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

Philips International Institute
No..... August 1984.

P.I.I. Report No....

Author: P.A. Silberfich

Subject: Design and implementation of a 16-bit digital servocontroller. P.I.D. Algorithm.

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

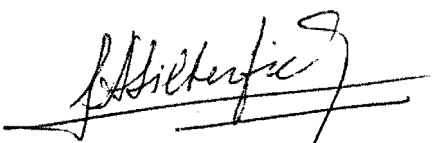
Chief/Coach: Ir. C.J. Heuvelman tel. 474561
P.I.I. Supervisor: Dr.Ir. P.M. Mul tel. 7-55893

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

Contents list — Pages

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.

2. 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 ε (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, there after, generating an error signal.

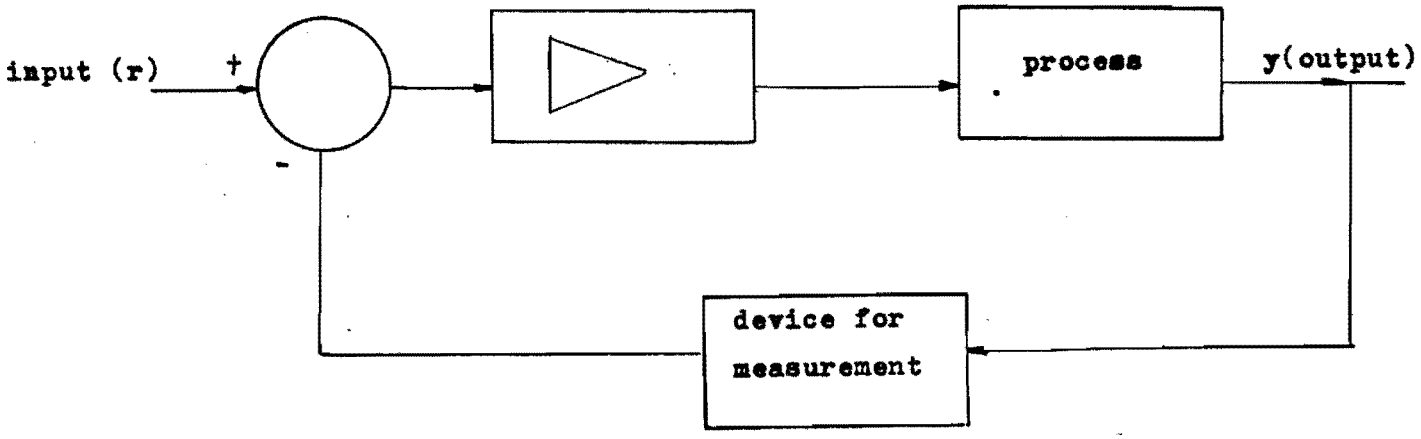


Fig. 1

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)

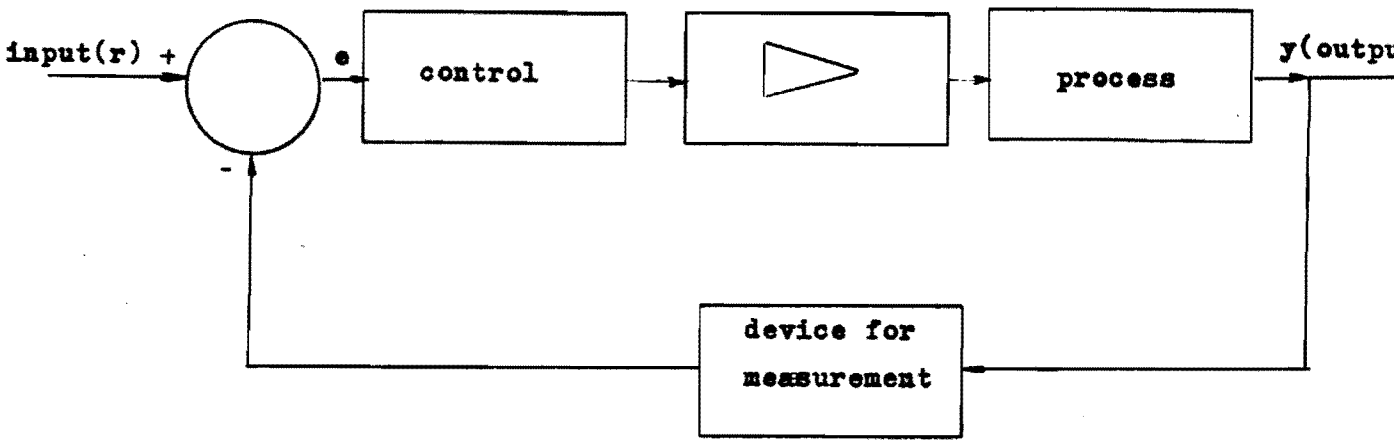


Fig. 2

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.q. 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)

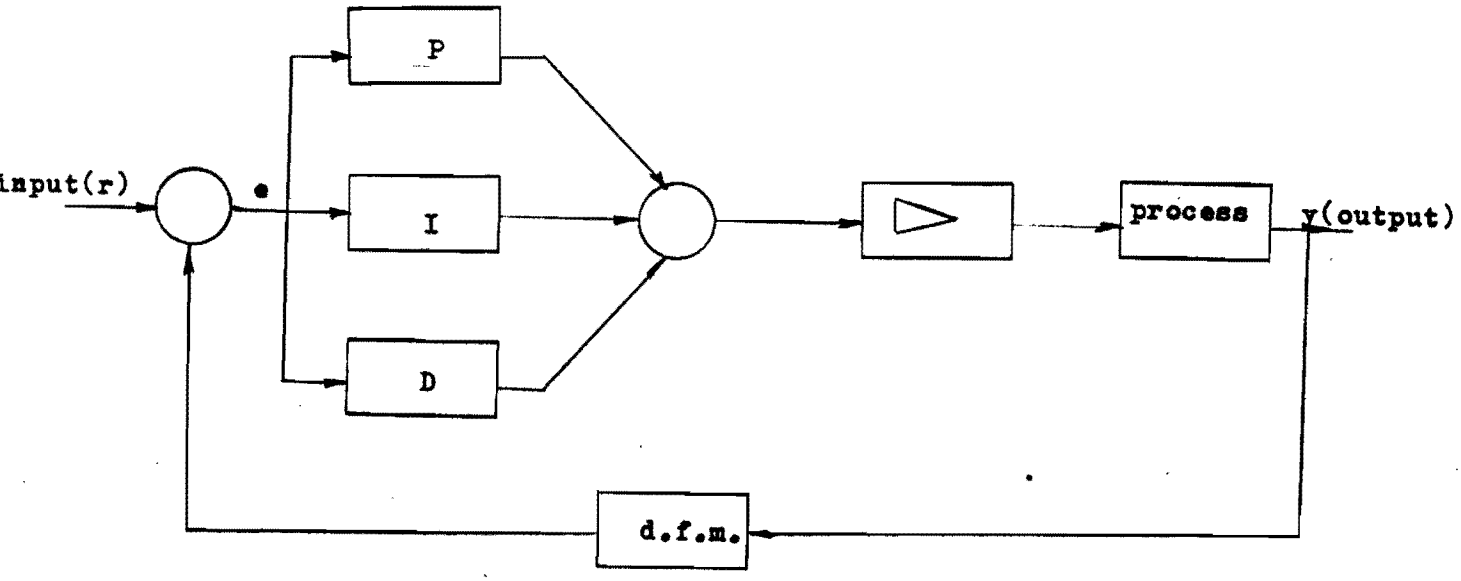


Fig. 3

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)

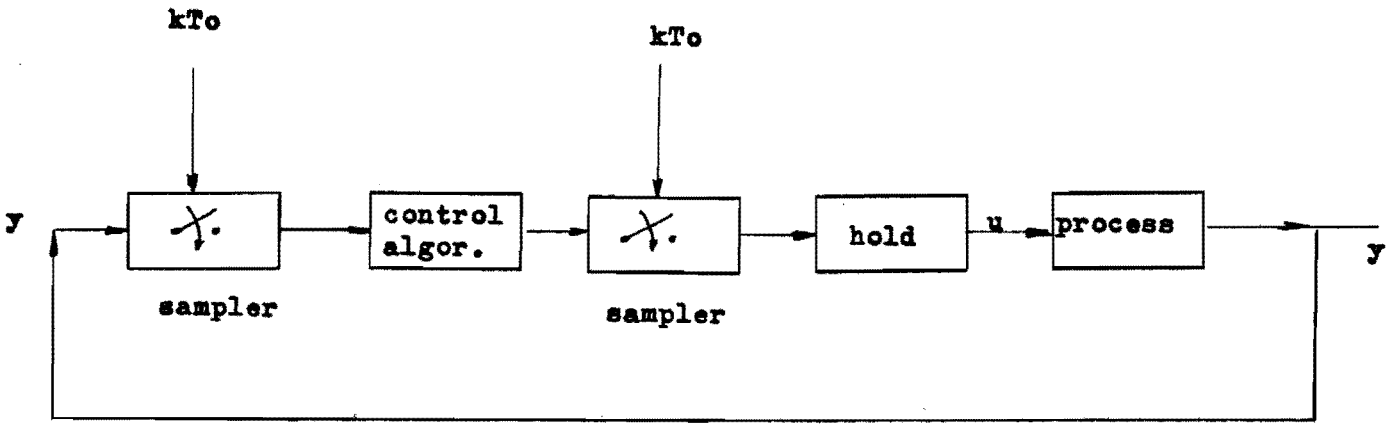


Fig. 4

Actually, the manipulated variable **u** is calculated by a control algorithm using the control variable $\underline{y}$ and the reference value $\underline{w}$ as inputs ($\underline{w}$ 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)

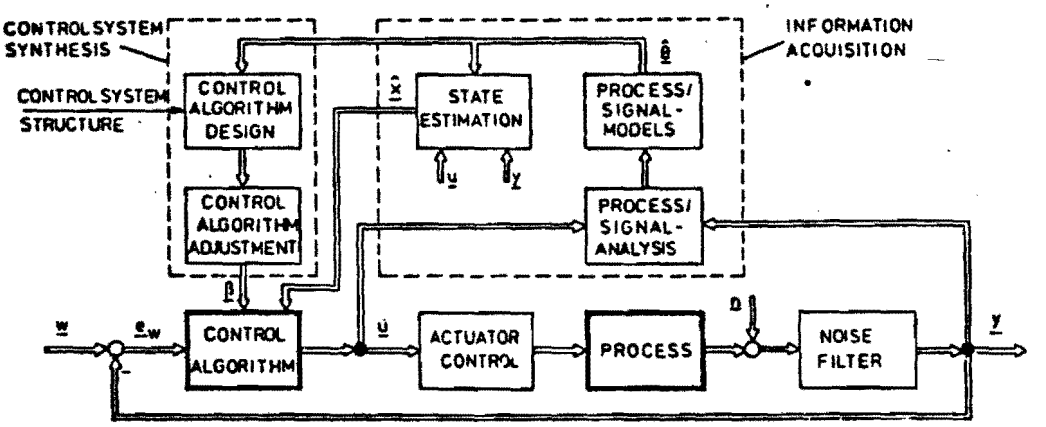
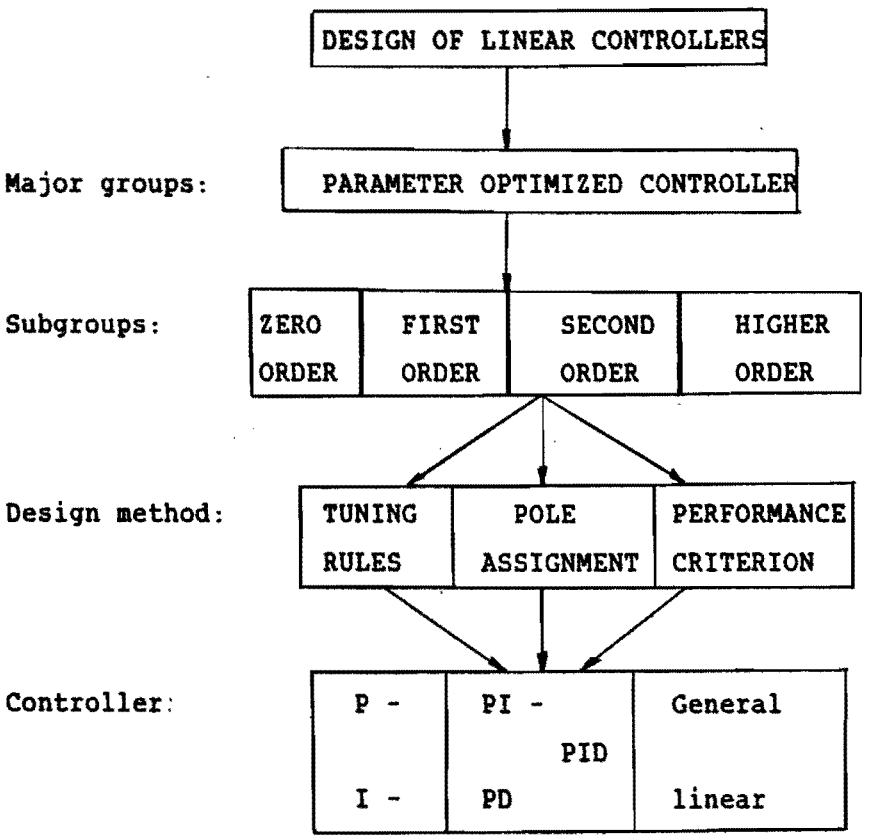


Fig. 5

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



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

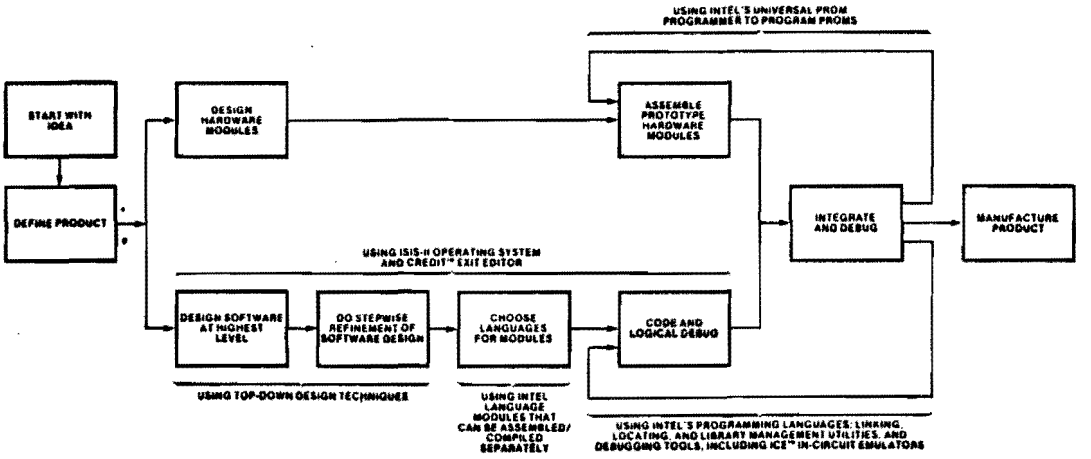
Although there's a difference between using analog and digital controllers, we can reduce this gap using fine quantizations and sampling time.

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 formuly 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).


USING INTEL'S UNIVERSAL PROM PROGRAMMER TO PROGRAM PROMS
START WITH IDEA
DEFINE PRODUCT
DESIGN HARDWARE MODULES
ASSEMBLE PROTOTYPE HARDWARE MODULES
INTEGRATE AND DEBUG
MANUFACTURE PRODUCT
USING ISIS-II OPERATING SYSTEM AND CREDIT™ EXIT EDITOR
DESIGN SOFTWARE AT HIGHEST LEVEL
DO STEPWISE REFINEMENT OF SOFTWARE DESIGN
CHOOSE LANGUAGES FOR MODULES
CODE AND LOGICAL DEBUG
USING TOP-DOWN DESIGN TECHNIQUES
USING INTEL LANGUAGE MODULES THAT CAN BE ASSEMBLED/ COMPILED SEPARATELY
USING INTEL'S PROGRAMMING LANGUAGES, LINKING, LOCATING, AND LIBRARY MANAGEMENT UTILITIES, AND DEBUGGING TOOLS, INCLUDING ICE™ IN-CIRCUIT EMULATORS

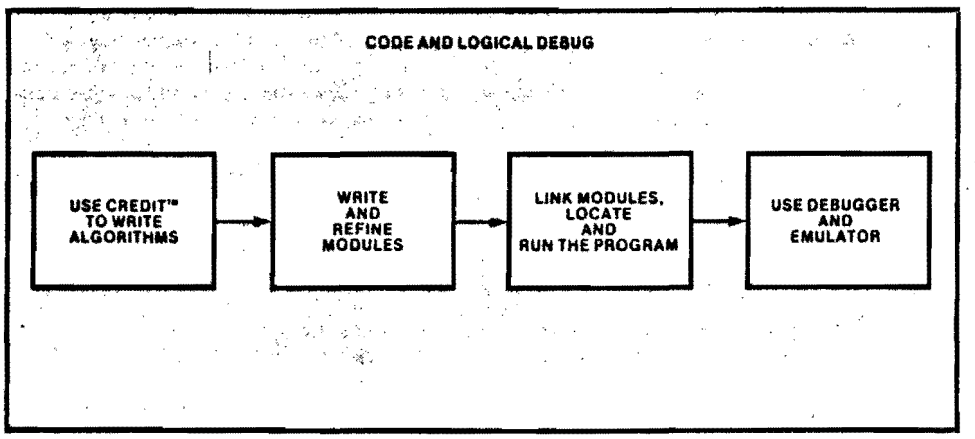
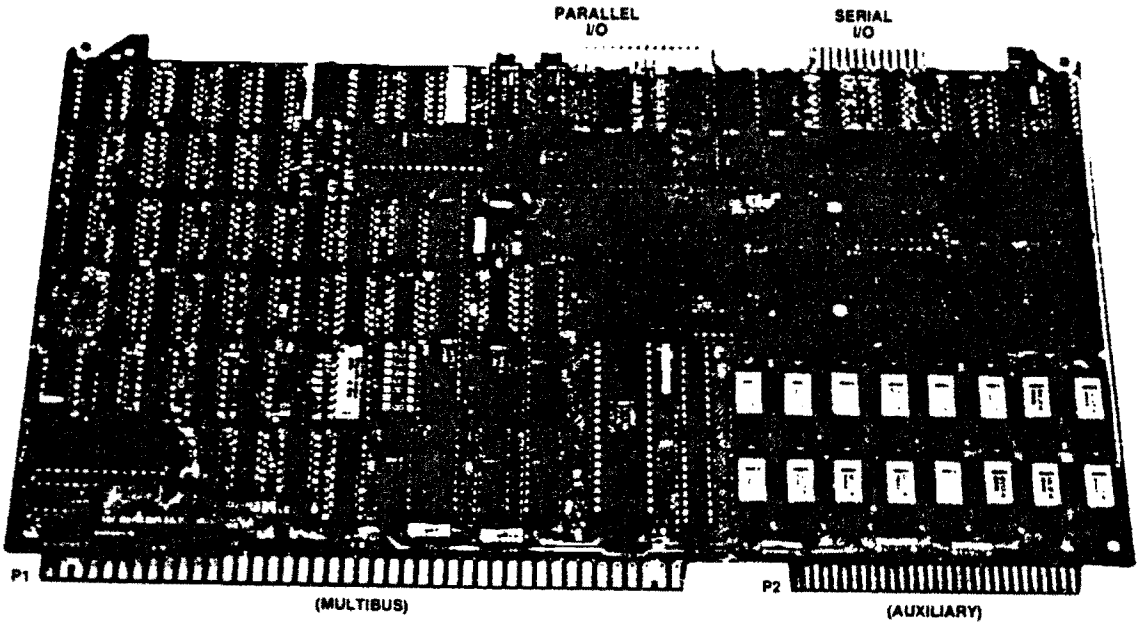

CODE AND LOGICAL DEBUG
USE CREDIT™ TO WRITE ALGORITHMS
WRITE AND REFINE MODULES
LINK MODULES, LOCATE AND RUN THE PROGRAM
USE DEBUGGER AND EMULATOR

Fig. 6

4.2.2 ISBC 86/12

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)



iSBC 86/12 Single Board Computer

Fig. 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 8086/8888 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).

Data Type	Bits	Significant Digits (Decimal)	Approximate Range (Decimal)
Word integer	16	4	$-32{,}768 \leq X \leq +32{,}767$
Short integer	32	9	$-2\times10^{9} \leq X \leq +2\times10^{9}$
Long integer	64	18	$-9\times10^{18} \leq X \leq +9\times10^{18}$
Packed decimal	80	18	$-99\ldots99 \leq X \leq +99\ldots99$ (18 digits)
Short real*	32	6-7	$8.43\times10^{-37} \leq \lvert X\rvert \leq 3.37\times10^{38}$
Long real*	64	15-16	$4.19\times10^{-307} \leq \lvert X\rvert \leq 1.67\times10^{308}$
Temporary real	80	19	$3.4\times10^{-4932} \leq \lvert X\rvert \leq 1.2\times10^{4932}$

Fig. 8

We can see the internal structure of the chip in the figs. 9 and 10.

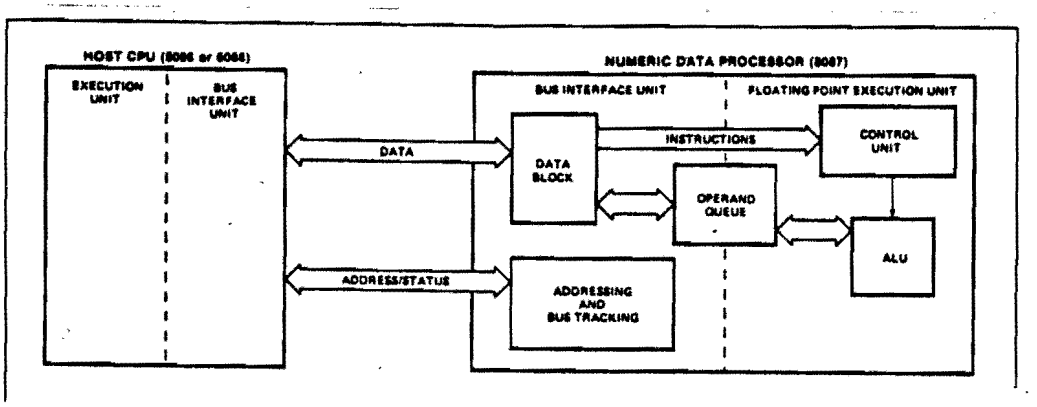


Fig. 9

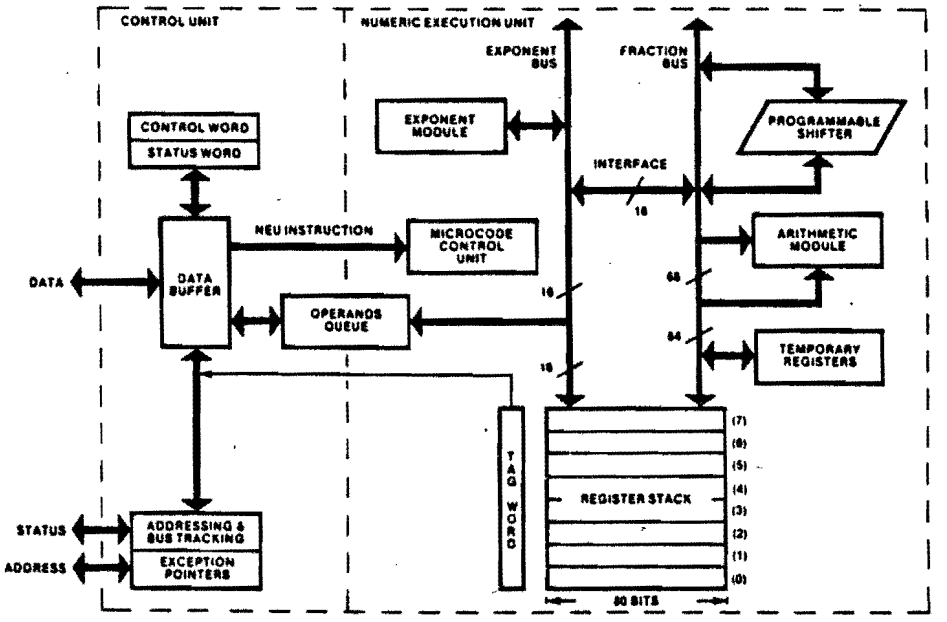


Fig. 10

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).

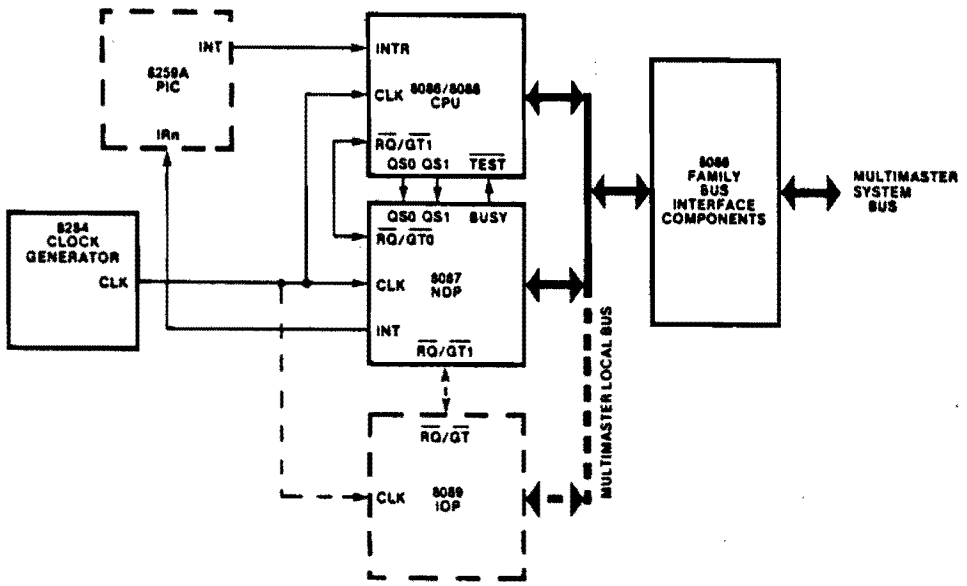


Fig. 11

To obtain the interconection 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:

AX: accumulator
BX: base
CX: count
DX: data

} 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-DØ 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.

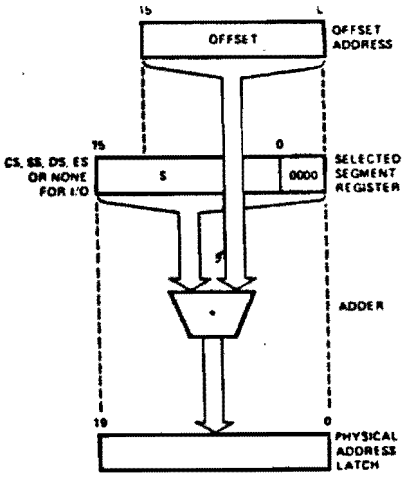
Other instructions assume that a count has been loaded into either CL(8 bits) or CX(16 bits), and that's what we use to generate the square wave.

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).



Generating a 20-bit memory address from a segment address and a 16-bit offset or effective address.

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.

