APPENDIX A

THE HEWLETT-PACKARD (HP) 64000

A.1 System Description

The HP 64000 Microprocessor Logic Development System is a universal development system which provides all of the necessary tools to create, develop, modify, and debug software for microprocessor-based systems. In-circuit emulation provides the capability of performing an in-depth analysis of hardware and software interfacing during the integration phase of the development process.

The HP 64000 Microprocessor Logic Development System is a multi-user development system, allowing up to as many as six users to operate on the system simultaneously. All users of the system share a line printer and a common data base in the form of a 12-megabyte Winchester Technology Disc Drive or a selection of one to eight Multi-Access Controller (MAC) disk drives connected to the system via the HP Interface Bus, commonly referred to as HP-IB. Eight disk drives can provide up to 960 metabytes of HP-formatted storage space.

A.2 Development Station Description

Figure A.1 shows the front view of the HP 64000 Development Station. The keyboard (Figure A.2) is divided into four areas: (1) an ASCII-encoded typewriter-type keyboard; (2) a group of edit keys, which facilitate movement of text or cursor when in the edit mode; (3) special function keys, for system reset or pause or to access a command recall buffer; (4) the all important system "soft keys," eight unobtrusive large key pads just beneath the bezel which surrounds the display.

The soft keys provide a quick and easy means to invoke system commands, virtually eliminating the typographical errors one usually has to contend with when having to enter commands character by character. The definition of each soft key is written on the display just above the bezel. The soft key syntax changes depending on the mode of operation and the

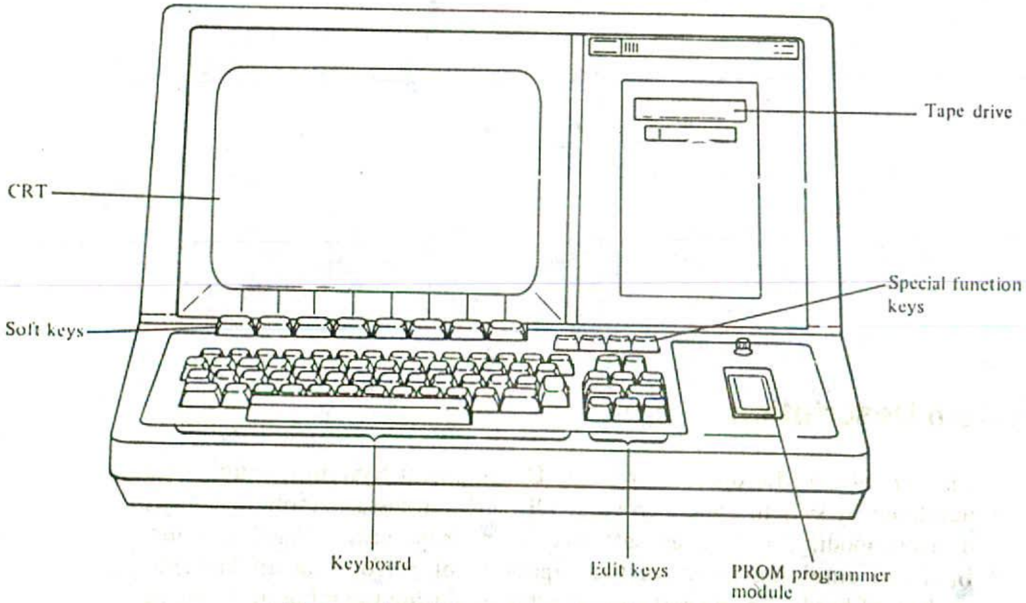


Figure A.1 Front view of the HP 64000 Development Station (Model 64100A). Front Panel: The seven major areas of the front panel are shown. Each area provides the interface necessary to operate and control the system. CRT Display: The CRT is a large-screen, raster-scan magnetic display. Screen capacity is 25 rows and 80 columns of characters. The standard 128-character (upper and lower case) ASCII set can be displayed. A blinking underline cursor is present as the prompt. Video enhancements are inverse video, blinking, and underline. Soft Keys: Just below the CRT are eight unlabeled keys. These keys are defined as the "soft keys." Each key ties to the soft key label line at the bottom of the CRT. During operation, the soft keys are labeled on the CRT screen. Source: Courtesy of Hewlett-Packard.

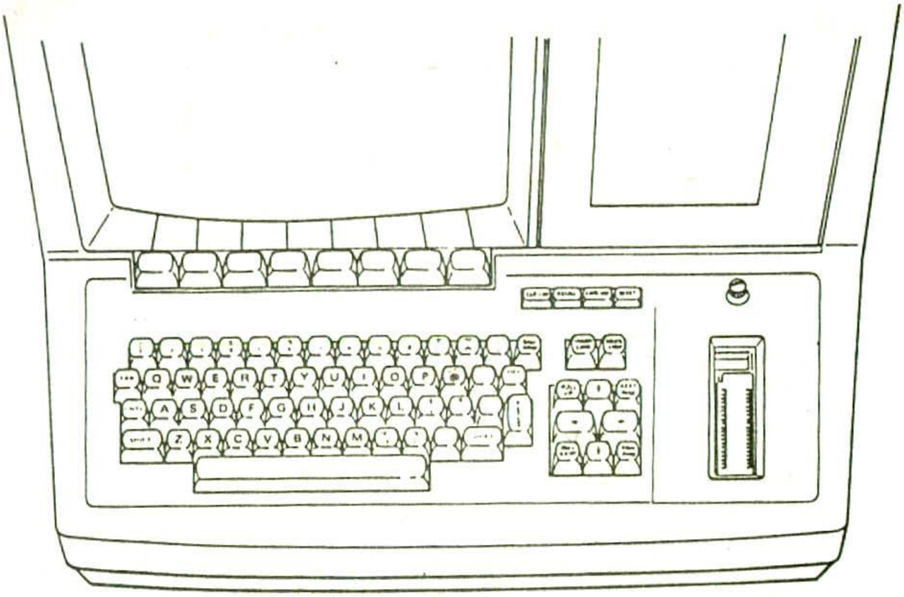


Figure A.2 Model 64100A keyboard. Source: Courtesy of Hewlett-Packard.

position of the cursor. This greatly enhances the ease of use of the system since it provides a list of alternatives available and guides the operator to use the system. In cases where the form of the input required is unknown, brackets surrounding a key word, a syntactical variable will prompt the user with the correct form of input the system expects.

The system display is a Raster Scan CRT which provides a display of 18 lines of text entry, a status line which always displays the system's status and date and time, three lines for command entry, and the soft key label line which indicates the function of each key. The display is 80 columns wide, but with the edit keys the display can be relocated to show text or data out to 240 columns. This is convenient for adding comments and really enhances the program documentation.

Other external station hardware includes RS232 ports for communication with either Data Communications Equipment (DCE) or Data Terminal Equipment (DTE). The RS232 port has a selectable baud rate up to 9,600 and uses the X-ON X-OFF convention for handshaking at baud rates 2,400 and above. There is a 20-mA current loop for TTY interfacing and two ports for triggering of external devices such as an oscilloscope during a logic trace. As system options, the front panel hosts a PROM programmer (Figure A.1) to the immediate right of the keyboard and a tape drive for file back-up. The tape drive performs a high-speed read and write and each cassette holds 250K bytes of data.

Figure A.3 shows special function keys and Table A.1 summarizes their functions.

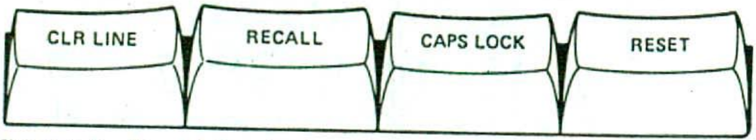


Figure A.3 Special function keys. Source: Courtesy of Hewlett-Packard.

Table A.1 Summary of Special Functions Keys

CLR LINE	Press to clear the current line containing cursor on the CRT.
RECALL	Used to recall, to the command line, previous commands from a stack. The commands are displayed one at a time for each time the RECALL key is pressed. The number of recallable commands is variable. Only valid commands are pushed into the stack. If the RECALL key is pressed and the buffer is empty the system responds with "Recall buffer is empty" message.
CAPS LOCK	Used to lock keyboard in all uppercase letters. A message is presented on the CRT indicating "CAPS LOCK on" or "CAPS LOCK off." At the next key stroke, the message is erased, but the mode remains in effect.
RESET	Pressing RESET once initiates a pause in system operation. A flashing "PAUSED" message, in inverse video, is presented on the status line. To continue operation, press any key except RESET. Pressing the RESET key the second time will clear the CRT and return the system to the system monitor.
SHIFT RESET	Holding the SHIFT key down and pressing RESET initiates a complete system reboot. This function should be regarded as a last resort when the system does not respond.
CNTL RESET	Holding CNTL key down and pressing RESET initiates system performance verification.

Source: Courtesy of Hewlett-Packard.

The following summarizes the HP 64000 soft keys, commands, assembler error codes, and other features.

System Monitor Soft Keys

The following provides a description of the system monitor soft keys:

userid

The userid or user identification identifies each user as being unique within the system. This facilitates file management in that once the userid command is invoked all future references to files will be to files within that userid unless explicitly stated otherwise. The HP 64000 uses six characters and must begin with an uppercase alpha character.

time

HH:MM. Allows the user to enter the correct time on the 24-hour clock displayed on the status line. This also facilitates file management since files can be referenced by time and date.

date

DD/MM/YY. (Day/Month/Year) Allows the user to enter the correct date into the system. This aids the file management system since files can be referenced by date and time.

store

This command will transfer files from the disk to the tape cartridge. The user specifies the file name and file type or all files. If all files are specified the system will store only the source files, linker command files, and emulator command files. Other file types may be stored but the file type must be specified. Other file types can readily be regenerated. This command will overwrite any previous contents of the tape cartridge.

append — Allows files to be appended to files previously stored on tape.

verify — Verify compares a file on the disk to a file resident on the tape cartridge. The user has the option of specifying a single file or all files on the tape assigned to the current userid.

restore — This command will transfer files from the tape cartridge to the disk. The user can specify file name or names and file type.

purge — This command will remove specified files from the active file list. A purged file can be recovered providing it has not been written over.

recover — Recover is used to recover files which have been purged. Files, if not written over, will be returned to the active file list.

rename — Allows the user to rename files. This is used to rename a file before recovering a previous file with the same name. This command also allows the user to transfer a file from one userid to another userid.

copy — Copy allows a disk file to be copied to, or from, the tape, display, or another file name or the RS232 port. The current display or a file may be copied to the printer.

directory — This command provides a listing of those files on the disk, the tape cartridge, and those recoverable files. The listing information consists of file name, file type, file size, last modified data, and last access date. Options: A directory can be made to include all userids, all types of files, before or after a specified date a file has been accessed or modified, and files on a specified disk unit.

library — This command is used to build libraries of relocatable files for use by the linker. These library files consist of relocatable files to be selectively loaded by the linker.

log — This creates a command file for all legal keystrokes. The log function is either toggled on or off by the "log" soft key.

(CMD_FILE) — This soft key represents a syntactical variable to be supplied by the user. This variable is a file name consisting of system commands which the development system will execute. A command file can be generated through the use of the editor or by using the log soft key.

Editor Commands

The 64000 editor commands are listed below:

revise This mode is toggled ON and OFF and allows text to be modified. Modification may include character insertion or deletion. All appropriate command soft keys including "insert" are operational within the "revise" mode.

delete This mode allows deletion of one line or a group of lines specified by the limit specified. The syntax "thru" includes deletion of the limit while the syntax "until" is not inclusive of the limit. The limit can be specified as a line #, string within a line, or as a start or end of text.

find This command allows the user to search the text for the occurrence of the string. The find parameters include a (string) consisting of a single character or any combination of characters; (limit) allows the user to specify the boundaries of the search.

replace This command allows text replacement of a string, a character, partial string with another character, string or partial string. There is an optional (limit) parameter that can specify boundaries of replacement.

(line #) This command causes the line to become the current line of text.

end This terminates the edit session and directs it to a specific destination. Usually this destination is a new file name. If no new file name is specified, the edit session terminates by purging the original file and replacing it with the edited file.

merge Merge allows the user to merge an entire file or portions of it into the file being edited. Any text added to the file being edited will be added after the current line. Delimiters can be specified to determine the amount to be merged.

copy Copy places specified text into a temporary storage buffer on disk for future use. The copy command will overwrite any text previously stored in the buffer. This is avoided by selecting the append option. The default value for (limit) is the current line only.

extract This command removes the specified lines and places them into temporary storage space. If the append option is not selected, the extracted text will overwrite previously stored text. If (limit) is not specified, the current line will be extracted.

retrieve This command retrieves the text from temporary storage and inserts into the program following the current line. The user has the option regarding the number of times the text is to be retrieved.

insert This allows insertion of a combination of ASCII characters, after the current line of text. Insert is executable in the command mode, revise and insert mode.

list This allows the user to list a file to another file or to a printer in numbered or unnumbered format. The listing will be exactly like the file text. There is also a (limit) option available.

renumber This command renumbers the edited text starting from line one.

repeat Repeat allows the user to duplicate the current line of text and add it immediately after the current line. The user can specify the number of times the repeat command is executed.

tabset This command allows the user to set tabs in the desired column. The user has the choice of all 240 columns. Any character can be used to set tabs in any desired location.

range Range restricts the columns to which find and replace commands are constrained. Columns 1 through 240 can be specified. The range function is toggled ON and OFF. When ON, the label range displays in inverse video.

autotab This function provides an automatic tab function that is based on the first nonblank column of the previous line of text. Depressing the shift and the tab keys simultaneously allows tab back from autotab position.

Assembler Soft Key Definitions

The following provides the definitions of the 64000 assembler soft keys.

Key Label **Definition**

(FILE) This indicates the name of the source file that will be assembled.

listfile This soft key specifies the destination of the assembler's output. The options available are listing the output to a specified file, to the display, to the printer, or to null (no generation of a list). If no list file option is specified, the assembler output listing defaults to the device previously specified by the user when the userid was declared.

options This soft key provides the user with a selection of five options specifying the type of output listing.

list This provides a listing of the source program excluding macro or data expansion. All no list pseudoinstructions in the source code are ignored.

nolist Selection of this soft key provides no listing except error messages. All list pseudoinstructions in the source code are ignored.

expand This soft key lists all source and macro generated codes. All list pseudoinstructions in the source program are ignored.

nocode This option causes the source program to be assembled without placing it in a relocatable file.

xref The selection of this option turns on the symbol cross-reference feature of the assembler and lists this table.

Assembler Pseudoinstructions

Pseudoinstructions are instructions used only by the assembler. They produce no executable code for the processor and normally do not take up any memory locations. They are used by the assembler to make programming easier. The following list contains those pseudo ops and their definitions supported by the HP 64000 assembler.

Op Code	**Function**
ASC	Stores data in memory in ASCII format.
BIN	Stores data in memory in binary format.
COMN	Assigns common block of data or code to a specific location in memory.
DATA	Assigns data to a specific location in memory.
DEC	Stores data in memory in decimal format.
END	Terminates the logical end of a program module. Operand field can be used to indicate starting address in memory for program execution.
EQU	Defines label field with operand field value. Symbol cannot be redefined.
EXPAND	Causes an output listing of all source and macro generated codes.
EXT	Indicates symbol defined in another program module.
GLB	Defines a global symbol that is used by other modules.
HEX	Stores data in memory in hexadecimal format.
LIST	Used to modify output listing of program.

MASK Performs and/or logical operations on designated ASCII string.

NAME Permits user to add comments for reference in the linker list.

NOLIST Suppresses output listings (except error messages).

ORG Sets program counter to specific memory address for absolute programming.

PROG Assigns source statements to a specific location in memory. Assembler default condition is "PROG" storage area.

REPT Enables user to repeat a source statement any given number of times.

SKIP Enables user to skip to a new page to continue program listing.

SPC Enables user to generate blank lines within program listing.

TITLE Enables user to create a text line at the top of each page listing for the source program.

The following pseudo ops are for the 8080 and 8085 assembler.

DB Stores data in consecutive memory locations starting with the current setting of the program counter.

DS Reserves the number of bytes of memory as indicated by the value in the operand field.

DW The define word pseudo stores each 16-bit value in the operand field as an address with the least significant byte stored at the current setting of the program counter. The most significant is byte stored at the next higher location.

Assembler Error Codes

The following provides a description of the 64000 assembler error codes.

Error Code **Definition**

AS ASCII string; the length of the ASCII string was not valid or the string was not terminated properly.

CL Conditional label: Syntax of a conditional macro source statement requires a conditional label that is missing.

DE Definition error: Indicated symbol must be defined prior to its being referenced. Symbol may be defined later in the program sequence.

DS Duplicate Symbol: Indicates that the symbol has been previously defined in the program. Th 's when the same symbol is equated to two values (usin; rective) or when the same symbol labels two instruction

DZ Division by zero: Invalid mathematica n resulting in the assembler trying to divide by zero.

EG External Global: Externals cannot be s globals.

EO External Overflow: Program module ıay external declarations (512 externals maximum).

ES Expanded Source: Indicates insuff buffer area to perform macro expansion. It could ult of too many arguments being specified for a pa ɔstitution, or too many symbols being entered into th inition.

ET Expression Type: The resulting t ression is invalid. Absolute expression was expected ɔund or expression contains an illegal combination ble types (refer to Chapter 2 of the *Assembler Man* s and conventions).

IC Illegal Constant: Indicates that ıbler encountered a constant that is not valid.

IE Illegal Expression: Specified exp ıther incomplete or an invalid term was found within tl ɔn.

IO Invalid Operand: Specified o either incomplete or inaccurately used for this oper , occurs when an unexpected operand is encountered erand is missing. If the required operand is an expre: error indicates that the first item in the operand field

IP Illegal Parameter: Illegal par the macro header.

IS Illegal Symbol: Syntax expec ıtifier and encountered an illegal character or token.

LR Legal Range: Address or dis t causes the location counter to exceed the maximum ocation of the instruction's addressing capability.

MC Macro Condition: Relatior ional) operator in macro is invalid.

MD Macro Definition: Macro before being defined in the source file. Macro definiti ecede the call.

ML Macro Label: Label not f n the macro body.

MM Missing Mend: Indicate: cro definition with a missing mend directive was inclı program.

MO Missing Operator: An a perator was expected but was not found.

MP Mismatched Parenthes right or left parenthesis.

MS Macro Symbol: A loc within a macro body was not found.

NM Nested Macro: A macro definition is not permitted within another macro.

PC Parameter Call: Invalid parameter in macro header.

PE Paremeter Error: An error has been detected in the macro parameter listed in the source statement.

RC Repeat Call: Repeat cannot precede a macro call.

RM Repeat Macro: The repeat pseudo operation code cannot precede a macro definition.

SE Stack Error: Indicates that a statement or expression does not conform to the required syntax.

TR Text Replacement: Indicates that the specified text replacement string is invalid.

UC Undefined Conditional: Conditional operation code invalid.

UO Undefined Operation code: Operation code encountered is not defined for the microprocessor, or the assembler disallows the operation to be processed in its current context. This occurs when the operation code is misspelled or an invalid delimiter follows the label field.

UP Undefined Parameter: The parameter found in macro body was not included in the macro header.

US Undefined Symbol: The indicated symbol is not defined as a label or declared an external.

Linker Commands

The 64000 linker commands are defined below:

Key Label	Definition
link	Initiates the link process.
(CMDFILE)	A syntactical variable supplied by the user. This would be the name of linker command file previously established.
listfile	Allows the user to select a destination other than the system default for the linker output listing.
display	Using this command designates the display as the output destination for the linker output listing.
(FILE)	Syntactical variable supplied by the user. This would be the name of a disk file to which the output of the linker would be directed.
null	Using this command suppresses the output listing. Error messages will still be output to the default destination as previously selected by the user.

printer This designates the printer to be the destination of the linker output listing.

options Soft key which precedes the selection of a linker option.

edit Available linker option to edit a previously established linker command file.

nolist Available linker option to suppress the generation of a linker load map.

Soft Key Definitions

The 64000 emulator soft key definitions are given below:

Label **Description**

run This starts program execution in the emulation processor. Execution begins at the location specified by "from" and ending under the conditions specified by "until." If no limits are specified, emulation will begin at the current address until halted by a "stop run" or by a boundary specified by "until."
Syntax: run from (ADDRESS OR SYMBOL) until (ADDRESS OR SYMBOL)

step This function causes the emulation processor to execute one instruction at a time. Once in the step mode, each depression of the return key will cause another instruction to be executed and displayed. The user can specify the number of steps to be executed each time the return key is pressed and the address from which stepping occurs. If these parameters are not specified, the system defaults to stepping from the current program counter location, executing one instruction each time the return key is pressed.
Syntax: step # of (STATES) from (ADDRESS)

trace This key is used to control the analysis function of the system, allowing the triggering and capturing of data of the emulation data bus.
Syntax: trace in_sequence—permits tracing on a sequence of events.
trace after—captures and displays data after the trigger qualifier word is satisfied.
trace about—captures and displays data before and after the trigger qualifier.
trace only—allows explicit definition of the information to be captured in the trace.

trace continuous—allows continuous monitoring of trace information without reentering the trace command.

display This command causes the system to display a variety of data types on the development station's screen. Data types can be specified as global symbols, local symbols, and last active trace specification (valid only with the analysis card), the last active run specification, the trace buffer (valid only with analysis card), contents of proper emulation microprocessor registers, absolute or relative time display (valid only with analysis card), or contents of user or emulation memory.

Syntax: display trace
Syntax: display register (REGISTER NAME)
Syntax: display memory (ADDRESS)
Syntax: display trace specification
Syntax: display run specification
Syntax: display count
Syntax: display global symbols
Syntax: display local symbols

The mode option for the trace, register, and memory display provides the user with a choice of how the data will be presented on the screen. The following modes are defined:

static The system will display the current conditions or contents one time only. No update will be shown.

dynamic The system will continually update the display as data are changed in the emulation system.

absolute The system displays data in absolute numeric code. (i.e., hexadecimal or octal).

mnemonic The system presents the data in the appropriate assembly language.

offset by The system displays program modules so that the address values are offset by a specified value.

no offset The system displays all addresses in program modules with those values assigned by the linking loader.

packed The system displays opcodes and operands on the same line.

block The system displays more data on the development station by displaying multiple columns of data.

modify This command allows the user to change the contents of the emulation memory or processor registers to correspond to data entered from the console keyboard.
Syntax: modify (ADDRESS) to (VALUE)

Syntax: modify memory (ADDRESS) thru (ADDRESS) to (VALUE)

Syntax: modify register (REGISTER NAME) to (VALUE)

stop — This command halts the execution of either the run or trace commands. If stop-run is executed, it can be continued by a run command without skipping any of the intervening of the program code.

Syntax: stop run

Syntax: stop trace

end — Selecting this soft key changes the operating mode of the station, allowing other tasks to be performed. "end" does not stop the emulation process. Emulation continues even as other functions are performed on the system.

Syntax: end_emulation

load — load_memory transfers abolute object files from the system's disk into emulation or user RAM memory.

Syntax: load memory (FILE)

count — The count command is used in conjunction with a trace command. The count command is used to measure the elapsed time or the number of times certain user-specified events occurred between the start and end times specified by the trace.

Syntax: count time
count address = (ADDRESS)

copy — This command allows the user to transfer data from one location of emulation or user memory to the system's disk. The content of memory from which the data are taken remains unchanged.

Syntax: copy (ADDRESS) thru (ADDRESS) to (FILENAME)

list_to — This command allows the user to make a permanent record of the contents of the stations display by writing it to a file on the disk or to the line printer.

Syntax: list display to printer

Syntax: list display to (FILE)

restart — Upon initializing the restart command, the microprocessor's program counter is reset to 0000H and the processor is reinitialized. It is important to execute the run command from the appropriate place in emulation memory.

edit_cnfg — (Edit-Configuration). This command recalls the series of queries which allows mapping of memory space and fault selection. When this command is invoked the previous responses can be modified by the user.

Syntax: edit_cnfg

Following are the monitor level soft keys which will be in effect after December 1981:

```
edit compile assemble link emulate prom_prog run ---etc---
directory purge rename copy library recover log ---etc---
uerid date&time opt_test terminal (CMDFILE) --TAPE--- ---etc---
```

"--TAPE---" Soft Key

After --TAPE--- is returned the following soft keys are available.

```
store restore append verify tension directory ---etc---
```

"date&time" Soft Key

After date&time is depressed the following soft keys are available.

```
(DATE) (TIME)
```

"opt_test" Soft Key

This executes option test, which provides performance verification tests for options that are present.

Terminal Mode

"terminal Soft Key

This puts the station in an RS232 terminal mode which allows it to be a terminal to another system.

Passwords

The capability to have increased file security using passwords has been added. Following is the new syntax for userid.

"userid" Soft Key

After userid is depressed the following soft keys are available.

```
(USERID) listfile
```

After USERID is entered the following soft keys are available.

```
listfile  password
```

After password is entered the following soft keys are available.

```
(PASSWD)
```

The user-types in his password. This is nonprinting so he will not see on the display what he entered.

"HOST" PASCAL

"HOST" PASCAL consists of a compiler to allow users of the 64000 system to write programs that will execute on the internal host processor. In order to execute these programs the following syntax is used.

"run" Soft Key

After run is depressed the following soft key is available.

```
(FILE)
```

After a file is specified the following soft keys are available.

```
input  output
```

After input is depressed the following soft keys are available.

```
(FILE)  keyboard
```

After output is depressed the following soft keys are available.

```
(FILE)  display  display1  printer  null
```

Summary of the HP 64000 Development System

Example A-1

This example shows how to create a new file and edit it. The file to be created is listed below:

```
"8085"
         ;   THIS PROGRAM STARTS AT 0 AND ADDS LOCATION 100H TO
         ;   LOCATION 101H AND STORES THE RESULT IN LOCATION 102H
             NAME    "ADD_WORKSHOP_1"
             ORG     2000H            ;   PROGRAM ORIGIN WILL BE AT
                                      ;    HEXADECIMAL 2000
START        LXI     H,100H           ;   LOAD HL PAIR WITH 100H
             MOV     A,M              ;   MOVE NUMBER IN LOCATION 100
                                      ;   INTO THE ACC.
             INX     H                ;   INCREMENT H AND L PAIR
             ADD     M                ;   ADD LOCATION 101 TO ACCUMULATOR
             INX     H                ;   INCREMENT HL PAIR
             MOV     M,A              ;   STORE ACC IN LOCATION 102H
             JMP     START
```

Procedure

Step 1: Press soft key "userid" and type in your USERID. The HP doesn't ask for the time and date after you enter your USERID. Rather, you have to press the soft key "Date&Time" to change it.

