External Project Report on
Computer Organization and Architecture
(EET 2211)

Design a system that replaces a character with a user input character in a given string using 8086 assembly language.

Submitted by

D. Balaji Patro	2241016307
Dipesh Kumar Patro	2241016308
E. Jagadeeswar Patro	2241016309
Essa Rani Patro	2241016310

B. Tech. CSE 4th Semester (Section - 2241044)

INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH
(FACULTY OF ENGINEERING)
SIKSHA 'O' ANUSANDHAN (DEEMED TO BE UNIVERSITY), BHUBANESWAR, ODISHA

Declaration

We, the undersigned students of B. Tech. of **Computer Science & Engineering** Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled **"Design a system that replaces a character with a user input character in a given string using 8086 assembly language."** submitted to **Siksha 'O' Anusandhan Deemed to be University, Bhubaneswar** for the partial fulfillment of the subject **Computer Organization and Architecture (EET 2211)**. We have taken care in all respect to honor the intellectual property right and have acknowledged the contribution of others for using them in academic purpose and further declare that in case of any violation of intellectual property right or copyright we, as the candidate(s), will be fully responsible for the same.

D. Balaji Patro	**Dipesh Kumar Patro**
2241016307	**2241016308**
E. Jagadeeswar Patro	**Essa Rani Patro**
2241016309	**2241016310**

DATE: 13th May, 2024

PLACE: Bhubaneswar

Abstract

This project explores assembly language programming in the context of computer science and architecture, aiming to design a system using 8086 assembly language to replace a character in a given string. Implemented in the emu8086 environment, the system prompts users to input the character to be replaced and the replacement character, providing an interactive interface.

The code consists of initialization, user input handling, character replacement logic, and string printing. Through a reverse iteration process, the program efficiently identifies and substitutes the target character with the user-defined replacement. With comprehensive comments, the code enhances readability for all levels of programmers.

This endeavor not only showcases proficiency in low-level programming but also demonstrates the practical application of assembly language in string manipulation tasks. By bridging theory and application, the project illustrates the symbiotic relationship between computer science principles and architectural intricacies, highlighting the versatility and power of assembly language programming in solving computational challenges.

Contents

Introduction

This project presents a straightforward solution to string manipulation using 8086 assembly language. Implemented in the emu8086 environment, the solution allows users to replace a specified character within a given string interactively.

The system works by prompting users to input both the character to be replaced and its replacement. Then, employing a simple yet effective algorithm, it iterates through the string in reverse, identifying and replacing the target character with the user-defined alternative.

What makes this solution remarkable is its clarity and accessibility. Detailed comments within the code ensure that users of all proficiency levels can understand the process. This project highlights the practical application of assembly language programming, demonstrating its efficiency in tackling real-world computational challenges in computer science and architecture.

Problem Statement

The task at hand requires the development of a robust system capable of performing character replacement within a string using 8086 assembly language. We are tasked with designing an algorithm that efficiently traverses the input string, identifies occurrences of a target character, and replaces them with a user-specified character. The primary challenge lies in devising an efficient and reliable methodology to ensure the integrity of the string is preserved while executing the character replacement process.

I Input : -

- **String 1:** String for performing replacement.
- **Character 1:** chosen by the user to be replaced.
- **Character 2:** to replace the chosen character with.

II Output :-

- **STR1:** Modified String back to the user.

Highlighting the constraints:

Memory Limitations: Due to limited memory resources in the 8086 architecture, the solution must manage memory efficiently to avoid overflow or exhaustion.

Instruction Set Constraints: The solution must operate within the capabilities of the 8086 instruction set, utilizing only supported instructions and operations.

Performance Concerns: Efficient algorithms and data manipulation techniques should be employed to optimize performance, especially for large input strings or frequent replacements.

Input Validation: Robust input validation and error handling mechanisms are essential to ensure system reliability and stability.

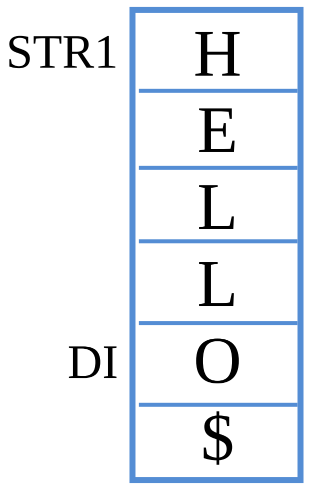
String Length Limitations: The solution must account for constraints on string length to prevent memory issues and performance degradation.

Navigating these constraints ensures the development of a reliable and efficient system for character replacement in 8086 assembly language.

Methodology

Logic Explanation:

Step 1: Initialize DI pointer to the last character address
Step 2: Load CX counter with string length
Step 3: Take copy of both entered characters into registers
Step 4: Compare entered character with address pointed by offset [DI]
Step 5: If match found replace character with new character or go to next step
Step 6: Decrement DI to point out next character
Step 7: Decrement CX, if it is not 0 go to step 4
Step 8: Print modified string



Instructions Needed:

MOV: Moves data from one location to another.
Syntax: MOV destination, source

LEA: Loads the effective address of the source operand (usually used for loading the address of variables).
Syntax: LEA destination, source

INT: Triggers a software interrupt, allowing interaction with the operating system's services.
Syntax: INT interrupt_number

CMP: Compares two operands.
Syntax: CMP operand1, operand2

JZ: Jump if zero (used for conditional branching).
Syntax: JZ target_address

JMP: Unconditional jump to a specified location.
Syntax: JMP target_address

LOOP: Decrement the loop counter and jump if the counter is not zero.
Syntax: LOOP target_address

DEC: Decrements the value of the specified operand.
Syntax: DEC operand

AH, AL, BH, BL, CH, CL, DH, DL: Registers for holding data (AH, BH, CH, DH - High byte, AL, BL, CL, DL - Low byte).

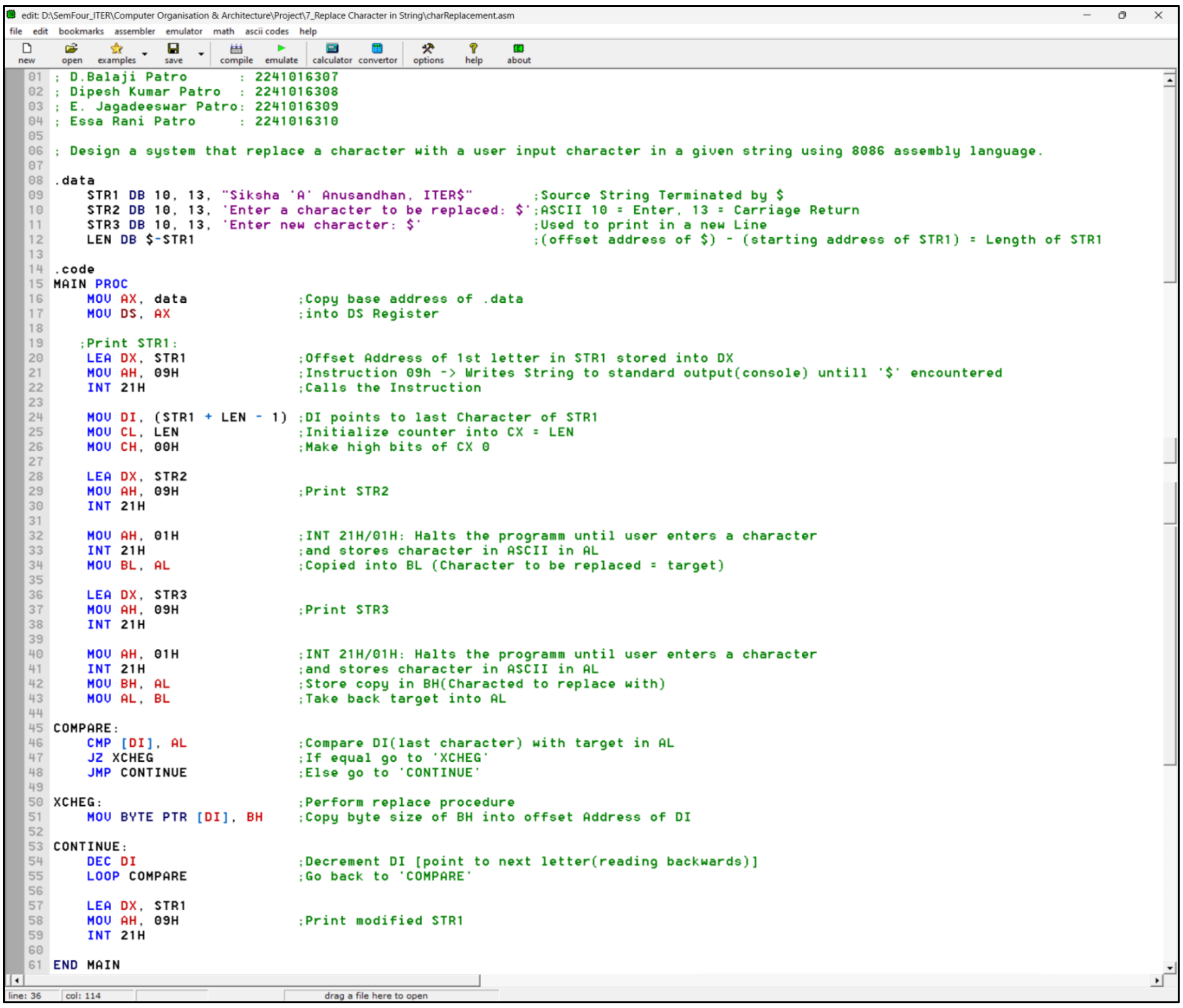
BYTE PTR: Specifies a memory operand size.
Syntax: BYTE PTR [operand]