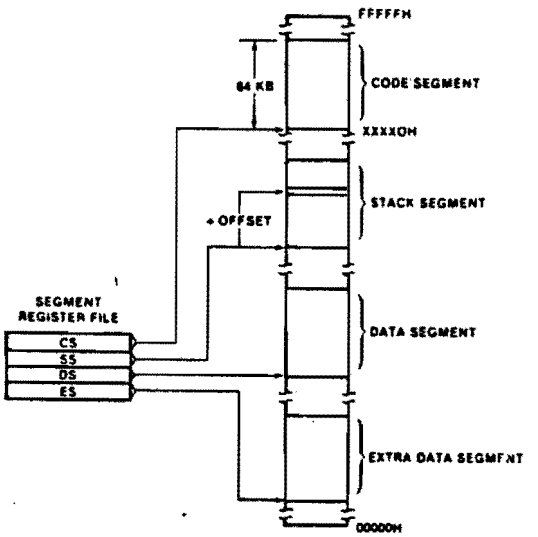
The SP (stack pointer) is used to provide part of a 20-bit address that the 8086 uses when any information is placed on, or taken off of the stack. It's used in conjunction with the stack segment register (SS).

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 execu ted 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.

.. Different Addressing Modes and the Registers Used

BASE INDEX	BX + SI + Displacement + Segment BX + DI + Displacement + Segment BP + SI + Displacement + Segment BP + DI + Displacement + Segment
INDEX	SI + Displacement + Segment DI + Displacement + Segment
BASE	BP + Displacement + Segment BX + Displacement + Segment
BASE INDEX (NO DISPLACEMENT)	BX + SI + Segment BX + DI + Segment BP + SI + Segment BP + DI + Segment
INDIRECT	SI + Segment DI + Segment BX + Segment BP + Segment
RELATIVE	Displacement + Instruction Pointer (IP)
DIRECT	Address + Segment

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).

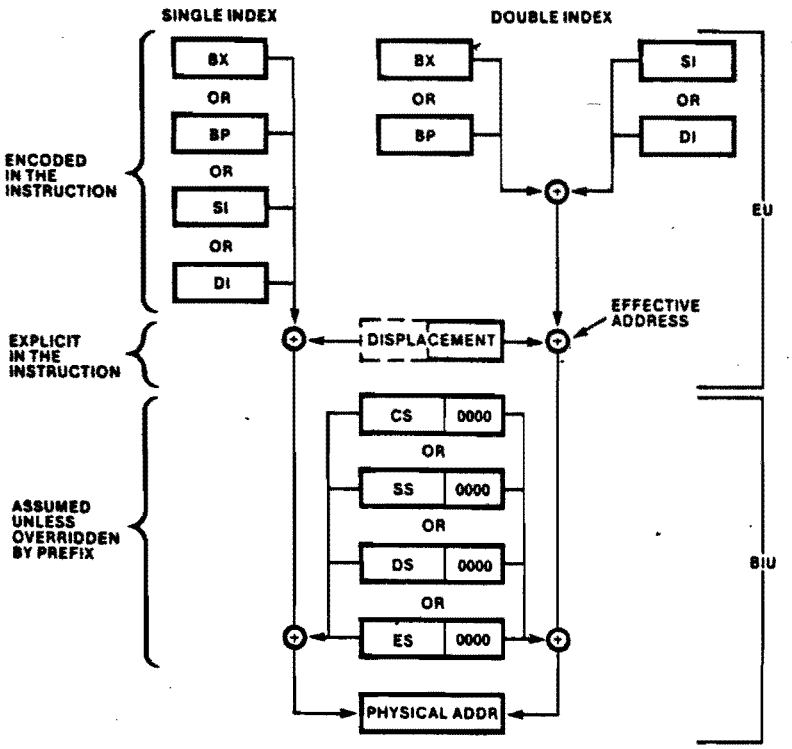
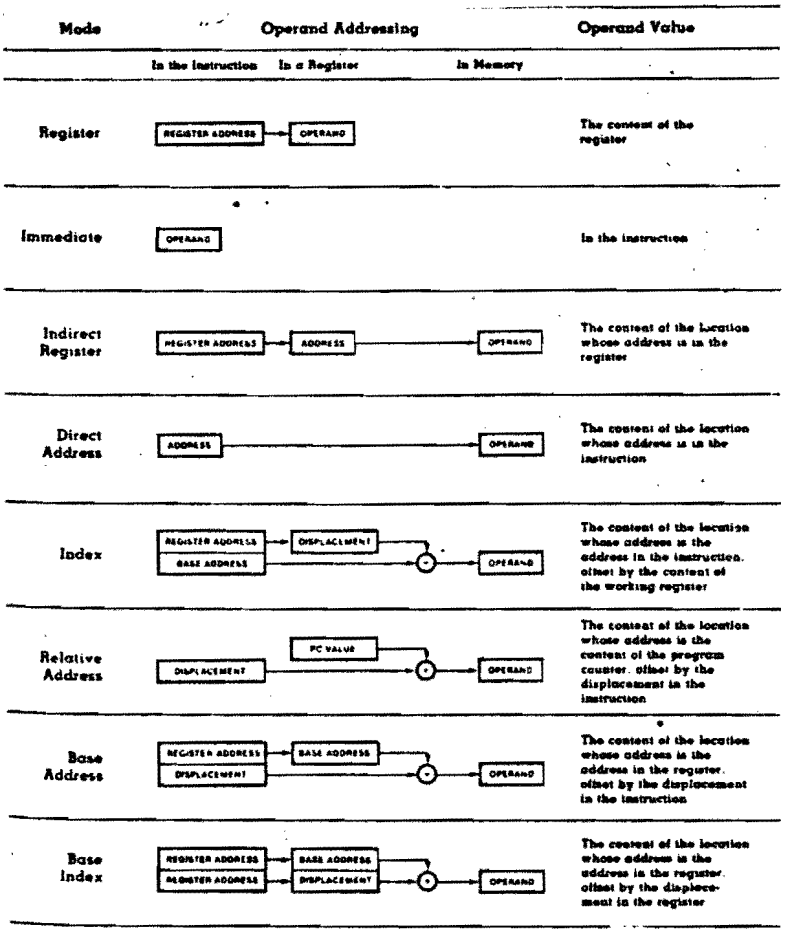

SINGLE INDEX
DOUBLE INDEX
BX
OR
BP
OR
SI
OR
DI
BX
OR
BP
SI
OR
DI
ENCODED IN THE INSTRUCTION
EXPLICIT IN THE INSTRUCTION
DISPLACEMENT
EFFECTIVE ADDRESS
EU
CS
0000
OR
SS
0000
OR
DS
0000
OR
ES
0000
ASSUMED UNLESS OVERRIDDEN BY PREFIX
PHYSICAL ADDR
BIU

Fig. 15



A summary of 8086 addressing modes.

Fig. 16

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 requences)

Applications: move and compare strings, scan for a value; everything possible to and from the acc.

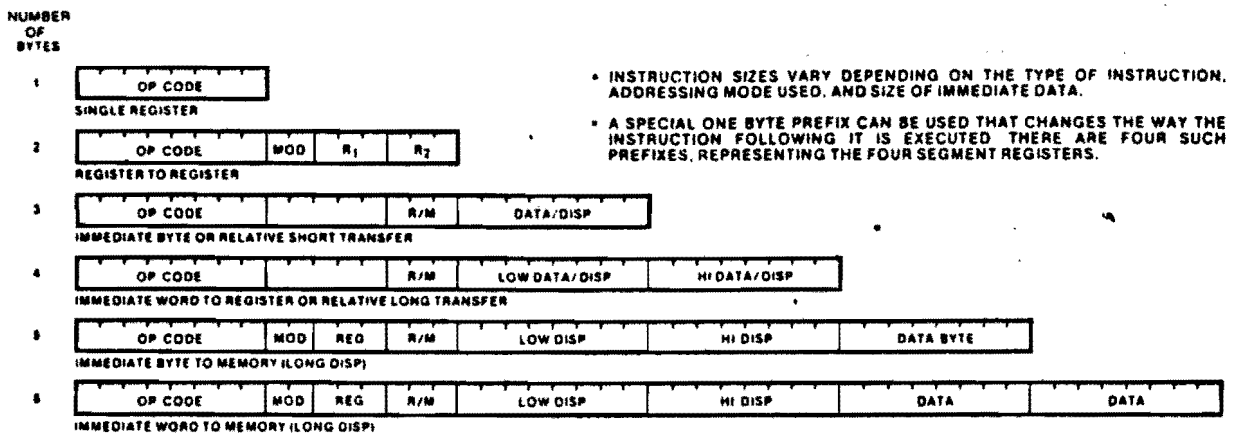
e) Program transfer instructions

- 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.



. The instruction format for the 8086.

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-DØ) 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 approximatively 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)

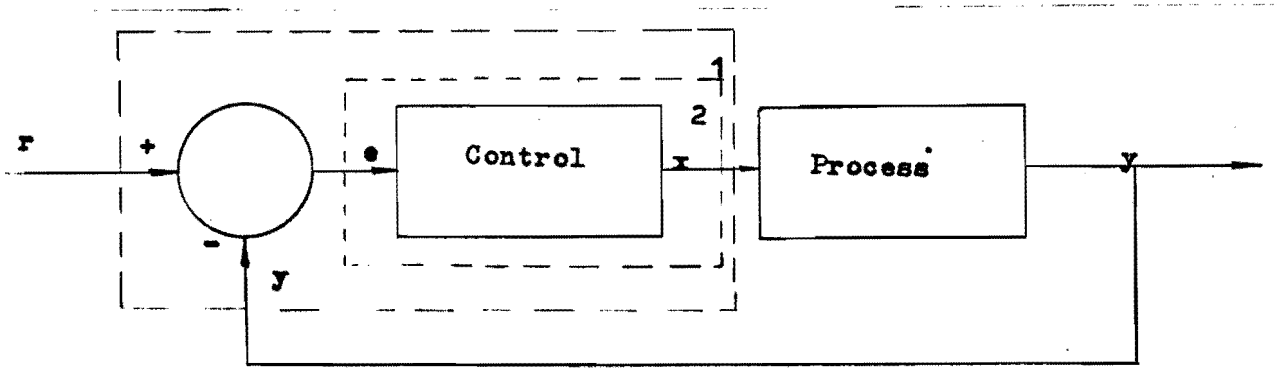


Fig. 18

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 e signal into a signal able to go to the process, obtaining a wanted signal y in the output.

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

Depending on that, after obtaining e = r - y, we obtain the correction signal x = 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: $x(T) = K * e(T)$

- Differential Action: $x(T) = T_d * (\frac{de}{dt})_{t=T}$

- 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:

$$x(T) = K * e(T) + \frac{1}{T_i} * \int_{t=0}^{T} e(t)dt + T_d * (\frac{de}{dt})_{t=T}$$

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:

$$x(nT) = K * e(nT) + \frac{1}{T_i} * \sum_{k=1}^{n} T_s * \frac{e(kT) + e(kT-T)}{2} +$$

$$+ T_d * \frac{e(nT) - e(nT-T)}{T_s}$$

T_s = sampling time

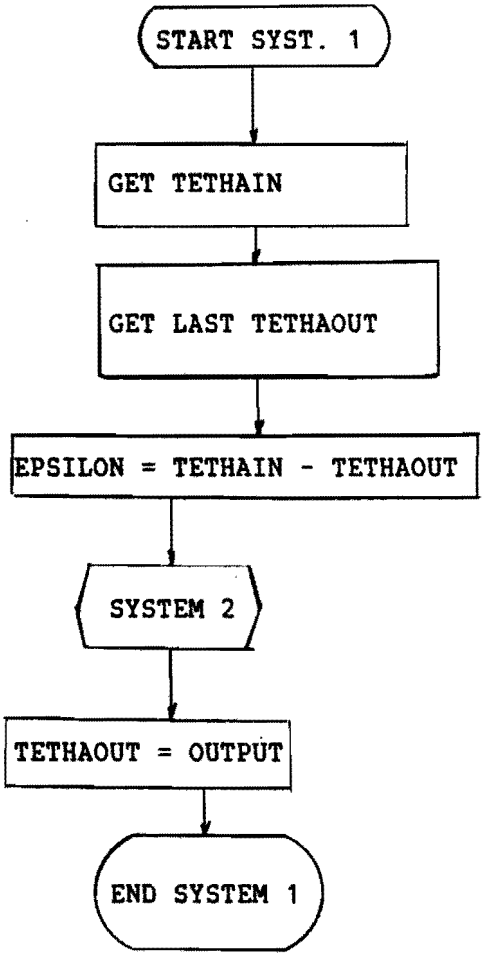
b) If we apply rectangular integration, we have:

$$x(nT) = K * e(nT) + \frac{Ts}{T_i} \sum_{k=1}^{n} e(kT-T) + T_d * \frac{e(nT) - e(nT-T)}{T_s}$$

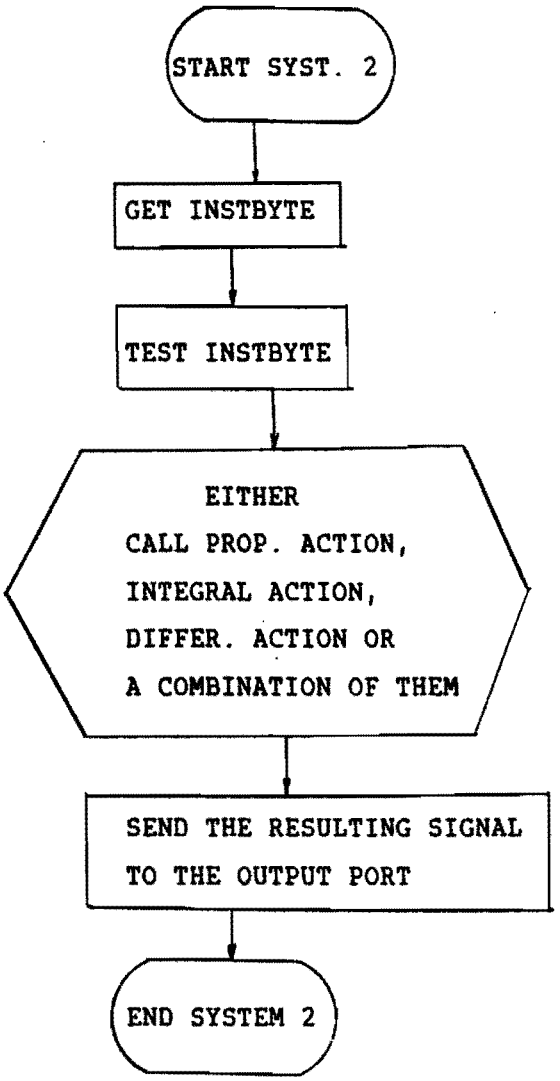
c) If we work these expressions out and calculate the difference between two succesive samples, we arrive to recursive control algorithms, more suitable for programming on computers.

5.2 Developing Flow-charts

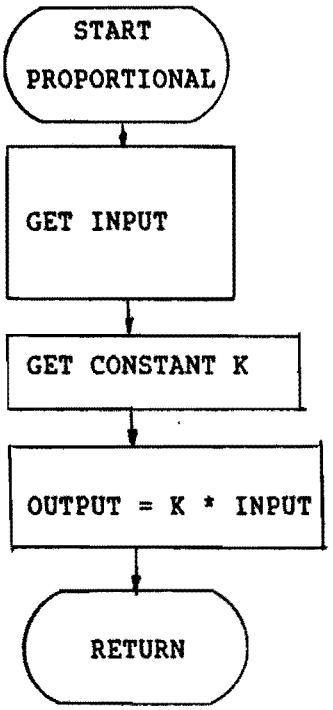
5.2.1 Flow chart of system 1



5.2.2 Flow chart of system 2

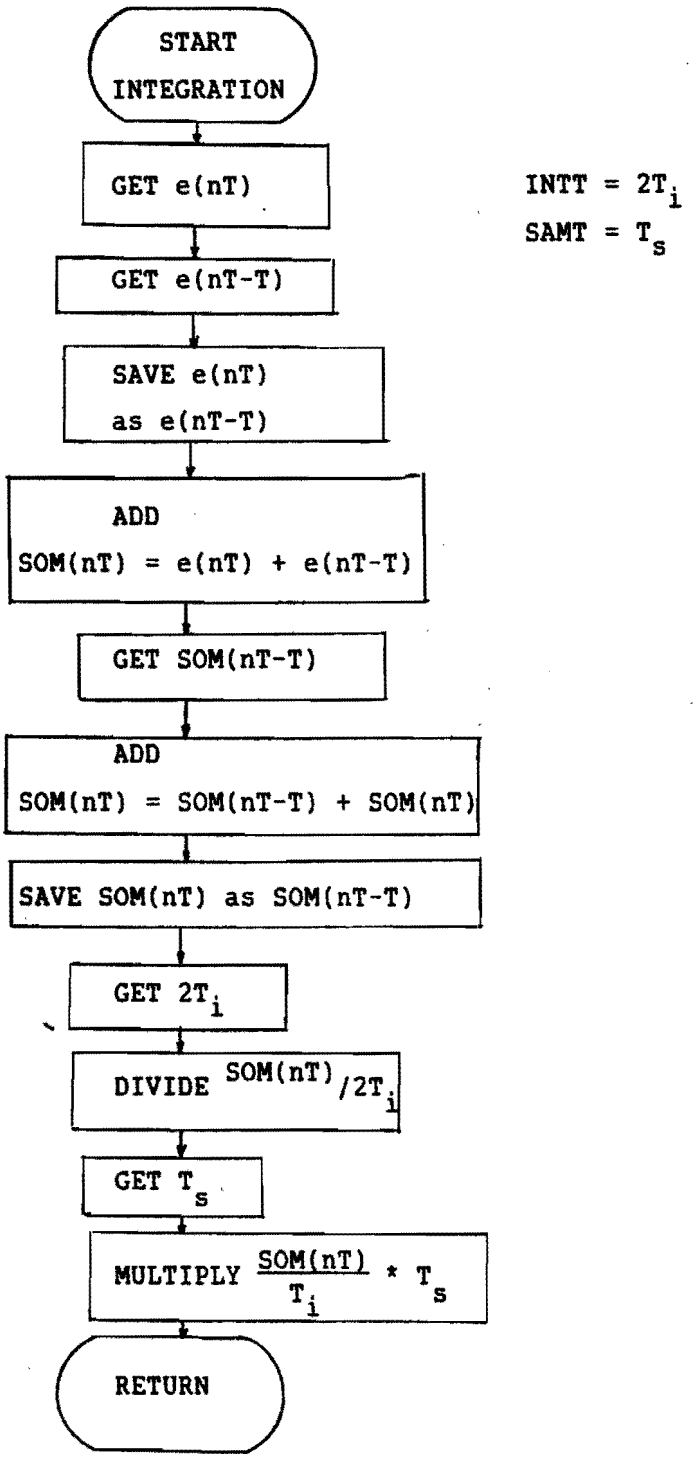


5.2.3 Flow chart of proportional action



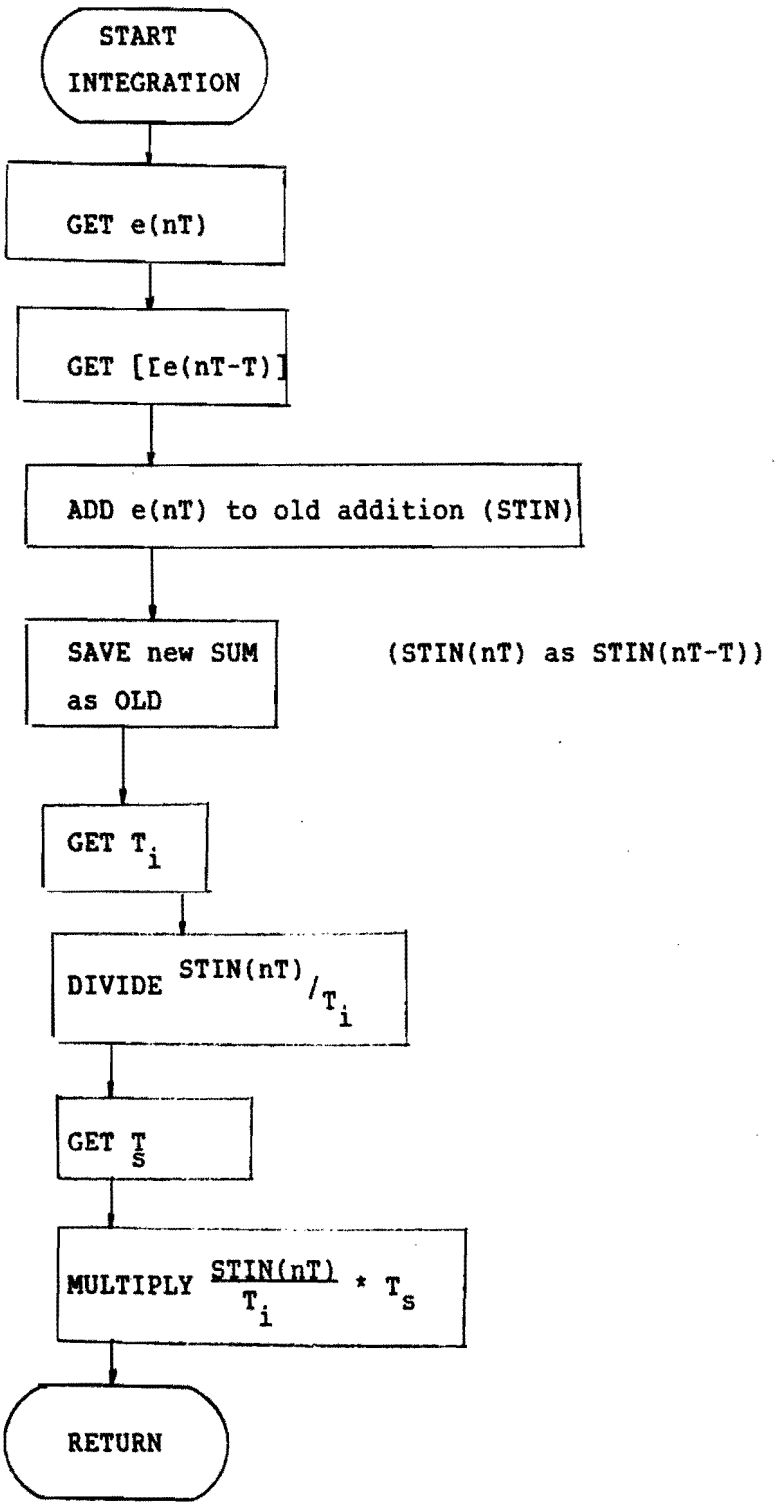
5.2.4 Flow chart of integral action

(Trapezoidal method)

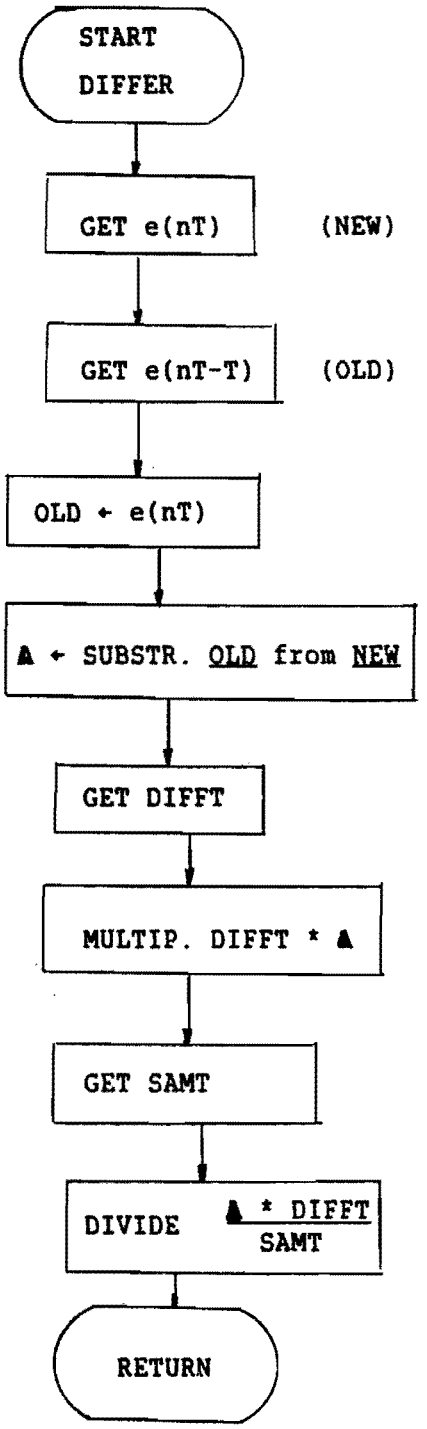


5.2.5 Flow chart of integral action

(Rectangular method) (adopted)



5.2.6 Flow chart of differential action



5.3 Integration Algorithms

1) Trapezoidal integration

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

```
BEGIN: (new value in AX)
            MOV  BX, STIN
            MOV  STIN, AX
            %    ADDIT
            MOV  BX, SOM
            %    ADDIT
            MOV  SOM, AX
            MOV  BX, INTT
            %    DIVIDE
            MOV  BX, SAMT
            %    MULTI          (90 USEG)
```

2) Rectangular integration

$$x(nT) = \frac{Ts}{T_i} * \sum_{k=1}^{n} e(kT-T)$$

```
BEGIN: (new value in AX)
            MOV  BX, STIN
            %    ADDIT
            MOV  STIN, AX
            MOV  BX, INTT
            %    DIVIDE
            MOV  BX, SAMT
            %    MULTI          (80 USEG)
                                (SEE APP.M)
```

