# MICROFRIEND
# DYNA-86

## User's Manual

**Dynalog (India) Ltd.**
Kailash Vaibhav, Park Site,
Vikhroli (West), Mumbai - 400 079. INDIA.
Tel : 5181900 Fax : 5181930 / 40.
email : sales@dynalogindia.com
**www.dynalogindia.com**

# TABLE OF CONTENTS

## CHAPTER 1 OVERVIEW 1-1

## CHAPTER 2 GETTING STARTED 2-1

## CHAPTER 3 MEMORY AND I/ O DETAILS 3-1

## CHAPTER 4 HEX KEYPAD MODE 4-1

## CHAPTER 5 SERIAL MODE 5-1

## CHAPTER 6 ASSEMBLER AND DISASSEMBLER 6-1

## CHAPTER 7 SAMPLE PROGRAMS 7-1

# CHAPTER 1
# OVERVIEW

## INTRODUCTION :

DYNA-86 is a general purpose Micro-Computer kit having advanced hardware and software features. Dynalog has designed DYNA-86 to be an ideal introduction to the rapidly expanding field of µPs. It is an excellent learning tool for learning 8086 microprocessor and also for development of 8086 based systems.

The kit has two boards -

**I)** DYNA-86 MB
**II)** Keyboard/display Module

There are two modes of operation - **I)** Hex Keypad Mode
**II)** Serial Mode

# SYSTEM HARDWARE OF DYNA-86 :

**CPU :**
DYNA-86 is based on Intel 8086 high performance CPU operating at 8 MHz speed.

**Numeric Co-processor :**

An optional socket is provided for 8087-2 NDP.

**Memory :**

Monitor Firmware in two 27256 EPROMs is placed in the highest 64 KB bank (F0000H to FFFFFH). 64 KB Static RAM with powerful battery backup is provided in the address range 00000 to 0FFFFH.

**Hexpad / Display Interface :**

8279 keyboard display controller is used for Hexpad keys & Displays (8 Nos. of 7 Segment Displays).

**Serial Interface :**

Serial interface is available through a RS-232 compatible port. 8251 USART along with 1488, 1489 driver chips provides necessary signals for this interface. The signals are brought out on the 9 pin D-type male connector (J5). Baud rates from 300 to 9600 can be selected through software.

**Timer :**

Three channels of 16 bit Timer/ Counter are provided using 8254. CHANNEL 0 is used for Baud rate generation. CH1 and CH2 signals are brought out on a 7 pin Relimate connector (J6) and can be used by the user.

### Interrupt Controller :

The 8259 Interrupt Controller provides 8 priortised interrupt levels. IRQ5 to IRQ7 are brought out on 50 pin FRC connector (J7) and can be used by the user. IRQ3 is connected to "INT" key of Hex keypad. 8259 is programmed for edge trigger. Except IRQ3 all other interrupts are masked.

### Parallel I/O Interface :

Two 8255's are present onboard, out of which 1 is used for DYNA-PIO cards and 1 for Printer Interface. All the 48 lines of 8255's are available to the user and are brought out on the two numbers of 26 pin FRC male connectors (J1, J2 ). The PIO cards from Dynalog supported on DYNA-86 are listed in Appendix B, and can be interfaced on connector J2. The connector J1 is used for printer interface in serial mode.

## SYSTEM SOFTWARE

The DYNA-86 Microprocessor kit has vast software features. It supports two different modes of operation :

1) **HEX KEYPAD** mode
2) **SERIAL** mode

## HEX KEYPAD COMMANDS

The **HEX KEYPAD** mode supports the following commands :

| | | |
|---|---|---|
| RES | RESET | This key terminates the present activity and returns the DYNA-86 to the initial state. |
| INT | Interrupt | This key is used to generate an interrupt through 8259, (IRQ3). |
| U1 | User Key | User definable key. Pressing this key, type F0H, Software interrupt of 8086 is executed. |

| | | |
|---|---|---|
| CODE | Coded Utility | This key is used to execute utilities like fill memory, copy, search etc. |
| STEP | Single step | Permits program instructions to be executed individually. |
| U2 | User Key | User definable key. Pressing this key, type F1H, Software interrupt of 8086 is executed. |
| REG | Register | This key is used when any of the 8086 registers have to be accessed for display and modification. |
| GO | Go | Executes the program from the specified location. |
| SER | Serial Mode | This key is used to select SERIAL MODE. |
| INR | Increment | This key is used to update the address field to the next consecutive memory location or next menu item. |
| EXEC | Execute | This key is the command terminator. When pressed, the current command is executed. |
| DCR | Decrement | This key is used to decrement the address field to the previous memory location or previous menu item. |
| EB | Examine Byte | Examine the contents of memory locations in bytes. |
| IB | Input Byte | Inputs data byte from an input port. |
| OB | Output Byte | Outputs data byte to the output port. |

| | | |
|---|---|---|
| BK | Break Point | This is used to set the break point anywhere in RAM. |
| EW | Examine Word | Examine the contents of memory locations in words. |
| IW | Input Word | Inputs data word from an input port. |
| OW | Output Word | Outputs data word to the output port. |
| PIO | DYNA-PIO Cards | Pressing this key, monitor will ask for different PIO cards to be studied on J2 connector. |
| DYNA | DYNA-series Study cards | This key is used to select the Dyna-Series study cards' experiments connected on J7 connector. |

The following commands work with DYNA-PIO Series DYNA-PIO-PGMR card only. For details refer to DYNA-PIO-PGMR manual.

| | | |
|---|---|---|
| RE | Read EPROM | To read the contents of EPROM in ZIF into memory. |
| BC | Blank Check | To blank check the EPROM. |
| VR | Verify EPROM | To verify the memory contents and programmed data of EPROM. |
| PR | Program EPROM | To program EPROM in ZIF. |
| SE | Set EPROM | To program EPROM in ZIF one byte at a time. |
| CK | Checksum | To calculate the checksum of EPROM in ZIF. |

## SERIAL MODE COMMANDS

The **SERIAL** mode supports the following commands :

<table>
<tr><th>Commands</th><th>Function / Syntax</th></tr>
<tr><td><b>D</b> Display Memory</td><td>Displays block of memory data<br><code>D[W][[seg:]strt[,end]]&lt;CR&gt;</code></td></tr>
<tr><td><b>E</b> Edit Memory</td><td>Modify Contents<br><code>E[W][[seg:]strt]&lt;CR&gt;</code></td></tr>
<tr><td><b>C</b> Copy Memory</td><td>Block Copy<br><code>C[seg:]strt,end,[seg:]dest&lt;CR&gt;</code></td></tr>
<tr><td><b>F</b> Fill Memory</td><td>Block Fill<br><code>F[W][seg:]strt,end,byte|word&lt;CR&gt;</code></td></tr>
<tr><td><b>I</b> Insert Byte/Word</td><td>Insert Byte/Word in a block<br><code>I[W][seg:]strt,end,byte|word&lt;CR&gt;</code></td></tr>
<tr><td><b>D</b> Delete Byte/Word</td><td>Delete Byte/Word in a block<br><code>DL[W][seg:]strt,end&lt;CR&gt;</code></td></tr>
<tr><td><b>S</b> Search Byte/Word</td><td>Search Byte/Word in a block<br><code>S[W][seg:]strt,end,byte|word&lt;CR&gt;</code></td></tr>
<tr><td><b>CM</b> Compare Blocks</td><td>Compare Blocks<br><code>CM[seg:]strt,end,[seg:]strt&lt;CR&gt;</code></td></tr>
<tr><td><b>R</b> Register</td><td>Examine/Modify Register<br><code>R[seg:]&lt;CR&gt;</code></td></tr>
<tr><td><b>T</b> Trace</td><td>Single step<br><code>T[[seg:]strt]&lt;CR&gt;</code></td></tr>
</table>

| Commands | Function / Syntax |
|---|---|
| **G** Go | Transfers 8086 control from monitor to user program<br>`G[[[seg:]strt],[[seg:]brk]]<CR>` |
| **HD** Hex to Dec | Hexadecimal to Decimal Conversion<br>`HDnum<CR>` |
| **DH** Dec to Hex | Decimal to Hexadecimal Conversion<br>`DHnum<CR>` |
| **L** Load file | Download hex file from terminal<br>`L[seg:]<CR>` |
| **W** Write file | Upload hex file to terminal<br>`W[seg:],strt,end<CR>` |
| **IN** Input Byte/Word | Input from port<br>`IN[W]addr<CR>` |
| **O** Output Byte/Word | Output to port<br>`O[W]addr,[byte\|word]<CR>` |
| **A** Assemble | Assemble to Memory<br>`A[[seg:]off]<CR>` |
| **U** Unassemble | Unassemble Memory block<br>`U[[seg:]off[,[seg:]off]]<CR>` |
| **P** Printer ON/OFF | Enables or disables the printer output connected to connector J1.<br>`P <CR>` |

# CHAPTER 2
# GETTING STARTED

## UNPACKING DYNA-86

While unpacking, make sure that the following items are present.

**1.** Motherboard consisting of :

a) 16-bit 8086 microprocessor.
b) 64 KB of ROM containing powerful Monitor Firmware in two 27256 (U30 & U32).
c) 64 KB of battery backed Static RAM in two 62256 chips (U31 & U33).
d) Support chips 8254, 8251, 8259, 8255, 8279, socket for optional 8087 NDP.
e) Hex Keypad containing 28 keys & 8 nos. of 7 segment LED Displays.

**2.** User's manual.

**3.** Item no. 1 housed in an attractive wooden box.

## OPTIONS FOR DYNA-86

**1.** Switch Mode Power Supply Model SMPS-04.
**2.** Serial Cable for RS232 Port.
**3.** 26 pin FRC cables for interfacing 8255 PIO lines.
**4.** PIO cards from Dynalog supported on 8255 port interface connector J2 as given in Appendix B.
**5.** DYNA-Series Study Cards from Dynalog as given in Appendix C.

# POWER SUPPLY

The kit is normally used with the Dynalog's SMPS 04 Model Power Supply. The 6 pin female connector can be plugged in 6 pin Male Connector soldered on board, (Connector J3).

The power requirement of DYNA-86 board is :

| | |
|---|---|
| + 5V | 3 Amps |
| + 12 V | 250 mA |
| - 12 V | 250 mA |

# CONNECTORS ON BOARD

The pin details of all the connectors are given in Appendix A.

## SERIAL CONNECTOR

All the signals for the RS 232C compatible Serial Interface are brought out on the 9-pin D type Male (DTM) connector (J5) onboard. The serial cables can be directly connected to this connector.

## Relimate Connector for TIMER

A 7-pin Relimate Connector (J6) is provided, which has Timer interface lines terminated on it. It can be used for user applications.

### FRC for 8255 I/O Interface

Two 26 pin FRC male connectors (J1 and J2) are provided onboard for 8255's I/O Interface. The 3 ports, 8 bit each, (24 lines) of each, 8255 are provided on this connector. Connector J2 is used for interfacing DYNA-PIO cards whereas J1 is used to connect printer in serial mode.

### FRC for Buffered Bus

A 50 pin FRC male connector provided is for Bus expansion purpose. All the address, Data and Control lines alongwith the DRQ & interrupts are terminated on this connector (J7). The same connector is used to interface DYNA-86 with DYNA-Series Study Cards given in Appendix C.

## Installation Procedure

1. First connect Power Supply (SMPS-04) cable (6 pin female connector to the system supply connector (J3) with proper orientation.
2. For Serial Mode connect the serial cable to 9 pin DTM connector (J5) & terminal.
3. Switch on the Power Supply, Display (7 segment) will show
   F r I E n d
4. Now the system is ready for use.

# CHAPTER 3
# MEMORY AND I/O DETAILS

## MEMORY MAP

The memory map of DYNA-86 is shown in the following table:

| Address | Socket No. | Chips | Total Capacity |
|---|---|---|---|
| 00000-0FFFFH | U31/U33 | Battery-backed<br>62256<br>SRAM | 64KB |
| 40000-BFFFFH | CS2 signal on<br>J7 connector | User Expansion | 512KB |
| F0000-FFFFFH | U30/U32 | Firmware<br>EPROM<br>27256 | 64KB |

## SYSTEM I/O MAP

The I/O devices are addressed using I/O mapped address space. The I/O map for different peripheral chips is given below :

## 8279 Keyboard/ Display Controller

BASE = 50H

| | Add. in Hex | Function |
|---|---|---|
| BASE + 0 | 50 | Data Register |
| BASE + 2 | 52 | Command/ Status Register |

## 8254 - Timer

BASE = 40H

| | Add. in Hex | Function |
|---|---|---|
| BASE + 0 | 40 | Counter 0 |
| BASE + 2 | 42 | Counter 1 |
| BASE + 4 | 44 | Counter 2 |
| BASE + 6 | 46 | Control Word Register |

## 8259 - Interrupt Controller

BASE = 00H

| | Add. in Hex | Function |
|---|---|---|
| BASE + 0 | 00 | ICW1, OCW2, OCW3 |
| BASE + 2 | 02 | ICW2, ICW3, ICW4, OCW1 |

***Note : Two 8255 chips are interfaced as 16-bit I/O.***

**8255 (#1) PPI (U6)**

BASE = 60H

If addressed individual i.e. Byte operation

| | Add. in Hex | Function |
|---|---|---|
| BASE + 0 | 60 | PORT A |
| BASE + 2 | 62 | PORT B |
| BASE + 4 | 64 | PORT C |
| BASE + 6 | 66 | Control Word Register |

**8255 (#2) PPI (U7)**

BASE = 61H

If addressed individual i.e. Byte operation

| | Add. in Hex | Function |
|---|---|---|
| BASE + 0 | 61 | PORT A |
| BASE + 2 | 63 | PORT B |
| BASE + 4 | 65 | PORT C |
| BASE + 6 | 67 | Control Word Register |

**OR**

**8255 (#1 & #2) (U6 & U7)**
BASE = 60H
If addressed as word.

| | Add. in Hex | Function |
|---|---|---|
| BASE+0 | 60 | PORT A (U6) & PORT A (U7) |
| BASE+2 | 62 | PORT B (U6) & PORT B (U7) |
| BASE+4 | 64 | PORT C (U6) & PORT C (U7) |
| BASE+6 | 66 | Control Word Register for U6 & U7. |

D0 to D7 = will contain data for U6, 8255
D8 to D15 = will contain data for U7, 8255.

**8251-USART**
BASE = 10H

| | Add. in Hex | Function |
|---|---|---|
| BASE + 0 | 10 | Data Register |
| BASE + 2 | 12 | Control/Status Register |

The following I/O chip selects are provided on J7 connector for DYNA-Series study cards :

**IOEXP* (J7-35)**
Address range 30H to 37H.

**IOEXP1* (J7-36)**
Address range 28H to 2FH

**IOEXP2* (J7-37)**
Address range 28H to 2FH

**IOEXP3* (J7-38)**
Address range 20H to 27H.

# CHAPTER 4
# HEX KEYPAD MODE

## INTRODUCTION

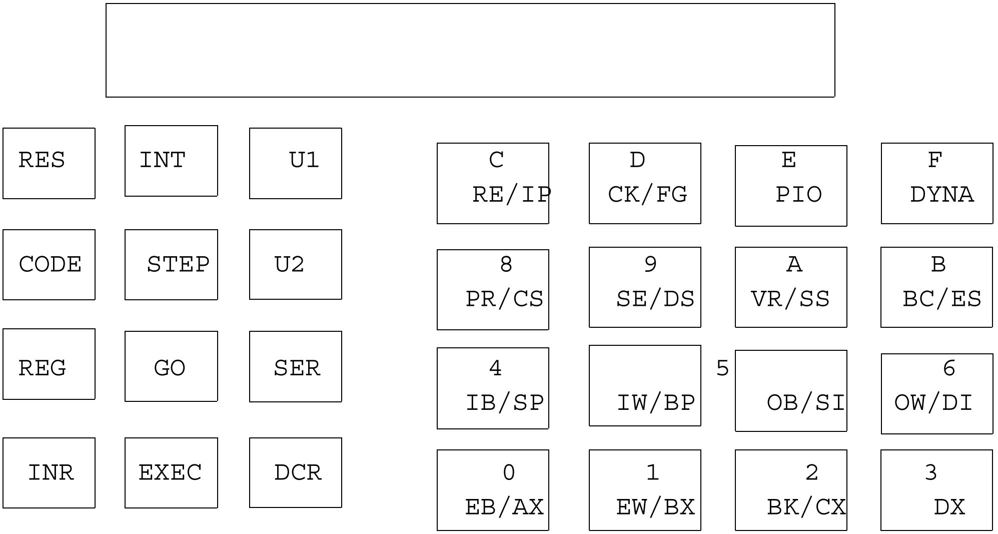
On power on the kit enters into HEX KEYPAD MODE and display shows "FrIEnd" on 7 segment LED displays. In this mde monitor processes all the inputs entered through the HEX KEYPAD and displays the results on the eight digits of seven segment LED display. Of the eight digits, four on the left are for the address field and four on the right for the data field.

## THE HEX KEYPAD

There are in all 28 keys present on the keypad. These keys serve dual purposes ie. used for both command and data entry. The keypad is divided into two groups, 16 hexadecimal keys on the right side and 12 function keys on the left side.

The 16 hexadecimal keys have combined functions as noted on their legends. The small letters present below the hexadecimal values are acronyms for individual monitor commands and 8086 register names.

In the following sections, acronyms on the left of the slash sign are monitor commands while on the right side 8086 register names are present. The function of the hexadecimal keys at any instant is dependent on the current state of the monitor and the inputs so far.



| **Hexadecimal key** | | **Command Acronym** | | **Register Acronym** |
|---|---|---|---|---|
| 0<br>EB/AX | **[EB]** | Examine Byte | **[AX]** | Accumulator (Reg. AX) |
| 1<br>EW/BX | **[EW]** | Examine Word | **[BX]** | Base (Reg. BX) |
| 2<br>BK/CX | **[BK]** | Break Point | **[CX]** | Count (Reg. CX) |
| 3<br>DX | | | **[DX]** | Data |

| **Hexadecimal key** | | **Command Acronym** | | **Register Acronym** |
|---|---|---|---|---|
| 4<br>IB/SP | **[IB]** | Input Byte | **[SP]** | Stack Pointer |
| 5<br>IW/BP | **[IW]** | Input Word | **[BP]** | Base Pointer |
| 6<br>OB/SI | **[OB]** | Output Byte | **[SI]** | Source Index |
| 7<br>OW/DI | **[OW]** | Output Word | **[DI]** | Destination Index |
| 8<br>PR/CS | **[PR]** | Program EPROM | **[CS]** | Code Segment |
| 9<br>SE/DS | [**SE**] | Program EPROM's 1 byte at a time | **[DS]** | Data Segment |
| A<br>VR/SS | **[VR]** | Verify EPROM | **[SS]** | Stack Segment |
| B<br>BC/ES | **[BC]** | Blank Check EPROM | **[ES]** | Extra Segment |
| C<br>RE/ IP | **[RE]** | Read EPROM | **[IP]** | Instruction Pointer |
| D<br>CK/ FG | **[CK]** | Checksum of EPROM | **[FG]** | Flags |

| Hexadecimal key | | Command Acronym | Register Acronym |
|---|---|---|---|
| E<br>PIO | **[PIO]** | DYNA-PIO cards experiments | ----- |
| F<br>DYNA | **[DYNA]** | DYNA Study Cards experiments | ---- |

The 12 function keys can be interpreted as follows :

| Function key | | Operation |
|---|---|---|
| RES | RESET | This key terminates the present activity and returns DYNA-86 to the initial state. When pressed the "FriEnd" sign-on message appears on the display and the monitor is ready for command entry. |
| INT | Interrupt | This key is used to generate a type B (INT B) interrupt through 8259. This interrupt is initialized at Power-On or Reset and goes to a routine in the monitor program which saves all the contents of the 8086 registers. Control is returned to the monitor for command entry. |
| U1 | User Key | Pressing this key the 8086 software interrupt type F0 will be executed. On power on or reset, this vector is initialised to monitor firmware to scroll message on LED display. |
| U2 | User Key | Pressing this key the 8086 software interrupt type F1 will be executed. On power on or reset, this vector is initialised to monitor firmware to scroll message on LED display. |

The U1 & U2 keys are directed to 8086 software interrupt INT F0 & INT F1.

U1 directed to INT F0.
U2 directed to INT F1.

The segment and offset of the user application to be executed using the (U1 & U2) keys is to be supplied at the location given below.

| | | |
|---|---|---|
| INT F0 | 0:3C0 | Lower byte of User program's offset address i.e. IP(L). |
| | 0:3C1 | Higher byte of User program's offset address i.e. IP(H). |
| | 0:3C2 | Lower byte of User program's' segment address i.e. CS(L). |
| | 0:3C3 | Higher byte of User program's segment address i.e. CS(H). |
| INT F1 | 0:3C4 | Lower byte of User program's offset address i.e. IP(L). |
| | 0:3C5 | Higher byte of User program's offset address i.e. IP(H) |
| | 0:3C6 | Lower byte of User program's segment addrerss i.e. CS(L) |
| | 0:3C7 | Higher byte of User program's segment address i.e. CS(H) |

| | | |
|---|---|---|
| CODE | Coded Programs | There are 14 programs which can be executed through this key. Pressing this key the user will be prompted for different programs like Copy, Fill, Search. The menu item can be changed by INR or DCR key and can be executed by pressing EXEC key. |
| STEP | Single Step | This key is used to single step through the program. |
| REG | Register | This key is used when any of the 8086 registers have to be displayed or modified. |
| GO | Run | This key is used to execute the program. User will be asked to provide segment and offset of the program to to be executed. |

SER Enter SERIAL Mode This key is used to enter SERIAL Mode. The user is prompted for baud rate. Different baud rate can be selected using "INR", "DCR" keys & terminating the desired baud rate with "EXEC" key on Hex Keypad.

INR Increment The INR key is used to seperate keypad entries and to increment the address to the next consecutive memory location or when it is in menu mode to go to next menu item.

EXEC Execute This key is the command terminator. When used, the current command is executed. Note that while using the Go command, the 8086 begins program execution at the address specified when this key is pressed.

DCR Decrement This key is used to decrement the address to the previous memory location or when monitor is in menu mode to go to previous menu item.

# DISPLAY SECTION

The display is divided into two groups of 4 characters each. Four on the left is refered to as the address field and four to the right as data field. In the following sections, all references to the hexadecimal values which will be displayed on the seven segment LEDs are as given in the Display format column.

| Hexadecimal value | Display format |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| A | A |
| B | b |
| C | C |
| D | d |
| E | E |
| F | F |

# REGISTER INITIALIZATION

When DYNA-86 is intialized during power-on or when [RES] key is pressed, the sign-on message is displayed. When initialized the 8086 registers are set to the values shown in the table given below :

| **Register** | | **Value** |
|---|---|---|
| CS | Code Segment | 0H |
| DS | Data Segment | 0H |
| ES | Extra Segment | 0H |
| SS | Stack Segment | 0H |
| IP | Instruction Pointer | 0H |
| FL | Flag | 0H |
| SP | Stack Pointer | 06FFH |

Whenever the system resets or during power-on, the monitor immediately terminates its present activity and jumps to its initialization routine. This routine initializes interrupt vectors 1 through 3 as follows :

| | |
|---|---|
| Interrupt 1 - Single Step | : Used with Single Step command (STEP key) |
| Interrupt B - IRQ3 | : Monitors [INT] key |
| Interrupt 3 - Breakpoint | : Used with the GO command |

Whenever the monitor is re-entered as a result of Single Step, or Breakpoint interrupt, the monitor temporarily stores the 8086 register contents in memory and subsequently restores them, before it requests for command entry. Since the SP register is initialized to 6FFH (base of the stack), the initial stack reserved for the user is FFH bytes (locations 600-6FFH).

## GENERAL OPERATION

In the Hex Keypad monitor mode, when the monitor is expecting a command entry, F appears in the most significant display digit of the address field. Pressing one of the command keys (keys 0-F) is interpreted as a command entry.

When the key is pressed, F disappears. Depending on the command, characters will appear within the address and data fields.

# MONITOR COMMANDS

## EB, EW (Examine Byte and Word) Commands

**Function :**

The Examine Byte [EB] and Examine Word [EW] commands are used to examine the contents of selected memory locations. Only RAM memory locations can be modified.

**Operation :**

To use either of the commands, press [EB / AX] key or [EW / BX] key respectively. When F or "FrIEnd" is displayed, in both the cases, the monitor is said to be in command mode. When either key is pressed, SEG is displayed on the address field and the present segment value in data field. Now the key pad will be directed to the data field to enter the segment.