Implementation

The final program is implemented using the **emu8086 Simulator** [2] :

```
.data
   STR1 DB 10, 13, "Siksha 'A' Anusandhan, ITER$"      ;Source String Terminated by $
   STR2 DB 10, 13, 'Enter a character to be replaced: $'   ;ASCII 10 = Enter,
                                                           ;13 = Carriage Return
   STR3 DB 10, 13, 'Enter new character: $'                ;Used to print in a new Line
   LEN DB $-STR1   ;(offset address of $)-(starting address of STR1)=Length of STR1

.code
MAIN PROC
   MOV AX, data              ;Copy base address of .data
   MOV DS, AX                ;into DS Register

                             ;Print STR1:
   LEA DX, STR1              ;Offset Address of 1st letter in STR1 stored into DX
   MOV AH, 09H               ;Instruction 09h -> Writes String to console until '$' encountered
   INT 21H                   ;Calls the Instruction

   MOV DI, (STR1 + LEN - 1)       ;DI points to last Character of STR1
   MOV CL, LEN                    ;Initialize counter into CX = LEN
   MOV CH, 00H                    ;Make high bits of CX 0

   LEA DX, STR2
   MOV AH, 09H               ;Print STR2
   INT 21H

   MOV AH, 01H               ;INT 21H/01H: Halts the program until user enters a character
   INT 21H                   ;and stores character in ASCII in AL
   MOV BL, AL                ;Copied into BL (Character to be replaced = target)

   LEA DX, STR3
   MOV AH, 09H               ;Print STR3
   INT 21H

   MOV AH, 01H               ;INT 21H/01H: Halts the program until user enters a character
   INT 21H                   ;and stores character in ASCII in AL
   MOV BH, AL                ;Store copy in BH (Character to replace with)
   MOV AL, BL                ;Take back target into AL

COMPARE:
   CMP [DI], AL              ;Compare DI(last character) with target in AL
   JZ XCHEG                  ;If equal go to 'XCHEG'
   JMP CONTINUE              ;Else go to 'CONTINUE'
```

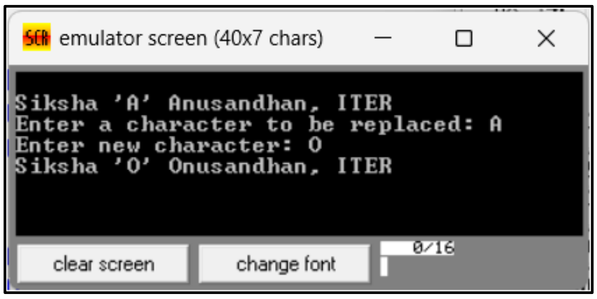
```
XCHEG:                              ;Perform replace procedure
   MOV BYTE PTR [DI], BH            ;Copy byte size of BH into offset Address of DI

CONTINUE:
   DEC DI                           ;Decrement DI [point to next letter(reading backwards)]
   LOOP COMPARE                     ;Go back to 'COMPARE'

   LEA DX, STR1
   MOV AH, 09H                      ;Print modified STR1
   INT 21H

END MAIN
```


edit: D:\SemFour_ITER\Computer Organisation & Architecture\Project\7_Replace Character in String\charReplacement.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
line: 36 col: 114 drag a file here to open

```
; D.Balaji Patro       : 2241016307
; Dipesh Kumar Patro   : 2241016308
; E. Jagadeeswar Patro: 2241016309
; Essa Rani Patro      : 2241016310

; Design a system that replace a character with a user input character in a given string using 8086 assembly language.

.data
    STR1 DB 10, 13, "Siksha 'A' Anusandhan, ITER$"        ;Source String Terminated by $
    STR2 DB 10, 13, 'Enter a character to be replaced: $';ASCII 10 = Enter, 13 = Carriage Return
    STR3 DB 10, 13, 'Enter new character: $'              ;Used to print in a new Line
    LEN DB $-STR1                                          ;(offset address of $) - (starting address of STR1) = Length of STR1

.code
MAIN PROC
    MOV AX, data                ;Copy base address of .data
    MOV DS, AX                  ;into DS Register

   ;Print STR1:
    LEA DX, STR1                ;Offset Address of 1st letter in STR1 stored into DX
    MOV AH, 09H                 ;Instruction 09h -> Writes String to standard output(console) untill '$' encountered
    INT 21H                     ;Calls the Instruction

    MOV DI, (STR1 + LEN - 1) ;DI points to last Character of STR1
    MOV CL, LEN                 ;Initialize counter into CX = LEN
    MOV CH, 00H                 ;Make high bits of CX 0

    LEA DX, STR2
    MOV AH, 09H                 ;Print STR2
    INT 21H

    MOV AH, 01H                 ;INT 21H/01H: Halts the programm until user enters a character
    INT 21H                     ;and stores character in ASCII in AL
    MOV BL, AL                  ;Copied into BL (Character to be replaced = target)

    LEA DX, STR3
    MOV AH, 09H                 ;Print STR3
    INT 21H

    MOV AH, 01H                 ;INT 21H/01H: Halts the programm until user enters a character
    INT 21H                     ;and stores character in ASCII in AL
    MOV BH, AL                  ;Store copy in BH(Characted to replace with)
    MOV AL, BL                  ;Take back target into AL

COMPARE:
    CMP [DI], AL                ;Compare DI(last character) with target in AL
    JZ XCHEG                    ;If equal go to 'XCHEG'
    JMP CONTINUE                ;Else go to 'CONTINUE'

XCHEG:                          ;Perform replace procedure
    MOV BYTE PTR [DI], BH       ;Copy byte size of BH into offset Address of DI

CONTINUE:
    DEC DI                      ;Decrement DI [point to next letter(reading backwards)]
    LOOP COMPARE                ;Go back to 'COMPARE'

    LEA DX, STR1
    MOV AH, 09H                 ;Print modified STR1
    INT 21H

END MAIN
```

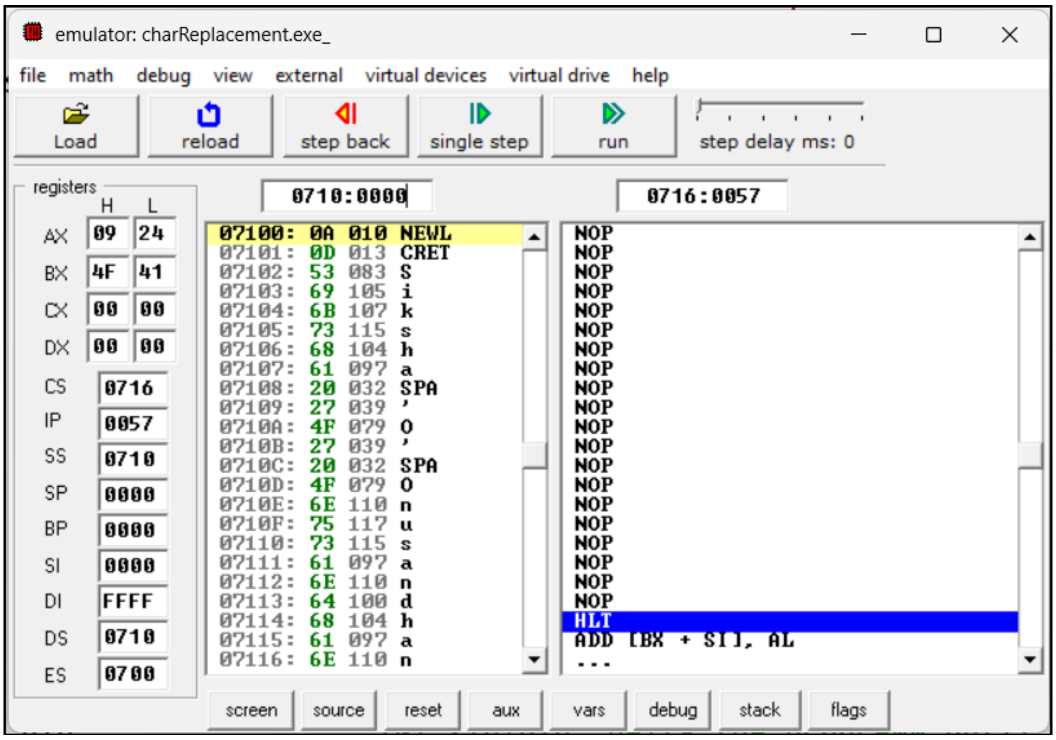
Results & Interpretation

Verification of the output through simulation:



emulator screen (40x7 chars)

```
Siksha 'A' Anusandhan, ITER
Enter a character to be replaced: A
Enter new character: O
Siksha 'O' Onusandhan, ITER
```