2) Rectangular integration

(another way)

$$x(nT) = \frac{Ts}{T_i} * \sum_{k=1}^{n} e(kT-T)$$

```
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

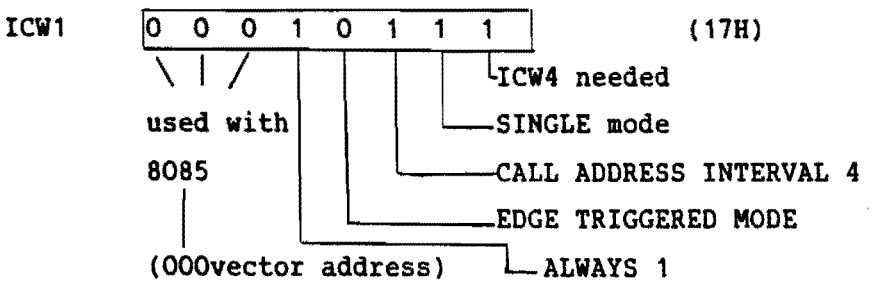
5.4.1 Explanations and use

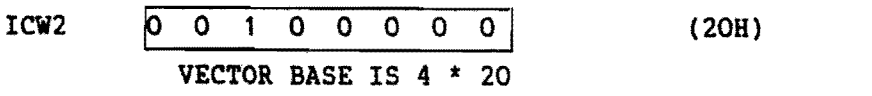
The PIDOUT program works following the flow diagram of system 1.
It generates the testing square wave, and depending on the INSTBYTE entered, it can execute different routines. To do it, euter:

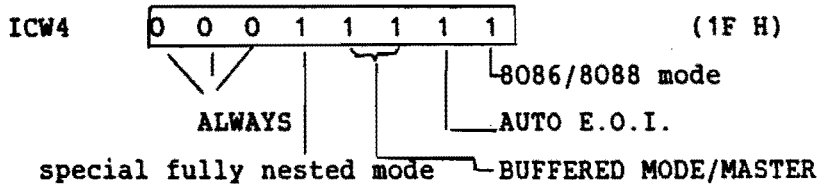
0: output equals to 0
1: D effect
2: I effect
3: I+D effect
4: P effect
5: P+D effect
6: P+I effect
7: P+I+D effect

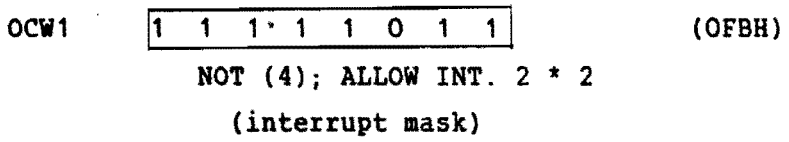
The input signal changes to its complement each 10 times the counter has reached 0.
Each time the counter reaches 0, an interrupt is originated and one service interrupt routine cycle is performed.
To do so, first of all, we should initialize the different chips, not allowing interrupts in meanwhile.

Then, we send control bytes to initialize the PIC

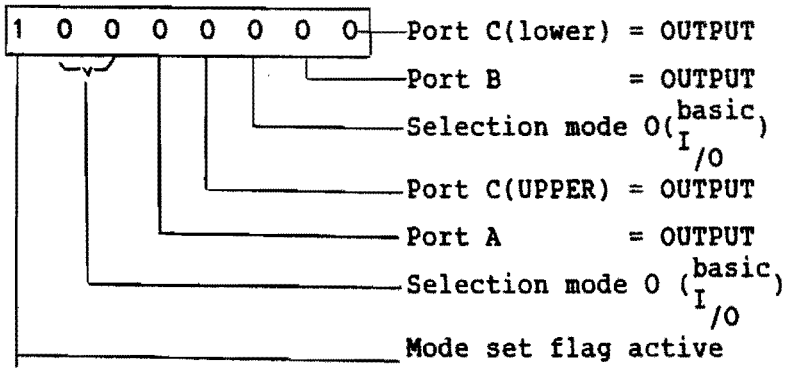








The last thing to do, after initializing the timers is to initialize the PPI with all the PORTS = OUTPUT
To do that, we send to the 8255: 080F

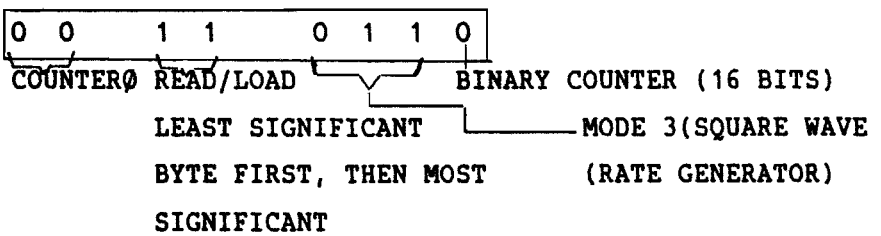


5.4.2 Testing signal - Software square wave

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  ODOH, AL  ;  LSB TIME
                  (492D CYCLES ≈ 400 useg)
MOV  AL, 01H
OUT  ODOH, 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 useg)

Rectangular integration with 8085:	1500	(assembly)
Rectangular integration with 8086:	80	(assembly)
Trapezoidal integration with 8086:	90	(assembly)
Rectangular integration with 8086:	180	(PASCAL)

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
- trascendental instructions needed
- other instructions needed.

8. BIBLIOGRAPHY

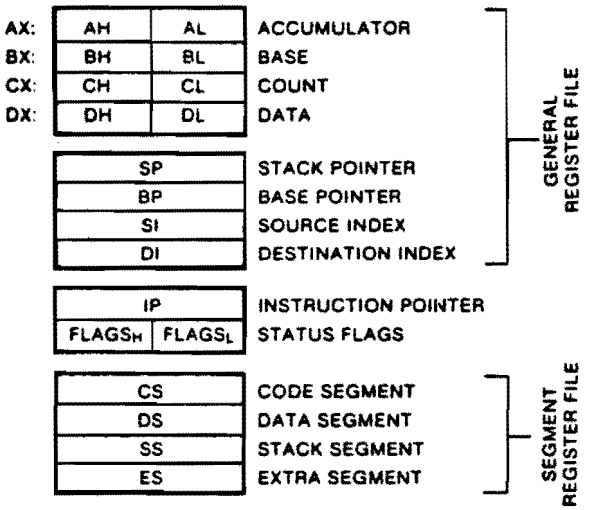
iAPX 86/88, 186/188 user's Manual.
Programmer's Reference 1983/INTEL iAPX 86,88 Family utilities user's guide /1982/ INTEL
ASM 86 Macro Assembler Operating Instructions
ASM 86 Language Reference Manual.

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

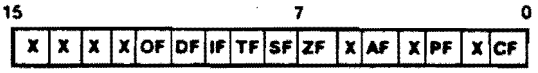
Intellec Series III-Microcomputer Development System-Console Operating Instructions
Intellec Series III-Microcomputer Development System-Programmer's Reference Manual
PASCAL-86 User's guide/1982/Intel
INTEL 8087-NUMERIC DATA COPROCESSOR
INTEL Component Data Catalog 1980
INTEL FAIR-Applications Handbook
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



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



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)
TF: TRAP — SINGLE STEP FLAG
— 8086 FLAGS

OPERAND SUMMARY

"reg" field Bit Assignments:

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

SECOND INSTRUCTION BYTE SUMMARY

mod	xxx	r/m

mod	Displacement
00	DISP = 0*, disp-low and disp-high are absent
01	DISP = disp-low sign-extended to 16-bits, disp-high is absent
10	DISP = disp-high: disp-low
11	r/m is treated as a "reg" field

r/m	Operand Address
000	(BX) + (SI) + DISP
001	(BX) + (DI) + DISP
010	(BP) + (SI) + DISP
011	(BP) + (DI) + DISP
100	(SI) + DISP
101	(DI) + DISP
110	(BP) + DISP*
111	(BX) + DISP

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 d w	mod reg r/m

Timing (clocks): register to register 2
memory to register 8+EA
register to memory 9+EA

Immediate to register/memory

1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w=1

Timing: 10+EA clocks

Immediate to register

1 0 1 1 w reg	data	data if w=1

Timing: 4 clocks

Memory to accumulator

1 0 1 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 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 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 1	mod 1 1 0 r/m

Timing (clocks): register 10
memory 16+EA

Register

0 1 0 1 0 reg

Timing: 10 clocks

Segment register

0 0 0 reg 1 1 0

Timing: 10 clocks

POP = Pop
Register/memory

1 0 0 0 1 1 1 1	mod 0 0 0 r/m

Timing (clocks): register 8
memory 17+EA

Register

0 1 0 1 1 reg

Timing: 8 clocks

Segment register

0 0 0 reg 1 1 1

Timing: 8 clocks

XCHG = Exchange
Register/memory with register

1 0 0 0 0 1 1 w	mod reg r/m

Timing (clocks): register with register 4
memory with register 17+EA

Register with accumulator

1 0 0 1 0 reg

Timing: 3 clocks

IN = Input to AL/AX from
Fixed port

1 1 1 0 0 1 0 w	port

Timing: 10 clocks

Variable port (DX)

1 1 1 0 1 1 0 w

Timing: 8 clocks

OUT = Output from AL/AX to
Fixed port

1 1 1 0 0 1 1 w	port

Timing: 10 clocks

Variable port (DX)

1 1 1 0 1 1 1 w

Timing: 8 clocks

XLAT = Translate byte to AL

1 1 0 1 0 1 1 1

Timing: 11 clocks

LEA = Load EA to register

1 0 0 0 1 1 0 1	mod reg r/m

Timing: 2+EA clocks

LDS = Load pointer to DS

1 1 0 0 0 1 0 1	mod reg r/m

Timing: 16+EA clocks

LES = Load pointer to ES

1 1 0 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 1

Timing: 4 clocks

SAHF = Store AH into flags

1 0 0 1 1 1 1 0

Timing: 4 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 d w	mod reg 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 0 s w	mod 0 0 0 r/m	data	data if s:w=01

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

Immediate from register/memory

1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s:w=01

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

Immediate from accumulator

0 0 1 0 1 1 0 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 d w	mod reg 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 0 s w	mod 0 1 1 r/m	data	data if s w=01

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

Immediate from accumulator

0 0 0 1 1 1 0 w	data	data if w=1

Timing: 4 clocks

DEC = Decrement
Register/memory

1 1 1 1 1 1 1 w	mod 0 0 1 r/m

Timing (clocks): register 2
memory 15+EA

Register

0 1 0 0 1 reg

Timing: 2 clocks

NEG = Change sign

1 1 1 1 0 1 1 w	mod 0 1 1 r/m

Timing (clocks): register 3
memory 16+EA

CMP = Compare
Register/memory and register

0 0 1 1 1 0 d w	mod reg r/m

Timing (clocks): register with register 3
memory with register 9+EA
register with memory 9+EA

ADC = Add with carry
Reg./memory with register to either

0 0 0 1 0 0 d w	mod reg 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 0 s w	mod 0 1 0 r/m	data	data if s:w=01

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

Immediate to accumulator

0 0 0 1 0 1 0 w	data	data if w=1

Timing: 4 clocks

INC = Increment
Register/memory

1 1 1 1 1 1 1 w	mod 0 0 0 r/m

Timing (clocks): register 2
memory 15+EA

Register

0 1 0 0 0 reg

Timing: 2 clocks

AAA = ASCII adjust for add

0 0 1 1 0 1 1 1

Timing: 4 clocks

DAA = Decimal adjust for add

0 0 1 0 0 1 1 1

Timing: 4 clocks

SUB = Subtract
Reg./memory and register to either

0 0 1 0 1 0 d w	mod reg r/m

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

Immediate with register/memory

1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s:w=01

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

Immediate with accumulator

0 0 1 1 1 1 0 w	data	data if w=1

Timing: 4 clocks

AAS = ASCII adjust for subtract

0 0 1 1 1 1 1 1

Timing: 4 clocks

DAS = Decimal adjust for subtract

0 0 1 0 1 1 1 1

Timing: 4 clocks

MUL = Multiply (unsigned)

1 1 1 1 0 1 1 w	mod 1 0 0 r/m

Timing (clocks): 8-bit 71+EA
16-bit 124+EA

IMUL = Integer multiply (signed)

1 1 1 1 0 1 1 w	mod 1 0 1 r/m

Timing (clocks): 8-bit 90+EA
16-bit 144+EA

AAM = ASCII adjust for multiply

1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0

Timing: 83 clocks

DIV = Divide (unsigned)

1 1 1 1 0 1 1 w	mod 1 1 0 r/m

Timing (clocks): 8-bit 90+EA
16-bit 155+EA

IDIV = Integer divide (signed)

1 1 1 1 0 1 1 w	mod 1 1 1 r/m

Timing (clocks): 8-bit 112+EA
16-bit 177+EA

AAD = ASCII adjust for divide

1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0

Timing: 60 clocks

CBW = Convert byte to word

1 0 0 1 1 0 0 0

Timing: 2 clocks

CWD = Convert word to double word

1 0 0 1 1 0 0 1

Timing: 5 clocks

LOGIC

NOT = Invert

1 1 1 1 0 1 1 w	mod 0 1 0 r/m

Timing (clocks): register 3
memory 16+EA

SHL/SAL = Shift logical/arithmetic left

1 1 0 1 0 0 v w	mod 1 0 0 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

SHR = Shift logical right

1 1 0 1 0 0 v w	mod 1 0 1 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

SAR = Shift arithmetic right

1 1 0 1 0 0 v w	mod 1 1 1 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

ROL = Rotate left

1 1 0 1 0 0 v w	mod 0 0 0 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

1 1 0 1 0 0 v w	mod 0 0 1 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

1 1 0 1 0 0 v w	mod 0 1 0 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

1 1 0 1 0 0 v w	mod 0 1 1 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 = And
Reg./memory and register to either

0 0 1 0 0 0 d w	mod reg 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 0 0 w	mod 1 0 0 r/m	data	data if w = 1

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

Immediate to accumulator

0 0 1 0 0 1 0 w	data	data if w = 1

Timing: 4 clocks

TEST = And function to flags, no result
Register/memory and register

1 0 0 0 0 1 0 w	mod reg r/m

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

Immediate data and register/memory

1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1

Timing (clocks): immediate with register 4
immediate with memory 10+EA

Immediate data and accumulator

1 0 1 0 1 0 0 w	data	data if w = 1

Timing: 4 clocks

OR = Or
Reg./memory and register to either

Immediate to register/memory

1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1

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

Immediate to accumulator

0 0 0 0 1 1 0 w	data	data if w = 1

Timing: 4 clocks

XOR = Exclusive or
Reg./memory and register to either

0 0 1 1 0 0 d w	mod reg 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 0 0 w	mod 1 1 0 r/m	data	data if w = 1

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

Immediate to accumulator

0 0 1 1 0 1 0 w	data	data if w = 1

Timing: 4 clocks

STRING MANIPULATION

REP = Repeat

1 1 1 1 0 0 1 z

Timing: 6 clocks/loop

MOVS = Move String

1 0 1 0 0 1 0 w

Timing: 17 clocks

CMPS = 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 0 1 0 0 0	disp-low	disp-high

Timing: 11 clocks

Indirect within segment

1 1 1 1 1 1 1 1	mod 0 1 0 r/m

Timing: 13+EA 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	mod 0 1 1 r/m

Timing: 29+EA 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+EA 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+EA clocks

RET = Return from CALL

Within segment

1 1 0 0 0 0 1 1

Timing: 8 clocks

Within seg. adding immed 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/JNGE = 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 0	disp

Timing (clocks): Jump is taken 8
Jump is not taken 4

JB/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 0 1 1 0	disp

Timing (clocks): Jump is taken 8
Jump is not taken 4

JP/JPE = Jump on parity/parity even

0 1 1 1 1 0 1 0	disp

Timing (clocks): Jump is taken 8
Jump is not taken 4

JO = Jump on overflow

0 1 1 1 0 0 0 0	disp

Timing (clocks): Jump is taken 8
Jump is not taken 4

JS = Jump on sign

0 1 1 1 1 0 0 0	disp

Timing (clocks): Jump is taken 8
Jump is not taken 4

JNE/JNZ = Jump on not equal/not zero

0 1 1 1 0 1 0 1	disp

Timing (clocks): Jump is taken 8
Jump is not taken 4

JNL/JGE = Jump on not less/greater or equal

0 1 1 1 1 1 0 1	disp

Timing (clocks): Jump is taken 8
Jump is not taken 4

JNLE/JG = Jump on not less or equal/greater

0 1 1 1 1 1 1 1	disp

Timing (clocks): Jump is taken 8
Jump is not taken 4

JNB/JAE = Jump on not below/above or equal

0 1 1 1 0 0 1 1	disp

Timing (clocks): Jump is taken 8
Jump is not taken 4

JNBE/JA = Jump on not below or equal/above

0 1 1 1 0 1 1 1	disp

Timing (clocks): Jump is taken 8
Jump is not taken 4

JNP/JPO = Jump on not parity/parity odd

0 1 1 1 1 0 1 1	disp

Timing (clocks): Jump is taken 8
Jump is not taken 4

JNO = Jump on not overflow

0 1 1 1 0 0 0 1	disp

Timing (clocks): Jump is taken 8
Jump is not taken 4

JNS = Jump on not sign

0 1 1 1 1 0 0 1	disp

Timing (clocks): Jump is taken 8
Jump is not taken 4

LOOP = Loop CX times

1 1 1 0 0 0 1 0	disp

Timing (clocks): Jump is taken 9
Jump is not taken 5

LOOPZ/LOOPE = Loop while zero/equal

1 1 1 0 0 0 0 1	disp

Timing (clocks): Jump is taken 11
Jump is not taken 5

LOOPNZ/LOOPNE = Loop while not zero/ not equal

1 1 1 0 0 0 0 0	disp

Timing (clocks): Jump is taken 11
Jump is not taken 5

JCXZ = Jump on CX zero

1 1 1 0 0 0 1 1	disp

Timing (clocks): Jump is taken 9
Jump is not taken 5

8086 CONDITIONAL TRANSFER OPERATIONS

Instruction	Condition	Interpretation
JE or JZ	ZF = 1	"equal" or "zero"
JL or JNGE	(SF xor OF) = 1	"less" or "not greater or equal"
JLE or JNG	((SF xor OF) or ZF) = 1	"less or equal" or "not greater"
JB or JNAE	CF = 1	"below" or "not above or equal"
JBE or JNA	(CF or ZF) = 1	"below or equal" or "not above"
JP or JPE	PF = 1	"parity" or "parity even"
JO	OF = 1	"overflow"
JS	SF = 1	"sign"
JNE or JNZ	ZF = 0	"not equal" or "not zero"
JNL or JGE	(SF xor OF) = 0	"not less" or "greater or equal"
JNLE or JG	((SF xor OF) or ZF) = 0	"not less or equal" or "greater"
JNB or JAE	CF = 0	"not below" or "above or equal"
JNBE or JA	(CF or ZF) = 0	"not below or equal" or "above"
JNP or JPO	PF = 0	"not parity" or "parity odd"
JNO	OF = 0	"not overflow"
JNS	SF = 0	"not sign"

*"Above" and "below" refer to the relation between two unsigned values, while "greater" and "less" refer to the relation between two signed values

INT = Interrupt
Type specified

1 1 0 0 1 1 0 1	type

Timing: 50 clocks

Type 3

1 1 0 0 1 1 0 0

Timing: 51 clocks

INTO = Interrupt on overflow

1 1 0 0 1 1 1 0

Timing: 52 clocks if pass 4 clocks if fail

IRET = Interrupt return

1 1 0 0 1 1 1 1

Timing: 24 clocks

PROCESSOR CONTROL

CLC = Clear carry

1 1 1 1 1 0 0 0

Timing: 2 clocks

STC = Set carry

1 1 1 1 1 0 0 1

Timing: 2 clocks

CMC = Complement carry

1 1 1 1 0 1 0 1

Timing: 2 clocks

NOP = No operation

1 0 0 1 0 0 0 0

Timing: 3 clocks

CLD = Clear direction

1 1 1 1 1 1 0 0

Timing: 2 clocks

STD = Set direction

1 1 1 1 1 1 0 1