Note that all memory addresses consists of segment value and offset value. When the segment value is same, then pressing INR, DCR or EXEC key will prompt the user to enter the offset value. After entering the offset value, pressing INR, DCR or EXEC will show the data of the address entered. Segment value will not be displayed. Pressing INR or DCR key will increment or decrement the address field respectively. The segment and offset value is limited to 4 digits, if more than 4 digits are entered then the last 4 digits are valid which are displayed on the 7-segment LED display.

To modify the contents of memory location enter the data and press INR key. To discard the new entered data press DCR key.

In order to terminate the command, press [EXEC] key and press [INR] key to examine the next memory location. Note that the data field is limited to two digits for EB and four digits for EW. If more characters are entered the last two digits for byte and last four digits for word are valid. The data is not updated unless the [INR] key is pressed.

**Error conditions :** Attempting to modify a non-existent or read- only (ROM or EPROM) memory location, gives an error. This error will be detected only after pressing the [INR] key. When an error is detected, "Err" is displayed in the address field.

**EXAMPLES :**

**1.** Examining a series of memory byte locations from 0:1234.

| **KEY** | **ADDRESS FIELD** | **DATA FIELD** | **COMMENT** |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| 0<br>EB / AX | S E G ❐ | 0 0 0 0 | Examine byte command |
| INR | O F F ❐ | 0 0 0 0 | Modify segment if other than 0 is desired |
| 1<br>EW / BX | O F F ❐ | 0 0 0 1 | |
| 2<br>BK / CX | O F F ❐ | 0 0 1 2 | |
| 3<br>DX | O F F ❐ | 0 1 2 3 | |
| 4<br>IB / SP | O F F ❐ | 1 2 3 4 | |
| INCR | 1 2 3 4 | ❐ ❐ x x• | Data contents |
| INCR | 1 2 3 5 | ❐ ❐ x x• | Next memory location contents |
| EXEC | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | Command termination prompt |

**2.** Examining and modifying memory word location from 500:340.

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| 1<br>EW / BX | S E G | 0 0 0 0 | Examine word command |
| 5<br>IW / BP | S E G ❐ | ❐ ❐ ❐ 5 | |
| 0<br>EB / AX | S E G ❐ | ❐ ❐ 5 0 | |
| 0<br>EB / AX | S E G ❐ | ❐ 5 0 0 | |
| INR | O F F ❐ | 0 0 0 0 | Offset value |
| 3<br>DX | O F F ❐ | 0 0 0 3 | |
| 4<br>IB / SP | O F F ❐ | 0 0 3 4 | |
| 0<br>DX | O F F ❐ | 0 3 4 0 | |
| INR | 0 3 4 0 | x x x x | Data contents |
| INR | 0 3 4 2 | x x x x | Old data |
| 1<br>EW / BX | 0 3 4 2 | 0 0 0 1 | |
| 2<br>BK / CX | 0 3 4 2 | 0 0 1 2 | |

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| 3<br>EW / BX | 0 3 4 2 | 0 1 2 3 | |
| 4<br>EW / BX | 0 3 4 2 | 1 2 3 4 | New data entered |
| INR | 0 3 4 4 | x x x x | Old data |
| 5<br>IW / BP | 0 3 4 4 | 0 0 0 5 | New data |
| DCR | 0 3 4 4 | x x x x | Old data |
| DCR | 0 3 4 2 | 1 2 3 4 | New data entered at 0342 offset |

## REG (Examine Register) Command

### Function

The Examine Register (REG) command is used to examine and if desired, modify the contents of any of the 8086 registers.

Operation

To examine the contents of a register, press the (REG) key, when prompted for command entry. rEG is displayed at the left of address field. Now the subsequent keypad entry will be interpret ed as register name (acronym on the right side of the slash on the keys). When the key is pressed the corresponding register abbreviation will be displayed in the address field alongwith its 16-bit contents in the data field.

The display abbreviation for the registers are :

| Register Name | Keypad Acronym | Display Abbreviation |
|---|---|---|
| Accumulator | AX | A |
| Base | BX | b |
| Count | CX | C |
| Data | DX | d |
| Stack Pointer | SP | SP |
| Base Pointer | BP | bP |
| Source Index | SI | SI |
| Destination Index | DI | dI |
| Code Segment | CS | CS |
| Data Segment | DS | dS |
| Stack Segment | SS | SS |
| Extra Segment | ES | ES |
| Instruction Pointer | IP | IP |
| Flag | FG | FLG |

When the register contents are displayed the register contents can be modified. Key in the new value from the keypad and the register contents will be updated when the [INR] key is pressed. If [EXEC] key is pressed the command is terminated and command prompt is displayed. If [INR] key is pressed, the next register with its contents will be displayed. If [DCR] key is pressed then the previous register contents are displayed.

**EXAMPLES :**

**1.** Examining a register.

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❒ F r I | E n d ❒ | Sign-on |
| REG | r E G r | ❒ ❒ ❒ ❒ | Examine and modify register command |
| A<br>VR / SS | ❒ ❒ S S | 0 0 0 0 | SS register |
| INR | ❒ ❒ E S | 0 0 0 0 | ES register |
| INR | ❒ ❒ I P | 0 0 0 0 | IP register |
| INR | F L G ❒ | x x x x | Flag register |
| INR | ❒ ❒ ❒ A | x x x x | AX register |
| INR | ❒ ❒ ❒ B | x x x x | BX register |
| EXEC | F ❒ ❒ ❒ | ❒ ❒ ❒ ❒ | Command terminated. |

**2.** Modifying registers DX to 55 AA.

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❒ F r I | E n d ❒ | Sign-on |
| 3<br>DX | ❒ ❒ ❒ d | 0 0 0 0 | DX register |
| 5<br>IW / BP | ❒ ❒ ❒ d | ❒ ❒ ❒ 5 | |
| 5<br>IW / BP | ❒ ❒ ❒ d | ❒ ❒ 5 5 | |

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| A<br>VR/SS | ❐ ❐ ❐ d | ❐ 5 5 A | |
| A<br>VR/SS | ❐ ❐ ❐ ❐. | 5 5 A A | New value |
| INR | ❐ ❐ S P | 0 0 0 0. | Stack segment register contents |
| EXEC | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | Command terminated |

## IB, IW (Input Byte and Input Word) Commands

**Function :**

The Input Byte [IB] and Input Word [IW] commands are used to input an 8 bit or 16 bit data from an input port.

**Operation :**

To use the Input Byte or Word command, press the corresponding hexadecimal key when prompted for command entry. When the key is pressed "Addr" appears in the address field to indicate that a port address entry is required. Using the keypad enter the port address to be read. I/O port address range is from 0000H to FFFFH, hence no segment value is permitted with the port address.

After the port address is entered, press the [INR] key. The input byte or word will be displayed in the data field. If [INR] is pressed again, the data is updated at the address input port. The [EXEC] key terminates the command and prompts for command entry.

There are two 8255 chips present onboard. The ports of 8255 (#1) are designated as PAL, PBL and PCL for Ports A, B and C. The port addresses are given below.

| 8255 (#1) Ports | I/O Address |
|---|---|
| A | 60 |
| B | 62 |
| C | 64 |
| Control word | 66 |

**EXAMPLES :**

**1.** Byte input from address 60H 8255# 1 A port. (After initialising Port A of 8255 (#1) as input)

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| 4<br>IB / SP | A d d r | x x x x | |
| 6<br>OB / SI | A d d r | 0 0 0 6 | |
| 0<br>EB / AX | A d d r | 0 0 6 0 | Port address |
| INR | d a t a | ❐ ❐ x x | Input data |
| INR | d a t a | ❐ ❐ x x | New Input data |
| EXEC | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | Command termination |

**2.** Word input from address 60 port A of both the 8255 chips.

| **KEY** | **ADDRESS FIELD** | **DATA FIELD** | **COMMENT** |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| 5<br>IW / BP | A d d r | x x x x | |
| 6<br>OB / SI | A d d r | 0 0 0 6 | |
| 0<br>EB / AX | A d d r | 0 0 6 0 | Port address |
| INR | d a t a | x x x x | Input data |
| INR | d a t a | x x x x | New Input data |
| EXEC | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | Command termination |

## OB, OW (Output Byte and Output Word) Commands

**Function :**

The Output Byte [OB] or Output Word [OW] commands are used to output a byte or word to an output port.

**Operation :**

To use the Output Byte or Word command, press the corresponding hexadecimal key when prompted for command entry. When the key is pressed "Addr" is displayed in the address field to indicate that a port address entry is required. Using the keypad enter the port address. I/O Port address range is from 0000H to FFFFH, hence no segment value is asked with the port address.

After the port address is entered, press the [INR] key. "data" is displayed in the address field indicates that the data to be outputted can be entered. Using the keypad enter the data byte or word. After entering the data press [EXEC], to output the data to the port and terminate the command. [INR] key outputs the next data to the addressed port. The port addresses of the 8255 (#1) chip are :

| 8255 (# 1) Ports | I/O Address |
|---|---|
| A | 60 |
| B | 62 |
| C | 64 |
| Control word | 66 |

**EXAMPLES :**

**1.** Output 55H and then AA to Port A of 8255#1.

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❒ F r I | E n d ❒ | Sign-on |
| 6<br>OB / SI | A d d r | x x x x | |
| 6<br>OB / SI | A d d r | 0 0 0 6 | |
| 0<br>EB / AX | A d d r | 0 0 6 0 | Port address |
| INR | d A t A | ❒ ❒ x x | |
| 5<br>IW / BP | d A t A | ❒ ❒ 0 5 | |

| | | | |
|---|---|---|---|
| 5<br>IW/BP | d A t A | ❐ ❐ 5 5 | |
| INR | d A t A | ❐ ❐ 5 5 | 55H is outputted to port 60H |
| A<br>VR/SS | d A t A | ❐ ❐ 0 A | |
| A<br>VR/SS | d A t A | ❐ ❐ A A | |
| EXEC | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | AAH is outputed to port 60H & Command terminated |

## GO Command

**Function :**

[GO] command is used to execute user program from the address stated in this command.

**Operation :**

The [GO] key is pressed when prompted for a command entry and the current CS register contents are displayed in the data field. SEG in the address indicates that a new seg value can be entered from the keypad. Press INR or EXEC or DCR to enter the start i.e. offset value of the program. To begin program execution, press the [EXEC] key. "E" is displayed in the most significant digit of the address field and the program is executed. To illustrate the operation of [GO] command, the following sample program can be entered using the Examine Byte command from memory location 0:1000H. This program initiaties rolling of the sign-on message ie. Dyna 86. This program continues until the [RES] or [ INT] key is pressed.

```
0:  1000        B0 00              MOV AL,00
    1002        E6 52              OUT 52, AL
    1004        BB 00 00           MOV BX, D
    1007        8E DB              MOV DS, BX
    1009        BB 00 20           MOV BX, 2000
```

```
100C    89 1E 00 30    MOV [3000], BX
1010    BB 08 20       MOV BX, 2008
1013    89 1E 02 30    MOV [3002], BX
1017    8B 1E 00 30    MOV BX, [3000]
101B    8A 07          MOV AL, [BX]
101D    E6 50          OUT 50, AL
101F    43             INC BX
1020    3B 1E 02 30    CMP BX, [3002]
1024    75 F5          JNZ 101B
1026    8B 1E 00 30    MOV BX, [3000]
102A    43             INC BX
102B    89 1E 00 30    MOV [3000], BX
102F    8B 1E 02 30    MOV BX, [3002]
1033    43             INC BX
1034    89 1E 02 30    MOV [3002], BX
1038    E8 08 00       CALL 1043
103B    81 FB 10 20    CMP BX, 2010
103F    75 D6          JNZ 1017
1041    EB C6          JMP 1009
1043    BA 03 00       MOV DX, 0003
1046    B9 FF FF       MOV CX, FFFF
1049    49             DCE CX
104A    75 FD          JNZ 1049
104C    4A             DEC DX
104D    75 F7          JNZ 1046
104F    C3             RET
```

Enter the following data (DYNA 86) at memory location 0:2000H

```
0:2000  00  00  00  00  00  00  00  00  7D  7F  40  77  54  6E  5E  00
```

**NOTE :** *After the program is entered in memory, it will remain in memory even if the power is turned off because of the powerful battery back up provided to SRAM.*

To exit from the program being executed and return the control to the monitor, press either [RES] or [INT] keys. If [RES] key is pressed, the monitor is re-entered and the appropriate 8086 registers are initialized. If [INT] key is pressed, the monitor is re-entered and all the 8086 register contents are saved and the monitor prompts for a command entry. Now if the [GO] key is pressed, the current IP register value (offset address of the next program instruction to be executed, when the program was interrupted by the [INT] key) and the byte contents of that location (addressed by both IP and CS registers) are displayed. Pressing the [EXEC] key transfers control to the monitor program at the instruction addressed and the program execution continues.

The [GO] command optionally permits a ‘breakpoint address’ to be entered. A breakpoint address has the same effect as pressing the [INT] key while the program is executed. After entering the starting address, press the [INR] key to enter the breakpoint address.

When the breakpoint address is specified, the default segment value is the starting address segment or the current CS register contents. In addition the location specified by the breakpoint address must contain the first (opcode or prefix) byte of the instruction. When the [EXEC] key is pressed, the monitor replaces the instruction at the breakpoint address with an interrupt instruction and saves the breakpointed instruction before transfering control to the user program. When the program reaches the breakpoint address, control is returned to the monitor, the breakpointed instruction is restored in the program, all registers are saved and the monitor displays address of the break point and data field shows the data at that address.

**NOTE :** *If used, the breakpoint address must be specified each time the program is to be executed with a breakpoint.*

**EXAMPLES:**

**1.** Execute a program stored at memory location 1000H.

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| GO | S E G ❐ | 0 0 0 0 | GO command (IP reg. offset addr. and data contents) |
| INR | S t r t | 0 0 0 0 | |
| 1<br>EW / BX | S t r t | 0 0 0 1 | |
| 0<br>EB / AX | S t r t | 0 0 1 0 | |
| 0<br>EB / AX | S t r t | 0 1 0 0 | |
| 0<br>EB / AX | S t r t | 1 0 0 0 | |
| EXEC | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | DYNA-86 message will start rolling. |

**2.** Entering and executing a breakpoint in the sample program.

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| GO | S E G ❐ | 0 0 0 0 | GO command (IP reg. offset addr. and data contents) |
| INR | S t r t | 0 0 0 0 | |

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| 1<br>EW / BX | Strt | 0001 | |
| 0<br>EB / AX | Strt | 0010 | |
| 0<br>EB / AX | Strt | 0100 | |
| 0<br>EB / AX | Strt | 1000 | |
| INR | SEG❒ | xxxx | Enter break point at seg 0H & offset 1007H |
| 0<br>EB / AX | SEG❒ | 0000 | |
| INR | br❒❒ | xxxx | |
| 1<br>EW / BX | br❒❒ | 0001 | |
| 0<br>EB / AX | br❒❒ | 0010 | |
| 0<br>EB / AX | br❒❒ | 0100 | |
| 7<br>OW / DI | br❒❒ | 1007 | |
| EXEC | 1007 | 8Edb | Breakpoint at 0:1007 |
| INR | 100C | 891E | |
| EXEC | F❒❒❒ | ❒❒❒❒ | |

# BK Set Breakpoint

**Function :**

The [BK] command allows to set the breakpoint anywhere in the program stored in RAM.

**Operation :**

The operation of this command is same as in [GO] command i.e. to set the break point. When this key is pressed, the user is prompted for segment and offset value where breakpoint is to be inserted. After entering breakpoint address run the program using the [GO] command. When the program reaches the break point address the control is transferred to monitor program. The address field will show the breakpoint address and data in the data field at that address. User can single step the program using [INR] key or go to command mode using [EXEC] key.

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❒ F r I | E n d ❒ | Sign-on |
| 2<br>BK / CX | S E G ❒ | 0 0 0 0 | Segment offset |
| INR | b r ❒ ❒ | x x x x | |
| 1<br>EW / BX | b r ❒ ❒ | 0 0 0 1 | |
| 0<br>EB / AX | b r ❒ ❒ | 0 0 1 0 | |
| 0<br>EB / AX | b r ❒ ❒ | 0 1 0 0 | |
| 7<br>OW / DI | b r ❒ ❒ | 1 0 0 7 | Breakpoint offset address |
| EXEC | F ❒ ❒ ❒ | ❒ ❒ ❒ ❒ | |

Now if the program from 0:1000 is executed using GO command then,

| | | | |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| GO | S E G ❐ | 0 0 0 0 | GO command (IP reg. offset addr. and data contents) |
| INR | S t r t | 0 0 0 0 | |
| 1<br>EW / BX | S t r t | 0 0 0 1 | |
| 0<br>EB / AX | S t r t | 0 0 1 0 | |
| 0<br>EB / AX | S t r t | 0 1 0 0 | |
| 0<br>EB / AX | S t r t | 1 0 0 0 | |
| EXEC | 1 0 0 7 | 8 E d b | |
| INR | 1 0 0 C | 8 9 1 E | |
| EXEC | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | |

## STEP Command

**Function :**
The STEP command permits program instructions in memory to be executed individually. With each instruction executed, the program returns to the monitor.

**Operation :**

To use the Step command, press the [STEP] key when prompted for command entry. If the segment address is other than the address displayed, enter the required segment address. Then press [INR], [DCR] or [EXEC] key. Monitor prompts for offset i.e. Strt. Enter the start address. Now if the [INR] key is pressed, the instruction addressed is executed and the offset of the next instruction to be executed is displayed in the address field and its data in the data field. Again if the [INR] key is pressed the next instruction is executed and steps the program to the next instruction.

In order to use the Step command run the same sample program of the rolling sign-on message.

**RESTRICTIONS :**

➤ If an interrupt occurs prior to the completion of a single stepped instruction or if a single-stepped instruction generates an interrupt, when the monitor is reentered, CS and IP registers will contain the address of the interrupt service routine. Consequently a type 3 (breakpoint) interrupt instruction (0CC or 0CDH) should not be single stepped since its execution will step into the monitor.

➤ An instruction that is part of a sequence of instructions that switches between stack segments (i.e. changes the SS and SP register contents) cannot be single stepped.

➤ A MOV or POP instruction that modifies a segment register cannot be single stepped. Control is returned to the monitor after the next instruction (instruction that immediately follows the MOV and POP instruction) is executed.

**EXAMPLES :**

**1.** Program stepping. Run the same program used for the GO command. ie. Rolling the sign-on message. Single step the first few commands.

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| STEP | S E G ❐ | 0 0 0 0 | |
| INR | S t r t | 0 0 0 0 | |
| 1<br>EW / BX | S t r t | 0 0 0 1 | |
| 0<br>EB / AX | S t r t | 0 0 1 0 | |
| 0<br>EB / AX | S t r t | 0 1 0 0 | |
| 0<br>EB / AX | S t r t | 1 0 0 0 | |
| INR | 1 0 0 2 | E 6 5 2 | |
| INR | 1 0 0 4 | b b 0 0 | |
| INR | 1 0 0 7 | 8 E d b | |
| EXEC | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | Command prompt. Terminate Single step |

***NOTE :*** *For details of EPROM programming facility refer DYNA-PIO PGMR manual for DYNA-86.*

## CODE

[CODE] key is different from all other keys. It is used to select 1 out of 16 different utility programs stored in monitor EPROM. They are :

| Utility name | Key No. | Addr. field | Data field |
|---|---|---|---|
| Copy | 0 | Code | CPY |
| Fill byte | 1 | Code | FILb |
| Fill word | 2 | Code | FILL |
| Insert B/W | 3 | Code | Ins |
| Delete byte | 4 | Code | dEL |
| Search B/W | 5 | Code | Srch |
| Compare | 6 | Code | CPr |
| Hex to Decimal | 7 | Code | H2d |
| Decimal to Hex | 8 | Code | d2H |
| Hexadecimal Add | 9 | Code | Add |
| Hexadecimal Subtract | A | Code | SUb |
| Hexadecimal Multiplication | B | Code | Into |
| Hexadecimal division | C | Code | DIV |
| User code 1 (INT F2) | D | Code | USr 1 |
| User code 2 (INT F3) | E | Code | USr 2 |
| Rolling message | F | Code | dYnA |

The programs can also be selected through [INR] or [DCR] keys. With [INR] key after dYnA, CPY utility appears and with [DCR] key, after CPY, dYnA appears in the data field. To execute any of these utilities press ’EXEC key when the utility name is displayed in the data field.

**Code : COPY**

This utility copies a block of data from source memory block to the destination memory block.

Input required is Source start (segment and offset)
Source end (offset)
Destination start (segment and offset)

The example given below explains how to transfer a block of data from (0:1000 to 0:10FF) to destination memory block (0:2000).

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❒ F r I | E n d ❒ | Sign-on |
| CODE | C o d E | C P Y | |
| EXEC | S E G ❒ | x x x x | Give source start segment |
| 0<br>EB / AX | S E G ❒ | 0 0 0 0 | |
| INR | S t r t | x x x x | Source start offset |
| 1<br>EW / BX | S t r t | 0 0 0 1 | |
| 0<br>EB / AX | S t r t | 0 0 1 0 | |
| 0<br>EB / AX | S t r t | 0 1 0 0 | |
| 0<br>EB / AX | S t r t | 1 0 0 0 | |
| INR | E n d ❒ | x x x x | Source end offset<br>No segment is permitted hence block copy is limited to 64 KB boundary |
| 1<br>EW / BX | E n d ❒ | 0 0 0 1 | |
| 0<br>EB / AX | E n d ❒ | 0 0 1 0 | |
| F<br>DYNA | E n d ❒ | 0 1 0 F | |
| F<br>DYNA | E n d ❒ | 1 0 0 F | |
| INR | S E G ❒ | x x x x | Destination start segment |

| | | | |
|---|---|---|---|
| 0<br>EB / AX | S E G ❐ | 0 0 0 0 | |
| INR | d E s t | x x x x | Destination start offset |
| 2<br>BK / CX | d E s t | 0 0 0 2 | |
| 0<br>EB / AX | d E s t | 0 0 2 0 | |
| 0<br>EB / AX | d E s t | 0 2 0 0 | |
| 0<br>EB / AX | d E s t | 2 0 0 0 | |
| EXEC | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | Command prompt. |

Error : Destination is ROM or non-existent memory.

**Code : Fill byte**

To fill a block of memory with required data this utility can be used.

**Example :**

Fill memory block 0:1000 to 0:1100 with 55H

| **KEY** | **ADDRESS FIELD** | **DATA FIELD** | **COMMENT** |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| CODE | C o d E | C P Y | |
| 1<br>EW / BX | C o d E | F I L b | Select fill byte |

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| EXEC | S E G ❐ | x x x x | Run fill byte |
| 0<br>EB / AX | S E G ❐ | 0 0 0 0 | n block start segment |
| INR | S t r t | x x x x | n block start offset |
| 1<br>EW / BX | S t r t | 0 0 0 1 | |
| 0<br>EB / AX | S t r t | 0 0 1 0 | |
| 0<br>EB / AX | S t r t | 0 1 0 0 | |
| 0<br>EB / AX | S t r t | 1 0 0 0 | |
| INR | E n d ❐ | x x x x | n block end offset |
| 1<br>EW / BX | E n d ❐ | 0 0 0 1 | |
| 1<br>EW / BX | E n d ❐ | 0 0 1 1 | |
| 0<br>EB / AX | E n d ❐ | 0 1 1 0 | |
| 0<br>EB / AX | E n d ❐ | 1 1 0 0 | |
| INR | d A t A | ❐ ❐ x x | Data byte |
| 5<br>IW / BP | d A t A | ❐ ❐ 0 5 | |
| 5<br>IW / BP | d A t A | ❐ ❐ 5 5 | |
| EXEC | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | Command prompt. |

### Code : Fill word

This is similar to fill byte, the only difference is that instead of byte a word location is filled with required value.

**Example :**

Fill (0:1000 to 0:1100) block of memory with 1122H.

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❒ F r I | E n d ❒ | Sign-on |
| CODE | C o d E | C P Y | |
| 2<br>BK / CX | C o d E | F I L L | Select fill word |
| EXEC | S E G ❒ | x x x x | Block start segment |
| 0<br>EB / AX | S E G ❒ | 0 0 0 0 | |
| INR | S t r t | x x x x | Block start offset |
| 1<br>EW / BX | S t r t | 0 0 0 1 | |
| 0<br>EB / AX | S t r t | 0 0 1 0 | |
| 0<br>EB / AX | S t r t | 0 1 0 0 | |
| 0<br>EB / AX | S t r t | 1 0 0 0 | |
| INR | E n d ❒ | x x x x | Block end offset |
| 1<br>EW / BX | E n d ❒ | 0 0 0 1 | |

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| 1<br>EW / BX | E n d ❐ | 0 0 1 1 | |
| 0<br>EB / AX | E n d ❐ | 0 1 1 0 | |
| 0<br>EB / AX | E n d ❐ | 1 1 0 0 | |
| INR | d A t A | x x x x | Data word |
| 1<br>EB / AX | d A t A | 0 0 0 1 | |
| 1<br>EB / AX | d A t A | 0 0 1 1 | |
| 2<br>EW / BX | d A t A | 0 1 1 2 | |
| 2<br>EW / BX | d A t A | 1 1 2 2 | |
| EXEC | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | Command prompt. |

### Code : Insert byte

As name indicates this utility is used to insert a byte any where in a block of memory.