Step 2: To enter the edit mode, you have to create a new file. We'll call this new file "ADD".

```
edit into ADD (RETURN)
```

Step 2.5: It is a good idea to set up tabs so that if you want to you may jump to the opcode, operand or comment. You do this by pressing the softkey "Tabset". The editor will then display a tab row in which you can type "T" at the current cursor position. Then when you want to move faster, the "tab" key will jump to where you set your tabs. The way we did it was:

```
tabset 7 17 27 37
```

To save your tab sets, type the inverse softkey text "tabset" again.

Step 3: The first line of the program is the assembler directive, which lets the assembler know what microprocessor you wish to emulate.

```
"8085" (RETURN)
```

Step 4: To enter comments, type a "*" in column 1 and then start with your comments. If you want to start it anywhere else on that line, type a ";" and enter your comment.

```
* This is a comment (The "*" is at the leftmost edge of column 1)
; This is a comment (The ";" can be at any position).
```

Note: Your comments must come after you type "*" or ";".

Step 5: Enter "Name" and a brief explanation of the file. This lets you know what a particular file does in case you have to link many files. This is optional.

```
(TAB) NAME (TAB) "ADD_WORKSHOP_I" (RETURN)
```

Step 6: Enter "ORG" to let the assembler know the starting location of your program — in this case, at 2000H.

```
(TAB) ORG (TAB) 2000H (TAB) ; COMMENTS (RETURN)
```

Step 7: Enter the label "START" at column 1 of the next line along with the first instruction. This label helps the user to remember the English word rather than what the number was, if the user wants to loop or jump to that part of the program again.

```
START (TAB) LXI (TAB) H,100H (TAB) ; COMMENTS (RETURN)
```

Step 8: If you use only a few labels in your program, it is wise to use the softkey "AUTOTAB". This softkey jumps to the next line and moves the cursor right under the first word of the previous line. This saves time. Also note that the "TAB" key can also do this if you specified the tab sets.

Step 9: Enter the rest of the program. This is the program listing called "ADD".

Step 10: To list your file to the printer, make sure the printer is on-line (the light that's adjacent to the word should be on; if not, push the "on-line" button on the printer). Then type:

```
list printer all (RETURN)
```

Step 11: To save the file, type

```
end (RETURN)
```

Note: If your file wasn't named when you entered the edit mode, then type

```
end ADD (RETURN)
```

Step 12: The file is stored onto the hard drive or disk. To see your file on it, type

```
directory (RETURN)
```

You should then see the file "ADD" with the type "SOURCE". You will also see when you last modified it and accessed it. This information is important, so that you know how updated your file is. The directory listing will only show those files under your USERID.

Step 13: To re-edit the file, type

```
edit ADD (RETURN)
```

Notice that you don't type "edit into ADD". If you want to load your file from a disk, you have to specify the drive number. For disk drive X, type "ADD:X", where X is the disk drive number. If no drive number is specified, then the default is 0.

Step 14: To use the insert softkey, type a "NOP" after line 9 by

```
9 (RETURN)
insert (TAB) NOP (TAB) (TAB) ; NO OPERATION (RETURN)
```

Note: If you get an error, go to Step 15 and then back to 14. This may be because the editor was trying to find line 9, but your file has line numbers reading "NEW" instead.

Step 15: To renumber your file in order to give your editor a way to find what line to edit, type

```
renumber (RETURN)
```

Step 16: If your file is very large and you want to search your file for a particular word like "NOP", type

```
find "NOP" all (RETURN)
```

Note: Remember to enclose all strings with double quotes. If not, the editor will think it is a softkey command.

Step 17: To insert more text, type

```
insert (RETURN)
```

then move the cursor up, down, or sideways and begin typing the new line.

Step 18: To revise a line, enter

```
revise (RETURN)
```

This edits the line that the cursor is on. If that line isn't what you want, then move the cursor using the cursor keys.

Step 19: To move the display to allow for viewing all of the columns, depress the SHIFT and LEFT arrow keys simultaneously. Hitting SHIFT and

RIGHT keys will scroll the text right. To scroll the text up or down, hit the edit key ROLL UP or ROLL DOWN, respectively.

Step 20: To insert or delete character(s) when revising, hit the edit key INSERT CHAR or DELETE CHAR, respectively.

Step 21: To delete a line at the current cursor position, hit DELETE (RETURN). If you want to delete a line somewhere else, type

```
extract (RETURN)
```

Then move the cursor to where you want to insert that line and type

```
retrieve (RETURN)
```

Note: If you want to insert many copies of that line at the current cursor position, then type **retrieve** # (RETURN), where # is the number of copies.

Step 23: To abort the editor and not save your file, press the special function key RESET twice. Pressing it once will pause a running listing or program.

Step 24: To replace a word with another word type

```
replace "word1" with "word2" all
```

This will replace all word1s and word2s. You can also specify where you want to stop replacing by using **thru** or **until** a certain line number.

Step 25: The "copy" command lets you copy a group of lines without erasing those lines, like "extract" does. First, place the cursor at the starting location line and then type

```
copy thru line # (RETURN)
```

where # is the last of your lines to copy. Next, move the cursor to the place where you want it inserted, and type **retrieve.**

Step 26: The "merge" command lets you insert an entire file or copies a block of lines like the "copy" command does. To merge a file, type

```
merge ADD (RETURN)
```

This lets you insert the file ADD at the current cursor position.

Example A-2

This example goes through the steps in assembling a file. Upon completion, it will create a "reloc" file to be later used for linking purposes.

Procedure

Step 1: Enter your userid, and optionally enter the time and date. The HP 64000 already has the current date and time, so updating isn't necessary.

```
userid USERID (RETURN)
```

Step 2: To assemble your source file called "ADD" and make a printout of it with the cross-reference table, type

```
assemble ADD listfile printer options xref (RETURN)
```

The printer will then output the assembled file that has both a source and object listing. It will then show the cross-reference table that lists any labels you put into that file and what lines accessed it or needed it to run.

Step 3: To show how the assembler command treats an error, we'll put one in by doing this

```
edit ADD (RETURN)
11 (RETURN)
insert (TAB) MVI (TAB) D,FFH (RETURN)
end (RETURN)
```

Step 4: Now assemble the file by typing the commands from Step 2. The assembler treated "FFH" as a symbol, not a hex value, so it generated an error. You will see the number of errors is one and that the error is an undefined symbol. On the xref table, FFH is a type U, which means it's undefined.

Step 5: To fix this error, all hex values beginning with an alpha character must be preceded with a "0". Do this by

```
edit ADD (RETURN)
12 (RETURN)
revise (TAB) (TAB) D,0FFH (RETURN)
end (RETURN)
```

Step 6: Now assemble the file again, and this time list it to the screen or display.

```
assemble ADD listfile display options xref (RETURN)
```

To pause the display from scrolling up so fast, type RESET. To resume scrolling, type any other key.

Step 7: Your assembled file is stored on your USERID directory. To see it type

```
directory (RETURN)
```

You will then see two more "ADD." files with the type "reloc" and "asmb_sym".

These files are useful for the assembler and linker programs. The "reloc" file is an object file containing the hex values of your program. It then must be made into an "absolute" file so it can run by itself.

Example A-3

This example goes through the steps in linking the "reloc" file to create an "absolute" file. This new file can then be run independently, emulated, or even used by the PROM programmer.

Procedure

Step 1: Initialize the linker and show the results to the display by

```
link listfile display (RETURN)
"Object files ?"
```

Step 2: This new message asks you what file(s) you want to be linked, so type

```
ADD (RETURN)
"Library files ?"
```

Step 3: There are no library routines in "ADD" so skip it by

```
(RETURN)
"Load addresses: PROG,DATA,COMN = 0000H,0000H,0000H"
```

Step 4: This command allows you to specify different memory areas for the program, data, and common modules. No memory assignment is needed because the "ADD" file already has an ORG statement, so skip it by

```
(RETURN)
"More files ?"
```

Step 5: Since there is only one file to be linked, respond by

```
 no (RETURN)
"LIST,XREF, overlap_check,comp_db = on off on off"
```

Step 6: The linker is prompting the user to specify the output and declaring the default for the output listing. It then checks to see if your memory assignments overlap as well. Ignore this and type

```
(RETURN)
"Absolute file name ?"
```

Step 7: The linker wants you to enter the file name to be assigned to the absolute file.

```
ADD (RETURN)
```

Step 8: You will then see the linker examining your file

```
"STATUS:Linker : HP 640000S linker: Pass1"
"STATUS:Linker : HP 640000S linker: Pass2"
"STATUS:Linker : HP 640000S linker: End of link"
```

Note that the above display will be on a single line.

Step 9: The linker output will display the start and end location of your program, the current date and time, the assembler pseudo name "ADD_WORKSHOP_I", and extra data like XFER address and the total bytes loaded.

Step 10: To view your new files on the directory, type

```
directory (RETURN)
```

You should then see three new files with types "link_sym", "link_com", and "absolute". The absolute file is used for the emulator or the PROM programmer. The "link_com" is a command file that holds all of the data that you entered from Steps 1 to 7. This is good if you want to keep the same link configuration, but want to change or re-edit your source file(s).

If you want to save a linker listing to a file for later viewing type

```
link ADD listfile ADD_L (RETURN)
```

This will create a link listing similar to that of Step 9. Specifying "link ADD" will tell the linker to link it using its "link_com" file.

Example A-4

This example goes through the steps of emulating an absolute file without external hardware. This gives you a good idea of the importance of a development system like the HP 64000.

Procedure

Step 1: Before beginning, you must go through Examples 1 through 3. Also, the HP 64000 must have an 8085 emulator. If it is configured or has a 68000 emulator, emulation program will use only it, so be wary of this.

Step 2: To enter the emulation mode and load the absolute file do this

```
emulate load ADD (RETURN)
"Processor clock ?"
```

Step 3: This question asks if you want the source of the processor clock to be internal or external. Since there is no external hardware being used, type

```
internal (RETURN)
"Restrict processor to real-time runs ?"
```

Step 4: This question asks if you want to restrict the processor to real-time runs, which will limit the analysis functions that can be performed, such as debugging your program. An example of this would be "display registers blocked". So answer

```
no (RETURN)
"Stop processor on illegal opcodes ?"
```

Step 5: Specify "yes" so that the emulator will stop if an illegal opcode is detected.

```
yes (RETURN)
```

Step 6: The emulator will then want to specify the memory range for your emulation ram/rom and user ram/rom. Since all memory is internal (no user external hardware for this file), address 100H to 103H is used for storing the variables used in the program. Define this to be emulation RAM. To protect your program, define the memory to be emulation ROM. Thus, if something writes to your program, it will generate an error.

```
100H thru 103H emulation ram
2000H thru 20FFH emulation rom
```

Also notice that the ram and rom ranges are from 100H-3FFH and 2000H-20FFH. This may be because the emulator can only provide a range of memory area rather than a specified one.

Step 7: To keep your defined memory area, type

```
end (RETURN)
"Modify simulated I/O ?"
```

Step 8: Since we are not using any I/O ports, type

```
no (RETURN)
"Modify interactive measurement specification ?"
```

Step 8.5: This question is not in the book so type

```
(RETURN)
"Command file name ?"
```

Step 9: The emulator wants you to specify a file name to which to assign the emul_ com file configuration. Specify with

```
ADD (RETURN)
```

Once doing this, the emulator will then load your absolute file to the memory areas you specified. You should then see

```
"STATUS: 8085--Program loaded"
```

If you want to make any modifications, you have to start again, by re-editing, assembling, linking and then emulating. If you want the same emulating configuration that you specified in Steps 1 to 9, then type **emulate ADD load ADD.**

Step 11: To display your program with mnemonics, type

```
display memory 2000H mnemonic (RETURN)
```

You should then see the locations with their corresponding instructions of your program.

Step 12: To change the values that your program uses to add two numbers, you have to modify the emulation RAM by

```
modify memory 100H thru 102H to 02H (RETURN)
display memory 100H blocked (RETURN)
```

You should now see a display of your edited bytes. The memory block map will show you the address, data, and ASCII translation of each byte.

Step 13: To run your program, you can type either

```
run from 2000H (RETURN)
```

or

```
run from glob_sym START (RETURN)
"STATUS: 8085--Running"
```

Your program will keep running, so you can now modify the memory locations from Step 12 to something else, and then see the changes.

Step 14: You can also single-step the program to execute a single instruction at a time. This is a good debugging tool on the 64000.

```
break (RETURN)
"STATUS: 8085--break in background"
display registers (RETURN)
step from 2000H (RETURN)
```

or

```
step from glob_sym START (RETURN)
step (RETURN)
(RETURN)
```

Continuously pressing return will execute the "step" command again. Remember that pressing "return" will execute anything on the command prompt, no matter where the cursor is. You should then see a display of each instruction being executed with its corresponding register values. This is really good because you can trace any program, and scan the instructions, registers, flags, stack pointer, and the next IP.

Step 15: You can also set up breakpoints to stop the program when it reaches a certain argument. An example of this would be "run" from 2000H until address 2006H.

Step 16: To set up a breakpoint at address 2005H, type

```
run from 2000H until address 2005H (RETURN)
```

Step 17: Now say you want to halt the program after a memory write:

```
run from 2000H until status memory_write (RETURN)
```

Step 18: To end the emulation session, type

```
end (RETURN)
```

You can also press "RESET" twice too.

Step 19: If you want to get back to the emulation and keep the same emulator configuration, type

```
emulate ADD load ADD (RETURN)
```

If you want to change the emulator configuration type

```
modify_configuration (RETURN)
```

Note that the HP 64000 can perform in-circuit emulation with or without a large-system hardware.

Operation of the 68000 Emulator

In order to use the 68000 emulator, a monitor program must be included in the linking of the user's program. The purpose of the monitor is to provide special functions during emulation (including register display, software breakpoint setting, etc.).

The steps in the emulation process are as follows:

1. Create a program using the 64000's text edition.
   A. Make sure that "68000", including quotations, is the first line in the editor.
   B. Make the second line in the application program "PROG". (This will cause the monitor program to be successfully linked with the application.)
   C. Use "H" for Hex instead of "$" signs.
   D. Write your application program.
   Locate the program between 0FFH and 10000H-application size.
2. Assemble the program you write by typing:

```
<Assemble> MON_68K
```

3. Make a copy of the assembly program "MON_68K" by typing:

```
COPY Mon_68K:HP:source TO MON_68K
```

   (Note upper and lower case. Type it exactly as shown.)
4. Now assemble this program by typing:

```
SOFTKEY
<Assemble> MON_68K
```

5. Now, the application and monitor program must be linked together. Type:

```
SOFTKEY
<LINK>
Object File?           MYFILE
Library Files?         <CR>
Prog,Data,Comn,A5=     000XXX,0H,0H,0H where XXX=100H to
                          10000H
More Files?            <yes> Soft Key
Object Files?          MON_68K
Library Files?         <CR>
Prog,Data,Comn,A5=     10000H,0H,0H,0H
More Files?            <No> Soft Key
Absolute File Name=    MYFILE
```

Assume MYFILE is the name of your application program file.

   A. Notes: The monitor program is position independent. Since the TARGET SYSTEM has limited address space, it is suggested that you locate your program in **THAT** address range. Furthermore, this allows the user to specify the monitor program's address in upper-address space and in emulation RAM.

   B. Care should be taken so that address ranges 000H→0FFH are reserved for vectors, and that user program addresses do not conflict with the monitor program whose size is about 1000_{16} bytes. I locate the monitor at address 10000H→10FFFH.

6. Finally, the emulator is entered by answering questions with their default values (unless the user wishes otherwise).

   A. For the memory map, the following should be entered:

0000H-0FFFH	Emulation ROM (Vectors, Interrupts, User Prog)
10000H-10FFFH	Emulation RAM (Monitor Program)
0WWWH-0VVVH	User RAM (other space for ports, tables, etc. that exist in either software or the target system)

   B. Some helpful commands during emulation (<XXXX> = Soft Key):

1. <Load> MYFILE	Loads both your file and the monitor program (linked)
2. <Modify><Config>	Lets you change the emulator configuration
3. <Modify><Softwre_bkpts>	Modifies software breakpoints so that you can stop program execution anywhere

NOTE: The 68000 emulator DOES NOT allow single-stepping.

4. <BREAK>	Enter the monitor program
5. <DISPLAY><MEMORY> <MNEMONIC>	Display disassembled code
6. <MODIFY><REGISTER> <MODIFY><MEMORY>	Modify address space and registers

DO NOT put ORG statements in your program except for the interrupt vectors.

APPENDIX B

MOTOROLA

Advance Information

MC68000L4
(4 MHz)
MC68000L6
(6 MHz)
MC68000L8
(8 MHz)
MC68000L10
(10 MHz)

HMOS
(HIGH-DENSITY, N-CHANNEL, SILICON-GATE DEPLETION LOAD)

16-BIT MICROPROCESSOR

16-BIT MICROPROCESSING UNIT

Advances in semiconductor technology have provided the capability to place on a single silicon chip a microprocessor at least an order of magnitude higher in performance and circuit complexity than has been previously available. The MC68000 is the first of a family of such VLSI microprocessors from Motorola. It combines state-of-the-art technology and advanced circuit design techniques with computer sciences to achieve an architecturally advanced 16-bit microprocessor.

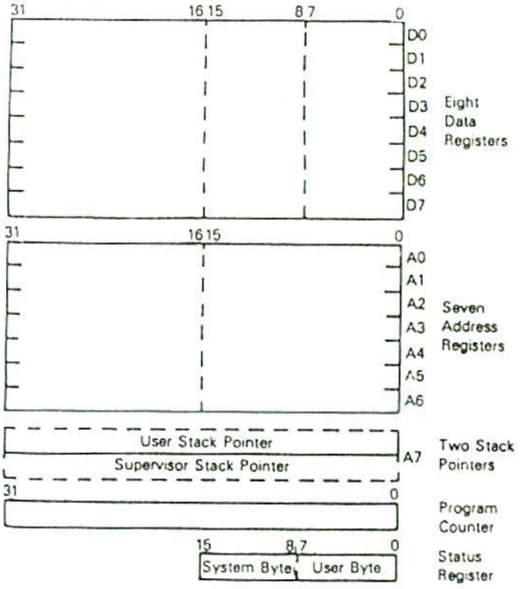
The resources available to the MC68000 user consist of the following:

- 32-Bit Data and Address Registers
- 16 Megabyte Direct Addressing Range
- 56 Powerful Instruction Types
- Operations on Five Main Data Types
- Memory Mapped I/O
- 14 Addressing Modes

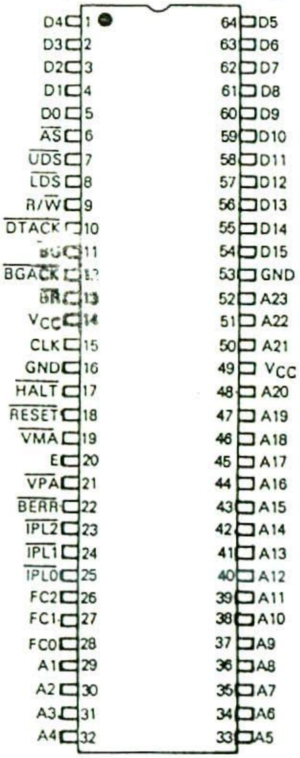
As shown in the programming model, the MC68000 offers seventeen 32-bit registers in addition to the 32-bit program counter and a 16-bit status register. The first eight registers (D0-D7) are used as data registers for byte (8-bit), word (16-bit), and long word (32-bit) data operations. The second set of seven registers (A0-A6) and the system stack pointer may be used as software stack pointers and base address registers. In addition, these registers may be used for word and long word address operations. All seventeen registers may be used as index registers.

L SUFFIX
CERAMIC PACKAGE
CASE 746

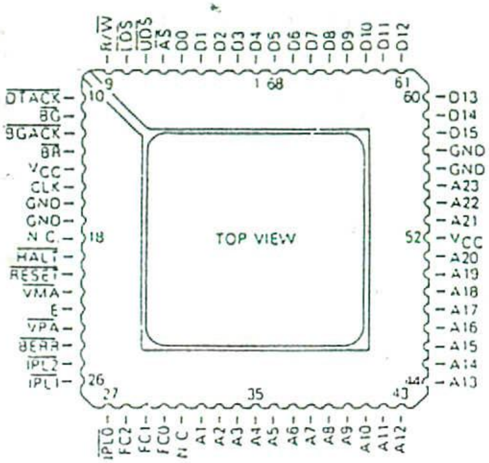
PROGRAMMING MODEL



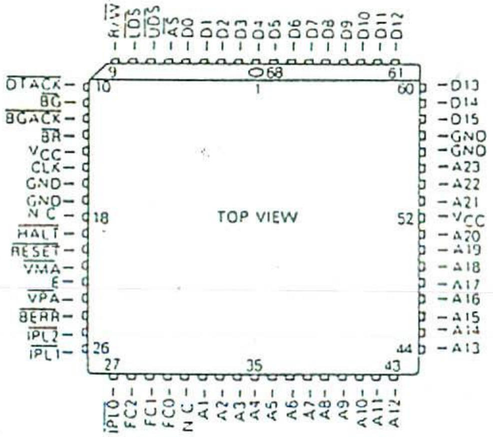
64-pin dual in-line package



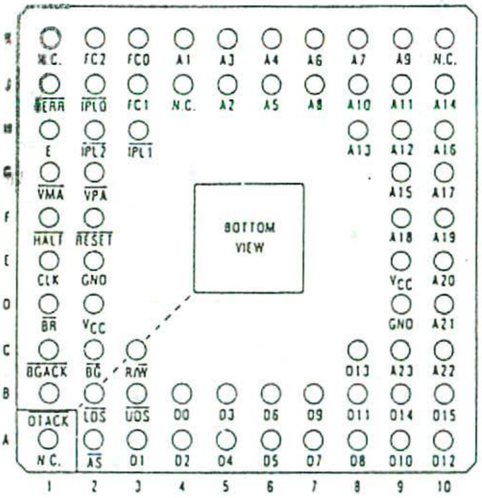
68-Terminal Chip Carrier



68-Pin Quad Pack



68-pin grid array.



	1	2	3	4	5	6	7	8	9	10
K	N.C.	FC2	FC0	A1	A3	A4	A6	A7	A9	N.C.
J	BERR	IPL0	FC1	N.C.	A2	A5	A8	A10	A11	A14
H	E	IPL2	IPL1					A13	A12	A16
G	VMA	VPA							A15	A17
F	HALT	RESET							A18	A19
E	CLK	GND							V_{CC}	A20
D	BR	V_{CC}							GND	A21
C	BGACK	BG	R/W					D13	A23	A22
B	DTACK	LDS	UDS	D0	D3	D6	D9	D11	D14	D15
A	N.C.	AS	D1	D2	D4	D5	D7	D8	D10	D12

MOTOROLA

MC68230L8
MC68230L10

Advance Information

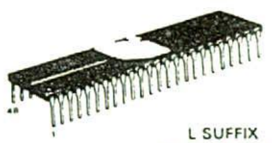
MC68230 PARALLEL INTERFACE/TIMER

The MC68230 Parallel Interface/Timer provides versatile double buffered parallel interfaces and an operating system oriented timer to MC68000 systems. The parallel interfaces operate in unidirectional or bidirectional modes, either 8 or 16 bits wide. In the unidirectional modes, an associated data direction register determines whether the port pins are inputs or outputs. In the bidirectional modes the data direction registers are ignored and the direction is determined dynamically by the state of four handshake pins. These programmable handshake pins provide an interface flexible enough for connection to a wide variety of low, medium, or high speed peripherals or other computer systems. The PI/T ports allow use of vectored or autovectored interrupts, and also provide a DMA Request pin for connection to the MC68450 Direct Memory Access Controller or a similar circuit. The PI/T timer contains a 24-bit wide counter and a 5-bit prescaler. The timer may be clocked by the system clock (PI/T CLK pin) or by an external clock (TIN pin), and a 5-bit prescaler can be used. It can generate periodic interrupts, a square wave, or a single interrupt after a programmed time period. Also it can be used for elapsed time measurement or as a device watchdog.

- MC68000 Bus Compatible
- Port Modes Include:
  Bit I/O
  Unidirectonal 8-Bit and 16-Bit
  Bidirectional 8-Bit and 16-Bit
- Selectable Handshaking Options
- 24-Bit Programmable Timer
- Software Programmable Timer Modes
- Contains Interrupt Vector Generation Logic
- Separate Port and Timer Interrupt Service Requests
- Registers are Read/Write and Directly Addressable
- Registers are Addressed for MOVEP (Move Peripheral) and DMAC Compatibility

HMOS
(HIGH-DENSITY N-CHANNEL
SILICON-GATE)

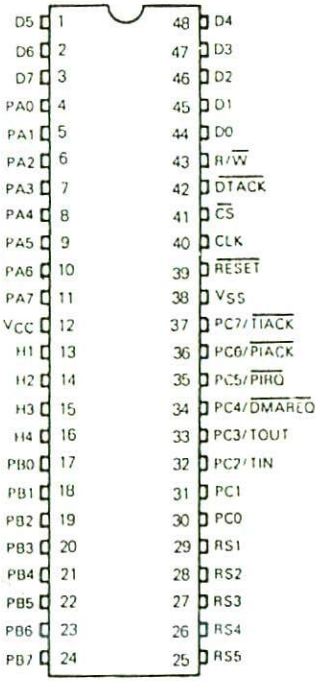
PARALLEL INTERFACE/TIMER



L SUFFIX
CERAMIC PACKAGE
CASE 740

P SUFFIX
PLASTIC PACKAGE
AVAILABLE 2Q82

PIN ASSIGNMENT



D5	1	48	D4
D6	2	47	D3
D7	3	46	D2
PA0	4	45	D1
PA1	5	44	D0
PA2	6	43	R/$\overline{W}$
PA3	7	42	$\overline{DTACK}$
PA4	8	41	$\overline{CS}$
PA5	9	40	CLK
PA6	10	39	$\overline{RESET}$
PA7	11	38	V_{SS}
V_{CC}	12	37	PC7/$\overline{TIACK}$
H1	13	36	PC6/$\overline{PIACK}$
H2	14	35	PC5/$\overline{PIRQ}$
H3	15	34	PC4/$\overline{DMAREQ}$
H4	16	33	PC3/TOUT
PB0	17	32	PC2/TIN
PB1	18	31	PC1
PB2	19	30	PC0
PB3	20	29	RS1
PB4	21	28	RS2
PB5	22	27	RS3
PB6	23	26	RS4
PB7	24	25	RS5

MOTOROLA

MC6821
(1.0 MHz)
MC68A21
(1.5 MHz)
MC68B21
(2.0 MHz)

PERIPHERAL INTERFACE ADAPTER (PIA)

The MC6821 Peripheral Interface Adapter provides the universal means of interfacing peripheral equipment to the M6800 family of microprocessors. This device is capable of interfacing the MPU to peripherals through two 8-bit bidirectional peripheral data buses and four control lines. No external logic is required for interfacing to most peripheral devices.

