Design and implementation of a 16-bit digital servocontroller P.I.D. algorithm
Silberfich, P.A.

Published: 01/01/1984

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. Jan. 2019
Author: P.A. Silberfich


Department: Mechanical Eng., T.H.E.
Acknowledgements

The author wishes to express his acknowledgements to Ir. Heuvelman, C.J. for the guidance throughout the whole work, and to Mr. Theuws, who contributed to the happy ending of the project, and for creating with the rest of the group, a very nice atmosphere of work during my stay in the FYSISCHE BEWERKINGEN.

Thanks also to Mia Lutters, who was very kind in accepting the job of typing this report.

Eindhoven, August 1984

PABLO A. SILBERFICH
<table>
<thead>
<tr>
<th>Contents list</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ABSTRACT-------------------------------------------------------------------</td>
<td>1</td>
</tr>
<tr>
<td>2. SUMMARY--------------------------------------------------------------------</td>
<td>2/3</td>
</tr>
<tr>
<td>3. INTRODUCTION----------------------------------------------------------------</td>
<td>4</td>
</tr>
<tr>
<td>4. THEORETICAL FORMULATIONS----------------------------------------------------</td>
<td>5/31</td>
</tr>
<tr>
<td>4.1 Control theory</td>
<td></td>
</tr>
<tr>
<td>4.1.1: Digital control systems</td>
<td>5/9</td>
</tr>
<tr>
<td>4.1.2: Deterministic control systems</td>
<td>10/11</td>
</tr>
<tr>
<td>4.2 Intel systems</td>
<td></td>
</tr>
<tr>
<td>4.2.1: Intellec series III</td>
<td>12/13</td>
</tr>
<tr>
<td>Microcomputer development system</td>
<td></td>
</tr>
<tr>
<td>4.2.2: ISBC 86/12</td>
<td>14/16</td>
</tr>
<tr>
<td>4.2.3: Working with I/O</td>
<td>17</td>
</tr>
<tr>
<td>4.2.4: The 8087 NDP and the ISBC 337</td>
<td>18/20</td>
</tr>
<tr>
<td>4.2.5: The microprocessor 8086</td>
<td>21/30</td>
</tr>
<tr>
<td>Intel in detail</td>
<td></td>
</tr>
<tr>
<td>4.2.6: Some other support</td>
<td>31</td>
</tr>
<tr>
<td>5. SOFTWARE/HARDWARE STRUCTURES</td>
<td>32/47</td>
</tr>
<tr>
<td>5.1 The P.I.D. controller</td>
<td>32/34</td>
</tr>
<tr>
<td>5.2 Developing flow-charts</td>
<td></td>
</tr>
<tr>
<td>5.2.1: System 1</td>
<td>35</td>
</tr>
<tr>
<td>5.2.2: System 2</td>
<td>36</td>
</tr>
<tr>
<td>5.2.3: Proportional Action</td>
<td>37</td>
</tr>
<tr>
<td>5.2.4: Integral Action (Trapezoidal method)</td>
<td>38</td>
</tr>
<tr>
<td>5.2.5: Integral Action (Rectangular method)</td>
<td>39</td>
</tr>
<tr>
<td>5.2.6: Differential Action</td>
<td>40</td>
</tr>
</tbody>
</table>
5.3 Integration Algorithms------------------------------- 41/42

5.4 PIDOUT Program
   5.4.1: Explanations------------------------------- 43/44
   5.4.2: Testing Signal----------------------------- 45/46
   5.4.3: Output Results------------------------------ 47

6. ONE STEP FURTHER----------------------------------- 48

7. CONCLUSIONS---------------------------------------- 49

8. BIBLIOGRAPHY--------------------------------------- 50

9. APPENDICES----------------------------------------- A1/07
   9.1 APP A: Instr. set 8086-------------------------A1, A9
   9.2 APP B: Addressing modes 8086/applications------B1, B1
   9.3 APP C: Parallel I/O port configuration--------C1, C3
   9.4 APP D: Specifications of the ISBC 86/12 S.B.C.---D1, D3
   9.5 APP E: I/O Address Assignments----------------E1, E1
   9.6 APP F: Programming information (8251A, 8253, 8255A, 8259A)---F1, F6
   9.7 APP G: Instr. set 8087--------------------------G1, G1
   9.8 APP H: ISBC 86/12 S.B.C.-----------------------H1, H8
   9.9 APP I: ISBC 337 Multimodule Numeric Data Processor------I1, I8
   9.10 APP J: DAC/OUTPUT to motor-drive-------------J1, J1
   9.11 APP K: Output results-------------------------K1, K5
   9.12 APP L: Monitor programs-----------------------L1, L17
   9.13 APP M: INITIO (integr. 8086)------------------M1, M3
   9.14 APP N: PIDOUT (P.I.D. 8086)-------------------N1, N5
   9.15 APP O: INTOUT (integr. PASCAL '86)------------O1, O7
1. ABSTRACT

With flexible automation systems, electromechanical servomotors with controllers are used. Since the load of these motors is variable, the controllers should be of the adaptive type. An adaptive controller practically can only be digital. Further requirements are speed and accuracy.

Controllers with the INTEL 8085 have already been made, but sometimes, they don't match the requirements.

Our project includes the design and implementation of a P.I.D. controller with an INTEL 8086/87 system, 16 bits.

After working in assembly and PASCAL languages, a comparison is given including all the algorithms developed.
Summary

Signal processing with either microprocessors or digital computers isn't limited to some few basic functions like conventional analog control. They are programmable and can perform complex calculations. Therefore many new methods can be developed for digital process control, which for the low levels can be realized as programmed algorithms and for the higher levels as programmed problem-solving methods.

We worked with these algorithms and the design of digital control systems with reference to process computers and microcomputers. They were ready for obtaining process models, for estimation of states and parameters, and for using in digital monitoring and optimization of systems.

Among the variety of controllers, we discussed the derivation and design, based on conventional analog controllers, of parameter-optimized control algorithms, with for instance P - PI - PID behaviour, as well as, separate from the continuous signals, general discrete time controllers of low order.

In short saying, we worked these algorithms out, and afterwards we compared the accuracy, delay and timing of all of them performed in:
- 8086 INTEL ASSEMBLY LANGUAGE
- 8087 INTEL ASSEMBLY LANGUAGE EXTENSION
- PASCAL '86 INTEL HIGH LEVEL LANGUAGE.

After working the programs on INTEL MICROCOMPUTER DEVELOPMENT SYSTEM "INTELLEC SERIES III", we run the programs in actual 8086/87 boards, obtaining big advantages in timing and precision with the 8086 over the 8085. It was not possible to run the complete set of 8087' programs, but the partial results showed that for uncomplicated algorithms, with
simple calculations, and when high accuracy is not necessary, there's no need for the 8087 extension.

The PASCAL programs are much easier to write, and our test-algorithms showed a rather good timing when comparing with the rest of our development.
3. INTRODUCTION

In the last 25 years, a new approach to the control, the direct digital control (D.D.C.) was the means to get closer in the theory of the process control in industry.

When the number of control loops increased, the traditional analog controllers began to present disadvantages.

With the appearance of the first digital control computers, or the microprocessors, a lot of the problems presented by the conventional control could be solved.

Nowadays, the process control necessities require the computer systems to achieve not only D.D.C. tasks, but also other functions such as supervisory control, adaptive controls optimization, communications with lower and higher levels of control, etc.

So, the speed of operations became an important matter for the selection of sampling periods, numbers of loops to be controlled, the amount of levels of hierarchies of control, etc.

Specially the speed, became highly critical when speaking about servomechanisms and fast variables like in the case we are dealing with.

Although we put emphasis in the speed, we can't forget the accuracy of our calculations.

These are only part of the reasons to conclude saying that when someone wants to implement a controller which is able to accept a wide range of parameters, as well as the possibility of getting several control actions, the discrete signals and the D.D.C. provide the basic tools for such an implementation.
4. THEORETICAL FORMULATIONS

4.1 Control Theory

4.1.1 Digital control systems

To get deeply into the subject, let's talk about the closed-loop systems, the controller's models and how to obtain our algorithms from them.

A closed-loop system is characterized by the following block diagram (fig. no. 1), which usually is a comparator that sets up the error $e$ (epsilon), sometimes an amplifier to increase its value; a process to be controlled, in our case the motor drive; a measuring device of the controlled variable ($y$) for comparison with the reference input, thereafter, generating an error signal.

![Block Diagram](image)

Fig. 1

Due to the bad performance of this system when we need special requirements in errors, overshoot, phase margin, etc., we prefer to present a better approach that includes a control block (fig. no. 2)
The main role of the controller is to improve the system performance, and, although this is a typical place in the loop, this block can also move to other places utilizing, e.g. feedfoward technique.

The controller's models derive from either modern control theory, classical theory or both.

The industrial practise has shown that one of the more popular controllers is directly or related with the proportional-integral-derivative one, since it, usually suffices most of the requirements of a control system.

It can be shown that the proportional action increases the damping, therefore the stability; the integral action reduces the steady-state error, and the derivative action provides a fast response since it's based on the error rate and not on the error itself.
These and more features prove that the PID controller is one which can offer both good transient and steady-state response. (See fig. no. 3)

Fig. 3

It's a common practice to derive digital controllers by means of simulations, adapting the analog model. These digital simulations are performed by using numerical methods to express derivatives and integrals, that is:

- differences for derivatives
- rectangular, recursive rectangular or trapezoidal sums for the integrals.

When dealing with a digital system we visualize the discretization effect in the way of fig. no. 4. We'll deal with Sampled-data control, and making simplifications to that, and forgetting the comparator for the moment, we can think of a control loop as: (fig. 4)
Actually, the manipulated variable $u$ is calculated by a control algorithm using the control variable $y$ and the reference value $w$ as inputs ($y$ is not in the figure).

For the design of digital control systems, generally speaking, we can consider:

1) Information on the processes and the signals:
   - direct measurable inputs, outputs, state variables
   - process models and signal models
   - state estimates of processes and signals

2) Control system structure:
   - single input/single output control systems
   - interconnected control systems
   - multi-input/multi-output control systems

3) Feedforward and feedback control algorithms (design and adjustment):
   - simple tuning rules for the parameters
   - computer-aided design
- self-optimizing adaptive control algorithms

4) Noise filtering:
   - analog and digital filters

5) Feedforward or feedback control of the actuators.
4.1.2 Deterministic control systems

Trying to make a scheme, we obtain (fig. no. 5)

Fig. 5
In our treatment of "Deterministic Control Systems", we'll design linear controllers, which we can see looking in our case, like:

**Major groups:**
- PARAMETER OPTIMIZED CONTROLLERS

**Subgroups:**
- ZERO ORDER
- FIRST ORDER
- SECOND ORDER
- HIGHER ORDER

**Design method:**
- TUNING RULES
- POLE ASSIGNMENT
- PERFORMANCE CRITERION

**Controller:**
- P - PI - General
- PD - PID
- linear

Concluding, we can say that there is no D.D.C. algorithm, which is better than the P.I.D. controller for the general purpose, single-loop control functions.

Although there's a difference between using analog and digital controllers, we can reduce this gap using fine quantizations and sampling time.
4.2.1 INTELLEC series III microcomputer development systems

The INTELLEC series III microcomputer development systems is more than a keyboard, a video display, disk drives, and a box with two microprocessors. It is a real tool for designing microcomputer software for the iAPX 86,88 processor family or for the 8080/8085 processors.

We used this system to program assembly for the 8086, 8087 and also PASCAL '86. We were also able to debug programs, link them, locate them, convert to hexadecimal code, and run them on the system itself, with emulators, or on the boards.

The Intellec Series III, with in-circuit emulation (I.C.E.) is a development solution to provide support for parallel hardware and software development efforts. The chart in fig. no. 6 summarizes a software development process, starting with an idea for a final product (Developing software on the Series III System).
Fig. 6
The ISBC 86/12 single board computer is a complete computer system on a single printed-circuit assembly. It includes a 16-bit central processing unit (CPU), 32 kBytes of RAM, a serial communications interface, three programmable parallel I/O ports, timers which can be programmed, priority interrupt control, multibus control logic, expansion to be interfaced with other compatible boards (fig. no. 7).

Among its features, we'll speak about those which are of special interest for our project.
The ISBC 86/12 includes 24 programmable parallel I/O lines implemented by means of an INTEL 8255A PPI, (Programmable Peripheral Interface). The system software is used to configure the I/O lines in any combination of unidirectional I/O and bidirectional ports.

The RS 232C compatible serial I/O port is controlled and interfaced by an INTEL 8251A USART (Universal Synchronous/Asynchronous Receiver/Transmitter) chip.

Three independent, fully programmable 16-bit interval/event counters are provided by an INTEL 8253 PIT (Programmable Interval Timer). Each counter is capable of operating in either BCD or binary modes, two of these counters are available to generate accurate timers, as we did use.

We could also make use of the INTEL 8259A PIC (Programmable Interrupt Controller); which was excitated by the output of the timer. This chip can handle up to eight interrupts. By using external PIC's slaved to the on-board 8259(master), the interrupt structure can be expanded to handle and resolve the priority of up to 64 sources.

The PIC, which can be programmed to respond to edge-sensitive or level-sensitive inputs, treats each true input signal condition as an interrupt request. After resolving the interrupt priority, the PIC issues a single interrupt request to the C.P.U. Interrupt priorities are independently programmable by means of software control.

The CPU includes a non-maskable interrupt (NMI) and a maskable interrupt (INTR). The INTR is driven by the 8259A which, on demand, provides an 8-bit identifier of the interrupting source. The CPU multiplies the 8-bit identifier by 4 to derive a pointer to the service routine for the interrupting device.

Tables about the parallel I/O ports can be seen in APP.C.
The specifications of the ISBC 86/12 Single Board Computer can be seen in APP.D. and APP.H.
4.2.3 Working with I/O

The CPU communicates with the on-board programmable chips through a sequence of I/O read and I/O write commands. To do that, we send different words to different addresses. These addresses can be seen in APP.E.

To initialize and send the correct sequence of data to the correct addresses of the:
- 8251A USART
- 8253 PIT
- 8255A PPI
- 8259A PIC

we suggest reading the 3rd chapter of the INTEL publication named "ISBC 86/12 Single Board Computer Hardware Reference Manual". We add a short information in the APP.F.
4.2.4 The 8087 NDP (Numeric Data Processor)

The 8087 NDP extends the 8086/8088 instruction set to provide serious advantages in capability.

It serves as a coprocessor attached to an 8086/8088, effectively adding eight 80-bit floating-point registers to the 8086/8088 register set.

It uses its own instruction queue to monitor the 8086/8088 instruction stream, executing only those instructions intended for it and ignoring the instructions intended for the 8086/88 CPU.

The 8087 NDP instructions include a full set of arithmetic functions as well as a powerful core of exponential, logarithmic, and trigonometric functions (see APP.G).

It uses a common 80-bit internal floating-point number format to handle seven different useful external formats (fig. 8).

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Bits</th>
<th>Significant Digits (Decimal)</th>
<th>Approximate Range (Decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word integer</td>
<td>16</td>
<td>4</td>
<td>-32,768 \leq X \leq 32,767</td>
</tr>
<tr>
<td>Short integer</td>
<td>32</td>
<td>9</td>
<td>-2 \times 10^8 \leq X \leq 2 \times 10^8</td>
</tr>
<tr>
<td>Long integer</td>
<td>64</td>
<td>18</td>
<td>-9 \times 10^{16} \leq X \leq 9 \times 10^{16}</td>
</tr>
<tr>
<td>Packed decimal</td>
<td>80</td>
<td>18</td>
<td>-99...99 \leq X \leq 99...99 (18 digits)</td>
</tr>
<tr>
<td>Short real*</td>
<td>32</td>
<td>6-7</td>
<td>8.43 \times 10^{-37} \leq</td>
</tr>
<tr>
<td>Long real*</td>
<td>64</td>
<td>15-18</td>
<td>4.19 \times 10^{-307} \leq</td>
</tr>
<tr>
<td>Temporary real</td>
<td>80</td>
<td>19</td>
<td>3.4 \times 10^{-4932} \leq</td>
</tr>
</tbody>
</table>

Fig. 8

We can see the internal structure of the chip in the figs. 9 and 10.
The 8087 adds extensive high-speed numeric processing capabilities to the CPU. It uses the standard iAPX 86/186 family instruction set plus over fifty numeric instructions; but programs can
be written in ASM-86 assembly language, or in INTEL high-level languages PL/M-86, Fortran 86 and PASCAL '86.

The NDP is a hardware extension because it will not run by itself. That is, it needs to have an 8086 or 8088 to run the data, address, and control buses which feed its instructions and operands. That can be seen in fig. (11).

Fig. 11

To obtain the interconnection to the ISBC 86/12A, we make use of the ISBC 337 board. (see APP.I)
4.2.5 The microprocessor 8086 - INTEL in detail

The 8086 was introduced in 1978. It's a very popular 16-bit microprocessor because of its particular features and the tremendous amount of support (both hardware and software). As other microprocessors, it's register oriented, which means that it is easier to manipulate data when it's stored in registers, rather than when it's stored in memory.

Making a brief summary, we can say that it has fourteen 16-bit registers in it, eight of which can be considered to be general purpose. These are:

- four general registers AX, BX, CX, DX which can be used by bytes: AL, AH, BL, BH, CL, CH, DL, DH
- four segment registers    CS (code segment register)  
    DS (data segment register)  
    ES (extra segment register)  
    SS (stack segment register)

- SP (stack pointer)
- BP (pointer base register)
- IP (instruction pointer)
- DI (destination index register)
- SI (source index register)
- FLAGS (flag register)

There are many typical functions associated to the registers, and hence, we have special names:

\[
\begin{align*}
AX & : \text{accumulator} \\
BX & : \text{base} \\
CX & : \text{count} \\
DX & : \text{data}
\end{align*}
\]

General Register Group

One rule about these registers is the fact that, if we consider them as 2 bytes, the L-type registers will contain D7-D0 and the H-type registers will contain D8-D15 of a 16-bit word.
Although these are general purpose registers, they are sometimes used by some instructions to store a base address, a count, or a data value. Some instructions assume that there's a 16-bit base address in the BX register.

Other instructions assume that a count has been loaded into either CL(8 bits) or CX(16 bits), and that's what we use to generate the square wave.

Finally, one of the functions of the DX register is to specify the 16-bit \(I/O\) address port that the 8086 is communicating with during the execution of an \(I/O\) instruction.

Going now to the segment register group, we can say that typically, these registers are used to store a 16-bit segment address, which the 8086 uses when it address memory. And speaking about this, we'll explain something about the addressing modes.

First of all, as we mentioned previously, the 8086 can address 1MByte. To do this, a 20-bit address is required. However, more of the registers that we have discussed can store a 20-bit address; they can only be used to store 16-bit values.

To generate 20-bit addresses we introduce then the concepts of segmentation, offset and segment addresses. Therefore, an offset or effective address (EA) is added to a segment address as shown in fig. (12).
That is, the segment address which is contained in one of the four segment registers can be thought of as being shifted to the left four times before it's added to the EA. Then, the result is a '20-bit long word'.

Let's comment something about the pointer and index register group so that to have a complete idea of the addressing modes.

The SP (stack pointer) is used to provide part of a 20-bit address that the 8086 uses when any information is placed on, or taken off of the stack. It's used in conjunction with the stack segment register (SS).

The BP (base pointer), SI (source index) and DI (destination index) are often used in different addressing modes, generally with the data segment register (DS) or extra segment register (ES).
As an example we can say that the 20-bit memory address used to fetch instructions from memory is generated by adding the content of the CS (16 bits shifted 4 bits to the left) to the 16-bit of the IP.

When a stack instructions is executed by the 8086, the SP is added to the SS.

Since the segment registers are all independent of each other, and the base, stack and instruction pointers are all 16 bits wide, separate 64Kbyte blocks of the microcomputer memory can be allocated solely for data, stack, code and extra, as we can see in fig. (13)

---

Segment registers addressing different portions of the 1M-byte address space.

Fig. 13
There can be also, a kind of segment-overlapping. One nice result of segmentation is that programs can be moved to different sections of memory and still be executed, because references to data, instructions and stack are relative to the content of the segment registers.

Then we suggest the use of relative transfer-of-control instructions.

To conclude, we can look at the fig. (14), and add to that the register and immediate addressing, which are two simple addressing modes used when the information is actually contained within the instruction.

<table>
<thead>
<tr>
<th><strong>Different Addressing Modes and the Registers Used</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BASE INDEX</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>INDEX</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>BASE</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>BASE INDEX</strong></td>
</tr>
<tr>
<td>(NO DISPLACEMENT)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>INDIRECT</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>RELATIVE</strong></td>
</tr>
<tr>
<td><strong>DIRECT</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Note: Displacement may be either 8 or 16 bits.

Fig. 14

The calculations of the actual addresses are in the following figs. (15 and 16).
ENCODED IN THE INSTRUCTION

SINGLE INDEX

- BX
- OR
- BP
- OR
- SI
- OR
- DI

DOUBLE INDEX

- BX
- OR
- BP
- OR
- SI
- OR
- DI

ASSUMED UNLESS OVERRIDDEN BY PREFIX

EXPlicit IN THE INSTRUCTION

Fig. 15
Uses of the addressing modes in arrays, and other applications can be seen in the appendix B.