Example : Set the memory location from 0:1000 to 0:1010 with "EB" command as below.

0:1000 00, 01, 02, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F, 55

0:1010 FF

| | | | |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| CODE | C o d E | C P Y | |

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| 3<br>DX | C o d E | I n S ❐ | Select insert byte |
| EXEC | S E G ❐ | x x x x | |
| 0<br>EB / AX | S E G ❐ | 0 0 0 0 | |
| INR | S t r t | x x x x | offset where byte is to be inserted |
| 1<br>EW / BX | S t r t | 0 0 0 1 | |
| 0<br>EB / AX | S t r t | 0 0 1 0 | |
| 0<br>EB / AX | S t r t | 0 1 0 0 | |
| 3<br>DX | S t r t | 1 0 0 3 | |
| INR | E n d ❐ | x x x x | End of block to be shifted 1 byte down |
| 1<br>EW / BX | E n d ❐ | 0 0 0 1 | |
| 0<br>EB / AX | E n d ❐ | 0 0 1 0 | |
| 0<br>EB / AX | E n d ❐ | 0 1 0 0 | |
| E<br>PIO | E n d ❐ | 1 0 0 E | |
| INR | d A t A | ❐ ❐ x x | Data to be inserted |
| 3<br>DX | d A t A | ❐ ❐ 0 3 | |
| EXEC | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | Command prompt. |

Result : Examine the memory location from 0:1000 to 0:1010 using "EB" command.

0:1000 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F

0:1010 FF

**Code : Delete byte**

This utility is opposite of the Insert byte utlity. It deletes a data byte from the given start address and shifts 1 byte up the remaining block.

**Example :**

Set the memory location from 0:1000 to 0:1010 with "EB" command as below.

0:1000 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F

0:1010 55

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| CODE | C o d e | C P Y | |
| 4<br>IB/SP | C o d e | d E L ❐ | Select delete byte |
| EXEC | S E G ❐ | x x x x | Segment of block |
| 0<br>EB/AX | S E G ❐ | 0 0 0 0 | |
| INR | S t r t | x x x x | offset from where byte is to be deleted |
| 1<br>EW/BX | S t r t | 0 0 0 1 | |

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| 0<br>EB/AX | Strt | 0010 | |
| 0<br>EB/AX | Strt | 0100 | |
| 3<br>DX | Strt | 1003 | |
| INR | End❐ | xxxx | End of block to be shifted 1 byte up |
| 1<br>EW/BX | End❐ | 0001 | |
| 0<br>EB/AX | End❐ | 0010 | |
| 0<br>EB/AX | End❐ | 0100 | |
| F<br>DYNA | End❐ | 100F | |
| EXEC | F❐❐❐ | ❐❐❐❐ | Command prompt. |

Result : Examine the memory location from 0:1000 to 0:1010 using "EB" command.

| | |
|---|---|
| 0:1000 | 00, 01, 02, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F, 0F |
| 0:1010 | 55 |

**Code : Search byte**

Search byte utility can be used to search required data byte in a block of memory. Block size is limited to 64 KB.

**Example :** Search data byte 55H from 0:1000 to 0:1100H. Fill 0:1000 to 0:1100 with 00 using fill byte utility. Set the following memory location with 55H using "EB" command.

```
0:1004 55
0:1008   55
0:1009   55
0:1036   55
0:10FE   55
```

| **KEY** | **ADDRESS FIELD** | **DATA FIELD** | **COMMENT** |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| CODE | C o d E | C P Y | |
| 5<br>IW/BP | C o d E | S r C H | Select search byte |
| EXEC | S E G ❐ | x x x x | Segment of block start |
| 0<br>EB / AX | S E G ❐ | 0 0 0 0 | |
| INR | S t r t | x x x x | Offset of block start |
| 1<br>EW / BX | S t r t | 0 0 0 1 | |
| 0<br>EB / AX | S t r t | 0 0 1 0 | |
| 0<br>EB / AX | S t r t | 0 1 0 0 | |
| 0<br>EB / AX | S t r t | 1 0 0 0 | |
| INR | E n d ❐ | x x x x | Offset of end of block |
| 1<br>EW / BX | E n d ❐ | 0 0 0 1 | |

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| 1<br>EW / BX | E n d ❐ | 0 0 1 1 | |
| 0<br>EB / AX | E n d ❐ | 0 1 1 0 | |
| 0<br>EB / AX | E n d ❐ | 1 1 0 0 | |
| INR | d A t A | ❐ ❐ x x | Data byte to be searched |
| 5<br>IW / BP | d A t A | ❐ ❐ 0 5 | |
| 5<br>IW / BP | d A t A | ❐ ❐ 5 5 | |
| INR | 1 0 0 4 | ❐ ❐ 5 5 | |
| INR | 1 0 0 8 | ❐ ❐ 5 5 | |
| INR | 1 0 0 9 | ❐ ❐ 5 5 | |
| INR | 1 0 3 6 | ❐ ❐ 5 5 | |
| INR | 1 0 F E | ❐ ❐ 5 5 | |
| INR | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | No more data required in specified memory block. |

**Code : Compare block**

This command is used to compare the block of memory byte by byte.

**Example :**

To compare 0:1000 to 0:10FF block of data with 0:1100 to 0:11FF
Fill 0:1000 to 0:11FF with data 55H using fill utility.

Set 0:1008 with 01
1080 with 02
10B3 with 03
1133 with 04
11FD with 05 using "EB" command.

| **KEY** | **ADDRESS FIELD** | **DATA FIELD** | **COMMENT** |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| CODE | C o d E | C P Y | |
| 6<br>OB / SI | C o d E | C p r | Select search byte |
| EXEC | S E G ❐ | x x x x | Compare from Segment |
| 0<br>EB / AX | S E G ❐ | 0 0 0 0 | |
| INR | S t r t | x x x x | Compare from offset |
| 1<br>EW / BX | S t r t | 0 0 0 1 | |
| 0<br>EB / AX | S t r t | 0 0 1 0 | |
| 0<br>EB / AX | S t r t | 0 1 0 0 | |
| 0<br>EB / AX | S t r t | 1 0 0 0 | |
| INR | E n d ❐ | x x x x | Compare upto end offset |
| 1<br>EW / BX | E n d ❐ | 0 0 0 1 | |

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| 0<br>EB / AX | S t r t | 0 0 1 0 | |
| F<br>DYNA | S t r t | 0 1 0 F | |
| F<br>DYNA | S t r t | 1 0 F F | |
| INR | S E G ❐ | x x x x | Compare to segment |
| 0<br>EB / AX | S E G ❐ | 0 0 0 0 | |
| INR | d E S t | x x x x | Compare to start offset |
| 1<br>EW / BX | d E S t | 0 0 0 1 | |
| ✑<br>EW / BX | S t r t | 0 0 1 1 | |
| 0<br>EB / AX | S t r t | 0 1 1 0 | |
| 1<br>EW / BX | S t r t | 1 1 0 0 | |
| EXEC | 1 0 0 8 | 0 1 5 5 | The MSD i.e. 01 is the contents of 0:1008 and 55 is the content of 0:1108 |
| INR | 1 0 3 3 | 5 5 0 4 | The MSD i.e. 55 is the contents of 0:1033 and 04 is the content of 0:1133 |
| INR | 1 0 8 0 | 0 2 5 5 | The MSD i.e. 02 is the contents of 0:1080 and 55 is the content of 0:1180 |

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| INR | 1 0 B 3 | 0 3 5 5 | The MSD i.e. 03 is the contents of 0:10B3 and 55 is the content of 0:11B3 |
| INR | 1 0 F D | 5 5 0 5 | The MSD i.e. 55 is the contents of 0:10FD and 05 is the content of 0:11FD |
| INR | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | Command prompt<br>The given block of data is compared. |

Result : After entering the source start, end and destination address, pressing [INR] key shows the offset address and the contents of memory location of source and destination if they are not matching. The MSD shows the source data whereas LSD shows destination data.

**Code : Hex to decimal**

Converts a hexadecimal number to decimal no.

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| CODE | C o d E | C P Y | |
| 7<br>OB / SI | C o d E | H 2 d | Select  Hex to decimal |
| EXEC | d A t A | x x x x | |
| 1<br>EB / AX | d A t A | 0 0 0 1 | |
| 2<br>EW / BX | d A t A | 0 0 1 2 | Hex value |
| INR | 0 0 0 0 | 0 0 1 8 | Decimal conversion |

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| INR | d A t A | x x x x | |
| 5<br>IW / BP | d A t A | 0 0 0 5 | |
| 0<br>EB / AX | d A t A | 0 0 5 0 | Hex value |
| INR | 0 0 0 0 | 0 0 8 0 | Decimal conversion |
| EXEC | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | Command prompt |

**Code : Decimal to Hex**

To convert a decimal number to hexadecimal.

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| CODE | C o d E | C P Y | |
| 8<br>PR/CS | C o d E | d 2 H | Select Hex to decimal |
| EXEC | d A t A | x x x x | |
| 2<br>GO/CX | d A t A | 0 0 0 2 | Enter decimal no. 256 |
| 5<br>IW / BP | d A t A | 0 0 2 5 | |
| 6<br>OB / SI | d A t A | 0 2 5 6 | |

| **KEY** | **ADDRESS FIELD** | **DATA FIELD** | **COMMENT** |
|---|---|---|---|
| INR | d A t A | 0 1 0 0 | Converted Hex value of 256 is 100H |
| INR | d A t A | x x x x | |
| EXEC | F ❐ ❐ ❐ | ❐ ❐ ❐ ❐ | Command prompt |

**Code : Hexadecimal addition**

This command is used for hexadecimal addition. The result is limited to 2 bytes i.e. 4 nibble. Carry out of 4 nibble is discarded.

**Example :**

Add numbers n1 to n2

| **KEY** | **ADDRESS FIELD** | **DATA FIELD** | **COMMENT** |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| CODE | C o d E | C P Y | |
| 9<br>SE/DS | C o d E | A d d | Select Addition |
| EXEC | n 1 ❐ ❐ | x x x x | |
| 3<br>DX | n 1 ❐ ❐ | 0 0 0 3 | |
| 0<br>EB/AX | n 1 ❐ ❐ | 0 0 3 0 | Adder |
| INR | n 2 ❐ ❐ | x x x x | |
| 1<br>EW / BX | n 2 ❐ ❐ | 0 0 0 1 | |

| | | | |
|---|---|---|---|
| 0<br>EB/AX | n 2 ❐ ❐ | 0 0 1 0 | Adder |
| INR | r e S ❐ | 0 0 4 0 | Result |

**Code : Hexadecimal subtraction**

This command is used for hexadecimal subtraction.

**Example :**
Subtract number n2 from n1.

| **KEY** | **ADDRESS FIELD** | **DATA FIELD** | **COMMENT** |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| CODE | C o d E | C P Y | |
| A<br>VR / SS | C o d E | S u b | Select subraction |
| EXEC | n 1 ❐ ❐ | x x x x | |
| 3<br>DX | n 1 ❐ ❐ | 0 0 0 3 | |
| 0<br>EB/AX | n 1 ❐ ❐ | 0 0 3 0 | Subtractor |
| INR | n 2 ❐ ❐ | x x x x | |
| 1<br>EW / BX | n 2 ❐ ❐ | 0 0 0 1 | |
| 0<br>EB/AX | n 2 ❐ ❐ | 0 0 1 0 | Subtractor |
| INR | r e S ❐ | 0 0 2 0 | Result |

## Code : Hexadecimal Multiplication

This command is used for Hexadecimal multiplication.

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| CODE | C o d E | C P Y | |
| B<br>BC / ES | C o d E | I n t O | Select multiplication |
| EXEC | n 1 ❐ ❐ | x x x x | |
| 1<br>EW/BX | n 1 ❐ ❐ | 0 0 0 1 | |
| 0<br>EB/AX | n 1 ❐ ❐ | 0 0 1 0 | Multiplicant |
| INR | n 2 ❐ ❐ | x x x x | |
| 5<br>IW/BP | n 2 ❐ ❐ | 0 0 0 5 | Multiplier |
| INR | r e S ❐ | 0 0 5 0 | Result |

## Code : Hexadecimal Division

This command is used for Hexadecimal division.

| | | | |
|---|---|---|---|
| RES | ❐ F r I | E n d ❐ | Sign-on |
| CODE | C o d E | C P Y | |
| C<br>BC / ES | C o d E | d I V ❐ | Select division |

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| EXEC | n 1 ❐ ❐ | x x x x | |
| 5<br>IW/BP | n 1 ❐ ❐ | 0 0 0 5 | |
| 0<br>EB / AX | n 1 ❐ ❐ | 0 0 5 0 | Dividend |
| INR | n 2 ❐ ❐ | ❐ ❐ x x | |
| 5<br>IW/BP | n 2 ❐ ❐ | ❐ ❐ 0 5 | Divisor |
| INR | r e S ❐ | 0 0 1 0 | Quotient |

### Code : User 1 and User 2

This utility is directed to 8086 software interrupt INT F2 & INT F3.
User 1 directed to INT F2
User 2 directed to INT F3.

The segment and offset of the user application to be executed using the [CODE] key is to be supplied at the location given below.

| | | |
|---|---|---|
| INT F2 | 0 : 3C8 | Lower byte of User program’s offset address i.e. IP(L) |
| | 0 : 3C9 | Higher byte of User program’s offset address i.e. IP(H) |
| | 0 : 3CA | Lower byte of User program’s segment address i.e. CS(L) |
| | 0 : 3CB | Higher byte of User program’s segment address i.e. CS(H) |
| INT F2 | 0 : 3CC | Lower byte of User program’s offset address i.e. IP(L) |
| | 0 : 3CD | Higher byte of User program’s offset address i.e. IP(H) |
| | 0 : 3CE | Lower byte of User program’s segment address i.e. CS(L) |
| | 0 : 3CF | Higher byte of User program’s segment address i.e. CS(H) |

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❒ F r I | E n d ❒ | Sign-on |
| CODE | C o d E | C P Y | |
| D<br>CK/FG | C o d E | U S r 1 | |
| EXEC | The system will execute INT F2. | | |
| RES | ❒ F r I | E n d ❒ | Sign-on |
| CODE | C o d E | C P Y | |
| E<br>PIO | C o d E | U S r 2 | |
| EXEC | The system will execute INT F3. | | |

***NOTE :** On reset or power-on INT F2 & INT F3 vector is initiliased to rolling display.*

**Code : Rolling display**

This is the demo program to roll the message "dYnALOG HELPS YOU In LEArnInG UP" on the seven segment LED display.

| KEY | ADDRESS FIELD | DATA FIELD | COMMENT |
|---|---|---|---|
| RES | ❒ F r I | E n d ❒ | Sign-on |
| CODE | C o d E | C P Y | |
| F<br>DYNA | C o d E | D Y n A | |
| EXEC | | | |

# CHAPTER 5
# SERIAL MODE

## INTRODUCTION

The **SERIAL** mode of DYNA-86 contains almost all the commands available in the **HEX KEYPAD** mode, assembler and disassembler. The serial communication takes place through RS-232C compatible port which can be integrated with a CRT terminal or a computer like IBM-PC emulating a terminal. After entering this mode all the keys on the Hex Keypad except [RES] and [INT] are disabled in the **SERIAL** mode. All commands are typed on the terminal keyboard and results are displayed on the terminal screen.

This chapter describes the command set and command formats supported by the **SERIAL** mode. On Power-On or Reset, a jump to the **HEX KEYPAD** mode takes place and control is passed to its monitor. To switch to the **SERIAL** mode,

1) Press a key [SER], display shows b A U d in the address field and initial baud rate "3 0 0" in the data field.
2) Press the key corresponding the desired baud rate given in the following table :

| Baud Rate | Key |
|---|---|
| 300 | 0 |
| 600 | 1 |
| 1200 | 2 |
| 2400 | 3 |
| 4800 | 4 |
| 9600 | 5 |

or press [INR] or [DCR] to select the other baud rate.

***Note :*** The other parameters are as follows:

| | |
|---|---|
| Parity | :- No parity |
| data length | :- 8 bit data |
| stop bit | :- 1 stop bit |

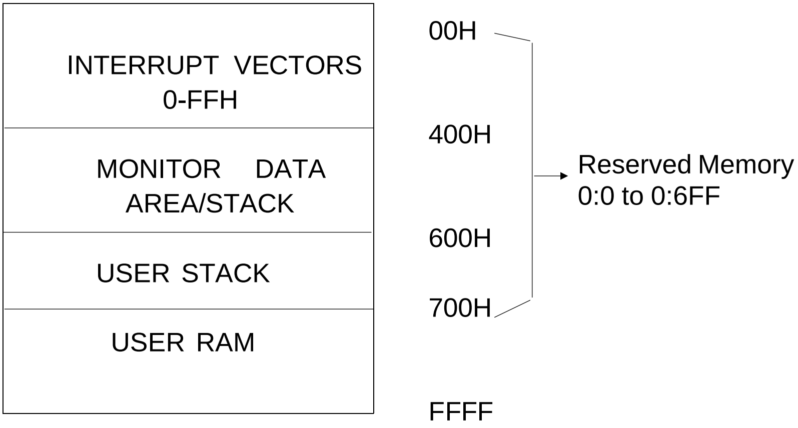
These parameters are fixed & cannot be changed.

3) Press [EXEC] key to enter serial mode, display shows `SErIAL` and terminal screen shows `DYNA-86`.

On Reset or Power-on, the monitor initialises the following parameters in its initialisation routine.

| | | |
|---|---|---|
| Interrupt 1 : Single Step | : | Used with the Single Step command |
| Interrupt B : IRQ3 | : | Invoked by [INT] key |
| Interrupt 3 : Breakpoint | : | Used with the Go command. |

The routine also initializes the general purpose registers of 8086 ie. CS, DS, SS, IP and FL registers to 0000H and the SP register to 6FF (base of the stack).
The memory map is as shown below :



**Memory Map**

**NOTE :** *The user program should start from 0000:0700H or above.*

Whenever the monitor is re-entered as the result of Single Step, INT or Breakpoint interrupt, the monitor temporarily stores the contents of the 8086 registers in RAM & subsequently restores the register contents from the memory before it prompts for command entry. Since the SP register is initialized to 6FFH, the initial stack reserved for the user is FF bytes (locations 600-6FF).

# USING THE IBM PC AS A TERMINAL

TANGO is a terminal emulation software which lets the IBM PC work as a Terminal. It is a menu driven software. Parameters like Terminal type, Baud rate, Start bit, Stop bit, Parity, can be selected in the menu. Procedure for connecting DYNA-86 in **SERIAL** mode to IBM PC is given in the following section.

### Procedure to connect a Terminal to DYNA-86

1. Connect a cable from the serial port of DYNA-86 to one of the serial ports of the IBM PC. The cable is made by connecting 3 wires between 9 pin and 9 or 25 pin D type female connectors as shown below :

| 9 PIN D-TYPE CONNECTOR<br>(DYNA-86 SIDE)<br>CONNECTOR J5 | | | 25 PIN D-TYPE CONNECTOR<br>(IBM SIDE) | | OR 9pin D-type<br>(IBM SIDE) |
|---|---|---|---|---|---|
| 3 | (TXD) | → | 3 | (RXD) | 2 (RXD) |
| 2 | (RXD) | ← | 2 | (TXD) | 3 (TXD) |
| 5 | (GND) | → | 7 | (GND) | 5 (GND) |

2. Execute the TANGO utility on the PC. Press F9 key. A menu will be displayed. Change baud rate as desired and select the COM port being used. The other parameters can be left to their default values. Press F9 key again. The IBM PC now works as a terminal.

3. On the DYNA-86 press [SER] key. Select baud rate. It goes into the Serial mode at the selected baud rate, 8 bit, no parity and 1 stop bit.

4. SErIAL is displayed on the seven segment LED display while the message `DYNA-86>` is displayed on the terminal. If not please check the serial cable or the parameter settings in TANGO.

DYNA-86 system is now ready to be used in the **SERIAL** mode from the terminal.

# COMMAND STRUCTURE

When the monitor is ready for a command entry, it outputs `DYNA-86>` at the beginning of a new line. This line is refered to as the "command line" and consists of either a one or two-character command mnemonics followed by one to three command parameters or "arguments." (If desired for visual separation, a space can be entered between the command mnemonic and the first argument). When more than one argument is present, a delimiter (" , ") or space is required. A command line is terminated by a carriage return. Only one command is permitted on a command line.

With the exception of the register abbreviations associated with the **R** (Examine/Modify Register) command, all arguments are entered as hexadecimal numbers. The valid range of hexadecimal values are from 00H to FFH for byte entries and from 0000H to FFFFH for word entries. If more than two (for byte) or four (for word) digits are entered, monitor reports appropriate error message. Address arguments consists of a segment value and an offset value. If a segment value is not entered, the default is the last contents of the segment register used in the previous command, unless specified otherwise in the command description. When both a segment value and an offset value are entered as an address argument, the first entry is the segment value, and the second entry is the offset value. A colon (":") is used as the separator.

Since command execution occurs only after a command terminator is entered. A command entry can be cancelled any time before the terminator is entered by pressing <ESC> key. When a command is cancelled, the command prompt "`DYNA-86>`" is output on the next line.

# SERIAL MODE COMMAND DESCRIPTION

The **SERIAL** mode supports following commands. Each command is detailed in the following sections. In the individual command descriptions, the following syntax is used :

| | |
|---|---|
| \| | This is intrpreted as OR |
| [A] | Item enclosed in the square brackets "A" is optional |
| <CR> | Indicates that a carriage return should be entered |

Note that the symbols "[ ], < >, |" are used only to clarify the command formats and they should not be entered as part of the command.

## H (Help) Command

**Function :**

Help command displays all the commands with their syntax, in serial mode.

**Syntax :**

```
H <CR>
```

List of Serial Commands

| **Command** | **Syntax** |
|---|---|
| Display Memory | D[W][[seg:]strt[,end]] |
| Edit Memory | E[W][[seg:]strt] |
| Copy Memory | C[seg:]strt,end,[seg:]dest |
| Fill Memory | F[W][seg:]strt,end,byte\|word |
| Insert Byte / Word | I[W][seg:]strt,end,byte\|word |
| Delete Byte / Word | DL[W][seg:]strt,end |
| Search Byte / Word | S[W][seg:]strt,end,byte\|word |

| | |
|---|---|
| Compare blocks | CM[seg:]strt,end,[seg:]strt |
| Register | R[reg] |
| Trace | T[[seg:]strt] |
| Go | G[[[seg:]strt],[[seg:]brk]] |
| Hex to Dec | HDnum |
| Dec to Hex | DHnum |
| Load file | L[seg:] |
| Write file | W[seg:],strt,end |
| Input from port | IN[W]addr |
| Output to port | O[W]addr,[byte|word] |
| Assemble | A[[seg:]off] |
| Unassemble | U[[seg:]off[,[seg:]off]] |

## D (Display Memory) Command

**Function :**

The Display Memory (**D**) command is used to display the contents of a block of memory at the terminal.

**Syntax :**

```
D[W][[seg:]strt[,end]]<CR>
```

**Operation :**

The command provides a line-formatted display of the memory block bounded by "start address" and "end address". The segment address is specified or implied for the start address while only the offset value can be specified for the "end address". Block size is consequently limited to 64K bytes or 32K words.

To use the Display Memory command, enter **`D`** (for byte) or **`DW`** (for word) when prompted for command entry and then enter "start address" of the memory data block, enter "end address" and a carriage return. Beginning on the next line the monitor will display the segment, the offset address and the data contents of consecutive locations separated by spaces. Each line consists of a maximum of either sixteen byte entries or eight word entries.

The Display Memory command can be canceled or the display can be stopped at any instant by entering control characters from the terminal keyboard. < ESC> immediately terminates the command and returns to the command entry mode. Space bar stops further display, but does not terminate the command. Any other key resumes the display.

**Error Conditions :**

End address less than the offset value of start address.