The functional configuration of the PIA is programmed by the MPU during system initialization. Each of the peripheral data lines can be programmed to act as an input or output, and each of the four control/interrupt lines may be programmed for one of several control modes. This allows a high degree of flexibility in the overall operation of the interface.

- 8-Bit Bidirectional Data Bus for Communication with the MPU
- Two Bidirectional 8-Bit Buses for Interface to Peripherals
- Two Programmable Control Registers
- Two Programmable Data Direction Registers
- Four Individually-Controlled Interrupt Input Lines; Two Usable as Peripheral Control Outputs
- Handshake Control Logic for Input and Output Peripheral Operation
- High-Impedance Three-State and Direct Transistor Drive Peripheral Lines
- Program Controlled Interrupt and Interrupt Disable Capability
- CMOS Drive Capability on Side A Peripheral Lines
- Two TTL Drive Capability on All A and B Side Buffers
- TTL-Compatible
- Static Operation

MOS

(N-CHANNEL, SILICON-GATE, DEPLETION LOAD)

PERIPHERAL INTERFACE ADAPTER

L SUFFIX
CERAMIC PACKAGE
CASE 715

S SUFFIX
CERDIP PACKAGE
CASE 734

P SUFFIX
PLASTIC PACKAGE
CASE 711

MAXIMUM RATINGS

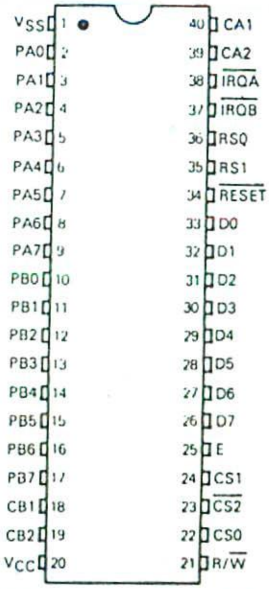
Characteristics	Symbol	Value	Unit
Supply Voltage	V_{CC}	−0.3 to +7.0	V
Input Voltage	V_{in}	−0.3 to +7.0	V
Operating Temperature Range MC6821, MC68A21, MC68B21 MC6821C, MC68A21C, MC68B21C	T_A	T_L to T_H 0 to 70 −40 to +85	°C
Storage Temperature Range	T_{stg}	−55 to +150	°C

THERMAL CHARACTERISTICS

Characteristic	Symbol	Value	Unit
Thermal Resistance Ceramic Plastic Cerdip	θ_{JA}	 50 100 60	°C/W

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage (i.e., either V_{SS} or V_{CC}).

PIN ASSIGNMENT



The expanded block diagram of the MC6821 is shown in Figure B.1.

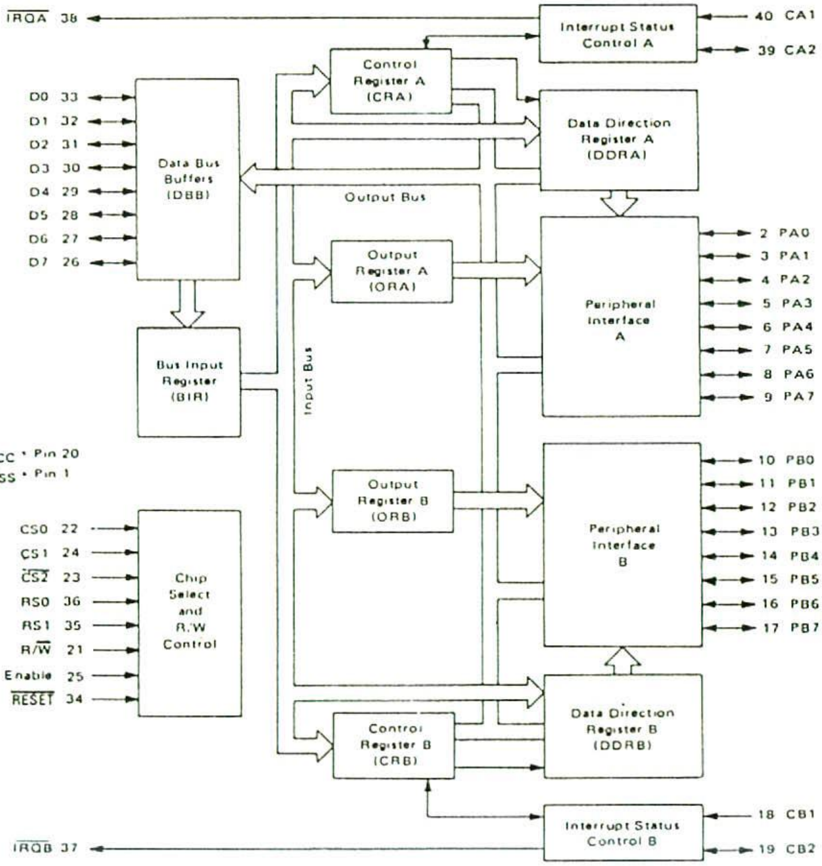


FIGURE B.1 EXPANDED BLOCK DIAGRAM

PIA INTERFACE SIGNALS FOR MPU

The PIA interfaces to the M6800 bus with an 8-bit bidirectional data bus, three chip select lines, two register select lines, two interrupt request lines, a read/write line, an enable line and a reset line. To ensure proper operation with the MC6800, MC6802, or MC6808 microprocessors, VMA should be used as an active part of the address decoding.

Bidirectional Data (D0-D7) – The bidirectional data lines (D0-D7) allow the transfer of data between the MPU and the PIA. The data bus output drivers are three-state devices that remain in the high-impedance (off) state except when the MPU performs a PIA read operation. The read/write line is in the read (high) state when the PIA is selected for a read operation.

Enable (E) – The enable pulse, E, is the only timing signal that is supplied to the PIA. Timing of all other signals is referenced to the leading and trailing edges of the E pulse.

Read/Write (R/$\overline{W}$) – This signal is generated by the MPU to control the direction of data transfers on the data bus. A low state on the PIA read/write line enables the input buffers and data is transferred from the MPU to the PIA on the E signal if the device has been selected. A high on the read/write line sets up the PIA for a transfer of data to the bus. The PIA output buffers are enabled when the proper address and the enable pulse E are present.

$\overline{\text{RESET}}$ – The active low $\overline{\text{RESET}}$ line is used to reset all register bits in the PIA to a logical zero (low). This line can be used as a power-on reset and as a master reset during system operation.

Chip Selects (CS0, CS1, and $\overline{\text{CS2}}$) – These three input signals are used to select the PIA. CS0 and CS1 must be high and $\overline{\text{CS2}}$ must be low for selection of the device. Data transfers are then performed under the control of the enable and read/write signals. The chip select lines must be stable for the duration of the E pulse. The device is deselected when any of the chip selects are in the inactive state.

Register Selects (RS0 and RS1) – The two register select lines are used to select the various registers inside the PIA. These two lines are used in conjunction with internal Control Registers to select a particular register that is to be written or read.

The register and chip select lines should be stable for the duration of the E pulse while in the read or write cycle.

Interrupt Request ($\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$) – The active low Interrupt Request lines ($\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$) act to interrupt the MPU either directly or through interrupt priority circuitry. These lines are "open drain" (no load device on the chip). This permits all interrupt request lines to be tied together in a wire-OR configuration.

Each Interrupt Request line has two internal interrupt flag bits that can cause the Interrupt Request line to go low. Each flag bit is associated with a particular peripheral interrupt line. Also, four interrupt enable bits are provided in the PIA which may be used to inhibit a particular interrupt from a peripheral device.

Servicing an interrupt by the MPU may be accomplished by a software routine that, on a prioritized basis, sequentially reads and tests the two control registers in each PIA for interrupt flag bits that are set.

The interrupt flags are cleared (zeroed) as a result of an MPU Read Peripheral Data Operation of the corresponding data register. After being cleared, the interrupt flag bit cannot be enabled to be set until the PIA is deselected during an E pulse. The E pulse is used to condition the interrupt control lines (CA1, CA2, CB1, CB2). When these lines are used as interrupt inputs, at least one E pulse must occur from the inactive edge to the active edge of the interrupt input signal to condition the edge sense network. If the interrupt flag has been enabled and the edge sense circuit has been properly conditioned, the interrupt flag will be set on the next active transition of the interrupt input pin.

PIA PERIPHERAL INTERFACE LINES

The PIA provides two 8-bit bidirectional data buses and four interrupt/control lines for interfacing to peripheral devices.

Section A Peripheral Data (PA0-PA7) – Each of the peripheral data lines can be programmed to act as an input or output. This is accomplished by setting a "1" in the corresponding Data Direction Register bit for those lines which are to be outputs. A "0" in a bit of the Data Direction Register causes the corresponding peripheral data line to act as an input. During an MPU Read Peripheral Data Operation, the data on peripheral lines programmed to act as inputs appears directly on the corresponding MPU Data Bus lines. In the input mode, the internal pullup resistor on these lines represents a maximum of 1.5 standard TTL loads.

The data in Output Register A will appear on the data lines that are programmed to be outputs. A logical "1" written into the register will cause a "high" on the corresponding data line while a "0" results in a "low." Data in Output Register A may be read by an MPU "Read Peripheral Data A" operation when the corresponding lines are programmed as outputs. This data will be read property if the voltage on the peripheral data lines is greater than 2.0 volts for a logic "1" output and less than 0.8 volt for a logic "0" output. Loading the output lines such that the voltage on these lines does not reach full voltage causes the data transferred into the MPU on a Read operation to differ from that contained in the respective bit of Output Register A.

Section B Peripheral Data (PB0-PB7) – The peripheral data lines in the B Section of the PIA can be programmed to act as either inputs or outputs in a similar manner to PA0-PA7. They have three-state capabiity, allowing them to enter a high-impedance state when the peripheral data line is used as an input. In addition, data on the peripheral data lines

PB0-PB7 will be read properly from those lines programmed as outputs even if the voltages are below 2.0 volts for a "high" or above 0.8 V for a "low". As outputs, these lines are compatible with standard TTL and may also be used as a source of up to 1 milliampere at 1.5 volts to directly drive the base of a transistor switch.

Interrupt Input (CA1 and CB1) — Peripheral input lines CA1 and CB1 are input only lines that set the interrupt flags of the control registers. The active transition for these signals is also programmed by the two control registers.

Peripheral Control (CA2) — The peripheral control line CA2 can be programmed to act as an interrupt input or as a peripheral control output. As an output, this line is compatible with standard TTL; as an input the internal pullup resistor on this line represents 1.5 standard TTL loads. The function of this signal line is programmed with Control Register A.

Peripheral Control (CB2) — Peripheral Control line CB2 may also be programmed to act as an interrupt input or peripheral control output. As an input, this line has high input impedance and is compatible with standard TTL. As an output it is compatible with standard TTL and may also be used as a source of up to 1 milliampere at 1.5 volts to directly drive the base of a transistor switch. This line is programmed by Control Register B.

INTERNAL CONTROLS

INITIALIZATION

A $\overline{\text{RESET}}$ has the effect of zeroing all PIA registers. This will set PA0-PA7, PB0-PB7, CA2 and CB2 as inputs, and all interrupts disabled. The PIA must be configured during the restart program which follows the reset.

There are six locations within the PIA accessible to the MPU data bus: two Peripheral Registers, two Data Direction Registers, and two Control Registers. Selection of these locations is controlled by the RS0 and RS1 inputs together with bit 2 in the Control Register, as shown in Table B.1

Details of possible configurations of the Data Direction and Control Register are as follows:

TABLE B.1 INTERNAL ADDRESSING

		Control Register Bit		
RS1	RS0	CRA-2	CRB-2	Location Selected
0	0	1	X	Peripheral Register A
0	0	0	X	Data Direction Register A
0	1	X	X	Control Register A
1	0	X	1	Peripheral Register B
1	0	X	0	Data Direction Register B
1	1	X	X	Control Register B

X = Don't Care

PORT A-B HARDWARE CHARACTERISTICS

As shown in Figure 1-7, the MC6821 has a pair of I/O ports whose characteristics differ greatly. The A side is designed to drive CMOS logic to normal 30% to 70% levels, and incorporates an internal pullup device that remains connected even in the input mode. Because of this, the A side requires more drive current in the input mode than Port B. In contrast, the B side uses a normal three-state NMOS buffer which cannot pullup to CMOS levels without external resistors. The B side can drive extra loads such as Darlingtons without problem. When the PIA comes out of reset, the A port represents inputs with pullup resistors, whereas the B side (input mode also) will float high or low, depending upon the load connected to it.

Notice the differences between a Port A and Port B read operation when in the output mode. When reading Port A, the actual pin is read, whereas the B side read comes from an output latch, ahead of the actual pin.

CONTROL REGISTERS (CRA and CRB)

The two Control Registers (CRA and CRB) allow the MPU to control the operation of the four peripheral control lines CA1, CA2, CB1, and CB2. In addition they allow the MPU to enable the interrupt lines and monitor the status of the interrupt flags. Bits 0 through 5 of the two registers may be written or read by the MPU when the proper chip select and register select signals are applied. Bits 6 and 7 of the two registers are read only and are modified by external interrupts occurring on control lines CA1, CA2, CB1, or CB2. The format of the control words is shown in Figure B.3

DATA DIRECTION ACCESS CONTROL BIT (CRA-2 and CRB-2)

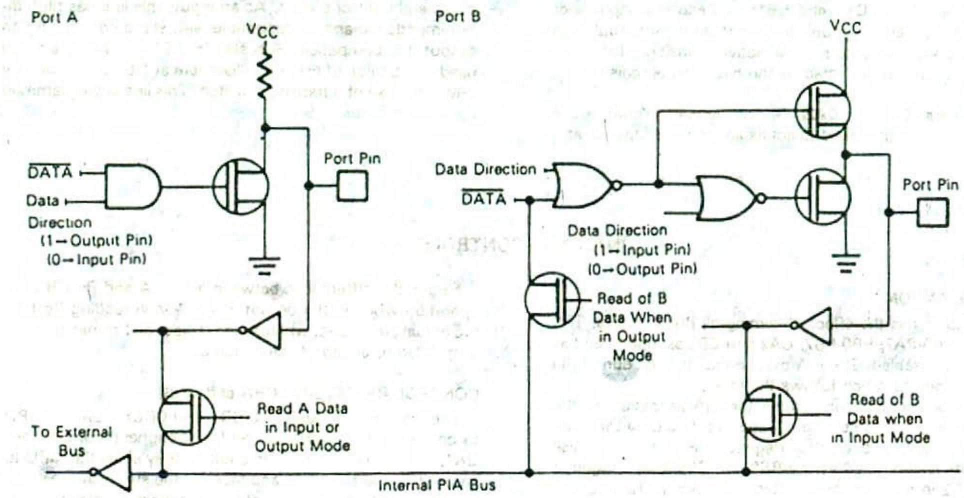
Bit 2, in each Control Register (CRA and CRB), determines selection of either a Peripheral Output Register or the corresponding Data Direction E Register when the proper register select signals are applied to RS0 and RS1. A "1" in bit 2 allows access of the Peripheral Interface Register, while a "0" causes the Data Direction Register to be addressed.

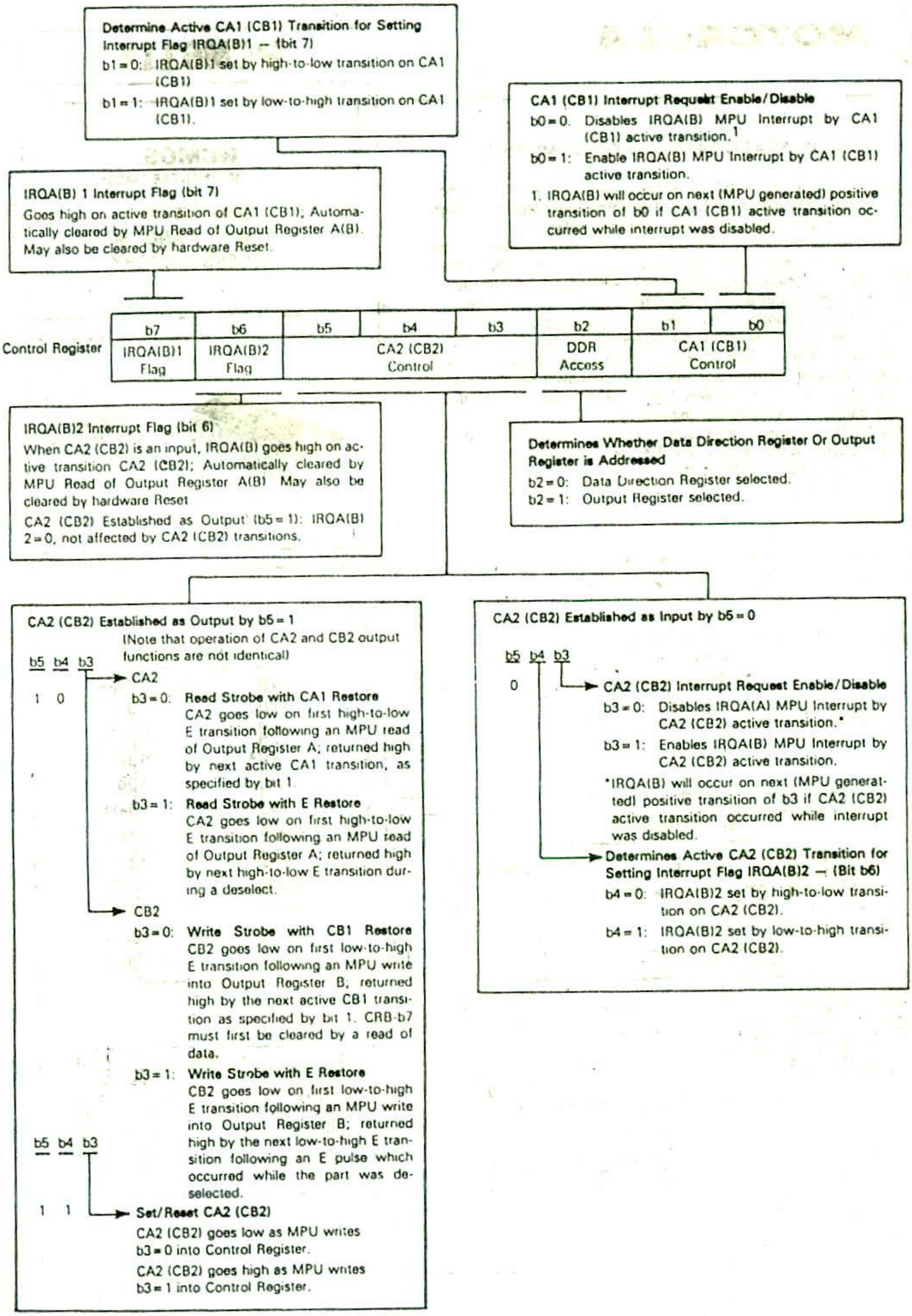
Interrupt Flags (CRA-6, CRA-7, CRB-6, and CRB-7) — The four interrupt flag bits are set by active transitions of signals on the four Interrupt and Peripheral Control lines when those lines are programmed to be inputs. These bits cannot be set directly from the MPU Data Bus and are reset indirectly by a Read Peripheral Data Operation on the appropriate section.

Control of CA2 and CB2 Peripheral Control Lines (CRA-3, CRA-4, CRA-5, CRB-3, CRB-4, and CRB-5) — Bits 3, 4, and 5 of the two control registers are used to control the CA2 and CB2 Peripheral Control lines. These bits determine if the control lines will be an interrupt input or an output control signal. If bit CRA-5 (CRB-5) is low, CA2 (CB2) is an interrupt input line similar to CA1 (CB1). When CRA-5 (CRB-5) is high, CA2 (CB2) becomes an output signal that may be used to control peripheral data transfers. When in the output mode, CA2 and CB2 have slightly different loading characteristics.

Control of CA1 and CB1 Interrupt Input Lines (CRA-0, CRB-1, CRA-1, and CRB-1) — The two lowest-order bits of the control registers are used to control the interrupt input lines CA1 and CB1. Bits CRA-0 and CRB-0 are used to enable the MPU interrupt signals $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$, respectively. Bits CRA-1 and CRB-1 determine the active transition of the interrupt input signals CA1 and CB1.

FIGURE B.2 PORT A AND PORT B EQUIVALENT CIRCUITS





Determine Active CA1 (CB1) Transition for Setting Interrupt Flag IRQA(B)1 — (bit 7)

b1 = 0: IRQA(B)1 set by high-to-low transition on CA1 (CB1)

b1 = 1: IRQA(B)1 set by low-to-high transition on CA1 (CB1).

CA1 (CB1) Interrupt Request Enable/Disable

b0 = 0. Disables IRQA(B) MPU Interrupt by CA1 (CB1) active transition.[1]

b0 = 1: Enable IRQA(B) MPU Interrupt by CA1 (CB1) active transition.

[1] IRQA(B) will occur on next (MPU generated) positive transition of b0 if CA1 (CB1) active transition occurred while interrupt was disabled.

IRQA(B) 1 Interrupt Flag (bit 7)

Goes high on active transition of CA1 (CB1); Automatically cleared by MPU Read of Output Register A(B). May also be cleared by hardware Reset.

	b7	b6	b5	b4	b3	b2	b1	b0
Control Register	IRQA(B)1 Flag	IRQA(B)2 Flag	CA2 (CB2) Control			DDR Access	CA1 (CB1) Control	

IRQA(B)2 Interrupt Flag (bit 6)

When CA2 (CB2) is an input, IRQA(B) goes high on active transition CA2 (CB2); Automatically cleared by MPU Read of Output Register A(B) May also be cleared by hardware Reset

CA2 (CB2) Established as Output (b5 = 1): IRQA(B) 2 = 0, not affected by CA2 (CB2) transitions.

Determines Whether Data Direction Register Or Output Register is Addressed

b2 = 0: Data Direction Register selected.

b2 = 1: Output Register selected.

CA2 (CB2) Established as Output by b5 = 1

(Note that operation of CA2 and CB2 output functions are not identical)

b5 b4 b3

1 0

→ CA2

b3 = 0: **Read Strobe with CA1 Restore**
CA2 goes low on first high-to-low E transition following an MPU read of Output Register A; returned high by next active CA1 transition, as specified by bit 1.

b3 = 1: **Read Strobe with E Restore**
CA2 goes low on first high-to-low E transition following an MPU read of Output Register A; returned high by next high-to-low E transition during a deselect.

→ CB2

b3 = 0: **Write Strobe with CB1 Restore**
CB2 goes low on first low-to-high E transition following an MPU write into Output Register B; returned high by the next active CB1 transition as specified by bit 1. CRB-b7 must first be cleared by a read of data.

b3 = 1: **Write Strobe with E Restore**
CB2 goes low on first low-to-high E transition following an MPU write into Output Register B; returned high by the next low-to-high E transition following an E pulse which occurred while the part was deselected.

b5 b4 b3

1 1 → **Set/Reset CA2 (CB2)**

CA2 (CB2) goes low as MPU writes b3 = 0 into Control Register.

CA2 (CB2) goes high as MPU writes b3 = 1 into Control Register.

CA2 (CB2) Established as Input by b5 = 0

b5 b4 b3

0

→ **CA2 (CB2) Interrupt Request Enable/Disable**

b3 = 0: Disables IRQA(A) MPU Interrupt by CA2 (CB2) active transition.*

b3 = 1: Enables IRQA(B) MPU Interrupt by CA2 (CB2) active transition.

*IRQA(B) will occur on next (MPU generatted) positive transition of b3 if CA2 (CB2) active transition occurred while interrupt was disabled.

→ **Determines Active CA2 (CB2) Transition for Setting Interrupt Flag IRQA(B)2 — (Bit b6)**

b4 = 0: IRQA(B)2 set by high-to-low transition on CA2 (CB2).

b4 = 1: IRQA(B)2 set by low-to-high transition on CA2 (CB2).

FIGURE B.3 CONTROL WORD FORMAT

MOTOROLA

MCM6116

HCMOS

(COMPLEMENTARY MOS)

2,048 × 8 BIT STATIC RANDOM ACCESS MEMORY

16K BIT STATIC RANDOM ACCESS MEMORY

The MCM6116 is a 16,384-bit Static Random Access Memory organized as 2048 words by 8 bits, fabricated using Motorola's high-performance silicon-gate CMOS (HCMOS) technology. It uses a design approach which provides the simple timing features associated with fully static memories and the reduced power associated with CMOS memories. This means low standby power without the need for clocks, nor reduced data rates due to cycle times that exceed access time.

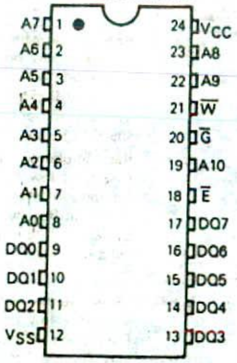
Chip Enable ($\overline{E}$) controls the power-down feature. It is not a clock but rather a chip control that affects power consumption. In less than a cycle time after Chip Enable ($\overline{E}$) goes high, the part automatically reduces its power requirements and remains in this low-power standby as long as the Chip Enable ($\overline{E}$) remains high. The automatic power-down feature causes no performance degradation.

The MCM6116 is in a 24-pin dual-in-line package with the industry standard JEDEC approved pinout and is pinout compatible with the industry standard 16K EPROM/ROM.

- Single +5 V Supply
- 2048 Words by 8-Bit Operation
- HCMOS Technology
- Fully Static: No Clock or Timing Strobe Required
- Maximum Access Time: MCM6116-12 – 120 ns
MCM6116-15 – 150 ns
MCM6116-20 – 200 ns
- Power Dissipation: 70 mA Maximum (Active)
15 mA Maximum (Standby-TTL Levels)
2 mA Maximum (Standby)
- Low Power Version Also Available – MCM61L16
- Low Voltage Data Retention (MCM61L16 Only): 50 μA Maximum

P SUFFIX
PLASTIC PACKAGE
CASE 709

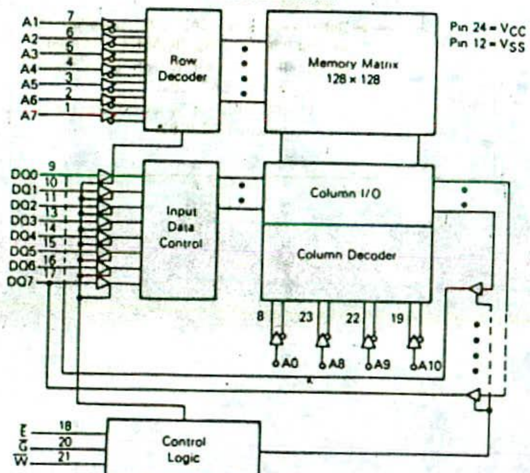
PIN ASSIGNMENTS



PIN NAMES

A0-A10	Address Input
DQ0-DQ7	Data Input/Output
$\overline{W}$	Write Enable
$\overline{G}$	Output Enable
$\overline{E}$	Chip Enable
V_{CC}	Power (+5 V)
V_{SS}	Ground

BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS (See Note)

Rating	Value	Unit
Temperature Under Bias	− 10 to + 80	°C
Voltage on Any Pin With Respect to V_{SS}	− 1.0 to + 7.0	V
DC Output Current	20	mA
Power Dissipation	1.2	Watt
Operating Temperature Range	0 to + 70	°C
Storage Temperature Range	− 65 to + 150	°C

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields, however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages to this high-impedance circuit.