In a general way, the 8086 can directly address 1MByte of memory and 64K of 8-bit input/output ports.
The large address space of the 8086 is complemented by a powerful instruction set (135 basic instructions) that can operate on individual bits, 8-bit bytes, 16-bit words and 32-bit double words.

The organization of the instruction set is as follows: (see also appendix A)

a) **Data transfer instructions**
   - Memory ↔ registers
   - AL ↔ I/O
   - AX ↔ I/O
   a1) general purpose
   a2) input/output
   a3) address object
   a4) flag transfer

b) **Arithmetic instructions**
   - unsigned binary
   - signed binary (integers)
   - unsigned packed decimal
   - unsigned unpacked decimal

c) **Bit manipulation instructions**
   - logicals
   - shifts
   - rotates

d) **String instructions** (bytes and words sequences)
   Applications: move and compare strings, scan for a value;
   everything possible to and from the acc.

e) **Program transfer instructions**
   - unconditional transfers
   - conditional transfers
   - iteration control instructions
   - interrupt - related instructions
f) Processor control instructions

- flag operations
- no operation
- external synchronization

We can see from fig. 17 that all of the instructions are from 1 to 6 bytes long.

<table>
<thead>
<tr>
<th>NUMBER OF BYTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>SINGLE REGISTER</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>REGISTER TO REGISTER</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>IMMEDIATE BYTE OR RELATIVE SHORT TRANSFER</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>IMMEDIATE WORD TO REGISTER OR RELATIVE LONG TRANSFER</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>IMMEDIATE BYTE TO MEMORY (LONG DISP)</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>IMMEDIATE WORD TO MEMORY (LONG DISP)</td>
</tr>
</tbody>
</table>

Fig. 17

Even though memory for the 8086 is organized as 16-bit words, either byte in a word can be addressed. The least significant byte (D7-D0) is stored in the byte memory locations with the lower address, and the
most significant byte (D15-D8) is stored in the next higher byte memory location. The 8086 instructions don't have to be word aligned.

The basic clock speed of an 8086-based microcomputer may be between 4MHz and 8MHz, depending on the 8086 chip used.

Assuming that a 5-MHZ 8086 (typical) is being used, the shortest instructions require just 400 nseg to be executed and the longest instructions may require approximately 40 useg.

In the instructions, we deal with variables, and then, it's wise to speak about their attributes.

They are:
- Segment: it identifies the segment that contains the variable.
- Offset: distance in bytes from the beginning of that segment.
- Type: it identifies the variable's allocation unit (byte = 1, word = 2, doubleword = 4)

Because of these three attributes, it is defined the form of instruction to generate.
4.2.6 Some other support

To be able to complete our project, we made use of some other chips; they were the 8253, 8255A and 8259A from INTEL. These chips allowed us to work with timers, with I/O ports, and with interrupts. A brief description of them can be seen in APP.F, and the explanations of the use we made of them, in the next sections.
5. SOFTWARE/HARDWARE STRUCTURES

5.1 The PID controller

The basic scheme of a regulator using feedback control can be simplified as follows: (fig 18)

Our idea is to develop the different control algorithms as main goal (syst. 2), and then, in one further step, to implement the system 1 that includes comparator, and as last objective the whole system.

The function of the control block is to convert the error signal into a signal able to go to the process, obtaining a wanted signal in the output.

Usually it's needed that y follows r as close as possible. The way of obtaining it depends on the feedback (may be also feedforward) and the control algorithm. Then, we can have overshoot, static and dynamic errors, different timing, delays, etc.

Depending on that, after obtaining \( e = r - y \), we obtain the correction signal \( x = \text{CONTROL TRANSFER} \# e \).
To bring our goals into the reality, we'll first make a summary of some ways of discretizing the differential equations of continuous P.I.D. controllers.

Working with sampled discrete-time signal, we can express the effects in a way of functions of the form:

- Proportional Action: \( x(T) = K \times e(T) \)

- Differential Action: \( x(T) = T_d \times \frac{de}{dt} \bigg|_{t=T} \)

- Integral Action: \( x(T) = \frac{1}{T_i} \int_{t=0}^{T} e(t)dt \)

where:

\( K \) = gain
\( T_i \) = integration time
\( T_d \) = derivative time

We can add the single effects, until we obtain a whole PID action:

\[
x(T) = K \times e(T) + \frac{1}{T_i} \int_{t=0}^{T} e(t)dt + T_d \times \left( \frac{de}{dt} \bigg|_{t=T} \right)
\]

But actually now, when we use the sampling theory to convert it into a discrete system, we can write, after replacing the derivative by a difference of first order and the integral by a sum:

a) Using a trapezoidal approach for the integration:

\[
x(nT) = K \times e(nT) + \frac{1}{T_i} \sum_{k=1}^{n} T_s \times \frac{e(kT) + e(kT-T)}{2} + \frac{T_d}{T_s} \times e(nT) - e(nT-T)
\]

\( T_s \) = sampling time
b) If we apply rectangular integration, we have:

\[ x(nT) = K \cdot e(nT) + \frac{T_s}{T_i} \sum_{k=1}^{n} e(kT - T) + T_d \cdot \frac{e(nT) - e(nT - T)}{T_s} \]

c) If we work these expressions out and calculate the difference between two successive samples, we arrive to recursive control algorithms, more suitable for programming on computers.
5.2 Developing Flow-charts

5.2.1 Flow chart of system 1

- **START SYST. 1**
- **GET TETHAIN**
- **GET LAST TETHAOUT**
- **EPSILON = TETHAIN - TETHAOUT**
- **SYSTEM 2**
- **TETHAOUT = OUTPUT**
- **END SYSTEM 1**
5.2.2 Flow chart of system 2

START SYST. 2

GET INSTBYTE

TEST INSTBYTE

EITHER
CALL PROP. ACTION,
INTEGRAL ACTION,
DIFFER. ACTION OR
A COMBINATION OF THEM

SEND THE RESULTING SIGNAL
TO THE OUTPUT PORT

END SYSTEM 2
5.2.3 Flow chart of proportional action

START
PROPORTIONAL

GET INPUT

GET CONSTANT K

OUTPUT = K * INPUT

RETURN
5.2.4 Flow chart of integral action

(Trapezoidal method)

START INTEGRATION

GET e(nT)

GET e(nT-T)

SAVE e(nT) as e(nT-T)

ADD
SOM(nT) = e(nT) + e(nT-T)

GET SOM(nT-T)

ADD
SOM(nT) = SOM(nT-T) + SOM(nT)

SAVE SOM(nT) as SOM(nT-T)

GET 2T_i

DIVIDE SOM(nT) / 2T_i

GET T_s

MULTIPLY SOM(nT) * T_s

RETURN

INIT = 2T_i
SAMT = T_s
5.2.5 Flow chart of integral action

(Rectangular method) (adopted)

START
INTEGRATION

GET e(nT)

GET \[ e(nT-T) \]

ADD e(nT) to old addition (STIN)

SAVE new SUM as OLD

GET \[ T_i \]

DIVIDE \( \frac{\text{STIN}(nT)}{T_i} \)

GET \[ \frac{T_s}{T_i} \]

MULTIPLY \( \frac{\text{STIN}(nT)}{T_i} \cdot T_s \)

RETURN
5.2.6 Flow chart of differential action

START
DIFFER

GET e(nT) (NEW)

GET e(nT-T) (OLD)

OLD + e(nT)

\( \Delta \leftarrow \text{SUBSTR. OLD from NEW} \)

GET DIFFT

MULTIP. DIFFT \( \times \) \( \Delta \)

GET SAMT

DIVIDE \( \Delta \leftarrow \text{DIFFT} \)
SAMT

RETURN
5.3 Integration Algorithms

1) Trapezoidal integration

\[ x(nT) = \frac{1}{T_i} \sum_{k=1}^{n} T_s \cdot \frac{e(kT) + e(kT-T)}{2} \]

BEGIN: (new value in AX)

```
MOVBX, STIN
MOV STIN, AX
% ADDIT
MOVBX, SOM
% ADDIT
MOVSOM, AX
MOVBX, INTT
% DIVIDE
MOVBX, SAMT
% MULTI
```

(90 USEG)

2) Rectangular integration

\[ x(nT) = \frac{T_s}{T_i} \sum_{k=1}^{n} e(kT-T) \]

BEGIN: (new value in AX)

```
MOVBX, STIN
% ADDIT
MOVSOM, AX
MOVBX, INTT
% DIVIDE
MOVBX, SAMT
% MULTI
```

(80 USEG)

(SEE APP.M)
2) Rectangular integration
(another way)

\[ x(nT) = T_S \sum_{k=1}^{n} e(kT-T) \]

BEGIN: (new value in AX)

```
MOV BX, INTT
% DIVIDE
MOV BX, SAMT
% MULTI
MOV BX, SOM
% ADDIT
MOV SOM, AX   (80 USEG)
```
5.4 **PIDOUT program**

5.4.1 **Explanations and use**

The PIDOUT program works following the flow diagram of system 1. It generates the testing square wave, and depending on the INSTBYTE entered, it can execute different routines. To do it, enter:

- 0: output equals to 0
- 1: \(D\) effect
- 2: \(I\) effect
- 3: \(I+D\) effect
- 4: \(P\) effect
- 5: \(P+D\) effect
- 6: \(P+I\) effect
- 7: \(P+I+D\) effect

The input signal changes to its complement each 10 times the counter has reached 0.

Each time the counter reaches 0, an interrupt is originated and one service interrupt routine cycle is performed. To do so, first of all, we should initialize the different chips, not allowing interrupts in meanwhile.

Then, we send control bytes to initialize the PIC

**ICW1**

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
\end{array}
\]

\(0000 \text{ vector address) --- ALWAYs 1}\)

**ICW2**

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\(0000 \text{ vector address) --- ALWAYs 1}\)
The last thing to do, after initializing the timers is to initialize the PPI with all the PORTS = OUTPUT.

To do that, we send to the 8255: 080F

- Port C(lower) = OUTPUT
- Port B = OUTPUT
- Selection mode 0(basic) I/O
- Port C(UPPER) = OUTPUT
- Port A = OUTPUT
- Selection mode 0 (basic) I/O
- Mode set flag active
5.4.2 Testing signal - Software square wave

After we've done, we must initialize the P.I.T. It's important to mention that in every case we tested our design with a testing input signal. First we thought of a function generator, but finally we decided to implement it as a square wave generated by software/hardware means. To get it, we made use of the INTEL 8253 chip. (see also APP.C)

We send a control byte to initialize the timer 0, and then a byte to give the initial count to our desired value, as follows:

MOV AL, 36H
OUT OD6H, AL, INITIALIZE TIMERO

MOV AL, OF6H
OUT O0DH, AL ; LSB TIME
(492D CYCLES ≈ 400 useg)

MOV AL, O1H
OUT OODH, AL ; MSB TIME

MOV CX, 10D ; COUNT TO 10 + 4 mSEG.

MOV AX, 5 ; AMPLITUDE OF SQUARE WAVE

LOOP INTEG ; MEANWHILE CX≠0, GO ON WITH INTEG; WHEN CX=10 THEN, NEG AX, MOV CX=10 AND GO ON
NEG    AX    ; INVERT OUTPUT FOR SQUARE WAVE (-5)
MOV    CX, 10D ; LOAD COUNT AGAIN WITH 10D

INTEG:  .

That means that the timer will count till 400 useg (492 D to 0 D). Then it will give an interrupt that will start the INTPTR routine, integrating or using the selected algorithm. After 10 cycles of the same input value, it will change to the same module but with inverted sign, and so on.
5.4.3 Output results

Finally, when the program is running, is always sending output signal through port A to the DAC and hardware support (the last output value corresponding to the last interrupt). (APP.J).

Synchronizing with the input signal, we can measure the time the procedures need to have the outputs updated.

We can see all the options and their results in the pictures in APP.K.
6. **ONE STEP FURTHER**

The PID controller 8086 is now working as well as the INTEGRATION ACTION written in PASCAL '86.

The following step is to close the loop in a feedback system, adding also a nicer way of entering data and the possibility of obtaining results on the screen.

The help doing that, we've made new programs to enter data by keyboard means, to send data to defined registers, to convert codes, to visualize register contents in decimal notation on the screen.

The listing of the programs can be found in APPs. L, M, N, O.
7. **CONCLUSIONS**

**Comparative table of integration cycle (in usec)**

<table>
<thead>
<tr>
<th>Method</th>
<th>Time</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rectangular integration with 8085</td>
<td>1500</td>
<td>(assembly)</td>
</tr>
<tr>
<td>Rectangular integration with 8086</td>
<td>80</td>
<td>(assembly)</td>
</tr>
<tr>
<td>Trapezoidal integration with 8086</td>
<td>90</td>
<td>(assembly)</td>
</tr>
<tr>
<td>Rectangular integration with 8086</td>
<td>180</td>
<td>(PASCAL)</td>
</tr>
</tbody>
</table>

It's important to mention that we have tried, under our possibilities some testing programs with the 8087 extension. The results showed that there was no actual difference compared with the 8086 programs. Actually, we can say that the longest instructions are those to divide and to multiplicate in our algorithms, and working inside the 8087 with 80 bits (TEMPREAL), it takes more or less the same time than with the 8086 with, of course, less bits.

That doesn't mean that the 8087 is not useful; what it means, is that we can leave the 8087 for the following cases:

- special calculations
- special data types
- larger ranges
- special rounding treatments
- better precision
- transcendental instructions needed
- other instructions needed.
8. **BIBLIOGRAPHY**


Programmer's Reference 1983/INTEL iAPX 86,88 Family utilities

user's guide /1982/ INTEL

ASM 86 Macro Assembler

Operating Instructions

ASM 86 Language Reference

Manual.

An introduction to ASM 86 /1981/ INTEL

A guide to Intellec Series III Microcomputer

Development Systems /1981/ INTEL

iSBC 86/05 Single Board Computer Hardware

Reference Manual /1981/ INTEL

iSBC 86/12 Single Board Computer Hardware

Reference Manual /1981/ INTEL

Intel Microcomputer Development Systems:

- ISIS-II-User's Guide
- ISIS-II-Credit-Editor/User's Guide
- ISIS-II-Universal PROM programmer

Intellec Series III-Microcomputer Development

System-Console Operating Instructions

Intellec Series III-Microcomputer Development


PASCAL-86 User's guide/1982/Intel

INTEL 8087-NUMERIC DATA COPROCESSOR

INTEL Component Data Catalog 1980

INTEL FAIR-Applications Handbook

INTEL-SYSTEMS DATA CATALOG/1981

8086/8088-16 bit Microprocessor Primer, by Ch.L. MORGAN and M. WAITE

De Digitale PID-regelaar, WPB'survey by H.M.M.G. Cordewener (T.H.E.)

PID Controller in D.D.C., using ISCOS 100 system by G.A. PALACIOS (PII report)

DISCRETE DATA CONTROL SYSTEMS, Prentice Hall, by B.C. Kuo

DISCRETE SYSTEMS, by ISERMAN

DIGITAL CONTROL USING MICROPROCESSORS, by KATZ.
8086/8088 Instruction Set

**8086**

**REGISTER MODEL**

<table>
<thead>
<tr>
<th>AX</th>
<th>AH</th>
<th>AL</th>
<th>ACCUMULATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>BX</td>
<td>BH</td>
<td>BL</td>
<td>BASE</td>
</tr>
<tr>
<td>CX</td>
<td>CH</td>
<td>CL</td>
<td>COUNT</td>
</tr>
<tr>
<td>DX</td>
<td>DM</td>
<td>DL</td>
<td>DATA</td>
</tr>
</tbody>
</table>

| SP \ BP | STACK POINTER \ BASE POINTER |
| SI \ DI | SOURCE INDEX \ DESTINATION INDEX |
| IP | INSTRUCTION POINTER |
| FLAGS | STATUS FLAGS |

| CS | CODE SEGMENT |
| DS | DATA SEGMENT |
| SS | STACK SEGMENT |
| ES | EXTRA SEGMENT |

Instructions which reference the flag register file as a 16-bit object, use the symbol **FLAGS** to represent the file:

\[
\begin{array}{cccccccc}
15 & X & X & X & XOF & DF & SF & ZF & XAF & XPF & XCF \\
\end{array}
\]

**X = Don't Care**

**AF**: AUXILIARY CARRY — BCD
**CF**: CARRY FLAG
**PF**: PARITY FLAG
**SF**: SIGN FLAG
**ZF**: ZERO FLAG

- **8080 FLAGS**

**DF**: DIRECTION FLAG (STRINGS)
**IF**: INTERRUPT ENABLE FLAG
**OF**: OVERFLOW FLAG (CF & SF)

- **8086 FLAGS**

**TF**: TRAP — SINGLE STEP FLAG
### OPERAND SUMMARY

**"reg" field Bit Assignments:**

<table>
<thead>
<tr>
<th>16-Bit (w = 1)</th>
<th>8-Bit (w = 0)</th>
<th>Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 AX</td>
<td>000 AL</td>
<td>00 ES</td>
</tr>
<tr>
<td>001 CX</td>
<td>001 CL</td>
<td>01 CS</td>
</tr>
<tr>
<td>010 DX</td>
<td>010 DL</td>
<td>10 SS</td>
</tr>
<tr>
<td>011 BX</td>
<td>011 BL</td>
<td>11 DS</td>
</tr>
<tr>
<td>100 SP</td>
<td>100 AH</td>
<td></td>
</tr>
<tr>
<td>101 BP</td>
<td>101 CH</td>
<td></td>
</tr>
<tr>
<td>110 SI</td>
<td>110 DH</td>
<td></td>
</tr>
<tr>
<td>111 DI</td>
<td>111 BH</td>
<td></td>
</tr>
</tbody>
</table>

### SECOND INSTRUCTION BYTE SUMMARY

<table>
<thead>
<tr>
<th>mod</th>
<th>Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>DISP = 0&quot;. disp-low and disp-high are absent</td>
</tr>
<tr>
<td>01</td>
<td>DISP = disp-low sign-extended to 16-bits, disp-high is absent</td>
</tr>
<tr>
<td>10</td>
<td>DISP = disp-high: disp-low</td>
</tr>
<tr>
<td>11</td>
<td>r/m is treated as a &quot;reg&quot; field</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r/m</th>
<th>Operand Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>(BX) + (SI) + DISP</td>
</tr>
<tr>
<td>001</td>
<td>(BX) + (DI) + DISP</td>
</tr>
<tr>
<td>010</td>
<td>(BP) + (SI) + DISP</td>
</tr>
<tr>
<td>011</td>
<td>(BP) + (DI) + DISP</td>
</tr>
<tr>
<td>100</td>
<td>(SI) + DISP</td>
</tr>
<tr>
<td>101</td>
<td>(DI) + DISP</td>
</tr>
<tr>
<td>110</td>
<td>(BP) + DISP*</td>
</tr>
<tr>
<td>111</td>
<td>(BX) + DISP</td>
</tr>
</tbody>
</table>

DISP follows 2nd byte of instruction before data if required...

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

**Operand Address (EA) Timing (clocks):**

Add 4 clocks for word operands at ODD ADDRESSES.