Timing: 2 clocks

CLI = Clear interrupt

1 1 1 1 1 0 1 0

Timing: 2 clocks

STI = Set interrupt

1 1 1 1 1 0 1 1

Timing: 2 clocks

HLT = Halt

1 1 1 1 0 1 0 0

Timing: 2 clocks

WAIT = Wait

1 0 0 1 1 0 1 1

Timing: 3 clocks

LOCK = Bus lock prefix

1 1 1 1 0 0 0 0

Timing: 2 clocks

ESC = Escape (to external device)

1 1 0 1 1 x x x	mod x x x r/m

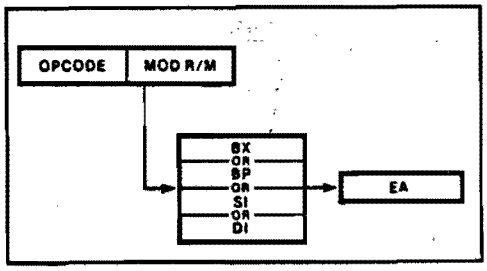
Timing: 7+EA clocks

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 (CL)
x = don't care
z is used for some string primitives to compare with ZF FLAG

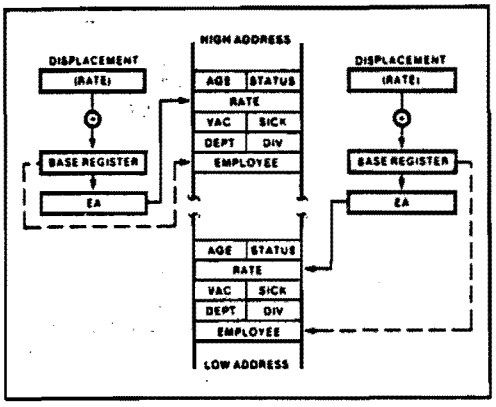
AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
DX = Variable port register
ES = Extra segment
Above/below refers to unsigned value
Greater = more positive;
Less = less positive (more negative) signed values
See page 1 for Operand Summary.
See page 2 for Segment Override Summary.

9.2 APP.B

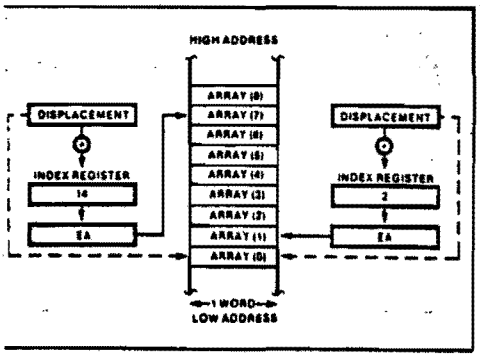
ADDRESSING MODES 8086 / APPLICATIONS



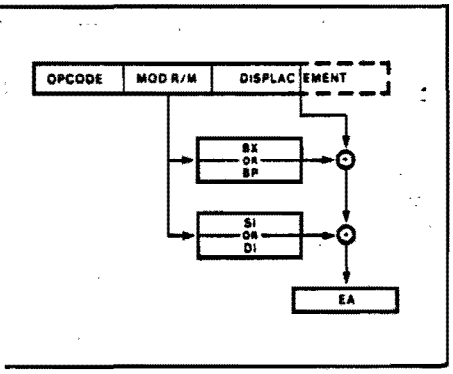
Register Indirect Addressing



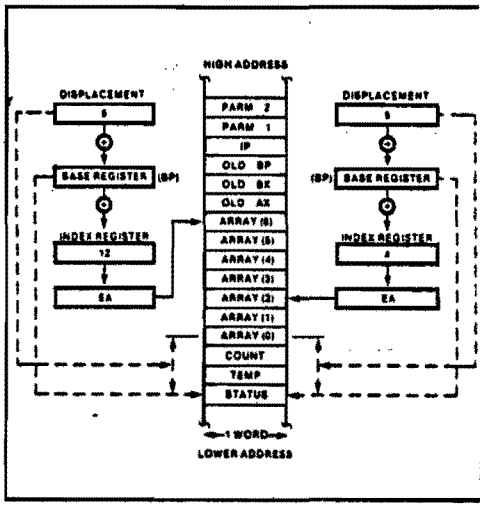
Accessing a Structure with Based Addressing



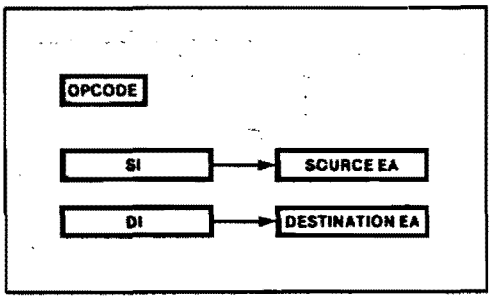
Accessing an Array with Indexed Addressing



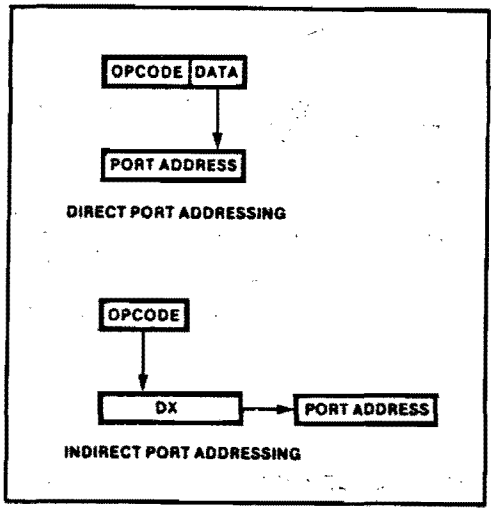
Based Indexed Addressing



Accessing a Stack Array with Based Indexed Addressing



String Operand Addressing



I/O Port Addressing

9.3APP. C

PARALLEL I/O PORT CONFIGURATION

Parallel I/O Port Configuration

Port	Mode	Driver (D)/ Terminator (T)	Jumper Configuration			Restrictions	
			Delete	Add	Effect	Port	
C8	0 Input	8226: A8, A9	*21-25	24-25	8226 = input enabled.	CA	None; can be in mode 0 or 1, input or output.
						CC	None; can be in Mode 0, input or output, unless Port CA is in Mode 1.
C8	0 Output (latched)	8226: A8, A9		*21-25	8226 = output enabled.	CA	None; can be in Mode 0 or 1, input or output.
						CC	None; can be in Mode 0, input or output, unless Port CA is in Mode 1.
C8	1 Input (strobed)	8226: A8, A9 T: A10 D: A11	*21-25	24-25	8226 = input enabled.	CA	None; can be in Mode 0 or 1, input or output.
				*15-16	Connects J1-26 to STB_A/ input.	CC	Port CC bits perform the following: • Bits 0, 1, 2 — Control for Port CA if Port CA is in Mode 1. • Bit 3 — Port C8 Interrupt (PA INTR) to interrupt jumper matrix • Bit 4 — Port C8 Strobe (STB/) input. • Bit 5 — Port C8 Input Buffer Full (IBF) output. • Bits 6, 7 — Port CC input or output (both, must be in same direction).
			*19-20 and *32-33	19-33	Connects IBF_A output to J1-18.		
				22-32	Connects INT_A output to interrupt matrix.		
C8	1 Output (latched)	8226: A8, A9 T: A10 D: A11		*21-25	8226 = output enabled.	CA	None; can be in Mode 0 or 1, input or output.
				*17-18	Connects J1-30 to ACK_A/ input.	CC	Port EA bits perform the following: • Bits 0, 1, 2 — Control for Port CA if Port CA is in Mode 1. • Bit 3 — Port C8 Interrupt (PA INTR) to interrupt jumper matrix. • Bits 4, 5 — Port CC input or output (both must be in same direction). • Bit 6 — Port C8 Acknowledge (ACK/) input. • Bit 7 — Port C8 Output Buffer Full (OBF/) output.
			*32-33 and *13-14	13-33	Connects OBF_A output to J1-18.		
				22-32	Connects INT_A output to interrupt matrix.		

*Default jumper connected at the factory.

Port	Mode	Driver (D)/ Terminator (T)	Jumper Configuration			Restrictions	
			Delete	Add	Effect	Port	
C8	2 (bidirectional)	8226: A8, A9 T: A10 D: A11	*21-25	17-25	Allows ACK_A/ input to control 8226 in/out direction.	CA	None; can be in Mode 0 or 1, input or output.
						CC	Port CC bits perform the following: • Bit 0 — Can only be used for jumper option (see figure 5-2 zone 9ZC6). • Bits 1, 2 — Can be used for input or output if Port CC is in Mode 0. • Bit 3 — Port C8 Interrupt (PA INTR) to interrupt jumper matrix. • Bit 4 — Port C8 Strobe (STB/) input. • Bit 5 — Port C8 Input Buffer Full (IBF) output. • Bit 6 — Port C8 Acknowledge (ACK/) input. • Bit 7 — Port C8 Output Buffer Full (OBF/) output.
				*15-16	Connects J1-26 to STB_A/ input.		
			*19-20 and *26-27	19-27	Connects IBF_A output to J1-24.		
				*17-18	Connects J1-30 to ACK_A/ input.		
			*13-14 and *32-33	13-33	Connects OBF_A/ output to J1-18.		
				22-32	Connects INT_A output to interrupt matrix.		
CA	0 Input	T: A12, A13	None	None		C8	None.
						CC	None; Port CC can be in Mode 0, input or output, if Port C8 is also in Mode 0.
CA	0 Output (latched)	D: A12, A13	None	None		C8	None
						CC	None; Port CC can be in Mode 0, input or output, if Port C8 is also in Mode 0.
CA	1 Input (strobed)	T: A10, A12, A13 D: A11		*28-29	Connects IBF_B output to J1-22.	C8	None.
						CC	Port CC bits perform the following: • Bit 0 — Port CA Interrupt (PB INTR) to interrupt jumper matrix. • Bit 1 — Port CA Input Buffer Full (IBF) output. • Bit 2 — Port CA Strobe (STB/) input.
			*13-14 *30-31	14-30	Connects J1-32 to STB_B/ input.		
				26-34	Connects INT_B output interrupt matrix.		

*Default jumper connected at the factory.

Parallel I/O Port Configuration

Port	Mode	Driver (D) Terminator (T)	Jumper Configuration			Restrictions	
			Delete	Add	Effect	Port	
							• Bit 3 — If Port C8 is in Mode 0, bit 3 can be input or output. Otherwise, bit 3 is reserved. • Bits 4, 5 — Depends on Port C8 mode. • Bits 6, 7 — Input or output (both must be in same direction).
CA	1 Output (latched)	T: A10 D: A11, A12, A13		*28-29	Connects OBF_B/ output output J1-22.	C8	None.
						CC	Port CC bits perform the following: • Bit 0 — Port CA interrupt (PB INTR) to interrupt jumper matrix. • Bit 1 — Port CA Output Buffer Full (OBF/) output. • Bit 2 — Port CA Acknowledge (ACK/) input. • Bit 3 — If Port C8 is in Mode 0, bit 3 can be input or output. Otherwise, bit 3 is reserved. • Bits 4, 5 — Input or output (both must be in same direction). • Bit 6, 7 — Depends on Port C8 mode.
			*13-14 and *30-31	14-30	Connects J1-32 to ACK_B/ input.		
			*26-27	26-34	Connects INT_B output to interrupt matrix.		
CC (upper)	0 Input	T: A10	None	*15-16 *19-20 *17-18 *13-14	Connects bit 4 to J1-26. Connects bit 5 to J1-28. Connects bit 6 to J1-30. Connects bit 7 to J1-32.	C8	Port C8 must be in Mode 0 for all four bits to be available.
						CA	Port CA must be in Mode 0 for all four bits to be available.
CC (lower)	0 Input	T: A11	None	*26-27 *28-29 *30-31 *32-33	Connects bit 0 to J1-24. Connects bit 1 to J1-22. Connects bit 2 to J1-20. Connects bit 3 to J1-18.	C8	Port C8 must be in Mode 0 for all four bits to be available.
						CA	Port CA must be in Mode 0 for all four bits to be available.
CA (upper)	0 Output (latched)	D: A10	None	Same as for Port CC (upper) mode 0 Input.		C8	Same as for Port CC (upper) Mode 0 Input.
CA (lower)	0 Output (latched)	D: A11	None	Same as for Port CC (lower) Mode 0 Input.		CC	Same as for Port CC (lower) Mode 0 Input.

*Default jumper connected at the factory.

9.4 APP. D SPECIFICATIONS OF THE iSBC 86/12 S.B.C.

Specifications

WORD SIZE

Instruction:	8, 16, 24, or 32 bits.
Data	8/16 bits.

CYCLE TIME: 800 nanosecond for fastest executable instruction (assumes instruction is in the queue). 1.2 microseconds for fastest executable instruction (assumes instruction is not in the queue).

MEMORY CAPACITY

On-Board ROM/EPROM:	Up to 16K bytes; user installed in 1K, 2K, or 4K byte increments.
On-Board Dynamic RAM:	32K bytes. Integrity maintained during power failure with user-furnished batteries.
Off-Board Expansion:	Up to 1 megabyte of user-specified combination of RAM, ROM, and EPROM.

MEMORY ADDRESSING

On-Board ROM/EPROM:	FF000-FFFFF$_H$ (using 2758 EPROM's), FE000-FFFFF$_H$ (using 2316E ROM's or 2716 EPROM's), and FC000-FFFFF$_H$ (using 2332 ROM's).
On-Board RAM: (CPU Access)	00000-07FFF$_H$.
On-Board RAM: (Multibus Access)	Jumpers 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.

SERIAL COMMUNICATIONS

Synchronous:	5-, 6-, 7-, or 8-bit characters. Internal; 1 or 2 sync characters. Automatic sync insertion.
Asynchronous:	5-, 6-, 7-, or 8-bit characters. Break character generation. 1, 1½, or 2 stop bits. False start bit detection.

Sample Baud Rate:

Frequency[1] (kHz, Software Selectable)	Baud Rate (Hz)[2]		
	Synchronous	Asynchronous	
		÷16	÷64
153.6	—	9600	2400
76.8	—	4800	1200
38.4	38400	2400	600
19.2	19200	1200	300
9.6	9600	600	150
4.8	4800	300	75
2.4	2400	150	—
1.76	1760	110	—

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:

Function	Single Timer		Dual Timers (Two Timers Cascaded)	
	Min.	Max.	Min.	Max.
Real-Time Interrupt Interval	1.63 μsec	427.1 msec	3.26 μsec	466.5 minutes
Rate Generator (Frequency)	2.342 Hz	613.5 kHz	0.000036 Hz	306.8 kHz

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	Receive Data
Data Set Ready	Secondary Receive Data*
Data Terminal Ready	Secondary CTS*
Request to Send	Transmit Clock*
Receive Clock	Transmit Data

*Can support only one.

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 18 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 gm (19 ounces).

POWER REQUIREMENTS:

CONFIGURATION	V_{CC} = +5V±5%	V_{DD} = +12V±5%	V_{BB} = −5V±5%	V_{AA} = −12V±5%
Without EPROM[1]	5.2A	350 mA	—	40 mA
RAM Only[3]	390 mA	40 mA	1.0 mA	—
With iSBC 530[4]	5.2A	450 mA	—	140 mA
With 4K EPROM[5] (Using 2758)	5.5A	450 mA	—	140 mA
With 8K ROM[5] (Using 2316E)	6.1A	450 mA	—	140 mA
With 8K EPROM[5] (Using 2716)	5.5A	450 mA	—	140 mA
With 16K ROM[5] (Using 2332)	5.4A	450 mA	—	140 mA

Notes: 1. Does not include power for optional ROM/EPROM, I/O drivers, and I/O terminators.

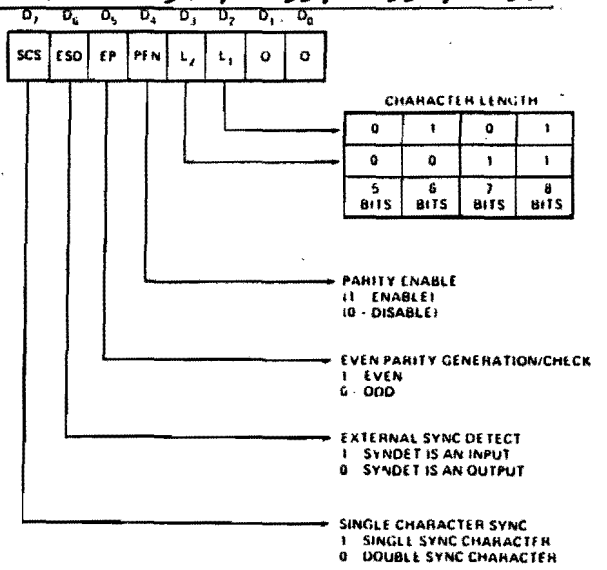
2. Does not include power required for optional ROM/EPROM, I/O drivers, and I/O terminators.
3. RAM chips powered via auxiliary power bus.
4. Does not include power for optional ROM/EPROM, I/O drivers, and I/O terminators. Power for iSBC 530 is supplied via serial port connector.
5. Includes power required for four ROM/EPROM chips, and I/O terminators installed for16 I/O lines; all terminator inputs low.

9.5 APP. E I/O ADDRESS ASSIGN. MENTS

I/O Address Assignments

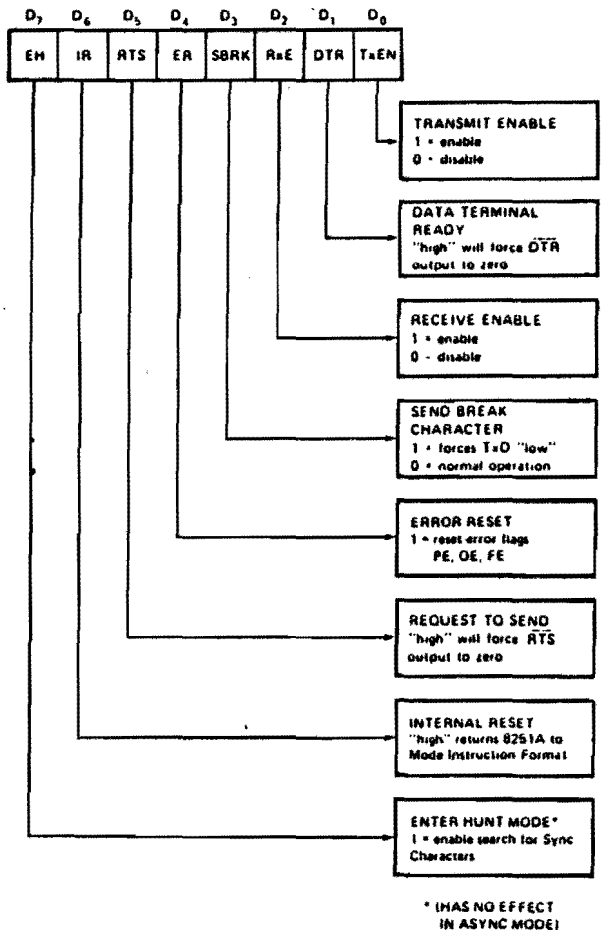
I/O Address*	Chip Select	Function
000C0 or 000C4	8259A PIC	Write: ICW1, OCW2, and OCW3 Read: Status and Poll
000C2 or 000C6		Write: ICW2, ICW3, ICW4, OCW1 (Mask) Read: OCW1 (Mask)
000C8	8255A PPI	Write: Port A (J1) Read: Port A (J1)
000CA		Write: Port B (J1) Read: Port B (J1)
000CC		Write: Port C (J1)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

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



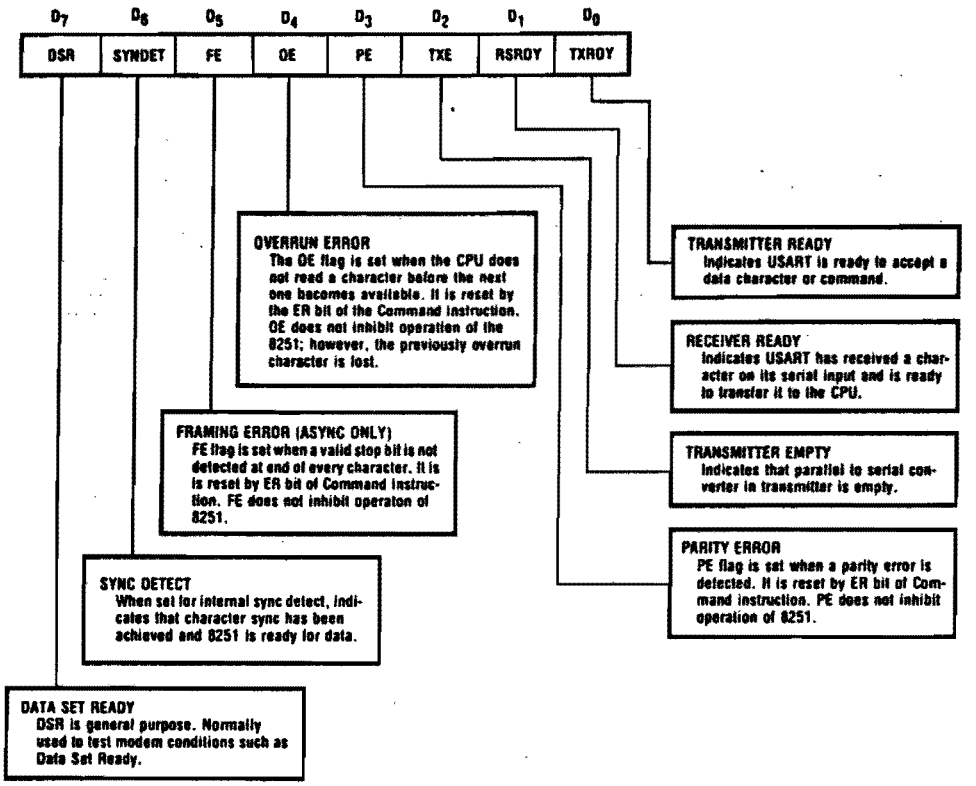
NOTE: IN EXTERNAL SYNC MODE, PROGRAMMING DOUBLE CHARACTER SYNC WILL AFFECT ONLY THE Tx.

USART Synchronous Mode Instruction Word Format

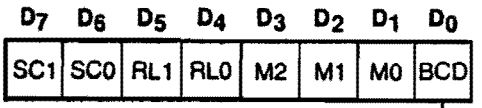


Note: Error Reset must be performed whenever RxEnable and Enter Hunt are programmed.

USART Command Instruction Word Format



USART Status Read Format



(BINARY/BCD)

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

M2	M1	M0	(MODE)	
0	0	0	Mode 0	
0	0	1	Mode 1	
X	1	0	Mode 2	
X	1	1	Mode 3	← Use Mode 3 for Baud Rate Generator
1	0	0	Mode 4	
1	0	1	Mode 5	

RL1	RL0	(READ/LOAD)
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.

SC1	SC0	(SELECT COUNTER)
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Illegal

PIT Mode Control Word Format

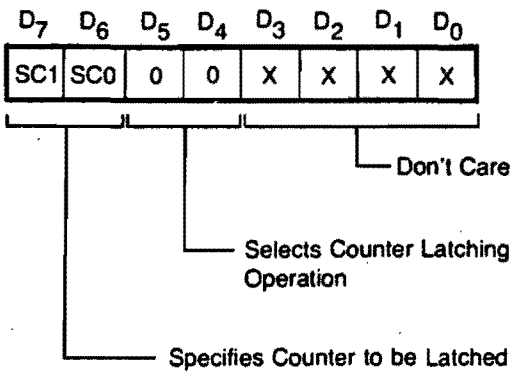
PROGRAMMING FORMAT

Step		
1		Mode Control Word Counter n
2	LSB	Count Register Byte Counter n
3	MSB	Count Register Byte Counter n

ALTERNATE PROGRAMMING FORMAT

Step		
1		Mode Control Word Counter 0
2		Mode Control Word Counter 1
3		Mode Control Word Counter 2
4	LSB	Counter Register Byte Counter 1
5	MSB	Count Register Byte Counter 1
6	LSB	Count Register Byte Counter 2
7	MSB	Count Register Byte Counter 2
8	LSB	Count Register Byte Counter 0
9	MSB	Count Register Byte Counter 0

PIT Programming Sequence Examples

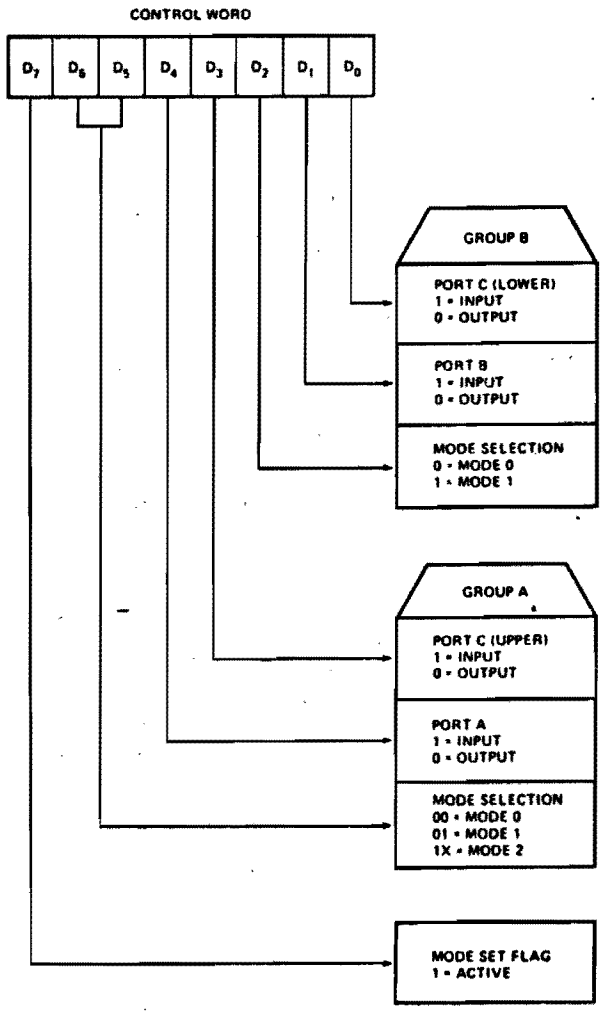


PIT Counter Register Latch Control Word Format

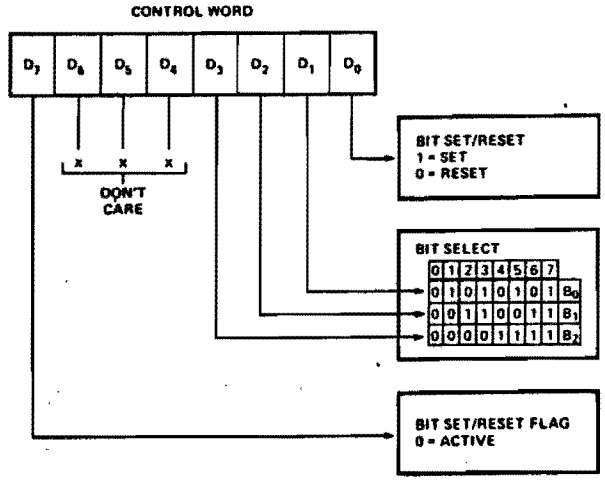
PIT Time Intervals Vs Timer Counts

T	N*
10 μsec	12
100 μsec	123
1 msec	1229
10 msec	12288
50 msec	61440

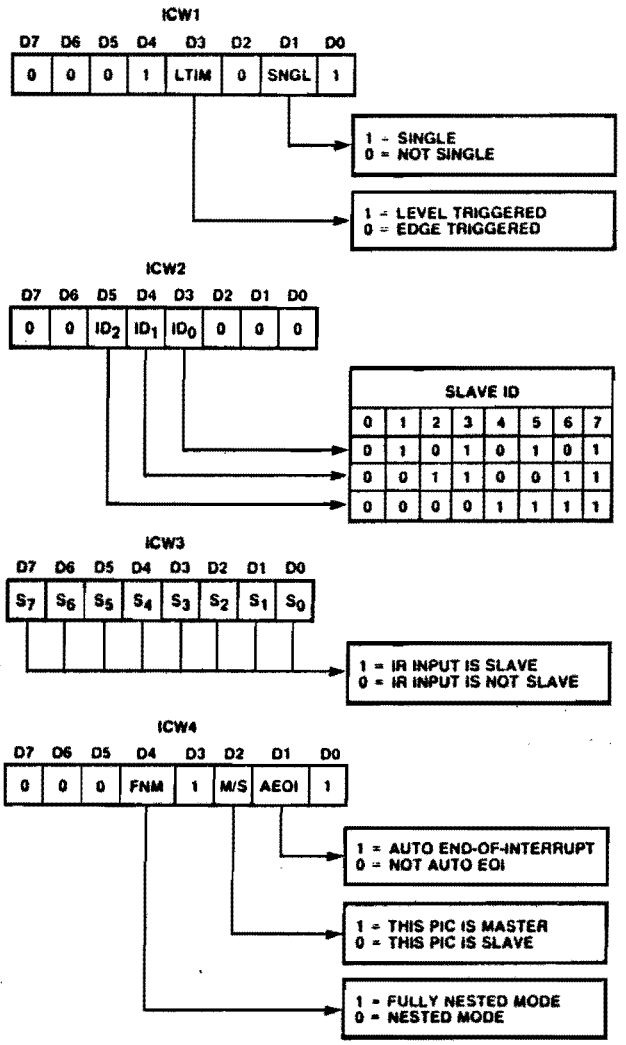
*Count Values (N) assume clock is 1.23 MHz. Count Values (N) are in decimal.



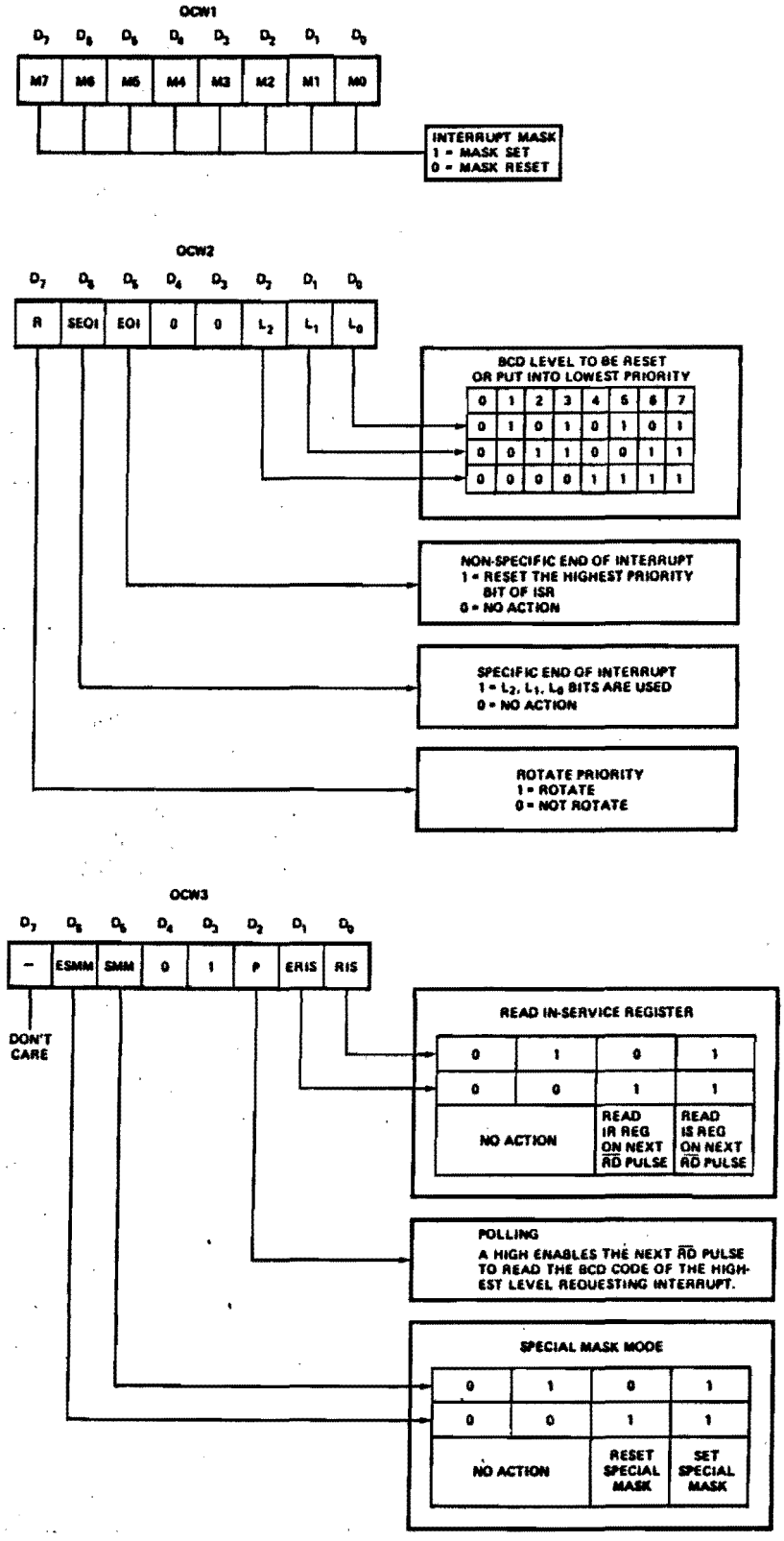
PPI Control Word Format



PPI Port C Bit Set/Reset Control Word Format



PIC Initialization Command Word Formats



PIC Operation Control Word Formats

9.7
APP. G

INSTRUCTION SET 8087

Addition	
FADD	Add real
FADDP	Add real and pop
FIADD	Integer add
Subtraction	
FSUB	Subtract real
FSUBP	Subtract real and pop
FISUB	Integer subtract
FSUBR	Subtract real reversed
FSUBRP	Subtract real reversed and pop
FISUBR	Integer subtract reversed
Multiplication	
FMUL	Multiply real
FMULP	Multiply real and pop
FIMUL	Integer multiply
Division	
FDIV	Divide real
FDIVP	Divide real and pop
FIDIV	Integer divide
FDIVR	Divide real reversed
FDIVRP	Divide real reversed and pop
FIDIVR	Integer divide reversed
Other Operations	
FSQRT	Square root
FSCALE	Scale
FPREM	Partial remainder
FRNDINT	Round to integer
FXTRACT	Extract exponent and significand
FABS	Absolute value
FCHS	Change sign

FPTAN	Partial tangent
FPATAN	Partial arctangent
F2XM1	2^X-1
FYL2X	$Y \bullet \log_2 X$
FYL2XP1	$Y \bullet \log_2(X+1)$

FLDZ	Load +0.0
FLD1	Load +1.0
FLDPI	Load π
FLDL2T	Load $\log_2 10$
FLDL2E	Load $\log_2 e$
FLDLG2	Load $\log_{10} 2$
FLDLN2	Load $\log_e 2$

FINIT/FNINIT	Initialize processor
FDISI/FNDISI	Disable interrupts
FENI/FNENI	Enable interrupts
FLDCW	Load control word
FSTCW/FNSTCW	Store control word
FSTSW/FNSTSW	Store status word
FCLEX/FNCLEX	Clear exceptions
FSTENV/FNSTENV	Store environment
FLDENV	Load environment
FSAVE/FNSAVE	Save state
FRSTOR	Restore state
FINCSTP	Increment stack pointer
FDECSTP	Decrement stack pointer
FFREE	Free register
FNOP	No operation
FWAIT	CPU wait

FCOM	Compare real
FCOMP	Compare real and pop
FCOMPP	Compare real and pop twice
FICOM	Integer compare
FICOMP	Integer compare and pop
FTST	Test
FXAM	Examine

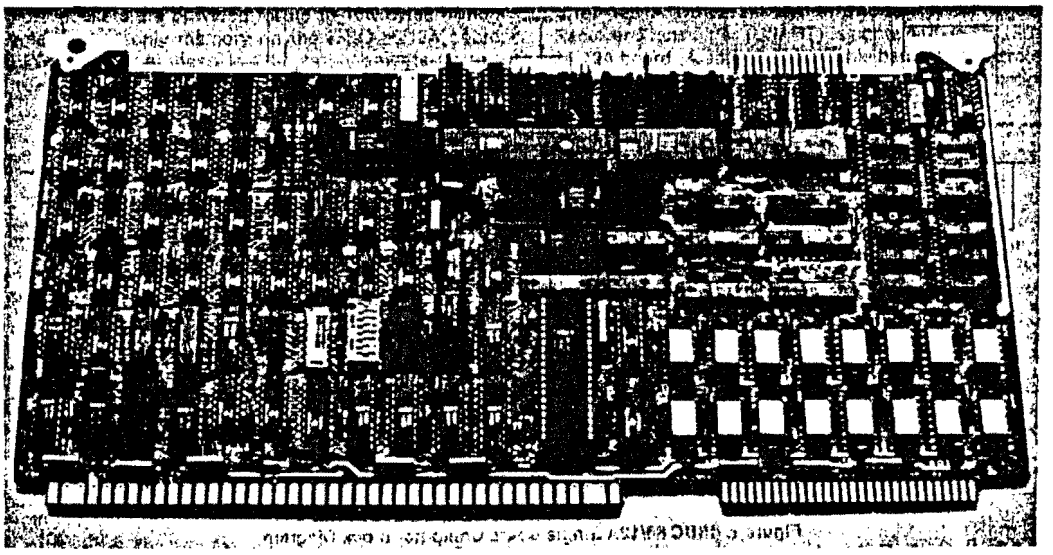
Real Transfers	
FLD	Load real
FST	Store real
FSTP	Store real and pop
FXCH	Exchange registers
Integer Transfers	
FILD	Integer load
FIST	Integer store
FISTP	Integer store and pop
Packed Decimal Transfers	
FBLD	Packed decimal (BCD) load
FBSTP	Packed decimal (BCD) store and pop

intel