NOTE: Permanent device damage may occur if ABSOLUTE MAXIMUM RATINGS are exceeded. Functional operation should be restricted to RECOMMENDED OPERATING CONDITIONS. Exposure to higher than recommended voltages for extended periods of time could affect device reliability.

DC OPERATING CONDITIONS AND CHARACTERISTICS

(Full operating voltage and temperature ranges unless otherwise noted.)

RECOMMENDED OPERATING CONDITIONS

Parameter	Symbol	Min	Typ	Max	Unit
Supply Voltage	V_{CC}	4.5	5.0	5.5	V
	V_{SS}	0	0	0	V
Input Voltage	V_{IH}	2.2	3.5	6.0	V
	V_{IL}	− 1.0*	–	0.8	V

*The device will withstand undershoots to the − 1.0 volt level with a maximum pulse width of 50 ns at the − 0.3 volt level. This is periodically sampled rather than 100% tested.

RECOMMENDED OPERATING CHARACTERISTICS

Parameter		Symbol	MCM6116			MCM61L16			Unit
			Min	Typ*	Max	Min	Typ*	Max	
Input Leakage Current (V_{CC} = 5.5 V, V_{in} = GND to V_{CC})		$\lvert I_{LI} \rvert$	–	–	1	–	–	1	μA
Output Leakage Current ($\overline{E}$ = V_{IH} or $\overline{G}$ = V_{IH} $V_{I/O}$ = GND to V_{CC})		$\lvert I_{LO} \rvert$	–	–	1	–	–	1	μA
Operating Power Supply Current ($\overline{E}$ = V_{IL}, $I_{I/O}$ = 0 mA)		I_{CC}	–	35	70	–	35	55	mA
Average Operating Current Minimum cycle, duty = 100%		I_{CC2}	–	35	70	–	35	55	mA
Standby Power	($\overline{E}$ = V_{IH})	I_{SB}	–	5	15	–	5	12	mA
Supply Current	($\overline{E} \geq V_{CC} - 0.2$ V, $V_{in} \geq V_{CC} - 0.2$ V or $V_{in} \leq 0.2$ V)	I_{SB1}	–	20	2000	–	4	100	μA
Output Low Voltage (I_{OL} = 2.1 mA)		V_{OL}	–	–	0.4	–	–	0.4	V
Output High Voltage (I_{OH} = − 1.0 mA)**		V_{OH}	2.4	–	–	2.4	–	–	V

*V_{CC} = 5 V, T_A = 25°C

**Also, output voltages are compatible with Motorola's new high-speed CMOS logic family if the same power supply voltage is used.

CAPACITANCE (f = 1.0 MHz, T_A = 25°C, periodically sampled rather than 100% tested.)

Characteristic	Symbol	Typ	Max	Unit
Input Capacitance except $\overline{E}$	C_{in}	3	5	pF
Input/Output Capacitance and $\overline{E}$ Input Capacitance	$C_{I/O}$	5	7	pF

MODE SELECTION

Mode	$\overline{E}$	$\overline{G}$	$\overline{W}$	V_{CC} Current	DQ
Standby	H	X	X	I_{SB}, I_{SB1}	High Z
Read	L	L	H	I_{CC}	Q
Write Cycle (1)	L	H	L	I_{CC}	D
Write Cycle (2)	L	L	L	I_{CC}	D

AC OPERATING CONDITIONS AND CHARACTERISTICS

(Full operating voltage and temperature unless otherwise noted.)

Input Pulse Levels	0 Volt to 3.5 Volts	Input and Output Timing Reference Levels	1.5 Volts
Input Rise and Fall Times	10 ns	Output Load	1 TTL Gate and $C_L = 100$ pF

READ CYCLE

Parameter	Symbol	MCM6116-12 MCM61L16-12		MCM6116-15 MCM61L16-15		MCM6116-20 MCM61L16-20		Unit
		Min	Max	Min	Max	Min	Max	
Address Valid to Address Don't Care (Cycle Time when Chip Enable is Held Active)	t_{AVAX}	120	–	150	–	200	–	ns
Chip Enable Low to Chip Enable High	t_{ELEH}	120	–	150	–	200	–	ns
Address Valid to Output Valid (Access)	t_{AVQV}	–	120	–	150	–	200	ns
Chip Enable Low to Output Valid (Access)	t_{ELQV}	–	120	–	150	–	200	ns
Address Valid to Output Invalid	t_{AVQX}	10	–	15	–	15	–	ns
Chip Enable Low to Output Invalid	t_{ELQX}	10	–	15	–	15	–	ns
Chip Enable High to Output High Z	t_{EHQZ}	0	40	0	50	0	60	ns
Output Enable to Output Valid	t_{GLQV}	–	80	–	100	–	120	ns
Output Enable to Output Invalid	t_{GLQX}	10	–	15	–	15	–	ns
Output Enable to Output High Z	t_{GLQZ}	0	40	0	50	0	60	ns
Address Invalid to Output Invalid	t_{AXQX}	10	–	15	–	15	–	ns
Address Valid to Chip Enable Low (Address Setup)	t_{AVEL}	0	–	0	–	0	–	ns
Chip Enable to Power-Up Time	t_{PU}	0	–	0	–	0	–	ns
Chip Disable to Power-Down Time	t_{PD}	–	30	–	30	–	30	ns

WRITE CYCLE

Parameter	Symbol	MCM6116-12 MCM61L16-12		MCM6116-15 MCM61L16-15		MCM6116-20 MCM61L16-20		Unit
		Min	Max	Min	Max	Min	Max	
Chip Enable Low to Write High	t_{ELWH}	70	–	90	–	120	–	ns
Address Valid to Write High	t_{AVWH}	105	–	120	–	140	–	ns
Address Valid to Write Low (Address Setup)	t_{AVWL}	20	–	20	–	20	–	ns
Write Low to Write High (Write Pulse Width)	t_{WLWH}	70	–	90	–	120	–	ns
Write High to Address Don't Care	t_{WHAX}	5	–	10	–	10	–	ns
Data Valid to Write High	t_{DVWH}	35	–	40	–	60	–	ns
Write High to Data Don't Care (Data Hold)	t_{WHDX}	5	–	10	–	10	–	ns
Write Low to Output High Z	t_{WLQZ}	0	50	0	60	0	60	ns
Write High to Output Valid	t_{WHQV}	5	–	10	–	10	–	ns
Output Disable to Output High Z	t_{GHQZ}	0	40	0	50	0	60	ns

TIMING PARAMETER ABBREVIATIONS

```
                                              t X X X X
      signal name from which interval is defined ─┘ │ │ │
           transition direction for first signal ───┘ │ │
        signal name to which interval is defined ─────┘ │
          transition direction for second signal ───────┘
```

The transition definitions used in this data sheet are:

H = transition to high
L = transition to low
V = transition to valid
X = transition to invalid or don't care
Z = transition to off (high impedance)

TIMING LIMITS

The table of timing values shows either a minimum or a maximum limit for each parameter. Input requirements are specified from the external system point of view. Thus, address setup time is shown as a minimum since the system must supply at least that much time (even though most devices do not require it). On the other hand, responses from the memory are specified from the device point of view. Thus, the access time is shown as a maximum since the device never provides data later than that time.

APPENDIX C

intel® 8085

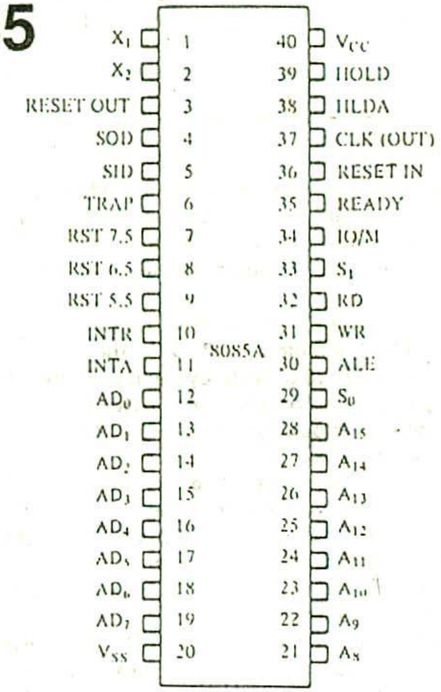


Figure C.1 8085 pinout

Figure C.1 shows 8085 pins and signals. The following table describes the function of each pin:

SYMBOL	FUNCTION
A_8–A_{15} (Output, three-state)	Address bus: The most significant 8 bits of the memory address or the 8 bits of the I/O address.
AD_{0-7} (Input/output, three-state)	Multiplexed address/data bus: Lower 8-bits of the memory address (or I/O address) appear on the bus during the first clock cycle (T state) of a machine cycle. It then becomes the data bus during the second and third clock cycles.
ALE (Output)	Address Latch Enable: It occurs during the first clock state of a machine cycle and enables the address to get latched into the on-chip latch.
S_0,S_1 and IO/$\overline{M}$ (Output)	Machine cycle status:

IO/$\overline{M}$	S_1	S_0	STATUS
0	0	1	Memory write
0	1	0	Memory read
1	0	1	I/O write

$IO/\overline{M}$	S_1	S_0	Status
1	1	0	I/O read
0	1	1	Op code fetch
1	1	1	Interrupt acknowledge
*	0	0	Halt
*	X	X	Hold
*	X	X	Reset

* – 3-state (high impedance)
X – unspecified

Symbol	Function
	S_1 can be used as an advanced $R/\overline{W}$ status. $IO/\overline{M}$, S_0, and S_1 become valid at the beginning of a machine cycle and remain stable throughout the cycle. The falling edge of ALE may be used to latch the state of these lines.
$\overline{RD}$ (Output, three-state)	READ control: A low level on $\overline{RD}$ indicates the selected memory or I/O device is to be read.
$\overline{WR}$ (Output, three-state)	WRITE control: A low level on $\overline{WR}$ indicates the data on the data bus is to be written into the selected memory or I/O location.
READY (Input)	If READY is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If READY is low, the CPU will wait an integral number of clock cycles for READY to go high before completing the read or write cycle.
HOLD (Input)	HOLD indicates that another master is requesting the use of the address and data buses. The CPU, upon receiving the hold request, will relinquish the use of the bus as soon as the completion of the current bus transfer. Internal processing can continue. The processor can regain the bus only after the HOLD is removed. When the HOLD is acknowledged, the address, data, $\overline{RD}$, $\overline{WR}$, and $IO/\overline{M}$ lines are three-stated.
HLDA (Output)	HOLD ACKNOWLEDGE: Indicates that the CPU has received the HOLD request and that it will relinquish the bus in the next clock cycle. HLDA goes low after the HOLD request is removed. The CPU takes the bus one-half clock cycle after HLDA goes low.
INTR (Input)	INTERRUPT REQUEST: Is used as a general-purpose interrupt. It is sampled only during the next to the last clock cycle of an instruction and during HOLD and HALT states. If it is active, the PC will be inhibited from incrementing and an $\overline{INTA}$ will be issued. During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine. The INTR is enabled and disabled by software. It is disabled by RESET and immediately after an interrupt is accepted.
INTA (Output)	INTERRUPT ACKNOWLEDGE: Is used instead of (and has the same timing as) $\overline{RD}$ during the instruction cycle after an INTR is accepted. It can be used to activate the 8259 interrupt chip or some other interrupt port.
RST5.5 RST6.5 RST7.5 (Inputs)	RESTART INTERRUPTS: These three inputs have the same timing as INTR except they cause an internal RESTART to be automatically inserted.
TRAP (Input)	Trap interrupt is a nonmaskable RESTART interrupt. It is recognized at the same time as INTR or RST5.5-7.5. It is unaffected by any mask or interrupt enable. It has the highest priority of any interrupt.
$\overline{RESET\ IN}$ (Input)	Sets the program counter to zero and resets the interrupt enable and HLDA flip-flops.
RESET OUT (Output)	Indicates CPU is being reset. Can be used as a system reset.

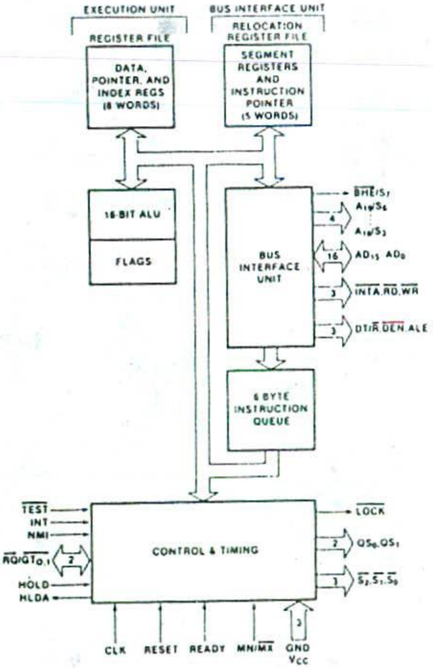
Symbol	Function
X_1,X_2 (Input)	X_1 and X_2 are connected to a crystal, LC, or RC network to drive the internal clock generator. X_1 can also be an external clock input from a logic gate. The input frequency is divided by 2 to give the processor's internal operating frequency.
CLK (Output)	Clock output for use as a system clock. The period of CLK is twice the X_1, X_2 input period.
SID (Input)	Serial Input Data line. The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.
SOD (Output)	Serial Output Data line. The output SOD is set or reset as specified by the SIM instruction.
V_{CC}	+5 V supply.
V_{SS}	Ground reference.

intel®

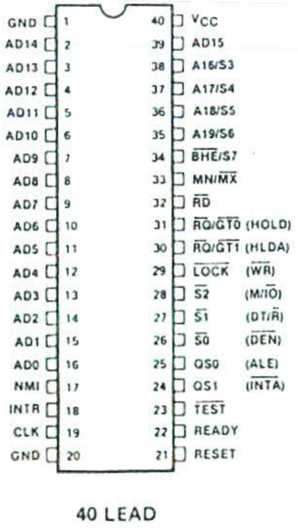
8086/8086-2/8086-4
16-BIT HMOS MICROPROCESSOR

- **Direct Addressing Capability to 1 MByte of Memory**
- **Assembly Language Compatible with 8080/8085**
- **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**
- **5 MHz Clock Rate (8 MHz for 8086-2) (4 MHz for 8086-4)**
- **MULTIBUS™ System Compatible Interface**

The Intel® 8086 is a new generation, high performance microprocessor implemented in N-channel, depletion load, silicon gate technology (HMOS), and packaged in a 40-pin CerDIP package. The processor has attributes of both 8- and 16-bit microprocessors. It addresses memory as a sequence of 8-bit bytes, but has a 16-bit wide physical path to memory for high performance.



8086 CPU Functional Block Diagram



8086 Pin Diagram

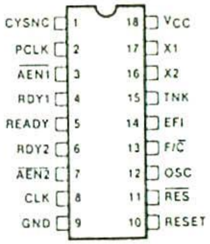
intel®

I8284
CLOCK GENERATOR AND DRIVER FOR 8086, 8088, 8089 PROCESSORS

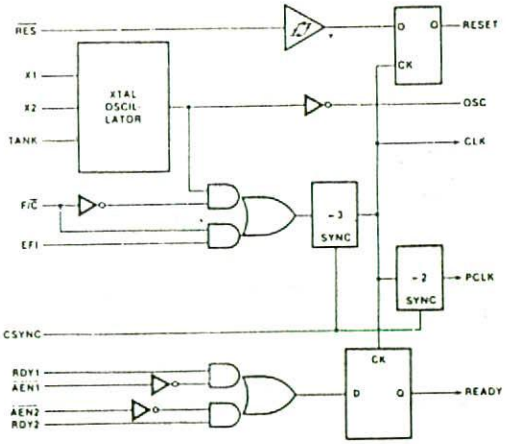
- Generates the System Clock for the 8086, 8088 and 8089
- Uses a Crystal or a TTL Signal for Frequency Source
- Single +5V Power Supply
- 18-Pin Package
- Generates System Reset Output from Schmitt Trigger Input
- Provides Local Ready and MULTIBUS™ Ready Synchronization
- Capable of Clock Synchronization with other 8284's
- Industrial Temperature Range −40° to +85°C

The I8284 is a bipolar clock generator/driver designed to provide clock signals for the 8086, 8088 & 8089 and peripherals. It also contains READY logic for operation with two MULTIBUS™ systems and provides the processors required READY synchronization and timing. Reset logic with hysteresis and synchronization is also provided.

I8284 PIN CONFIGURATION



I8284 BLOCK DIAGRAM



I8284 PIN NAMES

X1 X2	CONNECTIONS FOR CRYSTAL
TANK	USED WITH OVERTONE CRYSTAL
F/$\overline{C}$	CLOCK SOURCE SELECT
EFI	EXTERNAL CLOCK INPUT
CSYNC	CLOCK SYNCHRONIZATION INPUT
RDY1 RDY2	READY SIGNAL FROM TWO MULTIBUS™ SYSTEMS
$\overline{AEN1}$ $\overline{AEN2}$	ADDRESS ENABLED QUALIFIERS FOR RDY1,2
$\overline{RES}$	RESET INPUT
RESET	SYNCHRONIZED RESET OUTPUT
OSC	OSCILLATOR OUTPUT
CLK	MOS CLOCK FOR THE PROCESSOR
PCLK	TTL CLOCK FOR PERIPHERALS
READY	SYNCHRONIZED READY OUTPUT
Vcc	+5 VOLTS
GND	0 VOLTS

intel®

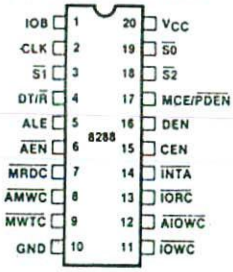
8288
BUS CONTROLLER
FOR 8086, 8088, 8089 PROCESSORS

- **Bipolar Drive Capability**
- **Provides Advanced Commands**
- **Provides Wide Flexibility in System Configurations**
- **3-State Command Output Drivers**
- **Configurable for Use with an I/O Bus**
- **Facilitates Interface to One or Two Multi-Master Busses**

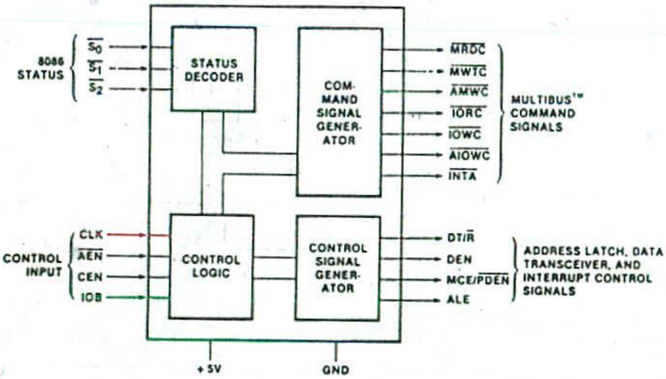
The Intel® 8288 Bus Controller is a 20-pin bipolar component for use with medium-to-large 8086 processing systems. The bus controller provides command and control timing generation as well as bipolar bus drive capability while optimizing system performance.

A strapping option on the bus controller configures it for use with a multi-master system bus and separate I/O bus.

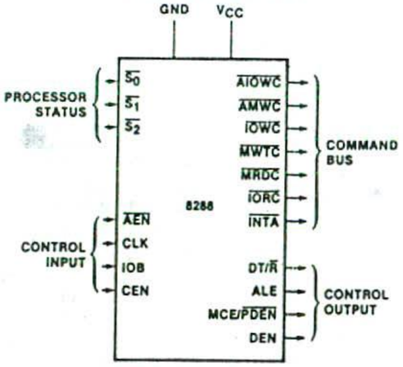
PIN CONFIGURATION



BLOCK DIAGRAM



FUNCTIONAL PIN-OUT



intel®

2716
16K (2K × 8) UV ERASABLE PROM

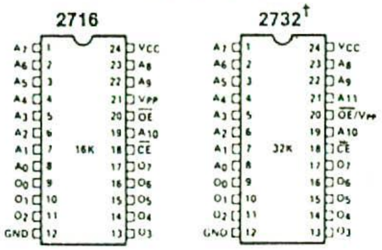
- **Fast Access Time**
  - **— 350 ns Max. 2716-1**
  - **— 390 ns Max. 2716-2**
  - **— 450 ns Max. 2716**
  - **— 650 ns Max. 2716-6**
- **Single +5V Power Supply**
- **Low Power Dissipation**
  - **— 525 mW Max. Active Power**
  - **— 132 mW Max. Standby Power**
- **Pin Compatible to Intel® 2732 EPROM**
- **Simple Programming Requirements**
  - **— Single Location Programming**
  - **— Programs with One 50 ms Pulse**
- **Inputs and Outputs TTL Compatible during Read and Program**
- **Completely Static**

The Intel® 2716 is a 16,384-bit ultraviolet erasable and electrically programmable read-only memory (EPROM). The 2716 operates from a single 5-volt power supply, has a static standby mode, and features fast single address location programming. It makes designing with EPROMs faster, easier and more economical.

The 2716, with its single 5-volt supply and with an access time up to 350 ns, is ideal for use with the newer high performance +5V microprocessors such as Intel's 8085 and 8086. The 2716 is also the first EPROM with a static standby mode which reduces the power dissipation without increasing access time. The maximum active power dissipation is 525 mW while the maximum standby power dissipation is only 132 mW, a 75% savings.

The 2716 has the simplest and fastest method yet devised for programming EPROMs — single pulse TTL level programming. No need for high voltage pulsing because all programming controls are handled by TTL signals. Program any location at any time—either individually, sequentially or at random, with the 2716's single address location programming. Total programming time for all 16,384 bits is only 100 seconds.

PIN CONFIGURATION



†Refer to 2732 data sheet for specifications

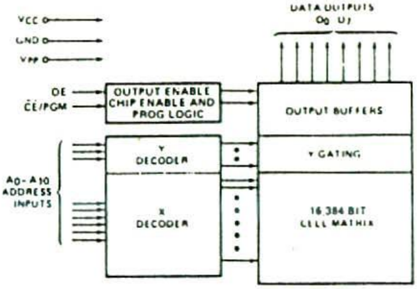
PIN NAMES

A_0- A_{10}	ADDRESSES
$\overline{CE}$/PGM	CHIP ENABLE/PROGRAM
$\overline{OE}$	OUTPUT ENABLE
O_0-O_7	OUTPUTS

MODE SELECTION

MODE \ PINS	$\overline{CE}$/PGM (18)	$\overline{OE}$ (20)	V_{PP} (21)	V_{CC} (24)	OUTPUTS (9-11, 13-17)
Read	V_{IL}	V_{IL}	+5	+5	D_{OUT}
Standby	V_{IH}	Don't Care	+5	+5	High Z
Program	Pulsed V_{IL} to V_{IH}	V_{IH}	+25	+5	D_{IN}
Program Verify	V_{IL}	V_{IL}	+25	+5	D_{OUT}
Program Inhibit	V_{IL}	V_{IH}	+25	+5	High Z

BLOCK DIAGRAM



2732
32K (4K x 8) UV ERASABLE PROM

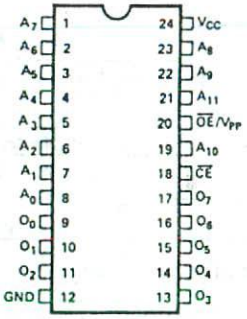
- **Fast Access Time:**
  - **— 450 ns Max. 2732**
  - **— 550 ns Max. 2732-6**
- **Single +5V ± 5% Power Supply**
- **Output Enable for MCS-85™ and MCS-86™ Compatibility**
- **Low Power Dissipation:**
  150mA Max. Active Current
  30mA Max. Standby Current
- **Pin Compatible to Intel® 2716 EPROM**
- **Completely Static**
- **Simple Programming Requirements**
  - **— Single Location Programming**
  - **— Programs with One 50ms Pulse**
- **Three-State Output for Direct Bus Interface**

The Intel® 2732 is a 32,768-bit ultraviolet erasable and electrically programmable read-only memory (EPROM). The 2732 operates from a single 5-volt power supply, has a standby mode, and features an output enable control. The total programming time for all bits is three and a half minutes. All these features make designing with the 2732 in microcomputer systems faster, easier, and more economical.

An important 2732 feature is the separate output control, Output Enable ($\overline{OE}$), from the Chip Enable control ($\overline{CE}$). The $\overline{OE}$ control eliminates bus contention in multiple bus microprocessor systems. Intel's Application Note AP-30 describes the microprocessor system implementation of the $\overline{OE}$ and $\overline{CE}$ controls on Intel's 2716 and 2732 EPROMs. AP-30 is available from Intel's Literature Department.

The 2732 has a standby mode which reduces the power dissipation without increasing access time. The maximum active current is 150mA, while the maximum standby current is only 30mA, an 80% savings. The standby mode is achieved by applying a TTL-high signal to the $\overline{CE}$ input.

PIN CONFIGURATION



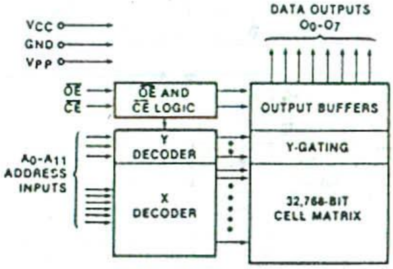
PIN NAMES

A_0-A_{11}	ADDRESSES
$\overline{CE}$	CHIP ENABLE
$\overline{OE}$	OUTPUT ENABLE
O_0-O_7	OUTPUTS

MODE SELECTION

PINS / MODE	$\overline{CE}$ (18)	$\overline{OE}/V_{PP}$ (20)	V_{CC} (24)	OUTPUTS (9-11,13-17)
Read	V_{IL}	V_{IL}	+5	D_{OUT}
Standby	V_{IH}	Don't Care	+5	High Z
Program	V_{IL}	V_{PP}	+5	D_{IN}
Program Verify	V_{IL}	V_{IL}	+5	D_{OUT}
Program Inhibit	V_{IH}	V_{PP}	+5	High Z

BLOCK DIAGRAM



intel®

8355/8355-2
16,384-BIT ROM WITH I/O

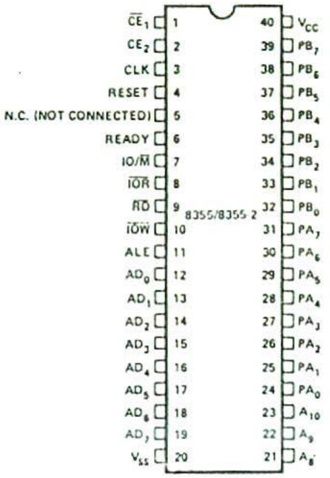
- 2048 Words × 8 Bits
- Single +5V Power Supply
- Directly compatible with 8085A and 8088 Microprocessors
- 2 General Purpose 8-Bit I/O Ports
- Each I/O Port Line Individually Programmable as Input or Output
- Multiplexed Address and Data Bus
- Internal Address Latch
- 40-Pin DIP

The Intel® 8355 is a ROM and I/O chip to be used in the 8085A and 8088 microprocessor systems. The ROM portion is organized as 2048 words by 8 bits. It has a maximum acess time of 400 ns to permit use with no wait states in the 8085A CPU.