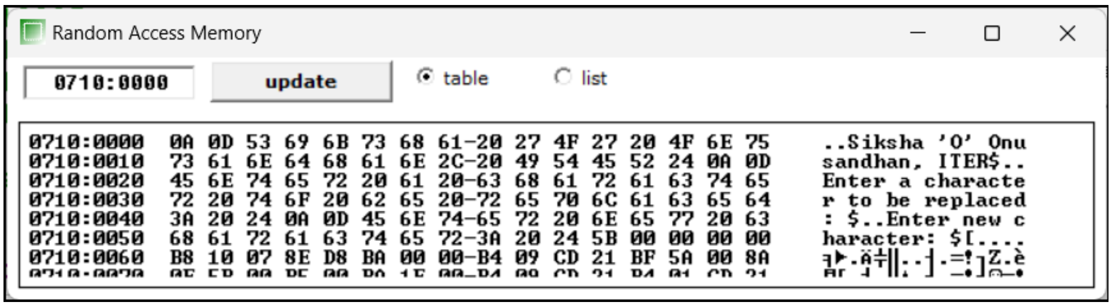


emulator: charReplacement.exe_

```
registers
     H    L
AX   09   24
BX   4F   41
CX   00   00
DX   00   00
CS   0716
IP   0057
SS   0710
SP   0000
BP   0000
SI   0000
DI   FFFF
DS   0710
ES   0700
```

```
0710:0000

07100: 0A 010 NEWL
07101: 0D 013 CRET
07102: 53 083 S
07103: 69 105 i
07104: 6B 107 k
07105: 73 115 s
07106: 68 104 h
07107: 61 097 a
07108: 20 032 SPA
07109: 27 039 '
0710A: 4F 079 O
0710B: 27 039 '
0710C: 20 032 SPA
0710D: 4F 079 O
0710E: 6E 110 n
0710F: 75 117 u
07110: 73 115 s
07111: 61 097 a
07112: 6E 110 n
07113: 64 100 d
07114: 68 104 h
07115: 61 097 a
07116: 6E 110 n
```

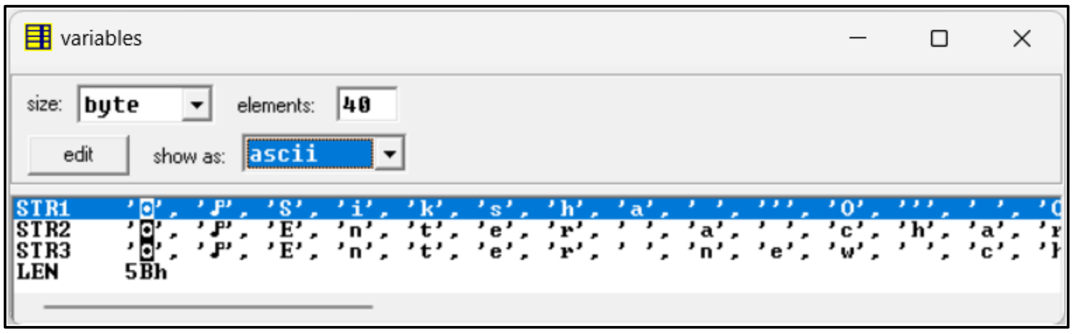
```
0716:0057

NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
HLT
ADD [BX + SI], AL
...
```



Random Access Memory

0710:0000 — update — [✓ table] list

```
0710:0000   0A 0D 53 69 6B 73 68 61-20 27 4F 27 20 4F 6E 75    ..Siksha 'O' Onu
0710:0010   73 61 6E 64 68 61 6E 2C-20 49 54 45 52 24 0A 0D    sandhan, ITER$..
0710:0020   45 6E 74 65 72 20 61 20-63 68 61 72 61 63 74 65    Enter a characte
0710:0030   72 20 74 6F 20 62 65 20-72 65 70 6C 61 63 65 64    r to be replaced
0710:0040   3A 20 24 0A 0D 45 6E 74-65 72 20 6E 65 77 20 63    : $..Enter new c
0710:0050   68 61 72 61 63 74 65 72-3A 20 24 5B 00 00 00 00    haracter: $[....
0710:0060   B8 10 07 8E D8 BA 00 00-B4 09 CD 21 BF 5A 00 8A    ╕▶.Ä╪║..┤.═!┐Z.è
```



variables

size: byte   elements: 40   show as: ascii

```
STR1    '◙', '♪', 'S', 'i', 'k', 's', 'h', 'a', ' ', ''', 'O', ''', ' ', 'O
STR2    '◙', '♪', 'E', 'n', 't', 'e', 'r', ' ', 'a', ' ', 'c', 'h', 'a', 'r
STR3    '◙', '♪', 'E', 'n', 't', 'e', 'r', ' ', 'n', 'e', 'w', ' ', 'c', 'h
LEN     5Bh
```

Conclusion

Culminating the implementation phase, our system effectively achieves the objective of character replacement within a string using 8086 assembly language. Through meticulous design and implementation, we demonstrate the practical application of assembly language in data manipulation tasks. The system efficiently handles character replacement, showcasing the versatility and effectiveness of assembly language programming in addressing real-world challenges.

References

[1] "AH=09H - 8086 INT 21H" Wikidev. https://wikidev.in/wiki/assembly/8086_INT_21H/AH09H (accessed May. 08, 2024).

[2] "EMU8086 - MICROPROCESSOR EMULATOR" Softonic. https://emu8086-microprocessor-emulator.en.softonic.com/? (accessed May. 05, 2024).

[3] W. Stallings, Computer Organization and Architecture, Designing for Performance 10th Edition

[4, Appendix] "8086 Datasheet (PDF) – Intel Corporation" ALLDATASHEET.COM, https://www.alldatasheet.com/datasheet-pdf/pdf/1154707/INTEL/8086.html (accessed May. 08, 2024).

intel®

8086 16-BIT HMOS MICROPROCESSOR 8086/8086-2/8086-1

- Direct Addressing Capability 1 MByte of Memory
- Architecture Designed for Powerful Assembly Language and Efficient High Level Languages
- 14 Word, by 16-Bit Register Set with Symmetrical Operations
- 24 Operand Addressing Modes
- Bit, Byte, Word, and Block Operations
- 8 and 16-Bit Signed and Unsigned Arithmetic in Binary or Decimal Including Multiply and Divide
- Range of Clock Rates:
  5 MHz for 8086,
  8 MHz for 8086-2,
  10 MHz for 8086-1
- MULTIBUS System Compatible Interface
- Available in EXPRESS
  — Standard Temperature Range
  — Extended Temperature Range
- Available in 40-Lead Cerdip and Plastic Package
  (See Packaging Spec. Order > 231369)

The Intel 8086 high performance 16-bit CPU is available in three clock rates: 5, 8 and 10 MHz. The CPU is implemented in N-Channel, depletion load, silicon gate technology (HMOS-III), and packaged in a 40-pin CERDIP or plastic package. The 8086 operates in both single processor and multiple processor configurations to achieve high performance levels.

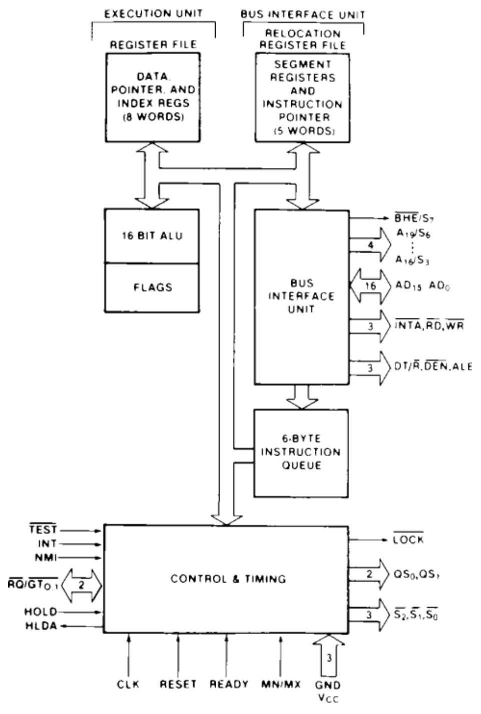
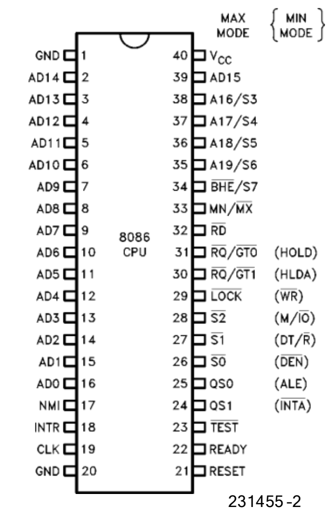


Figure 1. 8086 CPU Block Diagram

231455-1



231455-2

40 Lead

Figure 2. 8086 Pin Configuration

Table 1. Pin Description

The following pin function descriptions are for 8086 systems in either minimum or maximum mode. The "Local Bus" in these descriptions is the direct multiplexed bus interface connection to the 8086 (without regard to additional bus buffers).