iSBC 86/12A or (pSBC 86/12A*) SINGLE BOARD COMPUTER

- 8086 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 megabyte
- 24 programmable parallel I/O lines with sockets for interchangeable line drivers and terminators
- Programmable synchronous/asynchronous 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 MULTIMODULE 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.



FUNCTIONAL DESCRIPTION

Central Processing Unit

The central processor for the iSBC 86/12A board is Intel's 8086, a powerful 16-bit HMOS device. The 225 sq. mil chip contains 29,000 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 ease 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.

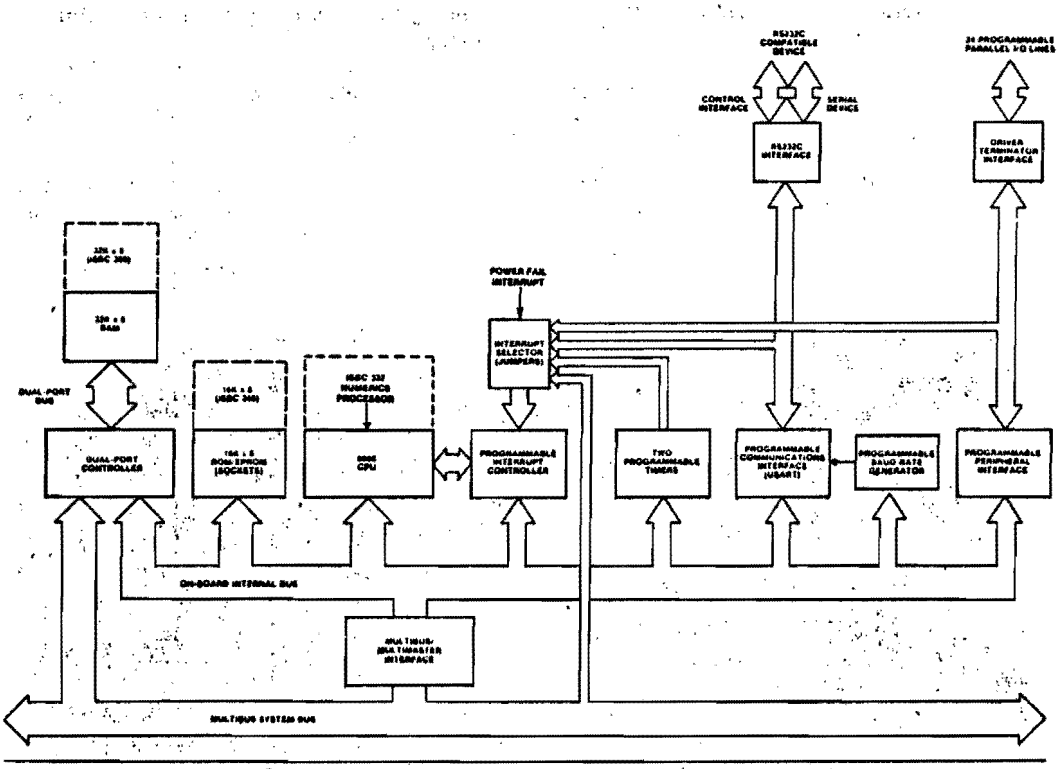


Figure 1. iSBC 86/12A Single Board Computer Block Diagram

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 may be expanded to 64K bytes with the ISBC 300 32K-byte MULTIMODULE 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 00000_H. 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 segments (or 16K-byte segments with ISBC 300 module). These features allow the multiprocessor 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 nonvolatile 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 paths. On-board EPROM capacity may be expanded to 32K bytes with the addition of the ISBC 340 16K-byte MULTIMODUE 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 MULITBUS 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

Table 1. Input/Output Port Modes of Operation

Port	Lines (qty)	Mode of Operation					
		Unidirectional				Bidirectional	Control
		Input		Output			
		Latched	Latched & Strobed	Latched	Latched & Strobed		
1	8	X	X	X	X	X	
2	8	X	X	X	X		
3	4	X		X			X[1]
	4	X		X			X[1]

Note
1. Part of port 3 must be used as a control port when either port 1 or port 2 are used as a latched and strobed input or a latched and strobed output port or port 1 is used as a bidirectional port.

frequencies. The USART can be programmed by the system software to select the desired asynchronous or synchronous serial data transmission technique (including IBM Bi-Sync). The mode of operation (i.e., synchronous 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 8259A 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 terminators 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/demultiplexing 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 msec.

Table 2. Programmable Timer Functions

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

Multimaster Capabilities — The iSBC 86/12A board is a full 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

receiving devices only. This design prevents slow master modules from being handicapped in their attempts to gain control of the bus, but does not restrict the speed at which faster modules can transfer data via the same bus. The most obvious applications for the master-slave capabilities of the bus are multiprocessor configurations, high speed peripheral control, but are by no means limited to these three.

Interrupt Capablity

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 00008_H. 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.

*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.

Table 3. Programmable Interrupt Modes

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

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 cardcages are available to support multiboard 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.

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 — 1.2μsec
— 400 nsec (assumes instruction in the queue)

Note:
Basic instruction cycle is defined as the fastest instruction time (i.e., two clock cycles)

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 Expandion — Up to 1 megabyte in user specified combinations of RAM and EPROM.

Note:
Read only memory may be added in 2K, 4K, or 8K-byte increments.

Memory Addressing

On-Board EPROM — FF000-FFFFF$_H$ (using 2758 EPROMs); FE000-FFFFF$_H$ (using 2716 EPROMs); FC000-FFFFF$_H$ (using 2732 EPROMS); F8000-FFFFF$_H$ (with iSBC 340 EPROM option and four additional 2732 EPROMs).

On-Board RAM — 32K bytes of dual port RAM. Optionally expandable to 64K bytes with iSBC 300 RAM option.

CPU Access — 32K bytes: 00000-07FFF$_H$; 64K bytes: 00000-0FFFF$_H$.

MULTIBUS Access — Jumper selectable for any 8K-byte boundary, but not crossing a 128K-byte boundary. Access for 8K, 16K, 24K or 32K (16K, 32K, 48K, 64K with iSBC 300 option) bytes may be selected for on-board CPU use only.

I/O Capacity

Parallel — 24 programmable lines using one 8255A.
Serial — 1 programmable line using one 8251A.

I/O Addressing

On-Board Programmable I/O

Port	8255A				USART	
	1	2	3	Control	Data	Control
Address	C8	CA	CC	CE	D8 or DC	DA or DE

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

Frequency (kHz) (Software Selectable)	Baud Rate (Hz)		
	Synchronous	Asynchronous	
		÷ 16	÷ 64
153.6	—	9600	2400
76.8	—	4800	1200
38.4	38400	2400	600
19.2	19200	1200	300
9.6	9600	600	150
4.8	4800	300	75
2.4	2400	150	—
1.76	1760	110	—

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 C8 Write: ICW2, ICW3, ICW4, 0CW1 (Mask Register)
Read: 0CW1 (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. 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 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 153.60 kHz ±0.1% (6.51 μs period nominal).

Note:
Above frequencies are user selectable.

Event Rate: 2.46 MHz max

Output Frequencies/Timing Intervals

Function	Single Timer/Counter		Dual Timer/Counter (Two Timers Cascaded)	
	Min	Max	Min	Max
Real-time interrupt	1.63 μs	427.1 ms	3.26 s	466.50 min
Programmable one-shot	1.63 μs	427.1 ms	3.26 s	466.50 min
Rate generator	2.342 Hz	613.5 kHz	0.000036 Hz	306.8 kHz
Square-wave rate generator	2.342 Hz	613.5 kHz	0.000036 Hz	306.8 kHz
Software triggered strobe	1.63 μs	427.1 ms	3.26 s	466.50 min
Hardware triggered strobe	1.63 μs	427.1 ms	3.26 s	466.50 min
Event counter	—	2.46 MHz	—	—

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

Interface	Pins (qty)	Centers (in.)	Mating Connectors
Bus	86	0.156	VIKING 3KH43/9AMK12
Parallel I/O	50	0.1	3M 3415-000
Serial I/O	26	0.1	3M 3462-000

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.

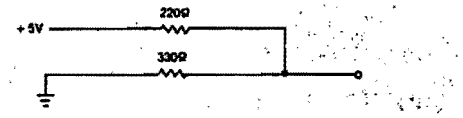
Driver	Characteristic	Sink Current (mA)
7438	I,OC	48
7437	I	48
7432	NI	16
7426	I,OC	16
7409	NI,OC	16
7408	NI	16
7403	I,OC	16
7400	I	16

Note:
I = inverting; NI = non-inverting; OC = open collector.

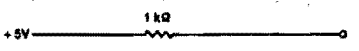
Port 1 of the 8255A has 20 mA totem-pole bidirectional drivers and 1 kΩ terminators.

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

220Ω/330Ω (iSBC 901 OPTION)



1 K Ω (iSBC 902 OPTION)



Bus Drivers

Function	Characteristic	Sink Current (mA)
Data	Tri-state	50
Address	Tri-state	50
Commands	Tri-state	32

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

Configuration	Current Requirements			
	V_{CC} = +5V ±5% (max)	V_{DD} = +12V ±5% (max)	V_{BB} = −5V ±5% (max)	V_{AA} = −12V ±5% (max)
Without EPROM[1]	5.2A	350 mA	—	40 mA
RAM Only[3]	390 mA	40 mA	1.0 mA	—
With iSBC 530[4]	5.2A	450 mA	—	140 mA
With 4K EPROM[5] (using 2758)	5.5A	350 mA	—	40 mA
With 8K EPROM[5] (using 2716)	5.5A	350 mA	—	40 mA
With 16K EPROM[5] (using 2732)	5.4A	350 mA	—	40 mA

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.

Environmental Characteristics

Operating Temperature — 0°C to 55°C
Relative Humidity — to 90% (without condensation)

Reference Manual

9803074-01 — iSBC 86/12A Single Board Computer Hardware Reference Manual (NOT SUPPLIED)

Reference manuals are shipped with each product only if designated SUPPLIED (see above). Manuals may be ordered from any Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

Part Number	Description
SBC 86/12A	Single Board Computer with 32K bytes RAM

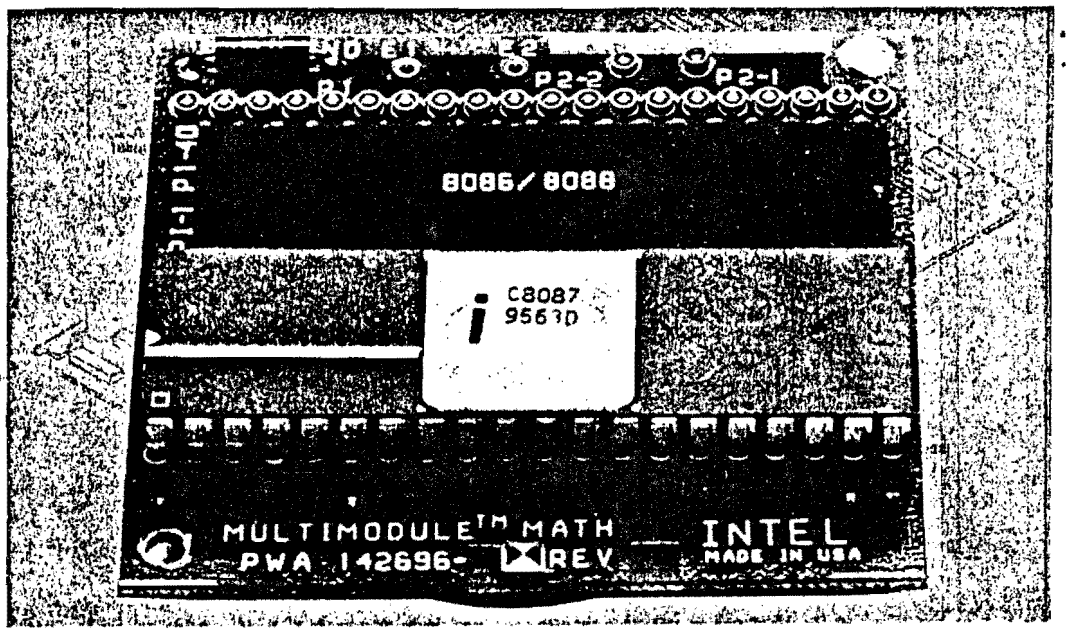
9.9APP. I iSBC 337 Multimodule N.D.P.

intel

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 × 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

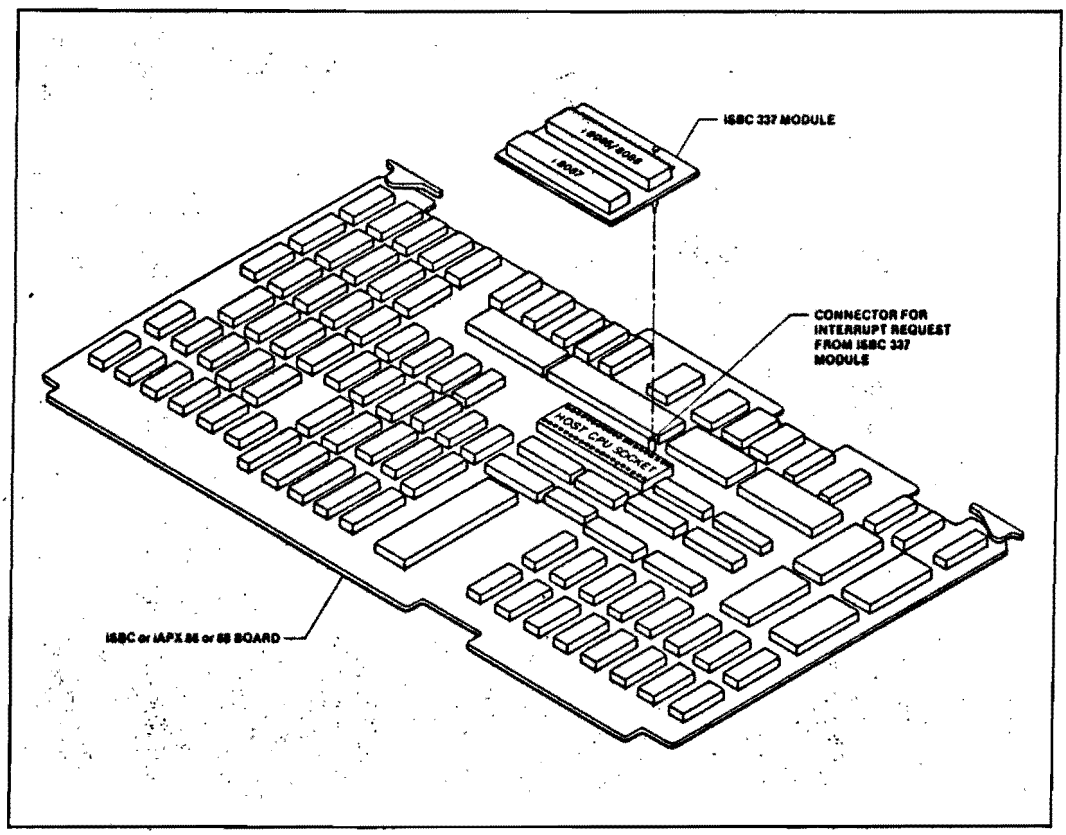


Fig 1 iSBC 337 Module Installation

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

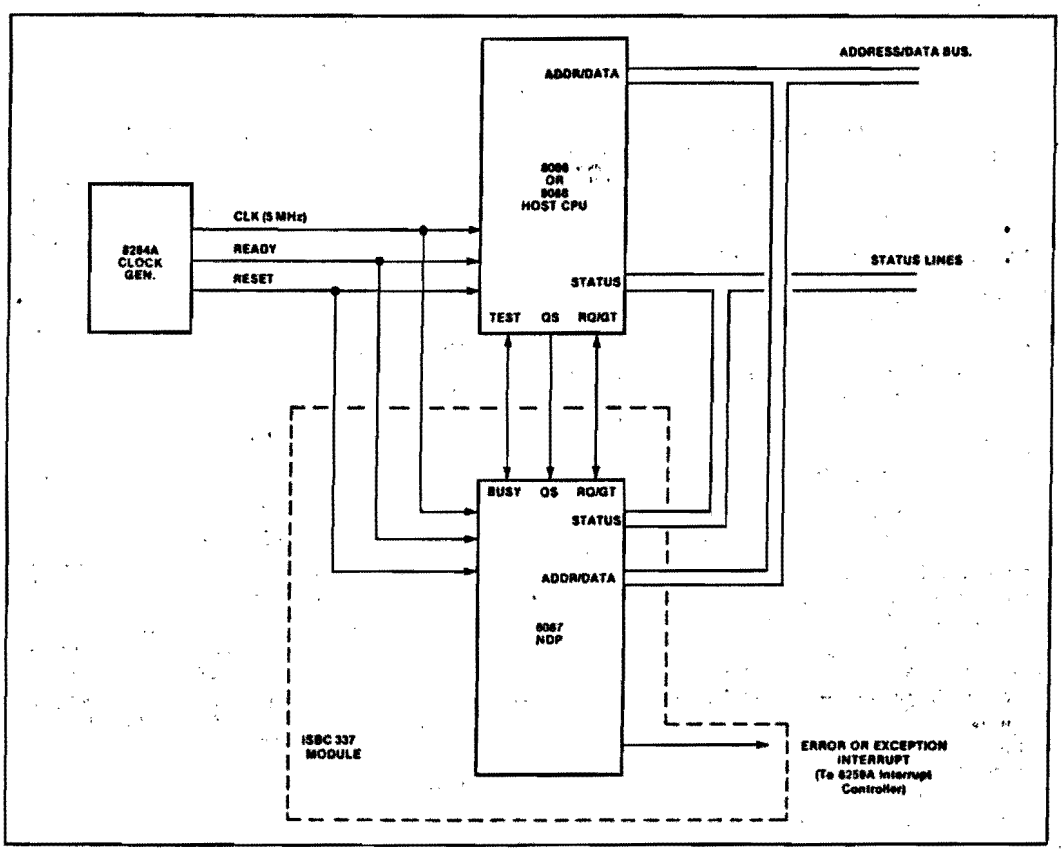


Figure 2. ISBC 337 System Configuration

Table 1

8087 Datatypes

Data Formats	Range	Precision	Most Significant Byte (7 07 07 07 07 07 07 07 07 07 0)
Word Integer	10^4	16 Bits	I_{15} ... I_0 — Two's Complement
Short Integer	10^9	32 Bits	I_{31} ... I_0 — Two's Complement
Long Integer	10^{19}	64 Bits	I_{63} ... I_0 — Two's Complement
Packed BCD	10^{18}	18 Digits	S — D_{17} D_{16} ... D_1 D_0
Short Real	$10^{\pm 38}$	24 Bits	S E_7 E_0 F_1 ... F_{23} — F_0 Implicit
Long Real	$10^{\pm 308}$	53 Bits	S E_{10} E_0 F_1 ... F_{52} — F_0 Implicit
Temporary Real	$10^{\pm 4932}$	64 Bits	S E_{14} ... E_0 F_0 ... F_{63}

Note:

Integer: I **Sign:** S **Packed BCD:** $(-1)^S(D_{17} \ldots D_0)$ **Bias** = 127 for Short Real
Fraction: F **BCD Digit (4 Bits):** D **Real:** $(-1)^S 2^{(E-BIAS)} (F_0 F_{-1} \ldots)$ 1023 for Long Real
Exponent: E 16383 for Temp Real

the interrupt request signal for connection to the base board and then to the 8259A. Another alternative is to use a wire to establish this connection.

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 64-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.

Table 2 Execution Time for Selected 8087 Actual and Emulated Instructions

Floating Point Instruction	Approximate Execution Time (ms)	
	8087 (5 MHz Clock)	8086 Emulation
Add/Subtract Magnitude	14/18	1,600
Multiply (single precision)	19	1,600
Multiply (extended precision)	27	2,100
Divide	39	3,200
Compare	9	1,300
Load (double precision)	10	1,700
Store (double precision)	21	1,200
Square Root	36	19,600
Tangent	90	13,000
Exponentiation	100	17,100

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-

Table 3. 8087 Instruction Set

Data Transfer Instructions

Real Transfers	
FLD	Load real
FST	Store real
FSTP	Store real and pop
FXCH	Exchange registers
Integer Transfers	
FILD	Integer load
FIST	Integer store
FISTP	Integer store and pop
Packed Decimal Transfers	
FBLD	Packed decimal (BCD) load
FBSTP	Packed decimal (BCD) store and pop

Comparison Instructions

FCOM	Compare real
FCOMP	Compare real and pop
FCOMPP	Compare real and pop twice
FICOM	Integer compare
FICOMP	Integer compare and pop
FTST	Test
FXAM	Examine

Transcendental Instructions

FPTAN	Partial tangent
FPATAN	Partial arctangent
F2XM1	2^X-1
FYL2X	$Y*\log_2 X$
FYL2XP1	$Y*\log_2(X+1)$

Arithmetic Instructions

Addition	
FADD	Add real
FADDP	Add real and pop
FIADD	Integer add
Subtraction	
FSUB	Subtract real
FSUBP	Subtract real and pop
FISUB	Integer subtract
FSUBR	Subtract real reversed
FSUBRP	Subtract real reversed and pop
FISUBR	Integer subtract reversed
Multiplication	
FMUL	Multiply real
FMULP	Multiply real and pop
FIMUL	Integer multiply
Division	
FDIV	Divide real
FDIVP	Divide real and pop
FIDIV	Integer divide
FDIVR	Divide real reversed
FDIVRP	Divide real reversed and pop
FIDIVR	Integer divide reversed
Other Operations	
FSQRT	Square root
FSCALE	Scale
FPREM	Partial remainder
FRNDINT	Round to integer
FXTRACT	Extract exponent and significand
FABS	Absolute value
FCHS	Change sign

Processor Control Instructions

FINIT/FNINIT	Initialize processor
FDISI/FNDISI	Disable interrupts
FENI/FNENI	Enable interrupts
FLDCW	Load control word
FSTCW/FNSTCW	Store control word
FSTSW/FNSTSW	Store status word
FCLEX/FNCLEX	Clear exceptions
FSTENV/FNSTENV	Store environment
FLDENV	Load environment
FSAVE/FNSAVE	Save state
FRSTOR	Restore state
FINCSTP	Increment stack pointer
FDECSTP	Decrement stack pointer
FFREE	Free register
FNOP	No operation
FWAIT	CPU wait

tion stream. The CPU fetches all instructions from memory; by monitoring the status signals emitted by the CPU, the NDP control unit determines when an 8086 instruction is being fetched. The CU taps the bus in parallel with the CPU and obtains that portion of the data stream.

After decoding the instruction, the host executes all opcodes but ESCAPE (ESC), while the 8087 executes only the ESCAPE class instructions. (The first five bits of all ESCAPE instructions are identical). 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, or will require storing one or more operands from 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.

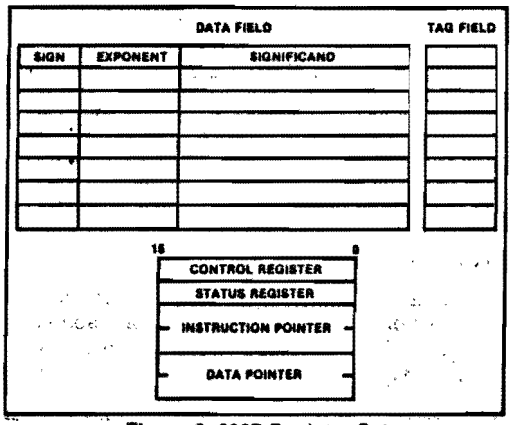


Figure 3. 8087 Register Set

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 (C_0-C_3) 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.

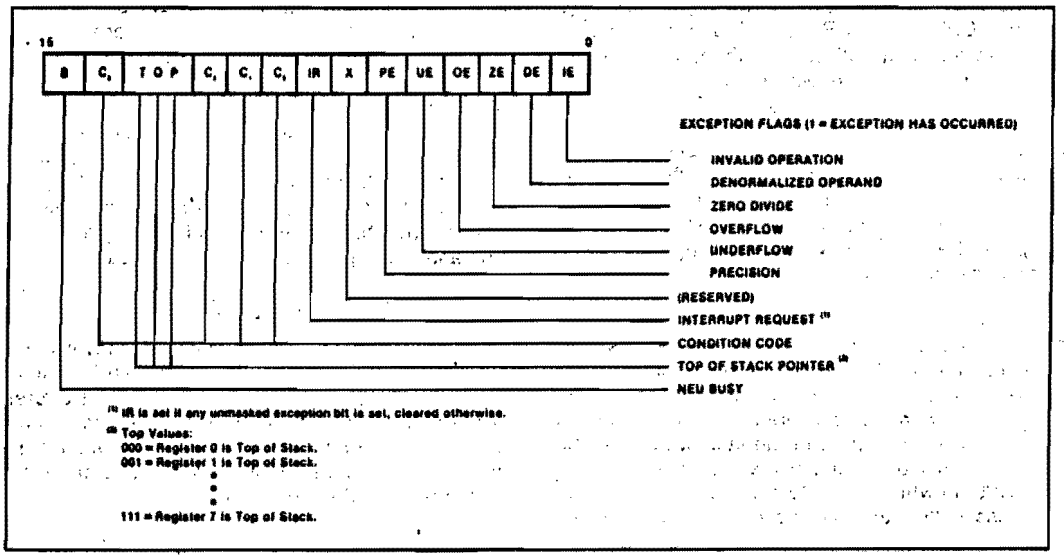


Figure 4. 8087 Status Word

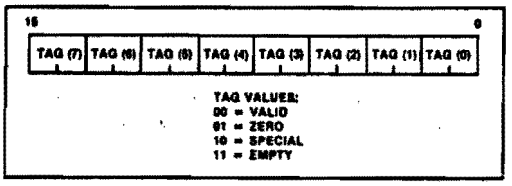


Figure 5. 8087 Tag Word

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.

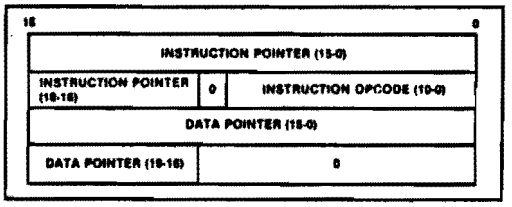


Figure 6. 8087 Instruction and Data Pointers

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, $\infty - \infty$, 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.

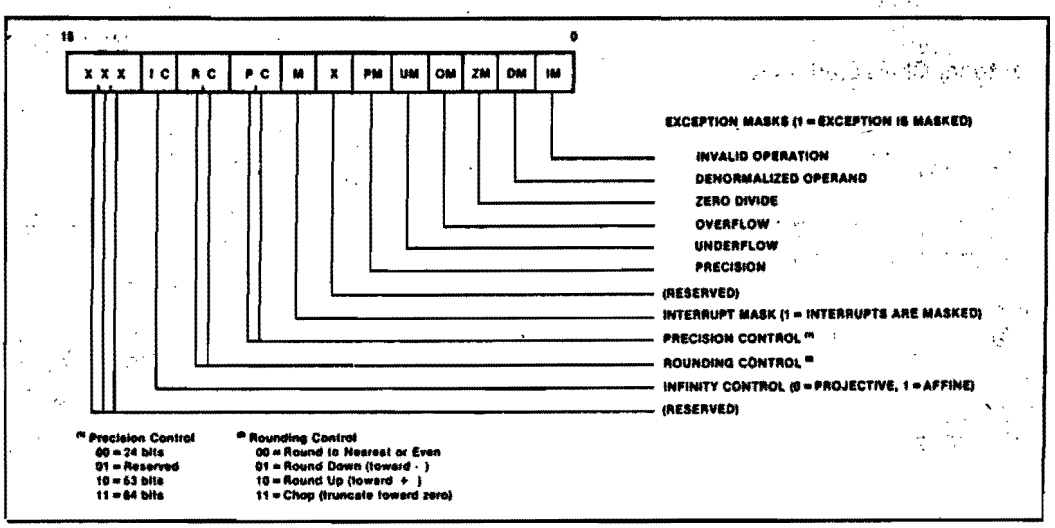


Figure 7. 8087 Control Word

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.

6. **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 emulator 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") iSBC 337 board + host board

Weight — 17.33 grams (.576 oz.)

Electrical Characteristics

DC Power Requirements (8087 only)

$V_{cc} = 5V \pm 5\%$ $I_{cc} = 475$ 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.

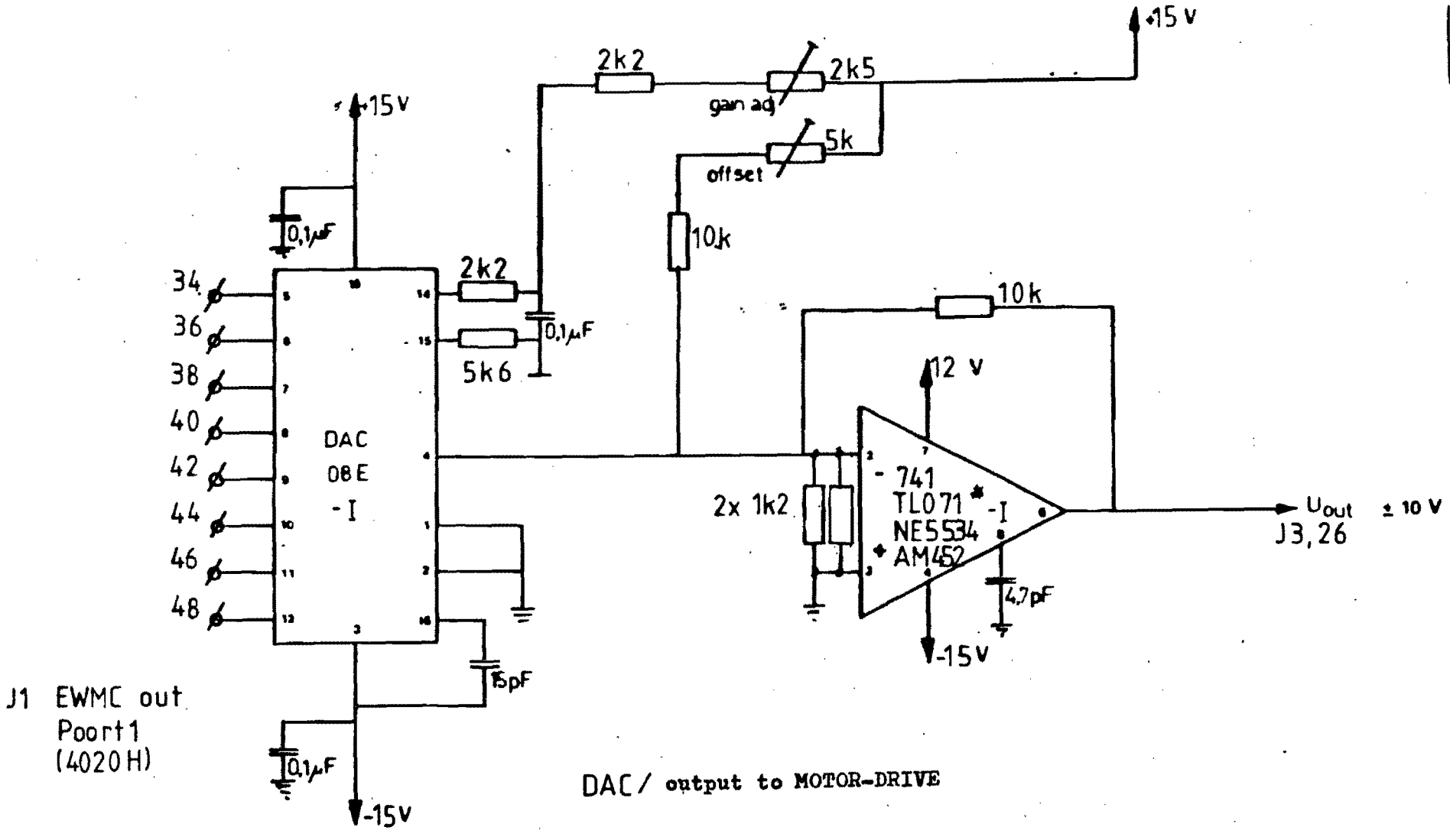
Reference Manual

142687-001 — 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.

ORDERING INFORMATION

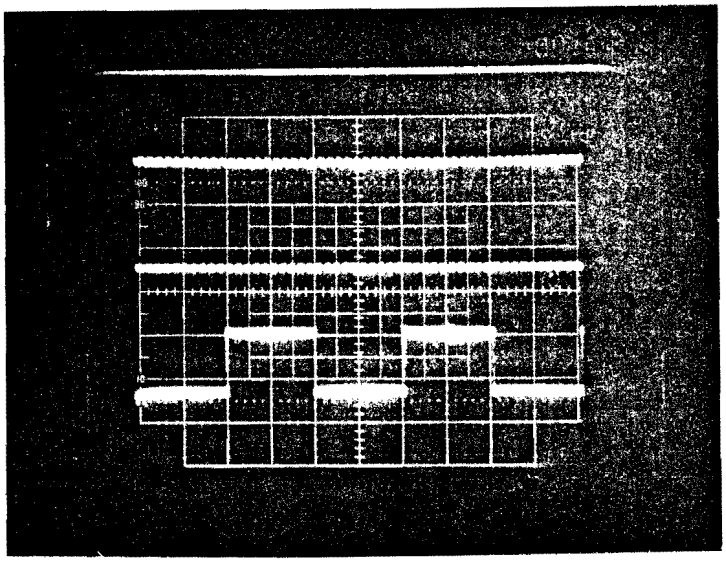
Part Number	Description
SBC 337	MULTIMODULE Numeric Data Processor



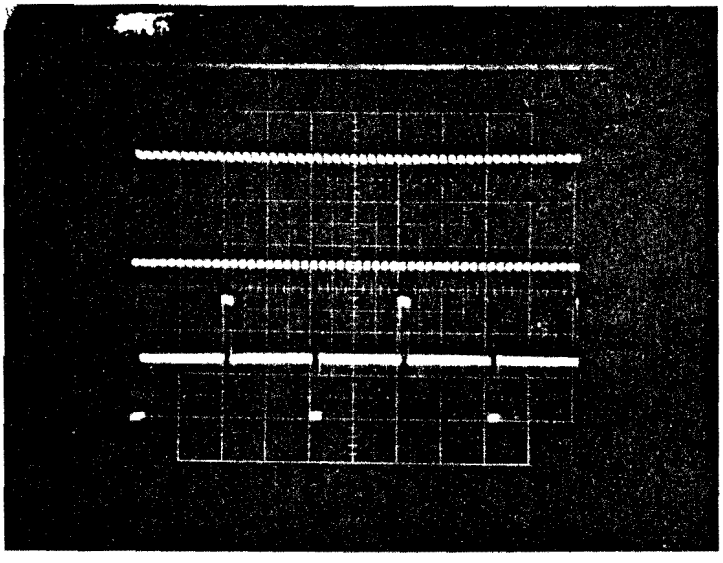
DAC / output to MOTOR-DRIVE

9.11 APP. K OUTPUT RESULTS

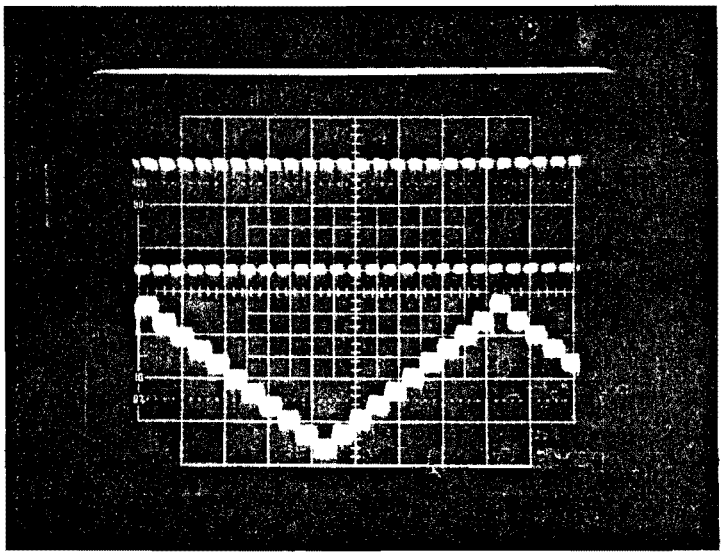
P_EFFECT



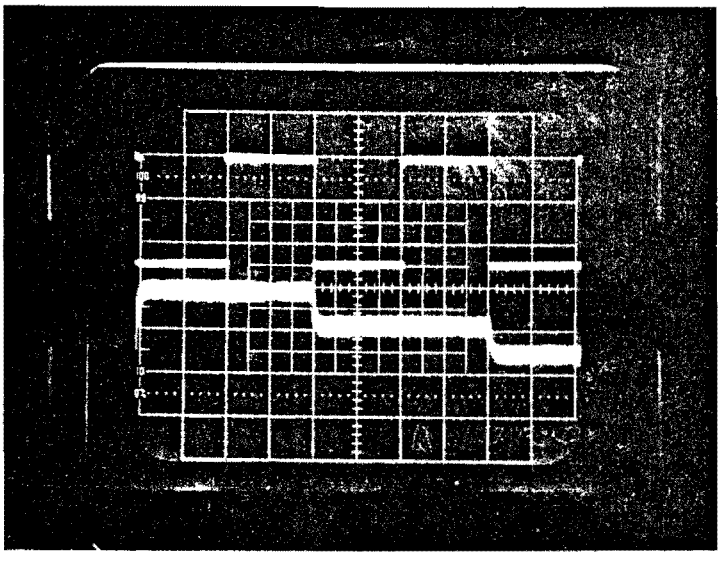
D_EFFECT



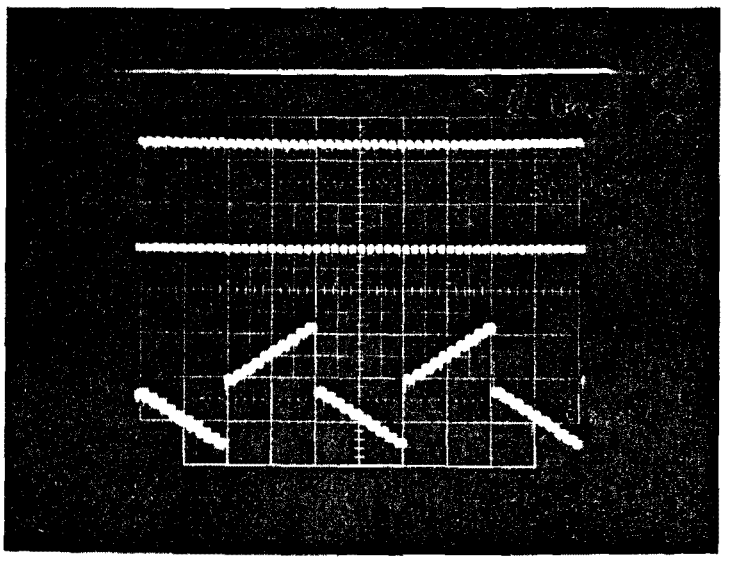
I_EFFECT



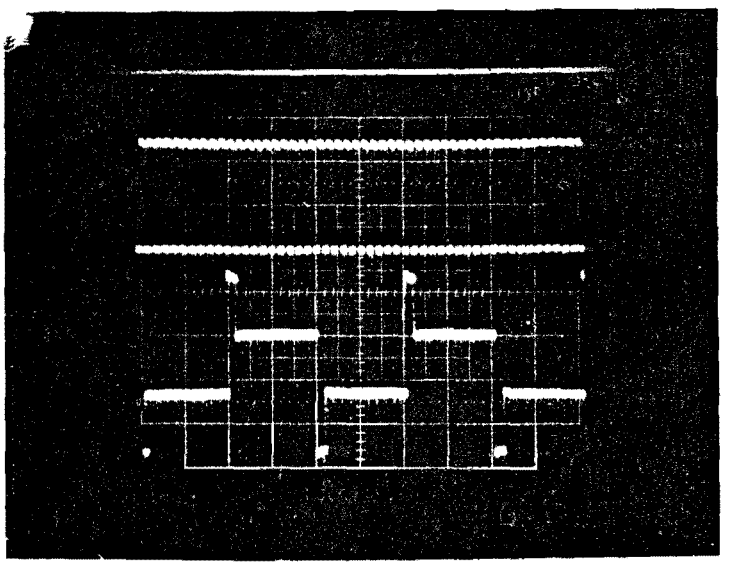
I_EFFECT (EXPANDED)



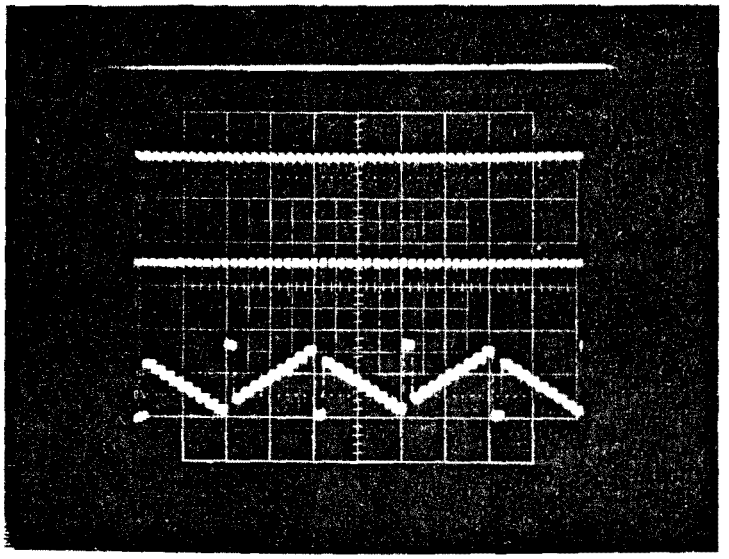
P-I _EFFECTS



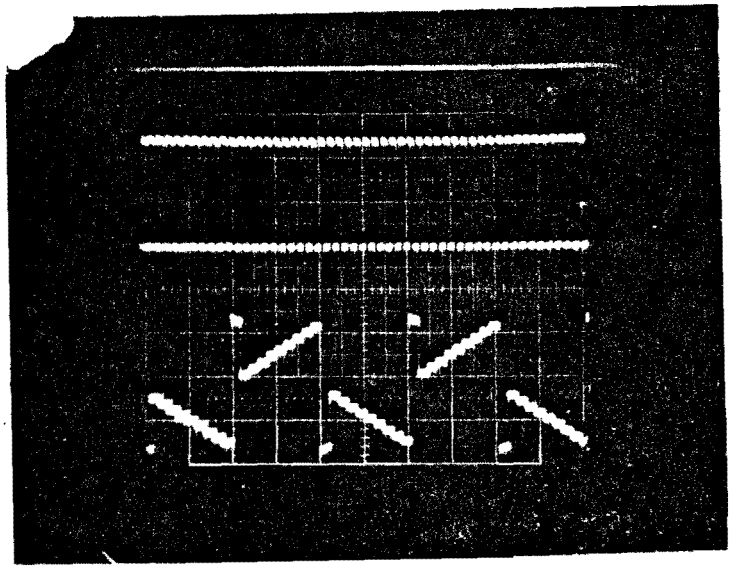
P-D _EFFECTS



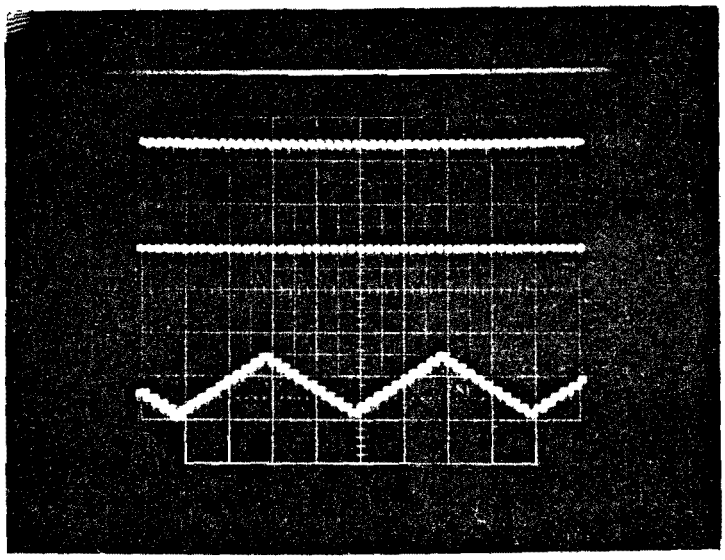
I-D _EFFECTS



P-I-D _EFFECTS



I_EFFECT (PASCAL86)