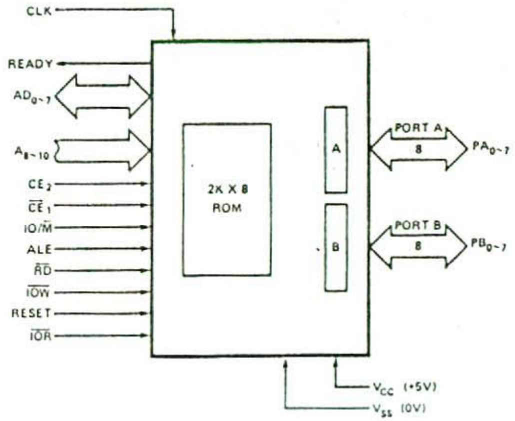
The I/O portion consists of 2 general purpose I/O ports. Each I/O port has 8 port lines and each I/O port line is individually programmable as input or output.

The 8355-2 has a 300ns access time for compatibility with the 8085A-2 and full speed 5 MHz 8088 microprocessors.

PIN CONFIGURATION



BLOCK DIAGRAM



8355/8355-2

Symbol	Function
ALE (Input)	When ALE (Address Latch Enable is high, AD$_{0-7}$, IO/$\overline{M}$, A$_{8-10}$, CE, and $\overline{CE}$ enter address latched. The signals (AD, IO/$\overline{M}$, A$_{8-10}$, CE, $\overline{CE}$) are latched in at the trailing edge of ALE.
AD$_{0-7}$ (Input)	Bidirectional Address/Data bus. The lower 8-bits of the ROM or I/O address are applied to the bus lines when ALE is high. During an I/O cycle, Port A or B are selected based on the latched value of AD$_0$. If $\overline{RD}$ or $\overline{IOR}$ is low when the latched chip enables are active, the output buffers present data on the bus.
A$_{8-10}$ (Input)	These are the high order bits of the ROM address. They do not affect I/O operations.
$\overline{CE_1}$ CE$_2$ (Input)	Chip Enable Inputs: $\overline{CE_1}$ is active low and CE$_2$ is active high. The 8355 can be accessed only when BOTH Chip Enables are active at the time the ALE signal latches them up. If either Chip Enable input is not active, the AD$_{0-7}$ and READY outputs will be in a high impedance state.
IO/$\overline{M}$ (Input)	If the latched IO/$\overline{M}$ is high when $\overline{RD}$ is low, the output data comes from an I/O port. If it is low the output data comes from the ROM.
$\overline{RD}$ (Input)	If the latched Chip Enables are active when $\overline{RD}$ goes low, the AD$_{0-7}$ output buffers are enabled and output either the selected ROM location or I/O port. When both $\overline{RD}$ and $\overline{IOR}$ are high, the AD$_{0-7}$ output buffers are 3-state.
$\overline{IOW}$ (Input)	If the latched Chip Enables are active, a low on $\overline{IOW}$ causes the output port pointed to by the latched value of AD$_0$ to be written with the data on AD$_{0-7}$. The state of IO/$\overline{M}$ is ignored.
CLK (Input)	The CLK is used to force the READY into its high impedance state after it has been forced low by $\overline{CE}$ low, CE high and ALE high.
READY (Output)	Ready is a 3-state output controlled by $\overline{CE_1}$, CE$_2$, ALE and CLK. READY is forced low when the Chip Enables are active during the time ALE is high, and remains low until the rising edge of the next CLK (see Figure 6).
PA$_{0-7}$ (Input/ Output)	These are general purpose I/O pins. Their input/output direction is determined by the contents of Data Direction Register (DDR). Port A is selected for write operations when the Chip Enables are active and $\overline{IOW}$ is low and a 0 was previously latched from AD$_0$. Read operation is selected by either $\overline{IOR}$ low and active Chip Enables and AD$_0$ low, or IO/$\overline{M}$ high, $\overline{RD}$ low, active chip enables, and AD$_0$ low.
PB$_{0-7}$ (Input/ Output)	This general purpose I/O port is identical to Port A except that it is selected by a 1 latched from AD$_0$.
RESET (Input)	An input high on RESET causes all pins in Port A and B to assume input mode.
$\overline{IOR}$ (Input)	When the Chip Enables are active, a low on $\overline{IOR}$ will output the selected I/O port onto the AD bus. $\overline{IOR}$ low performs the same function as the combination IO/$\overline{M}$ high and $\overline{RD}$ low. When $\overline{IOR}$ is not used in a system, $\overline{IOR}$ should be tied to V$_{CC}$ ("1").
V$_{CC}$	+5 volt supply.
V$_{SS}$	Ground Reference.

intel®

8755A/8755A-2
16,384-BIT EPROM WITH I/O

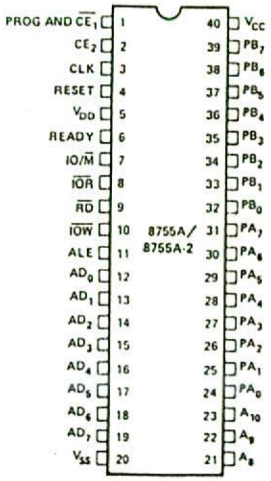
- **2048 Words × 8 Bits**
- **Single +5V Power Supply (V_{CC})**
- **Directly Compatible with 8085A and 8088 Microprocessors**
- **U.V. Erasable and Electrically Reprogrammable**
- **Internal Address Latch**
- **2 General Purpose 8-Bit I/O Ports**
- **Each I/O Port Line Individually Programmable as Input or Output**
- **Multiplexed Address and Data Bus**
- **40-Pin DIP**

The Intel® 8755A is an erasable and electrically reprogrammable ROM (EPROM) and I/O chip to be used in the 8085A and 8088 microprocessor systems. The EPROM portion is organized as 2048 words by 8 bits. It has a maximum access time of 450 ns to permit use with no wait states in an 8085A CPU.

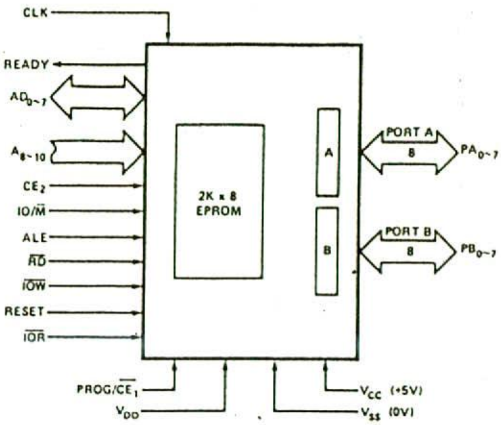
The I/O portion consists of 2 general purpose I/O ports. Each I/O port has 8 port lines, and each I/O port line is individually programmable as input or output.

The 8755A-2 is a high speed selected version of the 8755A compatible with the 5 MHz 8085A-2 and the full speed 5 MHz 8088.

PIN CONFIGURATION



BLOCK DIAGRAM



8755A FUNCTIONAL PIN DEFINITION

Symbol	Function
ALE (input)	When Address Latch Enable goes high, AD$_{0-7}$, IO/M, A$_{8-10}$, CE$_2$, and $\overline{CE_1}$ enter the address latches. The signals (AD, IO/M, A$_{8-10}$, CE) are latched in at the trailing edge of ALE.
AD$_{0-7}$ (input/output)	Bidirectional Address/Data bus. The lower 8-bits of the PROM or I/O address are applied to the bus lines when ALE is high. During an I/O cycle, Port A or B are selected based on the latched value of AD$_0$. If $\overline{RD}$ or $\overline{IOR}$ is low when the latched Chip Enables are active, the output buffers present data on the bus.
A$_{8-10}$ (input)	These are the high order bits of the PROM address. They do not affect I/O operations.
PROG/$\overline{CE_1}$ CE$_2$ (input)	Chip Enable Inputs: $\overline{CE_1}$ is active low and CE$_2$ is active high. The 8755A can be accessed only when *BOTH* Chip Enables are active at the time the ALE signal latches them up. If either Chip Enable input is not active, the AD$_{0-7}$ and READY outputs will be in a high impedance state. $\overline{CE_1}$ is also used as a programming pin. (See section on programming.)
IO/$\overline{M}$ (input)	If the latched IO/$\overline{M}$ is high when $\overline{RD}$ is low, the output data comes from an I/O port. If it is low the output data comes from the PROM.
$\overline{RD}$ (input)	If the latched Chip Enables are active when $\overline{RD}$ goes low, the AD$_{0-7}$ output buffers are enabled and output either the selected PROM location or I/O port. When both $\overline{RD}$ and $\overline{IOR}$ are high, the AD$_{0-7}$ output buffers are 3-stated.
$\overline{IOW}$ (input)	If the latched Chip Enables are active, a low on $\overline{IOW}$ causes the output port pointed to by the latched value of AD$_0$ to be written with the data on AD$_{0-7}$. The state of IO/$\overline{M}$ is ignored.
CLK (input)	The CLK is used to force the READY into its high impedance state after it has been forced low by $\overline{CE_1}$ low, CE$_2$ high, and ALE high.
READY (output)	READY is a 3-state output controlled by CE$_2$, $\overline{CE_1}$, ALE and CLK. READY is forced low when the Chip Enables are active during the time ALE is high, and remains low until the rising edge of the next CLK. (See Figure 6.)
PA$_{0-7}$ (input/output)	These are general purpose I/O pins. Their input/output direction is determined by the contents of Data Direction Register (DDR). Port A is selected for write operations when the Chip Enables are active and $\overline{IOW}$ is low and a 0 was previously latched from AD$_0$, AD$_1$. Read operation is selected by either $\overline{IOR}$ low and active Chip Enables and AD$_0$ and AD$_1$ low, *or* IO/$\overline{M}$ high, $\overline{RD}$ low, active Chip Enables, and AD$_0$ and AD$_1$ low.
PB$_{0-7}$ (input/output)	This general purpose I/O port is identical to Port A except that it is selected by a 1 latched from AD$_0$ and a 0 from AD$_1$.
RESET (input)	In normal operation, an input high on RESET causes all pins in Ports A and B to assume input mode (clear DDR register).
$\overline{IOR}$ (input)	When the Chip Enables are active, a low on $\overline{IOR}$ will output the selected I/O port onto the AD bus. $\overline{IOR}$ low performs the same function as the combination of IO/$\overline{M}$ high and $\overline{RD}$ low. When $\overline{IOR}$ is not used in a system, $\overline{IOR}$ should be tied to V$_{CC}$ ("1").
V$_{CC}$	+5 volt supply.
V$_{SS}$	Ground Reference.
V$_{DD}$	V$_{DD}$ is a programming voltage, and must be tied to +5V when the 8755A is being read. For programming, a high voltage is supplied with V$_{DD}$ = 25V, typical. (See section on programming.)

intel®

8155/8156/8155-2/8156-2
2048 BIT STATIC MOS RAM WITH I/O PORTS AND TIMER

- 256 Word x 8 Bits
- Single +5V Power Supply
- Completely Static Operation

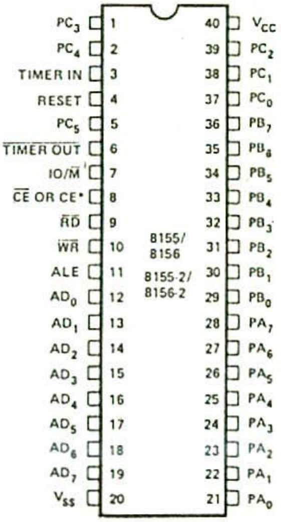
  Internal Address Latch
- 2 Programmable 8 Bit I/O Ports
- 1 Programmable 6-Bit I/O Port
- Programmable 14-Bit Binary Counter/ Timer
- Compatible with 8085A and 8088 CPU
- Multiplexed Address and Data Bus
- 40 Pin DIP

The 8155 and 89156 are RAM and I/O chips to be used in the 8085A and 8088 microprocessor systems. The RAM portion is designed with 2048 static cells organized as 256 x 8. They have a maximum access time of 400 ns to permit use with no wait states in 8085A CPU. The 8155-2 and 8156-2 have maximum access times of 330 ns for use with the 8085A-2 and the full speed 5 MHz 8088 CPU.

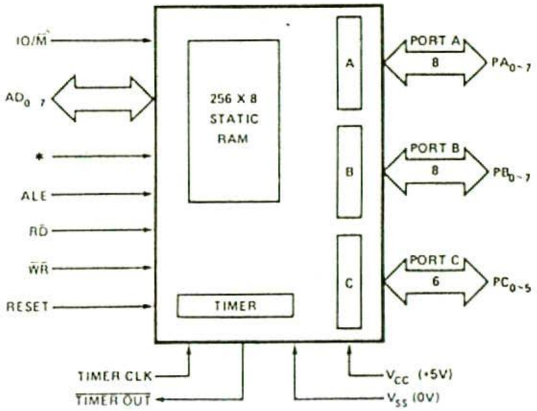
The I/O portion consists of three general purpose I/O ports. One of the three ports can be programmed to be status pins, thus allowing the other two ports to operate in handshake mode.

A 14-bit programmable counter/timer is jalso included on chip to provide either a square wave or terminal count pulse for the CPU system depending on timer mode.

PIN CONFIGURATION



BLOCK DIAGRAM



: 8155/8155-2 = $\overline{CE}$, 8156/8156-2 = CE

8155/8156 PIN FUNCTIONS

Symbol	Function
RESET (input)	Pulse provided by the 8085A to initialize the system (connect to 8085A RESET OUT). Input high on this line resets the chip and initializes the three I/O ports to input mode. The width of RESET pulse should typically be two 8085A clock cycle times.
AD$_{0-7}$ (input)	3-state Address/Data lines that interface with the CPU lower 8-bit Address/Data Bus. The 8-bit address is latched into the address latch inside the 8155/56 on the falling edge of ALE. The address can be either for the memory section or the I/O section depending on the IO/$\overline{\text{M}}$ input. The 8-bit data is either written into the chip or read from the chip, depending on the $\overline{\text{WR}}$ or $\overline{\text{RD}}$ input signal.
CE or $\overline{\text{CE}}$ (input)	Chip Enable: On the 8155, this pin is $\overline{\text{CE}}$ and is ACTIVE LOW. On the 8156, this pin is CE and is ACTIVE HIGH.
$\overline{\text{RD}}$ (input)	Read control: Input low on this line with the Chip Enable active enables and AD$_{0-7}$ buffers. If IO/$\overline{\text{M}}$ pin is low, the RAM content will be read out to the AD bus. Otherwise the content of the selected I/O port or command/status registers will be read to the AD bus.
$\overline{\text{WR}}$ (input)	Write control: Input low on this line with the Chip Enable active causes the data on the Address/Data bus to be written to the RAM or I/O ports and command/status register depending on IO/$\overline{\text{M}}$.
ALE (input)	Address Latch Enable: This control signal latches both the address on the AD$_{0-7}$ lines and the state of the Chip Enable and IO/$\overline{\text{M}}$ into the chip at the falling edge of ALE.
IO/$\overline{\text{M}}$ (input)	Selects memory if low and I/O and command/status registers if high.
PA$_{0-7}$(8) (input/output)	These 8 pins are general purpose I/O pins. The in/out direction is selected by programming the command register.
PB$_{0-7}$(8) (input/output)	These 8 pins are general purpose I/O pins. The in/out direction is selected by programming the command register.
PC$_{0-5}$(6) (input/output)	These 6 pins can function as either input port, output port, or as control signals for PA and PB. Programming is done through the command register. When PC$_{0-5}$ are used as control signals, they will provide the following: PC$_0$ — A INTR (Port A Interrupt) PC$_1$ — ABF (Port A Buffer Full) PC$_2$ — $\overline{\text{A STB}}$ (Port A Strobe) PC$_3$ — B INTR (Port B Interrupt) PC$_4$ — $\overline{\text{B BF}}$ (Port B Buffer Full) PC$_5$ — B STB (Port B Strobe)
TIMER IN (input)	Input to the counter-timer.
$\overline{\text{TIMER OUT}}$ (output)	Timer output. This output can be either a square wave or a pulse depending on the timer mode.
V$_{CC}$	+5 volt supply.
V$_{SS}$	Ground Reference.

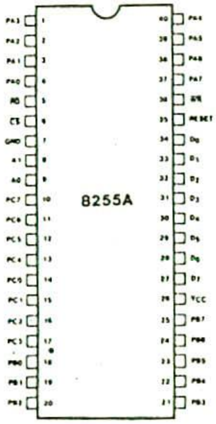
intel®

8255A/8255A-5
PROGRAMMABLE PERIPHERAL INTERFACE

- **MCS-85™ Compatible 8255A-5**
- **24 Programmable I/O Pins**
- **Completely TTL Compatible**
- **Fully Compatible with Intel® Microprocessor Families**
- **Improved Timing Characteristics**
- **Direct Bit Set/Reset Capability Easing Control Application Interface**
- **40-Pin Dual In-Line Package**
- **Reduces System Package Count**
- **Improved DC Driving Capability**

The Intel® 8255A is a general purpose programmable I/O device designed for use with Intel® microprocessors. It has 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bidirectional bus mode which uses 8 lines for a bidirectional bus, and 5 lines, borrowing one from the other group, for handshaking.

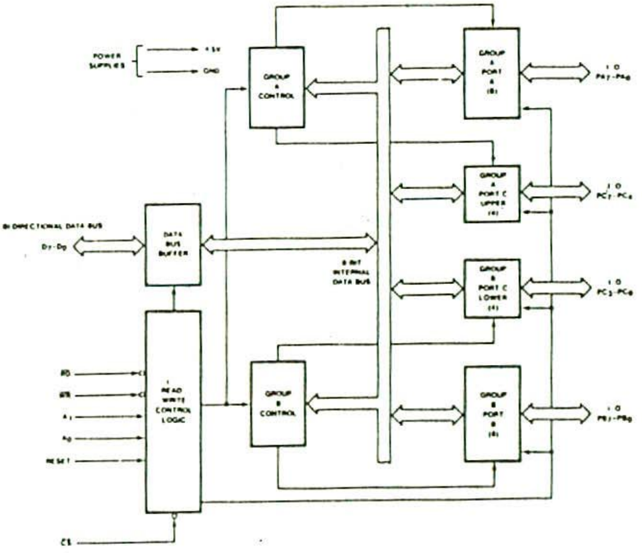
PIN CONFIGURATION



PIN NAMES

D_7-D_0	DATA BUS (BI DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	0 VOLTS

8255A BLOCK DIAGRAM



8279/8279-5
PROGRAMMABLE KEYBOARD/DISPLAY INTERFACE

- **Simultaneous Keyboard Display Operations**
- **Scanned Keyboard Mode**
- **Scanned Sensor Mode**
- **Strobed Input Entry Mode**
- **8-Character Keyboard FIFO**
- **2-Key Lockout or N-Key Rollover with Contact Debounce**
- **Dual 8- or 16-Numerical Display**
- **Single 16-Character Display**
- **Right or Left Entry 16-Byte Display RAM**
- **Mode Programmable from CPU**
- **Programmable Scan Timing**
- **Interrupt Output on Key Entry**
- **Available in EXPRESS**
  - **—Standard Temperature Range**
  - **—Extended Temperature Range**

The Intel® 8279 is a general purpose programmable keyboard and display I/O interface device designed for use with Intel® microprocessors. The keyboard portion can provide a scanned interface to a 64-contact key matrix. The keyboard portion will also interface to an array of sensors or a strobed interface keyboard, such as the hall effect and ferrite variety. Key depressions can be 2-key lockout or N-key rollover. Keyboard entries are debounced and strobed in an 8-character FIFO. If more than 8 characters are entered, overrun status is set. Key entries set the interrupt output line to the CPU.

The display portion provides a scanned display interface for LED, incandescent, and other popular display technologies. Both numeric and alphanumeric segment displays may be used as well as simple indicators. The 8279 has 16x8 display RAM which can be organized into dual 16x4. The RAM can be loaded or interrogated by the CPU. Both right entry, calculator and left entry typewriter display formats are possible. Both read and write of the display RAM can be done with auto-increment of the display RAM address.

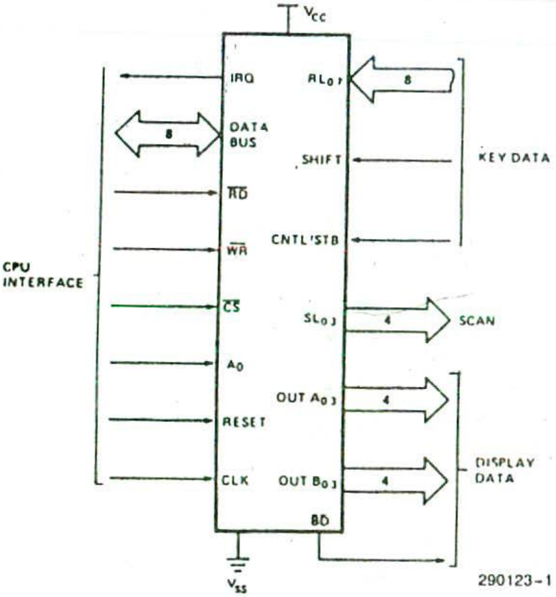


Figure 1. Logic Symbol

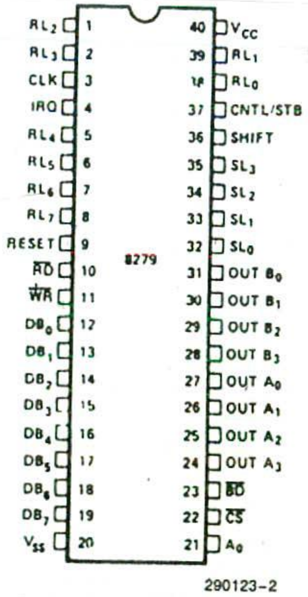


Figure 2. Pin Configuration

September 1987
Order Number: 290123-002

HARDWARE DESCRIPTION

The 8279 is packaged in a 40 pin DIP. The following is a functional description of each pin.

Table 1. Pin Description

Symbol	Pin No.	Name and Function
DB_0-DB_7	19–12	**BI-DIRECTIONAL DATA BUS:** All data and commands between the CPU and the 8279 are transmitted on these lines.
CLK	3	**CLOCK:** Clock from system used to generate internal timing.
RESET	9	**RESET:** A high signal on this pin resets the 8279. After being reset the 8279 is placed in the following mode: 1) 16 8-bit character display—left entry. 2) Encoded scan keyboard—2 key lockout. Along with this the program clock prescaler is set to 31.
CS	22	**CHIP SELECT:** A low on this pin enables the interface functions to receive or transmit.
A_0	21	**BUFFER ADDRESS:** A high on this line indicates the signals in or out are interpreted as a command or status. A low indicates that they are data.
$\overline{RD}$, $\overline{WR}$	10–11	**INPUT/OUTPUT READ AND WRITE:** These signals enable the data buffers to either send data to the external bus or receive it from the external bus.
IRQ	4	**INTERRUPT REQUEST:** In a keyboard mode, the interrupt line is high when there is data in the FIFO/Sensor RAM. The interrupt line goes low with each FIFO/Sensor RAM read and returns high if there is still information in the RAM. In a sensor mode, the interrupt line goes high whenever a change in a sensor is detected.
V_{SS}, V_{CC}	20, 40	**GROUND AND POWER SUPPLY PINS.**
SL_0-SL_3	32–35	**SCAN LINES:** Scan lines which are used to scan the key switch or sensor matrix and the display digits. These lines can be either encoded (1 of 16) or decoded (1 of 4).
RL_0-RL_7	38, 39, 1, 2, 5–8	**RETURN LINE:** Return line inputs which are connected to the scan lines through the keys or sensor switches. They have active internal pullups to keep them high until a switch closure pulls one low. They also serve as an 8-bit input in the Strobed Input mode.
SHIFT	36	**SHIFT:** The shift input status is stored along with the key position on key closure in the Scanned Keyboard modes. It has an active internal pullup to keep it high until a switch closure pulls it low.
CNTL/STB	37	**CONTROL/STROBED INPUT MODE:** For keyboard modes this line is used as a control input and stored like status on a key closure. The line is also the strobe line that enters the data into the FIFO in the Strobed Input mode. (Rising Edge). It has an active internal pullup to keep it high until a switch closure pulls it low.
OUT A_0–OUT A_3 OUT B_0–OUT B_3	27-24 31–28	**OUTPUTS:** These two ports are the outputs for the 16 x 4 display refresh registers. The data from these outputs is synchronized to the scan lines (SL_0–SL_3) for multiplexed digit displays. The two 4 bit ports may be blanked independently. These two ports may also be considered as one 8-bit port.
$\overline{BD}$	23	**BLANK DISPLAY:** This output is used to blank the display during digit switching or by a display blanking command.

APPENDIX D
MC68000 INSTRUCTION EXECUTION TIMES

D.1 INTRODUCTION

This Appendix contains listings of the instruction execution times in terms of external clock (CLK) periods. In this data, it is assumed that both memory read and write cycle times are four clock periods. A longer memory cycle will cause the generation of wait states which must be added to the total instruction time.

The number of bus read and write cycles for each instruction is also included with the timing data. This data is enclosed in parenthesis following the number of clock periods and is shown as: (r/w) where r is the number of read cycles and w is the number of write cycles included in the clock period number. Recalling that either a read or write cycle requires four clock periods, a timing number given as 18(3/1) relates to 12 clock periods for the three read cycles, plus 4 clock periods for the one write cycle, plus 2 cycles required for some internal function of the processor.

NOTE

> The number of periods includes instruction fetch and all applicable operand fetches and stores.

D.2 OPERAND EFFECTIVE ADDRESS CALCULATION TIMING

Table D-1 lists the number of clock periods required to compute an instruction's effective address. It includes fetching of any extension words, the address computation, and fetching of the memory operand. The number of bus read and write cycles is shown in parenthesis as (r/w). Note there are no write cycles involved in processing the effective address.

