
<td>9</td>
<td>4.02</td>
<td>-3.40</td>
</tr>
<tr>
<td>10</td>
<td>4.68</td>
<td>-3.40</td>
</tr>
</tbody>
</table>

**mean**

% error 2.68 -3.40

### TABLE 8

**RELATIVE CPU OVERHEADS AS A FUNCTION OF F**

<table>
<thead>
<tr>
<th>F (Bytes/Second)</th>
<th>Rp</th>
<th>Ri</th>
<th>Rl</th>
<th>Rr</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1.000</td>
<td>0.010</td>
<td>0.006</td>
<td>0.000</td>
</tr>
<tr>
<td>500</td>
<td>1.000</td>
<td>0.051</td>
<td>0.028</td>
<td>0.000</td>
</tr>
<tr>
<td>1000</td>
<td>1.000</td>
<td>0.102</td>
<td>0.056</td>
<td>0.001</td>
</tr>
<tr>
<td>5000</td>
<td>1.000</td>
<td>0.500</td>
<td>0.277</td>
<td>0.003</td>
</tr>
<tr>
<td>10000</td>
<td>1.000</td>
<td>1.000</td>
<td>0.554</td>
<td>0.006</td>
</tr>
<tr>
<td>20000</td>
<td>1.000</td>
<td>x</td>
<td>0.554</td>
<td>0.013</td>
</tr>
<tr>
<td>40000</td>
<td>1.000</td>
<td>x</td>
<td>0.554</td>
<td>0.013</td>
</tr>
</tbody>
</table>

**NOTE:** Calculated for M = 10, N = 128, Tj = 200 clocks. Symbol x denotes that the architecture cannot transfer at that frequency.
language, and the components of the system may not be driven at the speeds assumed in the model. Thus the model is an easy, accurate way to obtain performance estimates of any I/O subsystem before it is designed, allowing the designer to combine the estimates with cost, reliability and space requirements to select the most suitable I/O subsystem.
CHAPTER V
CONCLUSION

Architectural Performance

In this section, the performance model will be used to investigate the influence of the model variables upon the throughput gain and draw some general conclusions about the suitability of each architecture. Because the throughput gain is strongly a function of \( P \), which varies from system to system, general conclusions can only be made if the architectures can be compared in a manner independent of \( P \). As shown in equation 6:

\[
\lim_{R \to 0} \left( \frac{T'}{T} \right) = \frac{1}{1 - P}
\]

and

\[
\frac{T'}{T} \approx \frac{1}{1 - P}
\]

for

\[ R \leq 0.1 \quad \text{and} \quad P \geq 0.1 \]

This indicates that an I/O subsystem architecture provides maximum throughput gain in typical systems only when the relative CPU overhead is below 10%. Thus, comparing the operating regions in which the architectures provide maximum gain is a method of evaluating their
performance independently of $P$.

In Table 8 the relative CPU overhead as a function of the device transfer rate is shown. An example of this case would be updating a CRT screen at the lower transfer rates, or accessing ten sectors of a floppy disk at the higher transfer rates. The remote IOP architecture always provides maximum throughput gain. The interrupt and local IOP subsystems provide maximum gain only for transfer rates below approximately 1000 and 4000 bytes per second respectively. Furthermore, the interrupt architecture has an absolute maximum transfer rate of 16,000 bytes per second.

In Table 9, the relative CPU overhead as a function of the amount of I/O processing per byte, $T_j$, is shown. Again, the remote IOP always provides maximum throughput gain. The interrupt subsystem cannot provide maximum gain for more than 300 clocks of processing, or about 15 instructions per byte. The local IOP subsystem is similarly limited to approximately 25 instructions for maximum throughput gain. This case would be typical of transmitting or receiving a block of data from a medium speed modem.

The relative CPU overheads as a function of the number of bytes output each cycle, $N$, are shown in Table 10. Once again, the remote IOP architecture always
### Table 9

#### Relative CPU Overheads as a Function of Tj

<table>
<thead>
<tr>
<th>Tj (Clocks)</th>
<th>Rp</th>
<th>Ri</th>
<th>Rl</th>
<th>Rr</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>1.000</td>
<td>0.0884</td>
<td>0.0556</td>
<td>0.0007</td>
</tr>
<tr>
<td>400</td>
<td>1.000</td>
<td>0.1231</td>
<td>0.0905</td>
<td>0.0007</td>
</tr>
<tr>
<td>600</td>
<td>1.000</td>
<td>0.1578</td>
<td>0.1252</td>
<td>0.0007</td>
</tr>
<tr>
<td>800</td>
<td>1.000</td>
<td>0.1925</td>
<td>0.1599</td>
<td>0.0007</td>
</tr>
<tr>
<td>1000</td>
<td>1.000</td>
<td>0.2272</td>
<td>0.1952</td>
<td>0.0007</td>
</tr>
<tr>
<td>1200</td>
<td>1.000</td>
<td>0.2626</td>
<td>0.2299</td>
<td>0.0007</td>
</tr>
<tr>
<td>1400</td>
<td>1.000</td>
<td>0.2973</td>
<td>0.2646</td>
<td>0.0007</td>
</tr>
<tr>
<td>1600</td>
<td>1.000</td>
<td>0.3320</td>
<td>0.2993</td>
<td>0.0007</td>
</tr>
<tr>
<td>1800</td>
<td>1.000</td>
<td>0.3667</td>
<td>0.3340</td>
<td>0.0007</td>
</tr>
<tr>
<td>2000</td>
<td>1.000</td>
<td>0.4014</td>
<td>0.3694</td>
<td>0.0007</td>
</tr>
</tbody>
</table>

**NOTE:** Calculated for M = 1, N = 128, F = 873 bytes/second.

### Table 10

#### Relative CPU Overheads as a Function of N

<table>
<thead>
<tr>
<th>N (Bytes)</th>
<th>Rp</th>
<th>Ri</th>
<th>Rl</th>
<th>Rr</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1.000</td>
<td>0.650</td>
<td>0.320</td>
<td>0.003</td>
</tr>
<tr>
<td>256</td>
<td>1.000</td>
<td>0.650</td>
<td>0.320</td>
<td>0.003</td>
</tr>
<tr>
<td>512</td>
<td>1.000</td>
<td>0.650</td>
<td>0.320</td>
<td>0.003</td>
</tr>
<tr>
<td>1024</td>
<td>1.000</td>
<td>0.650</td>
<td>0.320</td>
<td>0.003</td>
</tr>
<tr>
<td>2048</td>
<td>1.000</td>
<td>0.650</td>
<td>0.320</td>
<td>0.003</td>
</tr>
<tr>
<td>4096</td>
<td>1.000</td>
<td>0.650</td>
<td>0.320</td>
<td>0.003</td>
</tr>
</tbody>
</table>

**NOTE:** Calculated for M = 1, Tj = 0, F = 8192 bytes/second.
provides maximum throughput gain, while neither of the other architectures come close. This case would be typical of transmitting or receiving a frame from a high-speed synchronous serial link.

From these calculations, it appears that the remote IOP provides the maximum throughput gain from any system over the widest operating region. The performance of the local IOP is similar to that of an interrupt I/O subsystem, with the advantage that the local IOP can transfer at 250,000 bytes per second, while the interrupt subsystem is limited to 16,000 bytes per second, and the polled subsystem is limited to 64,000 bytes per second. Both the interrupt and local IOP architectures provide throughput gain; however, it is over a very small operating region that they provide maximum gain.

Cost-Effectiveness of the 8089

The cost of the major components required to implement each architecture are shown in Table 11. Only those components required in addition to the polled I/O subsystem are listed. The cost of an interrupt subsystem depends upon the number of I/O devices employed. Since each 8259 can serve eight devices, the typical subsystem cost would be around forty dollars. The cost of a local IOP subsystem is
## TABLE 11

**COMPONENT COST OF ARCHITECTURES**

<table>
<thead>
<tr>
<th>Component</th>
<th>Quantity</th>
<th>Unit Cost ($)</th>
<th>Total Cost ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interrupt I/O</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QD8259A</td>
<td>1-9</td>
<td>8.70</td>
<td>8.70-78.30</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td>8.70-78.30</td>
</tr>
<tr>
<td><strong>Local IOP</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QD8089-3</td>
<td>1</td>
<td>38.70</td>
<td>38.70</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td>38.70</td>
</tr>
<tr>
<td><strong>Remote IOP</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QD8089-3</td>
<td>1</td>
<td>38.70</td>
<td>38.70</td>
</tr>
<tr>
<td>QD8284A</td>
<td>1</td>
<td>7.60</td>
<td>7.60</td>
</tr>
<tr>
<td>QD8288</td>
<td>1</td>
<td>12.55</td>
<td>12.55</td>
</tr>
<tr>
<td>QD8289</td>
<td>1</td>
<td>20.00</td>
<td>20.00</td>
</tr>
<tr>
<td>QD8287</td>
<td>3</td>
<td>6.55</td>
<td>19.65</td>
</tr>
<tr>
<td>QD8283</td>
<td>3</td>
<td>6.55</td>
<td>19.65</td>
</tr>
<tr>
<td>QD8286</td>
<td>1</td>
<td>6.55</td>
<td>6.55</td>
</tr>
<tr>
<td>QD8282</td>
<td>1</td>
<td>6.55</td>
<td>6.55</td>
</tr>
<tr>
<td>D8254</td>
<td>1</td>
<td>12.50</td>
<td>12.50</td>
</tr>
<tr>
<td>QD8251A</td>
<td>1</td>
<td>7.90</td>
<td>7.90</td>
</tr>
<tr>
<td>QD8255A</td>
<td>1</td>
<td>7.50</td>
<td>7.50</td>
</tr>
<tr>
<td>QD2114A-5</td>
<td>2</td>
<td>4.45</td>
<td>8.90</td>
</tr>
<tr>
<td>QD2732A-3</td>
<td>1</td>
<td>9.80</td>
<td>9.80</td>
</tr>
<tr>
<td>MC1488L</td>
<td>1</td>
<td>1.20</td>
<td>1.20</td>
</tr>
<tr>
<td>MC1489AL</td>
<td>1</td>
<td>1.35</td>
<td>1.35</td>
</tr>
<tr>
<td>miscellaneous, NAND, FF, XTAL</td>
<td></td>
<td></td>
<td>5.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td>185.40</td>
</tr>
</tbody>
</table>

**NOTE:** Prices are for quantities of 100 or more.

**SOURCE:** OEM Price List. Santa Clara, California: Intel Corporation, October 11, 1982.
approximately the same. Therefore, the local IOP architecture appears to be a cost-effective substitute for the interrupt subsystem in view of its larger bandwidth, slightly higher throughput gain, and similar cost. The cost of a remote IOP subsystem is much higher; however, these costs reflect a more general design than that used in the demonstration system. From a cost-effectiveness standpoint, the remote IOP is clearly superior only when its throughput gain is five to one or more, since its cost is approximately five times that of a local IOP subsystem. This will occur only for systems in which I/O activity consumes 80% of the available CPU time. Since this is the exception rather than the rule, the suitability of the remote IOP architecture will depend heavily on the particular characteristics and requirements of the system being designed, and no general conclusions regarding the cost-effectiveness of the remote IOP architecture can be reached. Only in systems requiring maximum throughput at any cost is the remote IOP subsystem a clear-cut choice.

**Summary**

The derivation of a predictive performance model for the iAPX 86 I/O architectures was presented, and the estimates of the model were compared with
measurements from a demonstration system. The comparisons indicated that the model provides very good estimates. The model was then used to examine the performance of each architecture. This was combined with the relative cost of each architecture to arrive at the general conclusion that the 8089 employed in the local configuration is superior to the interrupt and polled I/O subsystems from a cost-effectiveness standpoint, while the cost-effectiveness of the remote configuration depends upon the system under consideration.
APPENDICES
APPENDIX A

8089 OVERVIEW

The 8089 is a 40 pin DIP which contains two I/O channels. Each channel consists of a DMA controller and a specialized microprocessor.

The 8089 can address two separate spaces: a 1M byte system space, and a 64K byte I/O space. A channel is governed by a channel program of IOP instructions. The program can be located in either address space. To start a channel program, the 8089's channel attention line must be toggled with the select line being used to determine which channel will execute the program. The 8089 then reads the channel control block. There is one control block for each channel, and the locations of the blocks are established when the 8089 is initialized. In the control block the 8089 finds the channel command word which describes the operation that the channel is to perform. Primarily, it indicates in which space the channel program, or task block, resides. The control block also contains the channel busy flag, which is set when the channel is executing a program, and a pointer to the channel parameter block.
The 8089 loads this pointer into the PP register which the channel program uses to address the parameter block. The parameter block contains a pointer to the task block which the 8089 loads into the TP register, which is essentially a program counter. The remainder of the parameter block is free-form and is used to pass information between the 8086 and the 8089.

Once the TP is loaded, the 8089 starts fetching instructions from the channel program. In addition to the TP and PP registers, there is the CC register which controls the execution of DMA transfers. There are three general-purpose registers: GA, GB, and GC. In DMA transfers, GA and GB are used to point to the source and destination of the transfer, while GC points to the base of a translation table which can be used to translate each byte as it is transferred. The BC register is a counter, which is used for byte-count termination during DMA. The IX register is an index register for addressing, and the MC register is used for masked comparisons. Each channel has a set of these registers.

The 8089 has a repertoire of over fifty instructions and several addressing modes too lengthy to fully describe here. For further information, see the iAPX 86,88 User's Manual (5).
APPENDIX B

MODEL INSTRUCTION SEQUENCES

This appendix contains the instruction sequences used to implement the task steps in the model derivation. The sequences are presented in ASM-86 (16,17) and ASM-89 (15). Each step is independent, and their order is not significant. Moreover, although the listing is presented in assembly language for clarity, the listing should not be interpreted as a program. The execution time of a step is listed below its sequence, and the execution times of all steps are summarized in Tables 12 through 15.
All instructions are ASM-86 until noted otherwise.

Polled I/O steps until otherwise noted.

Typically device controllers require 3 bytes of setup information. This requires three repetitions of the sequence:

- `MOV AL, Setup_Data`
- `OUT Control_Port, AL`

; `Tsd = 3(14) = 42 clocks`

**Tpc: MOV AX, Buffer_Ptr**

`MOV AX`

`PUSH AX`

`MOV AX, N`

`PUSH AX`

`MOV AX, Data_Port`

`PUSH AX`

`MOV AX, Status_Port`

`PUSH AX`

`CALL Tsave`

; `Tpc = 79 clocks`

**Tsave: PUSH BP**

`MOV BP, SP`

`PUSH CX`

`PUSH DX`

`PUSHF`

; `Tsave = 45 clocks`

**Tload: MOV BX, [EBP+2]**

`MOV CX, [EBP+10]`

; `Tload = 34 clocks`

**Tw: MOV DX, [EBP+6]**

; `Tw = 50 clocks`

**T: IN AL, DX**

; `T = 8 clocks`
Tsi:
MOV [BX], AL ; 14 clocks
INC BX ; 2 clocks
; Tsi = 16 clocks
;
Tlc:
DEC CX ; 2 clocks
JNZ Jump_Table ; 4/16 clocks
; Short jump to long jump in table so there is no restriction
; on length of Tj code. Long jump takes 15 clocks
; Tlc = 33 - (27/Nl)
;
Trest:
POPF ; 8 clocks
POP BX ; 8 clocks
POP CX ; 8 clocks
POP BP ; 8 clocks
; Trest = 32 clocks
;
Tret:
RET 8 ; 12 clocks
;
Tmov:
MOV AL, [BX] ; 10 clocks
; Interrupt I/O steps until otherwise noted
;
Tsdi - Same as Tsd except additional byte sent to 8259 for
; new mask
;
Tsdi = 4(14) = 56 clocks
;
Tstore1:
MOV Buff_1, Buffer_Ptr ; 16 clocks
MOV Count_1, N ; 16 clocks
MOV Data_Port, Data_Port ; 16 clocks
MOV Stat_Port, Status_Port ; 16 clocks
; Tstore1 = 64 clocks
;
Tstore2 - Same as Tstore1 except add:
;
MOV Buff_2, Buffer_Ptr (16 clocks)
MOV Count_2, N (16 clocks)
; Tstore2 = 96 clocks
;
Ten:
MOV AL, Enable Interrupt_Command ; 4 clocks
OUT AL, Status_Port ; 10 clocks
; Ten = 14 clocks
; Tsave for interrupt I/O requires pushing AX,BX,CX,DX
; 4 PUSH instructions at 11 clocks each
; Tsave = 44 clocks