Symbol	Pin No.	Type	Name and Function
AD_{15}-AD_0	2-16, 39	I/O	ADDRESS DATA BUS: These lines constitute the time multiplexed memory/IO address (T_1), and data (T_2, T_3, T_W, T_4) bus. A_0 is analogous to BHE for the lower byte of the data bus, pins D_7-D_0. It is LOW during T_1 when a byte is to be transferred on the lower portion of the bus in memory or I/O operations. Eight-bit oriented devices tied to the lower half would normally use A_0 to condition chip select functions. (See $\overline{BHE}$.) These lines are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge".
A_{19}/S_6, A_{18}/S_5, A_{17}/S_4, A_{16}/S_3	35-38	O	ADDRESS/STATUS: During T_1 these are the four most significant address lines for memory operations. During I/O operations these lines are LOW. During memory and I/O operations, status information is available on these lines during T_2, T_3, T_W, T_4. The status of the interrupt enable FLAG bit (S_5) is updated at the beginning of each CLK cycle. A_{17}/S_4 and A_{16}/S_3 are encoded as shown. This information indicates which relocation register is presently being used for data accessing. These lines float to 3-state OFF during local bus "hold acknowledge."

A_{17}/S_4	A_{16}/S_3	Characteristics
0 (LOW)	0	Alternate Data
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data
S_6 is 0 (LOW)		

Symbol	Pin No.	Type	Name and Function
$\overline{BHE}/S_7$	34	O	BUS HIGH ENABLE/STATUS: During T_1 the bus high enable signal (BHE) should be used to enable data onto the most significant half of the data bus, pins D_{15}-D_8. Eight-bit oriented devices tied to the upper half of the bus would normally use BHE to condition chip select functions. BHE is LOW during T_1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus. The S_7 status information is available during T_2, T_3, and T_4. The signal is active LOW, and floats to 3-state OFF in "hold". It is LOW during T_1 for the first interrupt acknowledge cycle.

$\overline{BHE}$	A_0	Characteristics
0	0	Whole word
0	1	Upper byte from/to odd address
1	0	Lower byte from/to even address
1	1	None

Symbol	Pin No.	Type	Name and Function
$\overline{RD}$	32	O	READ: Read strobe indicates that the processor is performing a memory or I/O read cycle, depending on the state of the S_2 pin. This signal is used to read devices which reside on the 8086 local bus. RD is active LOW during T_2, T_3 and T_W of any read cycle, and is guaranteed to remain HIGH in T_2 until the 8086 local bus has floated. This signal floats to 3-state OFF in "hold acknowledge".

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
READY	22	I	READY: is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The READY signal from memory/IO is synchronized by the 8284A Clock Generator to form READY. This signal is active HIGH. The 8086 READY input is not synchronized. Correct operation is not guaranteed if the setup and hold times are not met.
INTR	18	I	INTERRUPT REQUEST: is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.
$\overline{\text{TEST}}$	23	I	$\overline{\text{TEST}}$: input is examined by the "Wait" instruction. If the $\overline{\text{TEST}}$ input is LOW execution continues, otherwise the processor waits in an "Idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.
NMI	17	I	NON-MASKABLE INTERRUPT: an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.
RESET	21	I	RESET: causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the Instruction Set description, when RESET returns LOW. RESET is internally synchronized.
CLK	19	I	CLOCK: provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V_{CC}	40		V_{CC}: a5V power supply pin.
GND	1, 20		GROUND
MN/$\overline{\text{MX}}$	33	I	MINIMUM/MAXIMUM: indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

The following pin function descriptions are for the 8086/8288 system in maximum mode (i.e., MN/$\overline{\text{MX}}$ e V_{SS}). Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

$\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$	26-28	O	STATUS: active during T_4, T_1, and T_2 and is returned to the passive state (1, 1, 1) during T_3 or during T_W when READY is HIGH. This status is used by the 8288 Bus Controller to generate all memory and I/O access control signals. Any change by $\overline{S_2}$, $\overline{S_1}$, or $\overline{S_0}$ during T_4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T_3 or T_W is used to indicate the end of a bus cycle.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$ (Continued)	26–28	O	These signals float to 3-state OFF in "hold acknowledge". These status lines are encoded as shown.

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Characteristics
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1 (HIGH)	0	0	Code Access
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

Symbol	Pin No.	Type	Name and Function
$\overline{RQ}/\overline{GT}_0$, $\overline{RQ}/\overline{GT}_1$	30, 31	I/O	REQUEST/GRANT: pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with RQ/GT_0 having higher priority than RQ/GT_1. RQ/GT pins have internal pull-up resistors and may be left unconnected. The request/grant sequence is as follows (see Page 2-24): 1. A pulse of 1 CLK wide from another local bus master indicates a local bus request ("hold") to the 8086 (pulse 1). 2. During a T_4 or T_1 clock cycle, a pulse 1 CLK wide from the 8086 to the requesting master (pulse 2), indicates that the 8086 has allowed the local bus to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge". 3. A pulse 1 CLK wide from the requesting master indicates to the 8086 (pulse 3) that the "hold" request is about to end and that the 8086 can reclaim the local bus at the next CLK. Each master-master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active LOW. If the request is made while the CPU is performing a memory cycle, it will release the local bus during T_4 of the cycle when all the following conditions are met: 1. Request occurs on or before T_2. 2. Current cycle is not the low byte of a word (on an odd address). 3. Current cycle is not the first acknowledge of an interrupt acknowledge sequence. 4. A locked instruction is not currently executing. If the local bus is idle when the request is made the two possible events will follow: 1. Local bus will be released during the next clock. 2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied.
$\overline{LOCK}$	29	O	$\overline{LOCK}$: output indicates that other system bus masters are not to gain control of the system bus while LOCK is active LOW. The LOCK signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state OFF in "hold acknowledge".

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
QS_1, QS_0	24, 25	O	QUEUE STATUS: The queue status is valid during the CLK cycle after which the queue operation is performed. QS_1 and QS_0 provide status to allow external tracking of the internal 8086 instruction queue.

QS_1	QS_0	Characteristics
0 (LOW)	0	No Operation
0	1	First Byte of Op Code from Queue
1 (HIGH)	0	Empty the Queue
1	1	Subsequent Byte from Queue

The following pin function descriptions are for the 8086 in minimum mode (i.e., MN/$\overline{MX}$ e V_{CC}). Only the pin functions which are unique to minimum mode are described; all other pin functions are as described above.

M/$\overline{IO}$	28	O	STATUS LINE: logically equivalent to S_2 in the maximum mode. It is used to distinguish a memory access from an I/O access. M/IO becomes valid in the T_4 preceding a bus cycle and remains valid until the final T_4 of the cycle (M e HIGH, IO e LOW). M/IO floats to 3-state OFF in local bus "hold acknowledge".
$\overline{WR}$	29	O	WRITE: indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the M/IO signal. WR is active for T_2, T_3 and T_W of any write cycle. It is active LOW, and floats to 3-state OFF in local bus "hold acknowledge".
$\overline{INTA}$	24	O	$\overline{INTA}$: is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T_2, T_3 and T_W of each interrupt acknowledge cycle.
ALE	25	O	ADDRESS LATCH ENABLE: provided by the processor to latch the address into the 8282/8283 address latch. It is a HIGH pulse active during T_1 of any bus cycle. Note that ALE is never floated.
DT/$\overline{R}$	27	O	DATA TRANSMIT/RECEIVE: needed in minimum system that desires to use an 8286/8287 data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically DT/R is equivalent to S_1 in the maximum mode, and its timing is the same as for M/IO. (T e HIGH, R e LOW.) This signal floats to 3-state OFF in local bus "hold acknowledge".
$\overline{DEN}$	26	O	DATA ENABLE: provided as an output enable for the 8286/8287 in a minimum system which uses the transceiver. DEN is active LOW during each memory and I/O access and for INTA cycles. For a read or INTA cycle it is active from the middle of T_2 until the middle of T_4, while for a write cycle it is active from the beginning of T_2 until the middle of T_4. DEN floats to 3-state OFF in local bus "hold acknowledge".
HOLD, HLDA	31, 30	I/O	HOLD: indicates that another master is requesting a local bus "hold." To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledgement in the middle of a T_4 or T_i clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor will LOWer the HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines. Hold acknowledge (HLDA) and HOLD have internal pull-up resistors. The same rules as for $\overline{RQ}/\overline{GT}$ apply regarding when the local bus will be released. HOLD is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the setup time.

FUNCTIONAL DESCRIPTION

General Operation

The internal functions of the 8086 processor are partitioned logically into two processing units. The first is the Bus Interface Unit (BIU) and the second is the Execution Unit (EU) as shown in the block diagram of Figure 1.

These units can interact directly but for the most part perform as separate asynchronous operational processors. The bus interface unit provides the functions related to instruction fetching and queuing, operand fetch and store, and address relocation. This unit also provides the basic bus control. The overlap of instruction pre-fetching provided by this unit serves to increase processor performance through improved bus bandwidth utilization. Up to 6 bytes of the instruction stream can be queued while waiting for decoding and execution.