- Immed Offset = 6
- Base + (BX, BP, SI, DI) = 5
- Base + DISP = 9
- Base + Index (BP + DI, BX + SI) = 7
- Base + Index (BP + SI, BX + DI) = 8
- Base + Index (BP + DI, BX + SI) + DISP = 11
- Base + Index (BP + SI, BX + DI) + DISP = 12
### DATA TRANSFER

**MOV = Move**
- **Register/memory to/from register**
  - `1 0 0 0 1 0 dw mod reg r/m`
  - **Timing (clocks):**
    - Register to register: 2
    - Memory to register: 8+EA
    - Register to memory: 9+EA

<table>
<thead>
<tr>
<th>Immediate to register/memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>1 1 0 0 0 1 1 w mod 0 0 0 r/m</code> data</td>
</tr>
<tr>
<td><strong>Timing:</strong> 10+EA clocks</td>
</tr>
</tbody>
</table>

**Immediate to register**
- `1 0 1 1 w reg` data | data if w=1 |
- **Timing:** 4 clocks

**Memory to accumulator**
- `1 0 0 0 0 0 w addr-low addr-high` |
- **Timing:** 10 clocks

**Accumulator to memory**
- `1 0 1 0 0 0 1 w addr-low addr-high` |
- **Timing:** 10 clocks

**Register/memory to segment register**
- `1 0 0 0 1 1 0 mod 0 reg r/m`
- **Timing (clocks):**
  - Register to register: 2
  - Memory to register: 8+EA

**Segment register to register/memory**
- `1 0 0 0 1 1 0 mod 0 reg r/m`
- **Timing (clocks):**
  - Register to register: 2
  - Register to memory: 9+EA

**PUSH = Push**
- **Register/memory**
  - `1 1 1 1 1 1 1 mod 1 1 0 r/m`
  - **Timing (clocks):**
    - Register: 10
    - Memory: 16+EA

**XCHG = Exchange**
- **Register/memory to/from register**
  - `1 0 0 0 0 1 1 w mod reg r/m`
  - **Timing (clocks):**
    - Register with register: 4
    - Memory with register: 17+EA

**Immediate to register**
- `1 0 1 0 w reg` |
- **Timing:** 4 clocks

**Memory to accumulator**
- `1 0 0 0 0 0 w addr-low addr-high`
- **Timing:** 10 clocks

**Register to memory**
- `1 0 0 0 1 1 1 mod 0 0 0 r/m`
- **Timing (clocks):**
  - Register to register: 2
  - Memory to register: 8+EA

**XLAT = Translate byte to AL**
- `1 1 0 1 0 0 1 mod 0 0 0 r/m`
- **Timing:** 10 clocks

**LEA = Load EA to register**
- `1 0 0 0 1 0 1 mod reg r/m`
- **Timing:** 2+EA clocks

**LDS = Load pointer to DS**
- `1 1 0 0 1 0 1 mod reg r/m`
- **Timing:** 16+EA clocks

**LES = Load pointer to ES**
- `1 1 0 0 1 0 0 mod reg r/m`
- **Timing:** 16+EA clocks

**LAHF = Load AH with flags**
- `1 0 0 1 1 1 1`
- **Timing:** 10 clocks

**SAHF = Store AH into flags**
- `1 0 0 1 1 1 1 0`
- **Timing:** 8 clocks

**PUSHF = Push flags**
- `1 0 0 1 1 1 0 0`
- **Timing:** 10 clocks

**POPF = Pop flags**
- `1 0 0 1 1 1 0 1`
- **Timing:** 8 clocks
ARITHMETIC

ADD = Add
Reg./memory with register to either

\[ 0\ 0\ 0\ 0\ 0\ 0\ \text{w} \mod \ 0\ 0\ \text{reg} \ \text{r/m} \]

Timing (clocks): register to register 3
memory to register 9+EA
register to memory 16+EA

Immediate to register/memory

\[ 1\ 0\ 0\ 0\ 0\ \text{s}\ \text{w} \mod \ 0\ 0\ 0\ \text{r/m} \]

data data if s = 01

Timing (clocks): immediate to register 4
immediate to memory 17+EA

Immediate from register/memory

\[ 1\ 0\ 0\ 0\ 0\ \text{s}\ \text{w} \mod \ 1\ 0\ 1\ \text{r/m} \]

data data if s = 01

Timing (clocks): immediate from register 4
immediate from memory 17+EA

Immediate from accumulator

\[ 0\ 0\ 1\ 0\ 1\ 1\ 0\ \text{w} \]

data data if w = 1

Timing: 4 clocks

SBB = Subtract with borrow
Reg./memory and register to either

\[ 0\ 0\ 0\ 1\ 1\ 0\ \text{d}\ \text{w} \mod \ 0\ \text{reg} \ \text{r/m} \]

Timing (clocks): register from register 3
memory from register 9+EA
register from memory 16+EA

Immediate from register/memory

\[ 1\ 0\ 0\ 0\ 0\ \text{s}\ \text{w} \mod \ 0\ 1\ 1\ \text{r/m} \]

data data if s = 01

Timing (clocks): immediate from register 4
immediate from memory 17+EA

Immediate from accumulator

\[ 0\ 0\ 0\ 1\ 1\ 1\ 0\ \text{w} \]

data data if w = 1

Timing: 4 clocks

DEC = Decrement
Register/memory

\[ 1\ 1\ 1\ 1\ 1\ 1\ \text{w} \mod \ 0\ 0\ 1\ \text{r/m} \]

Timing (clocks): register 2
memory 15+EA

Register

\[ 0\ 1\ 0\ 0\ 1\ \text{reg} \]

Timing: 2 clocks

NEG = Change sign

\[ 1\ 1\ 1\ 0\ 1\ 1\ \text{w} \mod \ 0\ 1\ 1\ \text{r/m} \]

Timing (clocks): register 3
memory 16+EA

CMP = Compare
Register/memory and register

\[ 0\ 0\ 1\ 1\ 1\ 0\ \text{d}\ \text{w} \mod \ 0\ \text{reg} \ \text{r/m} \]

Timing (clocks): register with register 3
memory with register 9+EA
register with memory 9+EA
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Format</th>
<th>Timing (clocks)</th>
<th>Immediate with register/memory</th>
<th>Immediate with register/memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Add with carry</td>
<td>0001000 d w mod reg r/m</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IMUL</td>
<td>Integer multiply (signed)</td>
<td>1111011 w mod 100 r/m</td>
<td>8-bit</td>
<td>9+EA</td>
<td>71+EA</td>
</tr>
<tr>
<td>AAM</td>
<td>ASCII adjust for multiply</td>
<td>11010100 00001010</td>
<td>83 clocks</td>
<td>90+EA</td>
<td>144+EA</td>
</tr>
<tr>
<td>DIV</td>
<td>Divide (unsigned)</td>
<td>1111011 w mod 110 r/m</td>
<td>8-bit</td>
<td>90+EA</td>
<td>155+EA</td>
</tr>
<tr>
<td>IDIV</td>
<td>Integer divide (signed)</td>
<td>1111011 w mod 111 r/m</td>
<td>8-bit</td>
<td>112+EA</td>
<td>177+EA</td>
</tr>
<tr>
<td>AAD</td>
<td>ASCII adjust for divide</td>
<td>11010101 00001010</td>
<td>80 clocks</td>
<td>90+EA</td>
<td>144+EA</td>
</tr>
<tr>
<td>CBW</td>
<td>Convert byte to word</td>
<td>10011000</td>
<td>2 clocks</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NOT</td>
<td>Invert</td>
<td>1111011 w mod 010 r/m</td>
<td>5 clocks</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SHL/SAL</td>
<td>Shift logical/arithmetic left</td>
<td>1101000 v w mod 10 r/m</td>
<td>2</td>
<td>15+EA</td>
<td>20+EA+4+bit</td>
</tr>
<tr>
<td>SHR</td>
<td>Shift logical right</td>
<td>1101000 v w mod 111 r/m</td>
<td>2</td>
<td>15+EA</td>
<td>20+EA+4+bit</td>
</tr>
<tr>
<td>SAR</td>
<td>Shift arithmetic right</td>
<td>1101000 v w mod 111 r/m</td>
<td>2</td>
<td>15+EA</td>
<td>20+EA+4+bit</td>
</tr>
</tbody>
</table>
**ROL - Rotate left**

110100 x mod 000 r/m

Timing clocks:  
- Single-bit register: 2  
- Single-bit memory: 15+EA  
- Variable-bit register: 8+4/bit  
- Variable-bit memory: 20+EA+4/bit

**ROR - Rotate right**

110100 x mod 001 r/m

Timing clocks:  
- Single-bit register: 2  
- Single-bit memory: 15+EA  
- Variable-bit register: 8+4/bit  
- Variable-bit memory: 20+EA+4/bit

**RCL - Rotate through carry left**

110100 x mod 010 r/m

Timing clocks:  
- Single-bit register: 2  
- Single-bit memory: 15+EA  
- Variable-bit register: 8+4/bit  
- Variable-bit memory: 20+EA+4/bit

**RCR - Rotate through carry right**

110100 x mod 011 r/m

Timing clocks:  
- Single-bit register: 2  
- Single-bit memory: 15+EA  
- Variable-bit register: 8+4/bit  
- Variable-bit memory: 20+EA+4/bit

**AND**

Reg/memory and register to either  
010000 x mod reg r/m

Timing clocks:  
- Register to register: 3  
- Memory to register: 9+EA  
- Register to memory: 16+EA

Immediate to register/memory  
100000 x mod 100 r/m

Timing clocks:  
- Immediate to register: 4  
- Immediate to memory: 17+EA

Immediate to accumulator  
001001 x
data
data if w=1

Timing: 4 clocks

**TEST**

And function to flags, no result  
Reg/memory and register  
100001 x mod reg r/m

Timing clocks:  
- Register to register: 3  
- Register with memory: 9+EA

Immediate data and register/memory  
111011 x mod 000 r/m

Timing clocks:  
- Immediate with register: 4  
- Immediate with memory: 10+EA

Immediate data and accumulator  
101010 x
data
data if w=1

Timing: 4 clocks

**OR**

Reg/memory and register to either
STRING MANIPULATION

REP = Repeat
1 1 1 1 0 0 1 0
Timing: 6 clocks/loop

MOVX = Move String
1 0 1 0 0 1 0 w
Timing: 17 clocks

CMPX = Compare String
1 0 1 0 0 1 1 w
Timing: 22 clocks

SCAS = Scan String
1 0 1 0 1 1 1 w
Timing: 15 clocks

LODS = Load String
1 0 1 0 1 1 0 w
Timing: 12 clocks

STOS = Store String
1 0 1 0 1 0 1 w
Timing: 10 clocks

CONTROL TRANSFER

NOTE: Queue reinitialization is not included in the timing information for transfer operations. To account for instruction loading, add 8 clocks to timing numbers.

CALL = Call
Direct within segment
1 1 1 1 0 1 0 0 disp-low disp-high
Timing: 11 clocks

Indirect within segment
1 1 1 1 1 1 1 1 mod 0 1 1 r/m
Timing: 13+E clocks

Direct intersegment
1 0 0 1 1 0 1 0 offset-low offset-high
Timing: 20 clocks seg-low seg-high

Indirect intersegment
1 1 1 1 1 1 1 1 1 mod 0 1 1 r/m
Timing: 29+E clocks

JMP = Unconditional Jump
Direct within segment
1 1 1 0 1 0 0 1 disp-low disp-high
Timing: 7 clocks

Direct within segment-short
1 1 1 0 1 0 1 1 disp
Timing: 7 clocks

Indirect within segment
1 1 1 1 1 1 1 1 mod 1 0 0 r/m
Timing: 7+E clocks

Direct intersegment
1 1 1 0 1 0 1 0 offset-low offset-high
Timing: 7 clocks seg-low seg-high

Indirect intersegment
1 1 1 1 1 1 1 1 mod 1 0 1 r/m
Timing: 16+E clocks

RET = Return from CALL
Within segment
1 1 0 0 0 0 1 0
Timing: 8 clocks

Within segment, adding immediate to SP
1 1 0 0 0 0 1 0 data-low data-high
Timing: 12 clocks

Intersegment
1 1 0 0 1 0 1 1
Timing: 18 clocks

Intersegment, adding immediate to SP
1 1 0 0 1 0 1 0 data-low data-high
Timing: 17 clocks

JE/JZ = Jump on equal/zero
0 1 1 1 0 1 0 0 disp
Timing (clocks): Jump is taken 8
Jump is not taken 4

JL/JNLE = Jump on less/not greater or equal
0 1 1 1 1 1 0 0 disp
Timing (clocks): Jump is taken 8
Jump is not taken 4

JLE/JNG = Jump on less or equal/not greater
0 1 1 1 1 1 1 disp
Timing (clocks): Jump is taken 8
Jump is not taken 4

JBE/JNAE = Jump on below/not above or equal
0 1 1 1 0 0 1 0 disp
Timing (clocks): Jump is taken 8
Jump is not taken 4

JBE/JNA = Jump on below or equal/not above
0 1 1 1 1 1 1 disp
Timing (clocks): Jump is taken 8
Jump is not taken 4

JPE/JPE = Jump on parity/parity even
0 1 1 1 1 0 1 disp
Timing (clocks): Jump is taken 8
Jump is not taken 4

JO = Jump on overflow
0 1 1 1 0 0 0 disp
Timing (clocks): Jump is taken 8
Jump is not taken 4

JS = Jump on sign
0 1 1 1 0 0 0 disp
Timing (clocks): Jump is taken 8
Jump is not taken 4
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Condition</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNE/JNZ</td>
<td>Jump on not equal/not zero</td>
<td>01110101 disp</td>
</tr>
<tr>
<td>JNL/JGE</td>
<td>Jump on not less/greater or equal</td>
<td>01111101 disp</td>
</tr>
<tr>
<td>JNE/JG</td>
<td>Jump on not equal/greater</td>
<td>01111111 disp</td>
</tr>
<tr>
<td>JNB/JAE</td>
<td>Jump on not below/above</td>
<td>01110011 disp</td>
</tr>
<tr>
<td>JNBE/JA</td>
<td>Jump on not below/above</td>
<td>01110011 disp</td>
</tr>
<tr>
<td>JNP/JPO</td>
<td>Jump on not parity/parity odd</td>
<td>01111011 disp</td>
</tr>
<tr>
<td>JNO</td>
<td>Jump on not overflow</td>
<td>01110001 disp</td>
</tr>
<tr>
<td>JNS</td>
<td>Jump on not sign</td>
<td>01111001 disp</td>
</tr>
<tr>
<td>LOOP = Loop CX times</td>
<td>11100010 disp</td>
<td>Jump is taken 9</td>
</tr>
<tr>
<td>LOOPZ/LOOPE</td>
<td>Loop while zero/equal</td>
<td>11100001 disp</td>
</tr>
<tr>
<td>LOOPNZ/LOOPNE</td>
<td>Loop while not zero/ not equal</td>
<td>11100000 disp</td>
</tr>
<tr>
<td>JCXZ = Jump on CX zero</td>
<td>11100011 disp</td>
<td>Jump is taken 9</td>
</tr>
</tbody>
</table>
### Processor Control

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Code</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLC</td>
<td>Clear carry</td>
<td>11111000</td>
<td>2 clocks</td>
</tr>
<tr>
<td>STC</td>
<td>Set carry</td>
<td>11111001</td>
<td>2 clocks</td>
</tr>
<tr>
<td>CMC</td>
<td>Complement carry</td>
<td>11110101</td>
<td>2 clocks</td>
</tr>
<tr>
<td>NOP</td>
<td>No operation</td>
<td>10010000</td>
<td>2 clocks</td>
</tr>
<tr>
<td>CLD</td>
<td>Clear direction</td>
<td>11111100</td>
<td>2 clocks</td>
</tr>
<tr>
<td>STD</td>
<td>Set direction</td>
<td>11111101</td>
<td>2 clocks</td>
</tr>
<tr>
<td>CLI</td>
<td>Clear interrupt</td>
<td>11110100</td>
<td>2 clocks</td>
</tr>
<tr>
<td>STI</td>
<td>Set interrupt</td>
<td>11110101</td>
<td>2 clocks</td>
</tr>
<tr>
<td>HLT</td>
<td>Halt</td>
<td>11110100</td>
<td>2 clocks</td>
</tr>
<tr>
<td>WAIT</td>
<td>Wait</td>
<td>10011011</td>
<td>3 clocks</td>
</tr>
<tr>
<td>LOCK</td>
<td>Bus lock prefix</td>
<td>11110000</td>
<td>2 clocks</td>
</tr>
<tr>
<td>ESC</td>
<td>Escape (to external device)</td>
<td>11011xxxx mod x x x x/m</td>
<td>7+EA clocks</td>
</tr>
</tbody>
</table>

**Footnotes:**
- If d = 1 then "to"; if d = 0 then "from"
- If w = 1 then word instruction; if w = 0 then byte instruction
- If s,w = 01 then 16 bits of immediate data form the operand
- If s,w = 11 then an immediate data byte is sign extended to form the 16-bit operand
- If v = 0 then "count" = 1; if v = 1 then "count" in (CLI)
- x = don't care
- z is used for some string primitives to compare with ZF flag
- AX = 8-bit accumulator
- AX = 16-bit accumulator
- DS = Data segment
- DX = Count register
- DX = Variable port register
- ES = Extra segment

Above/below refers to unsigned value
Greater = more positive;
Less = less positive; (more negative) signed values

See page 1 for Operand Summary.
See page 2 for Segment Override Summary.
ADDRESSING MODES 8086 / APPLICATIONS

Register Indirect Addressing

Accessing a Structure with Based Addressing

Accessing an Array with Indexed Addressing

Based Indexed Addressing

Accessing a Stack Array with Based Indexed Addressing

String Operand Addressing

I/O Port Addressing
### Parallel I/O Port Configuration

<table>
<thead>
<tr>
<th>Port</th>
<th>Mode</th>
<th>Driver (D)/Terminator (T)</th>
<th>Jumper Configuration</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Delete</td>
<td>Add</td>
</tr>
<tr>
<td>C8</td>
<td>0 input</td>
<td>8226: A8, A9</td>
<td>*21-25</td>
<td>24-25</td>
</tr>
<tr>
<td></td>
<td>0 Output</td>
<td>8226: A8, A9</td>
<td>*21-25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(latched)</td>
<td>T: A10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>D: A11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C8</td>
<td>1 input</td>
<td>8226: A8, A9</td>
<td>*21-25</td>
<td>24-25</td>
</tr>
<tr>
<td></td>
<td>(strobed)</td>
<td>T: A10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>D: A11</td>
<td>*15-16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>19-33</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>22-32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*19-20</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>and</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*32-33</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>and</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*13-14</td>
<td></td>
</tr>
<tr>
<td>C8</td>
<td>1 Output</td>
<td>8226: A8, A9</td>
<td>*21-25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(latched)</td>
<td>T: A10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>D: A11</td>
<td>*17-18</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13-33</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Default jumper connected at the factory.
<table>
<thead>
<tr>
<th>Port</th>
<th>Mode</th>
<th>Driver (D)/Terminator (T)</th>
<th>Jumper Configuration</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Delete</td>
<td>Add</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*21-25</td>
<td>17-25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*15-16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*17-18</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*19-14 and *32-33</td>
<td>13-33</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>22-32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 Input</td>
<td>T: A12, A13</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>0 Output (latched)</td>
<td>D: A12, A13</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*13-14</td>
<td>14-30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*30-31</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>26-34</td>
<td></td>
</tr>
</tbody>
</table>

*Default jumper connected at the factory.
### Parallel I/O Port Configuration

<table>
<thead>
<tr>
<th>Port</th>
<th>Mode</th>
<th>Driver (D) Terminator (T)</th>
<th>Jumper Configuration</th>
<th>Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Delete</td>
<td>Add</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*13-14</td>
<td>14-30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>and</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*30-31</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*26-27</td>
<td>26-34</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 Input</td>
<td>T: A10</td>
<td>None</td>
<td>*15-16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*19-20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*17-18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*13-14</td>
</tr>
<tr>
<td></td>
<td>0 Input</td>
<td>T: A11</td>
<td>None</td>
<td>*26-27</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*28-29</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*30-31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>*32-33</td>
</tr>
<tr>
<td></td>
<td>0 Output (latched)</td>
<td>D: A10</td>
<td>None</td>
<td>Same as for Port CC (upper) mode 0 Input.</td>
</tr>
<tr>
<td></td>
<td>0 Output (latched)</td>
<td>D: A11</td>
<td>None</td>
<td>Same as for Port CC (lower) Mode 0 Input.</td>
</tr>
</tbody>
</table>

*Default jumper connected at the factory.
Specifications

<table>
<thead>
<tr>
<th>WORD SIZE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction:</td>
<td>8, 16, 24, or 32 bits.</td>
</tr>
<tr>
<td>Data</td>
<td>8/16 bits.</td>
</tr>
<tr>
<td>CYCLE TIME:</td>
<td>800 nanosecond for fastest executable instruction (assumes instruction is in the queue).</td>
</tr>
<tr>
<td></td>
<td>1.2 microseconds for fastest executable instruction (assumes instruction is not in the queue).</td>
</tr>
<tr>
<td>MEMORY CAPACITY</td>
<td>Up to 16K bytes; user installed in 1K, 2K, or 4K byte increments.</td>
</tr>
<tr>
<td>On-Board ROM/EPROM:</td>
<td></td>
</tr>
<tr>
<td>On-Board Dynamic RAM:</td>
<td>32K bytes. Integrity maintained during power failure with user-furnished batteries.</td>
</tr>
<tr>
<td>Off-Board Expansion:</td>
<td>Up to 1 megabyte of user-specified combination of RAM, ROM, and EPROM.</td>
</tr>
<tr>
<td>MEMORY ADDRESSING</td>
<td>FF000-FFFFFH (using 2758 EPROM's),</td>
</tr>
<tr>
<td>On-Board ROM/EPROM:</td>
<td>FE000-FFFFFH (using 2316E ROM's or 2716 EPROM's), and</td>
</tr>
<tr>
<td></td>
<td>FC000-FFFFFH (using 2332 ROM's).</td>
</tr>
<tr>
<td>On-Board RAM:</td>
<td></td>
</tr>
<tr>
<td>(CPU Access)</td>
<td>00000-07FFFFH.</td>
</tr>
<tr>
<td>On-Board RAM:</td>
<td>Jumper and switches allow board to act as slave RAM device for access by another bus master. Addresses may be set within any 8K boundary of any 128K segment of the 1-megabyte system address space. Access is selectable for 8K, 16K, 24K, or 32K bytes.</td>
</tr>
<tr>
<td>(Multibus Access)</td>
<td></td>
</tr>
<tr>
<td>SERIAL COMMUNICATIONS</td>
<td>5-, 6-, 7-, or 8-bit characters.</td>
</tr>
<tr>
<td>Synchronous:</td>
<td>Internal; 1 or 2 sync characters.</td>
</tr>
<tr>
<td></td>
<td>Automatic sync insertion.</td>
</tr>
<tr>
<td>Asynchronous:</td>
<td>5-, 6-, 7-, or 8-bit characters.</td>
</tr>
<tr>
<td></td>
<td>Break character generation.</td>
</tr>
<tr>
<td></td>
<td>1, 1½, or 2 stop bits.</td>
</tr>
<tr>
<td></td>
<td>False start bit detection.</td>
</tr>
</tbody>
</table>

Sample Baud Rate:

<table>
<thead>
<tr>
<th>Frequency¹ (kHz, Software Selectable)</th>
<th>Baud Rate (Hz)²</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Synchronous</td>
</tr>
<tr>
<td>153.6</td>
<td>38400</td>
</tr>
<tr>
<td>76.8</td>
<td>19200</td>
</tr>
<tr>
<td>38.4</td>
<td>9600</td>
</tr>
<tr>
<td>19.2</td>
<td>4800</td>
</tr>
<tr>
<td>9.6</td>
<td>2400</td>
</tr>
<tr>
<td>4.8</td>
<td>1200</td>
</tr>
<tr>
<td>2.4</td>
<td>600</td>
</tr>
<tr>
<td>1.76</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes: 1. Frequency selected by I/O writes of appropriate 16-bit frequency factor to Baud Rate Register.
2. Baud rates shown here are only a sample subset of possible software-programmable rates available. Any frequency from 18.75 Hz to 613.5 kHz may be generated utilizing on-board crystal oscillator and 16-bit Programmable Interval Timer (used here as frequency divider).
INTERVAL TIMER AND BAUD RATE GENERATOR

Input Frequency (selectable):
- 2.46 MHz ±0.1% (0.41 μsec period nominal),
- 1.23 MHz ±0.1% (0.82 μsec period nominal), and
- 153.6 kHz ±0.1% (6.5 μsec period nominal).