Trem:  MOV DX,Stat_Port          ; 14 clocks
        MOV AL,Remove_Interrupt_Command ; 4 clocks
        OUT AL,DX                 ; 8 clocks
        ; Trem = 26 clocks

Teoi:  MOV AL,End_of_Interrupt_Command ; 4 clocks
        OUT AL,PIC_Control        ; 10 clocks
        ; Teoi = 14 clocks

Tload: MOV DX,Data_Port            ; 14 clocks
        MOV BX,Buff_1             ; 14 clocks
        ; Tload = 28 clocks
        ; T is the same as in Pollled I/O

Tsi:   MOV DX,AL              ; 14 clocks
        INC BX                 ; 2 clocks
        MOV Buff_1,BX          ; 15 clocks
        ; Tsi = 31 clocks

Tic:   DEC Count_1           ; 21 clocks
        JNE Trest              ; 9/16 clocks
        ; Tic = 37 - (12/4)

Tdis:  MOV DX,Stat_Port        ; 14 clocks
        MOV AL,Disable_Interrupt_Command ; 4 clocks
        OUT AL,DX               ; 8 clocks
        ; Tdis = 26 clocks

; Trest - Pop 4 registers off stack. Takes 4 POP instructions
; at 8 clocks each
; Trest = 32 clocks
Tiret: IRET ; 24 clocks

Tpc:
MOV AX,Buff_2 ; 10 clocks
PUSH AX
MOV AX,Count_2 ; 10 clocks
PUSH AX
CALL Tloadp
; Tpc = 61 clocks

Tloadp:
PUSH BP ; 11 clocks
MOV BP,SP
MOV CX,[BP+4] ; 2 clocks
MOV BX,[BP+6] ; 17 clocks
; Tloadp = 47 clocks
; See polled I/O for Tmov, Tsi, and Tlc

Tret:
PUSH BP ; 8 clocks
POP BP
RET 4 ; 12 clocks
; Tret = 20 clocks
; Local IOP steps until noted otherwise

Tpb:
LEA BX, Parm_Block ; 8 clocks
MOV WORD PTR [BX+3],OFFSET Task_Block ; 15 clocks
MOV WORD PTR [BX+23],SEG Task_Block ; 16 clocks
MOV WORD PTR [BX+43],OFFSET Buffer ; 16 clocks
MOV WORD PTR [BX+63],SEG Buffer ; 16 clocks
MOV WORD PTR [BX+83],N ; 16 clocks
MOV WORD PTR [BX+103],Data_Port ; 16 clocks
; Tpb = 103 clocks
; See polled I/O for Tds

Tcb:
LEA BX, Channel_Control_Block ; 8 clocks
MOV WORD PTR [BX+23],OFFSET Parm_Block ; 16 clocks
MOV WORD PTR [BX+43],SEG Parm_Block ; 16 clocks
MOV BYTE PTR [BX+43],Channel_Command_Word ; 15 clocks
; Tcb = 55 clocks

Tcas:
OUT AL, Channel ; 10 clocks
; All instructions ASM-89 until otherwise noted
Tccw - Requires no explicit instructions. Executed implicitly after a channel attention. See user's manual for timing.

Tccw = 108 clocks

Tload:
- MOV GA,[PPJ.10]
- LPD GB,[PPJ.4]
- MOV BC,[PPJ.8]
- MOVI CC,Control_Word

Tload = 89 clocks

Txfer:
- XFER
- WID 8+8

Txfer = 30 clocks if it doesn’t follow a jump
Txfer = 36 clocks if it does follow a jump

Tterm - No explicit instructions. Executed after DMA xfer.
Tterm = 12 clocks

Tmov:
- MOVE GC,[GBJ]?

? = -1 for Fig. 13; ? = ' ' for Fig. 14

Tst:
- MOVE GC01,?;GC

? = -1 for Fig. 13; ? = ' ' for Fig. 14

Tst = 24 clocks for Fig. 13
Tst = 27 clocks for Fig. 14

Tlc:
- JNZ BC;Processing_Loop

Tlc = 19 clocks

Treload:
- LPD GB,[PPJ.4]
- MOV BC,[PPJ.8]

Treload = 53 clocks

Tdec:
- DEC BC

Tdec = 10 clocks

Remote IOP steps until noted otherwise
All instructions ASM-86 until noted otherwise

Tpbl:
- LEA BX,Param_Block
- MOV WORD PTR [BXJ];Task_Block

Tpbl = 0 clocks
Tpbl = 15 clocks
MOV WORD PTR [BX+2], OFFSET Buffer ; 16 clocks
MOV WORD PTR [BX+4], SEG Buffer ; 16 clocks
MOV WORD PTR [BX+6], N ; 16 clocks
MOV WORD PTR [BX+8], Data_Port ; 16 clocks
MOV WORD PTR [BX+10], Cont_Port ; 16 clocks

; Tpb = 103 clocks

; Tcb and Tcs are the same as in Local IOP

; All instructions are ASM-89 until otherwise noted

; Only difference from Local IOP is

Tiomov: MOV IX, [GA] ; 22 clocks

; g(Tccw) - In executing the CCW, the 8089 must access the following:

; CCW Byte (4 clocks)
; Parm_Block_Ptr Dword (8 clocks)
; Task_Block_Ptr Word (4 clocks)
; Busy Byte (4 clocks)

; In the worst case, these are back to back accesses

; g(Tccw) = f(4 + 8 + 4 + 4) = 20 - 2 = 18 clocks

; g(Tload) - The 8089 must read the following:

; Buff_Ptr Dword (8 clocks)
; N_Word (4 clocks)
; Data_Port Word (4 clocks)
; Cont_Port Word (4 clocks)

; Even for consecutive reads using MOV and LPD, the instruction execution times are long enough to ensure at least 20 CPU clock separation between bus cycles, so these accesses may be treated separately

; g(Tload) = f(8) + 3f(4) = 6.002 + 3(2.228) = 12.686 clocks
### TABLE 12

**TASK STEP EXECUTION TIMES FOR POLLED I/O**

<table>
<thead>
<tr>
<th>Step</th>
<th>Execution Time (Clocks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>8</td>
</tr>
<tr>
<td>Tsd</td>
<td>42</td>
</tr>
<tr>
<td>Tpc</td>
<td>79</td>
</tr>
<tr>
<td>Tsave</td>
<td>45</td>
</tr>
<tr>
<td>Tload</td>
<td>34</td>
</tr>
<tr>
<td>Tsi</td>
<td>16</td>
</tr>
<tr>
<td>Tw</td>
<td>50</td>
</tr>
<tr>
<td>Tmov</td>
<td>10</td>
</tr>
<tr>
<td>Tlc</td>
<td>33 - (27/N)</td>
</tr>
<tr>
<td>Trest</td>
<td>32</td>
</tr>
<tr>
<td>Tret</td>
<td>12</td>
</tr>
</tbody>
</table>

### TABLE 13

**TASK STEP EXECUTION TIMES FOR LOCAL IOP**

<table>
<thead>
<tr>
<th>Step</th>
<th>Execution Time (Clocks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tds</td>
<td>42</td>
</tr>
<tr>
<td>Tpb</td>
<td>103</td>
</tr>
<tr>
<td>Tcb</td>
<td>55</td>
</tr>
<tr>
<td>Tca</td>
<td>10</td>
</tr>
<tr>
<td>Tccw</td>
<td>108</td>
</tr>
<tr>
<td>Tload</td>
<td>89</td>
</tr>
<tr>
<td>Thlt</td>
<td>25</td>
</tr>
<tr>
<td>Txf</td>
<td>30/36</td>
</tr>
<tr>
<td>Tterm</td>
<td>12</td>
</tr>
<tr>
<td>Tmov</td>
<td>19</td>
</tr>
<tr>
<td>Tst</td>
<td>24</td>
</tr>
<tr>
<td>Tlc</td>
<td>19</td>
</tr>
<tr>
<td>Treload</td>
<td>53</td>
</tr>
<tr>
<td>Tdec</td>
<td>10</td>
</tr>
<tr>
<td>Tsi</td>
<td>27</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Step</th>
<th>Execution Time (Clocks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>8</td>
</tr>
<tr>
<td>Tsd1</td>
<td>56</td>
</tr>
<tr>
<td>Tstore1</td>
<td>64</td>
</tr>
<tr>
<td>Tstore2</td>
<td>96</td>
</tr>
<tr>
<td>Ten</td>
<td>14</td>
</tr>
<tr>
<td>Tsave</td>
<td>44</td>
</tr>
<tr>
<td>Trem</td>
<td>26</td>
</tr>
<tr>
<td>Teoi</td>
<td>14</td>
</tr>
<tr>
<td>Tload</td>
<td>28</td>
</tr>
<tr>
<td>Tsi</td>
<td>31</td>
</tr>
<tr>
<td>Tic</td>
<td>37 - (12/N)</td>
</tr>
<tr>
<td>Tdis</td>
<td>26</td>
</tr>
<tr>
<td>Trest</td>
<td>32</td>
</tr>
<tr>
<td>Tpc</td>
<td>61</td>
</tr>
<tr>
<td>Tloadp</td>
<td>47</td>
</tr>
<tr>
<td>Tmov</td>
<td>10</td>
</tr>
<tr>
<td>Tlc</td>
<td>33 - (27/N)</td>
</tr>
<tr>
<td>Tret</td>
<td>20</td>
</tr>
<tr>
<td>Tiret</td>
<td>24</td>
</tr>
<tr>
<td>Tip</td>
<td>61</td>
</tr>
<tr>
<td>Tid</td>
<td>2</td>
</tr>
<tr>
<td>Step</td>
<td>Execution Time (Clocks)</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Tpb</td>
<td>103</td>
</tr>
<tr>
<td>Tcb</td>
<td>55</td>
</tr>
<tr>
<td>Tca</td>
<td>10</td>
</tr>
<tr>
<td>g(Tccw)</td>
<td>18</td>
</tr>
<tr>
<td>g(Tload)</td>
<td>13</td>
</tr>
<tr>
<td>Tterm</td>
<td>12</td>
</tr>
<tr>
<td>Tiomov</td>
<td>22</td>
</tr>
<tr>
<td>Tst</td>
<td>27</td>
</tr>
<tr>
<td>Tlc</td>
<td>19</td>
</tr>
<tr>
<td>Txfer</td>
<td>30/36</td>
</tr>
</tbody>
</table>
APPENDIX C

CPU IDLE TIME DUE TO IOP BUS ACCESS

To determine $T_{io}$ for the IOP configurations, it is necessary to determine the amount of time that the CPU is idle due to IOP bus accesses. The following analysis makes these assumptions:

1. There is a single 8086 CPU without an 8087
2. No locked transfers are performed by either the 8086 or the 8089
3. There is only one 8089
4. Only one 8089 channel is active
5. No wait states are required
6. All instruction operands are word-aligned

Local IOP

In the local configuration, bus arbitration is performed by a protocol of pulses on the request/grant line connecting the 8086 and the 8089. Once the 8086 releases the bus, it continues executing instructions from its queue until it requires bus access for data or the queue is emptied. Let:

$P_i(i) = \text{the probability that the CPU will be idled in the } i\text{th clock cycle}$
Pc(i) = the probability that the CPU will be able to run in the ith clock cycle

Pb(i) = the probability that the CPU will attempt to access the bus in the ith clock cycle

Pq(i) = the probability that the queue will be emptied in the ith clock cycle

ith clock cycle = the ith CPU clock period after the bus is released by the CPU

bus cycle = a data transfer between the CPU and memory or an I/O device which requires 4 CPU clock periods

Because the idle time is dependent upon the queue parameters and the CPU instruction stream when the bus is released, only an average value of CPU idle time has meaning, and only mean values for the above probabilities can be estimated, where mean in this sense is an average over many bus request/grant cycles.

If over several request/grant cycles, it is assumed that all instructions in the 8086 instruction set are equally probable of being executed, the best approximation for Pb(i) is to average the probability of bus access in the ith clock cycle of each instruction over the entire instruction set, with all instructions equally weighted. That is:

\[ Pb(i) = \frac{1}{N} \sum_{i=1}^{N} \frac{B_i}{E_i} \]  

where,
N = the total number of instructions in the 8086 instruction set

Bi = the number of clock cycles required by the ith instruction for bus accesses during its execution excluding that required for fetch

Ei = the execution time of the ith instruction

The equiprobability assumption is fairly good because the 8086 instruction set has a instruction frequency distribution which is typical of those found in most programs. Performing the calculation implied by equation 60 using data provided by the iAPX 86,88 User's Manual (5):

\[ P_b(i) = 0.22 \]

Since in any one clock cycle, either a bus cycle is started or in progress, or another byte of code is fetched from the queue, these are mutually exclusive events that can be added to form the probability that the CPU will go idle in the ith clock cycle:

\[ P_b(i) + P_q(i) = P_i(i) \]  \hspace{1cm} (61)  
\[ P_c(i) + P_i(i) = 1 \]  \hspace{1cm} (62)  
\[ P_b(i) + P_q(i) + P_c(i) = 1 \]  \hspace{1cm} (63)  

When the CPU goes idle, \( P_c(i) \) is zero. Hence the maximum value of \( P_q(i) \) is found from equation 63:

\[ P_q(i)_{\text{max}} = 1 - P_b(i) = 1 - 0.22 = 0.78 \]
A typical plot of the queue length as a function of time is shown below.

Since $L_0$ and $T_i$ vary widely with the instructions present in the queue at bus release, no expression for $P_q(i)$ will be accurate in any particular case, but a general form for an average over several request/grant cycles is a decaying exponential:

$$L = (L_0)\exp(-A(i)) \quad (64)$$

For the time being, assume that $P_b(i)$ is zero. Then:

$$P_q(i) + P_c(i) = 1 \quad (65)$$

which implies,

$$L = L_0(1 - P_q(i)) = (L_0)\exp(-A(i)) \quad (66)$$

or

$$P_q(i) = 1 - \exp(-A(i)) \quad (67)$$

Over many request/grant cycles, assuming every instruction is equiprobable, an average of $L'$ bytes will be removed from the queue every $E'$ clocks, where $L'$ is
the mean instruction length in bytes and \( E' \) is the mean instruction execution time in clocks. Thus:

\[
\text{Lo} - L' = (\text{Lo})\exp(-A(E'))
\]  
(68)

Solving for \( A \) gives:

\[
A = -(1/E')\ln((\text{Lo} - L')/\text{Lo})
\]  
(69)

From the data in the *iAPX 86,88 User's Manual* (5):

\( L' = 2.87 \) bytes  
\( E' = 20.65 \) clocks

If it is assumed that at least 16 to 20 clocks separate successive request/grant cycles, the queue will refill, and a conservative value for \( \text{Lo} \) is 4 bytes. Using equation 69:

\[
A = 0.061
\]

Since \( Pq(i)_{\text{max}} \) is 0.78, the fact that \( Pb(i) \) is not zero as was assumed above can be corrected by modifying equation 67 as:

\[
Pq(i) = 0.78(1 - \exp(-0.061i))
\]  
(70)

because

\[
Pb(i) = 0.22
\]

and

\[
Pc(i) = 1 - Pb(i) - Pq(i)
\]
then,

\[ P_c(i) = 0.78 \exp(-0.061i) \]  \hspace{1cm} (71)

If \( P_r(n) \) denotes the probability that the CPU is still running in the \( n \)th clock cycle, then \( P_r(n) \) is given by:

\[ P_r(n) = \frac{n}{\sum_{i=1}^{n} P_c(i)} \]  \hspace{1cm} (72)

If \( P_h(n) \) denotes the probability that the CPU is not still running in the \( n \)th clock cycle, then:

\[ P_h(n) = 1 - P_r(n) \]  \hspace{1cm} (73)

If \( x \) denotes the number of consecutive clock cycles between bus release and return, and \( f(x) \) denotes the CPU idle time, then:

\[ f(x) = \sum_{n=1}^{x} P_h(n) \]  \hspace{1cm} (74)

Substituting equation 71 into 72, and then 72 into 73 and 74, \( f(x) \) is given by:

\[ f(x) = \sum_{n=1}^{x} (1 - (0.78)^n \prod_{i=1}^{n} (0.941)^i) \]  \hspace{1cm} (75)
Remote IOP

With the remote IOP, the CPU can only be idled by IOP accesses to the Multibus. In the local configuration, it is idled by IOP accesses to the I/O bus as well as the Multibus. This difference means that the IN and OUT instructions are not affected by the IOP; therefore, these should not be included in the calculation of Pb(i). Since this involves only 8 instructions out of over 13000, the resulting Pb(i) will be identical for all practical purposes. Consequently, equation 75 is equally valid for the remote IOP case. It is not valid for a CPU with a resident bus, however.