**EXAMPLES :**

```
Dyna-86>D10 <CR>
1401:0010  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1401:0020  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1401:0030  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1401:0040  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1401:0050  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1401:0060  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1401:0070  FF FF FF 48 FF FF FF FF FF FF FF FF FF FF FF FF
1401:0080  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1401:0090  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1401:00A0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

```
Dyna-86>D 0:15 26 <CR>
0000:0015  00 00 00 01 00 00 00 01 00 00 00
0000:0020  01 00 00 00 01 00 00
```

```
Dyna-86>DW 1000 <CR>
0000:1000  5555 5555 5555 5555 5501 5555 5555 5555
0000:1010  5555 5555 5555 5555 5555 5555 5555 5555
0000:1020  5555 5555 5555 5555 5555 5555 5555 5555
0000:1030  5555 5555 5555 5555 5555 5555 5555 5555
0000:1040  5555 5555 5555 5555 5555 5555 5555 5555
0000:1050  5555 5555 5555 5555 5555 5555 5555 5555
0000:1060  5555 5555 5555 5555 5555 5555 5555 5555
0000:1070  5555 5555 5555 5555 5555 5555 5555 5555
0000:1080  5502 5555 5555 5555 5555 5555 5555 5555
0000:1090  5555 5555 5555 5555 5555 5555 5555 5555
```

## G (GO) Command

**Function :**

The Go (**G**) command is used to transfer control of the 8086 from the serial monitor program to a user's program in memory.

**Syntax :**

```
G[[[seg:]strt],[[seg:]brk]]<CR>
```

**Operation :**

To use the Go command, enter **G** when prompted for command entry, enter "start address". To begin program execution, enter a carriage return.

To exit from the executing program, press either the [RES] or the [INT] key on the Hex keypad. If the [RES] key is pressed, control is transferred to the monitor program and the appropriate 8086 registers are initialized. If the [INT] key is pressed, the program is interrupted, the **SERIAL** mode is re-entered, all of the 8086 registers are saved, and the following message is displayed.

```
    Break at aaaa:bbbb
```

In the above message, "aaaa" is the current CS register value, and "bbbb" is the current value of the IP register of the next program instruction to be executed. If a subsequent **G** command is entered, the current IP register contents "bbbb" and the data byte addressed by the CS and IP registers are displayed. With a carriage return control is transferred back to the program, and execution resumes from there.

The **G** command optionally permits a " breakpoint address" to be entered. A breakpoint address has the same affect as pressing the [INT] key while a program is being executed. Note that while specifying breakpoint address, the default segment value is either the "start address" segment value (if specified) or the current CS register contents. In addition the location specified by the breakpoint address must contain the first (opcode or prefix) byte of the instruction. When breakpoint address is specified, the monitor replaces the instruction at the addressed location with an interrupt instruction and saves the "break-pointed" instruction. When the program reaches the breakpoint address, control is returned to the monitor, "break-pointed" instruction is replaced in the program, all registers are saved, and the monitor displays the following message followed by a command prompt. If required the user can examine the register contents using the **R** command.

```
    Break at aaaa:bbbb
```

In the above message, "aaaa" is the current CS register value, and "bbbb" is the current IP register value. (The combined register value is the address of the "break-pointed" instruction.) If a subsequent **G** command is entered, execution resumes from the current instruction address.

**NOTE :** *If used, the break point address must be specified each time the program is executed.*

**Error Conditions :**

Attempting to breakpoint a program in read-only memory (ROM).

**EXAMPLES :**

Example 1 : Transfer control to the program at 0:1000H.

```
Dyna-86>G 0:1000 <CR>
```

Example 2 : Transfer control to the program at 0:1000H and break at the instruction in location 0:1037H.

```
Dyna-86>G 0:1000, 0:1037 <CR>
```

## IN (Port Input) Command

**Function :**

The Port Input (**IN**) command is used to display a byte or word at an input port.

**Syntax :**

```
IN[W]addr <CR>
```

**Operation :**

The Port Input command inputs a byte (**IN**) or word (**INW**) from the port specified by "port address" and displays the value on the terminal. 8086 has 64K of I/O byte addresses, hence no segment values are permitted with the port address. After port address enter carriage return.

When using the **INW** command, the low-order address should be specified as the port address.

**EXAMPLES :**

Example 1 : Input Single byte from Port address 10H.

```
Dyna-86>IN 10 <CR>
0D
```

Example 1 : Input Single word from Port address 10H.

```
Dyna-86>INW 10<CR>
35 0D
```

## C (COPY) Command

**Function :**

The Copy (**C**) command is used to copy a block of data within memory.

**Syntax :**

```
C[seg:]strt,end,[seg:]dest <CR>
```

**Operation :**

When using the **C** command, the contents of the memory block from the "start address" to "end address" are moved to consecutive memory locations beginning at "destination address". As with the **D** command, end address is relative to the segment address value specified with the start address (if no segment value is specified, the last segment value is used). Consequently, no segment value is permitted with end address, and block moves are limited to 64K bytes.

Since a copy is performed one byte at a time, the **C** command can be used to fill a block of memory with a constant value. This is accomplished by specifying a destination address which is one greater than the start address. The block of memory locations from start address to end address + 1 are filled with the value contained in start address. (The **E** (Edit) command can be used to define the constant at "start address").

**Error Conditions :**

1) Attempting to copy data to a read-only (ROM) or non-existent memory location.

2) Specifying an end address value which is less than the offset value of start address.

**EXAMPLES :**

Example 1 : Copy the contents of locations 0:00H through 0:FFH to the memory block beginning at 0:1000H.

```
Dyna-86>C 0:0 FF 0:1000 <CR>
```

## T (Trace) Command

**Function :**

The Single Step (**T**) command is used to execute a single instruction of the user-program. After each instruction is executed, control is returned to the monitor to study the effect of the instruction executed.

**Syntax :**

```
T[[seg:]strt] <CR>
```

**Operation :**

To use the Single Step command, enter **T** when prompted for command entry. If " start address" includes a segment value, both the CS and IP registers are modified. When <CR> is entered, the instruction addressed is executed and control is returned to the monitor. The monitor saves all of the register contents and outputs the address (IP register contents) and displays all registers and the next instruction to be executed in unassemble form.

**Restrictions :**

➤ If an interrupt occurs prior to the completion of a single-stepped instruction or if a single-stepped instruction generates an interrupt, when the monitor is re-entered, the CS & IP registers will contain the address of the interrupt service routine. Therefore, a type 3 (breakpoint) interrupt instruction (0CCH or OCDH) should not be single-stepped since its execution would step into the monitor.

➤ An instruction that is part of a sequence of instructions that switches between stack segments (i.e., changes the SS and SP register contents) cannot be single stepped.

➤ A MOV or a POP instruction that modifies a segment register cannot be single- stepped. Control is returned to the monitor after the next instruction (instruction that immediately follows the MOV or POP instruction) is executed.

**EXAMPLES :**

Example 1 : Single step a series of instructions beginning at 0:1000H. Single step two instructions and then repeat the single step on the second instruction again.

```
Dyna-86>T 0:1000 <CR>
0000:1003  MOV BX, 5678
AX= 1234   BX= 01F0   CX= 0114   DX= 0114   SP= 05F2   BP= 0120   SI= 0114
DI= 0114   CS= 0000   DS= 00F0   SS= 0000   ES= 0100   IP= 1003   FL= F102

Dyna-86>T <CR>
0000:1006  MOV CX, 9ABC
AX= 1234   BX= 5678   CX= 0114   DX= 0114   SP= 05F2   BP= 0120   SI= 0114
DI= 0114   CS= 0000   DS= 00F0   SS= 0000   ES= 0100   IP= 1006   FL= F102

Dyna-86>T <CR>
0000:1009  JMP 1009
AX= 1234   BX= 5678   CX= 9ABC   DX= 0114   SP= 05F2   BP= 0120   SI= 0114
DI= 0114   CS= 0000   DS= 00F0   SS= 0000   ES= 0100   IP= 1009   FL= F102
```

## O (Port Output) Command

**Function :**

The Port Output (O) command is used to output a byte or word to an output port.

**Syntax :**

```
O[W]addr,[byte|word] <CR>
```

**Operation :**

The Port Output command outputs the byte (`O`) or word (`OW`) entered as "data" to the output port specified by "port address". Like the Port Input command, I/O addressing is for the 64K I/O addresses supported by 8086. A carriage return following a data entry outputs the data and terminates the command.

To use the additional 8255A (# 2) onboard, it is necessary to first program it to either input or output mode. This can be accomplished by using the `O` command to output a control byte to the 8255 (# 2) control port. The table given below defines the control port address and the associated data byte to be outputted to the control port for input or output modes.

| **8255 (# 1)** | **Control Byte** | |
|---|---|---|
| **Control Port** | **Input Mode** | **Output Mode** |
| 67H | 9BH | 80H |

**EXAMPLES :**

Example 1 : Program parallel I/O port 8255 (# 2) for output and output 55H on Port A.

```
Dyna-86>O 67 80 <CR>
Dyna-86>O 61 55 <CR>
```

Example 2 : Program parallel I/O port 8255 (#1) and 8255 (#2) for output.

```
Dyna-86>O 66 80 80 <CR>
```

Example 3 : Output AAH on Port A of 8255 (#1) and 55H on Port A of 8255 (#2)..

```
Dyna-86>O 60 55 AA <CR>
```

## L (Load HEX File) Command

**Function :**

The Read Hex File (**L**) command allows the monitor to read a 8086 Hex file from a IBM PC (Downloading process) and load it into DYNA-86 memory.

**Syntax :**

```
L[seg:] <CR>
```

**Operation :**

To use the Load Hex File command, enter **L**, with segment value if required when prompted for command entry. When the terminal is ready to transmit, enter a carriage return. The data values from each '*data record* ' is placed in memory from the offset specified in the '*load address*' of each '*data record* '.

Refer to appendix on "**INTEL HEX FORMAT**" for the details of this standard and the terms used in the above paragraph.

> **NOTE :** *All relevant information of the Downloading procedure is covered in the section "***UPLOADING / DOWNLOADING OF DATA***" of this Chapter.*

**EXAMPLES :**

Example 1 : Load a file and load the data into memory addresses specified in the file and segment specified in command i.e. 500H.

```
Dyna-86>L 500 <CR>
Ready ...
```

Example 2 : Load a file and load the data in to memory address specified in Intel Hex file. Use the segment last used by any command.

```
Dyna-86>L <CR>
Ready ...
```

## E (Edit Memory) Command

**Function :**

The Edit Memory (**E**) command is used to examine the byte (**E**) or word (**EW**) contents of selected memory locations. The contents of memory can be displayed or updated with a new data value entered from the terminal.

**Syntax :**

```
E[W][[seg:]strt] <CR>
```

**Operation :**

To use the Edit Memory command, enter **E** or **EW** and the "address" of the memory location to be examined. Note that if the segment address value is not specified, the last contents of the segment register are used by default. After the address is entered, enter a <CR>. The monitor will then display the current data contents of the addressed memory location followed by a dash and a space to indicate that the addressed location is open for update. Note that while using the **EW** command, the byte contents of the memory address + 1 (higher byte) are displayed first and open for update, followed by display of the lower byte for update. If only one memory location is to be examined, press <ESC> to terminate the command.

If a series of continuous memory locations are to be examined and / or updated, enter a <CR> to advance to the next consecutive memory location (**E** command) or next two consecutive memory locations (**EW** command). If the data contents are not to be updated, enter a <CR> to examine the next memory location, or enter new data followed by a <CR> to update the current location and display the next location. Press <ESC> to terminate the command.

**Error Conditions :**

Attempting to modify a non-existent or read-only ROM memory location gives error.

**EXAMPLES :**

Example 1 : Examine RAM location from 0:1000 byte data.

```
Dyna-86>E 0:1000 <CR>
0000:1000  34-0 <CR>
0000:1001  12-1 <CR>
0000:1002  34- <CR>
0000:1003  12-3 <CR>
0000:1004  34-4 <CR>
0000:1005  12-5 <CR>
0000:1006  34- <ESC>
Dyna-86>
```

Example 2 : Examine RAM location from 0:1000 word data.

```
Dyna-86>EW 0:1000 <CR>
0000:1000  1234-3456 <CR>
0000:1002  1234-0012 <CR>
0000:1004  1234-0 <CR>
0000:1006  1234-55AA <CR>
0000:1008  1234- <ESC>
Dyna-86>
```

## W (Write HEX File) Command

### Function :

The Write Hex File (**W**) command allows a block of memory from Dyna-86 to be transfered to a terminal (Uploading process) in 8086 Hex file format.

### Syntax :

```
W[seg:],strt,end <CR>
```

### Operation :

To use the Write Hex File command, enter **W** , "start address" and "end address" of the memory block to be transfered. If no segment address value is specified with start address, the last segment value is used. No segment address value is permitted with end address. When the carriage return is entered, the following information is sent to the terminal.

- 60 null characters (leader).
- Multiple '*data records* '.
- An '*end of file record* '.

The data in the memory bounded by the start address and end address (inclusive) are sent as multiple '*data records* '. It sends a block (maximum of 16 bytes) of data per '*data record* '. The offset value of the start of each block in memory is sent as the '*address field* ' for the '*data record* '.

The last record sent is the '*end of file record* '.

Refer to appendix "**INTEL HEX FORMAT**" for this standard and details of the terms used in here.

**NOTE :** *All relevant information of the Uploading procedure is covered in the section "***UPLOADING/ DOWNLOADING OF DATA***".*

**Error Conditions :**

Specifying a value for end address that is less than the offset value of start address.

**EXAMPLES :**

Example 1 : Output the memory block from F000:0H to F000:FFH.

```
Dyna-86>W F000:0 FF <CR>
:1000 0000  B800  008E  D88E  D0BC  0020  8CC8  8EC0  A304   4D
:1000 1000  00A3  0E00  B8DF  06A3  0400  B81F  07A3  0C10  5E
:1000 2000  B000  E652  B0C0  E652  C606  5404  00BE  5712  F5
:1000 3000  B700  B300  0EE8  8800  90BE  5B12  B700  B301  B2
:1000 4000  0EE8  7C00  90EB  1C90  33C0  8ED8  8CC8  8EC0  1C
:1000 5000  B700  B301  0EE8  FB00  90BE  8712  B300  0EE8  B4
:1000 6000  5E00  9033  C08E  D88C  C88E  C00E  E800  0190   20
:1000 7000  3C17  7503  E96F  073C  0075  03E9  1D02  3C01  5D
:1000 8000  7503  E969  023C  1375  03E9  B902  3C14  7503   71
:1000 9000  E93C  053C  1275  03E9  1B06  3C15  7503  E9B1   03
:1000 A000  053C  0475  03E9  B304  3C05  7503  E9CE  043C  43
:1000 B000  0675  03E9  E904  3C07  7503  E9FA  04E9  5301   0D
:1000 C000  5051  5657  5583  C603  B090  80FB  0075  02B0   5F
:1000 D000  94E6  52B1  04BD  3712  33C0  268A  048B  F82E   41
:1000 E000  8A03  80F9  0475  0780  FF01  7502  0480  E650    D9
:0F00 F000  4EFE  C975  E35D  5F5E  5958  CB56  B300  0EE7
:0000 0001  FF
```

## R (Examine / Modify Register) Command

### Function :

The Examine / modify Register (**R**) command is used to examine and if desired, to modify any of the 8086's individual registers.

### Syntax :

```
R[reg] <CR>
```

### Operation :

To use the Examine / modify Register command, enter **R** when prompted for command entry. If you wish to examine the current contents of all the registers, enter a carriage return (contents of all the registers will be displayed.) If you wish to examine and optionally modify the contents of an individual register, enter the register's abbreviation according to the table given below :

| **Register** | **Abbreviation** |
|---|---|
| Accumulator | AX |
| Base | BX |
| Count | CX |
| Data | DX |
| Stack Pointer | SP |
| Base Pointer | BP |
| Source Index | SI |
| Destination Index | DI |
| Code Segment | CS |
| Data Segment | DS |
| Stack Segment | SS |
| Extra Segment | ES |
| Instruction Pointer | IP |
| Flag | FL |

When a register abbreviation is entered, the monitor displays register name, an equal sign ("="), the current register contents and the data prompt character ("-"). In order to change the register contents, enter new values followed by a carriage return. When a carriage return is entered, the register is updated (if new contents were entered) and the monitor returns to the command mode.

**EXAMPLES :**

Example 1 : Examine the 8086 registers.

```
Dyna-86>R <CR>
AX= 1401  BX= 1401  CX= 1401  DX= 1401  SP= 1401  BP= 1401  SI= 1401
DI = 1401  CS= 1401  DS= 1401  SS= 1401  ES= 1401  IP= 1401   FL= 0001
```

Example 2 : Examine the AX register.

```
Dyna-86>R AX <CR>
AX=1401-1234
```

```
Dyna-86>R <CR>
AX= 1234   BX= 1401  CX= 1401  DX= 1401  SP= 1401  BP= 1401  SI= 1401
DI= 1401    CS= 1401  DS= 1401  SS= 1401  ES= 1401  IP= 1401  FL= 0001
```

## F (Fill Byte / Word) Command

**Function :**

This command is used to fill an area of memory with a data byte or a word constant.

**Syntax :**

```
F[W][seg:]strt,end,byte|word <CR>
```

**EXAMPLES :**

Example 1 : Fill memory block 0:1000H to 0:10FFH with 1234. Display the contents of the same memory block.

```
Dyna-86>FW 1000 10FF 1234 <CR>


Dyna-86>DW 1000 <CR>
0000:1000  1234 1234 1234 1234 1234 1234 1234 1234
0000:1010  1234 1234 1234 1234 1234 1234 1234 1234
0000:1020  1234 1234 1234 1234 1234 1234 1234 1234
0000:1030  1234 1234 1234 1234 1234 1234 1234 1234
0000:1040  1234 1234 1234 1234 1234 1234 1234 1234
0000:1050  1234 1234 1234 1234 1234 1234 1234 1234
0000:1060  1234 1234 1234 1234 1234 1234 1234 1234
0000:1070  1234 1234 1234 1234 1234 1234 1234 1234
0000:1080  1234 1234 1234 1234 1234 1234 1234 1234
0000:1090  1234 1234 1234 1234 1234 1234 1234 1234
```

Example 2 : Fill memory block 0:1000 to 0:11FF with 12.

```
Dyna-86>F 0:1000 11FF 12
```

Example 3 : Fill memory block 0:1000 to 0:11FF with 1234.

```
Dyna-86>F 0:1000 11FF 1234
```

## I (Insert Byte / Word) Command

**Function :**

This command is used to insert a byte of data or a word constant in a given block of memory. The remaining block shifts down by one position, upto the end address specified.

**Syntax :**

```
I[W][seg:]strt,end,byte|word <CR>
```

**EXAMPLES :**

Example 1 : Display the memory contents of the memory block 0:1000 to 0:1010.
Insert 3H at 0:100E.
Again display the memory contents of the memory block
0:1000 to 0:1010.

```
Dyna-86>D 0:1000 1010
0000:1000  00 01 02 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 55
0000:1010  AA

Dyna-86>I 0:1003 100E 3

Dyna-86>D 0:1000 1010
0000:1000  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000:1010  AA
```

## DL (Delete Byte / Word) Command

**Function :**

This command is used to delete a byte of data or a word constant in a given block of memory. The remaining block shifts up by one position, upto the end address specified.

**Syntax :**

```
DL[W][seg:]strt,end <CR>
```

**EXAMPLES :**

Example 1 : Display the memory contents of the memory block 0:1000 to 0:1010.
Delete byte 3H at 0:100F.
Again display the memory contents of the memory block 0:1000 to 0:1010.

```
Dyna-86>D 0:1000 1010
0000:1000  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000:1010  AA


Dyna-86>DL 0:1003 100F


Dyna-86>D 0:1000 1010
0000:1000  00 01 02 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0F
0000:1010  AA
```

## S (Search Byte / Word) Command

**Function :**

The Search command searches a block of memory for a given pattern and lists the addresses where the pattern occurs.

**Syntax :**

```
S[W][seg:]strt,end,byte|word <CR>
```

**EXAMPLES :**

Example 1 : Search 0:0000H to 0:1FFFH for 55AA.

```
Dyna-86>SW 0:0 1FFF 55AA <CR>
1100-55AA
1E00-55AA
```

## CM (Compare) Command

**Function :**

This command is used to compare block of memory byte by byte.

**Syntax :**

```
CM[seg:]strt,end,[seg:]strt <CR>
```

**EXAMPLES :**

Example 1 : Compare 0:1000 to 0:17FF block of data with 0:2000 to 0:27FF.

```
Dyna-86>CM 0:1000 17FF 2000 <CR>
1100  03 55
111F  07 55
114B  02 55
116E  08 55
1199  04 55
11CA  0F 55
```

## HD (Hex to Decimal Conversion) Command

**Function :**

This command is used to convert a Hexadecimal number to its decimal equivalent.

**Syntax :**

```
HDnum <CR>
```

**EXAMPLES :**

Example 1 : Convert FEDH to its Decimal equivalent.

```
Dyna-86>HD FED <CR>
4077
```

Example 2 : Convert FFFFH to its Decimal equivalent.

```
Dyna-86>HD FFFF <CR>
065535
```

## DH (Decimal to Hex Conversion) Command

**Function :**

This command is used to convert a Decimal number to its Hexadecimal equivalent.

**Syntax :**

```
DHnum <CR>
```

**EXAMPLES :**

Example 1 : Convert Decimal number 4678 to its Hexadecimal equivalent.

```
Dyna-86>DH 4678
1246
```

Example 2 : Convert Decimal number 45E6 to its Hexadecimal equivalent.

```
Dyna-86>DH 45E6
```

Since 45E6 is not a valid decimal number, the user is prompted with an error message

```
Dyna-86>DH 45E6
             ^
Error ! Invalid Decimal number
```

## A (Assemble) Command

**Function :**

The A command accepts assembly language statements and assembles each statement into executable machine code. These assembled machine codes are stored in a specified memory on line by line basis at the time of entry. For details refer to Chapter 6.

**Syntax :**

```
A[[seg:]off] <CR>
```

**EXAMPLES :**

```
Dyna-86>A 0:1000 <CR>
0000:1000  MOV AX,1234
0000:1003  MOV BX,5678
0000:1006  MOV CX,9ABC
0000:1009  JMP 1009
0000:100B  ;PRESS ESC KEY
```

## U (Unassemble) Command

**Function :**

The U command followed by the address disassembles (unassemble) the program from the specified location. The U command translates machine instruction into assembly language mnemonics. For details refer to Chapter 6.

**Syntax :**

```
U[[seg:]off[,[seg:]off]] <CR>
```

**EXAMPLES :**

```
Dyna-86>U 0:1000 1008 <CR>
0000:1000  MOV AX, 1234
0000:1003  MOV BX, 5678
0000:1006  MOV CX, 9ABC
```

## P (Printer ON/OFF) Command

**Function :**

P command is used to enable or disable the printer (Hardcopy) output. On power ON or Reset, the printer is disabled. By pressing P first time in serial mode gives the message "Printer ON" on the terminal. After this everything displayed on the terminal will be dumped to the printer and you will get a Hardcopy. Printer can be connected to J1 connector using the printer cable supplied (optional).

If any error occurs in the printer like paper out, then printer output will be automatically disabled or the user can disable the printer output by pressing P followed by <CR>.

The P command toggles the printer ON/OFF flag.

Example 1 : To enable the printer.

```
Dyna-86>P <CR>
Printer ON.
```

Example 2 : To disable the printer.