Output Frequencies:

<table>
<thead>
<tr>
<th>Function</th>
<th>Single Timer</th>
<th>Dual Timers (Two Timers Cascaded)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min.</td>
<td>Max.</td>
</tr>
<tr>
<td>Real-Time Interrupt Interval</td>
<td>1.63 μsec</td>
<td>427.1 msec</td>
</tr>
<tr>
<td>Rate Generator (Frequency)</td>
<td>2.342 Hz</td>
<td>613.5 kHz</td>
</tr>
</tbody>
</table>

SYSTEM CLOCK (8086 CPU): 5.0 MHz ±0.1%.

I/O ADDRESSING:
All communication to Parallel I/O and Serial I/O Ports, Timer, and Interrupt Controller is via read and write commands from on-board 8086 CPU. Refer to table 3-2.

INTERFACE COMPATIBILITY
Serial I/O:
EIA Standard RS232C signals provided and supported:
- Clear to Send
- Data Set Ready
- Data Terminal Ready
- Request to Send
- Receive Clock
- Transmit Clock
- Secondary Receive Data
- Secondary CTS

Parallel I/O:
24 programmable lines (8 lines per port); one port includes bidirectional bus driver.
IC sockets included for user installation of line drivers and/or I/O terminators as required for interface ports. Refer to table 2-1.

INTERRUPTS:
8086 CPU includes non-maskable interrupt (NMI) and maskable interrupt (INTR). NMI interrupt is provided for catastrophic event such as power failure; NMI vector address is 00008. INTR interrupt is driven by on-board 8259A PIC, which provides 8-bit identifier of interrupting device to CPU. CPU multiplies identifier by four to derive vector address. Jumpers select interrupts from 16 sources without necessity of external hardware. PIC may be programmed to accommodate edge-sensitive or level-sensitive inputs.

COMPATIBLE CONNECTORS/CABLES:
Refer to table 2-2 for compatible connector details. Refer to paragraphs 2-21 and 2-22 for recommended types and lengths of I/O cables.

ENVIRONMENTAL REQUIREMENTS
Operating Temperature: 0° to 55°C (32° to 131°F).
Relative Humidity: To 90% without condensation.

PHYSICAL CHARACTERISTICS
Width: 30.48 cm (12.00 inches).
Height: 17.15 cm (6.75 inches).
Thickness: 1.78 cm (0.7 inch).
Weight: 539 grn (19 ounces).
POWER REQUIREMENTS:

<table>
<thead>
<tr>
<th>CONFIGURATION</th>
<th>$V_{CC} = +5V\pm5%$</th>
<th>$V_{DD} = +12V\pm5%$</th>
<th>$V_{BB} = -5V\pm5%$</th>
<th>$V_{AA} = -12V\pm5%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without EPROM$^1$</td>
<td>5.2A</td>
<td>350 mA</td>
<td>—</td>
<td>40 mA</td>
</tr>
<tr>
<td>RAM Only$^2$</td>
<td>390 mA</td>
<td>40 mA</td>
<td>1.0 mA</td>
<td>—</td>
</tr>
<tr>
<td>With iSBC 530$^4$</td>
<td>5.2A</td>
<td>450 mA</td>
<td>—</td>
<td>140 mA</td>
</tr>
<tr>
<td>With 4K EPROM$^5$</td>
<td>5.5A</td>
<td>450 mA</td>
<td>—</td>
<td>140 mA</td>
</tr>
<tr>
<td>(Using 2758)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>With 8K ROM$^6$</td>
<td>6.1A</td>
<td>450 mA</td>
<td>—</td>
<td>140 mA</td>
</tr>
<tr>
<td>(Using 2316E)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>With 8K EPROM$^6$</td>
<td>5.5A</td>
<td>450 mA</td>
<td>—</td>
<td>140 mA</td>
</tr>
<tr>
<td>(Using 2716)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>With 16K ROM$^5$</td>
<td>5.4A</td>
<td>450 mA</td>
<td>—</td>
<td>140 mA</td>
</tr>
<tr>
<td>(Using 2332)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes:
1. Does not include power for optional ROM/EPROM, I/O drivers, and I/O terminators.
2. Does not include power required for optional ROM/EPROM, I/O drivers, and I/O terminators.
3. RAM chips powered via auxiliary power bus.
4. Does not include power for optional ROM/EPROM, I/O drivers, and I/O terminators. Power for iSBC 530 is supplied via serial port connector.
5. Includes power required for four ROM/EPROM chips, and I/O terminators installed for 16 I/O lines; all terminator inputs low.
## I/O Address Assignments

<table>
<thead>
<tr>
<th>I/O Address*</th>
<th>Chip Select</th>
<th>Function</th>
</tr>
</thead>
</table>
| 000C0 or 000C4 | 8259A PIC | Write: ICW1, OCW2, and OCW3  
Read: Status and Poll |
| 000C2 or 000C6 | 8259A PIC | Write: ICW2, ICW3, ICW4, OCW1 (Mask)  
Read: OCW1 (Mask) |
| 000C8 | 8255A PPI | Write: Port A (J1)  
Read: Port A (J1) |
| 000CA | 8255A PPI | Write: Port B (J1)  
Read: Port B (J1) |
| 000CC | PPI | Write: Port C (J1)  
Read: Port C Status |
| 000CE | PPI | Write: Control  
Read: None |
| 000D0 | 8253 PIT | Write: Counter 0 (Load Count + N)  
Read: Counter 0 |
| 000D2 | 8253 PIT | Write: Counter 1 (Load Count + N)  
Read: Counter 1 |
| 000D4 | 8253 PIT | Write: Counter 2 (Load Count + N)  
Read: Counter 2 |
| 000D6 | 8253 PIT | Write: Control  
Read: None |
| 000D8 or 000DC | 8251A USART | Write: Data (J2)  
Read: Data (J2) |
| 000DA or 000DE | 8251A USART | Write: Mode or Command  
Read: Status |

*Odd addresses (i.e., 000C1, 000C3, .... 000DD) are illegal.
**APP. F**

**CHARACTER LENGTH**

<table>
<thead>
<tr>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**PARITY ENABLE**

1 - ENABLE
0 - DISABLE

**EVEN PARITY GENERATION/CHECK**

1 - EVEN
0 - ODD

**EXTERNAL SYNC DETECT**

1 - SYNDIC IS AN INPUT
0 - SYNDIC IS AN OUTPUT

**SINGLE CHARACTER SYNC**

1 - SINGLE SYNC CHARACTER
0 - DOUBLE SYNC CHARACTER

**NOTE:** IN EXTERNAL SYNCH MODE, PROGRAMMING DOUBLE CHARACTER SYNC WILL AFFECT ONLY THE 8

---

**USART Synchronous Mode**

**Instruction Word Format**

```
D7 D6 D5 D4 D3 D2 D1 D0
EH IR RTS ER SBRK RXE DTR TXEN
```

**TRANSMIT ENABLE**

1 - enable
0 - disable

**DATA TERMINAL READY**

"high" will force DTR output to zero

**RECEIVE ENABLE**

1 - enable
0 - disable

**SEND BREAK CHARACTER**

1 - forces TXD "low"
0 - normal operation

**ERROR RESET**

1 - reset error flags
PE, OE, PE

**REQUEST TO SEND**

"high" will force RTS output to zero

**INTERNAL RESET**

"high" returns 8251A to Mode Instruction Format

**ENTER HUNT MODE**

1 - enable search for Sync Characters

* HAS NO EFFECT IN ASYNC MODE

---

**Note:** Error Reset must be performed whenever RxEnable and Enter Hunt are programmed.

---

**USART Command**

**Instruction Word Format**
**Data Set Ready**
- DSR is generally used in test modem conditions such as Data Set Ready.

**SYNC Detect**
- When set for internal sync detect, indicates that character sync has been achieved and 8251 is ready for data.

**Communication Status**
- Indicates whether the CPU is ready to accept or has accepted a data character or command.

**Receiver Ready**
- Indicates the UART has received a character and is ready to transfer it to the CPU.

**Transmitter Empty**
- Indicates that parallel-to-serial converter or transceiver is empty.

**Parity Error**
- PE flag is set when a parity error is detected. It is read by E8 bit of Command instruction. PE does not inhibit operation of 8251.

**USART Status Read Format**

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC1</td>
<td>SC0</td>
<td>RL1</td>
<td>RL0</td>
<td>M2</td>
<td>M1</td>
<td>M0</td>
<td>BCD</td>
</tr>
</tbody>
</table>

**(Binary/BCD)**
- 0: Binary Counter (16-bits)
- 1: Binary Coded Decimal (BCD) Counter (4 Decades)

**(Mode)**
- M2 M1 M0

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Mode 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Mode 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>Mode 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>Mode 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Mode 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Mode 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Use Mode 3 for Baud Rate Generator**

**(Read/Load)**
- RL1 RL0

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- 0 0: Counter Latching operation (refer to paragraph 3-29).
- 1 0: Read/Load most significant byte only.
- 0 1: Read/Load least significant byte only.
- 1 1: Read/Load least significant byte first, then most significant byte.

**(Select Counter)**

<table>
<thead>
<tr>
<th>SC1</th>
<th>SC0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- 0 0: Select Counter 0
- 0 1: Select Counter 1
- 1 0: Select Counter 2
- 1 1: Illegal
### PROGRAMMING FORMAT

<table>
<thead>
<tr>
<th>Step</th>
<th>Mode Control Word Counter n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>LSB Count Register Byte Counter n</td>
</tr>
<tr>
<td>3</td>
<td>MSB Count Register Byte Counter n</td>
</tr>
</tbody>
</table>

### ALTERNATE PROGRAMMING FORMAT

<table>
<thead>
<tr>
<th>Step</th>
<th>Mode Control Word Counter n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Mode Control Word Counter 0</td>
</tr>
<tr>
<td>3</td>
<td>Mode Control Word Counter 1</td>
</tr>
<tr>
<td>4</td>
<td>LSB Count Register Byte Counter 1</td>
</tr>
<tr>
<td>5</td>
<td>MSB Count Register Byte Counter 1</td>
</tr>
<tr>
<td>6</td>
<td>LSB Count Register Byte Counter 2</td>
</tr>
<tr>
<td>7</td>
<td>MSB Count Register Byte Counter 2</td>
</tr>
<tr>
<td>8</td>
<td>LSB Count Register Byte Counter 0</td>
</tr>
<tr>
<td>9</td>
<td>MSB Count Register Byte Counter 0</td>
</tr>
</tbody>
</table>

### PIT Programming Sequence Examples

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC1</td>
<td>SC0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

- Don't Care
- Selects Counter Latching Operation
- Specifies Counter to be Latched

#### PIT Counter Register

Latch Control Word Format

### PIT Time Intervals Vs Timer Counts

<table>
<thead>
<tr>
<th>T</th>
<th>N*</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 µsec</td>
<td>12</td>
</tr>
<tr>
<td>100 µsec</td>
<td>123</td>
</tr>
<tr>
<td>1 msec</td>
<td>1229</td>
</tr>
<tr>
<td>10 msec</td>
<td>12288</td>
</tr>
<tr>
<td>50 msec</td>
<td>61440</td>
</tr>
</tbody>
</table>

*Count Values (N) assume clock is 1.23 MHz. Count Values (N) are in decimal.
PPI Control Word Format

PPI Port C Bit Set/Reset
Control Word Format
ICW1

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>LTIM</td>
<td>0</td>
<td>SINGL</td>
<td>1</td>
</tr>
</tbody>
</table>

- 1 = SINGLE
- 0 = NOT SINGLE

ICW2

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>ID2</td>
<td>ID1</td>
<td>ID0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- 1 = LEVEL_TRIGGERED
- 0 = EDGE_TRIGGERED

SLAVE ID

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- 1 = INPUT IS SLAVE
- 0 = INPUT IS NOT SLAVE

ICW3

| S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 |

ICW4

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>FNM</td>
<td>1</td>
<td>M/S</td>
<td>AEOI</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

- 1 = AUTO END-OF-INTERRUPT
- 0 = NOT AUTO EOI

- 1 = THIS PIC IS MASTER
- 0 = THIS PIC IS SLAVE

- 1 = FULLY NESTED MODE
- 0 = NESTED MODE

PIC Initialization Command Word Formats
PIC Operation Control Word Formats
### APP. G

**INSTRUCTION SET 8087**

<table>
<thead>
<tr>
<th>Addition</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FADD</td>
<td>Add real</td>
</tr>
<tr>
<td>FADDP</td>
<td>Add real and pop</td>
</tr>
<tr>
<td>FIADD</td>
<td>Integer add</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subtraction</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FSUB</td>
<td>Subtract real</td>
</tr>
<tr>
<td>FSUBP</td>
<td>Subtract real and pop</td>
</tr>
<tr>
<td>FISUB</td>
<td>Integer subtract</td>
</tr>
<tr>
<td>FSUBR</td>
<td>Subtract real reversed</td>
</tr>
<tr>
<td>FSUBRP</td>
<td>Subtract real reversed and pop</td>
</tr>
<tr>
<td>FISUBR</td>
<td>Integer subtract reversed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multiplication</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FMUL</td>
<td>Multiply real</td>
</tr>
<tr>
<td>FMULP</td>
<td>Multiply real and pop</td>
</tr>
<tr>
<td>FIMUL</td>
<td>Integer multiply</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Division</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FDIV</td>
<td>Divide real</td>
</tr>
<tr>
<td>FDIVP</td>
<td>Divide real and pop</td>
</tr>
<tr>
<td>FIDIV</td>
<td>Integer divide</td>
</tr>
<tr>
<td>FDIVR</td>
<td>Divide real reversed</td>
</tr>
<tr>
<td>FDIVRP</td>
<td>Divide real reversed and pop</td>
</tr>
<tr>
<td>FIDIVR</td>
<td>Integer divide reversed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other Operations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FSQRT</td>
<td>Square root</td>
</tr>
<tr>
<td>FSIGLE</td>
<td>Scale</td>
</tr>
<tr>
<td>FPREM</td>
<td>Partial remainder</td>
</tr>
<tr>
<td>FRNDINT</td>
<td>Round to integer</td>
</tr>
<tr>
<td>FXTRACT</td>
<td>Extract exponent and significand</td>
</tr>
<tr>
<td>FABS</td>
<td>Absolute value</td>
</tr>
<tr>
<td>FCHS</td>
<td>Change sign</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other Operations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FPTAN</td>
<td>Partial tangent</td>
</tr>
<tr>
<td>FPATAN</td>
<td>Partial arctangent</td>
</tr>
<tr>
<td>F2XM1</td>
<td>$2^x - 1$</td>
</tr>
<tr>
<td>FYL2X</td>
<td>$y = \log_2 x$</td>
</tr>
<tr>
<td>FYL2XP1</td>
<td>$y = \log_2 (x + 1)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Real Transfers</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FLD</td>
<td>Load real</td>
</tr>
<tr>
<td>FST</td>
<td>Store real</td>
</tr>
<tr>
<td>FSTP</td>
<td>Store real and pop</td>
</tr>
<tr>
<td>FXCH</td>
<td>Exchange registers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Integer Transfers</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FILD</td>
<td>Integer load</td>
</tr>
<tr>
<td>FIST</td>
<td>Integer store</td>
</tr>
<tr>
<td>FISTP</td>
<td>Integer store and pop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Packed Decimal Transfers</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FBLD</td>
<td>Packed decimal (BCD) load</td>
</tr>
<tr>
<td>FSSTP</td>
<td>Packed decimal (BCD) store and pop</td>
</tr>
</tbody>
</table>
iSBC 86/12A or (pSBC 86/12A*)
SINGLE BOARD COMPUTER

- 8088 16-bit HMOS microprocessor
- Central processor unit
- 32K bytes of dual-port read/write
memory expandable on-board to 64K
bytes with on-board refresh
- Sockets for up to 16K bytes of read only
memory expandable on-board to 32K
bytes
- System memory expandable to 1 me­
byte
- 24 programmable parallel I/O lines with
sockets for interchangeable line drivers
and terminators
- Programmable synchronous/asynchro­
nous RS232C compatible serial
interface with software selectable
baud rates
- Two programmable 16-bit BCD or binary
timers/event counters
- 9 levels of vectored interrupt control,
expandable to 65 levels
- Auxiliary power bus and power fail
interrupt control logic for read/write
memory battery backup
- MULTIBUS interface for multimaster
configurations and system expansion
- Compatible with ISBC 337 MULTI­
MODULE Numeric Data Processor
- Compatible with ISBC 80 family single
board computers, memory, digital and
analog I/O, and peripheral controller
boards

The iSBC 86/12A Single Board Computer is a member of Intel’s complete line of OEM microcomputer systems which take full advantage of Intel's LSI technology to provide economical self-contained computer based solutions for OEM applications. The iSBC 86/12A board is a complete computer system on a single 6.75 x 12.00-inch printed circuit card. The CPU, system clock, read/write memory, nonvolatile read only memory, I/O ports and drivers, serial communications interface, priority interrupt logic and programmable timers, all reside on the board. Full MULTIBUS Interface logic is included to offer compatibility with the Intel OEM Microcomputer Systems family of Single Board Computers, expansion memory options, digital and analog I/O expansion boards and peripheral controllers.
ISBC 86/12A

FUNCTIONAL DESCRIPTION

Central Processing Unit

The central processor for the ISBC 86/12A board is Intel’s 8086, a powerful 16-bit CMOS device. The 225 sq. mil chip contains 26,000 transistors and has a clock rate of 5MHz. The architecture includes four (4) 16-bit byte addressable data registers, two (2) 16-bit memory base pointer registers and two (2) 16-bit index registers, all accessed by a total of 24 operand addressing modes for complex data handling and very flexible memory addressing.

Instruction Set — The 8086 instruction repertoire includes variable length instruction format (including double operand-instructions), 8-bit and 16-bit signed and unsigned arithmetic operators for binary, BCD and unpacked ASCII data, and iterative word and byte string manipulation functions. In addition, the ISBC 337 MULTIMODULE Numeric Data Processor may be installed to add over 60 numeric instructions and hardware support for multiple precision integer and floating point data types.

Architectural Features — A 6-byte instruction queue provides pre-fetching of sequential instructions and can reduce the 1.2µsec minimum instruction cycle to 400 nsec for queued instructions. The stack oriented architecture facilitates nested subroutines and co-routines, reentrant code and powerful interrupt handling. The memory expansion capabilities offer a 1 megabyte addressing range. The dynamic relocation scheme allows easy in segmentation of pure procedure and data for efficient memory utilization. Four segment registers (code, stack, data, extra) contain program loaded offset values which are used to map 16-bit addresses to 20-bit addresses. Each register maps 64K-bytes at a time and activation of a specific register is controlled explicitly by program control and is also selected implicitly by specific functions and instructions.

Bus Structure

The ISBC 86/12A microcomputer has three buses: an internal bus for communicating with on-board memory and I/O options, the MULTIBUS system bus for referencing additional memory and I/O options, and the dual-port bus which allows access to RAM from the on-board CPU and the MULTIBUS system bus. Local (on-board) accesses do not require MULTIBUS communication, making the system bus available for use by other MULTIBUS masters (i.e. DMA devices and other single board computers transferring to additional system memory). This feature allows true parallel processing in a multiprocessor environment. In addition, the MULTIBUS interface can be used for system expansion through the use of other 8- and 16-bit ISBC computers, memory and I/O expansion boards.