```
8086/87/88/186 MACRO ASSEMBLER    GETDEC                    05/24/84  PAGE    1


SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE GETDEC
OBJECT MODULE PLACED IN :F1:GETDEC.OBJ
ASSEMBLER INVOKED BY:  ASM86.86 :F1:GETDEC.ASM


LOC  OBJ                  LINE     SOURCE

                             1     NAME    GETDEC
                             2     ;
                             3     ;
                             4     ;*************************************************************
                             5     ;*                                                           *
                             6     ;*      GETDEC LOADS KEYBOARD INPUT IN BX REGISTER AFTER      *
                             7     ;*      CONVERTING THE FIVE FIGURES INTO SIGNED HEXADECIMAL.  *
                             8     ;*      IT ACCEPTS AS FIRST CHARACTER:' ','+','-', OR DIRECTLY*
                             9     ;*      A DIGIT BETWEEN 0 AND 9.                             *
                            10     ;*      NO SIGN IN THE BEGINNING IS TREATED AS '+'.          *
                            11     ;*      EACH ' ' IS TREATED AS '0'.                          *
                            12     ;*      IF ENTERED LESS THAN FIVE DIGITS, THE MISSING ONES ARE*
                            13     ;*      TREATED AS '0'.                                      *
                            14     ;*      IT CONVERTS THE LAST FIVE DIGITS THAT HAVE BEEN TYPED.*
                            15     ;*      RANGE:-32768,+32767(DECIMAL).                        *
                            16     ;*                                                           *
                            17     ;*      IT CALLS: VAL09, DE3, CI, CO.                        *
                            18     ;*                                                           *
                            19     ;*************************************************************
                            20     ;
                            21     ;
                            22     ASSUME  CS:DCODE
                            23     ;
                            24             EXTRN   VAL09:FAR,DE3:FAR,CO:FAR,CI:FAR
----                        25     DCODE   SEGMENT WORD    PUBLIC
                            26             PUBLIC  GETDEC
0000                        27             GETDEC  PROC    FAR
                            28     ;
0000                        29     BEGIN:
0000 50                     30             PUSH    AX
0001 51                     31             PUSH    CX
0002 06                     32             PUSH    ES
0003                        33     MAIN:
0003 B83030                 34             MOV     AX,3030H                ;INIT. AX
0006 BB3030                 35             MOV     BX,3030H                ;INIT. BX
0009 B93000                 36             MOV     CX,0030H                ;INIT. CX
                            37     ;
000C 9A0000----      E      38             CALL    CI                      ;ENTER FIRST CHARACTER
0011 9A0000----      E      39             CALL    CO                      ;PUT IT ON THE SCREEN
0016 8AE8                   40             MOV     CH,AL                   ;MOVE IT INTO CH
                            41     ;
0018 80FD0D                 42             CMP     CH,0DH                  ;IS IT 'CR'?
001B 7465                   43             JE      CRDIR                   ;GO TO LEVEL CRDIR
                            44     ;
001D 80FD2B                 45             CMP     CH,2BH                  ;IS IT '+'?
0020 7446                   46             JE      POSITIVE1               ;GO TO LEVEL POSITIVE1
                            47     ;
0022 80FD2D                 48             CMP     CH,2DH                  ;IS IT '-'?
0025 7448                   49             JE      NEGATIVE1               ;GO TO LEVEL NEGATIVE1
                            50     ;
```

```
LOC  OBJ                  LINE    SOURCE

0027 BA0000                51              MOV     DX,0H           ;
002A 8EC2                  52              MOV     ES,DX           ;CLEAR SIGN FLAG
002C 9A0000----      E     53              CALL    VAL09           ;TEST VALIDITY
0031 8AC5                  54              MOV     AL,CH           ;LOAD IT INTO AL
                           55      ;
0033                       56      BACK1:
0033 50                    57              PUSH    AX              ;SAVE AX
0034 9A0000----      E     58              CALL    CI              ;ENTER NEXT CHARACTER
0039 9A0000----      E     59              CALL    CO              ;PUT IT ON SCREEN
003E 8AE8                  60              MOV     CH,AL           ;PUT IT INTO CH
0040 58                    61              POP     AX              ;RECOVER SAVED AX
0041 80FD0D                62              CMP     CH,0DH          ;IS IT 'CR'?
0044 7448                  63              JE      CONTEND         ;GO TO LEVEL CONTEND
0046 9A0000----      E     64              CALL    VAL09           ;TEST VALIDITY
004B EB2990                65              JMP     SHIFT           ;GO TO SHIFT LEVEL
                           66      ;
004E                       67      BACK2:
004E 9A0000----      E     68              CALL    CI              ;ENTER NEXT CHARACTER
0053 9A0000----      E     69              CALL    CO              ;PUT IT ON SCREEN
0058 8AE8                  70              MOV     CH,AL           ;PUT IT INTO CH
005A 80FD0D                71              CMP     CH,0DH          ;IS IT 'CR'?
005D 7426                  72              JE      CRDIR2          ;GO TO LEVEL CRDIR2
005F 9A0000----      E     73              CALL    VAL09           ;TEST VALIDITY
0064 8AC5                  74              MOV     AL,CH           ;PUT IT INTO AL
0066 EBC8                  75              JMP     BACK1           ;GO TO LEVEL BACK1
                           76      ;
0068                       77      POSITIVE1:
0068 BA0000                78              MOV     DX,0H           ;
006B 8EC2                  79              MOV     ES,DX           ;CLEAR SIGN FLAG
006D EBDF                  80              JMP     BACK2           ;GO TO LEVEL BACK2
                           81      ;
                           82      ;
006F                       83      NEGATIVE1:
006F BA0100                84              MOV     DX,1H           ;
0072 8EC2                  85              MOV     ES,DX           ;SET SIGN FLAG
0074 EBD8                  86              JMP     BACK2           ;GO TO LEVEL BACK2
                           87      ;
0076                       88      SHIFT:
0076 8ACF                  89              MOV     CL,BH           ;SHIFT
0078 8AFB                  90              MOV     BH,BL           ;SHIFT
007A 8ADC                  91              MOV     BL,AH           ;SHIFT
007C 8AE0                  92              MOV     AH,AL           ;SHIFT
007E 8AC5                  93              MOV     AL,CH           ;SHIFT
0080 EBB1                  94              JMP     BACK1           ;GO TO LEVEL BACK1
                           95      ;
0082                       96      CRDIR:
0082 B93000                97              MOV     CX,0030H        ;LOAD CH WITH '(+)',CL WITH 0
0085                       98      CRDIR2:
                           99
0085 B83030               100              MOV     AX,3030H        ;LOAD AH WITH 0
0088 BB3030               101              MOV     BX,3030H        ;LOAD BX WITH 0
008B EB0190               102              JMP     CONTEND         ;GO TO LEVEL CONTEND
                          103      ;
008E                      104      CONTEND:
008E 8CC2                 105              MOV     DX,ES           ;
```

```
LOC  OBJ                  LINE     SOURCE

0090 BAEA                  106              MOV     CH,DL                   ;LOAD CH WITH SIGN FLAG
0092 9A0000----      E     107              CALL    DE3                     ;CALL ROUTINE TO CONVERT
                           108      ;
0097 07                    109              POP     ES
0098 59                    110              POP     CX
0099 58                    111              POP     AX
009A CB                    112              RET                             ;GO BACK TO CALLER
                           113              GETDEC  ENDP
                           114      ;
----                       115      DCODE   ENDS
                           116      ;
                           117      END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
8086/87/88/186 MACRO ASSEMBLER    VAL09                                    05/24/84  PAGE    1


SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE VAL09
OBJECT MODULE PLACED IN :F1:VAL09.OBJ
ASSEMBLER INVOKED BY:  ASM86.86 :F1:VAL09.ASM