```
Dyna-86>P <CR>
Printer OFF.
```

## UPLOADING / DOWNLOADING OF DATA

The two serial commands **W** (Write Hex File) and **L** (Load Hex File) are used for UPLOADING and DOWNLOADING the data from the host system like IBM-PC.

The programs can be developed in these systems (assemble a program and generate a Intel Hex File) and can be Downloaded into the DYNA-86 memory. This Downloaded program can be executed, single stepped or debugged.

Any program developed or debugged on DYNA-86 can be stored as a file on host systems like IBM-PC by uploading the memory image of the data and program in the Extended Intel Hex format. This file can be retrieved back into the DYNA-86 memory by the downloading process.

**NOTE :** *Details on Intel Hex format is given in the appendix.*

### Downloading from IBM PC to DYNA-86 System

1) Connect the IBM PC to DYNA-86 with a serial cable as described in the Section USING THE IBM PC AS A TERMINAL and run the Tango utility.

2) The message `DYNA-86>` should be displayed on the terminal. DYNA-86 is now ready to accept commands from the terminal.

3) Type `L` (segment value if required) to receive data from terminal. DYNA-86 is now ready to accept data in the INTEL HEX format.

4) Press F9, the TANGO menu will appear on the screen. The M command is used to transfer the desired hex file.

| | |
|---|---|
| Press M key. | It responds with the message |
| "transmit file (y/n) :" | Press Y key |
| "Enter file name :" | Type File name and press < CR> |

The message " *transmitting* " will appear on left top corner of the screen indicating the IBM PC is ready to transmit.

Press F9 key to start transmitting the file.
After receiving the file, the prompt for a new command entry is displayed.

5) Data is received & stored in the memory locations defined in the INTEL HEX data records. The program can now be executed, single stepped or debugged.

**Uploading from DYNA-86 to the IBM PC**

1) Connect the IBM PC to DYNA-86 with a serial cable as described in the Section USING THE IBM PC AS A TERMINAL and run the Tango utility.

2) Press F9, the TANGO menu will appear on the screen. The "L" command is used to receive data and store it in a file.

| | |
|---|---|
| Press L key | It responds with the message |
| "Capture File (y/n):" | Press Y key |
| "Enter file name :" | Type the file name and press < CR> |

The message " *capturing* " will appear on left top corner of the screen indicating the IBM PC is ready to recieve data.

3) Press the F9 key to go back to the DYNA-86 **SERIAL** mode. Now execute the **W** command to send out the specified data. After all the data is sent out the prompt " `DYNA-86>` " is displayed.

4)

| | |
|---|---|
| Press F9 and type L | It responds with the following message |
| "End capture (y/n) :" | Press Y key |
| "Verify termination (y/n) :" | Press Y key |

The message "*Capturing* " should disappear. The IBM PC saves the received data under the given file name.

5) Press the F9 key to go back to the **SERIAL** mode.

## SERIAL MODE COMMAND SUMMARY

| **Command** | **Function/ Syntax** |
|---|---|
| **D** Display Memory | Displays block of memory data<br>`D[W][[seg:]strt[,end]]` |
| **G** Go | Transfers control of 8086 program to user’s program<br>`G[[[seg:]strt],[[seg:]brk]]` |
| **IN** Port Input | Displays byte/word at input port<br>`IN[W]addr` |
| **C** Copy Memory | Copies a block of data within memory<br>`C[seg:]strt,end,[seg:]dest` |
| **T** Trace | Executes a single instruction of the user program<br>`T[[seg:]strt]` |
| **O** Port Output | Displays byte/word at output port<br>`O[W]addr,[byte│word]` |

**L** Load Hex file Reads a 8086 Hex file from IBM PC

```
L[seg:]
```

**E** Edit Memory Examine byte/word contents of block of memory

```
E[W][[seg:]strt]
```

**W** Write Hex file Writes a 8086 Hex file to IBM PC

```
W[seg:],strt,end
```

**R** Examine/modify register Examines/modifies individual registers

```
R[(reg)][[new_contents)],]
```

**F** Fill Memory Fills a block of memory with required data

```
F[W][seg:]strt,end,byte|word
```

**I** Insert Byte / Word Inserts byte/word in a block of memory

```
I[W][seg:]strt,end,byte|word
```

**DL** Delete Byte / Word Delets byte/word from a block of memory

```
DL[W][seg:]strt,end
```

**S** Search Byte / Word Searches block of memory for required byte/word

```
S[W][seg:]strt,end,byte|word
```

**CM** Compare blocks Compares two blocks of memory byte by byte.

```
CM[seg:]strt,end,[seg:]strt
```

**HD** Hex to Dec Converts Hexadecimal number to decimal

```
HDnum
```

**DH** Dec to Hex Converts decimal number to Hexadecimal

```
DHnum
```

**A** Assemble Assembles the program from the specified location

```
A[[seg:]off]
```

**U** Unassemble Disassembles the program from the specified location.

```
U[[seg:]off[,[seg:]off]]
```

**P** Printer ON/OFF Enables/disables Printer

```
P
```

# CHAPTER 6
# ASSEMBLER AND DISASSEMBLER

## INTRODUCTION

DYNA-86 firmware contains Assembler/ Disassembler function through serial mode. The assembler / disassembler is an interactive assembler/ editor in which the source program is not saved. Each source line is translated into the proper 8086 Machine Language code and is stored in memory on a line-by-line basis at the time of entry. The DYNA-86 assembler has some limitations than full fledge assembler tool for creating modifying and debugging 8086 code.

Disassembler lists the contents of memory in Assembly Language Mnenonics.

There are two commands in serial mode to invoke Assembler & Disassembler. They are :

1. A, for Assembler
2. U, for Disassembler.

## Assembler : A command

The A command accepts assembly language statements & assembles each statement into executable machine code. These assembled machine codes are stored in a specified memory on line by line basis at the time of entry.

The address parameters specifies the location where assembly language mnemonics begin. If address is omitted, the assembler uses the address following the last instruction generated the last time A command was used.

When the Enter (<CR>) or <ESC> key is pressed, the assembly language is translated and the resulting machine code is stored in memory. Pressing Enter key alone at the address prompt terminates the A command.

- All numbers are assumed to be Hexadecimal integers and should not be entered with a trailing H character.

- Segment overrides must be specified by preceding the memory reference operand with CS:, DS:, ES: or SS:

  ```
  e.g. MOV AL, CS: [BX + SI]
  ```

- Specific Hexadecimal values, rather than program labels must be included.

- When data type (word or byte) is not implicit in the instruction, the type must be specified by preceeding the operand with BYTE or WORD.

  ```
  e.g. MOV BYTE [BX], 12.
  ```

- The size of the string must be specified by adding B(byte) or W(Word) to the string instruction mnemonic (e.g. LODSB or LODSW).

- Memory locations are differentiated from immediate operands by enclosing memory address in square brackets.

- Repeat prefixes as REP, REPZ, or REPNZ can be entered.

**EXAMPLE :**

To begin assembly code at address 0:1000H type

```
.A0:1000 <CR>
```

To assemble the instruction sequence.

```
0000:1000 LODS WORD [SI] <CR>
0000:1001 XCHG BX,AX <CR>
0000:1003 JMP [BX] <CR>
0000:1005 <CR>
```

The machine code of the given example are stored in RAM starting at 0:1000H location.

To continue assembling at the location following the last instruction generated by a previous A command type

```
.A <CR>
```

### Disassembler : U command

The U command followed by the address disassembles (unassemble) the program from the specified location. The U command translates machine instruction into assembly language menmonics.

The range parameters specifies the starting and ending address of the machine instructions to be disassembled. If range does not specify a segment, the disassembler uses a present CS.

If range is omitted, 15 instructions are disassembled, starting at the address following the last instruction disassembled by the previous U-command. Thus, the successive memory locations can be disassembled by entering just the U-command without parameters.

**EXAMPLE :**

To disassemble the 4 bytes of machine instruction starting at 0:1000 (the program which was entered using A command type

```
DYNA-86>U0:1000, 1003 <CR>
```

The program at 0:1000 is disassembled and appears in the following format :

```
0000:1000 LODSW WORD [S1]
0000:1001 XCHG BX,AX
0000:1003 JMP [BX]
```

To disassemble 15 instructions from 2000:100, type

```
DYNA-86>U2000:100 <CR>
```

# ERROR MESSAGES

During assembly, you may encounter any of the following error messages. The following lines describes the Error Messages.

**1. Invalid Instruction Mnemonic**

The assembler could not recognise the opcode.

**2. Extra characters on the line**

Sufficient information to definea statement has been received on a line, but some additional characters were also provided.

**3. Jump out of range**

A conditional jump was not within the required reange. For short jump, the range is 128 bytes backward or 127 bytes forward.
For near jump it is from -32768 to 32757.

**4. Invalid number specified as operand.**

A constant number contained invalid characters.

**5. Operand too big or Delimiter not found.**

The constant number contained either too many digits or the assembler could not find the delimiter of the number.

**6. Operand(s) missing.**

The instruction requires more operands than were provided.

**7. Operand must be an Accumulator Register.**

The instruction requires accumulator register, but some other register was provided.

**8. Invalid Operand or Addressing Mode.**

The operands specified were illegal.

**9. REP prefix can be used only with string instructions.**

Some instruction other than string instruction was specified with REP prefix.

**10. Operand must be a 16 bit register.**

An 8 bit register was specified when a 16 bit register was required in the instruction.

**11. Operand must have size.**

An operand was expected to have a specified size but no size was supplied.

**12. Operand combination illegal.**

Two operands were used with the instruction but does not allow the specified combination of operands.

**13. Register operand can be used only in intrasegment JMP & CALL instructions.**

**14. String Mnemonics Missing.**

A REP prefix was not followed by a string instruction.

**15. Comma Missing.**

A comma was missing between two operands.

**16. Immediate data out of range.**

An immediate data was too large for its contents.

# CHAPTER 7
# SAMPLE PROGRAMS

## INTRODUCTION

The following pages contain a number of sample programs for the first time user's of 8086 microprocessor. The user should be familiar with the basics of assembly programming. Most of the programs are for understanding the instruction set of 8086 CPU and a few standard algorithms.

In all the program listings the program code starts from OFFSET 100H while the data starts from OFFSET 200H.
To run a program, say in segment 0100H follow the procedure given below.

1) Enter the program codes as seen in the listing from offset 100H (ie. address 0100:100H).

2) Enter the data if any from offset 200H (ie. address 0100:200H). Note that the listings show the WORD data in the normal reading format. It should be entered in the reverse order. That is if the word is 1234H then first enter the byte 34H followed by byte 12H.

3) Initialise the CS register to 0100H and the IP register to 100H.

4) Initialise the DS, ES and the SS register to the same value as CS (ie. 0100H).

5) Now execute the program or trace it. Any of the commands for the selected mode of operation can be used. All the programs end with the INT 3 instruction, which hands back control to the monitor after execution.

In the **SERIAL** mode of operation in DYNA-86, the user can type the source program directly in the form of mnemonics codes using Assembler (A command). However the LABELS and SYMBOLS are not supported. The actual values for these will have to be entered.

The following programs are included in the following pages

| | |
|---|---|
| Program 1 | Byte multiplication |
| Program 2 | Word multiplication |
| Program 3 | Packed BCD from ASCII |
| Program 4 | BCD multiplication |
| Program 5 | BCD division |
| Program 6 | BCD subtraction |
| Program 7 | Signed byte to word |
| Program 8 | Scan string for character |
| Program 9 | IF-THEN-ELSE implementation |
| Program 10 | BCD to HEX (Register parameter passing) |
| Program 11 | Factorial by recursion |
| Program 12 | 32 bit division |
| Program 13 | Case conversion of String |
| Program 14 | BCD string addition |
| Program 15 | ASCII number to Binary |
| Program 16 | Square root using 8087 instructions |

# PROGRAM 1 BYTE MULTIPLICATION

```

                       PAGE 55,132

                       ;                     Program 1

                       ; This program demonstrates signed multiplication.
                       ; The program multiplies two bytes 80h and 40h and leaves
                       ; the result in AX register.

0000                   CODESEG  SEGMENT      PARA     PUBLIC   ’CODE’
                                ASSUME       CS:CODESEG, DS:CODESEG
                                ASSUME       ES:CODESEG, SS:CODESEG

0100                            ORG    100H
0100                   BEGIN:

0100                   START    PROC   NEAR

= 0080                 BYTE1    EQU    80H
= 0040                 BYTE2    EQU    40H

0100  B0 80                     MOV    AL,BYTE1      ; load AL with byte 1
0102  B1 40                     MOV    CL,BYTE2
0104  F6 E9                     IMUL   CL            ; multiply byte1 and byte2
                                                     ; product in AX
0106  CC                        INT    3

0107                   START    ENDP

0107                   CODESEG  ENDS
                                END    BEGIN
```

# PROGRAM 2 WORD MULTIPLICATION

```

                       PAGE 55,132
                       ;
                       ;                      Program 2
                       ;
                       ;
                       ; This program multiplies the two 16-bit  words  in  the  memory
                       ; locations called MULTIPLICAND  and MULTIPLIER. The result  is a
                       ; 22-bit word and is stored in the memory location called PRODUCT.


0000                   CODESEG   SEGMENT     PARA     PUBLIC   ’CODE’
                                 ASSUME      CS:CODESEG, DS:CODESEG
                                 ASSUME      ES:CODESEG, SS:CODESEG

0100                             ORG    100H
0100                   BEGIN:
0100                   START     PROC   NEAR

0100  A1 0200 R                  MOV    AX,[MULTIPLICAND]      ;get one word
0103  8B 0E 0202 R               MOV    CX,[MULTIPLIER]        ;get the second word
0107  F7 E1                      MUL    CX                     ;multiply them
0109  A3 0204 R                  MOV    [PRODUCT],AX           ;store low word of result
010C  89 16 0206 R               MOV    [PRODUCT+2],DX         ;store high word of result
0110  CC                         INT    3                      ;exit

0111                   START     ENDP

0200                             ORG    200H

0200  204A             MULTIPLICAND     DW   204AH
0202  3B2A             MULTIPLIER       DW   3B2AH
0204    02 [           PRODUCT          DW   2 dup (0)
             0000
                  ]


0208                   CODESEG   ENDS
                                 END    BEGIN
```

# PROGRAM 3 PACKED BCD FROM ASCII

```

                       PAGE 55,132
                       ;                     Program 6
                       ;
                       ; This  program produces a packed BCD byte from two ASCII encoded
                       ; digits. The first ASCII digit (5) is located in AL register and
                       ; the  second ASCII (9) is located in the BL register. The  result
                       ; (packed BCD) is stored in the AL register.


0000                   CODESEG   SEGMENT     PARA     PUBLIC   ’CODE’
                                 ASSUME      CS:CODESEG, DS:CODESEG
                                 ASSUME      ES:CODESEG, SS:CODESEG

0100                             ORG    100H
0100                   BEGIN:
0100                   START     PROC   NEAR

0100  B0 35                      MOV    AL,35H        ; load first ASCII digit into AL
0102  B3 39                      MOV    BL,39H        ; load second ASCII digit into BL

0104  24 0F                      AND    AL,0FH        ; mask upper  four  bits  of
                                                      ; first digit
0106  80 E3 0F                   AND    BL,0FH        ; mask upper  four  bits  of
                                                      ; second digit
0109  B1 04                      MOV    CL,04H        ; load  CX  for  4  rotates
                                                      ; required
010B  D2 C0                      ROL    AL,CL         ; rotate AL 4 bit positions
010D  02 C3                      ADD    AL,BL         ; combine nibbles, result in AL

010F  CC                         INT    3             ; Exit

0110                   START     ENDP

0110                   CODESEG   ENDS
                                 END    BEGIN
```

# PROGRAM 4 BCD MULTIPLICATION

```

                       PAGE 55,132
                       ;
                       ;                     Program 9
                       ;
                       ; This  program  describes the AAM Instruction (BCD  Adjust  after
                       ; Multiply). The unpacked BCD values in AL and BH are multiplied
                       ; togeather and then the result in AX is converted back into
                       ; unpacked BCD form again.


0000                   CODESEG   SEGMENT     PARA     PUBLIC   ’CODE’
                                 ASSUME      CS:CODESEG, DS:CODESEG
                                 ASSUME      ES:CODESEG, SS:CODESEG

0100                             ORG    100H
0100                   BEGIN:
0100                   START     PROC   NEAR

0100  B0 05                      MOV    AL,5          ; AL = 00000101 = unpacked BCD 5
0102  B7 09                      MOV    BH,9          ; BH = 00001001 = unpacked BCD 9
0104  F6 E7                      MUL    BH            ; AL x BH , result in AX
                                                      ; AX = 00000000 00101101 = 002DH
0106  D4 0A                      AAM                  ; AX = 00000100 00000101
                                                      ; which is unpacked BCD for 45
0108  CC                         INT    3

0109                   START     ENDP

0109                   CODESEG   ENDS
                                 END    BEGIN
```

# PROGRAM 5 BCD DIVISION

```

                       PAGE 55,132
                       ;                     Program 10
                       ;
                       ; This program describes the AAD Instruction
                       ; (Ascii Adjust before Division) ie. BCD to Binary Convertion
                       ; before Division.


0000                   CODESEG   SEGMENT     PARA     PUBLIC   ’CODE’
                                 ASSUME      CS:CODESEG, DS:CODESEG
                                 ASSUME      ES:CODESEG, SS:CODESEG

0100                             ORG    100H
0100                   BEGIN:
0100                   START     PROC   NEAR

0100  B8 0607                    MOV    AX,607H       ; AX=D607 unpacked BCD for 67 decimal
0103  B5 09                      MOV    CH,09H        ; CH = 09H
0105  D5 0A                      AAD                  ; adjust to binary before division
                                                      ; AX = 0043 = 43H = 67 decimal
0107  F6 F5                      DIV    CH            ; Divide AX by unpacked BCD in CH
                                                      ; AL = quotient = 07 unpacked BCD
                                                      ; AH = remainder = 04 unpacked BCD
                                                      ; PF = 0, SF = 0, ZF = 0
0109  CC                         INT    3

010A                   START     ENDP

010A                   CODESEG   ENDS
                                 END    BEGIN
```

# PROGRAM 6 BCD SUBTRACTION

```

                       PAGE 55,132
                       ;                     Program 11
                       ;
                       ; This  program  describes the AAS Instruction
                       ; (ASCII  Adjust  for Subtraction)


0000                   CODESEG  SEGMENT      PARA     PUBLIC   ’CODE’
                                ASSUME       CS:CODESEG, DS:CODESEG
                                ASSUME       ES:CODESEG, SS:CODESEG

0100                            ORG    100H
0100                   BEGIN:
0100                   START    PROC   NEAR

0100  B0 09                     MOV    AL,9H          ; AL = 0011 1001 = ASCII 9
0102  B3 05                     MOV    BL,5H          ; BL = 0011 0101 = ASCII 5
0104  2A C3                     SUB    AL,BL          ; (9-5) results :
                                                      ; AL = 0000 0100 = BCD 04
                                                      ; CF = 0
0106  3F                        AAS                   ; results :
                                                      ; AL = 0000 0100 = BCD 04
                                                      ; CF = 0 no borrow required
0107  CC                        INT    3

0108                   START    ENDP

0108                   CODESEG  ENDS
                                END    BEGIN
```

# PROGRAM 7 SIGNED BYTE TO WORD

```

                       PAGE 55,132
                       ;                      Program 12
                       ;
                       ; This  program describes the CBW Instruction
                       ; (Convert signed  Byte to signed Word)


0000                   CODESEG  SEGMENT      PARA     PUBLIC   ’CODE’
                                ASSUME       CS:CODESEG, DS:CODESEG
                                ASSUME       ES:CODESEG, SS:CODESEG

0100                            ORG    100H
0100                   BEGIN:
0100                   START    PROC   NEAR

0100  B8 009B                   MOV    AX,155        ;AX=00000000 10011011=-155 decimal
0103  98                        CBW                  ;convert signed byte in AL to signed
                                                     ;word in AX
                                                     ;result in AX  = 11111111 10011011
                                                     ;               = -155 decimal
0104  CC                        INT    3

0105                   START    ENDP

0105                   CODESEG  ENDS
                                END    BEGIN
```

# PROGRAM 8 SCAN STRING FOR CHARACTER

```

                       PAGE 55,132
                       ;                     Program 13
                       ;
                       ; This  program  describes the SCAS/SCAB/SCAW  Instruction
                       ; ( Scan string byte or a string word)
                       ; Scan a text string of 80 characters at offset 200h for a
                       ; carriage return (0Dh) character.


0000                   CODESEG   SEGMENT     PARA     PUBLIC   ’CODE’
                                 ASSUME      CS:CODESEG, DS:CODESEG
                                 ASSUME      ES:CODESEG, SS:CODESEG

0100                             ORG    100H
0100                   BEGIN:
0100                   START     PROC   NEAR

= 0200                 TXT_STR   EQU    200H

0100  B0 0D                      MOV    AL,0DH        ; byte to be scanned for in AL
0102  BF 0200                    MOV    DI,OFFSET TXT_STR      ; offset of string to DI
0105  B9 0080                    MOV    CX,80H        ; CX used as element counter
0108  FC                         CLD                  ; Clear DF so DI autoincrements
0109  F2/ AE                     REPNE  SCASB         ; compare byte in string with  byte
                                                      ; in AL in a loop.
                                                      ; If no match found CX will be 0,
                                                      ; else SI and DI will point to the
010B  CC                         INT    3             ; element after the first match.

010C                   START     ENDP

010C                   CODESEG   ENDS
                                 END    BEGIN
```

# PROGRAM 9 IF-THEN-ELSE IMPLEMENTATION

```
                       PAGE 55,132
                       ;                     Program 16
                       ;
                       ; IF_THEN_ELSE Implementation :
                       ;
                       ; This program reads the temperature of a solution and light one
                       ; of the  three  lamps  according  to the  temperature  read.
                       ;
                       ; ABSTRACT:      This program section reads the temperature  of
                       ;                a  solution and light one of the  three  lamps
                       ;                according  to  the temperature  read.  If  the
                       ;                temperature  is below 30 degrees C,  a  yellow
                       ;                lamp is turned on. If the temperature is  =30
                       ;                and   40 degrees, a green lamp is turned  on.
                       ;                Temp =40 degrees will turn on a red lamp.
                       ;
                       ; Registers Used  : CS, AL, DX
                       ; Ports Used      : 8255(#2) Port A (E0h) as a temperature input
                       ;                            Port B (E1h) as lamp control output
                       ;                   (yellow = bit 0, green = bit 1, red = bit 2)
                       ;


0000                   CODESEG   SEGMENT     PARA     PUBLIC   ’CODE’
                                 ASSUME      CS:CODESEG, DS:CODESEG
                                 ASSUME      ES:CODESEG, SS:CODESEG

0100                             ORG    100H
0100                   BEGIN:
0100                   START     PROC   NEAR

= 00E0                 PORT      EQU    0E0H          ; base address of 8255 (#2)
                                                      ; initialize 8255 (#2) port B as an
                                                      ; output port and port A as input.
0100  BA 00E3                    MOV    DX,PORT+3     ; point DX to port control register
0103  B0 99                      MOV    AL,99H        ; load control word to setup
0105  EE                         OUT    DX,AL         ; output port
0106  BA 00E2          MAIN_PROG:MOV    DX,PORT+2
0109  EC                         IN     AL,DX         ; read the ph sensor
010A  BA 00E1                    MOV    DX,PORT+1     ; point DX as output port
010D  3C 1E                      CMP    AL,1EH        ; compare temp with 30 deg.C
010F  72 09                      JB     YELLOW        ; if temp  30, light  yellow lamp
```

```
0111  3C 28                      CMP    AL,28H        ; compare with 40 deg.
0113  72 0A                      JB     GREEN         ; if temp  40,  light  green lamp
0115  B0 04            RED:      MOV    AL,04H        ; temp = 40 so load code  to
                                                      ; light red lamp
0117  EE                         OUT    DX,AL         ; send code to light red lamp
0118  EB EC                      JMP    MAIN_PROG

011A  B0 01            YELLOW:   MOV    AL,01H        ; load code to  light  yellow
011C  EE                         OUT    DX,AL         ; send code to light yellow lamp
011D  EB E7                      JMP    MAIN_PROG

011F  B0 02            GREEN:    MOV    AL,02H        ; load code to light green lamp
0121  EE                         OUT    DX,AL         ; send code to light green lamp
0122  EB E2                      JMP    MAIN_PROG


0124                   START     ENDP

0124                   CODESEG    ENDS
                                  END   BEGIN
```

# PROGRAM 10 BCD TO HEX (REGISTER PARAMETER PASSING)

```
                       PAGE 55,132
                       ;                     Program 18
                       ;
                       ; This  program demonstrates BCD to HEX conversion. It shows  how
                       ; to use the AL register to pass the parameters.
                       ;
                       ; ABSTRACT :     Program fragment that uses a procedure to convert
                       ;                BCD  numbers to HEX (binary).It shows how to  use
                       ;                the   AL  register  to  pass  parameters to   the
                       ;                procedure


0000                   CODESEG   SEGMENT      PARA     PUBLIC   ’CODE’
                                 ASSUME      CS:CODESEG, DS:CODESEG
                                 ASSUME      ES:CODESEG, SS:CODESEG

0100                             ORG    100H
0100                   BEGIN:
0100                   START     PROC   NEAR

0100  A0 0200 R                  MOV    AL,BCD_INPUT
0103  E8 010A R                  CALL   BCD_HEX
0106  A2 0201 R                  MOV    HEX_VALUE,AL  ;store the result
0109  CC                         INT    3

010A                   START     ENDP


                       ; PROCEDURE:     BCD_HEX
                       ;
                       ; Converts   BCD  numbers  to  HEX  (binary),   uses
                       ; registers to pass parameters to the procedure
                       ;
                       ; SAVES:  All registers used except AH

010A                   BCD_HEX   PROC   NEAR

010A  9C                         PUSHF                ; save flags
010B  53                         PUSH   BX            ; and registers
010C  51                         PUSH   CX

                                 ; start conversion

010D  8A E0                      MOV    AH,AL        ; save copy of BCD in AH
```

```
010F  80 E4 0F                  AND    AH,0FH        ; seperate and save lower
0112  8A DC                     MOV    BL,AH         ; BCD digit
0114  24 F0                     AND    AL,0F0H       ; seperate upper nibble
0116  B1 04                     MOV    CL,04H        ; move upper BCD digit to low
0118  D2 C8                     ROR    AL,CL         ; nibble position for multiply
011A  B7 0A                     MOV    BH,0AH        ; load conversion factor in BH
011C  F6 E7                     MUL    BH            ; upper BCD digit in AL *  0AH
                                                     ; in BH, result in AX
011E  02 C3                     ADD    AL,BL         ; add lower BCD to result  of
                                                     ; MUL, final result in AL

                                ; end conversion, restore registers

0120  59                        POP    CX
0121  5B                        POP    BX
0122  9D                        POPF
0123  C3                        RET

0124                  BCD_HEX   ENDP


0200                            ORG    200H

0200  ??              BCD_INPUT DB     ?
0201  ??              HEX_VALUE DB     ?