The instruction stream queuing mechanism allows the BIU to keep the memory utilized very efficiently. Whenever there is space for at least 2 bytes in the queue, the BIU will attempt a word fetch memory cycle. This greatly reduces "dead time" on the memory bus. The queue acts as a First-In-First-Out (FIFO) buffer, from which the EU extracts instruction bytes as required. If the queue is empty (following a branch instruction, for example), the first byte into the queue immediately becomes available to the EU.

The execution unit receives pre-fetched instructions from the BIU queue and provides un-relocated operand addresses to the BIU. Memory operands are passed through the BIU for processing by the EU, which passes results to the BIU for storage. See the Instruction Set description for further register set and architectural descriptions.

MEMORY ORGANIZATION

The processor provides a 20-bit address to memory which locates the byte being referenced. The memory is organized as a linear array of up to 1 million bytes, addressed as 00000(H) to FFFFF(H). The memory is logically divided into code, data, extra data, and stack segments of up to 64K bytes each, with each segment falling on 16-byte boundaries. (See Figure 3a.)

All memory references are made relative to base addresses contained in high speed segment registers. The segment types were chosen based on the addressing needs of programs. The segment register to be selected is automatically chosen according to the rules of the following table. All information in one segment type share the same logical attributes (e.g. code or data). By structuring memory into relocatable areas of similar characteristics and by automatically selecting segment registers, programs are shorter, faster, and more structured.

Word (16-bit) operands can be located on even or odd address boundaries and are thus not constrained to even boundaries as is the case in many 16-bit computers. For address and data operands, the least significant byte of the word is stored in the lower valued address location and the most significant byte in the next higher address location. The BIU automatically performs the proper number of memory accesses, one if the word operand is on an even byte boundary and two if it is on an odd byte boundary. Except for the performance penalty, this double access is transparent to the software. This performance penalty does not occur for instruction fetches, only word operands.

Physically, the memory is organized as a high bank (D_{15} -D_8) and a low bank (D_7 -D_0) of 512K 8-bit bytes addressed in parallel by the processor's address lines A_{19}-A_1. Byte data with even addresses is transferred on the D_7 -D_0 bus lines while odd addressed byte data (A_0 HIGH) is transferred on the D_{15} -D_8 bus lines. The processor provides two enable signals, $\overline{BHE}$ and A_0, to selectively allow reading from or writing into either an odd byte location, even byte location, or both. The instruction stream is fetched from memory as words and is addressed internally by the processor to the byte level as necessary.

Memory Reference Need	Segment Register Used	Segment Selection Rule
Instructions	CODE (CS)	Automatic with all instruction prefetch.
Stack	STACK (SS)	All stack pushes and pops. Memory references relative to BP base register except data references.
Local Data	DATA (DS)	Data references when: relative to stack, destination of string operation, or explicitly overridden.
External (Global) Data	EXTRA (ES)	Destination of string operations: explicitly selected using a segment override.

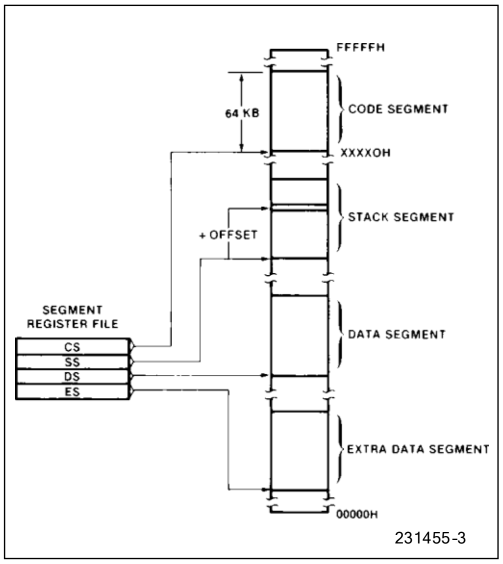


Figure 3a. Memory Organization

In referencing word data the BIU requires one or two memory cycles depending on whether or not the starting byte of the word is on an even or odd address, respectively. Consequently, in referencing word operands performance can be optimized by locating data on even address boundaries. This is an especially useful technique for using the stack, since odd address references to the stack may adversely affect the context switching time for interrupt processing or task multiplexing.

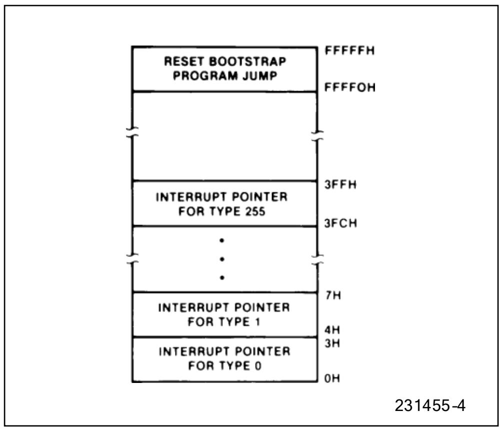

RESET BOOTSTRAP PROGRAM JUMP
FFFFFH
FFFF0H
3FFH
INTERRUPT POINTER FOR TYPE 255
3FCH
7H
INTERRUPT POINTER FOR TYPE 1
4H
3H
INTERRUPT POINTER FOR TYPE 0
0H
231455-4

Figure 3b. Reserved Memory Locations

Certain locations in memory are reserved for specific CPU operations (see Figure 3b). Locations from address FFFF0H through FFFFFH are reserved for operations including a jump to the initial program loading routine. Following RESET, the CPU will always begin execution at location FFFF0H where the jump must be. Locations 00000H through 003FFH are reserved for interrupt operations. Each of the 256 possible interrupt types has its service routine pointed to by a 4-byte pointer element consisting of a 16-bit segment address and a 16-bit offset address. The pointer elements are assumed to have been stored at the respective places in reserved memory prior to occurrence of interrupts.

MINIMUM AND MAXIMUM MODES

The requirements for supporting minimum and maximum 8086 systems are sufficiently different that they cannot be done efficiently with 40 uniquely defined pins. Consequently, the 8086 is equipped with a strap pin (MN/MX) which defines the system configuration. The definition of a certain subset of the pins changes dependent on the condition of the strap pin. When MN/MX pin is strapped to GND, the 8086 treats pins 24 through 31 in maximum mode. An 8288 bus controller interprets status information coded into S_0, S_2, S_2 to generate bus timing and control signals compatible with the MULTIBUS architecture. When the MN/MX pin is strapped to V_{CC}, the 8086 generates bus control signals itself on pins 24 through 31, as shown in parentheses in Figure 2. Examples of minimum mode and maximum mode systems are shown in Figure 4.

BUS OPERATION

The 8086 has a combined address and data bus commonly referred to as a time multiplexed bus. This technique provides the most efficient use of pins on the processor while permitting the use of a standard 40-lead package. This "local bus" can be buffered directly and used throughout the system with address latching provided on memory and I/O modules. In addition, the bus can also be demultiplexed at the processor with a single set of address latches if a standard non-multiplexed bus is desired for the system.

Each processor bus cycle consists of at least four CLK cycles. These are referred to as T_1, T_2, T_3 and T_4 (see Figure 5). The address is emitted from the processor during T_1 and data transfer occurs on the bus during T_3 and T_4. T_2 is used primarily for changing the direction of the bus during read operations. In the event that a "NOT READY" indication is given by the addressed device, "Wait" states (T_W) are inserted between T_3 and T_4. Each inserted "Wait" state is of the same duration as a CLK cycle. Periods