Figure 1. ISBC 86/12A Single Board Computer Block Diagram
RAM Capabilities
The ISBC 86/12A microcomputer contains 32K bytes of dynamic read/write memory using 16K-bit 2117 RAMs. In addition, the on-board RAM complement can be expanded to 64K bytes with the ISBC 300 32K-byte MULTI-MODULE RAM option. Power for the on-board RAM and refresh circuitry may be optionally provided on an auxiliary power bus, and memory protect logic is included for RAM battery backup requirements. The ISBC 86/12A board contains a dual-port controller which allows access to the on-board RAM (32K bytes or 64K bytes when the ISBC 300 module is included with the ISBC 86/12A board) from the ISBC 86/12A CPU and from any other MULTIBUS master via the system bus. The dual-port controller allows 8- and 16-bit accesses from the MULTIBUS system bus, and the on-board CPU transfers data to RAM over a 16-bit data path. Priorities have been established such that memory refresh is guaranteed by the on-board refresh logic and that the on-board CPU has priority over MULTIBUS system bus requests for access to RAM. The dual-port controller includes independent addressing logic for RAM access from the on-board CPU and from the MULTIBUS system bus. The on-board CPU will always access RAM starting at location 00000H. Address jumpers allow on-board RAM to be located starting on any 8K-byte boundary within a 1 megabyte address range for accesses from the MULTIBUS system bus. In conjunction with this feature, the ISBC 86/12A microcomputer has the ability to protect on-board memory from MULTIBUS access to any contiguous 8K-byte segment of 16K-byte segments with ISBC 300 module. These features allow the multi-processor systems to establish local memory for each processor and shared system (MULTIBUS) memory configurations where the total system memory size (including local on-board memory) can exceed 1 megabyte without addressing conflicts.

EPROM Capabilities
Four sockets are provided for up to 16K bytes of non-volatile read only memory on the ISBC 86/12A board. EPROM may be added in 2K-byte increments up to a maximum of 4K bytes by using Intel 2758 electrically programmable ROMs (EPROMs); in 4K-byte increments up to 8K bytes by using Intel 2716 EPROMs; or in 8K-byte increments up to 16K bytes using Intel 2732 EPROMs. On-board EPROM is accessed via 16-bit data path. On-board EPROM capacity may be expanded to 32K bytes with the addition of the ISBC 340 16K-byte MULTIMODULE EPROM option. It provides an additional four sockets for Intel 2732 EPROMs. With user modification of the ISBC 86/12A's on-board memory and MULTIBUS address decode, Intel 2758 and 2716 EPROMs may be optionally supported. System memory size is easily expanded by the addition of MULTIBUS system bus compatible memory boards available in the ISBC product family.

Parallel I/O Interface
The ISBC 86/12A single board computer contains 24 programmable parallel I/O lines implemented using the Intel 8255A Programmable Peripheral Interface. The system software is used to configure the I/O lines in any combination of unidirectional input/output and bidirectional ports indicated in Table 1. Therefore, the I/O interface may be customized to meet specific peripheral requirements. In order to take full advantage of the large number of possible I/O configurations, sockets are provided for interchangeable I/O line drivers and terminators. Hence, the flexibility of the I/O interface is further enhanced by the capability of selecting the appropriate combination of optional line drivers and terminators to provide the required sink current, polarity, and drive/termination characteristics for each application. The 24 programmable I/O lines and signal ground lines are brought out to a 50-pin edge connector that mates with flat, woven, or round cable.

Serial I/O
A programmable communications interface using the Intel 8251A Universal Synchronous Asynchronous Receiver/Transmitter (USART) is contained on the ISBC 86/12A board. A software selectable baud rate generator provides the USART with all common communication features.
frequencies. The USART can be programmed by the
system software to select the desired asynchronous or
synchronous serial data transmission technique (includ­
ing IBM Bi-Sync). The mode of operation (i.e., synchro­
ous or asynchronous), data format, control character
format, parity, and baud rate are all under program
control. The 8251A provides full duplex, double buffered
transmit and receive capability. Parity, overrun, and
framing error detection are all incorporated in the
USART. The RS232C compatible interface on each
board, in conjunction with the USART, provides a direct
interface to RS232C compatible terminals, cassettes, and
asynchronous and synchronous modems. The RS232C
command lines, serial data lines, and signal ground line
are brought out to a 26 pin edge connector that mates with
RS232C compatible flat or round cable. The iSBC 530
Teletypewriter Adapter provides an optically isolated
interface for those systems requiring a 20 mA current
loop. The iSBC 530 unit may be used to interface the iSBC
86/12A board to teletypewriters or other 20 mA current
loop equipment.

Programmable Timers

The iSBC 86/12A board provides three independent, fully
programmable 16-bit interval timers/event counters
utilizing the Intel 8253 Programmable Interval Timer.
Each counter is capable of operating in either BCD or
binary modes. Two of these timers/counters are available to
the systems designer to generate accurate time intervals under software control. Routing for the outputs
and gate/trigger inputs of two of these counters is jumper
selectable. The outputs may be independently routed to
the 8255A Programmable Interrupt Controller and to the
I/O line drivers associated with the 8255A Programmable
Peripheral Interface, or may be routed as inputs to the
8255A chip. The gate/trigger inputs may be routed to I/O
terminals associated with the 8255A or as output
connections from the 8255A. The third interval timer in
the 8253 provides the programmable baud rate generator
for the iSBC 86/12A board RS232C USART serial port. In
utilizing the iSBC 86/12A board the systems designer
simply configures, via software, each timer independently
to meet system requirements. Whenever a given time
delay or count is needed, software commands to the
programmable timers/event counters select the desired
function. Seven functions are available, as shown in
Table 2. The contents of each counter may be read at any
time during system operation with simple read operations
for event counting applications, and special commands
are included so that the contents can be read "on the fly".

MULTIBUS System Bus and
Multimaster Capabilities

The MULTIBUS system bus features asynchronous data
transfers for the accommodation of devices with various
transfer rates while maintaining maximum throughput.
Twenty address lines and sixteen separate data lines
eliminate the need for address/data multiplexing/demul­
tiplexing logic used in other systems, and allow for data
transfer rates up to 5 megawords/sec. A failsafe timer is
included in the iSBC 86/12A board which can be used to
generate an interrupt if an addressed device does not
respond within 6 usec.

Table 2. Programmable Timer Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt on</td>
<td>When terminal count is reached, an interrupt</td>
</tr>
<tr>
<td>terminal count</td>
<td>request is generated. This function is</td>
</tr>
<tr>
<td>Programmable</td>
<td>extremely useful for generation of real-time</td>
</tr>
<tr>
<td>one-shot</td>
<td>clocks.</td>
</tr>
<tr>
<td>Rate generator</td>
<td>Output goes low upon receipt of an external</td>
</tr>
<tr>
<td></td>
<td>trigger edge or software command and returns</td>
</tr>
<tr>
<td></td>
<td>high when terminal count is reached. This</td>
</tr>
<tr>
<td></td>
<td>function is retriggerable.</td>
</tr>
<tr>
<td>Square-wave</td>
<td>Output will remain high until one-half the</td>
</tr>
<tr>
<td>rate generator</td>
<td>count has been completed, and go low for the</td>
</tr>
<tr>
<td></td>
<td>other half of the count.</td>
</tr>
<tr>
<td>Software</td>
<td>Output remains high until software loads count</td>
</tr>
<tr>
<td>triggered strobe</td>
<td>(N). N counts after count is loaded, output goes</td>
</tr>
<tr>
<td></td>
<td>low for one input clock period.</td>
</tr>
<tr>
<td>Hardware</td>
<td>Output goes low for one clock period N counts</td>
</tr>
<tr>
<td>triggered strobe</td>
<td>after rising edge counter trigger input. The</td>
</tr>
<tr>
<td></td>
<td>counter is retriggerable.</td>
</tr>
<tr>
<td>Event counter</td>
<td>On a jumper selectable basis, the clock input</td>
</tr>
<tr>
<td></td>
<td>becomes an input from the external system. CPU</td>
</tr>
<tr>
<td></td>
<td>may read the number of events occurring after the</td>
</tr>
<tr>
<td></td>
<td>counting &quot;window&quot; has been enabled or an</td>
</tr>
<tr>
<td></td>
<td>interrupt may be generated after N events</td>
</tr>
<tr>
<td></td>
<td>occur in the system.</td>
</tr>
</tbody>
</table>

Multimaster Capabilities — The iSBC 86/12A board is a
tul computer on a single board with resources capable of
supporting a great variety of OEM system requirements.
For those applications requiring additional processing
capacity and the benefits of multiprocessing (i.e., several
CPUs and/or controllers logically sharing system tasks
through communication over the system bus), the iSBC
86/12A board provides full MULTIBUS arbitration control
logic. This control logic allows up to three iSBC 86/12A
boards or other bus masters, including iSBC 80 family
MULTIBUS compatible 8-bit single board computers, to
share the system bus in serial (daisy chain) priority
fashion and up to 16 masters to share the MULTIBUS
system bus with the addition of an external priority
network. The MULTIBUS arbitration logic operates
synchronously with a MULTIBUS clock (provided by the
iSBC 86/12A board or optionally provided directly from
the MULTIBUS) while data is transferred via a handshake
between the master and slave modules. This allows
different speed controllers to share resources on the same
bus, and transfers via the bus proceed asynchronously.
Thus, transfer speed is dependent on transmitting and
Interrupt Capability

The ISBC 86/12A board provides 9 vectored interrupt levels. The highest level is the NMI (Non-maskable Interrupt) line which is directly tied to the 8086 CPU. This interrupt cannot be inhibited by software and is typically used for signalling catastrophic events (i.e., power failure). On servicing this interrupt, program control will be implicitly transferred through location 00000h. The Intel 8259A Programmable Interrupt Controller (PIC) provides vectoring for the next eight interrupt levels. As shown in Table 3, a selection of four priority processing modes is available to the systems designer for use in designing request processing configurations to match system requirements. Operating mode and priority assignments may be reconfigured dynamically via software at any time during system operation. The PIC accepts interrupt requests from the programmable parallel and serial I/O interfaces, the programmable timers, the system bus, or directly from peripheral equipment. The PIC then determines which of the incoming requests is of the highest priority, determines whether this request is of higher priority than the level currently being serviced, and, if appropriate, issues an interrupt to the CPU. Any combination of interrupt levels may be masked, via software, by storing a single byte in the interrupt mask register of the PIC. The PIC generates a unique memory address for each interrupt level. These addresses are equally spaced at 4 byte intervals. This 32-byte block may begin at any 32-byte boundary in the lowest 1K-bytes of memory,* and contains unique instruction pointers and code segment offset values (for expanded memory operation) for each interrupt level. After acknowledging an interrupt and obtaining a device identifier byte from the 8259A PIC, the CPU will store its status flags on the stack and execute an indirect CALL instruction through the vector location (derived from the device identifier) to the interrupt service routine. In systems requiring additional interrupt levels, slave 8259A PIC's may be interfaced via the MULTIBUS system bus, to generate additional vector addresses, yielding a total of 65 unique interrupt levels.

Interrupt Request Generation — Interrupt requests may originate from 18 sources. Two jumper selectable interrupt requests can be automatically generated by the programmable peripheral interface when a byte of information is ready to be transferred to the CPU (i.e., input buffer is full) or a byte of information has been transferred to a peripheral device (i.e., output buffer is empty). Two jumper selectable interrupt requests can be automatically generated by the USART when a character is ready to be transferred to the CPU (i.e., receive channel buffer is full, or a character is ready to be transmitted (i.e., transmit channel data buffer is empty). A jumper selectable request can be generated by each of the programmable timers. An additional interrupt request line may be jumpered directly from the parallel I/O driver terminator section. Eight prioritized interrupt request lines allow the ISBC 86/12A board to recognize and service interrupts originating from peripheral boards interfaced via the MULTIBUS system bus. The MULTIBUS fail safe timer of the ISBC 337 processor and the exception and error output signal also can be selected as interrupt sources.

Power-Fail Control

Control logic is also included to accept a power-fail interrupt in conjunction with the AC-low signal from the ISBC 635 and ISBC 640 Power Supply or equivalent.

Expansion Capabilities

Memory and I/O capacity may be expanded and additional functions added using Intel MULTIBUS compatible expansion boards. Memory may be expanded by adding user specified combinations of RAM boards, EPROM boards, or combination boards. Input/output capacity may be increased by adding digital I/O and analog I/O expansion boards. Mass storage capability may be achieved by adding single or double density diskette controllers, or hard disk controllers. Modular expandable backplanes and card cages are available to support multibus systems.

Note: Certain system restrictions may be incurred by the inclusion of some of the ISBC 80 family options in an ISBC 86/12A system. Consult the Intel OEM Microcomputer System Configuration Guide for specific data.

---

**Table 3. Programmable Interrupt Modes**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully nested</td>
<td>Interrupt request line priorities fixed at 0 as highest, 7 as lowest.</td>
</tr>
<tr>
<td>Auto-rotating</td>
<td>Equal priority. Each level, after receiving service, becomes the lowest priority level until next interrupt occurs.</td>
</tr>
<tr>
<td>Specific priority</td>
<td>System software assigns lowest priority level. Priority of all other levels based in sequence numerically on this assignment.</td>
</tr>
<tr>
<td>Polled</td>
<td>System software examines priority-encoded system interrupt status via interrupt status register.</td>
</tr>
</tbody>
</table>

*Note: The first 32 vector locations are reserved by Intel for dedicated vectors. Users who wish to maintain compatibility with present and future Intel products should not use these locations for user-defined vector addresses.
ISBC 86/12A

System Development Capabilities

The development cycle of ISBC 86/12A products can be significantly reduced by using the Intellec series microcomputer development system. The Assembler, High Level Languages, Locating Linker, Library Manager, Text Editor and System Monitor are all supported by the ISIS-II disk-based operating system.

In-Circuit Emulator — ICE-86 in-circuit emulator provides the necessary link between the software development environment provided by the Intellec system and the "target" ISBC 86/12A execution system. In addition to providing the mechanism for loading executable code and data into the ISBC 86/12A board, ICE-86 in-circuit emulator provides a sophisticated command set to assist in debugging software and final integration of the user hardware and software. ICE-86 in-circuit emulator maximizes the use of available development resources by allowing Intellec resident resources (e.g., memory and peripherals) to be accessed by software running on the target ISBC 86/12A system. In addition, software can be executed without an ISBC 86/12A execution vehicle, in 2K bytes of RAM resident in the ICE-86 system itself. Symbolic references to instruction and data locations can be made through ICE-86 in-circuit emulator to allow the user to reference memory locations with assigned names.

PL/M-86 — Intel’s high level programming language. PL/M-86, is also available as an Intellec Microcomputer Development System option. PL/M-86 provides the capability to program in a natural, algorithmic language and eliminates the need to manage register usage or allocate memory. PL/M-86 programs can be written in a much shorter time than assembly language programs for a given application. PL/M-86 includes byte and word, integer, pointer and floating point (32-bit) data types and also includes conditional compilation and macro features.

SPECIFICATIONS

Word Size
Instruction — 8, 16, 24, or 32 bits
Data — 8, 16 bits

Cycle Time
Basic Instruction Cycle — 1.2 μsec
— 400 nsec (assumes instruction in the queue)

Memory Capacity
On-Board Read Only Memory — 16K bytes (sockets only); expandable to 32K bytes with ISBC 340 16K-byte MULTIMODULE EPROM option.
On-Board RAM — 32K bytes; expandable to 64K bytes with ISBC 300 32K-byte MULTIMODULE RAM option.
Off-Board Expansion — Up to 1 megabyte in user specified combinations of RAM and EPROM.

Serial Communications Characteristics
Synchronous — 5—8 bit characters; internal or external character synchronization; automatic sync insertion.
Asynchronous — 5—8 bit characters; break character generation; 1, 1½, or 2 stop bits; false start bit detection.

Baud Rates

<table>
<thead>
<tr>
<th>Frequency (kHz) (Software Selectable)</th>
<th>Baud Rate (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td>Asynchronous</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>153.6</td>
<td>19000</td>
</tr>
<tr>
<td>78.8</td>
<td>9600</td>
</tr>
<tr>
<td>38.4</td>
<td>4800</td>
</tr>
<tr>
<td>19.2</td>
<td>2400</td>
</tr>
<tr>
<td>9.6</td>
<td>1200</td>
</tr>
<tr>
<td>4.8</td>
<td>600</td>
</tr>
<tr>
<td>2.4</td>
<td>300</td>
</tr>
<tr>
<td>1.76</td>
<td>150</td>
</tr>
</tbody>
</table>

Note: Frequency selected by I/O write of appropriate 16-bit frequency factor to baud rate register (8253 Timer 2).
Interrupts
Addresses for 8259A Registers (Hex notation I/O address space)
C0 or C4: Write: initialization Command Word 1 (ICW1) and Operation Control Words 2 and 3 (OCW2 and OCW3)
Read: Status and Poll Registers
C2 or C6: Write: ICW2, ICW3, ICW4, OCW1 (Mask Register)
Read: OCW1 (Mask Register)

Note:
Several registers have the same physical address; sequence of access and one data bit of control word determine which register will respond.

Interrupt Levels — 8086 CPU includes a non-maskable Interrupt (NMI) and a maskable interrupt (INTR). NMI interrupt is provided for catastrophic events such as power failure. INTR interrupt is driven by on-board 8259A PIC, which provides 8-bit identifier of interrupting device to CPU. CPU multiplies identifier by four to derive vector address. Jumpers select interrupts from 18 sources without necessity of external hardware. PIC may be programmed to accommodate edge-sensitive or level-sensitive inputs.

Timers
Register Addresses (Hex notation, I/O address space)
D0: Timer 0
D2: Timer 1
D4: Timer 2
D6: Control register

Note:
Timer counts are loaded as two sequential output operations to same address as given.

Input Frequencies
Reference: 2.46 MHz ± 0.1% (0.041 μs period, nominal); 1.23 MHz ± 0.1% (0.81 μs period, nominal); or 153.6 kHz ± 0.1% (6.51 μs period nominal).

Note:
Above frequencies are user selectable.

Event Rate: 2.46 MHz max

Output Frequencies/Timing Intervals

<table>
<thead>
<tr>
<th>Function</th>
<th>Single Timer/Counter</th>
<th>Dual Timer/Counter (Two Timers Cascaded)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>Real-time interrupt</td>
<td>1.63 μs</td>
<td>427.1 ms</td>
</tr>
<tr>
<td>Programmable one-shot</td>
<td>1.83 μs</td>
<td>427.1 ms</td>
</tr>
<tr>
<td>Rate generator</td>
<td>2.342 Hz</td>
<td>613.5 kHz</td>
</tr>
<tr>
<td>Square-wave rate generator</td>
<td>2.342 Hz</td>
<td>613.5 kHz</td>
</tr>
<tr>
<td>Software-triggered strobe</td>
<td>1.63 μs</td>
<td>427.1 ms</td>
</tr>
<tr>
<td>Hardware-triggered strobe</td>
<td>1.63 μs</td>
<td>427.1 ms</td>
</tr>
<tr>
<td>Event counter</td>
<td>—</td>
<td>2.46 MHz</td>
</tr>
</tbody>
</table>

Interfaces
MULTIBUS — All signals TTL compatible
Parallel I/O — All signals TTL compatible
Interrupt Requests — All TTL compatible
Timer — All signals TTL compatible
Serial I/O — RS232C compatible, data set configuration

System Clock (8086 CPU)
5.00 MHz ± 0.1%

Auxiliary Power
An auxiliary power bus is provided to allow separate power to RAM for systems requiring battery backup of read/write memory. Selection of this auxiliary RAM power bus is made via jumpers on the board.

Connectors

<table>
<thead>
<tr>
<th>Interface</th>
<th>Pin No.</th>
<th>Center (in.)</th>
<th>Mating Connectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus</td>
<td>86</td>
<td>0.156</td>
<td>VKING 3KH3J94MK12</td>
</tr>
<tr>
<td>Parallel I/O</td>
<td>50</td>
<td>0.1</td>
<td>J3 3415-000</td>
</tr>
<tr>
<td>Serial I/O</td>
<td>26</td>
<td>0.1</td>
<td>J3 3482-000</td>
</tr>
</tbody>
</table>

Memory Protect
An active low TTL compatible memory protect signal is brought out on the auxiliary connector which, when asserted, disables read/write access to RAM memory on the board. This input is provided for the protection of RAM contents during system power down sequences.

Line Drivers and Terminators
I/O Drivers — The following line drivers are all compatible with the I/O driver sockets on the iSBC 86/12A board.

<table>
<thead>
<tr>
<th>Driver</th>
<th>Characteristic</th>
<th>Sink Current (mA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7436</td>
<td>LOC</td>
<td>40</td>
</tr>
<tr>
<td>7437</td>
<td>I</td>
<td>40</td>
</tr>
<tr>
<td>7432</td>
<td>NI</td>
<td>10</td>
</tr>
<tr>
<td>7428</td>
<td>LOC</td>
<td>10</td>
</tr>
<tr>
<td>7409</td>
<td>NI,OC</td>
<td>10</td>
</tr>
<tr>
<td>7408</td>
<td>NI</td>
<td>10</td>
</tr>
<tr>
<td>7403</td>
<td>LOC</td>
<td>10</td>
</tr>
<tr>
<td>7400</td>
<td>I</td>
<td>10</td>
</tr>
</tbody>
</table>

Note:
I = inverting; NI = non-inverting; OC = open collector.
Port 1 of the 8255A has 20 mA totem-pole bidirectional drivers and 1 kΩ terminators.