Restrictions and Simplifications

The restriction that 16 to 20 clocks separate consecutive bus requests by the IOP results in a maximum device transfer rate of 1/20 bytes per clock to maintain the validity of equation 75.

It can be shown that an asymptotic expression for f(x) when x is greater than or equal to 12 is:

\[ f(x) = x - 2 \quad \text{for} \quad x \geq 12 \]  \hspace{1cm} (76)
APPENDIX D

DEMONSTRATION SOFTWARE LISTINGS

This appendix contains the listings for the demonstration system software. The first four modules are written in PL/M-86 (13,14) and are ultimately linked together and burned into the 86/12A EPROMs. The fifth module is written in ASM-89 (15) and contains the channel programs which are burned into the remote IOP EPROM.
Main Demonstrator Module: Contains demonstration calling sequence which uses procedures in other modules to obtain demonstration parameters from the user and setup the demonstration. Once the demonstration has been prepared, the module branches to one of three procedures which each implement the difference equation:

\[ y(k) - 1.3y(k - 1) + 0.4y(k - 2) = x(k) + 3x(k - 1) + x(k - 2) \]

with \( x(k) = \text{unit step} \) and \( y(-1), y(-2), x(-1), x(-2) = 0 \).

One procedure, or process, is provided for each I/O technique. Each procedure calculates successive values for the difference equation and transmits them to the terminal until the 8086 is interrupted by a time out counter.
PL/M-86 Compiler  MDS-86 I/O Demonstrator

Main Demonstrator Module

**PL/M-86 Compiler  MDS-86 I/O Demonstrator**

3/6/83

- **cpu**: Central processing unit, i.e., 8086
- **lf**: Line Feed
- **cr**: Carriage Return
- **toe**: CPU time out counter. Interrupts CPU upon terminal count.
- **eoi**: End of interrupt command for PIC

---

```plaintext
PUBLIC VARIABLES & LABELS

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>Indicates which board is connected to the terminal. It is one if the IOP is connected, zero if the 86/12A is connected.</td>
</tr>
<tr>
<td>Type</td>
<td>Indicates the type of I/O to be demonstrated when the 86/12A is connected to the terminal. It is one if interrupt I/O is desired, zero if polled I/O is desired.</td>
</tr>
<tr>
<td>Data Ready</td>
<td>Flag used to synchronize the interrupt process and the interrupt output procedure.</td>
</tr>
<tr>
<td>Digit Count</td>
<td>Equal to the number of digits in the current equation output which have been transmitted to the terminal. Used by the interrupt output procedure.</td>
</tr>
<tr>
<td>Char Count</td>
<td>Equal to the number of equation outputs which have been transmitted on the current line of the terminal. Used by all process procedures.</td>
</tr>
<tr>
<td>Computation Time</td>
<td>Equal to the time required to compute each output. Selected by the user; it is converted to 0.1 msec units, and then adjusted for the known time to compute each equation output, with the adjusted value used to simulate more computation time in each procedure.</td>
</tr>
<tr>
<td>Outputs</td>
<td>Equal to the number of equation outputs that have been transmitted to the terminal.</td>
</tr>
<tr>
<td>y of k</td>
<td>Three element FIFO storing the significant current and past outputs of the equation. y_of_k(0) is the current output.</td>
</tr>
<tr>
<td>iop out go</td>
<td>Flag used to synchronize IOP output procedure for IOP process.</td>
</tr>
<tr>
<td>iop out buff</td>
<td>Output buffer for IOP output procedure.</td>
</tr>
</tbody>
</table>
```
DECLARE RESTART LABEL PUBLIC;
DECLARE (CONNECTION, TYPE, DATA_READY, DIGIT_COUNT, CHAR_COUNT) BYTE PUBLIC;
DECLARE COMPUTATION_TIME WORD PUBLIC;
DECLARE OUTPUTS WORD PUBLIC;
DECLARE Y_OF_K(3) INTEGER PUBLIC;
DECLARE IOP_OUT.GO BYTE AT (07002H);
DECLARE IOP_OUT_BUFF(8) BYTE AT (06000H);

/*------------------------------- PRIVATE VARIABLES -------------------------------*/

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>x of k</td>
<td>Three element fifo storing current and significant past values of the input, x of k(0) is the current input.</td>
</tr>
</tbody>
</table>

DECLARE X_OF_K(3) INTEGER;

DECLARE UNTIL LITERALLY 'WHILE NOT';
FOREVER       LITERALLY 'WHILE 1';
NEW_MASK      LITERALLY '0';
ENABLE_TXRDY  LITERALLY '1111#0101B';
COUNTERS_ON   LITERALLY '1';

/*--------------------------------- PROCEDURES ---------------------------------*/

Procedure          Function
-------------------- -----------------------------------------------
iop process out start Loads data structures required to start iop output channel program.
(iop utilities module) Issues channel attention and waits for acknowledgement.

pic manager
(cpu utilities module)
Either loads new mask into 8259 pic or sends an end-of-interrupt (eoi) command to pic, as specified by op code parameter. Mask is mask to be
loaded and ir level is request line
whose in service bit is to be reset.
Either sets up the ppi, or sets/resets
the port bit which controls the
cpu time out counter.
output string
Outputs a string of bytes to the
terminal using polled i/o given a
pointer to the base of the string and
the offset of the last string element.
If or
Transmits a carriage return and either
one or two line feeds to the terminal
based on the one or two parameter.
hex to ascii
Converts the hex parameter to an
ascii string in a eight byte buffer
starting at the pointer parameter.
fixed or float specifies either one
or two decimal places should appear
in the ascii string.
demo setup
Obtains demonstration parameters
from user and prepares demonstration
system accordingly.
*/

11 1 IOP_PROCESS_OUT_START: PROCEDURE EXTERNAL;
12 2 END IOP_PROCESS_OUT_START;
13 1 PIC_MANAGER: PROCEDURE(OP_CODE, MASK, IR_LEVEL) EXTERNAL;
14 2 DECLARE (OP_CODE, MASK, IR_LEVEL) BYTE;
15 2 END PIC_MANAGER;
16 1 PPI_MANAGER: PROCEDURE(OP_CODE) EXTERNAL;
17 2 DECLARE OP_CODE BYTE;
18 2 END PPI_MANAGER;
19 1 OUTPUT$STRING: PROCEDURE(ASCII_PTR, ELEMENTS) EXTERNAL;
20 2 DECLARE ASCII_PTR POINTER, ELEMENTS BYTE;
21 2 END OUTPUT$STRING;
22 1 LF_CR: PROCEDURE(ONE OR TWO) EXTERNAL;
23 2 DECLARE ONE OR TWO BYTE;
24 2 END LF_CR;
25 1 HEX_TO_ASCII: PROCEDURE (HEX, STRING_PTR, FIXED OR FLOAT) EXTERNAL;
26 2 DECLARE HEX INTEGER:
DECLARE STRING_PTR POINTER;
DECLARE FIXED OR FLOAT BYTE;
END HEX TO ASCII;

DEMO SETUP: PROCEDURE EXTERNAL;
END DEMO SETUP;

/*----------------------------- PRIVATE PROCEDURES -----------------------------*/

POLLED_PROCESS: PROCEDURE;
/* buffer for ascii representation of equation output */
DECLARE OUTBUFF(8) BYTE;
/* enable interrupts from pic */
ENABLE:
CHAR_COUNT OUTPUTS = 0;
/* set equation initial conditions */
Y_OF_K(0) + Y_OF_K(1) + Y_OF_K(2) = 0;
X_OF_K(0) + X_OF_K(1) + X_OF_K(2) = 0;
/* repeat until interrupted by toc */
DO FOREVER:
/* simulated computation time, time procedure provides delay equal to 0.1 msec times value of parameter */
CALL TIME(COMPUTATION_TIME);
/* calculate next equation output */
X_OF_K(2) = X_OF_K(1);
X_OF_K(1) = X_OF_K(0);
X_OF_K(0) = 10;
Y_OF_K(2) = Y_OF_K(1);
Y_OF_K(1) = Y_OF_K(0);
Y_OF_K(0) = (10*X_OF_K(0) + 30*X_OF_K(1) + 10*X_OF_K(2) + 13*Y_OF_K(1) - 4*Y_OF_K(2))/10;
/* if current terminal line is full */
IF CHAR_COUNT = 10
THEN DO:

/* then insert a lf cr and restart char counter */
CALL LF_CR(0);
CHAR_COUNT = 0;
END;
/* convert current output to ascii and fill buffer */
CALL HEX_TO_ASCII(Y_OF_K(0),@OUTBUFF,1);
/* transmit current output */
CALL OUTPUTS_STRING(@OUTBUFF,7);
/* update output counter */
OUTPUTS = OUTPUTS + 1;
/* update char counter */
CHAR_COUNT = CHAR_COUNT + 1;
END;
END POLLED_PROCESS;

INTERRUPT_PROCESS: PROCEDURE:

/* initialize counters */
CHAR_COUNT, OUTPUTS, DIGIT_COUNT = 0;
/* precalculate first output to satisfy first uart interrupt request coming as soon as the interrupt path is open */
X_OF_K(0), Y_OF_K(0) = 10;
X_OF_K(1), X_OF_K(2), Y_OF_K(1), Y_OF_K(2) = 0;
/* unmask uart interrupt requests at the pic */
CALL PIC_MANAGER(NEW_MASK, ENABLE_TRDLY, 3);
/* repeat until interrupted by toc */
DO FOREVER;
/* enable pic interrupts */
ENABLE;
/* signal data available for interrupt procedure */
DATA_READY = 1;
/* simulated computation time, time procedure provides... */
delay equal to 0.1 msec times value of parameter x/

65 3 CALL TIME(COMPUTATION_TIME);
/* calculate next output */
66 3 X_OF_K(2) = X_OF_K(1);
67 3 X_OF_K(1) = X_OF_K(0);
68 3 X_OF_K(0) = 10;
69 3 Y_OF_K(2) = Y_OF_K(1);
70 3 Y_OF_K(1) = Y_OF_K(0);
71 3 Y_OF_K(0) = (16*Y_OF_K(0) + 30*X_OF_K(1) + 10*X_OF_K(2) +
13*Y_OF_K(1) - 4*Y_OF_K(2))/10;
/* wait until interrupt output procedure has completed
transmission of previous output */
72 3 DO UNTIL DATA READY = 0;
73 4 END;
/* disable PIC interrupts while new mask is loaded */
74 3 DISABLE;
/* unmask USART requests masked by interrupt output procedure
when it completed */
75 3 CALL PIC MANAGER(NEW MASK; ENABLE TX; READY ;);
76 3 END;
77 2 END INTERRUPT PROCESS;
/*----------------------------------------------------------*/
| iop process: implements difference equation with
| a scale factor of 10. Output is in
| 0.1 units. Uses iop output through
| iop USART.
| globals accessed: outputs: y of k; outputs: iop out go
| iop outbuf
|----------------------------------------------------------*/

78 1 IOP_PROCESS PROCEDURE;
/* enable PIC interrupts */
79 2 ENABLE;
80 2 OUTPUTS = 0;
/* set up equation initial conditions */
81 2 Y_OF_K(0); Y_OF_K(1); Y_OF_K(2) = 0;
82 2 X_OF_K(0); X_OF_K(1); X_OF_K(2) = 0;
/* start iop output channel program */
83 2 CALL IOP_PROCESS_OUT START;
/* repeat until interrupted by tic */
DO FOREVER;
    /* simulated computation time, time provides delay of 0.1 msec times parameter value */
    CALL TIME(COMPUTATION_TIME);
    /* calculate next equation output */
    X_OF_K(2) = X_OF_K(1);
    X_OF_K(1) = X_OF_K(0);
    X_OF_K(0) = 10;
    Y_OF_K(2) = Y_OF_K(1);
    Y_OF_K(1) = Y_OF_K(0);
    Y_OF_K(0) = (10*X_OF_K(0) + 30*Y_OF_K(1) + 10*X_OF_K(2) + 13*Y_OF_K(1) - 9*Y_OF_K(2))/10;
    /* wait until channel program completes transmission of previous output */
    DO WHILE IOP_OUT_GO = 1;
    END;
    /* convert current output to ascii and place in iop output buffer */
    CALL HEX_TO_ASCII(Y_OF_K(0),@IOP_OUTPUTBUFF1);
    /* signal new data available for channel program */
    IOP_OUT_GO = 1;
    /* update output count */
    OUTPUTS = OUTPUTS + 1;
    END;
END IOP_PROCESS;

******************************************************************************* EXECUTABLE STATEMENTS *******************************************************************************

/* disable pic interrupts until it is intialized */
DISABLE;
/* assume iopb connected to the terminal */
CONNECTION = 1;
/* restart loaction if the assumption was wrong */
RESTART: DISABLE;
/* setup demonstration, if the above assumption was wrong, the 8086 will receive a ready time out interrupt when the iopb does not respond to a channel attention. This will vector execution to an interrupt routine which corrects the assumption by setting connection equal to zero and branching to restart */
CALL DEMO_SETUP;
/* if iopb is really connected */
IF CONNECTION = 1
THEN DO;
    /* then start toc and invoke iop process */
    CALL PPI_MANAGER(COUNTERS_ON);
    CALL IOP_PROCESS;
    END;
ELSE DO;
/* otherwise 86/12A connected */
IF TYPE = 0
THEN DO;
    CALL PPI_MANAGER(COUNTERS_ON);
    CALL POLLED_PROCESS;
    END;
ELSE DO;
/* otherwise, interrupt i/o desired, start toc and invoke interrupt process */
    CALL PPI_MANAGER(COUNTERS_ON);
    CALL INTERRUPT_PROCESS;
    END;
END MAIN_DEMONSTRATOR_MODULE;
<table>
<thead>
<tr>
<th>DEFN ADDR</th>
<th>SIZE</th>
<th>NAME, ATTRIBUTES, AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 0014H</td>
<td>1</td>
<td>CHAR_COUNT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE PUBLIC</td>
</tr>
<tr>
<td>4 0000H</td>
<td>2</td>
<td>COMPUTATION_TIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WORD PUBLIC</td>
</tr>
<tr>
<td>3 0010H</td>
<td>1</td>
<td>CONNECTION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE PUBLIC</td>
</tr>
<tr>
<td>10 000H</td>
<td>1</td>
<td>COUNTERS_OK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LITERALLY</td>
</tr>
<tr>
<td>3 0012H</td>
<td>1</td>
<td>DATA_READY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE PUBLIC</td>
</tr>
<tr>
<td>30 0000H</td>
<td>1</td>
<td>DEMO_SETUP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROCEDURE EXTERNAL(6) STACK=0000H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 0013H</td>
<td>1</td>
<td>DIGIT_COUNT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE PUBLIC</td>
</tr>
<tr>
<td>19 0000H</td>
<td>1</td>
<td>ELEMENTS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE PARAMETER</td>
</tr>
<tr>
<td>10 000H</td>
<td>1</td>
<td>ENABLE_TXCROY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LITERALLY</td>
</tr>
<tr>
<td>25 0000H</td>
<td>1</td>
<td>FIXED_OR_FLOAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE PARAMETER</td>
</tr>
<tr>
<td>10 000H</td>
<td>1</td>
<td>FOREVER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LITERALLY</td>
</tr>
<tr>
<td>25 0000H</td>
<td>2</td>
<td>HEX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTEGER PARAMETER</td>
</tr>
<tr>
<td>25 0000H</td>
<td>1</td>
<td>HEX_TO_ASCII</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROCEDURE EXTERNAL(5) STACK=0000H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57 0155H</td>
<td>248</td>
<td>INTERRUPT_PROCESS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROCEDURE STACK=0000CH</td>
</tr>
<tr>
<td>8 0000H</td>
<td>8</td>
<td>IOP_OUTBUFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE ARRAY(8) AT ABSOLUTE</td>
</tr>
<tr>
<td>7 7002H</td>
<td>1</td>
<td>IOP_OUTGO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE AT ABSOLUTE</td>
</tr>
<tr>
<td>78 0240H</td>
<td>225</td>
<td>IOP_PROCESS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROCEDURE STACK=0000EH</td>
</tr>
<tr>
<td>11 0000H</td>
<td>1</td>
<td>IOP_PROCESS_OUT_START</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROCEDURE EXTERNAL(0) STACK=0000H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 0000H</td>
<td>1</td>
<td>IR_LEVEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE PARAMETER</td>
</tr>
<tr>
<td>22 0000H</td>
<td>1</td>
<td>LF_CR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROCEDURE EXTERNAL(4) STACK=0000H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 000AH</td>
<td>92</td>
<td>MAIN_DEMONSTRATOR_MODULE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROCEDURE STACK=0010H</td>
</tr>
<tr>
<td>13 0000H</td>
<td>1</td>
<td>MASK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE PARAMETER</td>
</tr>
<tr>
<td>10 000H</td>
<td>1</td>
<td>NEW_MASK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LITERALLY</td>
</tr>
<tr>
<td>22 0000H</td>
<td>1</td>
<td>ONE_OP_THO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE PARAMETER</td>
</tr>
<tr>
<td>16 0000H</td>
<td>1</td>
<td>OP_CODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE PARAMETER</td>
</tr>
<tr>
<td>13 0000H</td>
<td>1</td>
<td>OP_CODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE PARAMETER</td>
</tr>
<tr>
<td>33 0015H</td>
<td>8</td>
<td>OUTBUFF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BYTE ARRAY(8)</td>
</tr>
<tr>
<td>5 0002H</td>
<td>2</td>
<td>OUTFUTS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WORD PUBLIC</td>
</tr>
<tr>
<td>19 0000H</td>
<td>1</td>
<td>OUTPUTSTRING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROCEDURE EXTERNAL(3) STACK=0000H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PL/M-86 Compiler     MCS-86 I/O Demonstrator  
Main Demonstrator Module  