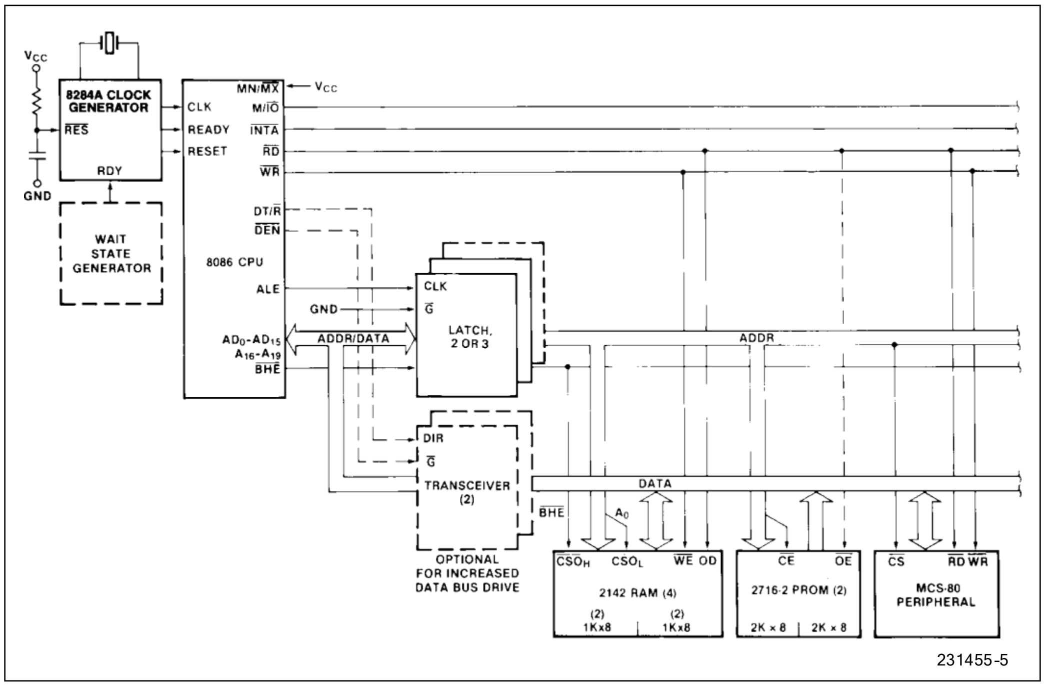


Figure 4a. Minimum Mode 8086 Typical Configuration

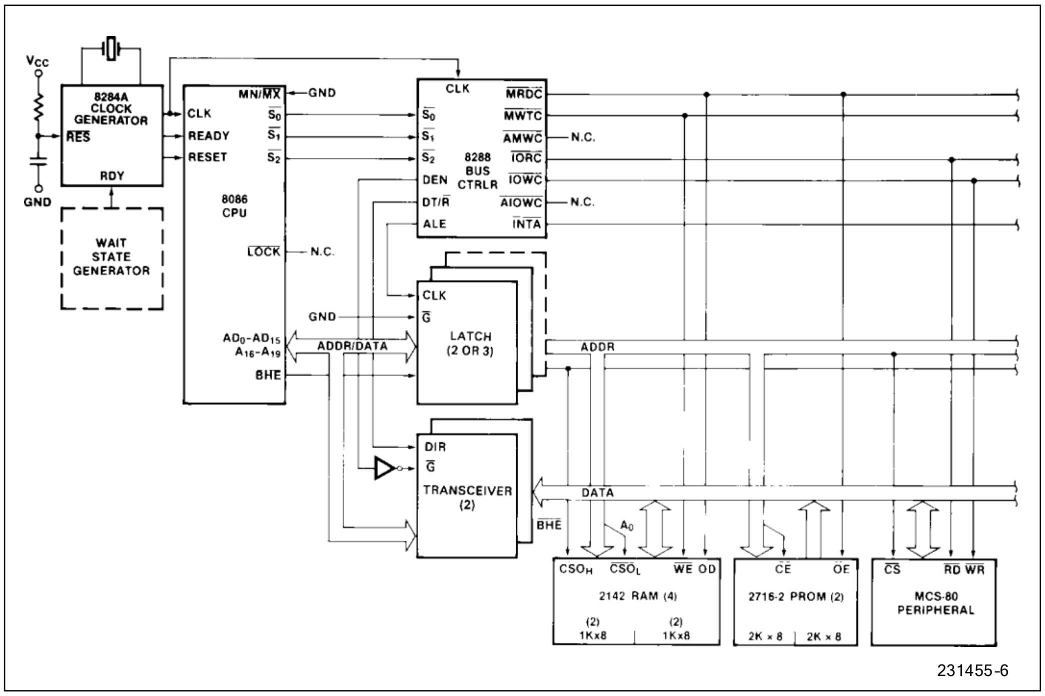


Figure 4b. Maximum Mode 8086 Typical Configuration

can occur between 8086 bus cycles. These are referred to as "Idle" states (T_i) or inactive CLK cycles. The processor uses these cycles for internal housekeeping.

During T_1 of any bus cycle the ALE (Address Latch Enable) signal is emitted (by either the processor or the 8288 bus controller, depending on the MN/$\overline{MX}$ strap). At the trailing edge of this pulse, a valid address and certain status information for the cycle may be latched.

Status bits $\overline{S_0}$, $\overline{S_1}$, and $\overline{S_2}$ are used, in maximum mode, by the bus controller to identify the type of bus transaction according to the following table:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Characteristics
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1 (HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

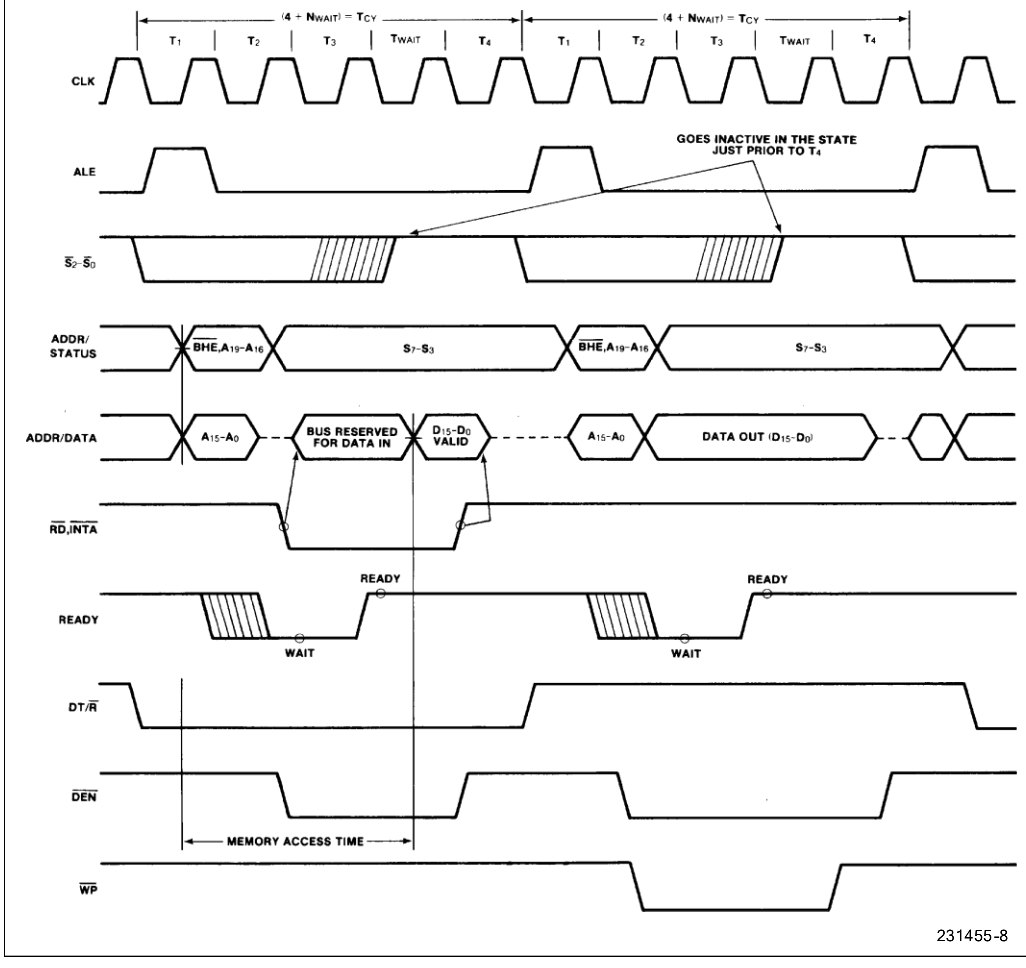


Figure 5. Basic System Timing

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ¿¿¿¿¿¿0ßC to 70ßC
Storage Temperature ¿¿¿¿¿¿¿¿¿¿b65ßC to a150ßC
Voltage on Any Pin with
Respect to Ground¿¿¿¿¿¿¿¿¿¿¿¿¿¿b1.0V to a7V
Power Dissipation¿¿¿¿¿¿¿¿¿¿¿¿¿¿¿¿¿¿¿¿¿¿¿¿¿2.5W

NOTICE: This is a production data sheet. The specifications are subject to change without notice.

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

D.C. CHARACTERISTICS

(8086: T_A e 0ßC to 70ßC, V_{CC} e 5V g10%)
(8086-1: T_A e 0ßC to 70ßC, V_{CC} e 5V g5%)
(8086-2: T_A e 0ßC to 70ßC, V_{CC} e 5V g5%)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	b0.5	a0.8	V	(Note 1)
V_{IH}	Input High Voltage	2.0	V_{CC} a 0.5	V	(Notes 1, 2)
V_{OL}	Output Low Voltage		0.45	V	I_{OL} e 2.5 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} e b 400 mA
I_{CC}	Power Supply Current: 8086 8086-1 8086-2		340 360 350	mA	T_A e 25ßC
I_{LI}	Input Leakage Current		g10	mA	0V s V_{IN} s V_{CC} (Note 3)
I_{LO}	Output Leakage Current		g10	mA	0.45V s V_{OUT} s V_{CC}
V_{CL}	Clock Input Low Voltage	b0.5	a0.6	V	
V_{CH}	Clock Input High Voltage	3.9	V_{CC} a 1.0	V	
C_{IN}	Capacitance of Input Buffer (All input except AD_0-AD_{15}, $\overline{RQ}/\overline{GT}$)		15	pF	fc e 1 MHz
C_{IO}	Capacitance of I/O Buffer (AD_0-AD_{15}, $\overline{RQ}/\overline{GT}$)		15	pF	fc e 1 MHz

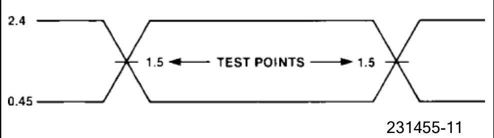
NOTES:
1. V_{IL} tested with MN/$\overline{MX}$ Pin e 0V. V_{IH} tested with MN/$\overline{MX}$ Pin e 5V. MN/$\overline{MX}$ Pin is a Strap Pin.
2. Not applicable to $\overline{RQ}$/GT0 and $\overline{RQ}/\overline{GT1}$ (Pins 30 and 31).
3. HOLD and HLDA I_{LI} min e 30 mA, max e 500 mA.

