AD8088 PROCESSOR CARD

ALF

The

Processor Card Owner's Manual

Complete Instructions

for the

10-5-7 AD8088 Processor Card

including the

Formula Transfer Link

and the

Multiple Event Timer

Copyright © 1982
ALF Products Inc.
1448 Estes
Denver, CO 80215
U.S.A.

Part Number 11-1-7A

The information in this manual was believed to be accurate at the time of publication. Although this manual has been carefully checked for accuracy by our inebriated technical staff, we assume no responsibility for errors or omissions. ALF reserves the right to make changes in the product and/or specifications without notice.

AD8088 PROCESSOR CARD
FULL 1 YEAR WARRANTY

ALF Products Inc. warrants that computer programs will function as described in their associated owner's manuals, and that all other items will be free of defects in material and workmanship. ALF will correct any fault in a computer program (or its manual, or both) or repair or replace (at ALF's choice) any defective item free of charge for one year from the date of sale by ALF.

To obtain warranty service, you must contact ALF at 1448 Estes, Denver, Colorado 80215 or (303) 234-0871 for a service address. You must send the complete product, proof of purchase date, and a detailed description of the difficulty to the service address. You pay for shipment to ALF, ALF pays for shipment back.

Any alteration of the product serial number voids this warranty. This warranty covers only ALF's products, so where local laws permit **ALF will not be liable for consequential damages.**

Ask your state government for details on their "implied warranty" which also covers this product.

The following statements, which shed no new meaning on this warranty, are required by Federal Trade Commission regulations and are meant to simplify warranty language: **"Some states do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you."** and **"This warranty gives you specific legal rights, and you may also have other rights which vary from state to state."**

CONGRATULATIONS!

Your new AD8088 Processor Card is designed to give you faultless performance for years to come. Using the Processor Card with programs like FTL is so easy, you won't need to pay any attention to the card itself. But there are a few features we think you might be interested in knowing. The Processor Card is constructed from top-quality components, carefully selected for optimum performance. To make the card run cooler, several special low-power circuits are used. All integrated circuits are installed in sockets for easy replacement should they ever fail.

The unique design of the Processor Card, made possible by ALF's years of experience in designing Apple-compatible products, includes a careful selection of functions. Each function is included for its usefulness to the user, not for how it will sound in an advertisement. We believe your Processor Card represents the most versatile and reliable design at a reasonable price. Additional functions can be added by means of designed-in expansion capability.

Your choice of the ALF Processor Card shows that you appreciate the same high standard of quality and craftsmanship that we do. ALF's products are chosen by thoughtful and intelligent computer users around the world. We hope you enjoy using your card as much as we've enjoyed creating it for you.

Hardware Design: John Ridges.
Software Design: John Ridges, Philip Tubb, Steve Wells.
Manual: Philip Tubb.
Graphics: Rick Harman.
Photo: Chuck Renstrom.

"Apple" is a trademark of Apple Computer Inc.

CONTENTS

1. INSTALLATION

1-1 Installing the card.
1-1 Tips.
1-2 Preparing the disk.
1-2 Typing "FP".
1-2 The RAM card.
1-2 Radio-TV interference.

2. FTL

2-1 Introduction.
2-1 Using FTL.
2-1 FTL stays in memory.
2-2 Is FTL in memory?
2-2 Auto slot.
2-2 Setting up FTL yourself.

3. MET

3-1 Introduction.
3-1 Setting up MET.
3-1 Picking a resolution.
3-2 Timing.
3-3 The MET READ program.
3-4 Commands.
3-4 View mode.
3-6 Plot mode.
3-6 Changing parameters.
3-7 Reading data directly.
3-7 Setting up MET yourself.

4. PROM ROUTINES

4-1 Introduction.
4-2 Miscellaneous commands.
4-3 The busy flag (and random).
4-4 Integer math commands.
4-4 Floating-point math commands.
4-6 Direct calls in 8088.
4-6 "Available" command codes.

5. HARDWARE

5-1 Memory allocation.
5-1 I/O allocation.
5-1 I/O interface status.
5-1 Expansion port.
5-3 PROM size selection.
5-3 On-board RAM expansion.
5-4 Schematic.
5-7 Repair illustration.
5-8 Photo.

INDEX

1
INSTALLATION

THIS MANUAL DOES NOT COVER USE OF THE APPLE II COMPUTER. READ THE MANUALS SUPPLIED WITH YOUR APPLE, AND FAMILIARIZE YOURSELF WITH ITS USE, BEFORE CONTINUING.

INSTALLING THE CARD

Installation of your Processor Card is easy. Just follow these instructions:

1. Turn the Apple off and remove the top cover (see your Apple manual for details).

2. All Apple-compatible circuit cards are sensitive to static electricity. Care should be taken to protect cards from excessive static. It is best to carry the card in one hand, and touch objects only with the other hand (thus avoiding discharge through the card). After opening the Apple, you should eliminate any static charge you may have accumulated (by walking on carpets, for example) by touching the metal power supply case in the left side of the Apple.