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
<th>Stack</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000H</td>
<td>PIC_MANAGER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0006H</td>
<td>POLLED_PROCESS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0009H</td>
<td>PPI_MANAGER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010H</td>
<td>RESTART</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0012H</td>
<td>TIME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013H</td>
<td>STRING_PTR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0014H</td>
<td>TYPE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0015H</td>
<td>UNTIL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0009H</td>
<td>X_OF_K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0004H</td>
<td>Y_OF_K</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Module Information:

- Code Area Size = 032EH  814D
- Constant Area Size = 0000H  00
- Variable Area Size = 0010H  29D
- Maximum Stack Size = 0010H  16D
- 372 Lines Read
- 0 Program Error(s)

End of PL/M-86 Compilation
DEMONSTRATOR SUPPORT MODULE

Demonstrator Support Module: Contains procedures used to prepare demonstration and establish serial communication with the terminal. Also contains all interrupt procedures.

PUBLIC VARIABLES

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu time</td>
<td>equals execution time available to process procedures. Units are seconds. Used to load toc.</td>
</tr>
</tbody>
</table>

EXTERNAL VARIABLES

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>see main demonstrator module public variables</td>
</tr>
</tbody>
</table>
data ready  
char count  
digit count  
connection  
computation time  
outputs  
y of k.

DECLARE RESTART LABEL EXTERNAL;

DECLARE (TYPE, DATA_READY, CHAR_COUNT, DIGIT_COUNT, CONNECTION) BYTE EXTERNAL;
DECLARE COMPUTATION_TIME WORD EXTERNAL;
DECLARE OUTPUTS WORD EXTERNAL;
DECLARE Y_OF_K (3) INTEGER EXTERNAL;

/*--------------------------------PRIVATE VARIABLES-----------------------------------------------*/

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>io process delay</td>
<td>equals the amount of time required to process each byte of an equation output in 0.1 msec units, used by interrupt output procedure.</td>
</tr>
<tr>
<td>int array</td>
<td>8086 interrupt vector table. 256 4 byte pointer entries. Pointers are locations of interrupt procedure entry points.</td>
</tr>
</tbody>
</table>

DECLARE IO_PROCESS_DELAY WORD;

DECLARE INT_ARRAY(256) POINTER AT (00000H);

DECLARE CR
   LITERALLY '0DH';
LF
   LITERALLY '0AH';
SETUP
   LITERALLY '0';
COUNTERS_OFF
   LITERALLY '2';
EDI
   LITERALLY '1';
NEW_Mask
   LITERALLY '0';
ENABLE_TXRDY
   LITERALLY '1111$0101B';
ENABLE_TIME_OUT
   LITERALLY '1111$1101B';
DISABLE_TXRDY
   LITERALLY '1111$1101B';
ENABLE_RTO
   LITERALLY '1111$1108';
PL/H-B6 COMPILER  MCS-B6 I/O DEMONSTRATOR
DEMONSTRATOR SUPPORT MODULE  3/6/83

102

DECLARE PIT_CONTROL LITERALLY '000D6H'; /\ 8253 control register \/
COUNTER_2   LITERALLY '000D4H'; /\ 8253 counter 2 \/
USART_CONTROL LITERALLY '000DAH'; /\ 8251 control register \/
USART_DATA   LITERALLY '000DBH'; /\ 8251 data buffer \/
PORT_C       LITERALLY '000CCH'; /\ 8255 port c \/

/XXXXXXXXXXXXXXXXXXXXXXXXXX PROCEDURES XXXXXXXXXXXXXXXXXXXXXXXXXXX/

/\--------------------------------- EXTERNAL PROCEDURES ----------------------\

Procedure            Function
--------------        --------------
IOP INIT             Initializes 8089 for 8 bit system bus
(IOP utilities module)
and 8 bit private I/O bus. Passes
location of channel control blocks.
IOP UART SETUP       Causes 8089 to run channel program
(IOP utilities module)
which sets up IOP UART and PIT.
IOP Halt             Causes 8089 to gracefully halt its
(IOP utilities module)
current channel program.
PIC SET UP           Initializes PIC with all interrupts
(CPU utilities module)
masked out.
PIC MANAGER          See main demonstrator module for
(CPU utilities module)
brief description.
CPU TIME OUT LOADER  Loads PIT timer that provides
(CPU utilities module)
cpu interrupt on time out, i.e., loads
the TOC with 1000 times CPU time.
PP1 MANAGER          See main demonstrator module for
(CPU utilities module)
brief description.
OUTPUT STRING       See main demonstrator module for
(CPU utilities module)
brief description.
IF OR                See main demonstrator module for
(CPU utilities module)
brief description.
ECHO BYTE            Returns a byte input from the
(CPU utilities module)
and echoes the input to the terminal.
HEX TO ASCII         See main demonstrator module for
(CPU utilities module)
for brief description.
INPUT STRING         Returns a word (two bytes) of input
(CPU utilities module)
from the terminal and echoes the
input to the terminal. Performs
ASCII to hex conversion, so word
returned may encode up to five input
IOP_INIT: PROCEDURE EXTERNAL;
END IOP_INIT;

IOP_USART_SETUP: PROCEDURE EXTERNAL;
END IOP_USART_SETUP;

IOP_HALT: PROCEDURE EXTERNAL;
END IOP_HALT;

PIC_SETUP: PROCEDURE EXTERNAL;
END PIC_SETUP;

PIC_MANAGER: PROCEDURE (OP_CODE, MASK, IR_LEVEL) EXTERNAL;
DECLARE (OP_CODE, MASK, IR_LEVEL) BYTE;
END PIC_MANAGER;

CPU_TIME_OUT_LOADER: PROCEDURE EXTERNAL;
END CPU_TIME_OUT_LOADER;

PPI_MANAGER: PROCEDURE (OP_CODE) EXTERNAL;
DECLARE OP_CODE BYTE;
END PPI_MANAGER;

OUTPUT$STRING: PROCEDURE (ASCII_PTR, ELEMENTS) EXTERNAL;
DECLARE ASCII_PTR POINTER, ELEMENTS BYTE;
END OUTPUT$STRING;

LF_CR: PROCEDURE (ONE_OR_TWO) EXTERNAL;
DECLARE ONE_OR_TWO BYTE;
END LF_CR;

ECHO_BYTE: PROCEDURE BYTE EXTERNAL;
END ECHO_BYTE;

HEX_TO_ASCII: PROCEDURE (HEX, STRING_PTR, FIXED_OR_FLOAT) EXTERNAL;
DECLARE HEX INTEGER;
DECLARE STRING_PTR POINTER;
DECLARE FIXED_OR_FLOAT BYTE;
END HEX_TO_ASCII;

INPUT$STRING: PROCEDURE WORD EXTERNAL;
END INPUT$STRING;

/*------------------------------------ PRIVATE PROCEDURES --------------------------------*/

/*
 * int vector load: loads the entry points of the four
 * interrupt procedures used into the
 * interrupt vector table.
 */

INTVECTOR_LOAD: PROCEDURE;
INT_ARRAY(1) = INTERRUPT$PTR(OVERFLOW_HANDLER);
INT_ARRAY(128) = INTERRUPT$PTR(GET_CONNECTION);
INT_ARRAY(129) = INTERRUPT$PTR(TIME_OUT_HANDLER);
INT_ARRAY(131) = INTERRUPT$PTR(INTERRUPT_OUTPUT);
END INTVECTOR_LOAD;

/*
 * link terminal: based upon the state of connection,
 * sets up a serial connection to the
 * terminal. If the iopb is connected,
 * the iop is initialized and its uart
 * and baud rate generator are prepared.
 * Otherwise, the 86/12 uart and baud rate
 * generator are prepared. Note that since
 * the main module initially assumes that
 * the iopb is connected, the call to iop
 * init will generate a ready time out interrupt
 * if its is not connected, the interrupt service
 * routine will correct the assumption as
 * explained in main demonstrator module executable
 * statements.
 */

DECLARE
51  2

53  3

55  3

56  2

57  3

58  3

59  3

60  3

61  3

62  3

63  2

*/

/* demo data acquisition: requests demonstration parameters
   from user and loads parameters into appropriate variables, in particular,
   obtains values for type, cpu time, computation time and io process delay,
   connection is determined by which board is connected to the terminal, and
   this is known by the time this procedure
DEM-O_DATA_ACQUISITION: PROCEDURE;

DECLARE SIGN_ON (X) BYTE DATA
('MCS-86 I/O DEMONSTRATOR, V1.0',CR,LF);

DECLARE TIME_PROMPT (X) BYTE DATA
('ENTER AVAILABLE CPU TIME IN SECONDS := ');

DECLARE TYPE_PROMPT (X) BYTE DATA
('ENTER I/O TYPE: POLLED OR INTERRUPT (P OR I) := ');

DECLARE COMPUTE_PROMPT (X) BYTE DATA
('ENTER CPU TIME REQUIRED TO COMPUTE EACH OUTPUT, ',LF,CR,'IN HILLISECONDS := ');

DECLARE IO_DELAY_PROMPT (X) BYTE DATA
('ENTER TIME REQUIRED TO PROCESS EACH OUTPUT BYTE, ',LF,CR,'IN TENTHS OF HILLISECONDS := ');

CALL OUTPUT$STRING(@SIGN_ON+LAST(SIGN_ON));

IF CONNECTION = 0 THEN
    IF SIGN_ON = 2 THEN
        CALL OUTPUT$STRING(@TYPE_PROMPT+LAST(TYPE_PROMPT));
        IF TYPE = 'ECHO_BYTE' THEN
            CALL LF_CR(1);
        ELSE
            CALL LF_CR(1);
            IF TYPE = 'I' THEN
                TYPE = 1;
            ELSE
                TYPE = 0;
            END;
        END;
    ELSE
        CALL OUTPUT$STRING(@TIME_PROMPT+LAST(TIME_PROMPT));
        CPU_TIME = INPUT$STRING;
    END;
END;
82 2 /* 1 if, icr */
83 2 CALL LF_CR(0);
84 2 /* request time needed to compute each equation output */
85 2 CALL OUTPUT$STRING($COMPUTE_PROMPT+LAST($COMPUTE_PROMPT));
86 2 /* input answer and echo, convert to hex, change to 0.1 msec
87 2 units, and subtract known time to compute each real output */
88 2 COMPUTATION_TIME = 10*INPUT$STRING - 5; /* TIME IN 0.1 MS -.5 MS */
89 2 /* 1 if, icr */
90 2 CALL LF_CR(0);
91 2 /* if interrupt i/o desired */
92 2 IF TYPE = 1 THEN DO:
93 3 /* request time needed to process each byte of equation output */
94 3 CALL OUTPUT$STRING($IO_DELAY_PROMPT+LAST($IO_DELAY_PROMPT));
95 3 /* input answer and echo, convert to hex */
96 3 IO_PROCESS_DELAY = INPUT$STRING;
97 3 /* 1 if, icr */
98 3 CALL LF_CR(0);
99 3 END;
100 2 END DEMO_DATA_ACQUISITION:

/*---------------------------------- PUBLIC PROCEDURES ----------------------------------*/
DEMO_SETUP: PROCEDURE PUBLIC;
  CALL PIC_SETUP;
  CALL PPI_MANAGER(SETUP);
  CALL INVECTOR_LOAD;
  CALL PPI_MANAGER(COUNTERS_OFF);
  IF CONNECTION = 1 THEN DO;
    CALL PIC_MANAGER(NEW_MASK, ENABLE_RTO, 0);
    ENABLE;
    END;
  CALL LINK_TERMINAL;
  CALL DEMO_DATA_ACQUISITION;
  CALL CPU_TIME_OUT_LOADER;
  CALL PPI_MANAGER(NEW_MASK, ENABLE_TIME_OUT, 1);
  END DEMO_SETUP;

    /* --------------------------------- INTERRUPT PROCEDURES --------------------------------- */

    /* ------------------------------ OVERFLOW_HANDLER INTERRUPT ------------------------------ */

OVERFLOW_HANDLER: PROCEDURE INTERRUPT 4;
  DECLARE OVERFLOW_COMMENT(*) BYTE DATA
    ('ARITHMETIC OVERFLOW',CR,LF);
  CALL OUTPUT*STRING(@OVERFLOW_COMMENT, LAST(OVERFLOW_COMMENT));
  HALT;
  END OVERFLOW_HANDLER;

    /* ------------------------------ INTERRUPT PROCEDURES --------------------------------- */
<table>
<thead>
<tr>
<th>globals accessed; connection, restart</th>
</tr>
</thead>
</table>

113 1 GET_CONNECTION: PROCEDURE INTERRUPT 128;
114 2  CONNECTION = 0;
115 2  CALL PIC_MANAGER(EOI;ENABLE_RTO;0);
116 2  GOTO RESTART;
117 2  END GET_CONNECTION;

118 1 TIME_OUT_HANDLER: PROCEDURE INTERRUPT 129;
119 2  /X mask which shuts out all requests to pic. */
120 2  DECLARE INITIAL_MASK LITERALLY 'OFFH';
121 2  /X encodes throughput in 0.01 outputs/sec, units */
122 2  DECLARE THROUGHPUT INTEGER;
123 2  /X buffer for ascii representation of throughput in outputs/sec */
124 2  DECLARE THRPUT_BUFFER (8) BYTE;
125 2  DECLARE THRPUT_COMMENT (X) BYTE DATA
      ('THROUGHPUT = ');
126 2  DECLARE THRPUT_UNITS (X) BYTE DATA
      ('OUTPUTS PER CPU SECOND';CR;LF;LF);
127 2  /X if iop connected */
128 2  IF CONNECTION = 1
129 2  THEN CALL IOP_HALTI;
130 2  /X calculate throughput */
131 2  THROUGHPUT = INT((100 * OUTPUTS) / CPU_TIME);
132 2  /X convert to ascii and correct units, fill buffer */
133 2  CALL HEX_TO_ASCII(THROUGHPUT;@THRPUT_BUFFER;0);
134 2  CALL LF_CR(1);
CALL OUTPUT$STRING(@THRUPUT_COMMENT,LAST(THRUPUT_COMMENT));
/* send throughput */
CALL OUTPUT$STRING(@THRUPUT_BUFFER,LAST(THRUPUT_BUFFER));
/* send units */
CALL OUTPUT$STRING(@THRUPUT_UNITS,LAST(THRUPUT_UNITS));
/* send eoi to pic to clear in service bit */
CALL PIC_MANAGER(EOI,ENABLE_TIME_OUT);1)
/* disable further interrupts to prevent interrupts during halt */
CALL PIC_MANAGER(NEW_MASK,INITIAL_MASK);1)
/* system stop */
HALT;
END TIME_OUT_HANDLER;