A.C. CHARACTERISTICS (8086: T_A e 0ßC to 70ßC, V_{CC} e 5V g 10%)
(8086-1: T_A e 0ßC to 70ßC, V_{CC} e 5V g 5%)
(8086-2: T_A e 0ßC to 70ßC, V_{CC} e 5V g 5%)

MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS

Symbol	Parameter	8086		8086-1		8086-2		Units	Test Conditions
		Min	Max	Min	Max	Min	Max		
TCLCL	CLK Cycle Period	200	500	100	500	125	500	ns	
TCLCH	CLK Low Time	118		53		68		ns	
TCHCL	CLK High Time	69		39		44		ns	
TCH1CH2	CLK Rise Time		10		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		5		20		ns	
TCLDX	Data in Hold Time	10		10		10		ns	
TR1VCL	RDY Setup Time into 8284A (SeeNotes 1, 2)	35		35		35		ns	
TCLR1X	RDY Hold Time into 8284A (See Notes 1, 2)	0		0		0		ns	
TRYHCH	READY Setup Time into 8086	118		53		68		ns	
TCHRYX	READY Hold Time into 8086	30		20		20		ns	
TRYLCL	READY Inactive to CLK (See Note 3)	b8		b10		b8		ns	
THVCH	HOLD Setup Time	35		20		20		ns	
TINVCH	INTR, NMI, $\overline{\text{TEST}}$ Setup Time (See Note 2)	30		15		15		ns	
TILIH	Input Rise Time (Except CLK)		20		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12		12	ns	From 2.0V to 0.8V

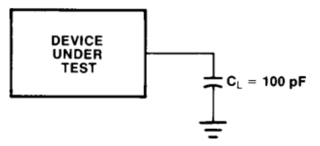
A.C. TESTING INPUT, OUTPUT WAVEFORM



231455-11

A.C. Testing: Inputs are driven at 2.4V for a Logic "1" and 0.45V for a Logic "0". Timing measurements are made at 1.5V for both a Logic "1" and "0".

A.C. TESTING LOAD CIRCUIT

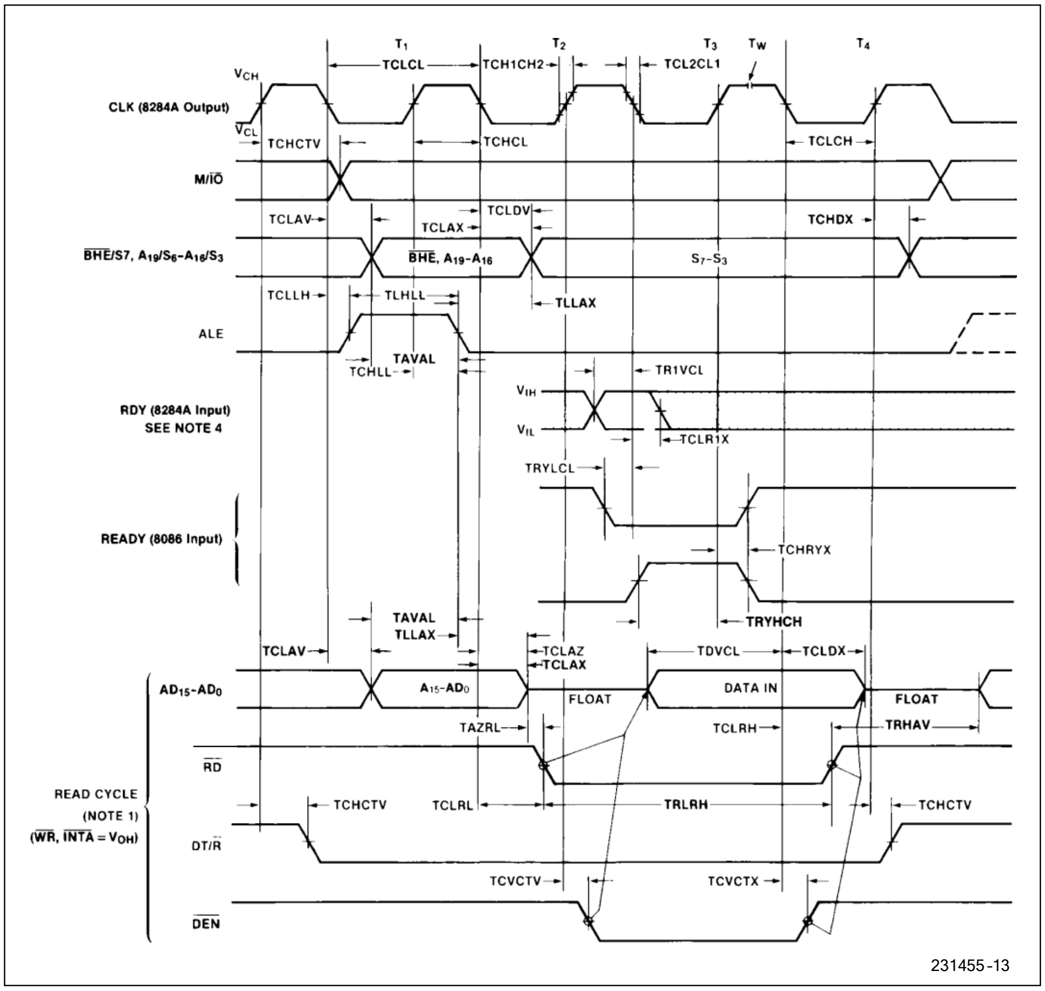


231455-12

C_L Includes Jig Capacitance

WAVEFORMS

MINIMUM MODE



NOTES:
1. All signals switch between V_{OH} and V_{OL} unless otherwise specified.
2. RDY is sampled near the end of T_2, T_3, T_W to determine if T_W machines states are to be inserted.
3. Two INTA cycles run back-to-back. The 8086 LOCAL ADDR/DATA BUS is floating during both INTA cycles. Control signals shown for second INTA cycle.
4. Signals at 8284A are shown for reference only.

All timing measurements are made at 1.5V unless otherwise noted.

Table 2. Instruction Set Summary

Mnemonic and Description	Instruction Code			
DATA TRANSFER				
MOV e Move:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Register/Memory to/from Register	1 0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w e 1
Immediate to Register	1 0 1 1 w reg	data	data if w e 1	
Memory to Accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high	
Accumulator to Memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/Memory to Segment Register	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment Register to Register/Memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		
PUSH e Push:				
Register/Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m		
Register	0 1 0 1 0 reg			
Segment Register	0 0 0 reg 1 1 0			
POP e Pop:				
Register/Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m		
Register	0 1 0 1 1 reg			
Segment Register	0 0 0 reg 1 1 1			
XCHG e Exchange:				
Register/Memory with Register	1 0 0 0 0 1 1 w	mod reg r/m		
Register with Accumulator	1 0 0 1 0 reg			
IN e Input from:				
Fixed Port	1 1 1 0 0 1 0 w	port		
Variable Port	1 1 1 0 1 1 0 w			
OUT e Output to:				
Fixed Port	1 1 1 0 0 1 1 w	port		
Variable Port	1 1 1 0 1 1 1 w			
XLAT e Translate Byte to AL	1 1 0 1 0 1 1 1			
LEA e Load EA to Register	1 0 0 0 1 1 0 1	mod reg r/m		
LDS e Load Pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m		
LES e Load Pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m		
LAHF e Load AH with Flags	1 0 0 1 1 1 1 1			
SAHF e Store AH into Flags	1 0 0 1 1 1 1 0			
PUSHF e Push Flags	1 0 0 1 1 1 0 0			
POPF e Pop Flags	1 0 0 1 1 1 0 1			

Mnemonics © Intel, 1978

Table 2. Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code			
ARITHMETIC	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
ADD e Add:				
Reg./Memory with Register to Either	0 0 0 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s: w e 01
Immediate to Accumulator	0 0 0 0 0 1 0 w	data	data if w e 1	
ADC e Add with Carry:				
Reg./Memory with Register to Either	0 0 0 1 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s: w e 01
Immediate to Accumulator	0 0 0 1 0 1 0 w	data	data if w e 1	
INC e Increment:				
Register/Memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m		
Register	0 1 0 0 0 reg			
AAA e ASCII Adjust for Add	0 0 1 1 0 1 1 1			
BAA e Decimal Adjust for Add	0 0 1 0 0 1 1 1			
SUB e Subtract:				
Reg./Memory and Register to Either	0 0 1 0 1 0 d w	mod reg r/m		
Immediate from Register/Memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s w e 01
Immediate from Accumulator	0 0 1 0 1 1 0 w	data	data if w e 1	
SSB e Subtract with Borrow				
Reg./Memory and Register to Either	0 0 0 1 1 0 d w	mod reg r/m		
Immediate from Register/Memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s w e 01
Immediate from Accumulator	0 0 0 1 1 1 w	data	data if w e 1	
DEC e Decrement:				
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m		
Register	0 1 0 0 1 reg			
NEG e Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m		
CMP e Compare:				
Register/Memory and Register	0 0 1 1 1 0 d w	mod reg r/m		
Immediate with Register/Memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s w e 01
Immediate with Accumulator	0 0 1 1 1 1 0 w	data	data if w e 1	
AAS e ASCII Adjust for Subtract	0 0 1 1 1 1 1 1			
DAS e Decimal Adjust for Subtract	0 0 1 0 1 1 1 1			
MUL e Multiply (Unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m		
IMUL e Integer Multiply (Signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m		
AAM e ASCII Adjust for Multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0		
DIV e Divide (Unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m		
IDIV e Integer Divide (Signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m		
AAD e ASCII Adjust for Divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0		
CBW e Convert Byte to Word	1 0 0 1 1 0 0 0			
CWD e Convert Word to Double Word	1 0 0 1 1 0 0 1			