3. Select which peripheral slot you wish to use. The slots are numbered from Ø (left) to 7 (right). Any of the eight slots may be used. You may wish to make a note of which slot you've selected for future reference.

4. Plug the Processor Card into the selected slot. Make sure the card plugs in completely, but avoid using enough pressure to bend the Apple's main board. The main Apple board can be damaged by excessive bending.

5. Replace the top cover (see your Apple manual for details). Installation is now complete, and you can switch the Apple on if you desire.

TIPS

- **Always turn the Apple off before inserting or removing any circuit card.** Considerable damage can occur to the card and your computer otherwise.

- Some of the parts used on the Processor Card are particularly static sensitive and may be protected by other parts on the card. Therefore, no part should be removed from the card unless special anti-static precautions are carefully followed. Leave repairs to professionals.

- Avoid dropping the Processor Card onto a hard surface or severely jolting it. Normal handling will not harm the card, but a jolt can chip the crystal (suspended inside the small metal can).

PREPARING THE DISK

Software for the Processor Card is supplied on a 13-sector DOS 3.2 disk. If you normally use DOS 3.2, you can use the disk as supplied. (You may wish to make a backup copy. Copyright law requires that you put our copyright notice on the backup copy. A label is supplied for your convenience.) If you normally use DOS 3.3 (16-sector), you will need to initialize a DOS 3.3 disk, and then move all the programs from the disk supplied onto your initialized disk (using the MUFFIN program supplied with DOS 3.3, see your DOS 3.3 manual for details). Place the copyright label supplied on the new disk.

TYPING "FP"

Many of the programs supplied with the Processor Card are a combination of BASIC and machine language. These programs will work properly under normal circumstances, but will not work if run after a program which modifies the Applesoft program pointers. The command FP will reset the appropriate pointers. If you're not sure if the programs you've run recently change the program pointers (or if you stop a program with control-C or reset), type FP before running (or loading) any of the Applesoft programs supplied.

THE RAM CARD

FTL requires a Language Card or any of the equivalent 16K RAM cards (or larger RAM cards that simulate a Language Card). This card will be referred to simply as "the RAM card" in this manual. You should not confuse this with the 64K RAM card accessory for the Processor Card, which will be referred to as "the AD64K RAM card". The memory on the AD64K RAM card can be accessed only by the 8Ø88 processor, and not by the Apple.

RADIO-TV INTERFERENCE

Due to the high speeds at which the circuit operates, it naturally generates a small amount of radio frequency energy. Shielding in the computer's case is designed to prevent this energy from escaping. Although the Processor Card and the Apple computer have been carefully designed to reduce emission, radio interference might possibly occur in unusual situations. You can determine if the computer is the source of interference by turning it on and off and observing whether the interference stops and starts. Reorienting television or radio antennas, or moving the radio or computer to a different location or electrical outlet might solve any interference problem. A booklet "How to Identify and Resolve Radio-TV Interference Problems", number ØØ4-ØØØ-ØØ345-4, is available from the U.S. Government Printing Office, Washington, DC 2Ø4Ø2. If you think this rambling paragraph is silly but required by government regulations, you're right.

2
FTL

INTRODUCTION

FTL, the Formula Transfer Link, is a program that routes Applesoft math functions to the Processor Card for computation. Since the 8Ø88 processor is faster than the Apple's 65Ø2 processor, the math functions are computed more quickly. FTL actually consists of three programs. One program, located in the RAM on the card, is written in 8Ø88 machine language. It performs some of the mathematical computations, and calls routines in the card's ROM to perform the other computations. Another "program", located inside the Applesoft language itself, is written in 65Ø2 machine language. It consists of small "patches" at the beginning of each math routine in Applesoft, and each patch routes the particular math function to the Processor Card. The third program is the FTL program supplied on disk with the card. It loads the other two programs into their appropriate memory locations so they can be used.

USING FTL

If you've already booted up and now wish to use FTL, you simply insert the supplied disk (see the instructions in the Installation section) and type RUN FTL. The FTL program will ask which slot your AD8Ø88 Processor Card is in. Type the slot number and press return. All necessary set-up will then be performed by the program.

If your computer has Integer BASIC in ROM, you must already have Applesoft BASIC in your RAM card (or a LANGUAGE NOT AVAILABLE message will be printed). Instructions for loading Applesoft into your RAM card are given in the DOS 3.3 manual. The FTL program will unprotect the RAM card, make the necessary changes to the Applesoft language, and reprotect the RAM card.

If your computer has Applesoft BASIC in ROM, the FTL program will unprotect the RAM card, copy ROM Applesoft into the RAM, make the necessary changes to the Applesoft language, and protect the RAM card.

FTL STAYS IN MEMORY

Once you've run FTL, it will stay in memory even as you run various Applesoft programs. Thus, you can run as many Applesoft programs as you like, and they will all run faster. The following things will cause FTL to be lost: (1) turning off the Apple, (2) booting up with DOS 3.3, (3) loading or running an Integer BASIC program or typing INT, if your computer has Applesoft in ROM, (4) running any program that changes the contents of the RAM card (or which bank is enabled), or (5) running a different Processor Card program (such as MET). If you do any of these things, you will need to run the FTL program again next time you wish to use it.