INTRERRUPT_OUTPUT: PROCEDURE INTERRUPT 131;
/* buffer for ascii version of equation output */
DECLARE INTER_BUFFER (8) BYTE;
/* if end of current line */
IF CHAR_COUNT = 10
THEN DO;
/* then insert a crlf and reset counter */
CALL LF_CR(0);
CHAR_COUNT = 0;

END:
/* if fresh output */

144 IF DIGIT_COUNT = 0
/* then convert to ascii */
THEN CALL HEX_TO_ASCII(Y_DF_K(0),@INTER_BUFFER,1);
/* simulated io processing, time procedure provides time delay equal to 0.1 msec times parameter */

145 CALL TIME(IO_PROCESS_DELAY);
/* send next byte of output */
OUTPUT(USART_DATA) = INTER_BUFFER(DIGIT_COUNT);
/* update byte counter */

146 DIGIT_COUNT = DIGIT_COUNT + 1;
/* send eoi to pic to clear in service bit */

147 CALL PIC_MANAGER(EDT,ENABLE_TXRDY,3);
/* if entire output transmitted */
IF DIGIT_COUNT = 8

148 THEN DO:
/* then update outputs, char counter, digit count, clear data ready and disable further uart requests */
OUTPUTS = OUTPUTS + 1;
CHAR_COUNT = CHAR_COUNT + 1;
DIGIT_COUNT = 0;
DATA_READY = 0;

149 CALL PIC_MANAGER(NEW_MASK,DISABLE_TXRDY,3);

150 END:
151 END INTERRUPT_OUTPUT;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

157 END DEMONSTRATOR_SUPPORT_MODULE;
<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>SIZE</th>
<th>NAME, ATTRIBUTES, AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>28 0000H</td>
<td>4</td>
<td>ASCII_PTR, 28</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>BAUD_RATE_COUNT, 50</td>
</tr>
<tr>
<td>4 0000H</td>
<td>1</td>
<td>CHAR_COUNT, 4</td>
</tr>
<tr>
<td>5 0000H</td>
<td>2</td>
<td>COMPUTATION_TIME, 5</td>
</tr>
<tr>
<td>68 0078H</td>
<td>68</td>
<td>COMPUTE_PROMPT, 68</td>
</tr>
<tr>
<td>4 0000H</td>
<td>1</td>
<td>CONNECTION, 4</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>COUNTERS_OFF, 10</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>COUNTER_2, 11</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>COUNTER_2_MODE, 50</td>
</tr>
<tr>
<td>2 0000H</td>
<td>2</td>
<td>CPU_TIME, 2</td>
</tr>
<tr>
<td>23 0000H</td>
<td>1</td>
<td>CPU_TIME_OUT_LOADER, 23</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>CR, 10</td>
</tr>
<tr>
<td>4 0000H</td>
<td>1</td>
<td>DATA_READY, 4</td>
</tr>
<tr>
<td>1 014CH</td>
<td></td>
<td>DEMONSTRATOR_SUPPORT_MODULE, 1</td>
</tr>
<tr>
<td>64 01CFH</td>
<td>200</td>
<td>DEMO_DATA_ACQUISITION, 64</td>
</tr>
<tr>
<td>93 0297H</td>
<td>77</td>
<td>DEMO_SETUP, 93</td>
</tr>
<tr>
<td>4 0000H</td>
<td>1</td>
<td>DIGIT_COUNT, 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>DISABLE_TXRDY, 10</td>
</tr>
<tr>
<td>34 0000H</td>
<td></td>
<td>ECHO_BYTE, 34</td>
</tr>
<tr>
<td>28 0000H</td>
<td>1</td>
<td>ELEMENTS, 28</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>ENABLE_RTD, 10</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>ENABLE_TIME_OUT, 10</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>ENABLE_TXRDY, 10</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>EDI, 10</td>
</tr>
<tr>
<td>36 0000H</td>
<td>1</td>
<td>FIXED_OR_FLOAT, 36</td>
</tr>
<tr>
<td>113 032CH</td>
<td>54</td>
<td>GET_CONNECTION, 113</td>
</tr>
<tr>
<td>36 0000H</td>
<td>2</td>
<td>HEX, 36</td>
</tr>
<tr>
<td>36 0000H</td>
<td></td>
<td>HEX_TO_ASCII, 36</td>
</tr>
<tr>
<td>Address Code</td>
<td>Instruction Description</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td>119 0000H</td>
<td>INITIAL_MASK</td>
<td>BUILTIN 59</td>
</tr>
<tr>
<td>41 0000H</td>
<td>INPUTSTRING</td>
<td>LITERALLY 133</td>
</tr>
<tr>
<td>136 0402H</td>
<td>INTERRUPT_OUTPUT</td>
<td>PROCEDURE WORD EXTERNAL(20) STACK=0000H</td>
</tr>
<tr>
<td>137 006EH</td>
<td>INTER_BUFFER</td>
<td>BYTE ARRAY(8)</td>
</tr>
<tr>
<td>43 014CH</td>
<td>INVECTOR_LOAD</td>
<td>PROCEDURE STACK=0002H</td>
</tr>
<tr>
<td>9 0000H 1024</td>
<td>INT_ARRAY</td>
<td>POINTER ARRAY(256) AT ABSOLUTE</td>
</tr>
<tr>
<td>16 0000H</td>
<td>IOP_HALT</td>
<td>PROCEDURE EXTERNAL(11) STACK=0000H</td>
</tr>
<tr>
<td>12 0000H</td>
<td>IOP_INIT</td>
<td>PROCEDURE EXTERNAL(9) STACK=0000H</td>
</tr>
<tr>
<td>14 0000H</td>
<td>IOP_USART_SETUP</td>
<td>PROCEDURE EXTERNAL(10) STACK=0000H</td>
</tr>
<tr>
<td>69 00BCH</td>
<td>IO_DELAY_PROMT</td>
<td>BYTE ARRAY(79) DATA</td>
</tr>
<tr>
<td>8 0002H</td>
<td>IO_PROCESS_DELAY</td>
<td>WORD 89 145</td>
</tr>
<tr>
<td>20 0000H</td>
<td>IR_LEVEL</td>
<td>BYTE PARAMETER 21</td>
</tr>
<tr>
<td>10 0000H</td>
<td>LF</td>
<td>BUILTIN 70 73 80 83</td>
</tr>
<tr>
<td>31 0000H</td>
<td>LF_CR</td>
<td>LITERALLY 65 68 69 109</td>
</tr>
<tr>
<td>49 019FH</td>
<td>LINK_TERMINAL</td>
<td>PROCEDURE EXTERNAL(17) STACK=0000H</td>
</tr>
<tr>
<td>20 0000H</td>
<td>MASK</td>
<td>PROCEDURE STACK=0006H</td>
</tr>
<tr>
<td>10 0000H</td>
<td>NEW_MASK</td>
<td>BUILTIN 58</td>
</tr>
<tr>
<td>31 0000H</td>
<td>ONE_OF_TWO</td>
<td>BYTE PARAMETER 21</td>
</tr>
<tr>
<td>25 0000H</td>
<td>OP_CODE</td>
<td>LITERALLY 100 106 133 155</td>
</tr>
<tr>
<td>20 0000H</td>
<td>OP_CODE</td>
<td>BYTE PARAMETER 21</td>
</tr>
<tr>
<td>6 0000H</td>
<td>2 OUTPUTS</td>
<td>BYTE PARAMETER 21</td>
</tr>
<tr>
<td>28 0000H</td>
<td>OUTPUTSTRING</td>
<td>WORD EXTERNAL(7)</td>
</tr>
<tr>
<td>109 0106H</td>
<td>OVERFLOW_COMMENT</td>
<td>PROCEDURE EXTERNAL(16) STACK=0000H</td>
</tr>
<tr>
<td>108 02FDH</td>
<td>OVERFLOW_HANDLER</td>
<td>BYTE ARRAY(21) DATA</td>
</tr>
</tbody>
</table>

Legend:
- **BUILTIN**: Built-in procedure
- **LITERALLY**: Literally defined
- **PROCEDURE**: Procedure

Notes:
- Addresses and instructions are for a specific system (MCS-86)
- Addresses and instructions are in hexadecimal format
- Instructions and procedures are listed along with their respective stacks and byte addresses

Date: 3/6/83
### Module Information:

<table>
<thead>
<tr>
<th>Code Area Size</th>
<th>0490H</th>
<th>11680</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant Area Size</td>
<td>0000H</td>
<td>0D</td>
</tr>
<tr>
<td>Variable Area Size</td>
<td>0016H</td>
<td>220</td>
</tr>
<tr>
<td>Maximum Stack Size</td>
<td>002AH</td>
<td>420</td>
</tr>
<tr>
<td>Lines Read</td>
<td>509</td>
<td></td>
</tr>
<tr>
<td>Program Error(s)</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

End of PL/H-86 Compilation
CPU Utilities Module: Contains drivers for 86/12 user, pic, ppi, procedure which loads toc, utility procedures for terminal communication used in preparing demonstration.

**--- EXTERNAL VARIABLES ---**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu time</td>
<td>see demonstrator support module public variables</td>
</tr>
<tr>
<td>connection</td>
<td>see main demonstrator module public variables</td>
</tr>
</tbody>
</table>

2 1 DECLARE CPU_TIME WORD EXTERNAL;
3 1 DECLARE CONNECTION BYTE EXTERNAL;
DECLARE UNTIL LITERALLY 'WHILE NOT';
DECLARE CR LITERALLY '0DH', LF LITERALLY '0AH';

DECLARE PIT_CONTROL LITERALLY '000D6H', /* pit control register */
COUNTER_0 LITERALLY '000DOH', /* pit counter 0 */
COUNTER_1 LITERALLY '000D2H', /* pit counter 1 */
USART_CONTROL LITERALLY '000DAH', /* uart control register */
USART_STATUS LITERALLY '000DAH', /* uart status register */
USART_DATA LITERALLY '000D8H', /* uart data buffer */
PIC_CONTROL_A LITERALLY '000D8H', /* pic control register a */
PIC_CONTROL_B LITERALLY '000C2H', /* pic control register b */
PORT_B LITERALLY '000CAH'; /* port b */

EXTERNAL PROCEDURES

Procedure
---
IOP_INPUT: PROCEDURE BYTE EXTERNAL;
IOP_OUTPUT: PROCEDURE CHAR EXTERNAL;

Function
---
returns a single byte of input from the terminal when it is connected to the iopb.
transmits the byte parameter to the terminal when it is connected to the iopb.

pic setup! initializes pic and loads initial mask that blocks out all interrupt requests.
interrupt request lines and vectors:

<table>
<thead>
<tr>
<th>request line</th>
<th>source</th>
<th>vector</th>
<th>priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>ir0</td>
<td>ready time out</td>
<td>128</td>
<td>1</td>
</tr>
<tr>
<td>ir1</td>
<td>pit counter 1</td>
<td>129</td>
<td>2</td>
</tr>
<tr>
<td>ir2</td>
<td>pit counter 0</td>
<td>130</td>
<td>3</td>
</tr>
<tr>
<td>ir3</td>
<td>usart tty</td>
<td>131</td>
<td>4</td>
</tr>
<tr>
<td>ir4</td>
<td>usart rxrdy</td>
<td>132</td>
<td>5</td>
</tr>
<tr>
<td>ir5</td>
<td>multibus int1</td>
<td>133</td>
<td>6</td>
</tr>
<tr>
<td>ir6</td>
<td>multibus int2</td>
<td>134</td>
<td>7</td>
</tr>
</tbody>
</table>

PIC_SETUP PROCEDURE PUBLIC:

DECLARE /* edge triggered interrupts, 8086 mode */
ICW1 LITERALLY '00010011B', /* vectors 00H to 07H */
ICW2 LITERALLY '10000000B', /* non sfha, buffered master, specific eoi */
ICW4 LITERALLY '00001101B', /* masks out all requests */
INITIAL_MASK LITERALLY 'OFFH'; /* send initialization command words icw1,icw2,icw4, and initial mask */

OUTPUT(PIC_CONTROL_A) = ICW1;
OUTPUT(PIC_CONTROL_B) = ICW2;
OUTPUT(PIC_CONTROL_B) = ICW4;
OUTPUT(PIC_CONTROL_B) = INITIAL_MASK;
END PIC_SETUP;

pic manager: either loads new mask into pic, or sends
eoi command to pic.

parameters: op code - determines whether new mask loaded
or eoi command sent, if op code
is zero, the mask parameter is sent
to pic as new mask, if op code is
one, eoi command sent to reset in
service bit of ir request line given by ir level parameter.
mask - new mask for pic, don't care when op is one.
ir level - request line whose in service bit is to be reset, don't care when op code is zero.

---

**PIC_MANAGER**: PROCEDURE (OP_CODE, MASK, IR_LEVEL) REENTRANT PUBLIC;
DECLARE (OP_CODE, MASK, IR_LEVEL) BYTE;
DECLARE NEW_MASK LITERALLY 'OP_CODE = 0',
    SPECIFIC_EOI LITERALLY 'OP_CODE = 1';
/* if new mask desired, then send mask to pic */
IF NEW_MASK THEN OUTPUT (PIC_CONTROL_B) = MASK;
/* if eoi command desired */
IF SPECIFIC_EOI /* then send eoi to pic with ir level */
    THEN OUTPUT (PIC_CONTROL_A) = 60H OR (IR_LEVEL AND 07H);
END PIC_MANAGER;

---

**CPU_TIME_OUT_LOADER**: PROCEDURE PUBLIC;
DECLARE /* counter 0 setup for square wave output, binary counting */
    COUNTER_0_MODE LITERALLY '00111110B',
/* counter 1 setup for interrupt on terminal */
count output, binary counting */
COUNTER_1_MODE LITERALLY '01110000B';
/* frequency divisor loaded into counter 0
divides counter 0 input of 1.23 MHz by 1230
to give counter 1 input a 1 msec period. */
DECLARE FREQ_DIVISOR LITERALLY '1230';

/* send counter 0 mode byte */
OUTPUT(PIT_CONTROL) = COUNTER_0_MODE;
/* pit write recovery delay, i.e., a nop
typical for 6 */
OUTPUT(PORT_E) = 00H;
/* send low byte of frequency divisor to counter 0 */
OUTPUT(COUNTER_0) = LOW(FREQ_DIVISOR);
OUTPUT(PORT_E) = 00H;
/* send high byte of frequency divisor to counter 0 */
OUTPUT(COUNTER_0) = HIGH(FREQ_DIVISOR);
OUTPUT(PORT_E) = 00H;
/* send counter 1 mode byte */
OUTPUT(PIT_CONTROL) = COUNTER_1_MODE;
/* send low byte of cpu time in milliseconds to counter 1 */
OUTPUT(COUNTER_1) = LOW(1000 × CPU_TIME);
OUTPUT(PORT_E) = 00H;
/* send high byte of cpu time in milliseconds to counter 1 */
OUTPUT(COUNTER_1) = HIGH(1000 × CPU_TIME);
OUTPUT(PORT_E) = 00H;
END CPU_TIME_OUT_LOADER;

------------------------------------------------------------------------

| ppi_manager: either sets up the ppi, or sets/resets port c bit 7 which is connected to counter1 gate input |
| parameters: op code - if 0, ppi is initialized. if 1, toc is turned on, if 2 toc is turned off. |

43 1 PPI_MANAGER: PROCEDURE (OP_CODE) PUBLIC;

44 2 DECLARE OP_CODE BYTE;
45 2 DECLARE SETUP LITERALLY 'OP_CODE = 0';
DECLARE COUNTERS_ON LITERALLY 'OP_CODE = 1';
COUNTERS_OFF LITERALLY 'OP_CODE = 2';

DECLARE /* port a input; ports b and c output */
    PPI_MODE LITERALLY '1001$0001B',
    /* set port c bit 7 */
    GATES_ON LITERALLY '0000$1111B',
    /* reset port c bit 7 */
    GATES_OFF LITERALLY '0000$1110B';

IF SETUP THEN OUTPUT(PPI_CONTROL) = PPI_MODE;
IF COUNTERS_ON THEN OUTPUT(PPI_CONTROL) = GATES_ON;
IF COUNTERS_OFF THEN OUTPUT(PPI_CONTROL) = GATES_OFF;
END PPI_MANAGER;

/*
 | polled output: outputs byte to terminal when connected to
 | 86/12.
 | parameters: char - byte to be transmitted
="/*/

POLLED_OUTPUT: PROCEDURE(CHAR) REENTRANT PUBLIC;

DECLARE CHAR BYTE;