0202                  CODESEG   ENDS
                                END    BEGIN
```

## PROGRAM 11 FACTORIAL BY RECURSION

```

                       PAGE 55,132
                       ;                      Program 22
                       ;
                       ; This program computes the factorial of a number between 1 and 9.
                       ; Using recursion.
                       ;
                       ; ABSTRACT :     This program computes the factorial of a  number
                       ;                between 1 and 9


0000                   CODESEG   SEGMENT      PARA     PUBLIC   'CODE'
                                 ASSUME       CS:CODESEG, DS:CODESEG
                                 ASSUME       ES:CODESEG, SS:CODESEG

0100                             ORG    100H
0100                   BEGIN:
0100                   START     PROC   NEAR

= 0003                 NUMBER    EQU    03

0100  83 EC 04                   SUB    SP,0004H       ; make space in  stack  for
                                                       ; factorial to be returned
0103  B8 0003                    MOV    AX,NUMBER      ; put number to be passed  on
                                                       ; stack
0106  50                         PUSH   AX
0107  E8 010D R                  CALL   FACTO          ; compute factorial of number
010A  58                         POP    AX             ; get result
010B  90                         NOP                   ; simulate  next mainline
                                                       ; instructions
010C  CC                         INT    3

010D                   START     ENDP


                       ; PROCEDURE :  FACTO
                       ; ABSTRACT :     Recursive procedure that computes the factorial of
                       ;                a  number. It takes its parameter from the  stack
                       ;                and returns the result on the stack.
                       ; SAVES    : All registers used


010D                   FACTO     PROC   NEAR

```

```
010D  9C                        PUSHF                 ; save flags and registers on
                                                      ; stack
010E  50                        PUSH   AX
010F  52                        PUSH   DX
0110  55                        PUSH   BP
0111  8B EC                     MOV    BP,SP          ; point BP at top of stack
0113  8B 46 0A                  MOV    AX,[BP+10]     ; copy no. from stack to AX
0116  3D 0001                   CMP    AX,001H        ; if no. not equal to 1then
                                                      ; go on
0119  75 0D                     JNE    GO_ON          ; and compute factorial
011B  C7 46 0C 0001             MOV    WORD PTR[BP+12],1H      ; else load factorial
                                                               ; of one in
0120  C7 46 0E 0000             MOV    WORD PTR[BP+14],0H      ; stack and return  to
                                                               ; mainline
0125  EB 1A 90                  JMP    EXIT

0128  83 EC 04         GO_ON:   SUB    SP,0004H       ; make space in stack for
                                                      ; preliminary factorial
012B  48                        DEC    AX             ; decrement number in AX
012C  50                        PUSH   AX             ; save number - 1 on stack
012D  E8 010D R                 CALL   FACTO          ; compute factorial of no.- 1
0130  8B EC                     MOV    BP,SP          ; point BP at top of stack
0132  8B 46 02                  MOV    AX,[BP+2]      ; last (N-1)! from stack  to
                                                      ; AX
0135  F7 66 10                  MUL    WORD PTR[BP+16]         ; multiply by previous N
0138  89 46 12                  MOV    [BP+18],AX     ; copy new factorial to stack
013B  89 56 14                  MOV    [BP+20],DX
013E  83 C4 06                  ADD    SP,0006H                ; point  stack  pointer to
                                                               ; pushed register

0141  5D               EXIT:    POP    BP                      ; restore registers
0142  5A                        POP    DX
0143  58                        POP    AX
0144  9D                        POPF
0145  C3                        RET
0146                   FACTO    ENDP

0146                   CODESEG  ENDS
                                END    BEGIN
```

# PROGRAM 12 32 BIT DIVISION

```

                       PAGE 55,132
                       ;                     Program 23
                       ;
                       ; This  program demonstrates the division of a 32-bit number  by a
                       ; 16-bit number.
                       ;
                       ; ABSTACT :      This procedure divides a 32-bit number by 16-bit
                       ;                number  to  give a 32-bit quotient  and a  16-bit
                       ;                remainder.  The parameters are passed to and from
                       ;                the procedure in the following way :
                       ;                Dividend : low word in AX, high word in DX
                       ;                Divisor  : word in CX
                       ;                Quotient : low word in AX and high word in DX
                       ;                Remainder : in CX
                       ; Carry : carry set if try to divide by zero
                       ; USES  : AX, BX, CX, DX, BP, FLAGS



0000                   CODESEG   SEGMENT     PARA     PUBLIC   ’CODE’
                                 ASSUME      CS:CODESEG, DS:CODESEG
                                 ASSUME      ES:CODESEG, SS:CODESEG

0100                             ORG    100H
0100                   BEGIN:
0100                   START     PROC   NEAR

0100  83 F9 00                   CMP    CX,0H         ; check  for   illegal
                                                      ; divide
0103  74 17                      JE     ERROR_EXIT    ; divisor = 0 so exit
0105  8B D8                      MOV    BX,AX         ; save lower  order  of
                                                      ; dividend
0107  8B C2                      MOV    AX,DX         ; position high word  for
                                                      ; divide
0109  BA 0000                    MOV    DX,0000H      ; zero DX
010C  F7 F1                      DIV    CX            ; AX/CX, quotient in  AX,
                                                      ; remainder in DX
010E  8B E8                      MOV    BP,AX         ; save higher  order  of
                                                      ; final result
0110  8B C3                      MOV    AX,BX         ; get back lower order of
```

```
                                                      ; dividend
0112  F7 F1                      DIV    CX            ; AX/CX quotient in AX
                                                      ; remainder in DX
0114  8B CA                      MOV    CX,DX         ; pass remainder back  in
                                                      ; CX
0116  8B D5                      MOV    DX,BP         ; pass  higher   order
                                                      ; result back in DX
0118  F8                         CLC                  ; clear carry to indicate
                                                      ; valid result
0119  EB 02 90                   JMP    EXIT          ; finished

011C  F9               ERROR_EXIT:STC                 ; set carry to  indicate
                                                      ; divide by zero
011D  CC               EXIT:     INT    3


011E                   START     ENDP

011E                   CODESEG   ENDS
                                 END    BEGIN
```

# PROGRAM 13    CASE CONVERSION OF STRING

```

                       PAGE 55,132
                       ;                      Program 24
                       ;
                       ; This  program  shows the conversion of characters from lowercase
                       ; to  uppercase. The string of 32 characters at offset 200h is
                       ; converted to upper case.


0000                   CODESEG   SEGMENT      PARA     PUBLIC   ’CODE’
                                 ASSUME       CS:CODESEG, DS:CODESEG
                                 ASSUME       ES:CODESEG, SS:CODESEG

0100                             ORG    100H
0100                   BEGIN:
0100                   START     PROC   NEAR

0100  B9 0020                    MOV    CX,32          ; no. of  characters  to
                                                       ; change
0103  8D 1E 0200 R               LEA    BX,TITLEX      ; first charac.to change
0107                   B20:
0107  8A 27                      MOV    AH,[BX]        ; charac. from TITLEX
0109  80 FC 61                   CMP    AH,61H         ; is it
010C  72 0A                      JB     B30            ; lower
010E  80 FC 7A                   CMP    AH,7AH         ; case
0111  77 05                      JA     B30            ; letter ?
0113  80 E4 DF                   AND    AH,11011111B   ; yes - convert
0116  88 27                      MOV    [BX],AH        ; restore in TITLEX
0118                   B30:
0118  43                         INC    BX             ; set for next character
0119  E2 EC                      LOOP   B20            ; loop for 32 times

011B  CC                         INT    3

011C                   START     ENDP


0200                             ORG    200H

0200 43 68 61 6E 67 65 TITLEX    DB     ’Change this to uppercase letters’
     20 74 68 69 73 20
     74 6F 20 75 70 70
     65 72 63 61 73 65
     20 6C 65 74 74 65
     72 73
```

```


0220                   CODESEG          ENDS
                                        END   BEGIN
```

# PROGRAM 14 BCD STRING ADDITION

```
                       PAGE 55,132
                       ;                      Program 25
                       ;
                       ; Addition of two unpacked BCD (ASCII) strings
                       ;

0000                   CODESEG   SEGMENT      PARA     PUBLIC   ’CODE’
                                 ASSUME       CS:CODESEG, DS:CODESEG
                                 ASSUME       ES:CODESEG, SS:CODESEG

0100                             ORG    100H
0100                   BEGIN:
0100                   START     PROC   NEAR

0100  F8                         clc                            ; no carry initially
0101  FC                         cld                            ; forward stings
0102  BE 0200 R                  mov    si,offset string_1      ; establish string pointers
0105  BF 0204 R                  mov    di,offset string_2
0108  B9 0004 90                 mov    cx,len_str
010C  E3 07                      jcxz   finish
010E  AC               cycle:    lods   string_1                ; get string_1 element
010F  12 05                      adc    al,[di]                 ; add string_2 element
0111  37                         aaa                            ; correct for ASCII
0112  AA                         stos   string_2                ; result into string_2
0113  E2 F9                      loop   cycle                   ; repeat for entire element

0115  CC               finish:   int    3

0116                   START     ENDP

0200                             org    200h

0200  31 37 35 32      string_1  db     ’1’,’7’,’5’,’2’           ; value is 2571
0204  33 38 31 34      string_2  db     ’3’,’8’,’1’,’4’           ; value is 4183
= 0004                 len_str          equ   string_2 - string_1

0208                   CODESEG   ENDS
                                 END    BEGIN
```

# PROGRAM 15 ASCII NUMBER TO BINARY

```

                       PAGE 55,132
                       ;                     Program 27
                       ;
                       ; This program converts ASCII values to Binary. Maximum ASCII
                       ; value that can be converted is 65535.


0000                   CODESEG   SEGMENT     PARA     PUBLIC   ’CODE’
                                 ASSUME      CS:CODESEG, DS:CODESEG
                                 ASSUME      ES:CODESEG, SS:CODESEG

0100                             ORG    100H
0100                   BEGIN:
0100                   START     PROC   NEAR

                                 ; convert ASCII to Binary

0100  B9 000A                    MOV    CX,10         ; mult factor
0103  8D 36 0205 R               LEA    SI,ASCVAL-1   ; address of ASCVAL
0107  8B 1E 0202 R               MOV    BX,ASCLEN     ; length of ASCVAL
010B                   B20:
010B  8A 00                      MOV    AL,[SI+BX]    ; select ASCII charac.
010D  25 000F                    AND    AX,000FH      ; remove 3-zone
0110  F7 26 0200 R               MUL    MULT10        ; multiply by 10 factor
0114  01 06 0204 R               ADD    BINVAL,AX     ; add to binary
0118  A1 0200 R                  MOV    AX,MULT10     ; calculate next
011B  F7 E1                      MUL    CX            ; 10 factor
011D  A3 0200 R                  MOV    MULT10,AX
0120  4B                         DEC    BX            ; last ASCII character
0121  75 E8                      JNZ    B20           ; no - continue

0123  CC                         INT    3

0124                   START     ENDP

0200                             ORG    200H

0200  0001             MULT10    DW     1             ; holds dec. multilpier
0202  0004             ASCLEN    DW     4             ; no. of bytes in value
0204  0000             BINVAL    DW     0             ; to hold binary value
0206  31 32 33 34      ASCVAL    DB     ’1234’        ; value to be converted


020A                   CODESEG   ENDS
                                 END    BEGIN
```

# PROGRAM 16 SQUARE ROOT USING 8087 INSTRUCTION

```


              PAGE 55,132
              ;                          Program 29
              ;
              ; Square root of number using 8087.
              ;
              ; Using 8087 instructions find square root of a number
              ;at offset S_DATA and store it at offset RESULT. The
              ;number and the results are stored in the SHORT REAL format.
              ;This format requires 4 bytes which includes the SIGN (1 bit),
              ;BIASED EXPONENT (8 bits with bias of 127 decimal) and
              ;SIGNIFICAND (23 BITS).
              ;
              ;for ex. the sq. root of 40800000h is 04000000h
              ;
              ;NOTE: MASM 1.0 DOES NOT SUPPORT 8087 INSTRUCTIONS.
              ;
              ;STAR-86 user’s should assemble this program with a later
              ;version of MASM or assemble it directly with DEBUG.
              ;
              ;
0000                    CODESEG   SEGMENT     PARA     PUBLIC    ’CODE
                        ASSUME    CS:CODESEG, DS:CODESEG
                        ASSUME    ES:CODESEG, SS:CODESEG

0100                              ORG    100H
0100                    BEGIN:
0100                    START     PROC   NEAR

0100  9B  D9  06  0200 R          fld    dword ptr s_data        ;data address
0105  9B  D9  E1                  fabs                           ;positive number
0108  9B  D9  FA                  fsqrt
010B  9B  D9  1E  0204 R          fstp   dword ptr result

0110  CC                          int    3

0110                    START     ENDP


0200                              ORG    200H

0200  00008040          s_data    dd     40800000h     ; 4 bytes for data
0204  00000000          result    dd     0             ; 4 bytes for result

0208                    CODESEG   ENDS
                        END       BEGIN
```

# APPENDIX A
# CONNECTOR DETAILS

## J1 (U6) 8255 # 1     Used to interface Printer in Serial mode

| PIN NO. | SIGNAL | PIN NO. | SIGNAL |
|---|---|---|---|
| 1 | LPA3 | 14 | LPC0 |
| 2 | LPA2 | 15 | LPC1 |
| 3 | LPA1 | 16 | LPC2 |
| 4 | LPA0 | 17 | LPC3 |
| 5 | VCC | 18 | LPB0 |
| 6 | LPA7 | 19 | LPB1 |
| 7 | LPA6 | 20 | LPB2 |
| 8 | LPA5 | 21 | LPB3 |
| 9 | LPA4 | 22 | LPB4 |
| 10 | LPC7 | 23 | LPB5 |
| 11 | LPC6 | 24 | LPB6 |
| 12 | LPC5 | 25 | LPB7 |
| 13 | LPC4 | 26 | GND |

# Printer Interace Signals

# J1 (U6) 8255 # 1

| PIN NO. | SIGNAL NAME |
|---|---|
| 4 | PD0 |
| 3 | PD1 |
| 2 | PD2 |
| 1 | PD3 |
| 9 | PD4 |
| 8 | PD5 |
| 7 | PD6 |
| 6 | PD7 |
| 13 | Strobe * - O/p |
| 14 | Busy * - I/p |

# J2 (U7) 8255 # 2 Used to interface DYNA-PIO Study cards

| PIN NO. | SIGNAL | PIN NO. | SIGNAL |
|---|---|---|---|
| 1 | UPA3 | 14 | UPC0 |
| 2 | UPA2 | 15 | UPC1 |
| 3 | UPA1 | 16 | UPC2 |
| 4 | UPA0 | 17 | UPC3 |
| 5 | VCC | 18 | UPB0 |
| 6 | UPA7 | 19 | UPB1 |
| 7 | UPA6 | 20 | UPB2 |
| 8 | UPA5 | 21 | UPB3 |
| 9 | UPA4 | 22 | UPB4 |
| 10 | UPC7 | 23 | UPB5 |
| 11 | UPC6 | 24 | UPB6 |
| 12 | UPC5 | 25 | UPB7 |
| 13 | UPC4 | 26 | GND |

# J8 (Bus Expansion Connector)

| PIN NO. | SIGNAL | PIN NO. | SIGNAL |
|---|---|---|---|
| 1 | BD8 | 11 | BA17 |
| 2 | BD9 | 12 | BA19 |
| 3 | BD10 | 13 | BA16 |
| 4 | BD11 | 14 | BA18 |
| 5 | BD12 | 15 | NC |
| 6 | BD13 | 16 | NC |
| 7 | BD14 | 17 | VCC |
| 8 | BD15 | 18 | VCC |
| 9 | LOCK* | 19 | GND |
| 10 | BBHE | 20 | GND |

# J9 (Keyboard/Display Connector)

| PIN NO. | SIGNAL | PIN NO. | SIGNAL |
|---|---|---|---|
| 1 | RL7 | 14 | VCC |
| 2 | RL6 | 15 | OA0 |
| 3 | RL5 | 16 | OA1 |
| 4 | RL4 | 17 | OA2 |
| 5 | RL3 | 18 | OA3 |
| 6 | RL2 | 19 | OB0 |
| 7 | RL1 | 20 | OB1 |
| 8 | RL0 | 21 | OB2 |
| 9 | VCC | 22 | OB3 |
| 10 | SL3 | 23 | IRQ3 |
| 11 | SL2 | 24 | KBDRST |
| 12 | SL1 | 25 | GND |
| 13 | SL0 | 26 | NC |

# J7 (DYNA Bus) To interface DYNA Study Cards

| PIN NO. | SIGNAL | PIN NO. | SIGNAL |
|---|---|---|---|
| 1 | VCC | 26 | BA10 |
| 2 | VCC | 27 | BA1 |
| 3 | GND | 28 | BA9 |
| 4 | GND | 29 | BA0 |
| 5 | BUSREADY | 30 | BA8 |
| 6 | ALE | 31 | WR* |
| 7 | BBD3 | 32 | RD* |
| 8 | BBD7 | 33 | IORQ* |
| 9 | BBD2 | 34 | IO/M* |
| 10 | BBD6 | 35 | IOEXP* |
| 11 | BBD1 | 36 | IOEXP1* |
| 12 | BBD5 | 37 | IOEXP2* |
| 13 | BBD0 | 38 | IOEXP3* |
| 14 | BBD4 | 39 | NMIRQ |
| 15 | BA7 | 40 | CS2* |
| 16 | BA15 | 41 | HLDA |
| 17 | BA6 | 42 | HOLD* |
| 18 | BA14 | 43 | INTA* |
| 19 | BA5 | 44 | NC |
| 20 | BA13 | 45 | BUSCLK |
| 21 | BA4 | 46 | IRQ6 |
| 22 | BA12 | 47 | CPURST |
| 23 | BA3 | 48 | CPURESET* |
| 24 | BA11 | 49 | IRQ5 |
| 25 | BA2 | 50 | IRQ7 |

***Note :***

BA0-BA15 = Buffered Address lines
BBD0-BBD7 = Buffered Data lines

IOEXP*, IOEXP1*, IOEXP2*, IOEXP3* are all active low I/O decoding chip select. Refer to Chapter 3 Memory & I/O details.

CS2* - is a active low memory decoding chip select. Refer to Chapter 3 Memory & I/O detials.

IRQ5, 6, 7 are edge triggered 8259 interrupt I/ps.

IORQ* - Active low I/O Request signal

IO/M* - Active low Mem Request signal

## J5 (Serial Connector 9 DTM)

| PIN NO. | SIGNAL |
|---|---|
| 1 | NC |
| 2 | RX |
| 3 | TX |
| 4 | DTR |
| 5 | GND |
| 6 | DSR |
| 7 | RTS |
| 8 | CTS |
| 9 | NC |

## J3 (Main Power Supply Connector)

| PIN NO. | SIGNAL |
|---|---|
| 1 | GND |
| 2 | VCC |
| 3 | VCC |
| 4 | + 12V |
| 5 | - 12V |
| 6 | + 30 |

# J6 (Timer Connector) 7-pin Relimate

| PIN NO. | SIGNAL |
|---|---|
| 1 | CLK1 |
| 2 | GATE1 |
| 3 | OUT1 |
| 4 | CLK2 |
| 5 | GATE2 |
| 6 | OUT2 |
| 7 | GND |

# J4 (Power Connector for Dyna-PIO Cards)

| PIN NO. | SIGNAL |
|---|---|
| 1 | GND |
| 2 | GND |
| 3 | VCC |
| 4 | VCC |
| 5 | + 12 V |
| 6 | - 12V |
| 7 | + 30V |

# APPENDIX B
# PIO CARDS FOR DYNA 86

## LIST OF PIO CARDS SUPPORTED BY DYNA 86

| | **Code** | **Card Function** |
|---|---|---|
| 1) | ADC-01 | A/D Card with ADC-0809 convertor chip. The card supports one channel of 8 bit resolution and 100 micro second conversion time. Voltage range is 0-5 V. |
| 2) | DAC-01 | D/A Card with DAC-0800 convertor chip. The card supports one channel of 8 bit resolution and 100 nano second settling time. |
| 3) | HEX-PAD | Hex Keypad Card with 24 keys arranged as a 4 x 6 matrix. It is very useful for testing routines for key closure and debounce. |
| 4) | ELV-SIM | Elevator Simulator card with LED indicators and push buttons for ground plus three floors. |
| 5) | LCI-5528-II | Logic Indicator Card with 16 LED indicators for output ports and 8 numbers of supply and ground strapping connectors each to be used for input ports. |
| 6) | STP | Stepper Motor driver Card . It has all the driver circuits required to interface one DC stepper motor. |

| | **Code** | **Card Function** |
|---|---|---|
| 7) | SER-DISP | Provides 4 seven segment LED displays controlled by serial data output on a PIO line from the kit. Serial to parallel conversion is done by using 74164 shift registers. |
| 8) | TRAFFIC-PIO | It is used to simulate the traffic control sequence at a junction. |

## PIO PROGRAMS - Using PIO Cards in Hex Keypad Mode

For Hex Keypad Mode, the program for PIO cards is given in the BIOS ROM. This program allows user to operate PIO cards in Hex Keypad mode of operation. These cards are connected to the DYNA-86 mother-board through a 26 line FRC cable to connector J2. All program inputs are taken from the Hex Keypad. All the display will be on the 7 segment displays. The procedure for using the program is as follows :

1) Press PIO key.
2) The 7 segment displays will show "PIO-PrOG".
3) Press INR key, the message "SELC" is displayed in the address field.
4) Select the desired sub-program by pressing one of the following keys :

a) Key 0 to select ADC01 Interface program.
b) Key 1 to select DAC01 Interface program.
c) Key 2 to select Elevator Simulator Interface.
d) Key 3 to select Stepper Motor Controller Interface.
e) Key 4 to select PIO Hexkeypad interface.
f) Key 5 to select Logic Controller interface.
g) Key 6 to select Serial Display card.
h) Key 7 to select Traffic Controller card.

# ADC01 INTERFACE

1) Connect ADC card to mother-board connector J2 with correct polarity.
2) Connect +12V and -12V supply lines to the ADC card.
3) Apply the analog input at the Phono Jack on the ADC card.
4) Press key 0 on the Hex Keypad.
5) Display will show "AdC" on address field and digital value (0 to FF) of the input is displayed on data field. It remains in a loop of scan and display.
6) Press any key (except hex keys 0 to F) to return to the main menu.

# DAC01 INTERFFACE

1) Connect the DAC card to mother-board connector J2 with correct polarity.
2) Connect +12V and -12V supply lines to the DAC card.
4) Press key 1 on the Hex Keypad.
5) Display will show "dAC" on address field.
a) Press key 0 to select Ramp Waveform generation.
b) Press key 1 to select Triangular Waveform generation.
c) Press key 2 to select Charging/Discharging type Waveform generation.
d) Press any key (except hex keys 0 to F) to return to the main program.
6) Display shows "rA" for ramp, "trIA" for triangular, "CHAr" for Charging/Discharging type waveform.
7) Observe the waveform on the ANALOG OUT Phone jack on DAC card with an oscilloscope.
8) Press any key (except hex keys 0 to F) to return to the DAC waveform selection menu.

# ELEVATOR SIMULATOR INTERFACE

1) Connect the card to the mother board connector J2 with correct polarity.
2) Press key 2 on the Hex Keypad.
3) Display will show "ELE" on address field.
4) Now the card can be operated as per the instructions given in the card manual.
5) Press any key (except hex keys 0 to F) to return to the main menu.

# STEPPER MOTOR CONTROLLER INTERFACE

1) Connect the card to connector J2 with correct polarity.
2) Press key 3 in main program.
3) Display will show "StEPPEr".
4) Select any of the following options.

   a) Press key 0 for continuous rotation with fixed speed and direction.
   b) Press key 1 for User selectable speed, steps and direction.
   c) Press any key (except hex keys 0 to F) to return main menu.

5) If your choice was key 0, display will show "COnt" on address field and motor starts rotating at fixed speed in the clockwise direction. Press any key (excepe 0 to F) to return to stepper selection program.
6) If your choice was key 1, display will show "SPd". Enter the value for speed (800 to FFFF). Press any key (except keys 0 to F) and display shows "StP" on address field. Enter number of steps value (1 to FFFF). Press any key (except hex keys 0 to F). Display shows "dIr". Enter 0 for clockwise or 1 for anticlock direction. Motor starts rotating in the desired direction as the display shows "dIr" on address field and "CLC" (for clockwise) or "ACLC" (for anticlockwise) direction.
7) After end of run it loops back to the stepper option selection program.

# HEX KEYPAD INTERFACE

1) Connect the card to mother board connector J2 with correct polarity.
2) Press key 4 on the Hex Keypad.
3) Display will show "HPAd" on address field.
4) Now the card can be operated. (Refer manual for operational details of the card).
5) Depending on the key pressed on Hex Keypad PIO card, codes 00 to 17H is displayed on the data field.
6) Press any key (except hex keys 0 to F) to return to main menu.

# LOGIC CONTROLLER INTERFACE CARD

1) Connect the card to connector J2 with correct polarity.
2) Press key 5 on the Hex keypad.
3) Display will show "LCI" on the address field.
4) Select any of the following options.

    a) Press key 0 to select Binary Count Display on LEDs.
    b) Press key 1 for alternate flashing of LED rows.
    c) Press key 2 to select Synchronised flashing of LED rows.
    d) Press any key (except hex keys 0 to F) to return main program.

5) Display shows "LCI" on address field and "bCnt" or "AFLS" or "FLSH" on data field depending on the selected option.
6) Press any key (except hex keys 0 to F) to return to LCI option selection menu.

## SERIAL DISPLAY CARD

1) Connect the card to connector J2 with correct polarity.
2) Press key 6 on the Hex keypad.
3) Display will show "SEr" on the address field.
4) Press any key (except 0 to F) on hex keypad. The number will be displayed in Serial display card and the previous number will be shifted rightside.
5) Press any key (except hex keys 0 to F) to return to main menu.

## TRAFFIC CONTROLLER CARD

1) Connect the card to connector J2 with correct polarity.
2) Press key 7 on the Hex keypad.
3) Display will show "trA" on the address field.
4) Press any key (except 0 to F) on hex keypad. A demo program for the traffic control of a junction will be executed.
5) Press any key (except hex keys 0 to F) to return to main menu.

# APPENDIX C

# DYNA-SERIES STUDY CARDS SUPPORTED BY DYNA-86

**DYNA-PIO/1** Study of 8212
Consists of one 8212, buffers to drive LEDs and VCC, GND tags.

**DYNA-PIO/2** Study of 8255
Consists of 8255 with tags for all I/O ports, buffers to drive LEDs, VCC and GND tags.

**DYNA-TIMER** Study of 8253
Consists of one 8253 with tags for all the counters, buffers, VCC and GND tags, LEDs.

**DYNA-SERIAL** Study of USART 8251
Consists of two USARTs (Universal Synchronous Asynchronous Receiver Transmitter), buffers to drive LEDs, interrupt and GND tags.

**DYNA-LBDR** Study of Latch, Buffer, Decoder and RAM
Consists of buffers, decoders, latches, RAM, GND tag and LEDs.

**DYNA-KBDISP** Study of 8279
Consists of one 8279, buffers, VCC and GND tags and LEDs.

**DYNA-DMA** Study of 8237
Consists of 8237-A, RAM, buffers to drive LEDs, VCC and GND tags.

**DYNA-DMA** Study of 8257
Consists of 8257-A, RAM, buffers to drive LEDs, VCC and GND tags.

**DYNA-THUMBWHEEL V2.1**

Consists of a latch, 8 bit magnitude comparator, two 7 segment displays, pair of THUMBWHEELS, one LED etc.

**DYNA-TRAFFIC CONTROLLER V2.1**

Consists of a buffer, 4 latches, LEDs etc.

**DYNA-DCM**

Consists of latch, DC Motor, LEDs etc.

For details refer to DYNA-86 Study Cards User’s Manual.

# APPENDIX D
# INTEL HEX FORMAT

The INTEL HEX FORMAT is one of the standards defined to transfer data between a target system and the host computer. This is used in the **SERIAL** mode of STAR-86 when it is connected to the host computer like the IBM PC.

Each record in the INTEL HEX FORMAT contains information about the record type, length, memory load address and checksum in addition to data. Each transfer is limited to 128 bytes of program data. The general format of a record, shown with spaces seperating each field, is :

| **Record Mark** | **Record Length** | **Load Address** | **Record Type** | **Program Data** | **Checksum** |
|---|---|---|---|---|---|
| : | ## | aaaa | tt | dd...dd | cc |

where :

: is the keyword used to signal start of record.

## is a two ASCII hexadecimal value indicating the record length. It is the number of data bytes in the record.

aaaa is a four ASCII hexadecimal value indicating the program memory load-address. It is the address at which the first byte is to be loaded. (For record types 01-03 [ next item ], this field contains "0000").

tt is a two ASCII hexadecimal value representing the record type.

| tt | ## |
|---|---|
| 00 - data record | actual data length |
| 01 - end of file record | 00 |
| 02 - extended address | 02 |
| 03 - start address record | 04 |

dd...dd is a two ASCII hexadecimal value per byte representation of the program. When the record type (tt) is extended address (02) the following four ASCII hexadecimal value (dddd) represents the Code Segment base for the subsequent data record. For each record type the data is as follows :

| tt | dd..dd |
|---|---|
| 00 | A pair of hex digits representing the ASCII code for each data byte, where the high order digit is the first digit of each pair. |
| 01 | None. |
| 02 | The Segment Base Address (SBA) is a four ASCII hexadecimal value. |
| 03 | CS and IP (8 digits). |

cc is a two ASCII hexadecimal value representing the negative sum of the record. Beginning with the record length "##" and ending with the check sum "cc", the hexadecimal sum, taken two at a time, modulo 256 should be zero.

The **W** Command in the **SERIAL** mode, would generate the Hex codes in the form given below. Spaces have been included to show the various fields. See the ‘tt’ field for the various record types.

- `W1000:0,1F`