Table D-1. Effective Address Calculation Times

Addressing Mode		Byte, Word	Long
	Register		
Dn	Data Register Direct	0(0/0)	0(0/0)
An	Address Register Direct	0(0/0)	0(0/0)
	Memory		
(An)	Address Register Indirect	4(1/0)	8(2/0)
(An)+	Address Register Indirect with Postincrement	4(1/0)	8(2/0)
−(An)	Address Register Indirect with Predecrement	6(1/0)	10(2/0)
d(An)	Address Register Indirect with Displacement	8(2/0)	12(3/0)
d(An, ix)*	Address Register Indirect with Index	10(2/0)	14(3/0)
xxx.W	Absolute Short	8(2/0)	12(3/0)
xxx.L	Absolute Long	12(3/0)	16(4/0)
d(PC)	Program Counter with Displacement	8(2/0)	12(3/0)
d(PC, ix)*	Program Counter with Index	10(2/0)	14(3/0)
#xxx	Immediate	4(1/0)	8(2/0)

*The size of the index register (ix) does not affect execution time.

D.3 MOVE INSTRUCTION EXECUTION TIMES

Tables D-2 and D-3 indicate the number of clock periods for the move instruction. This data includes instruction fetch, operand reads, and operand writes. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table D-2. Move Byte and Word Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An)+	−(An)	d(An)	d(An, ix)*	xxx.W	xxx.L
Dn	4(1/0)	4(1/0)	8(1/1)	8(1/1)	8(1/1)	12(2/1)	14(2/1)	12(2/1)	16(3/1)
An	4(1/0)	4(1/0)	8(1/1)	8(1/1)	8(1/1)	12(2/1)	14(2/1)	12(2/1)	16(3/1)
(An)	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)
(An)+	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)
−(An)	10(2/0)	10(2/0)	14(2/1)	14(2/1)	14(2/1)	18(3/1)	20(3/1)	18(3/1)	22(4/1)
d(An)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
d(An, ix)*	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	22(4/1)	24(4/1)	22(4/1)	26(5/1)
xxx.W	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
xxx.L	16(4/0)	16(4/0)	20(4/1)	20(4/1)	20(4/1)	24(5/1)	26(5/1)	24(5/1)	28(6/1)
d(PC)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
d(PC, ix)*	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	22(4/1)	24(4/1)	22(4/1)	26(5/1)
#xxx	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)

*The size of the index register (ix) does not affect execution time.

Table D-3. Move Long Instruction Execution Times

Source	Destination								
	Dn	An	(An)	(An)+	−(An)	d(An)	d(An, ix)*	xxx.W	xxx.L
Dn	4(1/0)	4(1/0)	12(1/2)	12(1/2)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
An	4(1/0)	4(1/0)	12(1/2)	12(1/2)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
(An)	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
(An)+	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
−(An)	14(3/0)	14(3/0)	22(3/2)	22(3/2)	22(3/2)	26(4/2)	28(4/2)	26(4/2)	30(5/2)
d(An)	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(6/2)
d(An, ix)*	18(4/0)	18(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
xxx.W	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(6/2)
xxx.L	20(5/0)	20(5/0)	28(5/2)	28(5/2)	28(5/2)	32(6/2)	34(6/2)	32(6/2)	36(7/2)
d(PC)	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(5/2)
d(PC, ix)*	18(4/0)	18(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
#xxx	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)

*The size of the index register (ix) does not affect execution time.

D.4 STANDARD INSTRUCTION EXECUTION TIMES

The number of clock periods shown in Table D-4 indicates the time required to perform the operations, store the results, and read the next instruction. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Table D-4 the headings have the following meanings: An = address register operand, Dn = data register operand, ea = an operand specified by an effective address, and M = memory effective address operand.

Table D-4. Standard Instruction Execution Times

Instruction	Size	op<ea>, An†	op<ea>, Dn	op Dn, <M>
ADD	Byte, Word	8(1/0) +	4(1/0) +	8(1/1) +
	Long	6(1/0) + • •	6(1/0) + • •	12(1/2) +
AND	Byte, Word	–	4(1/0) +	8(1/1) +
	Long	–	6(1/0) + • •	12(1/2) +
CMP	Byte, Word	6(1/0) +	4(1/0) +	–
	Long	6(1/0) +	6(1/0) +	–
DIVS	–	–	158(1/0) + •	–
DIVU	–	–	140(1/0) + •	–
EOR	Byte, Word	–	4(1/0) • • •	8(1/1) +
	Long	–	8(1/0) • • •	12(1/2) +
MULS	–	–	70(1/0) + •	–
MULU	–	–	70(1/0) + •	–
OR	Byte, Word	–	4(1/0) +	8(1/1) +
	Long	–	6(1/0) + • •	12(1/2) +
SUB	Byte, Word	8(1/0) +	4(1/0) +	8(1/1) +
	Long	6(1/0) + • •	6(1/0) + • •	12(1/2) +

NOTES:

+ add effective address calculation time
† word or long only
• indicates maximum value
• • The base time of six clock periods is increased to eight if the effective address mode is register direct or immediate (effective address time should also be added).
• • • Only available effective address mode is data register direct.
DIVS, DIVU – The divide algorithm used by the MC68000 provides less than 10% difference between the best and worst case timings.
MULS, MULU – The multiply algorithm requires 38 + 2n clocks where n is defined as:
MULU: n = the number of ones in the <ea>
MULS: n = concatanate the <ea> with a zero as the LSB; n is the resultant number of 10 or 01 patterns in the 17-bit source; i.e., worst case happens when the source is $5555.

D.5 IMMEDIATE INSTRUCTION EXECUTION TIMES

The number of clock periods shown in Table D-5 includes the time to fetch immediate operands, perform the operations, store the results, and read the next operation. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Table D-5, the headings have the following meanings: # = immediate operand, Dn = data register operand, An = address register operand, and M = memory operand. SR = status register.

Table D-5. Immediate Instruction Execution Times

Instruction	Size	op #, Dn	op #, An	op #, M
ADDI	Byte, Word	8(2/0)	–	12(2/1) +
	Long	16(3/0)	–	20(3/2) +
ADDQ	Byte, Word	4(1/0)	8(1/0)•	8(1/1) +
	Long	8(1/0)	8(1/0)	12(1/2) +
ANDI	Byte, Word	8(2/0)	–	12(2/1) +
	Long	16(3/0)	–	20(3/1) +
CMPI	Byte, Word	8(2/0)	–	8(2/0) +
	Long	14(3/0)	–	12(3/0) +
EORI	Byte, Word	8(2/0)	–	12(2/1) +
	Long	16(3/0)	–	20(3/2) +
MOVEQ	Long	4(1/0)	–	–
ORI	Byte, Word	8(2/0)	–	12(2/1) +
	Long	16(3/0)	–	20(3/2) +
SUBI	Byte, Word	8(2/0)	–	12(2/1) +
	Long	16(3/0)	–	20(3/2) +
SUBQ	Byte, Word	4(1/0)	8(1/0)•	8(1/1) +
	Long	8(1/0)	8(1/0)	12(1/2) +

+ add effective address calculation time

• word only

D.6 SINGLE OPERAND INSTRUCTION EXECUTION TIMES

Table D-6 indicates the number of clock periods for the single operand instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table D-6. Single Operand Instruction Execution Times

Instruction	Size	Register	Memory
CLR	Byte, Word	4(1/0)	8(1/1)+
	Long	6(1/0)	12(1/2)+
NBCD	Byte	6(1/0)	8(1/1)+
NEG	Byte, Word	4(1/0)	8(1/1)+
	Long	6(1/0)	12(1/2)+
NEGX	Byte, Word	4(1/0)	8(1/1)+
	Long	6(1/0)	12(1/2)+
NOT	Byte, Word	4(1/0)	8(1/1)+
	Long	6(1/0)	12(1/2)+
Scc	Byte, False	4(1/0)	8(1/1)+
	Byte, True	6(1/0)	8(1/1)+
TAS	Byte	4(1/0)	10(1/1)+
TST	Byte, Word	4(1/0)	4(1/0)+
	Long	4(1/0)	4(1/0)+

+ add effective address calculation time

D.7 SHIFT/ROTATE INSTRUCTION EXECUTION TIMES

Table D-7 indicates the number of clock periods for the shift and rotate instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table D-7. Shift/Rotate Instruction Execution Times

Instruction	Size	Register	Memory
ASR, ASL	Byte, Word	6 + 2n(1/0)	8(1/1)+
	Long	8 + 2n(1/0)	–
LSR, LSL	Byte, Word	6 + 2n(1/0)	8(1/1)+
	Long	8 + 2n(1/0)	–
ROR, ROL	Byte, Word	6 + 2n(1/0)	8(1/1)+
	Long	8 + 2n(1/0)	–
ROXR, ROXL	Byte, Word	6 + 2n(1/0)	8(1/1)+
	Long	8 + 2n(1/0)	–

+ add effective address calculation time
n is the shift count

D.8 BIT MANIPULATION INSTRUCTION EXECUTION TIMES

Table D-8 indicates the number of clock periods required for the bit manipulation instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table D-8. Bit Manipulation Instruction Execution Times

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Byte	–	8(1/1)+	–	12(2/1)+
	Long	8(1/0)•	–	12(2/0)•	–
BCLR	Byte	–	8(1/1)+	–	12(2/1)+
	Long	10(1/0)•	–	14(2/0)•	–
BSET	Byte	–	8(1/1)+	–	12(2/1)+
	Long	8(1/0)•	–	12(2/0)•	–
BTST	Byte	–	4(1/0)+	–	8(2/0)+
	Long	6(1/0)	–	10(2/0)	–

+ add effective address calculation time
• indicates maximum value

D.9 CONDITIONAL INSTRUCTION EXECUTION TIMES

Table D-9 indicates the number of clock periods required for the conditional instructions. The number of bus read and write cycles is indicated in parenthesis as (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table D-9. Conditional Instruction Execution Times

Instruction	Displacement	Branch Taken	Branch Not Taken
B$_{CC}$	Byte	10(2/0)	8(1/0)
	Word	10(2/0)	12(2/0)
BRA	Byte	10(2/0)	–
	Word	10(2/0)	–
BSR	Byte	18(2/2)	–
	Word	18(2/2)	–
DB$_{CC}$	CC true	–	12(2/0)
	CC false	10(2/0)	14(3/0)

+ add effective address calculation time
• indicates maximum value

D.10 JMP, JSR, LEA, PEA, AND MOVEM INSTRUCTION EXECUTION TIMES

Table D-10 indicates the number of clock periods required for the jump, jump-to-subroutine, load effective address, push effective address, and move multiple registers instructions. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table D-10. JMP, JSR, LEA, PEA, and MOVEM Instruction Execution Times

Instr	Size	(An)	(An) +	– (An)	d(An)	d(An, ix) +	xxx.W	xxx.L	d(PC)	d(PC, ix)*
JMP	–	8(2/0)	–	–	10(2/0)	14(3/0)	10(2/0)	12(3/0)	10(2/0)	14(3/0)
JSR	–	16(2/2)	–	–	18(2/2)	22(2/2)	18(2/2)	20(3/2)	18(2/2)	22(2/2)
LEA	–	4(1/0)	–	–	8(2/0)	12(2/0)	8(2/0)	12(3/0)	8(2/0)	12(2/0)
PEA	–	12(1/2)	–	–	16(2/2)	20(2/2)	16(2/2)	20(3/2)	16(2/2)	20(2/2)
MOVEM M → R	Word	12 + 4n (3 + n/0)	12 + 4n (3 + n/0)	–	16 + 4n (4 + n/0)	18 + 4n (4 + n/0)	16 + 4n (4 + n/0)	20 + 4n (5 + n/0)	16 + 4n (4 + n/0)	18 + 4n (4 + n/0)
	Long	12 + 8n (3 + 2n/0)	12 + 8n (3 + 2n/0)	–	16 + 8n (4 + 2n/0)	18 + 8n (4 + 2n/0)	16 + 8n (4 + 2n/0)	20 + 8n (5 + 2n/0)	16 + 8n (4 + 2n/0)	18 + 8n (4 + 2n/0)
MOVEM R → M	Word	8 + 4n (2/n)	–	8 + 4n (2/n)	12 + 4n (3/n)	14 + 4n (3/n)	12 + 4n (3/n)	16 + 4n (4/n)	– –	– –
	Long	8 + 8n (2/2n)	– –	8 + 8n (2/2n)	12 + 8n (3/2n)	14 + 8n (3/2n)	12 + 8n (3/2n)	16 + 8n (4/2n)	– –	– –

n is the number of registers to move

* is the size of the index register (ix) does not affect the instruction's execution time

D 11 MULTI-PRECISION INSTRUCTION EXECUTION TIMES

Table D-11 indicates the number of clock periods for the multi-precision instructions. The number of clock periods includes the time to fetch both operands, peform the operations, store the results, and read the next instructions. The number of read and write cycles is shown in parenthesis as (r/w).

In Table D-11, the headings have the following meanings: Dn = data register operand and M = memory operand.

Table D-11. Multi-Precision Instruction Execution Times

Instruction	Size	op Dn, Dn	op M, M
ADDX	Byte, Word	4(1/0)	18(3/1)
	Long	8(1/0)	30(5/2)
CMPM	Byte, Word	–	12(3/0)
	Long	–	20(5/0)
SUBX	Byte, Word	4(1/0)	18(3/1)
	Long	8(1/0)	30(5/2)
ABCD	Byte	6(1/0)	18(3/1)
SBCD	Byte	6(1/0)	18(3/1)

D.12 MISCELLANEOUS INSTRUCTION EXECUTION TIMES

Tables D-12 and D-13 indicate the number of clock periods for the following miscellaneous instructions. The number of bus read and write cycles is shown in parenthesis as (r/w). The number of clock periods plus the number of read and write cycles must be added to those of the effective address calculation where indicated.

Table D-12. Miscellaneous Instruction Execution Times

Instruction	Size	Register	Memory
ANDI to CCR	Byte	20(3/0)	–
ANDI to SR	Word	20(3/0)	–
CHK	–	10(1/0) +	–
EORI to CCR	Byte	20(3/0)	–
EORI to SR	Word	20(3/0)	–
ORI to CCR	Byte	20(3/0)	–
ORI to SR	Word	20(3/0)	–
MOVE from SR	–	6(1/0)	8(1/1) +
MOVE to CCR	–	12(2/0)	12(2/0) +
MOVE to SR	–	12(2/0)	12(2/0) +
EXG	–	6(1/0)	–
EXT	Word	4(1/0)	–
	Long	4(1/0)	–
LINK	–	16(2/2)	–
MOVE from USP	–	4(1/0)	–
MOVE to USP	–	4(1/0)	–
NOP	–	4(1/0)	–
RESET	–	132(1/0)	–
RTE	–	20(5/0)	–
RTR	–	20(5/0)	–
RTS	–	16(4/0)	–
STOP	–	4(0/0)	–
SWAP	–	4(1/0)	–
TRAPV	–	4(1/0)	–
UNLK	–	12(3/0)	–

+ add effective address calculation time

Table D-13. Move Peripheral Instruction Execution Times

Instruction	Size	Register → Memory	Memory → Register
MOVEP	Word	16(2/2)	16(4/0)
	Long	24(2/4)	24(6/0)

D.13 EXCEPTION PROCESSING EXECUTION TIMES

Table D-14 indicates the number of clock periods for exception processing. The number of clock periods includes the time for all stacking, the vector fetch, and the fetch of the first two instruction words of the handler routine. The number of bus read and write cycles is shown in parenthesis as (r/w).

Table D-14. Exception Processing Execution Times

Exception	Periods
Address Error	50(4/7)
Bus Error	50(4/7)
CHK Instruction	44(5/4)+
Divide by Zero	42(5/4)
Illegal Instruction	34(4/3)
Interrupt	44(5/3)*
Privilege Violation	34(4/3)
$\overline{\text{RESET}}$**	40(6/0)
Trace	34(4/3)
TRAP Instruction	38(4/4)
TRAPV Instruction	34(4/3)

+ add effective address calculation time

*The interrupt acknowledge cycle is assumed to take four clock periods.

**Indicates the time from when $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ are first sampled as negated to when instruction execution starts

APPENDIX E

8086 INSTRUCTION SET REFERENCE DATA

AAA	AAA (no operands) ASCII adjust for addition			Flags O D I T S Z A P C U U U X U X
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	4	—	1	AAA

AAD	AAD (no operands) ASCII adjust for division			Flags O D I T S Z A P C U X X U X U
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	60	—	2	AAD

AAM	AAM (no operands) ASCII adjust for multiply			Flags O D I T S Z A P C U X X U X U
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	83	—	1	AAM

AAS	AAS (no operands) ASCII adjust for subtraction			Flags O D I T S Z A P C U U U X U X
Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	4	—	1	AAS

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

ADC	**ADC** destination,source Add with carry	Flags	O	D	I	T	S	Z	A	P	C
			X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
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	**ADD** destination,source Addition	Flags	O	D	I	T	S	Z	A	P	C
			X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
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	**AND** destination,source Logical and	Flags	O	D	I	T	S	Z	A	P	C
			0				X	X	U	X	0

Operands	Clocks	Transfers*	Bytes	Coding Example
register, register	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,0F0H
memory, immediate	17+EA	2	3-6	AND BETA, 01H
accumulator, immediate	4	—	2-3	AND AX, 01010000B

CALL	**CALL** target Call a procedure	Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Examples
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]

CBW	**CBW** (no operands) Convert byte to word	Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	CBW

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

CLC

CLC (no operands)
Clear carry flag

Flags	O	D	I	T	S	Z	A	P	C
									0

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	CLC

CLD

CLD (no operands)
Clear direction flag

Flags	O	D	I	T	S	Z	A	P	C
		0							

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	CLD

CLI

CLI (no operands)
Clear interrupt flag

Flags	O	D	I	T	S	Z	A	P	C
			0						

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	CLI

CMC

CMC (no operands)
Complement carry flag

Flags	O	D	I	T	S	Z	A	P	C
									X

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	CMC

CMP

CMP destination,source
Compare destination to source

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
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

CMPS dest-string,source-string
Compare string

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
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

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

CWD	**CWD** (no operands) Convert word to doubleword			**Flags** O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	5	—	1	CWD

DAA	**DAA** (no operands) Decimal adjust for addition			**Flags** O D I T S Z A P C X X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4	—	1	DAA

DAS	**DAS** (no operands) Decimal adjust for subtraction			**Flags** O D I T S Z A P C U X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4	—	1	DAS

DEC	**DEC** destination Decrement by 1			**Flags** O D I T S Z A P C X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg16	2	—	1	DEC AX
reg8	3	—	2	DEC AL
memory	15+EA	2	2-4	DEC ARRAY [SI]

DIV	**DIV** source Division, unsigned			**Flags** O D I T S Z A P C U U U U U U
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg8	80-90	—	2	DIV CL
reg16	144-162	—	2	DIV BX
mem8	(86-96) +EA	1	2-4	DIV ALPHA
mem16	(150-168) +EA	1	2-4	DIV TABLE [SI]

ESC	**ESC** external-opcode,source Escape			**Flags** O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
immediate, memory	8+EA	1	2-4	ESC 6,ARRAY [SI]
immediate, register	2	—	2	ESC 20,AL

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

HLT

HLT (no operands)
Halt

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	HLT

IDIV

IDIV source
Integer division

Flags	O	D	I	T	S	Z	A	P	C
	U				U	U	U	U	U

Operands	Clocks	Transfers*	Bytes	Coding Example
reg8	101-112	—	2	IDIV BL
reg16	165-184	—	2	IDIV CX
mem8	(107-118) +EA	1	2-4	IDIV DIVISOR_BYTE [SI]
mem16	(171-190) +EA	1	2-4	IDIV [BX].DIVISOR_WORD

IMUL

IMUL source
Integer multiplication

Flags	O	D	I	T	S	Z	A	P	C
	X				U	U	U	U	X

Operands	Clocks	Transfers*	Bytes	Coding Example
reg8	80-98	—	2	IMUL CL
reg16	128-154	—	2	IMUL BX
mem8	(86-104) +EA	1	2-4	IMUL RATE_BYTE
mem16	(134-160) +EA	1	2-4	IMUL RATE_WORD [BP] [DI]

IN

IN accumulator,port
Input byte or word

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
accumulator, immed8	10	1	2	IN AL, 0FFEAH
accumulator, DX	8	1	1	IN AX, DX

INC

INC destination
Increment by 1

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	

Operands	Clocks	Transfers*	Bytes	Coding Example
reg16	2	—	1	INC CX
reg8	3	—	2	INC BL
memory	15+EA	2	2-4	INC ALPHA [DI] [BX]

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

INT	**INT** interrupt-type Interrupt			
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
immed8 (type = 3) immed8 (type ≠ 3)	52 51	5 5	1 2	INT 3 INT 67

Flags	O	D	I	T	S	Z	A	P	C
			0	0					

INTR†	**INTR** (external maskable interrupt) Interrupt if INTR and IF=1			
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	61	7	N/A	N/A

Flags	O	D	I	T	S	Z	A	P	C
			0	0					

INTO	**INTO** (no operands) Interrupt if overflow			
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	53 or 4	5	1	INTO

Flags	O	D	I	T	S	Z	A	P	C
			0	0					

IRET	**IRET** (no operands) Interrupt Return			
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	24	3	1	IRET

Flags	O	D	I	T	S	Z	A	P	C
	R	R	R	R	R	R	R	R	R

JA/JNBE	**JA/JNBE** short-label Jump if above/Jump if not below nor equal			
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JA ABOVE

Flags	O	D	I	T	S	Z	A	P	C

JAE/JNB	**JAE/JNB** short-label Jump if above or equal/Jump if not below			
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JAE ABOVE_EQUAL

Flags	O	D	I	T	S	Z	A	P	C

JB/JNAE	**JB/JNAE** short-label Jump if below/Jump if not above nor equal			
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JB BELOW

Flags	O	D	I	T	S	Z	A	P	C

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

†INTR is not an instruction; it is included in table 2-21 only for timing information.

JBE/JNA

JBE/JNA short-label
Jump if below or equal/Jump if not above

Flags O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JNA NOT_ABOVE

JC

JC short-label
Jump if carry

Flags O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JC CARRY_SET

JCXZ

JCXZ short-label
Jump if CX is zero

Flags O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	18 or 6	—	2	JCXZ COUNT_DONE

JE/JZ

JE/JZ short-label
Jump if equal/Jump if zero

Flags O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JZ ZERO

JG/JNLE

JG/JNLE short-label
Jump if greater/Jump if not less nor equal

Flags O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JG GREATER

JGE/JNL

JGE/JNL short-label
Jump if greater or equal/Jump if not less

Flags O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JGE GREATER_EQUAL

JL/JNGE

JL/JNGE short-label
Jump if less/Jump if not greater nor equal

Flags O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
short-label	16 or 4	—	2	JL LESS

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

JLE/JNG	**JLE/JNG** short-label Jump if less or equal/Jump if not greater			Flags ODITSZAPC
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JNG NOT_GREATER

JMP	**JMP** target Jump			Flags ODITSZAPC
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	15	—	2	JMP SHORT
near-label	15	—	3	JMP WITHIN_SEGMENT
far-label	15	—	5	JMP FAR_LABEL
memptr16	18+EA	1	2-4	JMP [BX].TARGET
regptr16	11	—	2	JMP CX
memptr32	24+EA	2	2-4	JMP OTHER.SEG [SI]

JNC	**JNC** short-label Jump if not carry			Flags ODITSZAPC
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JNC NOT_CARRY

JNE/JNZ	**JNE/JNZ** short-label Jump if not equal/Jump if not zero			Flags ODITSZAPC
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JNE NOT_EQUAL

JNO	**JNO** short-label Jump if not overflow			Flags ODITSZAPC
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JNO NO_OVERFLOW

JNP/JPO	**JNP/JPO** short-label Jump if not parity/Jump if parity odd			Flags ODITSZAPC
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JPO ODD_PARITY

JNS	**JNS** short-label Jump if not sign			Flags ODITSZAPC
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JNS POSITIVE

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

JO	JO short-label Jump if overflow			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JO SIGNED_OVRFLW

JP/JPE	JP/JPE short-label Jump if parity/Jump if parity even			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JPE EVEN_PARITY

JS	JS short-label Jump if sign			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	16 or 4	—	2	JS NEGATIVE

LAHF	LAHF (no operands) Load AH from flags			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	4	—	1	LAHF

LDS	LDS destination,source Load pointer using DS			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers**	**Bytes**	**Coding Example**
reg16, mem32	16+EA	2	2-4	LDS SI,DATA.SEG [DI]

LEA	LEA destination,source Load effective address			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg16, mem16	2+EA	—	2-4	LEA BX, [BP] [DI]

LES	LES destination,source Load pointer using ES			Flags O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg16, mem32	16+EA	2	2-4	LES DI, [BX].TEXT_BUFF

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

LOCK	LOCK (no operands) Lock bus			**Flags** O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	2	—	1	LOCK XCHG FLAG,AL

LODS	LODS source-string Load string			**Flags** O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
source-string (repeat) source-string	12 9+13/rep	1 1/rep	1 1	LODS CUSTOMER_NAME REP LODS NAME

LOOP	LOOP short-label Loop			**Flags** O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	17/5	—	2	LOOP AGAIN

LOOPE/LOOPZ	LOOPE/LOOPZ short-label Loop if equal/Loop if zero			**Flags** O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	18 or 6	—	2	LOOPE AGAIN

LOOPNE/LOOPNZ	LOOPNE/LOOPNZ short-label Loop if not equal/Loop if not zero			**Flags** O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
short-label	19 or 5	—	2	LOOPNE AGAIN

NMI†	NMI (external nonmaskable interrupt) Interrupt if NMI = 1			**Flags** O S I T S Z A P C 0 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	50	5	N/A	N/A

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

†NMI is not an instruction; it is included in table 2-21 only for timing information.