LOC  OBJ                      LINE     SOURCE

                                1     NAME    VAL09
                                2     ;
                                3     ;*************************************************************
                                4     ;*                                                           *
                                5     ;*      VAL09 TESTS IF THE CONTENTS OF THE CH REGISTER       *
                                6     ;*      IS BETWEEN 0 AND 9, OR ' ', IN THE LAST CASE,        *
                                7     ;*      IT CONVERTS IT INTO THE ';' ASCII.                   *
                                8     ;*                                                           *
                                9     ;*      IT CALLS:TEXT.                                       *
                               10     ;*                                                           *
                               11     ;*************************************************************
                               12     ;
                               13     ;
                               14     ASSUME  CS:CCODE,DS:DDATA
                               15     ;
----                           16     DDATA   SEGMENT WORD    PUBLIC
                               17     ;
0000 0D                        18     ERR1    DB      0DH,0AH,'ERROR: OUT OF RANGE 0..9',0AH,0DH,0
0001 0A
0002 4552524F523B20
     4F55543204F4620
     52414E47452030
     2E2E39
001A 0A
001B 0D
001C 00
                               19     ;
----                           20     DDATA   ENDS
                               21     ;
                               22     ;
                               23             EXTRN   TEXT:FAR
----                           24     CCODE   SEGMENT WORD    PUBLIC
                               25             PUBLIC  VAL09
                               26     ;
0000                           27             VAL09   PROC    FAR
                               28     ;
0000 BA-----          R        29             MOV     DX,DDATA                ;
0003 8EDA                      30             MOV     DS,DX                   ;DS IS DATA BASE
                               31     ;
0005 8AED                      32             MOV     CH,CH                   ;
0007 80FD20                    33             CMP     CH,20H                  ;IS IT ' '?
000A 741B                      34             JE      EMPTY                   ;GO TO LEVEL EMPTY
000C 80FD30                    35             CMP     CH,30H                  ;IS IT <=0?
000F 7208                      36             JB      ERROR1                  ;GO TO LEVEL ERROR1
0011 80FD39                    37             CMP     CH,39H                  ;IS IT >=9?
0014 7703                      38             JA      ERROR1                  ;GO TO LEVEL ERROR1
0016 EB1190                    39             JMP     SUCC                    ;GO TO LEVEL SUCC
0019                           40     ERROR1:
0019 50                        41             PUSH    AX                      ;SAVE AX
001A 53                        42             PUSH    BX                      ;SAVE BX
```

```
8086/87/88/186 MACRO ASSEMBLER    VAL09                                    05/24/84  PAGE    2


LOC  OBJ                  LINE     SOURCE

001B 8D1E0000       R       43             LEA     BX,ERR1              ;LOAD BX WITH BEGINNING OF TEXT
001F 9A0000----     E       44             CALL    TEXT                 ;GO TO TEXT RUTINE
0024 5B                     45             POP     BX                   ;RECOVER SAVED BX
0025 58                     46             POP     AX                   ;RECOVER SAVED AX
0026 CC                     47             INT     3                    ;GO BACK TO MONITOR
                            48     ;
0027                        49     EMPTY:
0027 B530                   50             MOV     CH,30H          ;CONSIDER ' ' AS 0
                            51     ;
0029                        52     SUCC:
0029 CB                     53             RET                     ;GO BACK TO CALLER
                            54     ;
                            55     VAL09   ENDP
                            56     ;
----                        57     CCODE   ENDS
                            58     ;
                            59     END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE DE3
OBJECT MODULE PLACED IN :F1:DE3.OBJ
ASSEMBLER INVOKED BY:  ASM86.86 :F1:DE3.ASM


LOC  OBJ                    LINE     SOURCE

                              1      NAME    DE3
                              2      ;*******************************************************
                              3      ;  CALL DECHEX FOR                                     *
                              4      ;  BCD<HEXADECIMAAL CONVERTOR                          *
                              5      ;  INPUT: CL,BX,AX REGISTERS, CL MOST SIGNIFICANT      *
                              6      ;         NUMBER, TOTAL 5 NUMBERS, MAX 65535           *
                              7      ;  OUTPUT:BX, BINARY                                   *
                              8      ;                                                      *
                              9      ;*******************************************************
                             10      ;
                             11      ASSUME  CS:CCODE, DS:DDATA
                             12      ;
----                         13      DDATA   SEGMENT WORD    PUBLIC
                             14      ;
0000 0D                      15      TEXTER  DB      0DH,0AH,'ERROR. 1ST CHAR. IS NOT:0+,1-',0AH,0DH,0
0001 0A
0002 4552524F522E20
     3153542043484
     522E204953204E
     4F543A302B2C31
     2D
001F 0A
0020 0D
0021 00
                             16      ;
----                         17      DDATA   ENDS
                             18      ;
                             19              EXTRN   TEXT:FAR,DECHEX:FAR
----                         20      CCODE   SEGMENT WORD    PUBLIC
                             21              PUBLIC  DE3
0000                         22              DE3     PROC    FAR
                             23      ;
0000                         24      BEGIN:
0000 BA----          R       25              MOV     DX,DDATA        ;DS IS DATA BASE
0003 8EDA                    26              MOV     DS,DX
0005                         27      MAIN:
                             28      ;
                             29      ;
0005 2C30                    30              SUB     AL,30H          ;
0007 80EC30                  31              SUB     AH,30H          ;
000A 80EB30                  32              SUB     BL,30H          ;
000D 80EF30                  33              SUB     BH,30H          ;
0010 80E930                  34              SUB     CL,30H          ;PUT AL,AH,BL,BH,CL IN ORDER TO GO TO SUBROUTINE
0013 9A0000----      E       35              CALL    DECHEX          ;GO TO SUBROUTINE DECHEX (BCD->HEX/SIGNED)
                             36      ;
0018 8AED                    37              MOV     CH,CH           ;
001A 80FD00                  38              CMP     CH,0H           ;IS IT '+'?
001D 740A                    39              JE      POSITIVE        ;GO TO LEVEL POSITIVE
001F 80FD01                  40              CMP     CH,1H           ;IS IT '-'?
0022 7403                    41              JE      NEGATIVE        ;GO TO LEVEL NEGATIVE
```

```
8086/87/88/186 MACRO ASSEMBLER    DE3                                      05/24/84  PAGE    2


LOC  OBJ                  LINE     SOURCE

0024 EB0690                 42             JMP     ERROR           ;IF NOT, GO TO LEVEL ERROR
0027                        43     NEGATIVE:
0027 F7DB                   44             NEG     BX,             ;COMPLEMENT THE NUMBER
0029                        45     POSITIVE:
0029 EB0F90                 46             JMP     BACK            ;GO TO LEVEL BACK
002C                        47     ERROR:
002C 50                     48             PUSH    AX              ;SAVE AX
002D 53                     49             PUSH    BX              ;SAVE BX
002E 8D1E0000       R       50             LEA     BX,TEXTER       ;LOAD ORIGIN OF TEXT
0032 9A0000----     E       51             CALL    TEXT            ;GO TO SUBROUTINE TEXT
0037 5B                     52             POP     BX              ;RECOVER SAVED BX
0038 58                     53             POP     AX              ;RECOVER SAVED AX
0039 CC                     54             INT     3               ;GO BACK TO MONITOR
003A                        55     BACK:
003A CB                     56             RET                     ;GO BACK TO CALLER
                            57             DE3     ENDP
----                        58             CCODE   ENDS
                            59     ;
                            60     END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
8086/87/88/186 MACRO ASSEMBLER    DECHEX                                   05/24/84  PAGE    1


SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE DECHEX
OBJECT MODULE PLACED IN :F1:DECHEX.OBJ
ASSEMBLER INVOKED BY:  ASM86.86 :F1:DECHEX.ASM


LOC  OBJ                  LINE     SOURCE

                             1     NAME    DECHEX
                             2     ;*******************************************************
                             3     ;
                             4     ;  BCD<HEXADECIMAAL CONVERTOR
                             5     ;  INPUT: CL,BX,AX REGISTERS, CL MOST SIGNIFICANT
                             6     ;         NUMBER, TOTAL 5 NUMBERS, MAX 65535
                             7     ;  OUTPUT:BX, BINARY
                             8     ;
                             9     ;*******************************************************
                            10     ;
                            11     ASSUME  CS:CCODE, DS:DDATA
                            12     ;
----                        13     DDATA   SEGMENT WORD    PUBLIC
                            14     ;
0000 0A                     15     TENS    DB      10
0001 64                     16     HUNDR   DB      100
0002 E803                   17     THOUS   DW      1000
0004 1027                   18     TENTHD  DW      10000
                            19     ;
----                        20     DDATA   ENDS
                            21     ;
----                        22     CCODE   SEGMENT WORD    PUBLIC
                            23             PUBLIC  DECHEX
0000                        24             DECHEX  PROC    FAR
                            25     ;
0000                        26     BEGIN:
0000 BA----         R       27             MOV     DX,DDATA        ;DS IS DATA BASE
0003 8EDA                   28             MOV     DS,DX
0005                        29     MAIN:
0005 BA0000                 30             MOV     DX,0H           ;CLEAR DX
0008 8AD0                   31             MOV     DL,AL           ;UNITS TO OUTPUT
000A 8AC4                   32             MOV     AL,AH
000C F6260000       R       33             MUL     DS:TENS         ;CALCULATE THE TENTHS
0010 03D0                   34             ADD     DX,AX           ;AND ADD TO DX
0012 8AC3                   35             MOV     AL,BL           ;NOW THE HUNDREDS
0014 F6260100       R       36             MUL     DS:HUNDR
0018 03D0                   37             ADD     DX,AX
001A 8AC7                   38             MOV     AL,BH           ;NOW THE THOUSENDS
001C B400                   39             MOV     AH,0H           ;CLEAR AH REGISTER
001E 8BDA                   40             MOV     BX,DX           ;RESULT NOW TO BX
0020 F7260200       R       41             MUL     DS:THOUS
0024 03D8                   42             ADD     BX,AX
0026 8AC1                   43             MOV     AL,CL           ;TENTHOUSENDS
0028 B400                   44             MOV     AH,0H
002A F7260400       R       45             MUL     DS:TENTHD
002E 03D8                   46             ADD     BX,AX           ;READY
0030 CB                     47             RET                     ;BACK TO CALLER
                            48             DECHEX  ENDP
                            49     ;
----                        50     CCODE   ENDS
```

```
LOC  OBJ                  LINE     SOURCE

                            51     ;
                            52     END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE TEXT
OBJECT MODULE PLACED IN :F1:TEXT.OBJ
ASSEMBLER INVOKED BY:  ASM86.86 :F1:TEXT.ASM


LOC  OBJ                  LINE     SOURCE

                             1     NAME TEXT
                             2     ;
                             3     ;*************************************************************
                             4     ;*                                                           *
                             5     ;*       TEXT IS A ROUTINE THAT WRITES A STRING TO THE       *
                             6     ;*       SCREEN. THE ADDRESS OF THE BEGINNING OF THE STRING  *
                             7     ;*       MUST BE IN THE BX REGISTER. THE LAST CHARACTER MUST  *
                             8     ;*       BE A 0.                                             *
                             9     ;*                                                           *
                            10     ;*************************************************************
                            11     ;
                            12     ASSUME  CS:CCODE
                            13     ;
                            14             EXTRN   CO:NEAR
----                        15     CCODE   SEGMENT WORD    PUBLIC
                            16             PUBLIC  TEXT
0000                        17             TEXT    PROC    NEAR
                            18     ;
0000                        19     BEGIN:
0000 BF0000                 20             MOV     DI,0
0003                        21     LOOPTX:
0003 8BC7                   22             MOV     AX,DI           ; INDEX-ADDRESS OF CHAR. TO AX
0005 D7                     23             XLATB                   ; CHAR. TO AL
0006 24FF                   24             AND     AL,0FFH         ; 0?
0008 7406                   25             JZ      UIT
000A E80000         E       26             CALL    CO              ; CHAR. TO CONSOLE
000D 47                     27             INC     DI              ; INDEX INCREMENTED
000E EBF3                   28             JMP     LOOPTX          ; TO FOLLOWING CHARACTER
0010                        29     UIT:
0010 C3                     30             RET
                            31             TEXT    ENDP
----                        32     CCODE   ENDS
                            33     END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
8086/87/88/186 MACRO ASSEMBLER    CI                                    05/24/84  PAGE    1

SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE CI
OBJECT MODULE PLACED IN :F1:CI.OBJ
ASSEMBLER INVOKED BY:  ASM86.86 :F1:CI.ASM


LOC  OBJ                  LINE     SOURCE

                             1     NAME    CI
                             2     ;*******************************************************
                             3     ;*                                                     *
                             4     ;*   CI WRITES ONE ASCII CHARACTER FROM THE USART      *
                             5     ;*   TO THE AL REGISTER                                *
                             6     ;*                                                     *
                             7     ;*******************************************************
                             8     ;
                             9     ASSUME  CS:CCODE
                            10     ;
----                        11     CCODE   SEGMENT WORD    PUBLIC
                            12             PUBLIC  CI
0000                        13             CI      PROC    FAR
                            14     ;
0000                        15     STATUS:
0000 E4DA                   16             IN      AL,0DAH         ;GET THE USART'S STATUS
0002 2402                   17             AND     AL,2            ;TEST FOR RXRDY
0004 74FA                   18             JZ      STATUS          ;
0006 E4D8                   19             IN      AL,0D8H         ;GET THE CHARACTER
0008 CB                     20             RET                     ;RETURN TO CALLER
                            21             CI      ENDP
                            22     ;
----                        23     CCODE   ENDS
                            24     ;
                            25     END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
8086/87/88/186 MACRO ASSEMBLER    CO                                    05/24/84  PAGE    1


SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE CO
OBJECT MODULE PLACED IN :F1:CO.OBJ
ASSEMBLER INVOKED BY:  ASM86.86 :F1:CO.ASM


LOC  OBJ                  LINE     SOURCE

                             1     NAME    CO
                             2     ;*************************************************************
                             3     ;*                                                           *
                             4     ;*    CO SENDS ONE CHARACTER THAT IS IN AL TO THE USART      *
                             5     ;*                                                           *
                             6     ;*************************************************************
                             7     ;
                             8     ASSUME  CS:CCODE
                             9     ;
----                        10             CCODE   SEGMENT WORD    PUBLIC
                            11             PUBLIC  CO
0000                        12             CO      PROC    NEAR
0000                        13     BEGIN:
0000 50                     14             PUSH    AX
0001 E4DA                   15     STATUS: IN      AL,0DAH          ;STATUS OPHALEN
0003 2401                   16             AND     AL,1             ;TEST VOOR TX GEREED
0005 74FA                   17             JZ      STATUS
0007 58                     18             POP     AX
0008 E6D8                   19             OUT     0D8H,AL          ;ZEND CHARACTER WEG
000A C3                     20             RET
                            21             CO      ENDP
----                        22     CCODE   ENDS
                            23     END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
8086/87/88/186 MACRO ASSEMBLER    DECOUT                                   05/24/84  PAGE    1


SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE DECOUT
OBJECT MODULE PLACED IN :F1:DECOUT.OBJ
ASSEMBLER INVOKED BY:  ASM86.86 :F1:DECOUT.ASM


LOC  OBJ                  LINE     SOURCE

                            1     ;
                            2     NAME    DECOUT
                            3      ;*************************************************
                            4      ;*                                               *
                            5      ;* OUTPUTS THE CONTENTS OF BX REGISTER, AFTER *
                            6      ;* CONVERTING THE SIGNED NUMBER INTO BCD CODE *
                            7      ;*                                               *
                            8      ;*************************************************
                            9     ;
                           10     ASSUME  CS:CCODE
                           11     ;
                           12             EXTRN   TEXT0:FAR,HEXBCD:FAR
----                       13     CCODE   SEGMENT WORD    PUBLIC
                           14             PUBLIC  DECOUT
                           15     ;
0000                       16             DECOUT  PROC    FAR
                           17     ;
0000                       18     MAIN:
0000 50                    19             PUSH    AX
0001 51                    20             PUSH    CX
0002 8BDB                  21             MOV     BX,BX           ;
0004 F6C780                22             TEST    BH,10000000B    ;TEST SIGN
0007 7805                  23             JS      CHANGE          ;GO TO CHANGE SIGN
0009 B52B                  24             MOV     CH,2BH          ;PUT '+'
000B EB0590                25             JMP     LEV1            ;PRINT '+'
000E B52D                  26     CHANGE: MOV     CH,2DH          ;PUT '-'
0010 F7DB                  27             NEG     BX              ;COMPLEMENT THE NUMBER
0012 50                    28     LEV1:   PUSH    AX              ;
0013 8AC5                  29             MOV     AL,CH           ;
0015 9A0000----      E     30             CALL    TEXT0           ;PRINT
001A 58                    31             POP     AX              ;
001B 9A0000----      E     32             CALL    HEXBCD          ;CONVERT HEX->BCD
                           33
0020 50                    34             PUSH    AX              ;TO USE WITH UNITS
0021 B500                  35             MOV     CH,0H           ;INITIALIZE FLAG=0
0023 8AC1                  36             MOV     AL,CL           ;BEGINNING WITH TENTHOUSANS
0025 3C00                  37             CMP     AL,0H           ;CL:0
0027 7507                  38             JNE     ASCIICL         ;IF CL=0
0029 B020                  39             MOV     AL,20H          ;WRITE ' '
002B B501                  40             MOV     CH,1H           ;SET THE FLAG=1
002D EB0590                41             JMP     COCL            ;PUT CL ON THE SCREEN
0030 0430                  42     ASCIICL:ADD     AL,30H          ;CONVERT IT INTO ASCII
0032 B500                  43             MOV     CH,0H           ;RESET THE FLAG=0
0034 9A0000----      E     44     COCL:   CALL    TEXT0           ;PUT CL ON THE SCREEN
                           45     ;
0039 8AC7                  46             MOV     AL,BH           ;BEGINNING WITH THOUSANDS
003B 3C00                  47             CMP     AL,0H           ;BH:0
003D ....                  ..             ....    ASCIIBH         ;IF BH=0
003F F6C501                49             TEST    CH,1H           ;TEST IF THE OTHERS ARE ALSO 0
0042 7407                  50             JE      ASCIIBH         ;BH=0, BUT NOT THE OTHERS
```

```
LOC  OBJ                  LINE    SOURCE

0044 B020                   51            MOV     AL,20H          ;WRITE ' '
0046 B501                   52            MOV     CH,1H           ;SET THE FLAG=1
0048 EB0590                 53            JMP     COBH            ;PUT BH ON THE SCREEN
004B 0430                   54    ASCIIBH:ADD     AL,30H          ;CONVERT IT INTO ASCII
004D B500                   55            MOV     CH,0H           ;RESET THE FLAG=0
004F 9A0000----      E      56    COBH:   CALL    TEXT0           ;PUT BH ON THE SCREEN
                            57    ;
0054 8AC3                   58            MOV     AL,BL           ;BEGINNING WITH HUNDREDS
0056 3C00                   59            CMP     AL,0H           ;BL:0
0058 750C                   60            JNE     ASCIIBL         ;IF BL=0
005A F6C501                 61            TEST    CH,1H           ;TEST IF THE OTHERS ARE ALSO 0
005D 7407                   62            JE      ASCIIBL         ;BL=0, BUT NOT THE OTHERS
005F B020                   63            MOV     AL,20H          ;WRITE ' '
0061 B501                   64            MOV     CH,1H           ;SET THE FLAG=1
0063 EB0590                 65            JMP     COBL            ;PUT BL ON THE SCREEN
0066 0430                   66    ASCIIBL:ADD     AL,30H          ;CONVERT IT INTO ASCII
0068 B500                   67            MOV     CH,0H           ;RESET THE FLAG=0
006A 9A0000----      E      68    COBL:   CALL    TEXT0           ;PUT BL ON THE SCREEN
                            69    ;
006F 8AC4                   70            MOV     AL,AH           ;BEGINNING WITH TENTHS
0071 3C00                   71            CMP     AL,0H           ;AH:0
0073 750C                   72            JNE     ASCIIAH         ;IF AH=0
0075 F6C501                 73            TEST    CH,1H           ;TEST IF THE OTHERS ARE ALSO 0
0078 7407                   74            JE      ASCIIAH         ;AH=0, BUT NOT THE OTHERS
007A B020                   75            MOV     AL,20H          ;WRITE ' '
007C B501                   76            MOV     CH,1H           ;SET THE FLAG=1
007E EB0590                 77            JMP     COAH            ;PUT AH ON THE SCREEN
0081 0430                   78    ASCIIAH:ADD     AL,30H          ;CONVERT IT INTO ASCII
0083 B500                   79            MOV     CH,0H           ;RESET THE FLAG=0
0085 9A0000----      E      80    COAH:   CALL    TEXT0           ;PUT AH ON THE SCREEN
                            81    ;
008A 58                     82            POP     AX              ;RECOVER AL VALUE
008B 8AC0                   83            MOV     AL,AL           ;BEGINNING WITH UNITS
008D 0430                   84            ADD     AL,30H          ;CONVERT IT INTO ASCII
008F B500                   85            MOV     CH,0H           ;RESET THE FLAG=0
0091 9A0000----      E      86            CALL    TEXT0           ;PUT AL ON THE SCREEN
0096 2C30                   87            SUB     AL,30H          ;RESTORE AL
0098 59                     88            POP     CX
0099 58                     89            POP     AX
009A CB                     90            RET                     ;GO BACK TO CALLER
                            91            DECOUT  ENDP
                            92    ;
----                        93    CCODE   ENDS
                            94    ;
                            95    END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
8086/87/88/186 MACRO ASSEMBLER    TEXT0                                    05/24/84  PAGE    1


SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE TEXT0
OBJECT MODULE PLACED IN :F1:TEXT0.OBJ
ASSEMBLER INVOKED BY:  ASM86.86 :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
                                9     ASSUME  CS:CCODE,DS:DDATA
                               10
----                           11     DDATA   SEGMENT WORD    PUBLIC
                               12     ;
0000 0D                        13     TEXTER  DB      0DH,0AH,'IT IS AN ERROR; OUT OF RANGE: 0..9',0AH,0DH,0
0001 0A
0002 49542049532041
     4E204552524F52
     3B204F5554204F
     462052414E4745
     3A20302E2E39
0024 0A
0025 0D
0026 00
                               14     ;
----                           15     DDATA   ENDS
                               16     ;
                               17             EXTRN   TEXT:FAR,CO:FAR
----                           18     CCODE   SEGMENT WORD    PUBLIC
                               19             PUBLIC  TEXT0
0000                           20             TEXT0   PROC    FAR
0000 3C20                      21             CMP     AL,20H          ;TEST IF 0
0002 7428                      22             JE      CONT            ;
0004 3C2B                      23             CMP     AL,2BH          ;TEST IF '+'
0006 7424                      24             JE      CONT            ;
0008 3C2D                      25             CMP     AL,2DH          ;TEST IF '-'
000A 7420                      26             JE      CONT            ;
000C 3C30                      27             CMP     AL,30H          ;TEST IF <0
000E 7207                      28             JB      ERROR           ;
0010 3C39                      29             CMP     AL,39H          ;TEST IF >9
0012 7703                      30             JA      ERROR           ;
0014 EB1690                    31             JMP     CONT            ;
0017 50                        32     ERROR:  PUSH    AX              ;SAVE
0018 53                        33             PUSH    BX              ;SAVE
0019 BB----          R         34             MOV     BX,DDATA        ;INIT. DS
001C 8EDB                      35             MOV     DS,BX           ;
001E 8D1E0000        R         36             LEA     BX,TEXTER       ;INIT. TEXT
0022 9A0000----      E         37             CALL    TEXT            ;
0027 5B                        38             POP     BX              ;RECOVER
0028 58                        39             POP     AX              ;RECOVER
0029 EB0690                    40             JMP     ENDLEVEL        ;
002C 9A0000----      E         41     CONT:   CALL    CO              ;
```

```
8086/87/88/186 MACRO ASSEMBLER    TEXT0                                    05/24/84  PAGE    2


LOC  OBJ                  LINE      SOURCE

0031                        42      ENDLEVEL:                ;
0031 CB                     43              RET              ;GO TO CALLER
                            44              TEXT0   ENDP     ;END OF PROCEDURE
----                        45      CCODE   ENDS
                            46      ;
                            47      END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE HEXBCD
OBJECT MODULE PLACED IN :F1:HEXBCD.OBJ
ASSEMBLER INVOKED BY:  ASM86.86 :F1:HEXBCD.ASM


LOC  OBJ                  LINE     SOURCE

                             1     NAME HEXBCD
                             2
                             3     ;*************************************************************
                             4     ;*                                                           *
                             5     ;*  HEXADECIMAL -> BCD CONVERTOR                             *
                             6     ;*  INPUT :BX, BINARY                                        *
                             7     ;*  NUMBER, TOTAL 4 HEXA, MAX FFFF                           *
                             8     ;*  OUTPUT:CL,BX,AX REGISTERS, CL MOST SIGNIFICANT           *
                             9     ;*                                                           *
                            10     ;*************************************************************
                            11     ;
                            12     ASSUME  CS:CCODE,DS:DDATA
                            13     ;
----                        14             DDATA   SEGMENT WORD    PUBLIC
                            15     ;
0000 0A00                   16             TEN     DW      10
                            17     ;
----                        18             DDATA   ENDS
                            19     ;
                            20     ;
----                        21             CCODE   SEGMENT WORD    PUBLIC
                            22                     PUBLIC  HEXBCD
0000                        23                     HEXBCD  PROC    FAR
                            24     ;
0000 BA----         R       25     HEXDEC: MOV     DX,DDATA
0003 8EDA                   26             MOV     DS,DX           ;DS IS DATA BASE
                            27     ;
0005 8BC3                   28     MAIN:   MOV     AX,BX           ;AX IS PART OF THE DIVIDEND
0007 BA0000                 29             MOV     DX,0H           ;DX IS 0, THEN DIVIDEND=DX:AX=0000:BX
000A F7360000       R       30             DIV     DS:TEN          ;DIVIDE BY 10, MODULO IN DX, QUOTIENT IN AX
000E 8ACA                   31             MOV     CL,DL           ;DL=UNITS, TEMP. STORE IN CL
0010 BA0000                 32             MOV     DX,0H           ;NEW DIVIDEND=0000:BX/10
0013 F7360000       R       33             DIV     DS:TEN          ;DIVIDE BY 10, MODULO IN DX, QUOTIENT IN AX
0017 8AEA                   34             MOV     CH,DL           ;DL=TENTHS, TEMP.STORE IN CH
0019 BA0000                 35             MOV     DX,0H           ;NEW DIVIDEND=0000:BX/100
001C F7360000       R       36             DIV     DS:TEN          ;DIVIDE BY 10, MODULO IN DX, QUOTIENT IN AX
0020 8ADA                   37             MOV     BL,DL           ;BL=HUNDREDS
0022 BA0000                 38             MOV     DX,0H           ;NEW DIVIDEND=0000:BX/1000
0025 F7360000       R       39             DIV     DS:TEN          ;DIVIDE BY 10, MODULO IN DX, QUOTIENT IN AX
0029 8AFA                   40             MOV     BH,DL           ;BH=THOUSANDS
002B 86C1                   41             XCHG    AL,CL           ;AL=UNITS, CL=TENTHOUSANDS
002D 86E5                   42             XCHG    AH,CH           ;AH=TENTHS, CH=0
002F CB                     43             RET
                            44             HEXBCD  ENDP
----                        45             CCODE   ENDS
                            46     ;
                            47     END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE INITIO
OBJECT MODULE PLACED IN :F1:INITIO.OBJ
ASSEMBLER INVOKED BY:  ASM86.86 :F1:INITIO.ASM