DECLARE /* usart transmission complete */
    TX_EMPTY LITERALLY 'SHR(INPUT(USART_STATUS),2)';
    /* 0.5 msec delay between successive outputs required
     * by adm3a. */
    CALL TIME(5);
    /* send byte */
    OUTPUT(USART_DATA) = CHAR;
    /* wait until transmission complete */
    DO UNTIL TX_EMPTY;
    END;
END POLLED_OUTPUT;

/*
 | polled input: receives a byte from the terminal and returns
 | it.
="/*/
POLLED_INPUT: PROCEDURE BYTE PUBLIC;
/* drives uart rts/ terminal cts/ low */
DECLARE ENABLE_ADM_XMIT LITERALLY '0010$01118';
/* drives uart rts/ terminal cts/ high */
DECLARE DISABLE_ADM_XMIT LITERALLY '0000$00118';
/* uart has acquired the input */
DECLARE CHAR_READY LITERALLY 'SHR(INPUT(USART_STATUS)1)';
/* temporary variable */
DECLARE DUMMY_READ BYTE;
/* allow the terminal to transmit */
OUTPUT(USART_CONTROL) = ENABLE_ADM_XMIT;
/* read the garbage from uart input buffer */
DUMMY_READ = INPUT(USART_DATA);
/* wait until input acquired */
DO UNTIL CHAR_READY;
END;
/* get the input */
DUMMY_READ = INPUT(USART_DATA);
/* prevent further transmission from the terminal */
OUTPUT(USART_CONTROL) = DISABLE_ADM_XMIT;
/* return the input to caller */
RETURN DUMMY_READ;
END POLLED_INPUT;

---------------------------------------------------------------
output string: outputs a string of bytes to the terminal
regardless of its connection.
parameters: ascii ptr - pointer to base of string
           elements - offset of last element of string
globals accessed: connection
---------------------------------------------------------------

OUTPUT$STRING: PROCEDURE(ASCII_PTR, ELEMENTS) REENTRANT PUBLIC;
DECLARE ASCII_PTR POINTER; ELEMENTS BYTE;
DECLARE (ASCII_STRING BASED ASCII_PTR) (1) BYTE;
DECLARE K BYTE;
/* send string byte by byte */
DO K = 0 TO ELEMENTS;
   IF CONNECTION = 1
```plaintext
/* use iop output if iopb connected to terminal */
THEN CALL IOP_OUTPUT(ASCII_STRING(K));
/* use polled output if 86/12 connected to terminal */
ELSE CALL POLLED_OUTPUT(ASCII_STRING(K));
END;
END OUTPUT$STRING:

/*
  lf_cr: inserts a carriage return and one or two line feeds to the terminal regardless of its connection.
  parameters: one or two - one if if equal to zero, two if if equal to one.
  globals accessed: connection
*/

LF_CR: PROCEDURE(ONE_OR_TWO) REENTRANT PUBLIC;
DECLARE ONE_OR_TWO BYTE;
IF CONNECTION = 1
THEN DO:
  /* use iop output if iopb connected */
  CALL IOP_OUTPUT(CR);
  CALL IOP_OUTPUT(LF);
  IF ONE_OR_TWO = 1
  THEN CALL IOP_OUTPUT(LF);
  END;
ELSE DO:
  /* use polled output if 86/12 connected */
  CALL POLLED_OUTPUT(CR);
  CALL POLLED_OUTPUT(LF);
  IF ONE_OR_TWO = 1
  THEN CALL POLLED_OUTPUT(LF);
  END;
END LF_CR;

/*
  echo byte: inputs a byte from the terminal and echos it, regardless of connection, input is returned
*/
```
ECHO_BYTE: PROCEDURE BYTE PUBLIC;
DECLARE TEMPO BYTE;
IF CONNECTION = 1 THEN DO;
   /* input from iopb if it's connected */
   TEMPO = IOP_INPUT;
   /* echo it */
   CALL IOP_OUTPUT(TEMPO);
END; ELSE DO;
   /* input from 86/12 if its connected */
   TEMPO = POLLED_INPUT;
   /* echo it */
   CALL POLLED_OUTPUT(TEMPO);
END; /* return input to caller */
RETURN TEMPO;
END ECHO_BYTE;

/*-------------------------------------------------------------
  ascii to hex: returns word value of a string of ascii         
  decimal digits, length is variable                          
  parameters: buffer ptr - pointer to base of string          
  digits - number of digits in string                          
------------------------------------------------------------------*/

ASCII_TO_HEX: PROCEDURE(BUFFER_PTR, DIGITS) WORD PUBLIC;
DECLARE BUFFER_PTR POINTER;
DECLARE DIGITS BYTE;
DECLARE (ASCII_BUFFER BASED BUFFER_PTR) (1) BYTE;
DECLARE L BYTE, TEMP2 WORD;
/* get value of most significant digit */
TEMP2 = ASCII_BUFFER(0) - 30H;
DO L = 1 TO DIGITS - 1;
/* recursive relation: obtained by factorizing
value = (10**n)Dn + (10**(n-1))Dn-1 + 10D1 + D0
thus,
Vn-1 = 10 * Vn + Dn-1  initial Vn = Dn
and V0 is value of number */

121 3
122 3
123 2
124 2

TEMP2 = 10*TEMP2 + (ASCII_BUFFER(L) - 30H);
END:
/* return value to caller */
RETURN TEMP2;
END ASCII_TO_HEX;

/* ---------------------------------------------------------------------- */

HEX_TO_ASCII: PROCEDURE (HEX, STRING_PTR, TENTHS_OR_HUNDRETHS) REENTRANT PUBLIC:
125 1
126 2
DECLARE HEX INTEGER;
127 2
DECLARE STRING_PTR POINTER;
128 2
DECLARE (STRING BASED STRING_PTR) (1) BYTE;
129 2
DECLARE TENTHS_OR_HUNDRETHS BYTE;
130 2
DECLARE (J, REMAINDER) INTEGER;
/* get ascii for sign of hex */
131 2
IF (TENTHS_OR_HUNDRETHS = 1) AND (HEX < 0)
THEN STRING(0) = '-';
ELSE STRING(0) = ' '; /* get magnitude */
133 2
HEX = IABS(HEX);
/* get ascii characters for digits */
134 2
DO J = 5 TO 1 BY -1;
135 2
REMAINDER = HEX MOD 10 + 3OH;
STRING(J) = LOW(UNSIGN(REMAINDER));
HEX = HEX/10;
END;

IF TENTHS_OR_HUNDRETHS = 1
THEN DO:
   /* tenths digit */
   STRING(6) = STRING(5); /* move last magnitude digit down */
   STRING(5) = '.'; /* insert point ahead of it */
   STRING(7) = ' ' ; /* trailing space */
   END;
ELSE DO:
   /* hundreths digit */
   STRING(6) = STRING(5); /* move last magnitude digit down */
   STRING(5) = STRING(4); /* move next to last down */
   STRING(4) = '.'; /* insert point ahead of these */
   STRING(7) = ' ' ; /* trailing space */
   END;

J = 1;
/* replace leading zeros with spaces, except in ones position */
DO WHILE (STRING(J) = '30H') AND (J < 5);
   STRING(J) = ' ';   
   J = J + 1;
   END;
END HEX_TO_ASCII;

/* input string: returns word value of up to five ascii decimal digits typed by user, digits are echoed */
/* globals accessed: connection */

INPUT STRING: PROCEDURE WORD PUBLIC;
DECLARE RESPONSE WORD;
DECLARE (I, TEMP1) BYTE;
   /* buffer for input */
DECLARE NUMBER_BUFFER (5) BYTE;
DO;
   TEMP1 = 0;
   I = 0;
/* input until cr or five digits have been received */
DO WHILE (TEMP1 < CR) AND (I <= 4);
  IF CONNECTION = 1 THEN /* use iop input output */
    DO;
      /* get character and echo */
      TEMP1+NUMBER_BUFFER(I) = IOP_INPUT;
      CALL IOP_OUTPUT(NUMBER_BUFFER(I));
    END;
  ELSE /* use polled input and output */
    DO;
      /* get input byte and echo */
      TEMP1+NUMBER_BUFFER(I) = POLLED_INPUT;
      CALL POLLED_OUTPUT(NUMBER_BUFFER(I));
    END;
    /* increment input counter */
    I = I + 1;
  END;
/* insert a cr if */
CALL LF_CR(0);
/* convert input digits to hex, pass number of digits input by user, subtract off cr sent by user */
RESPONSE = ASCII_TO_HEX(NUMBER_BUFFER(I-1));
/* return value of input to caller */
RETURN RESPONSE;
END INPUT$STRING;

END CPU_UTILITIES_MODULE;
<table>
<thead>
<tr>
<th>ADDR</th>
<th>SIZE</th>
<th>NAME, ATTRIBUTES, AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000H</td>
<td>1</td>
<td>ASCII_BUFFER, BYTE BASED(BUFFER_PTR) ARRAY(1)</td>
</tr>
<tr>
<td>0008H</td>
<td>4</td>
<td>ASCII_PTR, POINTER PARAMETER AUTOMATIC</td>
</tr>
<tr>
<td>0009H</td>
<td>1</td>
<td>ASCII_STRING, BYTE BASED(ASCII_PTR) ARRAY(1)</td>
</tr>
<tr>
<td>01E4H</td>
<td>88</td>
<td>ASCII_TO_HEX, PROCEDURE WORD PUBLIC STACK=0008H</td>
</tr>
<tr>
<td>0008H</td>
<td>4</td>
<td>BUFFER_PTR, POINTER PARAMETER AUTOMATIC</td>
</tr>
<tr>
<td>0228H</td>
<td>1</td>
<td>CHAR, BYTE PARAMETER AUTOMATIC</td>
</tr>
<tr>
<td>0006H</td>
<td>1</td>
<td>CHAR, BYTE PARAMETER AUTOMATIC</td>
</tr>
<tr>
<td>0000H</td>
<td>1</td>
<td>CHAR_READY, LITERALLY</td>
</tr>
<tr>
<td>0000H</td>
<td>1</td>
<td>CONNECTION, BYTE EXTERNAL(1)</td>
</tr>
<tr>
<td>0009H</td>
<td>1</td>
<td>COUNTERS_OFF, LITERALLY</td>
</tr>
<tr>
<td>0009H</td>
<td>1</td>
<td>COUNTERS_ON, LITERALLY</td>
</tr>
<tr>
<td>0009H</td>
<td>1</td>
<td>COUNTER_0, LITERALLY</td>
</tr>
<tr>
<td>0009H</td>
<td>1</td>
<td>COUNTER_0_MODE, LITERALLY</td>
</tr>
<tr>
<td>0009H</td>
<td>1</td>
<td>COUNTER_1, LITERALLY</td>
</tr>
<tr>
<td>0009H</td>
<td>1</td>
<td>COUNTER_1_MODE, LITERALLY</td>
</tr>
<tr>
<td>0000H</td>
<td>2</td>
<td>CPU_TIME, WORD EXTERNAL(0)</td>
</tr>
<tr>
<td>003EH</td>
<td>62</td>
<td>CPU_TIME_OUT_LOADER, PROCEDURE PUBLIC STACK=0004H</td>
</tr>
<tr>
<td>0000H</td>
<td>1</td>
<td>CPU_UTILITIES_MODULE, PROCEDURE STACK=0000H</td>
</tr>
<tr>
<td>0006H</td>
<td>1</td>
<td>DIGITS, BYTE PARAMETER AUTOMATIC</td>
</tr>
<tr>
<td>0000H</td>
<td>1</td>
<td>DISABLE_OKXMIT, LITERALLY</td>
</tr>
<tr>
<td>0004H</td>
<td>1</td>
<td>DUMMY_READ, BYTE</td>
</tr>
<tr>
<td>01A9H</td>
<td>58</td>
<td>ECHO_BYTE, PROCEDURE BYTE PUBLIC STACK=000AH</td>
</tr>
<tr>
<td>0006H</td>
<td>1</td>
<td>ELEMENTS, BYTE PARAMETER AUTOMATIC</td>
</tr>
<tr>
<td>0000H</td>
<td>1</td>
<td>ENABLE_OKXMIT, LITERALLY</td>
</tr>
<tr>
<td>0000H</td>
<td>1</td>
<td>FREQ_DIVISOR, LITERALLY</td>
</tr>
<tr>
<td>0000H</td>
<td>1</td>
<td>GATES_OFF, LITERALLY</td>
</tr>
<tr>
<td>0000H</td>
<td>1</td>
<td>GATES_ON, LITERALLY</td>
</tr>
<tr>
<td>00CH</td>
<td>2</td>
<td>HEX, INTEGER PARAMETER AUTOMATIC</td>
</tr>
<tr>
<td>023CH</td>
<td>253</td>
<td>HEX_TO_ASCII, PROCEDURE PUBLIC REENTRANT STACK=0010H</td>
</tr>
<tr>
<td>0007H</td>
<td>1</td>
<td>I, BYTE</td>
</tr>
</tbody>
</table>

**Notes:**
- **ASCII_BUFFER**: Byte based on BUFFER_PTR array (1).
- **ASCII_PTR**: Pointer parameter automatic.
- **ASCII_STRING**: Byte based on ASCII_PTR array (1).
- **ASCII_TO_HEX**: Procedure word public stack=0008H.
- **BUFFER_PTR**: Pointer parameter automatic.
- **CHAR**: Byte parameter.
- **CHAR_READY**: Literally.
- **CONNECTION**: Byte external (1).
- **COUNTERS_OFF**: Literally.
- **COUNTERS_ON**: Literally.
- **COUNTER_0**: Literally.
- **COUNTER_0_MODE**: Literally.
- **COUNTER_1**: Literally.
- **COUNTER_1_MODE**: Literally.
- **CPU_TIME**: Word external (0).
- **CPU_TIME_OUT_LOADER**: Procedure public stack=0004H.
- **CPU_UTILITIES_MODULE**: Procedure stack=0000H.
- **DIGITS**: Byte parameter.
- **DISABLE_OKXMIT**: Literally.
- **DUMMY_READ**: Byte.
- **ECHO_BYTE**: Procedure byte public stack=000AH.
- **ELEMENTS**: Byte parameter.
- **ENABLE_OKXMIT**: Literally.
- **FREQ_DIVISOR**: Literally.
- **GATES_OFF**: Literally.
- **GATES_ON**: Literally.
- **HEX**: Integer parameter.
- **HEX_TO_ASCII**: Procedure public reentrant stack=0010H.
- **I**: Byte.
<table>
<thead>
<tr>
<th>Address</th>
<th>Symbol</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0389H</td>
<td>INPUTSTRING</td>
<td>PROCEDURE</td>
<td></td>
</tr>
<tr>
<td>0000H</td>
<td>IOP_INPUT</td>
<td>PROCEDURE</td>
<td></td>
</tr>
<tr>
<td>0000H</td>
<td>IOP_OUTPUT</td>
<td>PROCEDURE</td>
<td></td>
</tr>
<tr>
<td>0008H</td>
<td>IR_LEVEL</td>
<td>BYTE PARAMETER</td>
<td>20</td>
</tr>
<tr>
<td>FFFEH</td>
<td>J</td>
<td>INTEGER AUTOMATIC</td>
<td>135, 137, 152, 153</td>
</tr>
<tr>
<td>FFFFH</td>
<td>K</td>
<td>BYTE AUTOMATIC</td>
<td>79, 81, 82</td>
</tr>
<tr>
<td>0006H</td>
<td>L</td>
<td>BYTE AUTOMATIC</td>
<td>120, 121</td>
</tr>
<tr>
<td>0154H</td>
<td>LF_CR</td>
<td>PROCEDURE</td>
<td></td>
</tr>
<tr>
<td>0009H</td>
<td>NUMBER_BUFFER</td>
<td>BYTE ARRAY(5)</td>
<td>168, 169, 172, 173, 178</td>
</tr>
<tr>
<td>0006H</td>
<td>ONE_OR_TWO</td>
<td>BYTE PARAMETER</td>
<td>86, 91, 97</td>
</tr>
<tr>
<td>0006H</td>
<td>OP_CODE</td>
<td>BYTE PARAMETER</td>
<td>44, 47, 49</td>
</tr>
<tr>
<td>000AH</td>
<td>OP_CODE</td>
<td>BYTE PARAMETER</td>
<td>20, 22, 24</td>
</tr>
<tr>
<td>0105H</td>
<td>OUTPUTSTRING</td>
<td>PROCEDURE</td>
<td></td>
</tr>
<tr>
<td>0015H</td>
<td>PIC_MANAGER</td>
<td>PROCEDURE</td>
<td></td>
</tr>
<tr>
<td>0000H</td>
<td>PIC_SETUP</td>
<td>PROCEDURE</td>
<td></td>
</tr>
<tr>
<td>006H</td>
<td>PIC_CONTROL</td>
<td>PROCEDURE</td>
<td></td>
</tr>
<tr>
<td>00AH</td>
<td>POLLED_INPUT</td>
<td>PROCEDURE</td>
<td></td>
</tr>
<tr>
<td>00AH</td>
<td>POLLED_OUTPUT</td>
<td>PROCEDURE</td>
<td></td>
</tr>
<tr>
<td>00AH</td>
<td>PORT_B</td>
<td>LITERALLY</td>
<td>31, 33, 35, 37, 39, 41</td>
</tr>
<tr>
<td>00AH</td>
<td>PPI_CONTROL</td>
<td>LITERALLY</td>
<td>48, 50, 52</td>
</tr>
</tbody>
</table>
### PL/M-86 Compiler