I/O Terminators — 220Ω/330Ω divider or 1 kΩ pullup

220Ω/330Ω (iSBC 8612 OPTION)

Note:
- +5V
- 1kΩ (iSBC 862 OPTION)
### Physical Characteristics

- **Width**: 12.00 in. (30.48 cm)
- **Height**: 6.75 in. (17.15 cm)
- **Depth**: 0.70 in. (1.78 cm)
- **Weight**: 19 oz. (539 gm)

### Electrical Characteristics

#### DC Power Requirements

<table>
<thead>
<tr>
<th>Configuration</th>
<th>V(_{CC}) = ±5V</th>
<th>V(_{DD}) = ±12V</th>
<th>I(_{BA}) = -5V</th>
<th>I(_{AA}) = -12V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without EPROM(^1)</td>
<td>5.2A</td>
<td>350 mA</td>
<td>—</td>
<td>40 mA</td>
</tr>
<tr>
<td>RAM Only(^2)</td>
<td>390 mA</td>
<td>40 mA</td>
<td>10 mA</td>
<td>—</td>
</tr>
<tr>
<td>With EPROM(^3)</td>
<td>5.2A</td>
<td>450 mA</td>
<td>—</td>
<td>140 mA</td>
</tr>
<tr>
<td>Using 2716</td>
<td>5.5A</td>
<td>350 mA</td>
<td>—</td>
<td>40 mA</td>
</tr>
<tr>
<td>With 8K EPROM(^4)</td>
<td>5.5A</td>
<td>350 mA</td>
<td>—</td>
<td>40 mA</td>
</tr>
<tr>
<td>Using 2725</td>
<td>5.5A</td>
<td>350 mA</td>
<td>—</td>
<td>40 mA</td>
</tr>
<tr>
<td>With 16K EPROM(^5)</td>
<td>5.4A</td>
<td>350 mA</td>
<td>—</td>
<td>40 mA</td>
</tr>
</tbody>
</table>

**Notes:**

1. Does not include power for optional EPROM, I/O drivers, and I/O terminators.
2. Does not include power required for optional EPROM, I/O drivers, and I/O terminators.
3. RAM chips powered via auxiliary power bus.
4. Does not include power for optional EPROM, I/O drivers, and I/O terminators. Power for ISBC 530 is supplied via serial port connector.
5. Includes power required for four EPROM chips, and I/O terminators installed for 16 I/O lines; all terminator inputs low.

### ORDERING INFORMATION

- **Part Number**: SBC 86/12A
- **Description**: Single Board Computer with 32K bytes RAM
iSBC 337
MULTIMODULE NUMERIC DATA PROCESSOR

- High speed fixed and floating point functions for ISBC 86, 88 and IAPX 86, 88 systems
- MULTIMODULE option containing 8087 Numeric Data Processor
- Supports seven data types including single and double precision integer and floating point
- Implements proposed IEEE Floating Point Standard for high accuracy
- Extends host CPU instruction set with arithmetic, logarithmic, transcendental and trigonometric instructions
- 50 x performance improvements in Whetstone benchmarks over IAPX 86/10 performance
- Software support through ASM-86/88 Assembly Language and High Level Languages

The Intel iSBC 337 MULTIMODULE Numeric Data Processor offers high performance numerics support for ISBC 86 and ISBC 88 Single Board Computer users, for applications including simulation, instrument automation, graphics, signal processing and business systems. The coprocessor interface between the 8087 and the host CPU provides a simple means of extending the instruction set with over 60 additional numeric instructions supporting six additional data types. The data formats conform to the proposed IEEE Floating Point Standard insuring highly accurate results. The MULTIMODULE implementation allows the iSBC 337 module to be used on all ISBC 86 and ISBC 88 Microcomputers and can be added as an option to custom IAPX 86 and IAPX 88 board designs.
OVERVIEW

The ISBC 337 MULTIMODULE Numeric Data Processor provides arithmetic and logical instruction extensions to the 8086 and 8088 CPU's of the IAPX 86 and IAPX 88 families, to provide IAPX 86/20 and IAPX 88/20 Numeric Data Processors. The instruction set consists of arithmetic, transcendental, logical, trigonometric and exponential instructions which can all operate on seven different data types. The data types are 16, 32, and 64 bit integer, 32 and 64 bit floating point, 18 digit packed BCD and 80 bit temporary.

Coprocessor Interface

The coprocessor interface between the host CPU (8086 and 8088) and the ISBC 337 processor provides easy to use and high performance math processing. Installation of the ISBC 337 processor is simply a matter of removing the host CPU from its socket, installing the ISBC 337 processor into the host's CPU socket, and reinstalling the host CPU chip into the socket provided for it on the ISBC 337 processor (see Figure 1). All synchronization and timing signals are provided via the coprocessor interface with the host CPU. The two processors also share a common address/data bus. (See Figure 2.) The 8087 Numeric Data Processor (NDP) component is capable of recognizing and executing 8087 numeric instructions as they are fetched by the host CPU. This interface allows concurrent processing by the host CPU and the 8087. It also allows 8087 and host CPU instructions to be intermixed in any fashion to provide the maximum overlapped operation and the highest aggregate performance.

High Performance and Accuracy

The 80-bit wide internal registers and data paths contribute significantly to high performance and
minimizes the execution time difference between single and double precision floating point formats. This 80-bit architecture, in conjunction with the use of the proposed IEEE Floating Point Standard provides very high resolution and accuracy. This precision is complemented by extensive exception detection and handling. Six different types of exceptions can be reported and handled by the 8087. The user also has control over internal precision, infinity control and rounding control.

SYSTEM CONFIGURATION

As a coprocessor to an 8086 or 8088, the 8087 is wired in parallel with the CPU as shown in Figure 2. The CPU's status and queue status lines enable the NDP to monitor and decode instructions in synchronization with the CPU and without any CPU overhead. Once started, the 8087 can process in parallel with and independent of the host CPU. For resynchronization, the NDP's BUSY signal informs the CPU that the NDP is executing an instruction and the CPU WAIT instruction tests this signal to insure that the NDP is ready to execute subsequent instructions. The NDP can interrupt the CPU when it detects an error or exception. The interrupt request line is routed to the CPU through an 8259A Programmable Interrupt Controller. This interrupt request signal is brought down from the ISBC 337 module to the ISBC 86, 88 Single Board Computer through a single pin connector (see Figure 1). The signal is then routed to the interrupt matrix for jumper connection to the 8259A Interrupt Controller. Other IAPX 86 and 88 designs may use a similar arrangement, or by masking off the 8086's "READ" pin from the ISBC 337 socket, provisions are made to allow the now vacated pin of the host's CPU socket to be used to bring down
PROGRAMMING INTERFACE

Table 1 lists the seven data types the 8087 supports and presents the format for each type. Internally, the 8087 holds all numbers in the temporary real format. Load and store instructions automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 54-bit floating point numbers or 18-digit packed BCD numbers into temporary real format and, vice versa.

Computations in the 8087 use the processor's register stack. These eight 80-bit registers provide the equivalent capacity of 40 16-bit registers. The 8087 register set can be accessed as a stack, with instructions operating on the top stack element, or as a fixed register set, with instructions operating on explicitly designated registers.

Table 3 lists the 8087's instructions by class. Assembly language programs are written in ASM 86/88, the IAPX 86, 88 assembly language. Table 2 gives the execution times of some typical numeric instructions and their equivalent time on a 5 MHz 8086.

FUNCTIONAL DESCRIPTION

The NDP is internally divided into two processing elements, the control unit (CU) and the numeric execution unit (NEU), providing concurrent operation of the two units. The NEU executes all numeric instructions, while the CU receives and decodes instructions, reads and writes memory operands and executes processor control instructions.

Control Unit

The CU keeps the 8087 operating in synchronization with its host CPU. 8087 instructions are intermixed with CPU instructions in a single instruc-
structlons, however. CU makes use of a memory address and initiates a bus cycle, but does not store the address which the CPU calculates the operand starting at the specified address. The CPU does provide addressing for ESC instructions, however.

An 8087 instruction either will not reference memory, will require loading one or more operands from memory into the 8087 into memory. In the first case a non-memory reference escape is used to start 8087 operation. In the last two cases, the CU makes use of a "dummy read" cycle initiated by the CPU, in which the CPU calculates the operand address and initiates a bus cycle, but does not capture the data. Instead, the CPU captures and saves the address which the CPU places on the bus. If the instruction is a load, the CU additionally captures the data word when it becomes available on the local data bus. If data required is longer than one word, the CU immediately obtains the bus from the CPU using the request/grant protocol and reads the rest of the information in consecutive bus cycles. In a store operation, the CU captures and saves the store address as in a load, and ignores the data word that follows in the "dummy read" cycle. When the 8087 is ready to perform the store, the CU obtains the bus from the CPU and writes the operand starting at the specified address.

Numeric Execution Unit

The NEU executes all instructions that involve the register stack; these include arithmetic, logical, transcendental, constant and data transfer instructions. The data path in the NEU is 80 bits wide (64 fraction bits, 15 exponent bits and a sign bit) which allows internal operand transfers to be performed at very high speeds.

When the NEU begins executing an instruction, it activates the 8087 BUSY signal. This signal is
used in conjunction with the CPU WAIT instruction to resynchronize both processors when the NEU has completed its current instruction.

Register Set

The 8087 register set is shown in Figure 3. Each of the eight data registers in the 8087's register stack is 80 bits wide and is divided into "fields" corresponding to the NDP's temporary real data type. The register set may be addressed as a push down stack, through a top of stack pointer or any register may be addressed explicitly relative to the top of stack.

Status Word

The status word shown in Figure 4 reflects the overall state of the 8087; it may be stored in memory and then inspected by CPU code. The status word is a 16-bit register divided into fields as shown in Figure 4. The busy bit (bit 15) indicates whether the NEU is executing an instruction (B = 1) or is idle (B = 0). Several instructions which store and manipulate the status word are executed exclusively by the CU, and these do not set the busy bit themselves.

The four numeric condition code bits (C0-C3) are similar to the flags in a CPU; various instructions update these bits to reflect the outcome of NDP operations.

Bits 14-12 of the status word point to the 8087 register that is the current top-of-stack (TOP).

Bit 7 is the interrupt request bit. This bit is set if any unmasked exception bit is set and cleared otherwise.

Bits 5-0 are set to indicate that the NEU has detected an exception while executing an instruction.

Tag Word

The tag word marks the content of each register as shown in Figure 5. The principal function of the tag word is to optimize the NDP's performance. The tag word can be used, however, to interpret the contents of 8087 registers.
Instruction and Data Pointers

The instruction and data pointers (see Figure 6) are provided for user-written error handlers. Whenever the 8087 executes an NEU Instruction, the CU saves the instruction address, the operand address (if present) and the Instruction opcode. The 8087 can then store this data in memory.

Control Word

The NDP provides several processing options which are selected by loading a word from memory into the control word. Figure 7 shows the format and encoding of the fields in the control word.

Exception Handling

The 8087 detects six different exception conditions that can occur during Instruction execution. Any or all exceptions will cause an interrupt if unmasked and interrupts are enabled.

If interrupts are disabled the 8087 will simply suspend execution until the host clears the exception. If a specific exception class is masked and that exception occurs, however, the 8087 will post the exception in the status register and perform an on-chip default exception handling procedure, thereby allowing processing to continue. The exceptions that the 8087 detects are the following:

1. INVALID OPERATION: Stack overflow, stack underflow, Indeterminate form (0/0, -, etc.) or the use of a Non-Number (NAN) as an operand. An exponent value is reserved and any bit pattern with this value in the exponent field is termed a Non-Number and causes this exception. If this exception is masked, the 8087's default response is to generate a specific NAN called INDEFINITE, or to propagate already existing NANs as the calculation result.

2. OVERFLOW: The result is too large in magnitude to fit the specified format. The 8087 will generate the code for infinity if this exception is masked.

---

**Figure 5. 8087 Tag Word**

**Figure 6. 8087 Instruction and Data Pointers**

**Figure 7. 8087 Control Word**
3. ZERO DIVISOR: The divisor is zero while the dividend is a non-infinite, non-zero number. Again, the 8087 will generate the code for infinity if this exception is masked.

4. UNDERFLOW: The result is non-zero but too small in magnitude to fit in the specified format. If this exception is masked the 8087 will denormalize (shift right) the fraction until the exponent is in range. This process is called gradual underflow.

5. DENORMALIZED OPERAND: At least one of the operands or the result is denormalized; it has the smallest exponent but a non-zero significand. Normal processing continues if this exception is masked off.

5. INEXACT RESULT: If the true result is not exactly representable in the specified format, the result is rounded according to the rounding mode, and this flag is set. If this exception is masked, processing will simply continue.

SOFTWARE SUPPORT
The iSBC 337 module is supported by Intel's ASM-86/88 Assembly Language and PL/M-86/88 Systems Implementation Language. In addition to the instructions provided in the languages to support the additional math functions, a software simulator is also available to allow the execution of IAPX 86/20 instructions without the need for the iSBC 337 module. This allows for the development of software in an environment without the IAPX 86/20 processor and then transporting the code to its final run time environment with no change in mathematical results.

SPECIFICATIONS
Physical Characteristics
Width — 5.33 cm (2.100")
Length — 5.08 cm (2.000")
Height — 1.82 cm (.718")
SBC 337 board +
host board
Weight — 17.33 grams (.576 oz.)

Electrical Characteristics
DC Power Requirements (8087 only)

Vcc = 5V ± 5% Icc = 475 mA max.

Environmental Characteristics
Operating Temperature — 0°C to 55°C
Free air moving across base board and iSBC 337 module.
Relative Humidity — up to 90% R.H. without condensation.

Reference Manual
142887-001 — iSBC 337 MULTIMODULE Numeric Data Processor Hardware Reference Manual (NOT SUPPLIED)
Manuals may be ordered from any Intel sales representative, distributor office, or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.
J1 EWMC out
Poort1 (4020 H)

DAC/ output to MOTOR-DRIVE
I_EFFECT

I_EFFECT (EXPANDED)
P-I EFFECTS

P-D EFFECTS
I-D EFFECTS

P-I-D EFFECTS
I_EFFECT (PASCAL86)
SOURCE NAME: GETDEC

**********

ASSUME CS:CODE

EXTRN VAL09HAR,DE3:FAI~,CMFAR,CI:FAR

DCODE SEGMENT WORD PUBLIC

PUBLIC GETDEC

GETDEC PROC FAR

BEGIN:

PUSH AX
PUSH CX
PUSH ES

MAIN:

MOV AX,3130H
IN11.

MOV BX,3030H
INn.

MOV CX,11030H
;INlT.

CALL Cl iENTER FIRST CHARACTER
CALL CO .PUT IT ON THE SCREEN

CMP CH,2EH
;IT IS '0'?

...IE PUSIT!T! IT INTO CH

MOV CH,AL
MOVE IT INTO CH

CMP CH+ODH
;IS IT '0'?

JE CHOIR
;GO TO LEVEL CHOIR

CMP CH+2EH
;IS IT '+'?

JE PUSITIVE1
;GO TO LEVEL PUSITIVE1

CMP CH+2DH
;IS IT '-'?

JE NEGATIVE1
;GO TO LEVEL NEGATIVE1
LOC  OBJ  LINE  SOURCE
0027  1A0000  51  MOV  DX+0H
002A  0E0001  52  MOV  ES+DX
002C  9A0000----  E  53  CALL  VAL09
0031  BAC5  54  MOV  AL,CH
0033  055  55
0033  56  BACK1:
0034  9A0000----  E  56  CALL  CI
0039  0A0000----  E  59  CALL  CO
003E  BAE8  60  MOV  CH+AL
0040  5B  61  POP  AX
0041  0F000D  62  CMP  CH,00H
0044  744B  63  JE  CONTEND
0046  9A0000----  E  64  CALL  VAL09
004B  E52990  65
004C  055  66
004E  9A0000----  E  67  BACK2:
0053  9A0000----  E  68  CALL  CI
0050  BAE8  70  MOV  CH+AL
005A  0F000D  71  CMP  CH,00H
005D  7426  72  JE  CRDIR2
005F  9A0000----  E  73  CALL  VAL09
0064  BAC5  74  MOV  AL,CH
0066  E6CB  75  JMP  BACK1
0068  055  76
006B  9A0000  77  MOV  DX+0H
006C  BAC2  79  MOV  ES+DX
006D  EEDF  80  JMP  BACK2
006F  0A000  81
0072  BAC2  83  NEGATIVE1:
0074  E6DB  85  MOV  DX+0H
0074  E6DB  86  JMP  BACK2
0076  0000  88  SHIFT:
0076  BACF  89  MOV  CL+EH
0078  0A08  90  MOV  BL+BL
007A  BADC  91  MOV  BL+AH
007C  BAE8  92  MOV  AH+AL
007E  BACF  93  MOV  AL+CH
0080  EB1  94  JMP  BACK1
0082  055  95
0082  E93000  97  CROIR:
0085  EB1  98
0085  B93030  99  MOV  AX:030H
0088  B63030  100  MOV  ES:030H
008B  EB0190  102  JMP  CONTEND
008E  104
008E  8CC2  105  MOV  DX+ES
8086/87/88/186 MACRO ASSEMBLER  GETDEC

<table>
<thead>
<tr>
<th>LOC</th>
<th>DEJ</th>
<th>LINE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0090</td>
<td>BAE</td>
<td>106</td>
<td>MOV CH, CLD</td>
</tr>
<tr>
<td>0092</td>
<td>9A0000</td>
<td>107</td>
<td>CALL DE3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>0097</td>
<td>0/</td>
<td>109</td>
<td>POP ES</td>
</tr>
<tr>
<td>0098</td>
<td>59</td>
<td>110</td>
<td>POP CX</td>
</tr>
<tr>
<td>0099</td>
<td>5B</td>
<td>111</td>
<td>POP AX</td>
</tr>
<tr>
<td>009A</td>
<td>CB</td>
<td>112</td>
<td>RET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>113</td>
<td>GETDEC ENDP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>114</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>115</td>
<td>DCODE ENDS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>116</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>117</td>
<td>END</td>
</tr>
</tbody>
</table>

ASSEMBLY COMPLETE, NO ERRORS FOUND
SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE VAL09
OBJECT MODULE PLACED IN IFI:VAL09.OBJ
ASSEMBLER INVOKED BY: ASMB86.86 IFI:VAL09.ASM