```
: 02 0000 02 10 00 EC
: 10 0000 00 00 00 FF FE EF F7 DF E9 FF DF FF FF FF 7F FF FF ED
: 10 0010 00 FE 03 FF 60 FF A0 FF 46 FF FF  FF FF FF FF FF FF A4
: 00 0000 01 FF
```

- `W1000:0,1F,1000:10`

```
: 04 0000 03 10 00 00 10 D9
: 02 0000 02 10 00 EC
: 10 0000 00 00 00 FF FE EF F7 DF E9 FF DF FF FF FF 7F FF FF ED
: 10 0010 00 FE 03 FF 60 FF A0 FF 46  FF FF FF FF FF FF FF FF A4
: 00 0000 01 FF
```

# APPENDIX E
# 8086 INSTRUCTION SET TABLE

| Instruction | Operands | Clocks | Transfers | Bytes | Coding Example |
|---|---|---|---|---|---|
| AAA | (no operands) | 4 | -- | 1 | AAA |
| AAD | (no operands) | 60 | -- | 2 | AAD |
| AAM | (no operands) | 83 | -- | 1 | AAM |
| AAS | (no operands) | 4 | -- | 1 | AAS |
| ADC | register, register | 3 | -- | 2 | ADC AX, SI |
| | register, memory | 9 +EA | 1 | 2-4 | ADC DX, BETA [SI] |
| | memory, register | 16+EA | 2 | 2-4 | ADC ALPHA [BX] [SI], DI |
| | register, immediate | 4 | -- | 3-4 | ADC BX, 256 |
| | memory, immediate | 17+EA | 2 | 3-6 | ADC GAMMA, 30H |
| | accumulator, immediate | 4 | -- | 2-3 | ADC AL, 5 |
| ADD | register, register | 3 | -- | 2 | ADD CX, DX |
| | register, memory | 9+EA | 1 | 2-4 | ADD DI, [BX], ALPHA |
| | memory, register | 16+EA | 2 | 2-4 | ADD TEMP, CL |
| | register, immediate | 4 | -- | 3-4 | ADD CL, 2 |
| | memory, immediate | 17+EA | 2 | 3-6 | ADD ALPHA, 2 |
| | accumulator, immediate | 4 | -- | 2-3 | ADD AX, 200 |
| AND | register, regoster | 3 | -- | 2 | AND AL, BL |
| | register, memory | 9+EA | 1 | 2-4 | AND CX, FLAG_WORD |
| | memory, register | 16+ EA | 2 | 2-4 | AND ASCII [DI], AL |
| | register, immediate | 4 | -- | 3-4 | AND CX, F0H |
| | memory, immediate | 17+ EA | 2 | 3-6 | AND BETA, 01H |
| | accumulator, immediate | 4 | -- | 2-3 | AND AX, 01010000B |
| CALL | near-proc | 19 | 1 | 3 | CALL NEAR_PROC |
| | far-proc | 28 | 2 | 5 | CALL FAR_PROC |
| | memptr 16 | 21+ EA | 2 | 2-4 | CALL PROC_TABLE [SI] |
| | regptr 16 | 16 | 1 | 2 | CALL AX |
| | memptr 32 | 37+ EA | 4 | 2-4 | CALL [BX], TASK [SI] |

| **Instruction** | **Operands** | **Clocks** | **Transfers** | **Bytes** | **Coding Example** |
|---|---|---|---|---|---|
| CBW | (no operands) | 2 | -- | 1 | CBW |
| CLC | (no operands) | 2 | -- | 1 | CLC |
| CLD | (no operands) | 2 | -- | 1 | CLD |
| CLI | (no operands) | 2 | -- | 1 | CLI |
| CMC | (no operands) | 2 | -- | 1 | CMC |
| CMP | register, register | 3 | -- | 2 | CMP BX, CX |
| | register, memory | 9+ EA | 1 | 2-4 | CMP DH, ALPHA |
| | memory, register | 9+ EA | 1 | 2-4 | CMP [BP+ 2], SI |
| | register, immediate | 4 | -- | 3-4 | CMP BL, 02H |
| | memory, immediate | 10+ EA | 1 | 3-6 | CMP [BX], RADAR [DI], 3420H |
| | accumulator, immediate | 4 | -- | 2-3 | CMP AL, 00010000B |
| CMPS | dest-string, source-string | 22 | 2 | 1 | CMPS BUFF1, BUFF2 |
| | (repeat) dest-string, source-string | 9+ 22/ rep | 2/ rep | 1 | REPE CMPS ID, KEY |
| CWD | (no operands) | 5 | -- | 1 | CWD |
| DAA | (no operands) | 4 | -- | 1 | DAA |
| DAS | (no operands) | 4 | -- | 1 | DAS |
| DEC | reg 16 | 2 | -- | 1 | DEC AX |
| | reg 8 | 3 | -- | 2 | DEC AL |
| | memory | 15+ EA | 2 | 2-4 | DEC ARRAY [SI] |
| DIV | reg 8 | 80-90 | -- | 2 | DIV CL |
| | reg 16 | 144-162 | -- | 2 | DIV BX |
| | mem8 | (86-96) +EA | 1 | 2-4 | DIV ALPHA |
| | mem 16 | (150-166) + EA | 1 | 2-4 | DIV TABLE [SI] |
| ESC | immediate, memory | 8+EA | 1 | 2-4 | ESC 6, ARRAY [SI] |
| | immediate, register | 2 | -- | 2 | ESC 20, AL |

| Instruction | Operands | Clocks | Transfers | Bytes | Coding Example |
|---|---|---|---|---|---|
| HLT | (no operands) | 2 | -- | 1 | HLT |
| IDIV | reg 8 | 101-112 | -- | 2 | IDIV BL |
| | reg 16 | 165-184 | -- | 2 | IDIV CX |
| | mem 8 | (107-118)+EA | 1 | 2-4 | IDIV DIVISOR_BYTE [SI] |
| | mem 16 | (171-190) +EA | 1 | 2-4 | IDIV[BX], DIVISOR_WORD |
| IMUL | reg 8 | 80-98 | -- | 2 | IMUL CL |
| | reg 16 | 128-154 | -- | 2 | IMUL BX |
| | mem 8 | (86-104) + EA | 1 | 2-4 | IMUL RATE_BYTE |
| | mem 16 | (134-160) +EA | 1 | 2-4 | IMUL RATE_WORD [BP]\| [DI] |
| IN | accumulator, immed 8 | 10 | 1 | 2 | IN AL, 0FFEAH |
| | accumulator, DX | 8 | 1 | 1 | IN AX, DX |
| INC | reg 16 | 2 | -- | 1 | INC CX |
| | reg 8 | 3 | -- | 2 | INC BL |
| | memory | 15+ EA | 2 | 2-4 | INC ALPHA [DI] [BX] |
| INT | immed 8 (type = 3) | 52 | 5 | 1 | INT 3 |
| | immed 8 (type = 3) | 51 | 5 | 2 | INT 67 |
| INTO | (no operands) | 53 or 4 | 5 | 1 | INTO |
| IRET | (no operands) | 24 | 3 | 1 | IRET |
| JA/ JNBE | short-label | 16 or 4 | -- | 2 | JA ABOVE |
| JAE/ JNB | short-label | 16 or 4 | -- | 2 | JAE ABOVE_EQUAL |
| JB/ JNAE | short-label | 16 or 4 | -- | 2 | JB BELOW |
| JBE/ JNA | short-label | 16 or 4 | -- | 2 | JNA NOT_ABOVE |
| JC | short-label | 16 or 4 | -- | 2 | JC CARRY_SET |
| JCXZ | short-label | 18 or 6 | -- | 2 | JCXZ COUNT_DONE |

| Instruction | Operands | Clocks | Transfers | Bytes | Coding Example |
|---|---|---|---|---|---|
| JE/ JZ | short-label | 16 or 4 | -- | 2 | JZ ZERO |
| JG/ JNLE | short-label | 16 or 4 | -- | 2 | JG GREATER |
| JGE/ JNL | short-label | 16 or 4 | -- | 2 | JGE GREATER_EQUAL |
| JL/ JNGE | short-label | 16 or 4 | -- | 2 | JL LESS |
| JLE/ JNG | short-label | 16 or 4 | -- | 2 | JNG NOT_GREATER |
| JMP | short-label | 15 | -- | 2 | JMP SHORT |
| | near-label | 15 | -- | 3 | JMP WITHIN_SEGMENT |
| | far-label | 15 | -- | 5 | JMP FAR_LABEL |
| | memptr 16 | 18+ EA | 1 | 2-4 | JMP [BX], TARGET |
| | regptr 16 | 11 | -- | 2 | JMP CX |
| | memptr 32 | 24+ EA | 2 | 2-4 | JMP OTHER, SEG [SI] |
| JNC | short-label | 16 or 4 | -- | 2 | JNC NOT_CARRY |
| JNE/ JNZ | short-label | 16 or 4 | -- | 2 | JNE NOT_EQUAL |
| JNO | short-label | 16 or 4 | -- | 2 | JNO NO_OVERFLOW |
| JNP/ JPO | short-label | 16 or 4 | -- | 2 | JPO ODD_PARITY |
| JNS | short-label | 16 or 4 | -- | 2 | JNS POSITIVE |
| JO | short-label | 16 or 4 | -- | 2 | JO SIGNED_OVRFLW |
| JP/ JPE | short-label | 16 or 4 | -- | 2 | JPE EVEN_PARITY |
| JS | short-label | 16 or 4 | -- | 2 | JS NEGATIVE |
| LAHF | (no operands) | 4 | -- | 1 | LAHF |
| LDS | reg 16, mem 32 | 16+ EA | 2 | 2-4 | LDS SI, DATA, SEG [DI] |
| LOCK | (no operands) | 2 | -- | 1 | LOCK XCHG FLAG, AL |

| Instruction | Operands | Clocks | Transfers | Bytes | Coding Example |
|---|---|---|---|---|---|
| LODS | source-string | 12 | 1 | 1 | LODS CUSTOMER_NAME |
| | (repeat) source-string | 9+ 13/ rep | 1/ rep | 1 | REP LODS_NAME |
| LOOP | short-label | 17/ 5 | -- | 2 | LOOP AGAIN |
| LOOPE/LOOPZ | short-label | 18 or 6 | -- | 2 | LOOPE AGAIN |
| LOOPNE/ LOOPNZ | short-label | 19 or 5 | -- | 2-4 | LOOPNE AGAIN |
| LEA | reg 16, mem 16 | 2+ EA | -- | 2-4 | LEA BX, [BP] [DI] |
| LES | reg 16, mem 32 | 16+ EA | 2 | 2-4 | LES DI, [BX], TEXT_BUFF |
| MOV | memory, accumulator | 10 | 1 | 3 | MOV ARRAY [SI], AL |
| | accumulator, memory | 10 | 1 | 3 | MOV AX, TEMP_RESULT |
| | register, register | 2 | -- | 2 | MOV AX, CX |
| | register, memory | 8+ EA | 1 | 2-4 | MOV BP, STACK_TOP |
| | memory, register | 9+ EA | 1 | 2-4 | MOV COUNT [DI], CX |
| | register, immediate | 4 | -- | 2-3 | MOV CL, 2 |
| | memory, immediate | 10+ EA | 1 | 3-6 | MOV MASK [BX] [SI], 2CH |
| | seg-reg, reg 16 | 2 | -- | 2 | MOV ES, CX |
| | seg-reg, mem 16 | 8+ EA | 1 | 2-4 | MOV DS, SEGMENT_BASE |
| | reg 16, seg-reg | 2 | -- | 2 | MOV BP, SS |
| | memory, seg-reg | 9+ EA | 1 | 2-4 | MOV [BX], SEG_SAVE, CS |
| MOVS | dest-string, source-string | 18 | 2 | 1 | MOVS LINE, EDIT_DATA |
| | (repeat) dest-string, source-string | 9+ 17 rep | 2/ rep | 1 | REP MOVS SCREEN, BUFFER |
| MOVSB/ | (no operands) | 18 | 2 | 1 | MOVSB |
| MOVSW | (repeat) (no operands) | 9+ 17/ rep | 2/ rep | 1 | REP MOVSW |
| MUL | reg 8 | 70-77 | -- | 2 | MUL BL |
| | reg 16 | 118-133<br>(76-83)+ EA | -- | 2 | MUL CX |
| MUL | mem 16 | (124-139)+EA | 1 | 2-4 | MUL BAUD_RATE |

| Instruction | Operands | Clocks | Transfers | Bytes | Coding Example |