**MCS-86 I/O Demonstrator**

**CPU Utilities Module**

**3/6/83**

<table>
<thead>
<tr>
<th>Address</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>007CH</td>
<td>PPI_MANAGER</td>
</tr>
<tr>
<td>46</td>
<td></td>
<td>PROCEDURE PUBLIC STACK=0004H</td>
</tr>
<tr>
<td>130</td>
<td>FFFCH</td>
<td>REMAINDER</td>
</tr>
<tr>
<td>159</td>
<td>0002H</td>
<td>RESPONSE</td>
</tr>
<tr>
<td>45</td>
<td></td>
<td>SETUP</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>SPECIFIC_EOI</td>
</tr>
<tr>
<td>128</td>
<td>0000H</td>
<td>STRING</td>
</tr>
<tr>
<td>125</td>
<td>0008H</td>
<td>STRING_PTR</td>
</tr>
<tr>
<td>102</td>
<td>0005H</td>
<td>TEMPO</td>
</tr>
<tr>
<td>160</td>
<td>0008H</td>
<td>TEMPL</td>
</tr>
<tr>
<td>118</td>
<td>0000H</td>
<td>TEMP2</td>
</tr>
<tr>
<td>125</td>
<td>0006H</td>
<td>TENTHS_OR_HUNDRETHS</td>
</tr>
<tr>
<td>56</td>
<td></td>
<td>TIME</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>UNTIL</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>USART_CONTROL</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>USART_DATA</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>USART_STATUS</td>
</tr>
</tbody>
</table>

**Module Information:**

**Code Area Size**: 03E1H 993D

**Constant Area Size**: 0000H 00

**Variable Area Size**: 0000H 140

**Maximum Stack Size**: 0012H 18D

510 Lines Read

0 Program Error(s)

End of PL/M-86 Compilation
IOP UTILITIES MODULE: Contains procedures for initializing 8089, and invoking channel programs which prepare serial communication on iopb, byte i/o, and output procedure for iop process, and channel program halt.

/** CHANNEL ATTENTION PORT NUMBERS ***************************/

2 \ Declare CHANNEL _1 LITERALLY '0801H'; \* port \# for channel 1 attention */
CHANNEL_2 LITERALLY '0800H'; \* port \# for channel 2 attention */

/** 8089 DATA STRUCTURES *******************************************/
DECLARE SYSTEM_CONFIG_POINTER STRUCTURE
SYSTEM_BUS BYTE,
RESERVED BYTE,
SYSTEM_CONFIG_BLOCK_POINTER POINTER) AT (07FF6H);

DECLARE SYSTEM_CONFIG_BLOCK STRUCTURE
SYSTEM_OPERATION_COMMAND BYTE,
RESERVED BYTE,
CHANNEL_BLOCK_POINTER POINTER) AT (07FF0H);

DECLARE CHANNEL_CONTROL_BLOCK (2) STRUCTURE
CHANNEL_COMMAND_WORD BYTE,
BUSY BYTE,
PARAMETER_BLOCK_POINTER POINTER,
RESERVED WORD) AT (07FE0H);

DECLARE PARAMETER_BLOCK (2) STRUCTURE
TASK_BLOCK_POINTER WORD,
TRANSFER_BYTE BYTE) AT (07000H);

/*********************************************************************************/

DECLARE SCP LITERALLY 'SYSTEM_CONFIG_POINTER';
/*
  iop init: fills 8089 data structures and links the blocks together, then issues a channel attention to channel 1, and waits for the 8089 to clear the channel 1 busy flag. The first channel attention after reset is interpreted as an initialization command by the 8089. It assumes that the system configuration pointer is at OFFFE0H, but since only the lower 13 address bits are driven by the iopb, the address appearing on the multibus will be OFFF6H; the remaining blocks are formed into a linked list, which the 8089 traverses, obtaining configuration information and the location of the channel control blocks. Once this is done, the 8089 clears the channel busy flag.
*/

8 1 IOP_INIT: PROCEDURE PUBLIC:
  / * 8 bit system bus, fill pointer to SCB using iopb
      address space location */
  SCP.SYSTEM$BUS = 0000#0000B;
  SCP.SCB#PTR = 01FF0H;
  / * 8 bit i/o bus, don't care rq/gt mode, fill pointer to channel control blocks using iopb space location */
  SCB.SOC = 0000#0000B;
PL/M-86 COMPILER  MCS-86 I/O DEMONSTRATOR
IOP UTILITIES MODULE

12  2  
SCB.CB$PTR = 01FE0H;
/* set channel 1 busy flag to a known state */
/* then send channel attention to channel 1 */
13  2  
CCB(0).BUSY = OFFH;
14  2  
OUTPUT(CHANNEL_1) = 00H;
/* wait until busy flag clears; initialization complete */
15  2  
DO WHILE CCB(0).BUSY;
16  3  
END;
17  2  
END IOP_INIT;

/*---------------------------------------------------------------
| iop USART setup; invokes channel program resident at 3000H in 8089 private i/o address space, this program initializes iopb USART and PIT
|---------------------------------------------------------------*/

18  1  IOU_USART_SETUP: PROCEDURE PUBLIC;
/* execute channel program in i/o space */
19  2  
CCB(0).CCW = 1$0011$001B;
/* set busy flag to a known state */
20  2  
CCB(0).BUSY = OFFH;
/* link channel 1 control block to parameter block using iopb space location */
21  2  
CCB(0).PB$PTR = 1000H;
/* task block is at 3000H in private i/o space */
22  2  
P0B(0).TB$PTR = 3000H;
/* channel attention */
23  2  
OUTPUT(CHANNEL_1) = 00H;
/* wait till done */
24  2  
DO WHILE CCB(0).BUSY;
25  3  
END;
26  2  
END IOU_USART_SETUP;

/*---------------------------------------------------------------
| iop output; analogous to polled output procedure in cpu utilities module, invokes channel program which transmits byte parameter to terminal.
|---------------------------------------------------------------*/
IOP_OUTPUT: PROCEDURE (CHAR) REENTRANT PUBLIC;
DECLARE CHAR BYTE;
/* execute channel program from i/o space */
CCB(O).CCW = 100011001B;
/* set busy flag to a known state */
CCB(O).BUSY = OFFH;
/* link in parameter block using iopb space location */
CCB(O).PB$PTR = 1000H;
/* program starts at 3100H in i/o space */
P8(O).TB$PTR = 3100H;
/* pass character in parameter block */
P8(O).TRANSFER_BYTE = CHAR;
/* channel attention */
OUTPUT(CHANNEL_1) = OOH;
/* wait till done */
DO WHILE CCB(O).BUSY;
END;
END IOP_OUTPUT;

IOP_INPUT: PROCEDURE BYTE REENTRANT PUBLIC;
/* execute program from i/o space */
CCB(O).CCW = 100011001B;
/* set busy flag to a known state */
CCB(O).BUSY = OFFH;
/* link to parameter block */
CCB(O).PB$PTR = 1000H;
/* program starts at 3200H in i/o space */
P8(O).TB$PTR = 3200H;
/* channel attention */
PL/M-86 COMPILER

IOP UTILITIES MODULE

43 2
44 2
45 3
46 2
47 2

OUTPUT(CHANN EEAL_1) = 00H;
/* wait till done */
DO WHILE CCB(0),BUSY;
END;
/* get input from parameter block and return to caller */
RETURN PB(0),TRANSFER_BYTE;
END IOP_INPUT;

/*------------------------------------------------------*/
/* iop_process_out_start: invokes channel program at 3300H which */
/* performs i/o for the iop process in the */
/* main demonstrator module. */
/*------------------------------------------------------*/

48 1
49 2
50 2
51 2
52 2
53 2
54 2
55 2

IOP_PROCESS_OUT_START: PROCEDURE PUBLIC;
/* execute program from i/o space */
CCB(0),CCW = 1000*111001B;
/* busy flag to known state */
CCB(0),BUSY = OFFH;
/* link in parameter block */
CCB(0),PB$PTR = 1000H;
/* program starts at 3300H in i/o space */
PB(0),TRANSFER$PTR = 3300H;
/* zero program go flag: causing it to wait until iop */
/* process has its first output */
PB(0),TRANSFER$BYTE = 0;
/* start program */
OUTPUT(CHANN EEAL_1) = 00H;
END IOP_PROCESS_OUT_START;

/*------------------------------------------------------*/
/* iop_halt: halts the iop process output program when it has */
/* completed the current output, this is necessary */
/* because after time out, the 8089 is needed to transmit */
/* the throughput, thus, this program must be stopped; */
/* and stopped gracefully to keep the usart from hanging */
/* up in the transmission mode. */
/*------------------------------------------------------*/
IOP_HALTS: PROCEDURE PUBLIC;
    /* halt channel program command */
    CCB(0),CCW = 1$00$11$111B;
    /* set busy flag to a known state */
    CCB(0),BUSY = OFFH;
    /* wait until iop process output program has completed
       transmission of the current output */
    DO WHILE PB(0),TRANSFER_BYTE;
    END;
    /* then issue the halt command */
    OUTPUT(CHANNEL_1) = 00H;
    /* wait until it is carried out */
    DO WHILE CCB(0),BUSY;
    END;
    END IOP_HALTS;

END IOP_UTILITIES_MODULE;
### CROSS-REFERENCE LISTING

<table>
<thead>
<tr>
<th>DEFN</th>
<th>ADDR</th>
<th>SIZE</th>
<th>NAME, ATTRIBUTES, AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0001H</td>
<td>1</td>
<td>BUSY</td>
</tr>
<tr>
<td>7</td>
<td>CBPTR</td>
<td>2</td>
<td>LITERALLY</td>
</tr>
<tr>
<td>7</td>
<td>CCE</td>
<td>2</td>
<td>LITERALLY</td>
</tr>
<tr>
<td>7</td>
<td>CCW</td>
<td>2</td>
<td>LITERALLY</td>
</tr>
<tr>
<td>2</td>
<td>CHANNEL_1</td>
<td>2</td>
<td>LITERALLY</td>
</tr>
<tr>
<td>2</td>
<td>CHANNEL_2</td>
<td>2</td>
<td>LITERALLY</td>
</tr>
<tr>
<td>4</td>
<td>0002H</td>
<td>4</td>
<td>CHANNEL_BLOCK_POINTER</td>
</tr>
<tr>
<td>5</td>
<td>0000H</td>
<td>1</td>
<td>CHANNEL_COMMAND_WORD</td>
</tr>
<tr>
<td>5</td>
<td>7FEOH</td>
<td>16</td>
<td>CHANNEL_PARAMETER_BLOCK</td>
</tr>
<tr>
<td>27</td>
<td>0006H</td>
<td>1</td>
<td>CHAR</td>
</tr>
<tr>
<td>56</td>
<td>01EH</td>
<td>68</td>
<td>PROCEDURE STACK=0000H</td>
</tr>
<tr>
<td>8</td>
<td>0010H</td>
<td>110</td>
<td>PROCEDURE STACK=0008H</td>
</tr>
<tr>
<td>38</td>
<td>0138H</td>
<td>96</td>
<td>PROCEDURE BYTE PUBLIC REENTRANT STACK=0000-6H</td>
</tr>
<tr>
<td>27</td>
<td>0005H</td>
<td>99</td>
<td>PROCEDURE STACK=0006H</td>
</tr>
<tr>
<td>48</td>
<td>0198H</td>
<td>78</td>
<td>PROCEDURE STACK=0006H</td>
</tr>
<tr>
<td>18</td>
<td>007EH</td>
<td>87</td>
<td>PROCEDURE STACK=0006H</td>
</tr>
<tr>
<td>1</td>
<td>0010H</td>
<td>1</td>
<td>PROCEDURE STACK=0000H</td>
</tr>
<tr>
<td>6</td>
<td>7000H</td>
<td>6</td>
<td>STRUCTURE ARRAY(2) AT ABSOLUTE</td>
</tr>
<tr>
<td>5</td>
<td>0002H</td>
<td>4</td>
<td>STRUCTURE ARRAY(2) AT ABSOLUTE</td>
</tr>
</tbody>
</table>

**Note:** The table contains a list of symbols, addresses, and their corresponding attributes and references within the context of a compiler listing.
PL/M-86 COMPILATION
MCS-86 I/O DEMONSTRATOR
IOP UTILITIES MODULE

3/6/83

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>PB</td>
<td>LITERALLY 46</td>
</tr>
<tr>
<td>7</td>
<td>PBPTR</td>
<td>LITERALLY 21</td>
</tr>
<tr>
<td>5 0000H</td>
<td>2 RESERVED</td>
<td>WORD MEMBER</td>
</tr>
<tr>
<td>4 0001H</td>
<td>1 RESERVED</td>
<td>BYTE MEMBER</td>
</tr>
<tr>
<td>3 0001H</td>
<td>1 RESERVED</td>
<td>BYTE MEMBER</td>
</tr>
<tr>
<td>7</td>
<td>SCB</td>
<td>LITERALLY 10</td>
</tr>
<tr>
<td>7</td>
<td>SCBPTR</td>
<td>LITERALLY 10</td>
</tr>
<tr>
<td>7</td>
<td>SCP</td>
<td>LITERALLY 11</td>
</tr>
<tr>
<td>3 0000H</td>
<td>1 SYSTEMBUS</td>
<td>BYTE MEMBER</td>
</tr>
<tr>
<td>4 7FF0H</td>
<td>6 SYSTEM_CONFIG_BLOCK</td>
<td>STRUCTURE</td>
</tr>
<tr>
<td>3 0002H</td>
<td>4 SYSTEM_CONFIG_BLOCK_POINTER</td>
<td>POINTER MEMBER</td>
</tr>
<tr>
<td>3 7FF6H</td>
<td>6 SYSTEM_CONFIG_POINTER</td>
<td>STRUCTURE</td>
</tr>
<tr>
<td>4 0000H</td>
<td>1 SYSTEM_OPERATION_COMMAND</td>
<td>BYTE MEMBER</td>
</tr>
<tr>
<td>6 0000H</td>
<td>2 TASK_BLOCK_POINTER</td>
<td>WORD MEMBER</td>
</tr>
<tr>
<td>7</td>
<td>TBPTR</td>
<td>LITERALLY 22</td>
</tr>
<tr>
<td>6 0002H</td>
<td>1 TRANSFER_BYTE</td>
<td>BYTE MEMBER</td>
</tr>
</tbody>
</table>

MODULE INFORMATION:

CODE AREA SIZE = 022AH  554D
CONSTANT AREA SIZE = 0000H  0D
VARIABLE AREA SIZE = 0000H  0D
MAXIMUM STACK SIZE = 0000H  0D
277 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
## LOC OBJECT CODE  TIMING  INC MAC LINE SOURCE