MOV	**MOV** destination,source Move			**Flags** O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
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, reg16	2	—	2	MOV ES, CX
seg-reg, mem16	8+EA	1	2-4	MOV DS, SEGMENT_BASE
reg16, seg-reg	2	—	2	MOV BP, SS
memory, seg-reg	9+EA	1	2-4	MOV [BX].SEG_SAVE, CS

MOVS	**MOVS** dest-string,source-string Move string			**Flags** O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
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/MOVSW	**MOVSB/MOVSW** (no operands) Move string (byte/word)			**Flags** O D I T S Z A P C
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
(no operands)	18	2	1	MOVSB
(repeat) (no operands)	9+17/rep	2/rep	1	REP MOVSW

MUL	**MUL** source Multiplication, unsigned			**Flags** O D I T S Z A P C X U U U U X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
reg8	70-77	—	2	MUL BL
reg16	118-133	—	2	MUL CX
mem8	(76-83) +EA	1	2-4	MUL MONTH [SI]
mem16	(124-139) +EA	1	2-4	MUL BAUD_RATE

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

NEG

NEG destination
Negate

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	1*

Operands	Clocks	Transfers*	Bytes	Coding Example
register	3	—	2	NEG AL
memory	16 + EA	2	2-4	NEG MULTIPLIER

*0 if destination = 0

NOP

NOP (no operands)
No Operation

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	3	—	1	NOP

NOT

NOT destination
Logical not

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
register	3	—	2	NOT AX
memory	16 + EA	2	2-4	NOT CHARACTER

OR

OR destination,source
Logical inclusive or

Flags	O	D	I	T	S	Z	A	P	C
	0				X	X	U	X	0

Operands	Clocks	Transfers*	Bytes	Coding Example
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, 01101100B
register, immediate	4	—	3-4	OR CX,01H
memory, immediate	17 + EA	2	3-6	OR [BX].CMD_WORD,0CFH

OUT

OUT port,accumulator
Output byte or word

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
immed8, accumulator	10	1	2	OUT 44, AX
DX, accumulator	8	1	1	OUT DX, AL

POP

POP destination
Pop word off stack

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
register	8	1	1	POP DX
seg-reg (CS illegal)	8	1	1	POP DS
memory	17 + EA	2	2-4	POP PARAMETER

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

POPF

POPF (no operands)
Pop flags off stack

Flags	O	D	I	T	S	Z	A	P	C
	R	R	R	R	R	R	R	R	R

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	8	1	1	POPF

PUSH

PUSH source
Push word onto stack

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
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

PUSHF (no operands)
Push flags onto stack

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	10	1	1	PUSHF

RCL

RCL destination,count
Rotate left through carry

Flags	O	D	I	T	S	Z	A	P	C
	X								X

Operands	Clocks	Transfers*	Bytes	Coding Example
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

RCR designation,count
Rotate right through carry

Flags	O	D	I	T	S	Z	A	P	C
	X								X

Operands	Clocks	Transfers*	Bytes	Coding Example
register, 1	2	—	2	RCR BX, 1
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

REP (no operands)
Repeat string operation

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	REP MOVS DEST, SRCE

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

REPE/REPZ

REPE/REPZ (no operands)
Repeat string operation while equal/while zero

Flags O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	REPE CMPS DATA, KEY

REPNE/REPNZ

REPNE/REPNZ (no operands)
Repeat string operation while not equal/not zero

Flags O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	REPNE SCAS INPUT__LINE

RET

RET optional-pop-value
Return from procedure

Flags O D I T S Z A P C

Operands	Clocks	Transfers*	Bytes	Coding Example
(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

ROL destination,count
Rotate left

Flags	O	D	I	T	S	Z	A	P	C
	X								X

Operands	Clocks	Transfers	Bytes	Coding Examples
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

ROR destination,count
Rotate right

Flags	O	D	I	T	S	Z	A	P	C
	X								X

Operand	Clocks	Transfers*	Bytes	Coding Example
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

SAHF (no operands)
Store AH into flags

Flags	O	D	I	T	S	Z	A	P	C
					R	R	R	R	R

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	4	—	1	SAHF

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

SAL/SHL

SAL/SHL destination,count
Shift arithmetic left/Shift logical left

Flags	O	D	I	T	S	Z	A	P	C
	X								X

Operands	Clocks	Transfers*	Bytes	Coding Examples
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

SAR destination,source
Shift arithmetic right

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	U	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
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
memory, CL	20+EA+ 4/bit	2	2-4	SAR N_BLOCKS, CL

SBB

SBB destination,source
Subtract with borrow

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
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

SCAS dest-string
Scan string

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
dest-string	15	1	1	SCAS INPUT_LINE
(repeat) dest-string	9+15/rep	1/rep	1	REPNE SCAS BUFFER

SEGMENT†

SEGMENT override prefix
Override to specified segment

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	MOV SS:PARAMETER, AX

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

†ASM-86 incorporates the segment override prefix into the operand specification and not as a separate instruction. SEGMENT is included in table 2-21 only for timing information.

SHR

SHR destination,count
Shift logical right

Flags	O	D	I	T	S	Z	A	P	C
	X								X

Operands	Clocks	Transfers*	Bytes	Coding Example
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

SINGLE STEP†

SINGLE STEP (Trap flag interrupt)
Interrupt if TF = 1

Flags	O	D	I	T	S	Z	A	P	C
			0	0					

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	50	5	N/A	N/A

STC

STC (no operands)
Set carry flag

Flags	O	D	I	T	S	Z	A	P	C
									1

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	STC

STD

STD (no operands)
Set direction flag

Flags	O	D	I	T	S	Z	A	P	C
		1							

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	STD

STI

STI (no operands)
Set interrupt enable flag

Flags	O	D	I	T	S	Z	A	P	C
			1						

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	2	—	1	STI

STOS

STOS dest-string
Store byte or word string

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
dest-string	11	1	1	STOS PRINT_LINE
(repeat) dest-string	9+10/rep	1/rep	1	REP STOS DISPLAY

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

†SINGLE STEP is not an instruction; it is included in table 2-21 only for timing information.

SUB

SUB destination,source
Subtraction

Flags	O	D	I	T	S	Z	A	P	C
	X				X	X	X	X	X

Operands	Clocks	Transfers*	Bytes	Coding Example
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

TEST destination,source
Test or non-destructive logical and

Flags	O	D	I	T	S	Z	A	P	C
	0				X	X	U	X	0

Operands	Clocks	Transfers*	Bytes	Coding Example
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
register, immediate	5	—	3-4	TEST BX, 0CC4H
memory, immediate	11+EA	—	3-6	TEST RETURN_CODE, 01H

WAIT

WAIT (no operands)
Wait while $\overline{\text{TEST}}$ pin not asserted

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
(no operands)	3 + 5n	—	1	WAIT

XCHG

XCHG destination,source
Exchange

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
accumulator, reg16	3	—	1	XCHG AX, BX
memory, register	17+EA	2	2-4	XCHG SEMAPHORE, AX
register, register	4	—	2	XCHG AL, BL

XLAT

XLAT source-table
Translate

Flags	O	D	I	T	S	Z	A	P	C

Operands	Clocks	Transfers*	Bytes	Coding Example
source-table	11	1	1	XLAT ASCII_TAB

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

<table>
<tr><td><b>XOR</b></td><td colspan="3">XOR destination,source
Logical exclusive or</td><td>Flags O D I T S Z A P C
0 X X U X 0</td></tr>
<tr><th>Operands</th><th>Clocks</th><th>Transfers*</th><th>Bytes</th><th>Coding Example</th></tr>
<tr><td>register, register</td><td>3</td><td>—</td><td>2</td><td>XOR CX, BX</td></tr>
<tr><td>register, memory</td><td>9+EA</td><td>1</td><td>2-4</td><td>XOR CL, MASK_BYTE</td></tr>
<tr><td>memory, register</td><td>16+EA</td><td>2</td><td>2-4</td><td>XOR ALPHA [SI], DX</td></tr>
<tr><td>accumulator, immediate</td><td>4</td><td>—</td><td>2-3</td><td>XOR AL, 01000010B</td></tr>
<tr><td>register, immediate</td><td>4</td><td>—</td><td>3-4</td><td>XOR SI, 00C2H</td></tr>
<tr><td>memory, immediate</td><td>17+EA</td><td>2</td><td>3-6</td><td>XOR RETURN_CODE, 0D2H</td></tr>
</table>

*For the 8086, add four clocks for each 16-bit word transfer with an odd address. For the 8088, add four clocks for each 16-bit word transfer.

APPENDIX F

GLOSSARY/ASCII CODES

Absolute Addressing: The specific identification number (address) permanently assigned to a storage location, device, or register by the machine designer. Used to locate information and assist in circuit fault diagnosis.

Accumulator: Used for storing the result after most ALU operations; 8 bits long for an 8-bit microprocessor.

Address: A unique identification number (or locator) of some source or destination of data. That part of an instruction which specifies the register or memory location of an operand involved in the instruction.

Addressing Mode: The manner in which a microprocessor determines the operand and destination addresses in an instruction cycle.

Address Register: A register used to store the address (label for a memory location) of data being fetched or stored, a sequence of instructions to be executed, or the location to which control will be transferred.

Address Space: The number of storage locations in a microcomputer's memory that can be directly addressed by the microprocessor. The addressing range is determined by the number of address pins provided with the microprocessor chip.

American Standard Code for Information Interchange (ASCII): An 8-bit code commonly used with microprocessors for representing alphanumeric codes.

Analog-to-Digital (A/D) Converter: Transforms an analog voltage into its digital equivalent.

Architecture: The organizational structure or hardware configuration of a computer system.

Arithmetic and Logic Unit (ALU): A digital circuit which performs arithmetic and logic operations on two n-bit numbers.

Assembler: A program that translates an assembly language program into a machine language program.

Assembly Language: A type of microprocessor programming language that uses a semi-English-language statement.

Asynchronous Operation: The execution of a sequence of steps such that each step is initiated upon completion of the previous step. For bus structures, this implies a timing protocol that uses no clock and has no period; hence system operation proceeds at a rate governed by the time-constants of the enabled circuitry.

Asynchronous Serial Data Transmission: The transmitting device does not need to be synchronized with the receiving device.

Autodecrement Addressing Mode: The contents of the specified microprocessor register are first decremented by K (1 for byte, 2 for 16-bit, and 4 for 32-bit) and then the resulting value is used as the address of the operand.

Autoincrement Addressing Mode: The contents of a specified microprocessor register are used as the address of the operand first and then the register contents are automatically incremented by K (1 for byte, 2 for 16-bit, and 4 for 32-bit).

Bandwidth: Bandwidth of a bus or memory is a measure of communications throughput and can be represented as the product of the maximum number of transactions per second and number of data bits per transaction.

Barrel Shifter: A specially configured shift register that is normally included in 32-bit microprocessors for fast shift operations.

Base Address: An address that is used to convert all relative addresses in a program to absolute (machine) addresses.

Base Page Addressing: This instruction typically uses two bytes: the first byte is the op code, and the second byte is the low-order address byte. The high-order address byte is assumed to be the base-page number.

Baud Rate: Rate of data transmission in bits per second.

Binary-Coded Decimal (BCD): The representation of 10 decimal digits, 0 through 9, by their corresponding 4-bit binary numbers.

Bit: An abbreviation for a binary digit. A unit of information equal to one binary decision or one of two possible states (one or zero, on or off, true or false) and represents the smallest piece of information in a binary notation system.

Bit-Slice Microprocessor: Divides the elements of a central processing unit (ALU, registers, and control unit) among several ICs. The registers and ALU are usually contained in a single chip. These microprocessors can be cascaded to produce microprocessors of variable word lengths such as 8, 12, 16, 32. The control unit of a bit-slice microprocessor is typically microprogrammed.

Block Transfer DMA: A peripheral device requests the DMA transfer via the DMA request line, which is connected directly or through a DMA controller chip to the microprocessor. The DMA controller chip completes the DMA transfer and transfers the control of the bus to the microprocessor.

Branch: The branch instruction allows the computer to skip or jump out of program sequence to a designated instruction either unconditionally or conditionally (based on conditions such as carry or sign).

Breakpoint: Allows the user to execute the section of a program until one of the breakpoint conditions is met. It is then halted. The designer may then single step or examine memory and registers. Typically breakpoint conditions are program counter address or data references. Breakpoints are used in debugging assembly language programs.

Buffer: A temporary memory storage device designed to compensate for the different data rates between a transmitting device and a receiving device (for example, between a CPU and a peripheral). Current amplifiers are also sometimes referred to as buffers.

Bus: A collection of parallel unbroken electrical signal lines that interconnect or link computer modules. The typical microcomputer interface includes separate buses for address, data, control, and power functions.

Bus Arbitration: Bus operation protocols that guarantee conflict-free access to a bus. Arbitration is the process of selecting one respondent from a collection of several candidates that concurrently request service.

Bus Cycle: The period of time in which a microprocessor carries out all the necessary bus communications to implement a standard operation.

Byte: An 8-bit word.

Cache Memory: An ultra-high speed, directly accessible, relatively small semiconductor memory block used to store data/instructions that the microcomputer may need in the immediate future. Increases system bandwidth by reducing the number of external memory fetches required by the processor. Typical 32-bit microprocessors are normally provided with on-chip cache memory.

Cathode Ray Tube (CRT): Evacuated glass tube with a fluorescent coating on the inner side of the screen.

Central Processing Unit (CPU): The portion of a computer containing the ALU, register section, and control unit.

Clock: Timing signals providing synchronization among the various components in a microcomputer system.

Code: A system of symbols or sets of rules for the representation of data in a digital computer. Some examples include binary, BCD, and ASCII.

Compiler: A software program which translates the source code written in a high-level programming language into machine language that is understandable to the processor.

Complementary Metal Oxide Semiconductor (CMOS): Provides low power density and high noise immunity.

Concurrency: The occurrence of one or more operations at a time (see Parallel Operation).

Conditional Branching: Conditional branch instructions are used to change the order of execution of a program based on the conditions set by the status flags.

Condition Code Register: Contains information such as carry, sign, zero, and overflow based on ALU operations.

Control Store: Used to contain microcode (usually in ROM) in order to provide for microprogrammed "firmware" control functions. An integral part of a microprogrammed system controller.

Control Unit: Part of the microprocessor; its purpose is to read and decode instructions from the memory.

Controller/Sequencer: The hardware circuits which provide signals to carry out selection and retrieval of instructions from storage in sequence, interpret them, and initiate the required operation. The system functions may be implemented by hardware control, firmware control, or software control.

Coprocessor: A companion microprocessor that performs specific functions such as floating-point operations independently from the microprocessor to speed up overall operations.

CPU Space: Protected memory space addressable only by the microprocessor itself; it is used for a processor's internal functions or vectored exception processing.

CRT Controller: Provides all logic functions for interfacing the microprocessor to a CRT.

Cycle Stealing DMA: The DMA controller transfers a byte of data between the microcomputer's memory and a peripheral device such as the disk by stealing a clock cycle of the microprocessor.

Data: Basic elements of information represented in binary form (that is, digits consisting of bits) that can be processed or produced by a microcomputer. Data represents any group of operands made up of numbers, letters, or symbols denoting any condition, value, or state. Current typical microcomputer operand sizes include: a word, which typically contains 2 bytes or 16 bits; a long word, which contains 4 bytes or 32 bits; a quad word, which contains 8 bytes or 64 bits.

Data Counter (DC): Also known as Memory Address Register (MAR). Stores the address of data; typically, 16 bits long for 8-bit microprocessors.

Data Register: A register used to temporarily hold operational data being sent to and from a peripheral device.

Debugger: A program that executes and debugs the object program generated by the assembler or compiler. The debugger provides a single stepping, breakpoints, and program tracing.

Decoder: A device capable of generating 2n output lines based on n inputs.

Direct Memory Access (DMA): A type of input/output technique in which data can be transferred between the microcomputer memory and external devices without the microprocessor's involvement.

Directly Addressable Memory: The memory address space in which the microprocessor can directly execute programs. The maximum directly addressable memory is determined by the number of the microprocessor's address pins.

Dynamic RAM: Stores data in capacitors and, therefore, must be refreshed; uses refresh circuitry.

EAROM (Electrically Alterable Read-Only Memory): Can be programmed without removing the memory from its sockets. This memory is also called read-mostly memory since it has much slower write times than read times.

Editor: A program that produces an error-free source program, written in assembly or high-level languages.

Effective Address: The final address used to carry out an instruction. Determined by the addressing mode.

Emulator: A hardware device that allows a microcomputer system to emulate (that is, mimic the procedures or protocols) another microcomputer system.

Encode: To apply the rules governing a specific code. For example, the selection of which hardware devices to enable during an operation can occur automatically by encoding individual device identifications into the instructions themselves. Hence, to encode is to convert data from its natural form into a machine-readable code usable to the computer.

EPROM (Erasable Programmable Read-Only Memory): Can be programmed and erased using ultraviolet light. The chip must be removed from the microcomputer system for programming.

Exception Processing: The CPU processing state associated with interrupts, trap instructions, tracing, and other exceptional conditions, whether they are initiated internally or externally.

Extended Binary-Coded Decimal Interchange Code (EBCDIC): An 8-bit code commonly used with microprocessors for representing character codes.

Firmware: Permanently stored, unalterable program instructions contained in the ROM section of a computer's memory (see Control Store).

Flag(s): An indicator, often a single bit, to indicate some conditions such as trace, carry, zero, and overflow.

Flash Memory: Nonvolatile and reprogrammable memory. Fabricated by using ETOXII (EPROM tunnel oxide) technology which is a combination of EPROM and EEPROM technologies. Can be reprogrammed while embedded in the board. However, one can only change a sector or block (consisting of multiple bytes) at a time.

Flowchart: Representation of a program in a schematic form. It is convenient to flowchart a problem before writing the actual programs.

Global Bus: A computer bus system that is available to and shared by a number of processors connected together in a multiprocessor system environment.

Handshaking: Data transfer via exchange of control signals between the microprocessor and an external device.

Hardware: The physical electronic circuits (chips) that make up the microcomputer system.

HCMOS: Low-power HMOS.

Hexadecimal Number System: Base-16 number system.

Hierarchical Memory: A memory organization or informational structure in which functional relationships are associated with different levels.

High-Level Language: A type of programming language that uses a more understandable human-oriented language such as Pascal.

HMOS: High-performance MOS reduces the channel length of the NMOS transistor and provides increased density and speed in LSI and VLSI circuits.

Immediate Address: An address that is used as an operand by the instruction itself.

Implied Address: An address not specified, but contained implicitly in the instruction.

In-Circuit Emulation: The most powerful hardware debugging technique; especially valuable when hardware and software are being debugged simultaneously.

Index: A symbol used to identify or place a particular quantity in an array (list) of similar quantities. Also, an ordered list of references to the contents of a larger body of data such as a file or record.

Indexed Addressing: Typically uses 3 bytes: the first byte for the op code and the next 2 bytes for the 16-bit address. The effective address of the instruction is determined by the sum of the 16-bit address and the contents of the index register.

Index Register: A register used to hold a value used in indexing data, such as when a value is used in indexed addressing to increment a base address contained within an instruction.

Indirect Address: A register holding a memory address to be accessed.

Instruction: A program statement (step) that causes the microcomputer to carry out an operation, and specifies the values or locations of all operands.

Instruction Cycle: The sequence of operations that a microprocessor has to carry out while executing an instruction.

Instruction Register (IR): A register storing instructions; typically 8 bits long for an 8-bit microprocessor.

Instruction Set: Lists all the instructions (available in machine code) that the microcomputer can execute.

Interleaved DMA: Using this technique, the DMA controller takes over the system bus when the microprocessor is not using it.

Internal Interrupt: Activated internally by exceptionally conditions such as overflow and division by zero.

Interpreter: A program that executes a set of machine language instructions in response to each high-level statement in order to carry out the function.

Interrupt I/O: An external device can force the microcomputer system to stop executing the current program temporarily so that it can execute another program known as the interrupt service routine.

Interrupts: A temporary break in a sequence of a program, initiated externally, causing control to pass to a routine, which performs some action while the program is stopped.

I/O (Input/Output): Describes that portion of a microcomputer system that exchanges data between the microcomputer system and the external world, or the data itself.

I/O Port: A module that contains control logic and data storage used to connect a microcomputer to external peripherals.

Keyboard: Has a number of pushbutton-type switches configured in a matrix form (rows × columns).

Keybounce: When a mechanical switch opens or closes, it bounces (vibrates) for a small period of time (about 10–20 ms) before settling down.

Large-Scale Integration (LSI): An LSI chip contains more than 100 gates.

Linkage Editors: Connect the individual programs together which are assembled or compiled independently.

Linked Programming: The process of joining a subprogram with a main program or joining two separate programs together to form a single program.

Local Area Network: A collection of devices and communication channels that connect a group of computers and peripherals devices together so that they can communicate with each other.

Logic Analyzer: A hardware development aid for microprocessor-based design; gathers data on the fly and displays it.

Logical Address Space: All storage locations with a programmer's addressing range.

Loops: A programming control structure where a sequence of microcomputer instructions are executed repeatedly (looped) until a terminating condition (result) is satisfied.

Machine Code: A binary code (composed of bit patterns) that a microcomputer can sense, read, interpret, recognize, and manipulate.

Machine Language: A type of microprocessor programming language that uses binary or hexadecimal numbers.

Macroinstruction: Commonly known as an instruction; initiates execution of a complete microprogram.

Macroprogram: The assembly language program.

Mask: A pattern of bits used to specify (or mask) which bit parts of another bit pattern are to be operated on and which bits are to be ignored or "masked" out.

Mask ROM: Programmed by a masking operation performed on the chip during the manufacturing process; its contents cannot be changed by the user.

Maskable Interrupt: Can be enabled or disabled by executing typically the instructions such as EI and DI, respectively. If the microprocessor's interrupt is disabled, the microprocessor ignores the interrupt.

Memory: Any storage device which can accept, retain, and read back data. Usually refers to a computer subsystem of internal RAM- or ROM-based storage devices.

Memory Access Time: Average time taken to read a unit of information from the memory.

Memory Address Register (MAR): Also known as the Data Counter (DC). Stores the address of the data; typically 16 bits long for 8-bit microprocessors.

Memory Cycle Time: Average time lapse between two successive read operations.

Memory Management Unit (MMU): Performs address translation and protection functions.

Memory Map: A representation of the physical location of software within a microcomputer's addressable main storage.

Memory-Mapped I/O: A microprocessor communications methodology (addressing scheme) where the data, address, and control buses extend throughout the system, with every connected device treated as if it were a memory location with a specific address. Manipulation of I/O data occurs in "interface registers" (as opposed to memory locations); hence there are no input (read) or output (write) instructions used in memory-mapped I/O.

Microcode: A set of "subcommands" or "pseudocommands" built into the hardware (usually stored in ROM) of a microcomputer (that is, firmware) to handle the decoding the execution of higher-level instructions such as arithmetic operation.

Microcomputer: Consists of a microprocessor, a memory unit, and an input/output unit.

Microcontroller: Typically includes a microcomputer, timer, A/D (Analog to Digital) and D/A (Digital to Analog) converters in the same chip.

Microinstruction: Most microprocessors have an internal memory called control memory. This memory is used to store a number of codes called microinstructions. These microinstructions are combined to design the instruction set of the microprocessor.

Microprocessor: The Central Processing Unit (CPU) of a microcomputer.

Microprocessor Development System: A tool for designing and debugging both hardware and software for microcomputer-based systems.

Microprocessor-Halt DMA: Data transfer is performed between the microcomputer's memory and a peripheral device either by completely stopping the microprocessor or by a technique called cycle stealing.

Microprogramming: The microprocessor can use microprogramming to design the instruction set. Each instruction in the instruction register initiates execution of a microprogram in the control unit to perform the operation required by the instruction.

Module: (1) Any single hardware arrangement (device or component) within a microcomputer system. (2) Any software, routine, or subroutine.

Monitor: Consists of a number of subroutines grouped together to provide "intelligence" to a microcomputer system. This intelligence gives the microcomputer system the capabilities for debugging a user program, system design, and displays.

Multiplexer: A hardware device which allows a microprocessor to be physically connected to a number of communication channels to receive or transmit data.

Multiprocessing: The process of executing two or more programs in parallel, handled by multiple processors all under common control. Typically each processor will be assigned specific processing tasks.

Multitasking: Operating system software that permits more than one program to run on a single microprocessor. Even though each program is given a small time slice in which to execute, the user has the impression that all tasks (different programs) are executing at the same time.

Multiuser: Describes a computer operating system that permits a number of users to access the system on a time-sharing basis.

Nested Subroutine: A commonly used programming technique that includes one subroutine entirely embedded within the "scope" of another subroutine.

Nibble: A 4-bit word.

NMOS: Denser and faster in comparison to PMOS. Most 8-bit microprocessors and some 16-bit microprocessors are fabricated using this technology.

Noncontiguous: Noncontiguous in nature. Refers to breaks in the linear sequential flow of any information structure.

Nonmaskable Interrupt: Occurrence of this type of interrupt cannot be ignored by the microprocessor, even though the interrupt capability of the microprocessor is disabled. Its effect cannot be disabled by instruction.

Non-Multiplexed: A non-multiplexed system indicates a direct single communication channel (that is, electrical wires) connection to the microprocessor.

Object Code: The binary (machine) code into which a source program is translated by a compiler, assembler, or interpreter.

Octal Number System: Base-8 number system.

One-Pass Assembler: This assembler goes through the assembly language program once and translates the assembly language program into a machine language program. This assembler has the problem of defining forward references. See Two-Pass Assembler.

Op Code (Operation Code): The instruction represented in binary form.

Operand: A datum or information item involved in an operation from which the result is obtained as a consequence of defined actions (that is, data which is operated on by an instruction). Various operand types contain information, such as source address, destination address, or immediate data.

Operating System: Consists of a number of program modules to provide resource management. Typical resources include microprocessors, disks, and printers.

Operation: (1) Means by which a result is obtained from an operand(s). (2) An action defined by a single instruction or single logical element.

Page: Some microprocessors, such as the Motorola 6800 and the MOS 6502, divide the 65,536 memory locations into 256 blocks. Each of these blocks is called a page and contains 256 addresses.

Parallel Operation: Any operation carried out simultaneously with a related operation.

Parallel Transmission: Each bit of binary data is transmitted over a separate wire.

Parity: The number of 1's in a word is odd for odd parity and even for even parity.

Peripherals: An I/O device capable of being operated under the control of a CPU through communication channels. Examples include disk drives, keyboards, CRTs, printers, modems, etc.

Personal Computer: Low-cost, affordable microcomputer used by an individual or a small group for video games, daily schedules, and industrial applications.

Physical Address Space: Address space is defined by the address pins of the microprocessor.

Pipeline: A technique that allows a microcomputer processing operation to be broken down into several steps (dictated by the number of pipeline levels or stages) so that the individual step outputs can be handled by the computer in parallel. Often used to fetch the processor's next instruction while executing the current instruction, which considerably speeds up the overall operation of the microcomputer.

Pointer: A storage location (usually a register within a microprocessor) that contains the address of (or points to) a required item of data or subroutine.

Polled Interrupt: A software approach for determining the source of interrupt in a multiple interrupt system.

POP Operation: Reading from the top or bottom of the stack.

Port: An access point (a register) for a microcomputer through which communication data may be passed to peripheral devices.

Primary Memory Store: That memory storage which is considered main, integral, or internal to the computing system. It is that storage which is physically most closely associated with the microprocessor and is directly controlled by it.

Primitives: A basic or fundamental unit; often refers to the lowest level of machine instruction or the lowest unit of programming language instruction.