LOC OBJ LINE SOURCE

1 NAME VAL09
2 
3 *******************************************************
4 * VAL09 TESTS IF THE CONTENTS OF THE CH REGISTER
5 * IS BETWEEN 0 AND 9, OR ' ', IN THE LAST CASE,
6 * IT CONVERTS IT INTO THE 'I' ASCII.
7 * IT CALLSISTEXT.
8 *
9 *
10 *
11 *******************************************************
12 
13 
14 ASSUME CS:CODE,DS:DATA
15 
16 DATA SEGMENT WORD PUBLIC
17 
18 ERR1 DB 0D,0A,OH,ERROR1 OUT OF RANGE 0,9',OAH,OAH,0
19 
20 DATA ENDS
21 
22 
23 EXTRN TEXTIFAR
24 
25 CODE SEGMENT WORD PUBLIC
26 
27 ; VAL09 PROC FAR
28 
29 mov dx,DATA
30 mov ds,dx
31
32 mov ch,ch
33 cmp ch,20h
34 jae EMPTY
35 cmp ch,39h
36 jae ERROR1
37 cmp ch,39h
38 jae ERROR1
39 cmp ch,39h
40 jae ERROR1
41 push ax
42 push bx
43
44
ASSEMBLY COMPLETE, NO ERRORS FOUND
```
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0D</td>
<td>01</td>
</tr>
<tr>
<td>0001</td>
<td>0A</td>
<td>02</td>
</tr>
<tr>
<td>0002</td>
<td>4952524F522E20</td>
<td>03</td>
</tr>
<tr>
<td>0003</td>
<td>31535420349441</td>
<td>04</td>
</tr>
<tr>
<td>0004</td>
<td>522E204932204E</td>
<td>05</td>
</tr>
<tr>
<td>0005</td>
<td>4F543A30262C31</td>
<td>06</td>
</tr>
<tr>
<td>0006</td>
<td>20</td>
<td>07</td>
</tr>
<tr>
<td>0007</td>
<td>0A</td>
<td>08</td>
</tr>
<tr>
<td>0008</td>
<td>0D</td>
<td>09</td>
</tr>
<tr>
<td>0009</td>
<td>0D</td>
<td>10</td>
</tr>
<tr>
<td>000A</td>
<td>0A</td>
<td>11</td>
</tr>
<tr>
<td>000B</td>
<td>0D</td>
<td>12</td>
</tr>
<tr>
<td>000C</td>
<td>0A</td>
<td>13</td>
</tr>
<tr>
<td>000D</td>
<td>0D</td>
<td>14</td>
</tr>
<tr>
<td>000E</td>
<td>0A</td>
<td>15</td>
</tr>
<tr>
<td>000F</td>
<td>0D</td>
<td>16</td>
</tr>
<tr>
<td>0010</td>
<td>0A</td>
<td>17</td>
</tr>
<tr>
<td>0011</td>
<td>0D</td>
<td>18</td>
</tr>
<tr>
<td>0012</td>
<td>0A</td>
<td>19</td>
</tr>
<tr>
<td>0013</td>
<td>00</td>
<td>20</td>
</tr>
<tr>
<td>0014</td>
<td>0D</td>
<td>21</td>
</tr>
<tr>
<td>0015</td>
<td>0A</td>
<td>22</td>
</tr>
<tr>
<td>0016</td>
<td>0D</td>
<td>23</td>
</tr>
<tr>
<td>0017</td>
<td>0A</td>
<td>24</td>
</tr>
<tr>
<td>0018</td>
<td>0D</td>
<td>25</td>
</tr>
<tr>
<td>0019</td>
<td>0A</td>
<td>26</td>
</tr>
<tr>
<td>001A</td>
<td>0D</td>
<td>27</td>
</tr>
<tr>
<td>001B</td>
<td>0A</td>
<td>28</td>
</tr>
<tr>
<td>001C</td>
<td>0D</td>
<td>29</td>
</tr>
<tr>
<td>001D</td>
<td>0A</td>
<td>30</td>
</tr>
<tr>
<td>001E</td>
<td>0D</td>
<td>31</td>
</tr>
<tr>
<td>001F</td>
<td>0A</td>
<td>32</td>
</tr>
<tr>
<td>0020</td>
<td>0D</td>
<td>33</td>
</tr>
<tr>
<td>0021</td>
<td>0A</td>
<td>34</td>
</tr>
<tr>
<td>0022</td>
<td>0D</td>
<td>35</td>
</tr>
<tr>
<td>0023</td>
<td>0A</td>
<td>36</td>
</tr>
<tr>
<td>0024</td>
<td>0D</td>
<td>37</td>
</tr>
<tr>
<td>0025</td>
<td>0A</td>
<td>38</td>
</tr>
<tr>
<td>0026</td>
<td>0D</td>
<td>39</td>
</tr>
<tr>
<td>0027</td>
<td>0A</td>
<td>40</td>
</tr>
<tr>
<td>0028</td>
<td>0D</td>
<td>41</td>
</tr>
</tbody>
</table>
```
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0024</td>
<td>E80690</td>
<td>42</td>
<td>JMP</td>
<td>ERROR</td>
</tr>
<tr>
<td>0027</td>
<td></td>
<td>43</td>
<td>NEGATIVE;</td>
<td></td>
</tr>
<tr>
<td>0027</td>
<td>F7DB</td>
<td>44</td>
<td>NLC</td>
<td>BX,</td>
</tr>
<tr>
<td>0029</td>
<td></td>
<td>45</td>
<td>POSITIVE;</td>
<td></td>
</tr>
<tr>
<td>0029</td>
<td>E80F90</td>
<td>46</td>
<td>JMP</td>
<td>BACK</td>
</tr>
<tr>
<td>002C</td>
<td></td>
<td>47</td>
<td>ERROR;</td>
<td></td>
</tr>
<tr>
<td>002C</td>
<td>59</td>
<td>58</td>
<td></td>
<td></td>
</tr>
<tr>
<td>002D</td>
<td>59</td>
<td>59</td>
<td>PUSH</td>
<td>AX</td>
</tr>
<tr>
<td>002D</td>
<td>59</td>
<td>59</td>
<td>PUSH</td>
<td>BX</td>
</tr>
<tr>
<td>002E</td>
<td>001E0000</td>
<td>50</td>
<td>LEA</td>
<td>BX,TEXT</td>
</tr>
<tr>
<td>0032</td>
<td>9A0000</td>
<td>51</td>
<td>CALL</td>
<td>TEXT</td>
</tr>
<tr>
<td>0037</td>
<td>59</td>
<td>59</td>
<td>POP</td>
<td>BX</td>
</tr>
<tr>
<td>0037</td>
<td>58</td>
<td>58</td>
<td>POP</td>
<td>AX</td>
</tr>
<tr>
<td>0039</td>
<td>CC</td>
<td>59</td>
<td>INT</td>
<td>3</td>
</tr>
<tr>
<td>003A</td>
<td>56</td>
<td>56</td>
<td>RET</td>
<td></td>
</tr>
<tr>
<td>003A</td>
<td>CB</td>
<td>56</td>
<td>DE3</td>
<td>ENOP</td>
</tr>
<tr>
<td>59</td>
<td></td>
<td>59</td>
<td>CCODE</td>
<td>ENDS</td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
<td>END</td>
<td></td>
</tr>
</tbody>
</table>

ASSEMBLY COMPLETE, NO ERRORS FOUND
LOCATION SEGMENT PUBLIC

NAME DECHEX

ECOHEXAOECIMAAL CONVERTOR

INPUT: CL,EX,AX REGISTERS, CL MOST SIGNIFICANT

OUTPUT: X, BINARY

ASSUME CS:CODE, DS:DATA

DATA SEGMENT WORD PUBLIC

LEN DB
HUNDB DB
THOUS DB
TENTH DB

DATA ENDS

CODE SEGMENT WORD PUBLIC

DECHEX PROC FAR

MOV DX,DATA
MOV DS,DX

BEGIN:

; CLEAR DX

MOV DL,AL

HINTS TO OUTPUT

; NOW THE TENTHS

MOV AL,DL

CALCULATE THE TENTHS

; AND ADD TO DX

ADD DX,AX

; NOW TO the HUNDREDs

MOV AL,DL

NOW THE THOUSANDS

; CLEAR AH REGISTER

MOV AH,DX

RESULT NOW TO BX

; TENTHONS

MOV AL,CL

100-0DS:THOUS

MOV AH,DX

MUL DS:THOUS

ADD DX,AX

; READY

RET

BACK TO CALLER

END

; DATA ENDP
ASSEMBLY COMPLETE, NO ERRORS FOUND
ASSEMBLY COMPLETE, NO ERRORS FOUND
MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE CI

OBJECT MODULE PLACED IN: F1:CI.OBJ
ASSEMBLER INVOKED BY: ASMBOB:FUCIA.SAM

ASSEMBLY COMPLETE, NO ERRORS FOUND
ASSEMBLY COMPLETE, NO ERRORS FOUND
```assembly
; DECOUT

; CODE SEGMENT

PUBLIC DECOUT
PUBLIC DECOU
PUBLIC OECOUT

; MAIN:

PUSH AX
PUSH CX
MOV EAX, EAX
TEST 13h, 0000FFFFh
JS CHANGE
MOV CH, 2BH
...IMP LEV1
CHANG: MOV CH, 20H
POH BX
LEVI: PUSH AX
MOV AL, CH
CAL1. TEXTU
POP AX
CALL TEXTO
MOV CH, OH
MOV AL, CL
CMP AL, 20H
LEA ENC, ADD AL, 0FH
ADD AL, 30H
MOV CH, UH
COCL! CALL TEXTO
MOV AL, BH
...IMP AL, 01h
A!:ALCH
TEST
SIGN
GO 1
CHANG SIGN
IFUT '-'
PRINT '+'
PRINT '-'
COMPLEMENT THE NUMBER

;RESET THE Flag

;BEGINNING WITH THOUSANDS

;IF 1'-1'0

;RESET .IF

;IF CL=0

;PRINT .CONVERT HEX->BCD

;TO USE WITH UNITS

;BEGINNING WITH TENTHOUANDS

;BEGINNING WITH THOUSANDS

;BEGINNING WITH OTHERS

;BEGINNING WITH OTHERS ARE ALSO 0

;PRINT .CONVERT ASCII

;PRINT .CONVERT ASCII

;PRINT .CONVERT ASCII

;PRINT .CONVERT ASCII
```

*Location* LOC | *OBJ* | *LINE* | *SOURCE*
---|---|---|---
1 | ; NAME DECOUT | | 
2 | ;| | 
3 | ; | | 
4 | ; | | 
5 | ; | | 
6 | ; | | 
7 | ; | | 
8 | ; | | 
9 | ; | | 
10 | ; | | 
11 | ; | | 
12 | ; | | 
13 | ; | | 
14 | ; | | 
15 | ; | | 
16 | ; | | 
17 | ; | | 
18 | ; | | 
19 | ; | | 
20 | ; | | 
21 | ; | | 
22 | ; | | 
23 | ; | | 
24 | ; | | 
25 | ; | | 
26 | ; | | 
27 | ; | | 
28 | ; | | 
29 | ; | | 
30 | ; | | 
31 | ; | | 
32 | ; | | 
33 | ; | | 
34 | ; | | 
35 | ; | | 
36 | ; | | 
37 | ; | | 
38 | ; | | 
39 | ; | | 
40 | ; | | 
41 | ; | | 
42 | ; | | 
43 | ; | | 
44 | ; | | 
45 | ; | | 
46 | ; | | 
47 | ; | | 
48 | ; | | 
49 | ; | | 
50 | ; | |
LOC   OBJ     LINE   SOURCE
0044 B020   51   MOV   AL,20H ;WRITE
0046 E001   52   MOV   CH,1H ;SET THE FLAG=1
0048 E0050   53   JMP   C00H ;PUT BH ON THE SCREEN
004E 0430   54   ASCII2ADD AL,30H ;CONVERT IT INTO ASCII
0050 E500   55   MOV   CH+BH ;RESET THE FLAG=0
0054 9A0000---- E 56   C0DH:  CALL   TEXT0 ;PUT BH ON THE SCREEN
0055 3C00   57   MOV   AL,BL ;BEGINNING WITH HUNDRES
0056 750C   58   CMP   AL,0H ;IF BL=0
005A F6C01   59   JNC   ASCII1L ;IF THE OTHERS ARE ALSO 0
005D 7407   60   JE ASCII2L ;BL=0 BUT NOT THE OTHERS
005F E020   61   MOV   AL,20H ;WRITE
0061 E501   62   MOV   CH+1H ;SET THE FLAG=1
0063 E050   63   JMP   COEL ;PUT BL ON THE SCREEN
0066 0430   64   ASCII2ADD AL,30H ;CONVERT IT INTO ASCII
0069 E500   65   MOV   CH+BH ;RESET THE FLAG=0
006A 9A0000---- E 66   COEL:  CALL   TEXT0 ;PUT BL ON THE SCREEN
006F BAC9   67   MOV   AL,AL ;BEGINNING WITH TENTHS
0071 3C00   68   CMP   AL,0H ;AH=0
0073 750C   69   JNC   ASCII1AH ;IF AH=0
0075 F6C01   70   JE ASCII1AL ;AH=0 BUT NOT THE OTHERS
007D 7407   71   MOV   AL,20H ;WRITE
007F E020   72   MOV   CH+1H ;SET THE FLAG=1
0082 EE20   73   JMP   COAH ;PUT AH ON THE SCREEN
0085 0430   74   ASCII2ADD AL,30H ;CONVERT IT INTO ASCII
0088 E500   75   MOV   CH+BH ;RESET THE FLAG=0
0089 9A0000---- E 76   COAH:  CALL   TEXT0 ;PUT AH ON THE SCREEN
008A 5B     77   POP   AX ;RECOVER AL VALUE
008B BAC9   78   MOV   AL,AL ;BEGINNING WITH UNITS
008D 0430   79   ADD   AL,30H ;CONVERT IT INTO ASCII
008F E500   80   MOV   CH+BH ;RESET THE FLAG=0
0091 9A0000---- E 81   CALL   TEXT0 ;PUT AL ON THE SCREEN
0093 E000   82   RJN   ASCII2 ;RESTORE AL
0095 59     83   POP   CX
0097 5B     84   POP   AX
0099 CB     85   RET
009A CB     86   DECOUT ENDP
009B 5B     87   POP   AX
009D CB     88   RET
009E 59     89   POP   CX
009F 5B     90   POP   AX
00A0 CB     91   RET
00A1 CB     92   DECOUT ENDP
00A2 CB     93   CCODE ENDS
00A3 CB     94   ;
00A4 CB     95   END

ASSEMBLY COMPLETE, NO ERRORS FOUND
SERIES-III 0086/07/08/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE TEXT0
OBJECT MODULE PLACED IN :F1:TEXT0.OBJ
ASSEMBLER INVOKED BY: ASM86.8A :F1:TEXT0.ASM

LOC OBJ LINE SOURCE
1 NAME TEXT0
2 ;
3 ;******************************************************************************
4 ; TEXT0 TESTS IF ASCII VALUE IN AX IS '+' '-' OR '
5 ; BETWEEN 0 AND 9.
6 ;******************************************************************************
7 ;
8 ; ASSUME CS:CODE,DS:DATA
9 ;
10 DATA SEGMENT PUBLIC
11 ;
12 TEXTER DB 0001 0A 00 0A, 'IT IS AN ERROR OUT OF RANGE: 0..9,OAH,0DH,0
13 ;
14 DATA ENDS
15 ;
16 CODE SEGMENT WORD PUBLIC
17 ;
18 PUBLIC TEXTO
19 TEXTO PROC FAR
20 ;******************************************************************************
21 CMP AL,20H ;TEST IF 0
22 JE CONT ;
23 CMP AL,2AH ;TEST IF '+'
24 JE CONT ;
25 CMP AL,30H ;TEST IF '-'
26 JE CONT ;
27 CMP AL,39H ;TEST IF <9
28 JB ERROR ;
29 CMP AL,3AH ;TEST IF >9
30 JA ERROR ;
31 JH CONT ;
32 ERROR1 PUSH AX ;SAVE
33 PUSH EX ;SAVE
34 MOV BX,DATA ;INIT. DS
35 MOV DS,EX ;
36 LEA BX,TEXTER ;INIT. TEXT
37 CALL TEXT ;
38 MOV AX,ES ;RECOVER
39 MOV AX,ES ;RECOVER
40 JMP ENDFL
41 CONT CALL CO ;

<table>
<thead>
<tr>
<th>LOC OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0031 0B</td>
<td>12</td>
<td>ENDLEVEL:</td>
</tr>
<tr>
<td>0031 CB</td>
<td>43</td>
<td>RET</td>
</tr>
<tr>
<td></td>
<td>44</td>
<td>TEXT0 ENDP</td>
</tr>
<tr>
<td></td>
<td>45</td>
<td>CODE: ENDS</td>
</tr>
<tr>
<td></td>
<td>46</td>
<td>;</td>
</tr>
<tr>
<td></td>
<td>47</td>
<td>END</td>
</tr>
</tbody>
</table>

ASSEMBLY COMPLETE, NO ERRORS FOUND
MACRO ASSEMBLER HEXBCD

SERIES-III 8086/80/80/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE HEXBCD
OBJECT MODULE PLACED IN IF1:HEXCD.OBJ
ASSEMBLER INVOKED BY: ASM86.B6 IF1:HEXCD.ASM

ASSEMBLY COMPLETE, NO ERRORS FOUND
```
APP. M
INITIO ( INTEGR. 6086 )

SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE INITIO
OBJECT MODULE PLACED IN :Fi:INITIO.OBJ
ASSEMBLER INVOKED BY: ASM686 1Fi:INITIO.ASM

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>FA</td>
<td>28</td>
<td>INITI</td>
</tr>
<tr>
<td>0001</td>
<td>E017</td>
<td>29</td>
<td>CLI</td>
</tr>
<tr>
<td>0002</td>
<td>E0C0</td>
<td>30</td>
<td>MOV AL,1H</td>
</tr>
<tr>
<td>0005</td>
<td>E020</td>
<td>31</td>
<td>OUT 0CHH,AL</td>
</tr>
<tr>
<td>0007</td>
<td>E6C2</td>
<td>32</td>
<td>MOV AL,20H</td>
</tr>
<tr>
<td>0009</td>
<td>E01F</td>
<td>33</td>
<td>OUT 02CH,AL</td>
</tr>
<tr>
<td>000B</td>
<td>E6C2</td>
<td>34</td>
<td>MOV AL,1FH</td>
</tr>
<tr>
<td>000D</td>
<td>E08B</td>
<td>35</td>
<td>OUT 02CH,AL</td>
</tr>
<tr>
<td>000F</td>
<td>E6C2</td>
<td>36</td>
<td>MOV AL,MRT(*)</td>
</tr>
<tr>
<td>0011</td>
<td>E036</td>
<td>37</td>
<td>MOV AL,JOH</td>
</tr>
<tr>
<td>0013</td>
<td>E006</td>
<td>38</td>
<td>OUT 006H,AL</td>
</tr>
<tr>
<td>0015</td>
<td>E07B</td>
<td>39</td>
<td>MOV AL,07BH</td>
</tr>
<tr>
<td>0017</td>
<td>E080</td>
<td>40</td>
<td>OUT 008H,AL</td>
</tr>
<tr>
<td>0019</td>
<td>E000</td>
<td>41</td>
<td>MOV AL,0</td>
</tr>
<tr>
<td>001B</td>
<td>E600</td>
<td>42</td>
<td>OUT 004H,AL</td>
</tr>
<tr>
<td>001D</td>
<td>E90A00</td>
<td>43</td>
<td>MOV CX,100</td>
</tr>
<tr>
<td>0020</td>
<td>E800</td>
<td>44</td>
<td>MOV AL,B0H</td>
</tr>
<tr>
<td>0022</td>
<td>E6CE</td>
<td>45</td>
<td>OUT 0CEH,AL</td>
</tr>
<tr>
<td>0024</td>
<td>OE----</td>
<td>46</td>
<td>MOV EX,DATA</td>
</tr>
<tr>
<td>0027</td>
<td>OECB</td>
<td>47</td>
<td>MOV DS,EX</td>
</tr>
<tr>
<td>0029</td>
<td>C70609000000</td>
<td>48</td>
<td>MOV STIN,0</td>
</tr>
<tr>
<td>002F</td>
<td>C70602009000</td>
<td>49</td>
<td>MOV SOH,0</td>
</tr>
</tbody>
</table>
```

---

LINE  | OBJ | SOURCE |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>INTPTR SEGMENT AT - 0H</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>TYPE2 DD TIMINT</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>INTPTR ENDS</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>ASSUME CS:CSEDE, DS:IDDATA</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>DDATA SEGMENT WORD PUBLIC</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>STIN DH ?</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>SOM DH ?</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>INT DH ?</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>SANT DH ?</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>DDATA ENDS</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>CSEDE SEGMENT WORD PUBLIC</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>INITI</td>
<td></td>
</tr>
</tbody>
</table>

---

The above code snippet is an assembly language listing for the INITIO module. It includes various assembly instructions and directives to initialize and set up the microprocessor's resources, and it appears to be part of a larger assembly source file.
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0035</td>
<td>C70604000500</td>
<td>R</td>
<td>51</td>
</tr>
<tr>
<td>0036</td>
<td>C70660000100</td>
<td>R</td>
<td>52</td>
</tr>
<tr>
<td>0041</td>
<td>880000</td>
<td>R</td>
<td>53</td>
</tr>
<tr>
<td>0044</td>
<td>E8</td>
<td></td>
<td>54</td>
</tr>
<tr>
<td>0045</td>
<td>E6CC</td>
<td></td>
<td>55</td>
</tr>
<tr>
<td>0046</td>
<td>E6CC</td>
<td></td>
<td>56</td>
</tr>
<tr>
<td>0047</td>
<td>EF0E</td>
<td></td>
<td>57</td>
</tr>
<tr>
<td>0048</td>
<td>F7DB</td>
<td></td>
<td>58</td>
</tr>
<tr>
<td>0049</td>
<td>ID25</td>
<td></td>
<td>59</td>
</tr>
<tr>
<td>004A</td>
<td>8B0A00</td>
<td></td>
<td>60</td>
</tr>
<tr>
<td>004B</td>
<td>8F05</td>
<td></td>
<td>61</td>
</tr>
<tr>
<td>004C</td>
<td>8F05</td>
<td></td>
<td>62</td>
</tr>
<tr>
<td>004D</td>
<td>8F05</td>
<td></td>
<td>63</td>
</tr>
<tr>
<td>004E</td>
<td>50</td>
<td></td>
<td>64</td>
</tr>
<tr>
<td>004F</td>
<td>E00500</td>
<td></td>
<td>65</td>
</tr>
<tr>
<td>0050</td>
<td>55</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td>0051</td>
<td>E6Ca</td>
<td></td>
<td>67</td>
</tr>
<tr>
<td>0052</td>
<td>E6Ca</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td>0053</td>
<td>E6Ca</td>
<td></td>
<td>69</td>
</tr>
<tr>
<td>0054</td>
<td>E6Ca</td>
<td></td>
<td>70</td>
</tr>
<tr>
<td>0055</td>
<td>E6Ca</td>
<td></td>
<td>71</td>
</tr>
<tr>
<td>0056</td>
<td>E6Ca</td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>0057</td>
<td>FB</td>
<td></td>
<td>73</td>
</tr>
<tr>
<td>0058</td>
<td>CF</td>
<td></td>
<td>74</td>
</tr>
<tr>
<td>0059</td>
<td>0059</td>
<td></td>
<td>75</td>
</tr>
<tr>
<td>005A</td>
<td>8B1E0000</td>
<td></td>
<td>76</td>
</tr>
<tr>
<td>005B</td>
<td>8B7E0000</td>
<td></td>
<td>77</td>
</tr>
<tr>
<td>005C</td>
<td>8B7E0000</td>
<td></td>
<td>78</td>
</tr>
<tr>
<td>005D</td>
<td>A30000</td>
<td></td>
<td>79</td>
</tr>
<tr>
<td>005E</td>
<td>B81E0400</td>
<td></td>
<td>80</td>
</tr>
<tr>
<td>005F</td>
<td>00C9</td>
<td></td>
<td>81</td>
</tr>
<tr>
<td>0060</td>
<td>710E</td>
<td></td>
<td>82</td>
</tr>
<tr>
<td>0061</td>
<td>7906</td>
<td></td>
<td>83</td>
</tr>
<tr>
<td>0062</td>
<td>80FF</td>
<td></td>
<td>84</td>
</tr>
<tr>
<td>0063</td>
<td>E50000</td>
<td></td>
<td>85</td>
</tr>
<tr>
<td>0064</td>
<td>8E90</td>
<td></td>
<td>86</td>
</tr>
<tr>
<td>0065</td>
<td>5F</td>
<td></td>
<td>87</td>
</tr>
<tr>
<td>0066</td>
<td>8E87</td>
<td></td>
<td>88</td>
</tr>
<tr>
<td>0067</td>
<td>8F70</td>
<td></td>
<td>89</td>
</tr>
<tr>
<td>0068</td>
<td>9E7A</td>
<td></td>
<td>90</td>
</tr>
<tr>
<td>0069</td>
<td>BA0000</td>
<td></td>
<td>91</td>
</tr>
<tr>
<td>006A</td>
<td>D7F3</td>
<td></td>
<td>92</td>
</tr>
<tr>
<td>006B</td>
<td>D1E2</td>
<td></td>
<td>93</td>
</tr>
<tr>
<td>006C</td>
<td>7007</td>
<td></td>
<td>94</td>
</tr>
<tr>
<td>006D</td>
<td>38D3</td>
<td></td>
<td>95</td>
</tr>
</tbody>
</table>