LOC  OBJ                  LINE     SOURCE

                             1     NAME    INITIO
                             2     ;
                             3     ;*************************************************
                             4     ;
                             5     ;   RECTANGULAR INTEGRATION
                             6     ;
                             7     ;*************************************************
                             8     ;
----                         9     INTPTR  SEGMENT AT  - - 0H
0088                        10             ORG     88H
0088 4900-----       R      11     TYPE2   DD      TIMINT
                            12     ;
----                        13     INTPTR  ENDS
                            14     ;
                            15     ASSUME  CS:CCODE, DS:DDATA
                            16     ;
----                        17     DDATA   SEGMENT WORD    PUBLIC
                            18     ;
0000 ????                   19     STIN    DW      ?
0002 ????                   20     SOM     DW      ?
0004 ????                   21     INTT    DW      ?
0006 ????                   22     SAMT    DW      ?
                            23     ;
----                        24     DDATA   ENDS
                            25     ;
----                        26     CCODE   SEGMENT WORD    PUBLIC
                            27     ;
0000                        28     INIT:
0000 FA                     29             CLI                     ;DISABLE INTERRUPTS
0001 B017                   30             MOV     AL,17H
0003 E6C0                   31             OUT     0C0H,AL         ;ICW1 TO PIC
0005 B020                   32             MOV     AL,20H          ;VECTOR BASE IS 4*20H
0007 E6C2                   33             OUT     0C2H,AL         ;ICW2 TO PIC
0009 B01F                   34             MOV     AL,1FH
000B E6C2                   35             OUT     0C2H,AL         ;ICW4 TO PIC
000D B0FB                   36             MOV     AL,NOT(4)       ;ALLOW INT2(2^2)
000F E6C2                   37             OUT     0C2H,AL         ;OCW TO PIC, ALLOW INT2
0011 B036                   38             MOV     AL,36H
0013 E6D6                   39             OUT     0D6H,AL         ;INIT TIMER 0 FOR 1MS COUNT
0015 B07B                   40             MOV     AL,07BH
0017 E6D0                   41             OUT     0D0H,AL         ;LSB TIME (123D CYCLES= 100 US)
0019 B000                   42             MOV     AL,0
001B E6D0                   43             OUT     0D0H,AL         ;MSB TIME
001D B90A00                 44             MOV     CX,10D          ;COUNT FOR 1MSEC
0020 B080                   45             MOV     AL,80H          ;INITIATE 8255 FOR ALL PORTS TO
0022 E6CE                   46             OUT     0CEH,AL         ;BE OUTPUT PORT
0024 BB-----         R      47             MOV     BX,DDATA        ;INITIATE DDATA SEGMENT
0027 8EDB                   48             MOV     DS,BX
0029 C70600000000    R      49             MOV     STIN,0          ;INITIATING OF INTEGRATOR
002F C70602000000    R      50             MOV     SOM,0
```

```
LOC  OBJ                  LINE     SOURCE

0035 C70604000500   R       51             MOV     INT1,5
003B C70606000100   R       52             MOV     SANT,1
0041 B80A00                 53             MOV     AX,10
0044 FB                     54             STI                     ;ALLOW INTERRUPTS
                            55     ;
0045                        56     MAIN:
0045 E6CC                   57             OUT     0CCH,AL         ;CONTENTS OF AL TO OUTPUT PORT C
0047 EBFC                   58             JMP     MAIN            ;MAIN IS MAINPROGRAMME TO BE INTERRUPTED
                            59     ;
0049                        60     TIMINT  PROC
0049 E205                   61             LOOP    INTEG
004B F7DB                   62             NEG     AX              ;INVERT OUTPUT FOR SQUARE WAVE
004D B90A00                 63             MOV     CX,10D          ;LOAD COUNT AGAIN
0050                        64     INTEG:
0050 50                     65             PUSH    AX
0051 E80500                 66             CALL    RECTIN
0054 E6C8                   67             OUT     0C8H,AL         ;INTEGRATOR OUTPUT TO PORT A
0056 58                     68             POP     AX              ;RESTORE AX
0057 FB                     69             STI
0058 CF                     70             IRET                    ;BACK TO MAIN
                            71     TIMINT  ENDP
                            72     ;
0059                        73     RECTIN  PROC
                            74     ;
                            75     ;
                            76
                            77
                            78     ;
                            79
                            80     ;
0059                        81     BEGINT:
0059 8B1E0000       R       82             MOV     BX,STIN
                            83 +1
005D 03C3                   84 +1          ADD     AX,BX
005F 7108                   85 +2          JNO     UIT01
0061 7906                   86 +2          JNS     NEGAT00
0063 B8FF7F                 87 +1          MOV     AX,7FFFH
0066 EB0490                 88 +2          JMP     UIT01
0069                        89 +2  NEGAT00:
0069 B80080                 90 +1          MOV     AX,8000H
006C                        91 +2  UIT01:
                            92 +1
006C A30000         R       93             MOV     STIN,AX
006F 8B1E0400       R       94             MOV     BX,INTT
                            95 +1
0073 0BC0                   96 +1          OR      AX,AX
0075 9C                     97 +1          PUSHF
0076 7902                   98 +2          JNS     DEEL02
0078 F7D8                   99 +1          NEG     AX
007A                       100 +2  DEEL02:
007A BA0000                101 +1          MOV     DX,0
007D F7F3                  102 +1          DIV     BX
007F D1E2                  103 +1          SHL     DX,1
0081 7007                  104 +2          JO      INCR03
0083 3BD3                  105 +1          CMP     DX,BX
```

```
LOC  OBJ                  LINE     SOURCE

0085 7303                  106 +2           JAE     INCR03
0087 EB0290                107 +2           JMP     UITS04
008A                       108 +2  INCR03:
008A 40                    109 +1           INC     AX,
008B                       110 +2  UITS04:
008B 9D                    111 +1           POPF
008C 7902                  112 +2           JNS     UIT05
008E F7D8                  113 +1           NEG     AX
0090                       114 +2  UIT05:
                           115 +1
0090 8B1E0600       R      116              MOV     BX,SAMT
                           117 +1
0094 F7EB                  118 +1           IMUL    BX
0096 710F                  119 +2           JNO     UIT07
0098 81E20080              120 +1           AND     DX,8000H
009C 7406                  121 +2           JZ      POSIT06
009E B80080                122 +1           MOV     AX,8000H
00A1 EB0490                123 +2           JMP     UIT07
00A4                       124 +2  POSIT06:
00A4 B8FF7F                125 +1           MOV     AX,7FFFH
00A7                       126 +2  UIT07:
                           127 +1
00A7 C3                    128              RET
                           129     RECTIN   ENDP
-----                      130     CCODE    ENDS
                           131     END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
;
;
;*****************************************************************************
;*                                                                           *
;* THIS PROGRAM PERFORMS A P.I.D. CONTROL ON AN INPUT SIGNAL EPSILON.        *
;* THE INSTBYTE SELECTS THE MODE:0-> OUTPUT=0                                *
;*                               1->DIFFER.                                  *
;*                               2->INTEGR.                                  *
;*                               3->INTEGR.+DIFFER.                          *
;*                               4->PROPOR.                                  *
;*                               5->PROPOR.+DIFFER.                          *
;*                               6->PROPOR.+INTEGR.                          *
;*                               7->PROPOR.+INTEGR.+DIFFER.                  *
;* THE INPUT SIGNAL IS GENERATED BY MEANS OF THE TIMER0 OF THE INTEL'S       *
;* BOARD. THE OUTPUT OF THE SYSTEM GOES TO PORTA OF THE INTEL 8255.          *
;*                                                                           *
;*****************************************************************************
;
;
;MACRO1: ADD AND ROUND
%*DEFINE(ADDIT) LOCAL NEGAT OUTS(
        ADD     AX,BX
        JNO     %OUTS
        JNS     %NEGAT
        MOV     AX,7FFFH
        JMP     %OUTS
%NEGAT:
        MOV     AX,8000H
%OUTS:
)
;MACRO2:SUBSTRACT AND ROUND
%*DEFINE(SUB) LOCAL NEGAT OUTS(
        SUB     AX,BX
        JNO     %OUTS
        JNS     %NEGAT
        MOV     AX,7FFFH
        JMP     %OUTS
%NEGAT:
        MOV     AX,8000H
%OUTS:
)
;MACRO3:MULTIPLY AND ROUND
%*DEFINE(MULTI) LOCAL POSIT OUTS(
        IMUL    BX
        JNO     %OUTS
        AND     DX,8000H
        JZ      %POSIT
        MOV     AX,8000H
        JMP     %OUTS
%POSIT:
        MOV     AX,7FFFH
%OUTS:
)
;MACRO4:DIVIDE AND ROUND
%*DEFINE(DIVIDE) LOCAL EDIV INCR OUT1 OUTS(
```

```
        OR      AX,AX
        PUSHF
        JNS     %EDIV
        NEG     AX
%EDIV:
        MOV     DX,0
        DIV     BX
        SHL     DX,1
        JO      %INCR
        CMP     DX,BX
        JAE     %INCR
        JMP     %OUT1
%INCR:
        INC     AX
%OUT1:
        POPF
        JNS     %OUTS
        NEG     AX
%OUTS:
)
;
INTPTR SEGMENT AT 0H
        ORG     88H
TYPE2   DD      TIMINT
;
INTPTR ENDS
;
ASSUME CS:CCODE, DS:DDATA
;
DDATA   SEGMENT WORD    PUBLIC
;
INSTBYTE        DB      ?       ;THIS BYTE SELECTS ALGORITHM
                                ;0:OUTPUT=0
                                ;1:DIFFER
                                ;2:INTEGR
                                ;3:INTEGR+DIFFER
                                ;4:PROPOR
                                ;5:PROPOR+DIFFER
                                ;6:PROPOR+INTEGR
                                ;7:PROPOR+INTEGR+DIFFER
EPSILON                 DW      ?       ;INPUT SIGNAL TO THE ALGORITHM
SAMT            DW      ?       ;SAMPLE TIME
STIN            DW      ?       ;INTEGRATOR STORAGE
KPROP           DW      1H      ;PROPORTIONAL CONSTANT
INTT            DW      ?       ;INTEGRATION TIME
DIFFT           DW      1H      ;DIFFERENTIATING TIME
OUTPUT          DW      ?       ;OUTPUT SIGNAL
OLD             DW      ?       ;USED IN DIFFER
NEW             DW      ?       ;USED IN DIFFER
;
DDATA   ENDS
;
;
;
CCODE   SEGMENT WORD    PUBLIC
;
```

```
INIT:
        CLI                     ;DISABLE INTERRUPTS
        MOV     AL,17H
        OUT     0C0H,AL         ;ICW1 TO PIC
        MOV     AL,20H          ;VECTOR BASE IS 4*20H
        OUT     0C2H,AL         ;ICW2 TO PIC
        MOV     AL,1FH
        OUT     0C2H,AL         ;ICW4 TO PIC
        MOV     AL,NOT(4)       ;ALLOW INT2(2)
        OUT     0C2H,AL         ;OCW TO PIC, ALLOW INT2
        MOV     AL,36H
        OUT     0D6H,AL         ;INIT TIMER 0 FOR 4MS COUNT
        MOV     AL,0F6H
        OUT     0D0H,AL         ;LSB TIME (0492D CYCLES=400US)
        MOV     AL,1H
        OUT     0D0H,AL         ;MSB TIME
        MOV     CX,10D          ;COUNT FOR 4MSEC
        MOV     AL,80H          ;INITIATE 8255: ALL PORTS=OUTPUT
        OUT     0CEH,AL         ;
        MOV     BX,DDATA        ;INITIATE DDATA SEGMENT
        MOV     DS,BX
        MOV     STIN,0          ;INITIATING OF INTEGRATOR
        MOV     OUTPUT,0        ;INITIATING OF OUTPUT
        MOV     OLD,0           ;INITIATING OF OLD
        MOV     INTT,5          ;INITIATING OF INTT
        MOV     SAMT,1          ;INITIATING OF SAMT
;
        MOV     AX,5            ;SQUAREWAVE'S AMPLITUDE
        STI                     ;ALLOW INTERRUPTS
;
;
MAIN:
        OUT     0CCH,AL         ;CONTENTS OF AL TO OUTPUT PORT C
        JMP     MAIN            ;MAIN IS MAIN PROGRAM TO BE INTERRUPTED
;
;
;
TIMINT PROC
        LOOP    ONE
        NEG     AX              ;INVERT OUTPUT FOR SQUARE WAVE
        MOV     CX,10D          ;LOAD COUNT AGAIN
ONE:
        MOV     OUTPUT,0        ;CLEAN OUTPUT
        MOV     EPSILON,AX      ;INITIAL VALUE
        PUSH    AX              ;
        CMP     INSTBYTE,0      ;TEST IF 0 REQUEST
        JNZ     PID
        MOV     AX,OUTPUT
        JMP     NEXT
PID:    TEST    INSTBYTE,4      ;TEST IF PROP. REQUEST
        JE      LINT
        CALL    PROPOR          ;PERFORM PROPROR. ROUTINE
        MOV     OUTPUT,AX
LINT:   TEST    INSTBYTE,2      ;TEST IF INTEGR. REQUEST
        JE      LDIFF
        CALL    INTEGR          ;PERFORM INTEGR. ROUTINE
```

```
        MOV     BX,OUTPUT
;MACRO1
        %ADDIT
        MOV     OUTPUT,AX
LDIFF:  TEST    INSTBYTE,1      ;TEST IF DIFFER. REQUEST
        JE      NEXT
        CALL    DIFFER          ;PERFORM DIFFER. ROUTINE
        MOV     BX,OUTPUT
;MACRO1
        %ADDIT
NEXT:           OUT     0C8H,AL         ;PID OUTPUT TO PORT A
        POP     AX
        STI
        IRET                    ;BACK TO MAIN
TIMINT ENDP
;
;
;
PROPOR PROC
        MOV     AX,EPSILON
        MOV     BX,KPROP
;MACRO3
        %MULTI
        RET
PROPOR ENDP
;
;
;
INTEGR PROC
        MOV     AX,EPSILON
        MOV     BX,STIN
;MACRO1
        %ADDIT
        MOV     STIN,AX
        MOV     BX,INTT
;MACRO4
        %DIVIDE
        MOV     BX,SAMT
;MACRO3
        %MULTI
        RET
INTEGR ENDP
;
;
;
DIFFER PROC
        MOV     AX,EPSILON
        MOV     BX,OLD
        MOV     OLD,AX
;MACRO2
        %SUB
        MOV     BX,DIFFT
;MACRO3
        %MULTI
        MOV     BX,SAMT
;MACRO4
```

```
SOURCE FILE: :F1:INTOUT.SRC
OBJECT FILE: :F1:INTOUT.OBJ
CONTROLS SPECIFIED: CODE.

STMT LINE NESTING      SOURCE TEXT: :F1:INTOUT.SRC
   1    1  0  0       PROGRAM INTOUT;

   2    3  0  0       LABEL   99;

   3    5  0  0       CONST
                      (*GENERAL DEFINITIONS*)
                          INTT          =02H;
   4    8  0  0             SAMT          =01H;
   5    9  0  0             SW       =TRUE;
                      (*8253 DEFINITIONS*)
   6   11  0  0             COUNTCONTROLPORT=0D6H;
   7   12  0  0             COUNTREG0     =0D0H;
   8   13  0  0             INITIALIZEREG0  =036H;
                      (*8259 DEFINITIONS*)
   9   15  0  0             ICONTROLPORT      =0C0H;
  10   16  0  0             IMASKREGPORT      =0C2H;
  11   17  0  0             IMASK         =0FBH;

  12   19  0  0             ICW1ADDR      =0C0H;
  13   20  0  0             ICW1DATA      =017H;
  14   21  0  0             ICW2ADDR      =0C2H;
  15   22  0  0             ICW2DATA      =020H;
  16   23  0  0             ICW4ADDR      =0C2H;
  17   24  0  0             ICW4DATA      =01FH;

                      (*8255 DEFINITIONS*)
  18   27  0  0             PPICONTROLWORD   =080H;(* ALL PORTS OUTPUT *)
  19   28  0  0             PPIPORTA      =0C8H;
  20   29  0  0             PPICONTROLPORT   =0CEH;

  21   31  0  0       TYPE    BBYTE         =-128..127;

  22   33  0  0       VAR STIN          :BBYTE;
  23   34  0  0           INPUT         :BBYTE;
  24   35  0  0           OUTPUT        :BBYTE;
  25   36  0  0           N         :INTEGER;

  26   38  0  0       PROCEDURE INITIALIZECHIP53;(* INITIALIZE THE TIMER 8253 BY SETTING THE CONTROL WORD TO SELECT C
                     -OUNTER0, READ/LOAD LOW-ORDER BYTE THEN HIGH-ORDER BYTE, SQUARE WAVE GENERATOR; THE PROCEDURE AL
                     -SO LOADS THE COUNTER WITH THE VALUE 496D, THEN THE COUNTER WILL C
                      OUNTDOWN FOR 400USEC. *)
  27   40  1  0       VAR COUNT: RECORD   CASE BOOLEAN OF
  28   41  1  1                         TRUE:(FULLWORD:WORD);
  29   42  1  1                         FALSE:(LOW,HIGH:0..255)
                                 END;
  30   44  1  0       BEGIN
  30   45  1  1           COUNT.FULLWORD:=496;(*400USEC*)
  31   46  1  1           OUTBYT(COUNTREG0,COUNT.LOW);
  32   47  1  1           OUTBYT(COUNTREG0,COUNT.HIGH)
                     END;(*INITIALIZECHIP53*)
```

```
STMT LINE NESTING       SOURCE TEXT: :F1:INTOUT.SRC
  33   50  0  0         PROCEDURE INITIALIZECHIP55;(*INITIALIZE PPI WITH ALL PORTS = OUTPUT*)
  34   51  1  0         BEGIN
  34   52  1  1             OUTBYT(PPICONTROLPORT,PPICONTROLWORD)
                        END;(*INITIALIZECHIP55*)

                        $INTERRUPT(SERVICEINTERRUPT)
  35   56  0  0         PROCEDURE SERVICEINTERRUPT;(*THIS PROCEDURE RUNS INT.SRV. 34 WHEN THAT INTERRUPT OCURRS *)
  36   57  1  0         BEGIN
  36   58  1  1             IF N=10 THEN BEGIN
  37   59  1  2                     INPUT:=-INPUT;
  38   60  1  2                     N:=0
                              END;
  40   62  1  1              STIN:=STIN+INPUT;
  41   63  1  1             IF STIN>127 THEN STIN:=127;
  43   64  1  1             IF STIN<-128 THEN STIN:=-128;
  45   65  1  1             OUTPUT:=(STIN*INTT)DIV(SAMT);
  46   66  1  1             IF OUTPUT>127 THEN OUTPUT:=127;
  48   67  1  1             IF OUTPUT<-128 THEN OUTPUT:=-128;
  50   68  1  1             N:=N+1;
  51   69  1  1             OUTBYT(PPIPORTA,OUTPUT)
                        END;(*SERVICEINTERRUPT*)

  52   72  0  0         BEGIN(*MAIN*)
  52   73  0  1             SETINTERRUPT(34,SERVICEINTERRUPT);
  53   74  0  1             DISABLEINTERRUPTS;
  54   75  0  1             STIN:=0;
  55   76  0  1             OUTPUT:=0;
  56   77  0  1             INPUT:=2;
  57   78  0  1             N:=0;
  58   79  0  1             OUTBYT(COUNTCONTROLPORT,INITIALIZEREG0);
  59   80  0  1             INITIALIZECHIP53;
  60   81  0  1             OUTBYT(ICW1ADDR,ICW1DATA);
  61   82  0  1             OUTBYT(ICW2ADDR,ICW2DATA);
  62   83  0  1             OUTBYT(ICW4ADDR,ICW4DATA);
  63   84  0  1             OUTBYT(IMASKREGPORT,IMASK);
  64   85  0  1             INITIALIZECHIP55;
  65   86  0  1             ENABLEINTERRUPTS;
  66   87  0  1         99:
                            WHILE SW=TRUE DO
  67   89  0  1             GOTO 99
                        END.
```

```
                        ; STATEMENT # 1

                        ; STATEMENT # 26

                        INITIALIZECHIP53
                                PROC NEAR
0000  55                        PUSH    BP
0001  8BEC                      MOV     BP,SP
0003  55                        PUSH    BP
0004  83EC02                    SUB     SP,2H

                        ; STATEMENT # 30

0007  C746FCF001                MOV     COUNT[BP],1F0H

                        ; STATEMENT # 31

000C  8A46FC                    MOV     AL,COUNT[BP]
000F  E6D0                      OUT     0D0H

                        ; STATEMENT # 32

0011  8A46FD                    MOV     AL,COUNT[BP+1H]
0014  E6D0                      OUT     0D0H
0016  8BE5                      MOV     SP,BP
0018  5D                        POP     BP
0019  C3                        RET
                        INITIALIZECHIP53
                                ENDP

                        ; STATEMENT # 33

                        INITIALIZECHIP55
                                PROC NEAR
001A  55                        PUSH    BP
001B  8BEC                      MOV     BP,SP
001D  55                        PUSH    BP

                        ; STATEMENT # 34

001E  B080                      MOV     AL,80H
0020  E6CE                      OUT     0CEH
0022  8BE5                      MOV     SP,BP
0024  5D                        POP     BP
0025  C3                        RET
                        INITIALIZECHIP55
                                ENDP

                        ; STATEMENT # 35

                        SERVICEINTERRUPT
```

```
                        PROC NEAR          ;INTERRUPT PROC
0026  06               PUSH    ES
0027  1E               PUSH    DS
0028  50               PUSH    AX
0029  51               PUSH    CX
002A  52               PUSH    DX
002B  53               PUSH    BX
002C  56               PUSH    SI
002D  57               PUSH    DI
002E  B80000           MOV     AX,SEG @DATA
0031  8ED8             MOV     DS,AX
0033  55               PUSH    BP
0034  8BEC             MOV     BP,SP
0036  55               PUSH    BP

                    ; STATEMENT # 36

0037  833E06000A       CMP     N,0AH
003C  750E             JNE     ?0

                    ; STATEMENT # 37

003E  A10200           MOV     AX,INPUT
0041  F7D8             NEG     AX
0043  A30200           MOV     INPUT,AX

                    ; STATEMENT # 38

0046  C70606000000     MOV     N,0H
                    ?0:

                    ; STATEMENT # 40

004C  A10200           MOV     AX,INPUT
004F  03060000         ADD     AX,STIN
0053  A30000           MOV     STIN,AX

                    ; STATEMENT # 41

0056  833E00007F       CMP     STIN,7FH
005B  7E04             JLE     ?1

                    ; STATEMENT # 42

005D  C70600007F00     MOV     STIN,7FH
                    ?1:

                    ; STATEMENT # 43

0063  833E000080       CMP     STIN,0FF80H
0068  7D06             JNL     ?2

                    ; STATEMENT # 44

006A  C70600008FF      MOV     STIN,0FF80H
                    ?2:
```

```
                        ; STATEMENT # 45

0070  B80200                  MOV     AX,2H
0073  F72E0000                IMUL    STIN
0077  B90100                  MOV     CX,1H
007A  F7F9                    IDIV    CX
007C  A30400                  MOV     OUTPUT,AX

                        ; STATEMENT # 46

007F  833E04007F              CMP     OUTPUT,7FH
0084  7E06                    JLE     ?3

                        ; STATEMENT # 47

0086  C70604007F00            MOV     OUTPUT,7FH
                      ?3:

                        ; STATEMENT # 48

008C  833E040080              CMP     OUTPUT,0FF80H
0091  7D06                    JNL     ?4

                        ; STATEMENT # 49

0093  C706040080FF            MOV     OUTPUT,0FF80H
                      ?4:

                        ; STATEMENT # 50

0099  A10600                  MOV     AX,N
009C  40                      INC     AX
009D  A30600                  MOV     N,AX

                        ; STATEMENT # 51

00A0  A10400                  MOV     AX,OUTPUT
00A3  E6C8                    OUT     0C8H
00A5  8BE5                    MOV     SP,BP
00A7  5D                      POP     BP
00A8  5F                      POP     DI
00A9  5E                      POP     SI
00AA  5B                      POP     BX
00AB  5A                      POP     DX
00AC  59                      POP     CX
00AD  58                      POP     AX
00AE  1F                      POP     DS
00AF  07                      POP     ES
00B0  CF                      IRET
                      SERVICEINTERRUPT
                              ENDP

                      INTOUT  PROGRAM
00B1  8BEC                    MOV     BP,SP
00B3  4D                      DEC     BP
```

```
00B4  4D                DEC     BP
00B5  55                PUSH    BP
00B6  9A00000000        CALL    INITFP
00BB  9A00000000        CALL    TU+001

                    ; STATEMENT # 52

00C0  B82200            MOV     AX,22H
00C3  50                PUSH    AX
00C4  2E8D0E2600        LEA     CX,CS:SERVICEINTERRUPT
00C9  0E                PUSH    CS
00CA  51                PUSH    CX
00CB  9A00000000        CALL    TU+302

                    ; STATEMENT # 53

00D0  FA                CLI

                    ; STATEMENT # 54

00D1  C70600000000      MOV     STIN,0H

                    ; STATEMENT # 55

00D7  C70604000000      MOV     OUTPUT,0H

                    ; STATEMENT # 56

00DD  C70602000200      MOV     INPUT,2H

                    ; STATEMENT # 57

00E3  C70606000000      MOV     N,0H

                    ; STATEMENT # 58

00E9  B036              MOV     AL,36H
00EB  E6D6              OUT     0D6H

                    ; STATEMENT # 59

00ED  E810FF            CALL    INITIALIZECHIP53

                    ; STATEMENT # 60

00F0  B017              MOV     AL,17H
00F2  E6C0              OUT     0C0H

                    ; STATEMENT # 61

00F4  B020              MOV     AL,20H
00F6  E6C2              OUT     0C2H

                    ; STATEMENT # 62

00F8  B01F              MOV     AL,1FH
```

```
00FA  E6C2                OUT     0C2H

                        ; STATEMENT # 63

00FC  B0FB                MOV     AL,0FBH
00FE  E6C2                OUT     0C2H

                        ; STATEMENT # 64

0100  E817FF              CALL    INITIALIZECHIP55

                        ; STATEMENT # 65

0103  FB                  STI

                        ; STATEMENT # 66

                        @99:
                        ?0:
0104  B001                MOV     AL,1H
0106  3C01                CMP     AL,1H
0108  7504                JNE     ?1

                        ; STATEMENT # 67

010A  EBF8                JMP     @99
010C  EBF6                JMP     ?0
                        ?1:
010E  9A00000000          CALL    TQ+999
                        INTOUT  ENDP
```

SUMMARY INFORMATION:

PROCEDURE	OFFSET	CODE SIZE		DATA SIZE		STACK SIZE	
INITIALIZECHIP53	0000H	001AH	26D			0006H	6D
INITIALIZECHIP55	001AH	000CH	12D			0006H	6D
SERVICEINTERRUPT	0026H	0086H	139D			0026H	38D
INTOUT	00B1H	0062H	98D	0008H	8D	000AH	10D
-CONST IN CODE-		0000H	0D				
TOTAL		0113H	275D	0008H	8D	0070H	112D

89 LINES READ.
0 ERRORS DETECTED.

DICTIONARY SUMMARY:

120KB MEMORY AVAILABLE.
6KB MEMORY USED (5%).
0KB DISK SPACE USED.
2KB OUT OF 16KB STATIC SPACE USED (12%).