Privileged Instructions: An instruction which is reserved for use by a computer's operating system, which will determine the range of system resources that the user is allowed to exploit.

Processor Memory: A set of microprocessor registers for holding temporary results when a computation is in progress.

Program: A self-contained sequence of computer software instructions (source code) that, when converted into machine code, directs the computer to perform specific operations for the purpose of accomplishing some processing task.

Program Array Logic (PAL): Similar to a ROM in concept except that it does not provide full decoding of the input lines. PAL's are used with 32-bit microprocessors for performing the memory decode function.

Program Counter (PC): A register that normally contains the address of the next instruction in the sequence of operations.

Programmed I/O: The microprocessor executes a program to perform all data transfers between the microcomputer system and external devices.

PROM (Programmable Read-Only Memory): Can be programmed by the user by using proper equipment. Once programmed, its contents cannot be altered.

Protocol: A list of data transmission conventions or procedures that encompass the timing, control, formatting, and data representations by which two devices are to communicate. Also known as hardware "handshaking", which is used to permit asynchronous communication.

PUSH Operation: Writing to the top or bottom of the stack.

Random Access Memory (RAM): A read/write memory. RAMs (static or dynamic) are volatile in nature (in other words, information is lost when power is removed).

Read-Only-Memory (ROM): A memory in which any addressable operand can be read from, but not written to, after initial programming. It is an asynchronous device whose access time is dictated by its internal circuit time delays. ROM storage is non-volatile (information is not lost when power is removed).

Real-Time Software: Computer code that allows processes to be performed during the actual time that a related physical I/O action takes place.

Reduced Instruction Set Computer (RISC): A necessary and sufficient instruction set is included. The RISC architecture maximizes speed by reducing clock cycles per instruction. Performs infrequent operations in software and frequent functions in hardware.

Register: A one-word, high-speed memory device usually constructed from flip-flops (electronic switches) that are directly accessible to the processor. It can also refer to a specific location in memory that contains word(s) used during arithmetic, logic, and transfer operations.

Register Indirect: Uses a register pair which contains the address of data.

Relative Address: An address used to designate the position of a memory location in a routine or program.

Rollover: Occurs when more than one key is pushed simultaneously.

Routine: A group of instructions for carrying out a specific processing operation. Usually refers to part of a larger program. A routine and subroutine have essentially the same meaning, but a subroutine could be interpreted as a self-contained routine nested within a routine or program.

Sample and Hold Circuit: When connected to the input of an A/D converter, it keeps a rapidly varying analog signal fixed during the A/D conversion process by storing it in a capacitor.

Scalar Microprocessor: Provided with one pipeline. Can execute one instruction per clock cycle. The 80486 is a scalar microprocessor.

Scaling: To adjust values or bring them into a range that is acceptable to a microcomputer.

Secondary Memory Storage: An auxiliary data storing device that supplements the main (primary) internal memory of a microcomputer. It is used to hold programs and data that would otherwise exceed the capacity of the main memory. Although it has a much slower access time, secondary storage is less expensive, Common devices include magnetic disk (floppy and hard), cassette tape, and videodisk.

Serial Transmission: Only one line is used to transmit the complete binary data bit by bit.

Single-Chip Microcomputer: Microcomputer (CPU, memory, and input/output) on a chip.

Single-Chip Microprocessor: Microcomputer CPU (microprocessor) on a chip.

Single Step: Allows the user to execute a program one instruction at a time and examine memory and registers.

Software: Programs in a microcomputer.

Source Code: The high-level language code used by a programmer to write computer instructions. This code must be translated to the object (machine) code to be usable to the microcomputer.

Stack: An area of read/write memory reserved to hold information about the status of a microcomputer the instant an interrupt occurs so that the microcomputer can continue processing after the interrupt has been handled. Another common use is in handling the accessing sequence of "nested" subroutines. The stacks are the last in/first out (LIFO) devices that are manipulated by using PUSH or POP instructions.

Stack Pointer: An address or register used to keep track of the storage and retrieval of each byte or word of information in the system stack.

Standard I/O: Utilizes a control pin on the microprocessor chip called the IO/M pin, in order to distinguish between input/output and memory; typically, IN and OUT instructions are used for performing input/output operations.

Static RAM: Stores data in flip-flops; does not need to be refreshed. Information is lost upon power failure unless backed up by battery.

Status Register: A register which contains information concerning the activity within the microprocessor or about the condition of a functional unit or peripheral device.

Subroutine: A program carrying out a particular function and which can be called by another program known as the main program. A subroutine needs to be placed only once in memory and can be called by the main program as many times as the programmer wants.

Superscalar Microprocessor: Provided with dual pipelining and executes more than one instruction per clock cycle. The Pentium is a superscalar microprocessor.

Supervisor: Provides the procedures or instructions for coordinating the use of system resources and maintaining the flow of operations through a microprocessor to perform I/O operations.

Supervisor State: When internal microprocessor system processing operations are conducted at a higher privilege level, it is usually in the supervisor state. An operating system typically executes in the supervisor state to protect the integrity of "basic" system operations from user influences.

Synchronous Operation: Operations that occur at intervals directly related to a clock period. Also, a bus protocol in such data transactions is controlled by a master clock and is completed within a fixed clock period.

Synchronous Serial Data Transmission: Data is transmitted or received based on a clock signal.

Tracing: A dynamic diagnostic technique in which a record of internal counter events is made to permit analysis (debugging) of the program's execution.

Tristate Buffer: Has three output states: logic 0, 1, and a high-impedance state. It is typically enabled by a control signal to provide logic 0 or 1 outputs. This type of buffer can also be disabled by the control signal to place it in a high-impedance state.

2's Complement: The 2's complement of a binary number is obtained by replacing each 0 with a 1 and each 1 with a 0 and adding one to the resulting number.

Two-Pass Assembler: This assembler goes through the assembly language program twice. In the first pass, the assembler defines the labels with the addresses. In the second pass, the assembler translates the assembly language program to the machine language. See One-Pass Assembler.

UART (Universal Asynchronous Receiver Transmitter): A chip that provides all the interface functions when a microprocessor transmits-or receives data to or from a serial device. Converts serial data to parallel and vice versa.

User State: Typical microprocessor operations processing conducted at the user level. The user state is usually at lower privilege level than the supervisor state. This protects basic system operation resources (the operating system).

Vectored Interrupts: A device identification technique in which the highest priority device with a pending interrupt request forces program execution to branch to an interrupt routine to handle exception processing for the device.

Very Large Scale Integration (VLSI): A VLSI chip contains more than 1000 gates.

Virtual Machine: A microcomputer whose hardware and software architecture is specifically designed to support virtual storage techniques. The virtual machine concept is widely used within multiprogramming environments.

Virtual Memory: A memory management operating system technique that allows programs or data to exceed the physical size of the main, internal, directly accessed memory. Program or data segments/pages are swapped from external disk storage as needed. The swapping is invisible (transparent) to the programmer. Therefore the programmer need not be concerned with the actual physical size of internal memory while writing the code.

Word: The bit size of a microprocessor refers to the number of bits that can be processed simultaneously by the basic arithmetic circuits of the microprocessor. A number of bits taken as a group in this manner is called a word.

Table for American Standard Code for Information Interchange (ASCII), Standard No. X3.4—1968 of the American National Standards Institute.

		$b_6b_5b_4$ (column)							
	Row	000	001	010	011	100	101	110	111
$b_3b_2b_1b_0$	(hex)	0	1	2	3	4	5	6	7
0000	0	NUL	DLE	SP	0	@	P	`	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	'	7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SUB	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	K	[	k	{
1100	C	FF	FS	,	<	L	\	l	|
1101	D	CR	GS	-	=	M	]	m	}
1110	E	SO	RS	.	>	N	^	n	~
1111	F	SI	US	/	?	O	_	o	DEL

Control codes

NUL	Null	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of Text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronize
BEL	Bell	ETB	End transmitted block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	EXC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete or rubout

Bibliography

Allison, D. R., "A Design Philosophy for Microcomputer Architectures", *IEEE Trans. Computers.*

Artwick, B. A., *Microcomputer Interfacing,* Prentice-Hall, 1980.

Baer, J.-L., *Computer Systems Architecture,* Computer Science Press, 1980.

Boyce, J. C., *Microprocessor and Microcomputer Basics,* Prentice-Hall, 1979.

Breeding, K., *Microprocessor System Design Fundamentals,* Prentice-Hall, 1995.

Brey, B., *The Motorola Microprocessor Family: 68000, 68008, 68010, 68020, 68030, and 68040,* Saunders College Publishing, 1992.

Burns, J., "Within the 68020," *Electronics and Wireless Word,* pp 209–212, February 1985; pp 103–106, March 1985.

Chi, C. S., "Advances in Mass Storage Technology," *IEEE Computer,* Vol. 15, no. 5, pp 60–74, May 1982.

Chow, C. K., "On Optimization of Storage Hierarchies," *IBM Journal of Research and Development,* pp 194–203, May 1974.

Cohn, D. L. and Melsa, J. L., *A Step by Step Introduction to 8080 Microprocessor Systems,* Dilithium Press, 1977.

Cramer, W. and Kane, G., *68000 Microprocessor Handbook,* 2nd ed., Osborne/McGraw-Hill, 1986.

Danhor, K. J. and Smith, C. L., *Computing System Fundamentals: An Approach Based on Microcomputers,* Addison-Wesley, 1981.

Denning, P. J., "Virtual Memory," *ACM Computing Surveys,* Vol. 2, no. 3, pp 153–159, September 1970.

Electronic Industries Association, Washington, D.C., EIA Standard RS-232-C Interface, Electronic Industries Association, 1969.

Faggin, F., "How VLSI Impacts Computer Architecture," *IEEE Spectrum,* pp 28–31, May 1978.

Feibus, M. and Slater, M., "Pentium Power," *PC Magazine,* April 27, 1993.

Fisher, E. and Jensen, C. W., *Pet and the IEEE 488 Bus (BPIB),* Osborne/McGraw-Hill, 1979.

Friedman, A. D., *Logical Design of Digital Systems,* Computer Science Press, 1975.

Garland, H., *Introduction to Microprocessor System Design,* McGraw-Hill, 1979.

Gay, "6800 Family Memory Management — Part 1," *Electronic Engineering,* pp 39–48, June 1986.

Gibson, G. A. and Liu, Y., *Microprocessors for Engineers and Scientists,* Prentice-Hall, 1980.

Gill, A., *Machine and Assembly Language Programming of the PDP-11,* 2nd ed., Prentice-Hall, 1983.

Girsline, G., *16-Bit Modern Microcomputers, The Intel 8086 Family,* Prentice-Hall, 1985.

Gladstone, B. E., "Comparing Microcomputer Development System Capabilities," *Computer Design,* pp 83–90, February 1979.

Goody, R. W., *Intelligent Microcomputer,* SRA, 1982.

Goody, R., *The Versatile Microcomputer, The Motorola Family,* SRA, 1984.

Greenfield, J. D., *Practical Digital Design Using IC's,* John Wiley & Sons, 1977.

Greenfield, J. D. and Wray, W. C., *Using Microprocessors and Microcomputers: The 6800 Family,* John Wiley & Sons, 1983.

Greenfield, J. D., *Practical Digital Design Using IC's,* John Wiley & Sons, 1983.

Grinich, V. H. and Jackson, H. G., *Introduction to Integrated Circuits,* McGraw-Hill, 1975.

Hall, D. V., *Microprocessors and Digital Systems,* McGraw-Hill, 1980.

Hamacher, V. C., Vranesic, Z. G., and Zaky, S. G., *Computer Organization,* McGraw-Hill, 1978.

Hamacher, V. C., Vranesic, Z. G., and Zaky, S. G., *Computer Organization,* McGraw-Hill, 1984.

Harman, T. L. and Lawson, B., *The Motorola MC68000 Microprocessor Family,* Prentice-Hall, 1984.

Hartman, B., "16-Bit 68000 Microprocessor Concepts on 32-Bit Frontier," MC 68000 Article Reprints, Motorola, pp 50–57, March 1981.

Hayes, J. P., *Computer Architecture and Organization,* McGraw-Hill, 1978.

Hayes, J. P., *Digital System Design and Microprocessors,* McGraw-Hill, 1984.

Haynes, J. L., "Circuit Design with Lotus 1-2-3," *BYTE,* Vol. 10, no. 11, pp 143–156, 1985.

Hewlett-Packard, "HP 64000," *Hewlett-Packard Journal,* 1980.

Hnatek, E. R., *A User's Handbook of Semiconductor Memories,* John Wiley & Sons, 1977.

Holt, C. A., *Electronic Circuits — Digital and Analog,* John Wiley & Sons, 1978.

Horden, I., "Microcontrollers Offer Realtime Robotics Control," *Computer Design,* pp 98–101, October 15, 1985.

IEEE, "Technology 1994" — *The Spectrum,* January 1994.

IEEE, "Technology 1995" — *The Spectrum,* January 1995.

Intel, *Microprocessors and Peripheral Handbook,* Vol. 1, Microprocessors, Intel Corporation, 1988.

Intel, *Microprocessors and Peripheral Handbook,* Vol. 2, Peripheral, Intel Corporation, 1988.

Intel, *80386 Programmer's Reference Manual,* Intel Corporation, 1986.

Intel, *80386 Hardware Reference Manual,* Intel Corporation, 1986.

Intel, 80386 Advance Information, Intel Corporation, 1985.

Intel, *80387 Programmer's Reference Manual,* 1987.

Intel, *Intel 486 Microprocessor Family Programmer's Reference Manual,* 1992.

Intel, *Intel 486 Microprocessor Hardware Reference Manual,* 1992.

Intel, *i960 SA/SB Microprocessor,* 1991.

Intel, *Pentium Processor User's Manual,* 1993.

Intel, *8080 and 8085 Assembly Language Programming Manual,* Intel Corporation, 1978.

Intel, *The 8086 Family User's Family,* Intel Corporation, 1979.

Intel, *Intel Component Data Catalog,* Intel Corporation, 1979.

Intel, *MCS-85 User's Manual,* Intel Corporation, 1978.

Intel, *MCS-86 User's Manual,* Intel Corporation, 1982.

Intel, *Memory Components Handbook,* Intel Corporation, 1982.

Intel, *SDK-85 User's Manual,* Intel Corporation, 1978.

Intel, "Marketing Communications," *The Semiconductor Memory Book,* John Wiley & Sons, 1978.

Isaacson, R. et al., "The Oregon Report — Personal Computing," selected reprints from *IEEE Computer,* pp 226–237.

Johnson, "A Comparison of Mc68000 Family Processors," *BYTE,* pp 205–218, September 1986.

Johnson, C. D., *Process Control Instrumentation Technology,* John Wiley & Sons, 1977.

Johnson, R. C., "Microsystems Exploit Mainframe Methods," *Electronics,* 1981.

Kane, G., *CRT Controller Handbook,* Osborne/McGraw-Hill, 1980.

Kane, G., Hawkins, D., and Leventhal, L., *68000 Assembly Language Programming,* Osborne/McGraw-Hill, 1981.

King, T. and Knight, B., *Programming the MC68000,* Addison-Wesley, 1983.

Krutz, R. L., *Microprocessors and Logic Design,* John Wiley & Sons, 1980.

Krutz, R. L., *Microprocessors and Logic Design,* John Wiley & Sons, 1977.

Lesea, A. and Zaks, R., *Microprocessor Interfacing Techniques,* Sybex, 1978.

Leventhal, L. A., *8080A/8085 Assembly Language Programming,* Osborne/McGraw-Hill, 1978.

Leventhal, L. A., *Introduction to Microprocessors: Software, Hardware Programming,* Prentice-Hall, 1978.

Leventhal, L. and Walsh, C., *Microcomputer Experimentation with the Intel SDK-85,* Prentice-Hall, 1980.

Lewin, M., *Logic Design and Computer Organization,* Addison-Wesley, 1983.

Lipschutz, S., *Essential Computer Mathematics,* Schaum Outline Series, McGraw-Hill, 1982.

MacGregor, Mothersole, Meyer, *The Motorola MC68020,"* IEEE MICRO, pp 101–116, August 1984.

MacGregor, "Diverse Applications Put Spotlight on 68020's Improvements," *Electronic Design,* pp 155–164, February 7, 1985.

MacGregor, "Hardware and Software Strategies for the MC68020," *EDN,* pp 163–168, June 20, 1985.

Mano, M., *Computer System Architecture,* Prentice-Hall, 1983.

McCartney, Groepler, "The 32-Bit 68020's Power Flows Fully Through a Versatile Interface," *Electronic Design,* pp 335–343, January 10, 1985.

Miller, M., Raskin, R., and Rupley, S., "The Pentium that Stole Christmas," *PC Magazine,* February 27, 1995.

MITS-ALTAIR, *S-100 Bus,* MITS, Inc., Albuquerque, NM.

Morse, S., *The 8086/8088 Primer,* 2nd ed., Hayden, 1982.

Motorola, *6809 Applications Notes,* Motorola Corporation, 1978.

Motorola, *MC68000 User's Manual,* Motorola Corporation, 1979.

Motorola, *16-Bit Microprocessor — MC68000 User's Manual,* 4th ed., Prentice-Hall, 1984.

Motorola, *MC68000 16-Bit Microprocessor User's Manual,* Motorola Corporation, 1982.

Motorola, *MC68000 Supplement Material (Technical Training),* Motorola Corporation, 1982.

Motorola, *Microprocessor Data Material,* Motorola Corporation, 1981.

Motorola, *MC68020 User's Manual,* Motorola Corporation, 1985.

Motorola, "MC68020 Course Notes," MTTA20 REV 2, July 1987.

Motorola, "MC68020/68030/88100 Audio Course Notes," 1988.

Motorola, *MC88100 Data Sheets,* Motorola Corporation, 1988.

Motorola, *MC68020 User's Manual,* 2nd ed., MC68020 UM/AD Rev. 1, Prentice-Hall, 1984.

Motorola, *Programmer's Reference Manual* (Includes CPU 32 Instructions), 1989.

Motorola, *MC68040 User's Manual,* 1989.

Motorola, *Power PC 601, RISC Microprocessor User's Manual,* 1993.

Motorola Technical Summary, *32-Bit Virtual Memory Microprocessor,* MC68020 BR243/D. Rev. 2, Motorola Corporation, 1987.

Myers, G. and Budde, D., *The 80960 Microprocessor Architecture,* John Wiley & Sons, 1988.

Osborne, A., *An Introduction to Microprocessors,* Vol. 1, Basic Concepts, rev. ed., Osborne/McGraw-Hill, 1980; 2nd ed., 1982.

Osborne, A. and Kane, G., *The Osborne Four- and Eight-Bit Microprocessor Handbook,* Osborne/McGraw-Hill, 1980.

Osborne, A. and Kane, G., *The Osborne 16-Bit Microprocessor Handbook,* Osborne/McGraw-Hill, 1981.

Rafiquzzaman, M., *Microprocessors and Microcomputer Development Systems — Designing Microprocessor-Based Systems,* Harper and Row, 1984.

Rafiquzzaman, M., *Microprocessors and Microcomputer Development Systems,* John Wiley & Sons, 1984.

Rafiquzzaman, M., *Microcomputer Theory and Applications with the INTEL SDK-85,* 2nd ed., John Wiley & Sons, 1987.

Rafiquzzaman, M., *Microprocessors — Theory and Applications — Intel and Motorola,* Prentice-Hall, 1992.

Rafiquzzaman, M. and Chandra, *Modern Computer Architecture,* West, 1988.

RCA, *Evaluation Kit Manual for the RCA CDP1802 COSMAC Microprocessor,* RCA Solid State Division, Somerville, NJ.

Rector, R. and Alexy, G., *The 8086 Book,* Osborne/McGraw-Hill, 1980.

Reichborn-Kjennerud, G., "Novel Methods of Integer Multiplication and Division," *BYTE,* Vol. 8, no. 6, pp 364–374, June 1983.

Ripps, Mushinsky, "32-Bit Up Speeds Code Design and Execution," *EDN*, pp 163–168, June 27, 1985.

Rockwell International, *Microelectronic Devices Data Catalog*, 1979.

Short, K. L., *Microprocessors and Programmed Logic*, Prentice-Hall, 1981.

Sloan, M. E., *Introduction to Minicomputers and Microcomputers*, Addison-Wesley, 1980.

Smith, J. and Weiss, S., "Power PC 601 and Alpha 21064: A Tale of Two RISCs," *IEEE Computer*, June 1994.

Solomon, "Motorola's Muscular 68020," *Computers & Electronics*, pp 74–79, October 1984.

Sowell, E. F., *Programming in Assembly Language, MACRO II*, Addison-Wesley, 1984.

Starnes, T. W., "Compact Instruction Set Gives the MC68000 Power While Simplifying Its Operation," MC68000 Article Reprints, Motorola, pp 43–47, March 1981.

Strauss, E., *The Waite Group, Inside the 80286*, A Brady Book published by Prentice-Hall, 1986.

Stone, H. S., *Introduction to Computer Architecture*, SRA, 1980.

Stone, H. S., *Microcomputer Interfacing*, Addison-Wesley, 1982.

Streitmatter, G. A. and Fiore, V., *Microprocessors, Theory and Applications*, Reston Publishing, 1979.

Stritter, E. and Gunter, T., "A Microprocessor Architecture for a Changing World: The Motorola 68000," *IEEE Computers*, Vol. 12, no. 2, pp 43–52, February 1970.

Tanenbaum, A. S., *Structured Computer Organization*, Prentice-Hall, 1984.

Teledyne, *Teledyne Semiconductor Catalog*, 1977.

Texas Instruments, *The TTL Data Book*, Vol. 1, 1984.

Texas Instruments, *The TTL Data Book for Design Engineers*, 2nd ed., 1976.

Tocci, R. J. and Laskowski, L. P., *Microprocessors and Microcomputers: Hardware and Software*, Prentice-Hall, 1979.

Triebel, W., *The 80386 DX Microprocessor*, Prentice-Hall, 1992.

Twaddel, "32-Bit Extension to the 68000 Family Addresses 4 GBytes, Runs at 3 MIPS," *EDN*, pp 75–77, July 12, 1984.

Wakerly, J. F., *Microcomputer Architecture and Programming*, John Wiley & Sons, 1981.

Zilog, *Z8000 Advance Specification*, Zilog, Inc., 1978.

Zoch, B., "68020 Dynamically Adjusts Its Data Transfers to Match Peripheral Ports," *Electronic Design*, pp 219–225, January 10, 1985.

Zorpette, G., "Microprocessors — The Beauty of 32-Bits," *IEEE Spectrum*, Vol. 22, no. 9, pp 65–71, September 1985.

Credits

The following material was reprinted by permission of the sources indicated below:

Motorola Corporation, Inc.: **Chapter 1:** Figures 1.10, 1.11, 1.12; **Chapter 5:** Figures 5.1 to 5.5, 5.7, 5.10 to 5.12, 5.14, 5.15, 5.17, 5.21 to 5.24, Tables 5.1, 5.2, 5.14 to 5.16, table on page 332; **Chapter 6:** Examples 6.1 and 6.8, all figures, tables, and graphics; **Chapter 7:** Figures 7.1 to 7.29, Tables 7.1 to 7.7; **Chapter 8:** Figures 8.5 to 8.18, Tables 8.7 to 8.18; **Chapter 9:** Figures 9.7, 9.8; **Appendix B:** data sheets; Appendix D: 68000 Instructions.

Intel Corporation: **Chapter 2:** Figures 2.3, 2.4; 2.6 to 2.8, 2.10 to 2.15, 2.17, 2.23 to 2.25, Tables 2.9, 2.12, 2.13; **Chapter 3:** Figures 3.1 to 3.3, 3.5 to 3.13, 3.24, 3.26a and b, Table 3.2, structure on page 165; **Chapter 4:** Figures 4.1, 4.2, 4.6 to 4.28, Tables 4.1 to 4.3, 4.5 to 4.11, table on page 244; **Chapter 7:** Figure 7.30; **Chapter 8:** Figures 8.1 to 8.4, Tables 8.1 to 8.3; **Chapter 9:** Figures 9.15 , 9.16; **Appendix C:** figures and data sheets; **Appendix E:** 8086 Instructions.

All mnemonics in Tables 2.1 and 3.1 are courtesy of Intel Corporation.

The 80386 microprocessor referred to in text as the i386™; the 80486 as the i486™ and the Pentium as the Pentium™, trademarks of Intel Corporation.

Rafiquzzaman, M., *Microcomputer Theory and Applications with the Intel SDK-85,* John Wiley & Sons, Inc., New York, New York, 1987, reprinted by permission of John Wiley & Sons, Inc.: **Chapter 2:** Tables 2.1 to 2.8, Section 2.6; **Chapter 5:** Tables 5.6 to 5.13, Examples 5.1 to 5.5.

Rafiquzzaman, M., *Microcomputer Theory and Applications with the Intel SDK-85,* John Wiley & Sons, Inc., New York, New York, 1987, reprinted by permission of Prentice Hall, Inc., Englewood Cliffs, New Jersey: **Chapter 2:** Figures 2.18 to 2.20, 2.24, 2.25, 2.29a and b, 2.30, figure on page 92, Sections 2.5 and 2.9.2, pages 53 to 56, 77 to 89, 93 to 99; **Chapter 5:** Figures 5.6, 5.7, 5.11, 5.14, 5.24, 5.31, 5.32, Tables 5.2, 5.4, 5.5, text on pages 280 to 303, 305 to 307, 309 to 312, 314 to 319, 323, 328, 334 to 335, 341 to 346.

Rafiquzzaman, M., *Microprocessors and Microcomputer Development Systems,* copyright John Wiley & Sons, Inc., New York, New York, ©1984, reprinted by permission of John Wiley & Sons, Inc.: **Chapter 9:** Figures 9.1, 9.2, 9.8 to 9.13, Section 9.1.2; **Chapter 10:** Problems 10.2 to 10.13; **Appendix A; Appendix C** (excluding figures and data sheets).

Morse, S. amd Albert, D., *The 80286 Architecture,* John Wiley & Sons, Inc., New York, New York, ©1986, reprinted by permission of John Wiley & Sons, Inc.: **Chapter 4:** Figures 4.3 and 4.4.

Practical Microprocessors — Hardware, Software and Troubleshooting, Hewlett Packard, Palo Alto, California: **Chapter 2:** Figure 2.1.

Burns, D. and Jones, D., Within the 68020, *Electronics and Wireless World,* Surrey, United Kingdom, ©1987: **Chapter 7:** Figures 7.3, 7.27, Example 7.1.

Gay, C., MC68000 Family Memory Management, *Electronic Engineering,* 58(714), June 1986, reprinted by permission of Electronic Engineering, London, United Kingdom, ©1986: **Chapter 6:** Figure 6.28.

Index