|---|---|---|---|---|---|
| NEG | register | 3 | -- | 2 | NEG AL |
| | memory | 16+ EA | 2 | 2-4 | NEG MULTIPLIER |
| NOP | (no operands) | 3 | -- | 1 | NOP |
| NOT | register | 3 | -- | 2 | NOT AX |
| | memory | 16+ EA | 2 | 2-4 | NOT CHARACTER |
| OR | register, register | 3 | -- | 2 | OR AL, BL |
| | register, memory | 9+ EA | 1 | 2-4 | OR DX, PORT ID [DI] |
| | memory, register | 16+ EA | 2 | 2-4 | OR FLAG_BYTE, CL |
| | accumulator, immediate | 4 | -- | 2-3 | OR AL, 0110110B |
| | register, immediate | 4 | -- | 3-4 | OR CX, 01FH |
| | memory, immediate | 17+ EA | 2 | 3-6 | OR [BX] CMD_WORD, 0CFH |
| OUT | immed 8, accumulator | 10 | 1 | 2 | OUT 44, AX |
| | DX, accumulator | 8 | 1 | 1 | OUT DX, AL |
| POP | register | 8 | 1 | 1 | POP DX |
| | seg-reg (CS illegal) | 8 | 1 | 1 | POP DS |
| | memory | 17+ EA | 2 | 2-4 | POP PARAMETER |
| POPF | (no operands) | 8 | 1 | 1 | POPF |
| PUSH | register | 11 | 1 | 1 | PUSH SI |
| | seg-reg (CS legal) | 10 | 1 | 1 | PUSH ES |
| | memory | 16+ EA | 2 | 2-4 | PUSH RETURN_CODE [SI] |
| PUSHF | (no operands) | 10 | 1 | 1 | PUSHF |
| RCL | register, 1 | 2 | -- | 2 | RCL CX, 1 |
| | register, CL | 8+ 4/ bit | -- | 2 | RCL AL, CL |
| | memory, 1 | 15+ EA | 2 | 2-4 | RCL ALPHA, 1 |
| | memory, CL | 20+ EA + 4/ bit | 2 | 2-4 | RCL [BP], PARM, CL |
| RCR | register, 1 | 2 | -- | 2 | RCR BX, 1 |

| Instruction | Operands | Clocks | Transfers | Bytes | Coding Example |
|---|---|---|---|---|---|
| RCR | register, CL | 8+ 4/bit | -- | 2 | RCR BL, CL |
| | memory, 1 | 15+ EA | 2 | 2-4 | RCR [BX], STATUS, 1 |
| | memory, CL | 20+ EA + 4/ bit | 2 | 2-4 | RCR ARRAY [DI], CL |
| REP | (no operands) | 2 | -- | 1 | REP MOVS DEST, SRCE |
| REPE/ REPZ | (no operands) | 2 | -- | 1 | REPE CMPS DATA, KEY |
| REPNE/ REPNZ | (no operands) | 2 | -- | 1 | REPNE SCAS INPUT_LINE |
| RET | intra-segment, no pop | 8 | 1 | 1 | RET |
| | intra-segment, pop | 12 | 1 | 3 | RET 4 |
| | inter-segment, no pop | 18 | 2 | 1 | RET |
| | inter-segment, pop | 17 | 2 | 3 | RET 2 |
| ROL | register, 1 | 2 | -- | 2 | ROL BX, 1 |
| | register, CL | 8+ 4/ bit | -- | 2 | ROL DI, CL |
| | memory, 1 | 15+ EA | 2 | 2-4 | ROL FLAG_BYTE [DI], 1 |
| | memory, CL | 20+ EA + 4/ bit | 2 | 2-4 | ROL ALPHA, CL |
| ROR | register, 1 | 2 | -- | 2 | ROR AL, 1 |
| | register, CL | 8+ 4/ bit | -- | 2 | ROR BX, CL |
| | memory, 1 | 15+ EA | 2 | 2-4 | ROR PORT_STATUS, 1 |
| | memory, CL | 20+ EA + 4/ bit | 2 | 2-4 | ROR CMD_WORD, CL |
| SAHF | (no operands) | 4 | -- | 1 | SAHF |
| SAL/ SHL | register, 1 | 2 | -- | 2 | SAL AL, 1 |
| | register, CL | 8+ 4/ bit | -- | 2 | SHL DI, CL |
| | memory, 1 | 15+ EA | 2 | 2-4 | SHL [BX], OVERDRAW, 1 |
| | memory, CL | 20+ EA + 4/ bit | 2 | 2-4 | SAL STORE_COUNT, CL |
| SAR | register, 1 | 2 | -- | 2 | SAR DX,1 |
| | register, CL | 8+ 4/ bit | -- | 2 | SAR DI, CL |
| | memory, 1 | 15+ EA | 2 | 2-4 | SAR N_BLOCKS, 1 |

| Instruction | Operands | Clocks | Transfers | Bytes | Coding Example |
|---|---|---|---|---|---|
| SAR | memory, CL | 20+ EA + 4/ bit | 2 | 2-4 | SAR N_BLOCKS, CL |
| SBB | register, register | 3 | -- | 2 | SBB BX, CX |
| | register, memory | 9+ EA | 1 | 2-4 | SBB DI, [BX], PAYMENT |
| | memory, register | 16+ EA | 2 | 2-4 | SBB BALANCE, AX |
| | accumulator, immediate | 4 | -- | 2-3 | SBB AX, 2 |
| | register, immediate | 4 | -- | 3-4 | SBB CL, 1 |
| | memory, immediate | 17+ EA | 2 | 3-6 | SBB COUNT [SI], 10 |
| SCAS | dest-string | 15 | 1 | 1 | SCAS INPUT_LINE |
| | (repeat) dest-string | 9+15 / rep | 1/ rep | 1 | REPNE SCAS BUFFER |
| SHR | register, 1 | 2 | -- | 2 | SHR SI, 1 |
| | register, CL | 8+ 4/ bit | -- | 2 | SHR SI, CL |
| | memory, 1 | 15+ EA | 2 | 2-4 | SHR ID_BYTE [SI] [BX], 1 |
| | memory, CL | 20+ EA + 4/ bit | 2 | 2-4 | SHR INPUT_WORD, CL |
| STC | (no operands) | 2 | -- | 1 | STC |
| STD | (no operands) | 2 | -- | 1 | STD |
| STI | (no operands) | 2 | -- | 1 | STI |
| STOS | dest-string | 11 | 1 | 1 | STOS PRINT_LINE |
| | (repeat) dest-string | 9+ 10/ rep | 1/ rep | 1 | REP STOS DISPLAY |
| SUB | register, register | 3 | -- | 2 | SUB CX, BX |
| | register, memory | 9+ EA | 1 | 2-4 | SUB DX, MATH_TOTAL [SI] |
| | memory, register | 16+ EA | 2 | 2-4 | SUB [BP+ 2], CL |
| | accumulator, immediate | 4 | -- | 2-3 | SUB AL, 10 |
| | register, immediate | 4 | -- | 3-4 | SUB SI, 5280 |
| | memory, immediate | 17+ EA | 2 | 3-6 | SUB [BP], BALANCE, 1000 |
| TEST | register, register | 3 | -- | 2 | TEST SI, DI |
| | register, memory | 9+ EA | 1 | 2-4 | TEST SI, END_COUNT |
| | accumulator, immediate | 4 | -- | 2-3 | TEST AL, 00100000B |

| Instruction | Operands | Clocks | Transfers | Bytes | Coding Example |
|---|---|---|---|---|---|
| TEST | register, immediate | 5 | -- | 3-4 | TEST BX, 0CC4H |
| | memory, immediate | 11+ EA | -- | 3-6 | TEST RETURN_CODE, 01H |
| WAIT | (no operands) | 3+ 5n | -- | 1 | WAIT |
| XCHG | accumulator, reg 16 | 3 | -- | 1 | XCHG AX, BX |
| | memory, register | 17+ EA | 2 | 2-4 | XCHG SEMAPHORE, AX |
| | register, register | 4 | -- | 2 | XCHG AL,BL |
| XLAT | source-table | 11 | 1 | 1 | XLAT ASCII_TAB |
| XOR | register, register | 3 | -- | 2 | XOR CX, BX |
| | register, memory | 9+ EA | 1 | 2-4 | XOR CL, MASK_BYTE |
| | memory , register | 16+ EA | 2 | 2-4 | XOR APLHA [SI], DX |
| | accumulator, immediate | 4 | -- | 2-3 | XOR AL, 01000010B |
| | register, immediate | 4 | -- | 3-4 | XOR SI, 00C2H |
| | memory, immediate | 17+ EA | 2 | 3-6 | XOR RETURN_CODE, 0D2H |

**NOTE :** *Add four clocks for each 16-bit word transfer with an odd address.*

The time required to execute each instruction is indicated by the number of clocks specified. If ‘+EA ’ appears in this column, it indicates that additional time is required to calculate the effective address of the operand that is located in the main memory.

This time also depends on the addressing mode used to access the operand and can be obtained from the table given on the following page.

## Effective Address Calculation Time

| EA Components | | Clocks * |
|---|---|---|
| Displacement Only | | 6 |
| Base or Index Only | (BX, BP, SI, DI) | 5 |
| Displacement | | |
| + | | 9 |
| Base or Index | (BX, BP, SI, DI) | |
| Base | BP + DI, BX + SI | 7 |
| + | | |
| Index | BP + SI, BX + DI | 8 |
| Displacement | BP + DI + DISP | |
| + | BX + SI + DISP | 11 |
| Base | | |
| + | BP + SI + DISP | |
| Index | BX + DI + DISP | 12 |

**NOTE :** ** Add 2 clocks for segment override.*

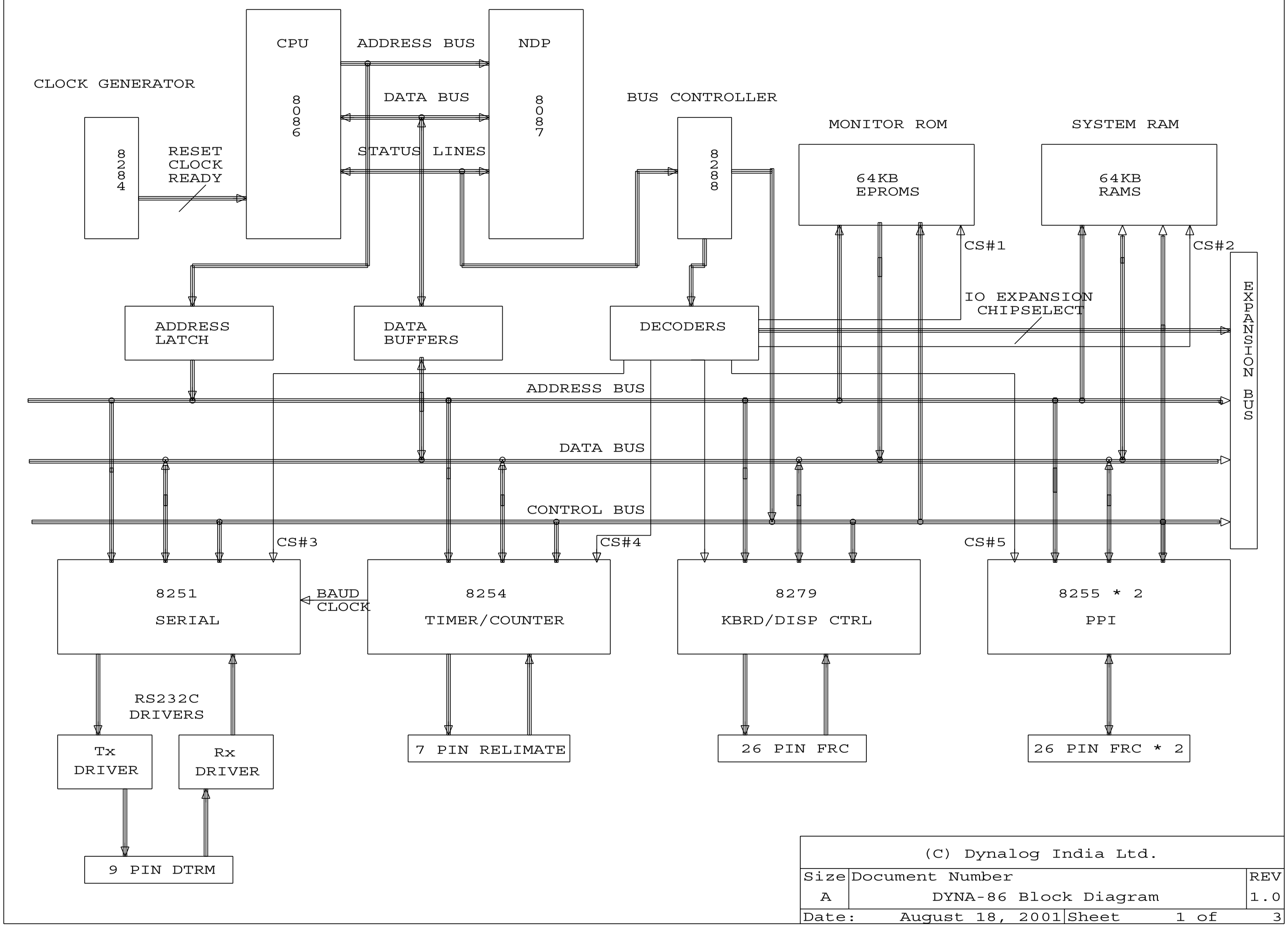
# APPENDIX F
# SYSTEM LAYOUT & CIRCUIT DIAGRAMS

The list of diagrams attached in this appendix are as follows :

1) DYNA-86 Block Diagram.

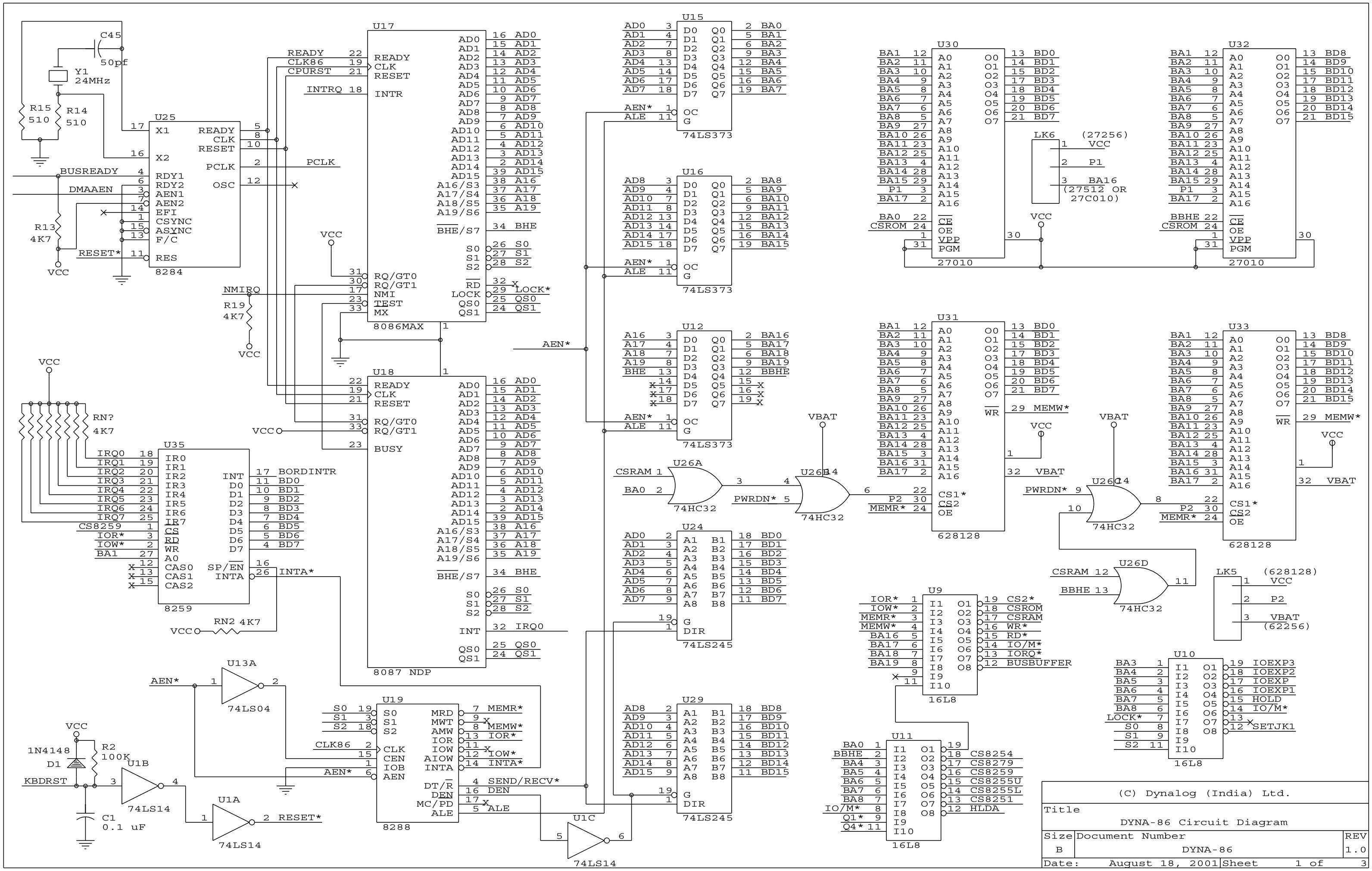
2) DYNA-86 Main Board circuit diagram.

3) DYNA-86 Hex Keypad circuit diagram.



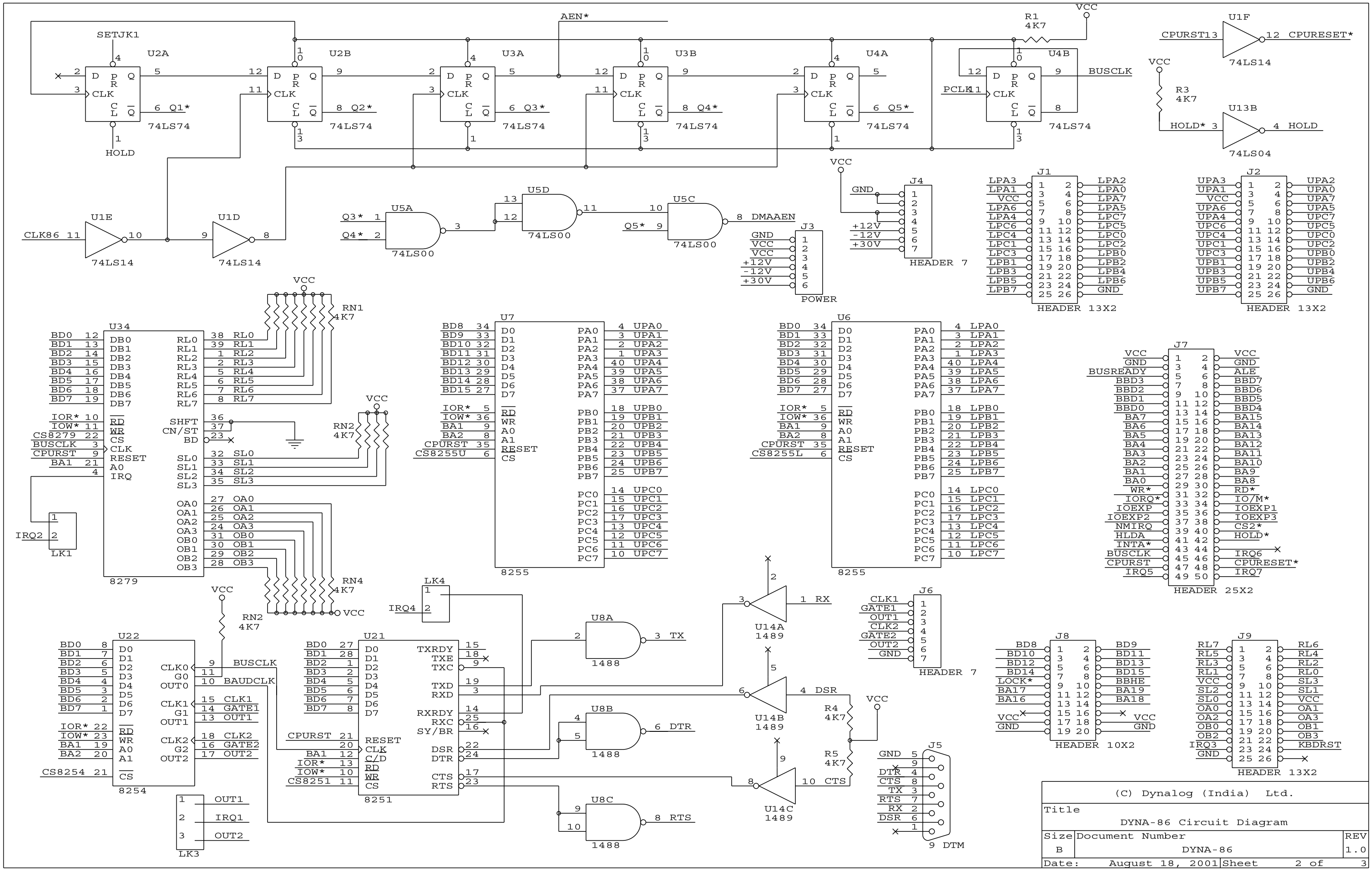
(C) Dynalog India Ltd.

| | | | |
|---|---|---|---|
| Size | Document Number | | REV |
| A | DYNA-86 Block Diagram | | 1.0 |
| Date: | August 18, 2001 | Sheet | 1 of 3 |



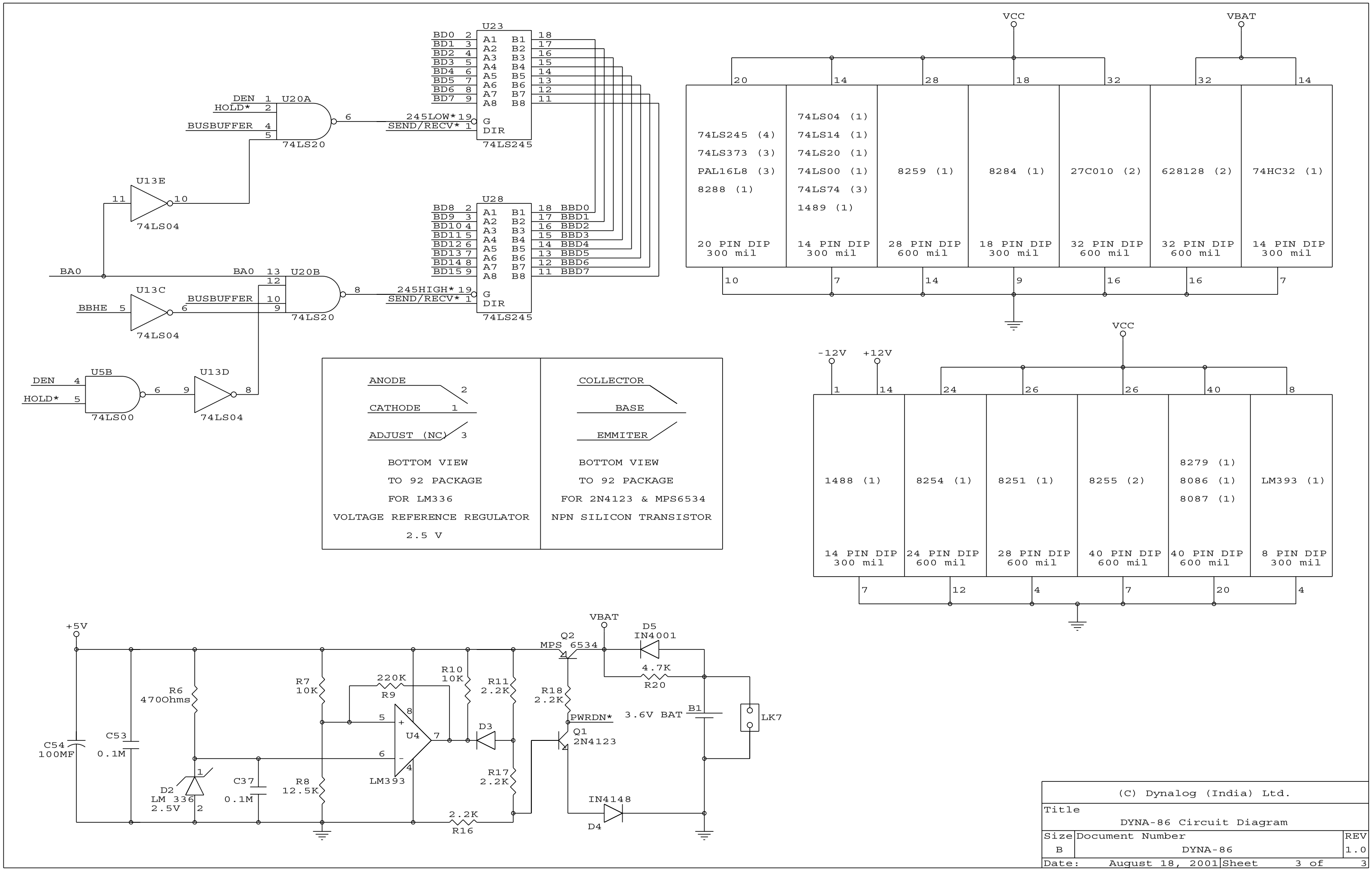
(C) Dynalog (India) Ltd.

| Title | | |
|---|---|---|
| | DYNA-86 Circuit Diagram | |
| Size | Document Number | REV |
| B | DYNA-86 | 1.0 |
| Date: August 18, 2001 | Sheet 1 of 3 | |

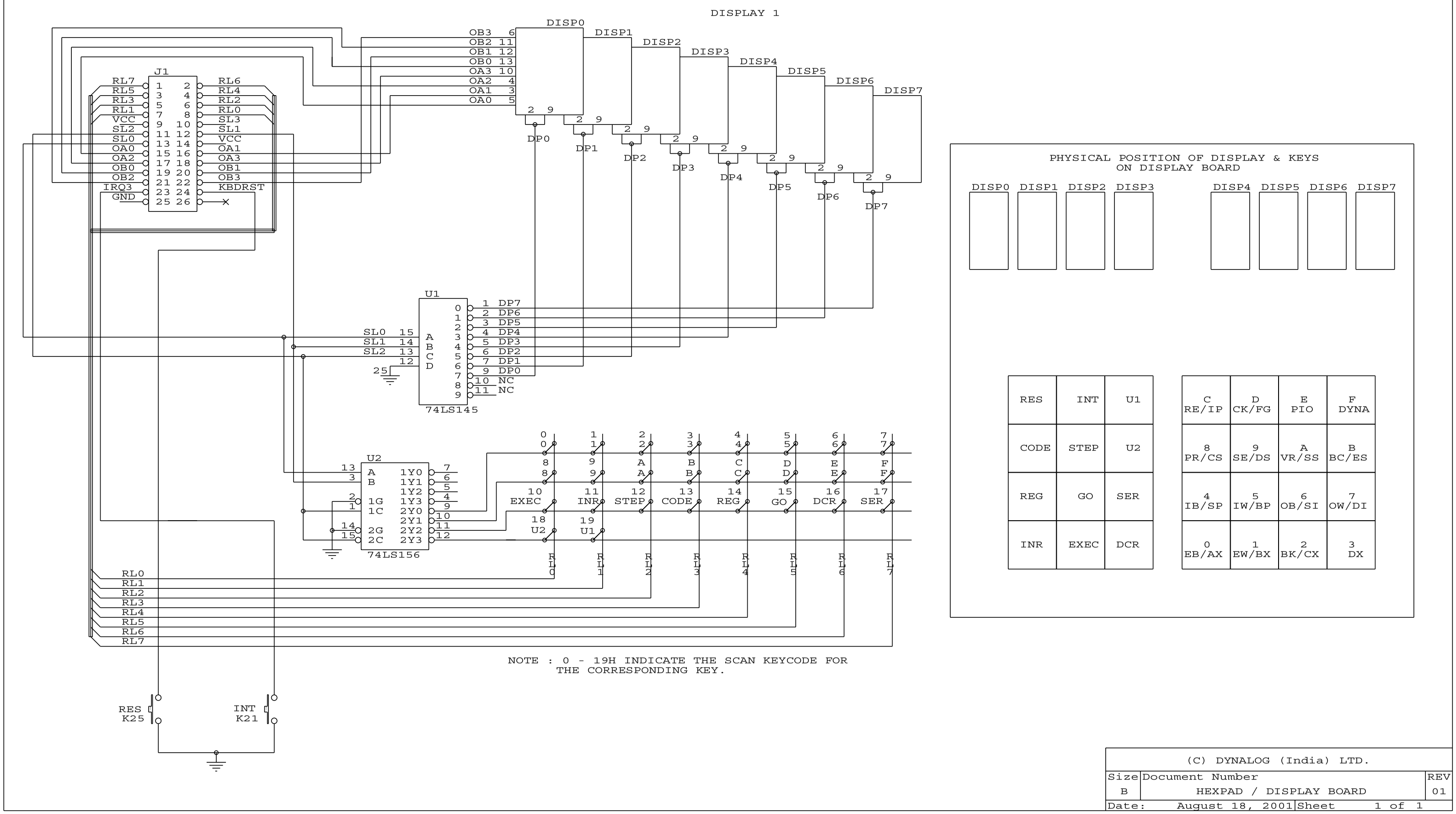


(C) Dynalog (India) Ltd.

| | | |
|---|---|---|
| Title | DYNA-86 Circuit Diagram | |
| Size<br>B | Document Number<br>DYNA-86 | REV<br>1.0 |
| Date: August 18, 2001 | Sheet 2 of 3 | |



| (C) Dynalog (India) Ltd. | | |
|---|---|---|
| Title<br>DYNA-86 Circuit Diagram | | |
| Size<br>B | Document Number<br>DYNA-86 | REV<br>1.0 |
| Date: August 18, 2001 | Sheet 3 of 3 | |



NOTE : 0 - 19H INDICATE THE SCAN KEYCODE FOR THE CORRESPONDING KEY.

(C) DYNALOG (India) LTD.

| Size | Document Number | REV |
|---|---|---|
| B | HEXPAD / DISPLAY BOARD | 01 |
| Date: August 18, 2001 | Sheet 1 of 1 | |