Mnemonics © Intel, 1978

Table 2. Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code			
LOGIC	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
NOT e Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m		
SHL/SAL e Shift Logical/Arithmetic Left	1 1 0 1 0 0 v w	mod 1 0 0 r/m		
SHR e Shift Logical Right	1 1 0 1 0 0 v w	mod 1 0 1 r/m		
SAR e Shift Arithmetic Right	1 1 0 1 0 0 v w	mod 1 1 1 r/m		
ROL e Rotate Left	1 1 0 1 0 0 v w	mod 0 0 0 r/m		
ROR e Rotate Right	1 1 0 1 0 0 v w	mod 0 0 1 r/m		
RCL e Rotate Through Carry Flag Left	1 1 0 1 0 0 v w	mod 0 1 0 r/m		
RCR e Rotate Through Carry Right	1 1 0 1 0 0 v w	mod 0 1 1 r/m		
AND e And:				
Reg./Memory and Register to Either	0 0 1 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w e 1
Immediate to Accumulator	0 0 1 0 0 1 0 w	data	data if w e 1	
TEST e And Function to Flags, No Result:				
Register/Memory and Register	1 0 0 0 0 1 0 w	mod reg r/m		
Immediate Data and Register/Memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w e 1
Immediate Data and Accumulator	1 0 1 0 1 0 0 w	data	data if w e 1	
OR e Or:				
Reg./Memory and Register to Either	0 0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w e 1
Immediate to Accumulator	0 0 0 0 1 1 0 w	data	data if w e 1	
XOR e Exclusive or:				
Reg./Memory and Register to Either	0 0 1 1 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w e 1
Immediate to Accumulator	0 0 1 1 0 1 0 w	data	data if w e 1	
STRING MANIPULATION				
REP e Repeat	1 1 1 1 0 0 1 z			
MOVS e Move Byte/Word	1 0 1 0 0 1 0 w			
CMPS e Compare Byte/Word	1 0 1 0 0 1 1 w			
SCAS e Scan Byte/Word	1 0 1 0 1 1 1 w			
LODS e Load Byte/Wd to AL/AX	1 0 1 0 1 1 0 w			
STOS e Stor Byte/Wd from AL/A	1 0 1 0 1 0 1 w			
CONTROL TRANSFER				
CALL e Call:				
Direct within Segment	1 1 1 0 1 0 0 0	disp-low	disp-high	
Indirect within Segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m		
Direct Intersegment	1 0 0 1 1 0 1 0	offset-low	offset-high	
		seg-low	seg-high	
Indirect Intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m		

Mnemonics © Intel, 1978

Table 2. Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code		
JMP e Unconditional Jump:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Direct within Segment	1 1 1 0 1 0 0 1	disp-low	disp-high
Direct within Segment-Short	1 1 1 0 1 0 1 1	disp	
Indirect within Segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m	
Direct Intersegment	1 1 1 0 1 0 1 0	offset-low	offset-high
		seg-low	seg-high
Indirect Intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	
RET e Return from CALL:			
Within Segment	1 1 0 0 0 0 1 1		
Within Seg Adding Immed to SP	1 1 0 0 0 0 1 0	data-low	data-high
Intersegment	1 1 0 0 1 0 1 1		
Intersegment Adding Immediate to SP	1 1 0 0 1 0 1 0	data-low	data-high
JE/JZ e Jump on Equal/Zero	0 1 1 1 0 1 0 0	disp	
JL/JNGE e Jump on Less/Not Greater or Equal	0 1 1 1 1 1 0 0	disp	
JLE/JNG e Jump on Less or Equal/ Not Greater	0 1 1 1 1 1 1 0	disp	
JB/JNAE e Jump on Below/Not Above or Equal	0 1 1 1 0 0 1 0	disp	
JBE/JNA e Jump on Below or Equal/ Not Above	0 1 1 1 0 1 1 0	disp	
JP/JPE e Jump on Parity/Parity Even	0 1 1 1 1 0 1 0	disp	
JO e Jump on Overflow	0 1 1 1 0 0 0 0	disp	
JS e Jump on Sign	0 1 1 1 1 0 0 0	disp	
JNE/JNZ e Jump on Not Equal/Not Zero	0 1 1 1 0 1 0 1	disp	
JNL/JGE e Jump on Not Less/Greater or Equal	0 1 1 1 1 1 0 1	disp	
JNLE/JG e Jump on Not Less or Equal/ Greater	0 1 1 1 1 1 1 1	disp	
JNB/JAE e Jump on Not Below/Above or Equal	0 1 1 1 0 0 1 1	disp	
JNBE/JA e Jump on Not Below or Equal/Above	0 1 1 1 0 1 1 1	disp	
JNP/JPO e Jump on Not Par/Par Odd	0 1 1 1 1 0 1 1	disp	
JNO e Jump on Not Overflow	0 1 1 1 0 0 0 1	disp	
JNS e Jump on Not Sign	0 1 1 1 1 0 0 1	disp	
LOOP e Loop CX Times	1 1 1 0 0 0 1 0	disp	
LOOPZ/LOOPE e Loop While Zero/Equal	1 1 1 0 0 0 0 1	disp	
LOOPNZ/LOOPNE e Loop While Not Zero/Equal	1 1 1 0 0 0 0 0	disp	
JCXZ e Jump on CX Zero	1 1 1 0 0 0 1 1	disp	
INT e Interrupt			
Type Specified	1 1 0 0 1 1 0 1	type	
Type 3	1 1 0 0 1 1 0 0		
INTO e Interrupt on Overflow	1 1 0 0 1 1 1 0		
IRET e Interrupt Return	1 1 0 0 1 1 1 1		

Table 2. Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code	
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
PROCESSOR CONTROL		
CLC e Clear Carry	1 1 1 1 1 0 0 0	
CMC e Complement Carry	1 1 1 1 0 1 0 1	
STC e Set Carry	1 1 1 1 1 0 0 1	
CLD e Clear Direction	1 1 1 1 1 1 0 0	
STD e Set Direction	1 1 1 1 1 1 0 1	
CLI e Clear Interrupt	1 1 1 1 1 0 1 0	
STI e Set Interrupt	1 1 1 1 1 0 1 1	
HLT e Halt	1 1 1 1 0 1 0 0	
WAIT e Wait	1 0 0 1 1 0 1 1	
ESC e Escape (to External Device)	1 1 0 1 1 x x x	mod x x x r/m
LOCK e Bus Lock Prefix	1 1 1 1 0 0 0 0	

NOTES:
AL e 8-bit accumulator
AX e 16-bit accumulator
CX e Count register
DS e Data segment
ES e Extra segment
Above/below refers to unsigned value
Greater e more positive;
Less e less positive (more negative) signed values
if d e 1 then "to" reg; if d e 0 then "from" reg
if w e 1 then word instruction; if w e 0 then byte instruction
if mod e 11 then r/m is treated as a REG field
if mod e 00 then DISP e 0*, disp-low and disp-high are absent
if mod e 01 then DISP e disp-low sign-extended to 16 bits, disp-high is absent
if mod e 10 then DISP e disp-high; disp-low
if r/m e 000 then EA e (BX) a (SI) a DISP
if r/m e 001 then EA e (BX) a (DI) a DISP
if r/m e 010 then EA e (BP) a (SI) a DISP
if r/m e 011 then EA e (BP) a (DI) a DISP
if r/m e 100 then EA e (SI) a DISP
if r/m e 101 then EA e (DI) a DISP
if r/m e 110 then EA e (BP) a DISP*
if r/m e 111 then EA e (BX) a DISP
DISP follows 2nd byte of instruction (before data if required)
*except if mod e 00 and r/m e 110 then EA e disp-high; disp-low.

if s w e 01 then 16 bits of immediate data form the operand
if s w e 11 then an immediate data byte is sign extended to form the 16-bit operand
if v e 0 then "count" e 1; if v e 1 then "count" in (CL)
x e don't care
z is used for string primitives for comparison with ZF FLAG

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

REG is assigned according to the following table:

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

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:
FLAGS e X:X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

DATA SHEET REVISION REVIEW

The following list represents key differences between this and the -004 data sheet. Please review this summary carefully.

1. The Intel 8086 implementation technology (HMOS) has been changed to (HMOS-III).
2. Delete all "changes from 1985 Handbook Specification" sentences.