0000

9003

3 PIT_CONTROL EQU 09003H ; AD/  
-ADDRESS OF 8253 CONTROL  
; RE/  

9000

5 COUNTER_0 EQU 09000H  ; AD/  
-ADDRESS OF 8253 COUNTER 0  
; AD/  

A001

6 USART_CONTROL EQU OA001H ; AD/  
-ADDRESS OF 8251A CONTROL  
; RE/  

A000

8 USART_DATA EQU OA000H ; AD/  
-ADDRESS OF 8251A DATA  
; RE/  

003E

10 COUNTER_MODE EQU 00111110B ; HD/  
-DEFINE FOR COUNTER 0 IS  
; BI/  

0020

13 LOW_DIVISOR EQU 20H  ; LO/  
-4 BYTE OF BAUD RATE  
; DI/  

0000

15 HIGH_DIVISOR EQU 00H  ; HI/  
-GH BYTE OF BAUD RATE  
; DI/  

00CE

17 USART_MODE EQU 11001110B ; US/  
-ART SETUP FOR 8 BITS  
; OF/  

18 - DATA, 2 STOP BITS AND  

19
LOC  OBJECT CODE  TIMING  INC MAC  LINE SOURCE

0013  D130 0000  17  25  - PARITY 0 16X

20  USART_COMMAND  EQU 00010011B  ; W1/
   -TH 2400 BAUD

0027  D130 0390  34  50

21  ENABLE_XMIT  EQU 00100111B  ; DR/
   -IVES USART RTS/;

22  -M3A CTS/ LOW

0003  D130 084C  60  80

23  DISABLE_XMIT  EQU 00000011B  ; DR/
   -IVES USART RTS/;

24  -M3A CTS/ HIGH

0000  D130 0390  34  50

25  PROCESS_BUFFER  EQU 00  ; IO/
   -P PROCESS OUTBUFF

26  - LOCATION OH IN IDP8  ; AT/

27  - ACE 6000H IN 36/12  ; SP/

28  - ACE

0008  D130 084C  60  80

29  BUFFER_LENGTH  EQU 8  ; B /
   -BYTE PROCESS BUFFER

F000  D130 084C  60  80

30  CHAR_COUNTER  EQU 0F00H  ; I/;
   -D RAM TEMP LOCATION

31  -UNTS NUMBER OF OUTPUTS  ; CD/

32  - CURRENT LINE

000D  D130 084C  60  80

33  CR  EQU 00H  ; AB/
   -CII CARRAIGE RETURN

000A  D130 084C  60  80

34  LF  EQU 0AH  ; AB/
   -CII LINE FEED

35  ; USART SETUP PREPARES THE USART AND PI/  
   -T FOR SERIAL COMMUNICATION

36  ; PROGRAM LOCATED AT 3000H IN PRIVATE I/  
   -/O SPACE

37  39  USART_SETUP:  MOV1 CC,0

38  40  MOV1 GA:PIT_CONTROL

39  41  MOVBI (GA) COUNTER_MQ/
   -DE  ; SEND MODE TO PIT

40  42  -  ; CONTROL REGISTER
MCS-86 I/O DEMONSTRATOR CHANNEL PROGRAMS 3/6/83

LOC  OBJECT CODE  TIMING  INC MAC  LINE SOURCE
0000  0000          75  98  43  NOP  /
0000D  1130 0090   93 123  44  ; 8253 WRITE  /
0011  084C 20      116 153  46  MOVIC [GA]!COUNTER_0
0014  0000          127 171  47  MOVIC [GA]!LOW_DIVISO/
0016  084C 00      152 201  48  -R  ; LOAD  /
0019  1130 01A0   171 256  49  MOVIC [GA]!HIGH_DIVIS/
001D  084C CE      194 256  50  OR  ; BAUD RATE DIVISOR
0020  0000          205 274  51  MOVIC [GA]!USART_CONTROL
0022  084C 13      231 304  52  MOVIC [GA]!USART_MODE/
0025  2048          253 329  54  SEND MODE & COMMAND
0020  0000          205 274  55  ; 8251 WRITE  /
0022  084C 13      231 304  56  ; RECOVERY DELAY  /
0025  2048          253 329  57  ; CLEAR BUSY FLAG

; PARAMETER BLOCK FORMAT FOR TERMINAL I/
; -PUT AND TERMINAL OUTPUT
; CHANNEL PROGRAMS, REFERENCES TO STRUCT/
; -TURE MEMBERS ARE OFFSETS
; FROM THE STRUCTURE BASE, WHICH IS THE/
; LOCATION OF THE PARAMETER
; BLOCK THAT IS LOADED INTO PP WHEN THE/
; CHANNEL ATTENTION IS
; RECEIVED, TASK_PTR IS CHANNEL PROGRAM/
; LOCATION IN I/O SPACE
; WHILE XFER BYTE IS USED FOR PASSING I/
; /O BYTE BETWEEN IOFB
; AND 86/12.

65  TERMINAL_IO_PARAMETER_BLOCK STRUC
66  -  DS 2  :  TASK_PTR
67  -  XFER_BYTE!  /
MCS-86 I/O DEMONSTRATOR CHANNEL PROGRAMS 3/6/83

0003  ORG 100H

0100 D130 0000 270 354
0104 1130 0040 287 379
0108 0293 02 007D 326 429

010D 3130 01A0 349 454
0111 0000 359 472
0113 4BB9 FD 364 504
0116 2048 402 529

8089 MACRO ASSEMBLER

LOC OBJECT CODE TIMING INC MAC LINE SOURCE

---

0003

---

ORG 100H

---

68 TERMINAL_IO_PARAMETER_BLOCK ENDS
69
70 ; TERMINAL OUTPUT PROCEDURE IS THE CHANNEL/PRIVATE I/O SPACE,
71 ; TERMINAL, USED IN CONJUNCTION WITH AN 8086
72 ; EMULATES THE POLLE/D OUTPUT PROCEDURE
73 ; DURING DEMONSTRATION SETUP WHEN THE I/O
74 ; TERMINAL PROGRAM STARTS AT 3100H IN
75
76
77
78 TERMINAL_OUTPUT: MOV C,0
79 MOV G,A,USART_DATA
80 MOV G,A,EPFJ,XFER_B/
81 ; GET DATA BYTE
82 ; FROM PB: SEND
83 ; SEND TO USART
84 WAIT_TILL_DONE: INP /
85 ; 8251A WRITE
86 ; RECOVERY DELAY
87 ; IS HIGH, I.E.
88 ; XMIT COMPLETE
89 HLT /
90 ; CLEAR BUSY FLAG
91 ; TERMINAL INPUT CHANNEL PROGRAM RETURN/ S A BYTE OF INPUT FROM
92 ; THE TERMINAL TO THE 86/12 IN THE PARALLELMETER BLOCK, USED IN
93 ; CONJUNCTION WITH A POLLE PROCEDURE TO E/
0200 D130 0000 419 554
0209 1130 0100 936 579
0208 084C 27 462 609
0208 3130 00A0 480 634
020F 0881 499 656
0211 0000 519 674
0213 2888 FD 539 706
0216 0091 02CF 02 592 756
021B 084C 03 605 786
021E 2048 623 811

MULATE THE POLLED INPUT
94 ; PROCEDURE WHEN THE IOPB IS CONNECTED /
95 ; TO THE TERMINAL. STARTS
96
97
98
99 TERMINAL_INPUT: MOVIC C, 0
100 MOVIGA, USART_CONTROL
101 MOVB [GA] + ENABLE_XMI1
102 - T ; ENABLE ADM
103 - ; TRANSMISSION
104 - ; BY LOWERING RTS
105 WAIT_TIL_RDY: MOV X, WAIT_TIL /
106 - ; READ RECOVERY DELAY
107 - ; WAIT TIL RXRDY IS
108 - ; HIGH, I.E. RECEPTION
109 - ; COMPLETE
110 MOVBG C, [GB] ; PUT DATA IN
111 - ; BYTE WINDOW
112 - IT ; DISABLE ADM
113 - ; TRANSMISSION
114 - ; BY RAISING RTS
115 ; PROCESS OUTPUT PARAMETER BLOCK FORMAT /
116 ; SAME AS ABOVE EXCEPT
117 ; THAT XFER BYTE IS CALLED GO FLAG; WHO /
118 ; TO SIGNAL THE CHANNEL PROGRAM WHEN A NEW /
119 ; DPUT IS READY, AND
120 ; TO SIGNAL THE IOP PROCESS WHEN THE DUT /
-TPUT HAS BEEN COMPLETED.

119
120 PROCESS_OUTPUT_PARAMETER_BLOCK STRUC
121
122 - DS 2 : TASK_PTR
123 GO_FLAG:
124 - DS 1
125 PROCESS_OUTPUT_PARAMETER_BLOCK ENDS
126
127 : PROCESS OUTPUT CHANNEL PROGRAM USED IF:
128 : OR TERMINAL OUTPUT BY THE
129 : IOP PROCESS, OR DEMONSTRATION PROCEDURE
130 : - RE, STARTS AT 3300H, ASSUMES
131 : 8 BYTE ASCII OUTPUT STRING A LOCATION
132 : - OH IN IOPD SPACE, LOCATION
133 : 6000H IN 86/12 SPACE, AND ASCII DATA
134 : -VALID WHEN GO FLAG GOES HIGH
135 : WHEN OUTPUT COMPLETE, PROGRAM CLEARS
136 : -GO FLAG, PROGRAM DOES NOT
137 : HALT.
138
139
140 PROCESS_OUTPUT: MOVI C0,5080H
141
142 MOVI CC,CHAR_COUNTER
143 - ZERO THE CHARACTER
144 MOVE:EGCJ, 0
145 ; COUNTOFF:
146 IDLE: JZB [PPJ.GO_FLAG,IDLE
147 - IDLE TILL GO SIGNAL
148
149 - FROM IOP PROCESS
150 LPDI 6A;PROCESS_BUFFER
151
152 MOVI 6B;USART_DATA
153 - LOAD BUFFER ADDRESS
154 - LOAD DESTINATION ADDR
155 MOVI 6C;BUFFER_LENGTH
156 - FILE BUFFER LENGTH
157 XFER_LOOP:1: MOVI IX,100
158 - APPROX. 1.8MS DELAY
159
160 AM3A_DELAY_1: DEC IX
<table>
<thead>
<tr>
<th>LOC</th>
<th>MACRO ASSEMBLER</th>
<th>OBJECT CODE</th>
<th>TIMING</th>
<th>INC MAC</th>
<th>LINE SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>007A</td>
<td>8089</td>
<td>A040 FD</td>
<td>056</td>
<td>1120</td>
<td>145</td>
</tr>
<tr>
<td>032A</td>
<td>3800</td>
<td>6000</td>
<td>067</td>
<td>1138</td>
<td>146</td>
</tr>
<tr>
<td>032C</td>
<td>8000</td>
<td>878</td>
<td>078</td>
<td>1156</td>
<td>147</td>
</tr>
<tr>
<td>032E</td>
<td>6840 F2</td>
<td>897</td>
<td>087</td>
<td>1179</td>
<td>148</td>
</tr>
<tr>
<td>0331</td>
<td>06EA</td>
<td>924</td>
<td>092</td>
<td>1209</td>
<td>149</td>
</tr>
<tr>
<td>0333</td>
<td>F130 0AFF</td>
<td>942</td>
<td>094</td>
<td>1239</td>
<td>150</td>
</tr>
<tr>
<td>0337</td>
<td>08B6 D3</td>
<td>967</td>
<td>096</td>
<td>1266</td>
<td>151</td>
</tr>
<tr>
<td>03A3</td>
<td>084E 00</td>
<td>993</td>
<td>099</td>
<td>1296</td>
<td>152</td>
</tr>
<tr>
<td>033D</td>
<td>1106 00000000</td>
<td>1028</td>
<td>102</td>
<td>1342</td>
<td>153</td>
</tr>
<tr>
<td>0343</td>
<td>3120 00A0</td>
<td>1046</td>
<td>104</td>
<td>1367</td>
<td>154</td>
</tr>
<tr>
<td>0347</td>
<td>7320 0200</td>
<td>1064</td>
<td>106</td>
<td>1392</td>
<td>155</td>
</tr>
<tr>
<td>0348</td>
<td>084C 00</td>
<td>1087</td>
<td>108</td>
<td>1422</td>
<td>156</td>
</tr>
<tr>
<td>034E</td>
<td>0A4C 01 0A</td>
<td>1113</td>
<td>111</td>
<td>1456</td>
<td>157</td>
</tr>
<tr>
<td>0352</td>
<td>B130 6400</td>
<td>1130</td>
<td>113</td>
<td>1481</td>
<td>158</td>
</tr>
<tr>
<td>0356</td>
<td>A03C</td>
<td>1140</td>
<td>114</td>
<td>1498</td>
<td>159</td>
</tr>
<tr>
<td>0358</td>
<td>8A40 FB</td>
<td>1159</td>
<td>115</td>
<td>1521</td>
<td>160</td>
</tr>
<tr>
<td>035E</td>
<td>6000</td>
<td>1174</td>
<td>117</td>
<td>1539</td>
<td>161</td>
</tr>
<tr>
<td>035F</td>
<td>8000</td>
<td>1189</td>
<td>118</td>
<td>1557</td>
<td>162</td>
</tr>
<tr>
<td>0362</td>
<td>8B20 A6</td>
<td>1222</td>
<td>122</td>
<td>1601</td>
<td>163</td>
</tr>
</tbody>
</table>
LDC OBJECT CODE TIMING INC MAC LINE SOURCE

- : WAIT FOR NEW OUTPUT

0365

167

168 CHANNEL_PROGRAMS ENDS

169 END
**Symbol Table Listing**

<table>
<thead>
<tr>
<th>DEFN VALUE TYPE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>144 0325 SYM</td>
<td>ADM3A_DELAY_1</td>
</tr>
<tr>
<td>160 0356 SYM</td>
<td>ADM3A_DELAY_2</td>
</tr>
<tr>
<td>29 0008 SYM</td>
<td>BUFFER_LENGTH</td>
</tr>
<tr>
<td>1 0000 SYM</td>
<td>CHANNEL_PROGRAMS</td>
</tr>
<tr>
<td>30 F000 SYM</td>
<td>CHAR_COUNTER</td>
</tr>
<tr>
<td>127 0308 SYM</td>
<td>CHAR_DONE</td>
</tr>
<tr>
<td>5 9000 SYM</td>
<td>COUNTER_0</td>
</tr>
<tr>
<td>10 003E SYM</td>
<td>COUNTER_MODE</td>
</tr>
<tr>
<td>23 0003 SYM</td>
<td>DISABLE_XMIT</td>
</tr>
<tr>
<td>21 0027 SYM</td>
<td>ENABLE_XMIT</td>
</tr>
<tr>
<td>122 0002 SYM</td>
<td>GO_FLAG</td>
</tr>
<tr>
<td>15 0000 SYM</td>
<td>HIGH_DIVISOR</td>
</tr>
<tr>
<td>138 030F SYM</td>
<td>IDLE</td>
</tr>
<tr>
<td>34 000A SYM</td>
<td>LF</td>
</tr>
<tr>
<td>13 0020 SYM</td>
<td>LOW_DIVISOR</td>
</tr>
<tr>
<td>3 9003 SYM</td>
<td>PIT_CONTROL</td>
</tr>
<tr>
<td>25 0000 SYM</td>
<td>PROCESS_BUFFER</td>
</tr>
<tr>
<td>124 0300 SYM</td>
<td>PROCESS_OUTPUT</td>
</tr>
<tr>
<td>120 0000 STR</td>
<td>PROCESS_OUTPUT_PARAMETER_BLOCK</td>
</tr>
<tr>
<td>99 0200 SYM</td>
<td>TERMINAL_INPUT</td>
</tr>
<tr>
<td>65 0000 STR</td>
<td>TERMINAL_ID_PARAMETER_BLOCK</td>
</tr>
<tr>
<td>78 0100 SYM</td>
<td>TERMINAL_OUTPUT</td>
</tr>
<tr>
<td>20 0013 SYM</td>
<td>USART_COMMAND</td>
</tr>
<tr>
<td>6 4001 SYM</td>
<td>USART_CONTROL</td>
</tr>
<tr>
<td>5 4000 SYM</td>
<td>USART_DATA</td>
</tr>
<tr>
<td>17 00CE SYM</td>
<td>USART_MODE</td>
</tr>
<tr>
<td>39 0000 SYM</td>
<td>USART_SETUP</td>
</tr>
<tr>
<td>84 0111 SYM</td>
<td>WAIT_TILL_DONE</td>
</tr>
<tr>
<td>105 0211 SYM</td>
<td>WAIT_TILL_RDY</td>
</tr>
<tr>
<td>67 0002 SYM</td>
<td>XFER_BYTE</td>
</tr>
<tr>
<td>143 0321 SYM</td>
<td>XFER_LOOP_1</td>
</tr>
<tr>
<td>159 0352 SYM</td>
<td>XFER_LOOP_2</td>
</tr>
</tbody>
</table>

*Assembly complete, no errors found*
REFERENCES


11. **Designing 8086, 8088, 8089 Multiprocessor Systems**  

12. **The Line Driver and Line Receiver Data Book**.  