; !ALLOW INTERRUPTS |
<p>| CONTENTS OF AL TO OUTPUT PORT C |
| MAIN IS MAIN PROGRAMME TO BE INTERRUPTED |
| !INTEGRATOR OUTPUT TO PORT A |
| !RESTORE AX |
| !BACK TO MAIN |</p>
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
<th>ALU Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0085</td>
<td>7309</td>
<td>106</td>
<td>+2</td>
<td>JAL</td>
</tr>
<tr>
<td>0087</td>
<td>E0290</td>
<td>107</td>
<td>+2</td>
<td>JMP</td>
</tr>
<tr>
<td>0089</td>
<td>40</td>
<td>108</td>
<td>+2</td>
<td>INC03</td>
</tr>
<tr>
<td>0088</td>
<td>9D</td>
<td>109</td>
<td>+1</td>
<td>INC</td>
</tr>
<tr>
<td>008C</td>
<td>7902</td>
<td>110</td>
<td>+2</td>
<td>UTS04</td>
</tr>
<tr>
<td>008E</td>
<td>F70B</td>
<td>111</td>
<td>+1</td>
<td>POP25</td>
</tr>
<tr>
<td>0090</td>
<td></td>
<td>112</td>
<td>+2</td>
<td>JSFW</td>
</tr>
<tr>
<td>0093</td>
<td></td>
<td>113</td>
<td>+1</td>
<td>MOV</td>
</tr>
<tr>
<td>0098</td>
<td>B61060</td>
<td>114</td>
<td>+2</td>
<td>MOV</td>
</tr>
<tr>
<td>0094</td>
<td>F7EB</td>
<td>115</td>
<td>+1</td>
<td>IMUL</td>
</tr>
<tr>
<td>0096</td>
<td>710F</td>
<td>116</td>
<td>+1</td>
<td>JML</td>
</tr>
<tr>
<td>0098</td>
<td>B1208B</td>
<td>117</td>
<td>+1</td>
<td>JML</td>
</tr>
<tr>
<td>009C</td>
<td>7906</td>
<td>118</td>
<td>+1</td>
<td>MOV</td>
</tr>
<tr>
<td>009E</td>
<td>B08000</td>
<td>119</td>
<td>+1</td>
<td>MOV</td>
</tr>
<tr>
<td>00A1</td>
<td>EB0A90</td>
<td>120</td>
<td>+1</td>
<td>MOV</td>
</tr>
<tr>
<td>00A4</td>
<td></td>
<td>121</td>
<td>+2</td>
<td>POSIT</td>
</tr>
<tr>
<td>00A5</td>
<td>B8FF7F</td>
<td>122</td>
<td>+1</td>
<td>MOV</td>
</tr>
<tr>
<td>00A7</td>
<td></td>
<td>123</td>
<td>+2</td>
<td>MOV</td>
</tr>
<tr>
<td>00A7</td>
<td>C3</td>
<td>124</td>
<td>+1</td>
<td>RET</td>
</tr>
<tr>
<td>00A7</td>
<td></td>
<td>125</td>
<td>+1</td>
<td>RECTN</td>
</tr>
<tr>
<td>128</td>
<td></td>
<td>126</td>
<td>+2</td>
<td>CODE</td>
</tr>
<tr>
<td>129</td>
<td></td>
<td>127</td>
<td>+1</td>
<td>RET</td>
</tr>
<tr>
<td>130</td>
<td></td>
<td>131</td>
<td>END</td>
<td></td>
</tr>
</tbody>
</table>

ASSEMBLY COMPLETE, NO ERRORS FOUND
APP. N  PIDOUT (P.I.D.) 8086

;**********************************************************************************************
;* THIS PROGRAM PLIFORMS A P.I.D. CONTROL ON AN INPUT SIGNAL EPSILON. *
;* THE INBYTE SELECTS THE MODE: 0 -> OUTPUT=0 *
;* 1->DIFFER. *
;* 2->INTEGR. *
;* 3->INTEGR.+DIFFER. *
;* 4->PROP. *
;* 5->PROP.+DIFFER. *
;* 6->PROP.+INTEGR. *
;* 7->PROP.+INTEGR.+DIFFER. *
;* THE INPUT SIGNAL IS GENERATED BY MEANS OF THE TIMER0 OF THE INTEL'S 8080 BOARD. THE OUTPUT OF THE SYSTEM GOES TO PORTA OF THE INTEL 8255. *
;**********************************************************************************************

;MACRO1: ADD AND ROUND

;DEFINE(ADD) LOCAL NEGAT OUTS(
    ADD AX, EX
    JNO XOUTS
    JNS XNEGAT
    MOV AX, 7FFFH
    JMP XOUTS
    XNEGAT:
    MOV AX, 8000H
    XOUTS:
)

;MACRO2: SUBTRACT AND ROUND

;DEFINE(SUB) LOCAL NEGAT OUTS(
    SUB AX, EX
    JNO XOUTS
    JNS XNEGAT
    MOV AX, 7FFFH
    JMP XOUTS
    XNEGAT:
    MOV AX, 8000H
    XOUTS:
)

;MACRO3: MULTIPLY AND ROUND

;DEFINE(MULTI) LOCAL POSIT OUTS(
    IMUL EX
    JNO XOUTS
    AND DX, 8000H
    JZ XPOSIT
    MOV AX, 8000H
    JMP XOUTS
    XPOSIT:
    MOV AX, 7FFFH
    XOUTS:
)

;MACRO4: DIVIDE AND ROUND

;DEFINE(DIVIDE) LOCAL EDIV INCR OUT1 OUTS(
ORG AX+AX
PUSHF
JNS XEDIV
NEG AX
XEDIV:
MOV DX,0
DIV EX
SHL DX,1
JO XINCR
CMP DX,EX
JAE XINCR
JMP XOUT1
XINCR:
INC AX
XOUT1:
POPF
JNS XOUTS
NEG AX
XOUTS:
INTPTR SEGMENT AT 0H
ORG 0BH
TYPE2 DD TIMINT
INTPTR ENDS
ASSUME CS:CODE, DS:DDATA
DDATA SEGMENT WORD PUBLIC
INSTBYTE DW ?? "THIS BYTE SELECTS ALGORITHM"
0:INPUT=0
1:DIFER
2:INTEGR
3:INTEGR+DIFFER
4:PROPUR
5:PROPUR+DIFFER
6:PROP+INTEGR
7:PROP+INTEGR+DIFFER
Epsilon DW ?? "INPUT SIGNAL TO THE ALGORITHM"
SMIT DW ?? "SAMPLE TIME"
STIN DW ?? "INTEGRATOR STORAGE"
PROP DW 1H "PROPORTIONAL CONSTANT"
INTT DW ?? "INTEGRATION TIME"
DIFF DW 1H "DIFFERENTIATING TIME"
OUTPUT DW ?? "OUTPUT SIGNAL"
OLD DW ?? "USED IN DIFFER"
NEW DW ?? "USED IN DIFFER"
DDATA ENDS

CODE SEGMENT WORD PUBLIC
INIT:

; DISABLE INTERRUPTS
CLI

; H2H TO PIC
MOV AL, 17H

; VECTOR BASE IS 4x20H
MOV AL, 20H

; H2H TO PIC
MOV AL, 1FH

; H2H TO PIC
MOV AL, NOT(4) ; ALLOW INT2(2)

; H2H TO PIC, ALLOW INT2
MOV AL, 36H

; INIT TIMER 0 FOR 1MS COUNT
MOV AL, 0FH
OUT 00H, AL

; 1MSB TIME (0492D CYCLES=400US)
MOV AL, 1H
OUT 00H, AL

; H3B TIME
MOV CX, 10D

; COUNT FOR 4MSEC
MOV AL, 80H

; INIT Slate: ALL PORTS=OUTPUT
OUT 0CH, AL

; INIT DDATA SEGMENT
MOV DS:BX, DDATA

; INITIATING OF INTEGRATOR
MOV STINn0

; INITIATING OF OUTPUT
MOV OUTPUT, 0

; INITIATING OF OLD
MOV INTT, 5

; INITIATING OF INT
MOV SANT, 1

; INITIATING OF SANT

; MOV AX, 5

; SQUAREWAVE'S AMPLITUDE

; ALLOW INTERRUPTS
STI

MAIN:

; CONTENTS OF AL TO OUTPUT FOR C
OUT OCCH, AL

; MAIN IS MAIN PROGRAM TO BE INTERRUPTED
JMP MAIN

; ; ;
TIMINT PROC
LOOP ONE

; INVERT OUTPUT FOR SQUARE WAVE
MLC AX

; LOAD COUNT AGAIN
MOV CX, 10D

; ONE:
MOV OUTPUT, 0

; CLEAN OUTPUT
MOV EPISILON, AX

; INITIAL VALUE
PUSH AX

; INSTRBYTE, 0
CHF INSTRBYTE, 0

; ITEST IF 0 REQUEST
JNZ PID

; INSTRBYTE, 0
MOV AX, OUTPUT

; NEXT
JMP PID

; TEST INSTRBYTE, 4

; ITEST IF PROP. REQUEST
JE LINT

; PROPER
CALL PROPER

; PERFORM PROPER ROUTINE
MOV OUTPUT, AX

; LINT:
TEST INSTRBYTE, 2

; ITEST IF INTEGR. REQUEST
JE LOFF

; INTEGR
CALL INTEGR

; PERFORM INTEGR. ROUTINE
NOV
U1ACRO1
:ADDIT
MOV
LDIFF: TEST INSTBYTE+1
JE NEXT
CALL DIFFER
MOV BX+OUTPUT
IMACRO1
:ADDIT
NEXT: OUT 0CH+AL SPID OUTPUT TO PORT A
POP AX
STI
IRET
TIIINT ENDP
;
;
PROPOR PROC
MOV AX, EPSILON
MOV BX, KPROP
IMACRO3
:MULTI
RET
PROPOR ENDP
;
;
INTEGR PROC
MOV AX, EPSILON
MOV BX, STIN
IMACRO1
:ADDIT
MOV STIN+AX
MOV BX, INTT
IMACRO4
:DIVIDE
MOV BX, SAHT
IMACRO3
:MULTI
RET
INTEGR ENDP
;
;
DIFFER PROC
MOV AX, EPSILON
MOV BX, OLD
MOV OLD+AX
IMACRO2
:XSUB
MOV BX, DIFFT
IMACRO3
:MULTI
MOV BX, SAHT
IMACRO4
INTOUT (INTEGR.) PASCAL '86

SOURCE FILE: F11:INTOUT.SRC
OBJECT FILE: F1:INTOUT.OBJ

CONTROLS SPECIFIED: CODE.

PROCEDURE INITIALIZECHIP53; (* INITIALIZE THE TIMER 8253 BY SETTING THE CONTROL WORD TO SELECT COUNTER, READ/LOAD LOW-ORDER BYTE THEN HIGH-ORDER BYTE, SQUARE WAVE GENERATOR; THE PROCEDURE ALSO LOADS THE COUNTER WITH THE VALUE 496D, THEN THE COUNTER WILL COUNT DOWN FOR 496D BEATS.*)

VAR COUNTER: RECORD CASE BOOLEAN OF TRUE: (FULLWORD:WORD); FALSE: (LOW,HIGH:..255)
END;

VAR COUNT: RECORD CASE BOOLEAN OF TRUE: (FULLWORD:WORD); FALSE: (LOW,HIGH:..255)
END;

BEGIN
  COUNTER.FULLWORD:='496D' (*INITIALIZE CHIP53*);
  OUTBYTE(COUNTER.FULLHIGH,COUNTER.FULLLOW);
  OUTBYTE(COUNTER.HIGH,COUNTER.LOW);
  OUTBYTE(COUNTER.LOW,COUNTER.HIGH);
END;
STMT LINE NESTING  
SOURCE TEXT: IF11INTOUT.SRC  
PROCEDURE INITIALIZECHIP53(*INITIALIZE PPI WITH ALL PORTS = OUTPUT *)  
BEGIN  
  OUTPUT(PPICONTROLSPORT,PPICONTROLHOLD)  
END;(*INITIALIZECHIP53*)  
*INTERRUPT(SERVICEINTERRUPT)  
PROCEDURE SERVICEINTERRUPT;(*THIS PROCEDURE RUNS INT.SRV. 34 WHEN THAT INTERRUPT OCCURS *)  
BEGIN  
  IF N=10 THEN BEGIN  
    INPUT1=INPUT;  
  END;  
  IF ST1N>127 THEN ST1N=127;  
  IF ST1N<-128 THEN ST1N=-128;  
  OUTPUT=(ST1N+INTT)DIV(SAMT);  
  IF OUTPUT>127 THEN OUTPUT=127;  
  IF OUTPUT<-128 THEN OUTPUT=-128;  
  N=N+1;  
END;(*SERVICEINTERRUPT*)  
BEGIN(*MAIN*)  
BEGIN;(*SERVICEINTERRUPT*)  
DISABLEINTERRUPT;  
ST1N=0;  
OUTPUT=0;  
INPUT=2;  
N=0;  
OUTPUT(COUNTCONTROLSPORT,INITIALIZE:GO);  
INITIALIZECHIP53;  
OUTPUT(ICH1ADDR,ICH1DATA);  
OUTPUT(ICH2ADDR,ICH2DATA);  
OUTPUT(ICH3ADDR,ICH3DATA);  
INITIALIZECHIP53;  
END;(*SERVICEINTERRUPT*)  
WHILE SW=TRUE DO  
  GOTO 99  
END.
; STATEMENT 0 1

; STATEMENT 0 26

INITIALIZECIIP53
PROC NEAR

0000 55  PUSH BP
0001 BEEC  MOV BP,SP
0003 55  PUSH BP
0004 83C02  SUB SP,2H

; STATEMENT 0 30

0007 C746FC001  MOV COUNT,IP+1FH

; STATEMENT 0 31

000C 8A46FC  MOV AL,COUNT+1
000F E600  OUT 000H

; STATEMENT 0 32

0011 BA46FD  MOV AL,COUNT+1H
0014 E600  OUT 000H
0016 BEEE  MOV SP+6P
0018 50  POP BP
0019 C3  RET

INITIALIZECIIP53
ENDP

; STATEMENT 0 33

INITIALIZECIIP55
;PROC NEAR

001A 55  PUSH BP
0016 BEEC  MOV BP,SP
001D 55  PUSH BP

; STATEMENT 0 34

001E E600  MOV AL,80H
0020 E6CE  OUT 0CEH
0022 BEEE  MOV SP+6P
0024 50  POP BP
0025 C3  RET

INITIALIZECIIP55
ENDP

; STATEMENT 0 35

SERVICE_INTERRUPT
ASSEMBLY LISTING OF GENERATED OBJECT CODE

PROC NEAR  INTERRUPT PROC

0026 06  PUSH  EB
0027 1E  PUSH  DS
0028 59  PUSH  AX
0029 51  PUSH  CX
002A 52  PUSH  DX
002B 53  PUSH  SI
002C 56  PUSH  DI
002D 57  PUSH  DI
002E 600000  MOV  AX,SEG DATA
0031 80B8  MOV  DS:AX
0033 55  PUSH  BP
0034 B8EC  MOV  BP,SP
0036 55  PUSH  BP

; STATEMENT # 36
0037 B3E0000A  CMP  N:BAH
003C 750E  JNE  70

; STATEMENT # 37
003E A10200  MOV  AX,INPUT
0041 F700  NEG  AX
0043 A30200  MOV  INPUT,AX

; STATEMENT # 38
0046 C706000000  MOV  N:OH
?0:

; STATEMENT # 40
004C A10200  MOV  AX,INPUT
004F 03060000  ADD  AX,STIN
0053 A30000  MOV  STIN,AX

; STATEMENT # 41
0056 B3E00007F  CMP  STIN,7FH
005B 7E06  JLE  71

; STATEMENT # 42
005D C70600007F00  MOV  STIN,7FH
?1:

; STATEMENT # 43
0063 B3E000080  CMP  STIN,0FF80H
0068 7006  JNL  72

; STATEMENT # 44
006A C706000080FF  MOV  STIN,0FF80H
?2:
; STATEMENT 45
0070 B80200  MOV  AX+2H
0073 FF7E0000  IMUL  STEN
0077 091000  MOV  CX+1H
007A FF9  IDIV  CX
007C A30400  MOV  OUTPUT+AX

; STATEMENT 46
007F 03E090007F  CMP  OUTPUT+7FH
0084 7E06  JLE  73

; STATEMENT 47
0086 C70604007FO0  MOV  OUTPUT+7FH

; STATEMENT 48
008C 03E0400800  CMP  OUTPUT+OFF80H
0091 7006  JAL  74

; STATEMENT 49
0093 C706040080FF  MOV  OUTPUT+OFF80H

; STATEMENT 50
0099 A10660  MOV  AX+N
009C 40  INC  AX
009D A30688  MOV  AX+AX

; STATEMENT 51
00A0 A10900  MOV  AX+OUTPUT
00A3 E6C8  OUT  0CH
00A5 BEES  MOV  SP+BP
00A7 5D  POP  BP
00A8 5F  POP  DI
00A9 5E  POP  SI
00AA 58  POP  BX
00AB 5A  POP  DX
00AC 59  POP  CX
00AD 58  POP  AX
00AE 1F  POP  DS
00AF 07  POP  ES
00B0 CF  INT

SERVICEINTERRUPT
ENDP

; INOUT
INTOUT  MOV  BP,SP
00B3 40  DLC  BP
ASSEMBLY LISTING OF GENERATED OBJECT CODE

0004 4D       DCC          RP
0005 55       PUSH        B
0006 9A00000000   CALL INITFP
0008 9A00000000   CALL T04001
;

; STATEMENT # 52

00C0 BB2200   MOV AX,22H
00C1 50       PUSH AX
00C2 2EB0E02600 LEA CX,C$SERVICE_INTERRUPT
00C3 0E       PUSH CS
00C4 51       PUSH CX
00C6 9A00000000 CALL T04302
;

; STATEMENT # 53

00D0 FA       CLI
;

; STATEMENT # 54

00D1 C70600000000 MOV STIN,0H
;

; STATEMENT # 55

00D7 C70604000000 MOV OUTPUT,0H
;

; STATEMENT # 56

00D9 C70602000200 MOV INPUT,2H
;

; STATEMENT # 57

00E2 C70606000000 MOV N+0H
;

; STATEMENT # 58

00E8 E036      MOV AL,34H
00EB E606      JUT 0066H
;

; STATEMENT # 59

00ED EB10FF    CALL INITIALIZECHIP53
;

; STATEMENT # 60

00F0 D017      MOV AL,1FH
00F2 E6C0      OUT 0C0H
;

; STATEMENT # 61

00F4 E020      MOV AL,20H
00F6 E6C2      OUT 0C2H
;

; STATEMENT # 62

00F8 D01F      MOV AL,1FH
ASSEMBLY LISTING OF GENERATED OBJECT CODE

00FA E6C2 OUT 0C2H
; STATEMENT # 63
00FC E0FB MOV AL, OFB
00FE E6C2 OUT 0C2H
; STATEMENT # 64
0100 E017FF CALL INITIALIZECHIP53
; STATEMENT # 65
0103 FB STI
; STATEMENT # 66
009F: ?0:
0104 E001 MOV AL, 1H
0106 3C01 CMP AL, 1H
0108 7504 JNE ?1
; STATEMENT # 67
010A E0F8 JMP 0099
010C E1F8 JMP 70
010E 9A00000000 CALL TGK-999
INTOUT ENOP

SUMMARY INFORMATION:

PROCEDURE OFFSET CODE SIZE DATA SIZE STACK SIZE
INITIALIZECHIP53 0000H 0016H 25D 0006H 6D
INITIALIZECHIP55 0016H 000CH 12D 0006H 6D
SERVICEINTERUPT 0026H 0006H 13D 0026H 3BD
INTOUT 0061H 0062H 9BD 0006H 8D 000AH 10D
-CONST IN CODE- 0086H 8D
TOTAL 0113H 275D 0006H 8D 0070H 112D

69 LINES READ.
0 ERRORS DETECTED.

DICTIONARY SUMMARY:

120KB MEMORY AVAILABLE.
64KB MEMORY USED (5%).
6KB DISK SPACE USED.
2KB OUT OF 16KB STATIC SPACE USED (12%).