Booting up a DOS 3.3 disk causes FTL to be lost because the DOS 3.3 boot-up procedure erases the RAM card. DOS 3.2 doesn't erase it. There's no apparent reason for DOS 3.3 to do this, but fortunately the erase routine is easily removed. To initialize a new DOS 3.3 disk that doesn't erase the RAM card,

simply type POKE -16429,173 (on a 48K Apple) before using the INIT command. If you have Applesoft in ROM, you must also do POKE -16432,Ø and POKE -16435,Ø before INIT to keep DOS 3.3 from switching to ROM Applesoft. Copy-protected DOS 3.3 disks are a problem since you can't remove the erase routine. Unless the software supplier has already removed the erase routine, you won't be able to use FTL with programs on a copy-protected DOS 3.3 disk.

If you type INT or try to load or run an Integer BASIC program on an Apple with Applesoft in ROM, FTL will be lost as just mentioned. (If your Apple has Integer BASIC in ROM, you'll be able to switch between Integer BASIC and Applesoft-with-FTL as usual.) If this happens, you can recover FTL just by typing ?PEEK(-16256). If you prefer, you can use PRINT PEEK(-16256) instead. Don't try this peek on a computer with Integer BASIC in ROM (of course, there's no need to).

IS FTL IN MEMORY?

You can tell whether FTL is active or not by typing ?7^2 and pressing return while in Applesoft (] prompt). (If you like, you can type PRINT 7^2 instead.) When 49.ØØØØØØ1 is printed, FTL is not in memory. When 49 is printed, FTL is active. The results are different because FTL uses different algorithms than Applesoft does. The results will differ only very slightly.

AUTO SLOT

The FTL program can be set to use a particular slot when it is run rather than ask you for the AD8Ø88 slot. This would be desirable if you're not going to move the Processor Card from slot to slot frequently. As an example, let's assume your Processor Card is in slot 3. To set FTL to use only slot 3 you would type:

```
]FP
]LOAD FTL
]LIST 1Ø
1Ø SLOT = 8        (the computer prints this line and the ] prompts)
]1Ø SLOT = 3
]SAVE FTL
```

Now when you run FTL it won't ask for the AD8Ø88 slot. To go back to normal (with FTL asking for the slot number), put the SLOT = 3 back to SLOT = 8, using the same procedure as above.

SETTING UP FTL YOURSELF

You can have your own program set up FTL automatically. First, poke the slot number of the AD8Ø88 card times 16 into memory location 6. Next, BLOAD FTL.B into memory and call it. For example, in an Applesoft program for a 48K

Apple with the AD8Ø88 card in slot 3, you could use:
1Ø POKE 6,48 : PRINT CHR$(4);"BLOAD FTL.B,A3712Ø" : CALL 3712Ø
In this example, your program must not use any string variables before the above line is executed, because the FTL.B program is BLOADed into the string variable area. The FTL.B program can be loaded into any memory area that doesn't conflict with anything else and where 12ØØ bytes are available (just substitute the appropriate address after the ",A" in the BLOAD and after the "CALL").

The FTL.B program can easily be moved to another disk. Load it into memory by typing BLOAD FTL.B,A3712Ø and then save it on the desired disk by typing BSAVE FTL.B,A3712Ø,L12ØØ (48K is required to load at this address). Remember you must have a label that reads "FTL © 1982 by ALF" on any disk you put the FTL.B program on. If you plan to sell your program and wish to put FTL.B on the disks to be sold, contact ALF for a license agreement.

3
MET

INTRODUCTION

MET, the Multiple Event Timer, is a program that allows the Processor Card to be used as a timer. MET is written entirely in 8Ø88 machine language. It doesn't access the Apple's memory while it is set up for timing, and thus doesn't affect execution speed of the Apple's processor. This allows it to be used to measure the execution time of programs or routines being run by the Apple's processor. 457 events can be stored with 2K of RAM on the Processor Card (114Ø events with 4K, 1823 with 6K, and 25Ø5 with 8K).

SETTING UP MET

The MET program can be set up simply by inserting the supplied disk (see the instructions in the Installation section) and typing RUN MET. Note that MET cannot be used while FTL is active (unless FTL is used on one Processor Card and MET on another, with MET set up before FTL). The message "FTL IN USE." will be printed if this is attempted. Normally, the set-up program asks for the AD8Ø88 slot and for the desired time resolution. The choices for time resolution are:

NUMBER TO TYPE	RESOLUTION	MAXIMUM DURATION (per event)
Ø	5Ø microseconds	3.2768 seconds
1	1ØØ microseconds	6.5536 seconds
2	5ØØ microseconds	32.768 seconds
3	1 millisecond	65.536 seconds
4	5 milliseconds	327.68 seconds
5	1Ø milliseconds	655.36 seconds
6	5Ø milliseconds	3,276.8 seconds
7	1ØØ milliseconds	6,553.6 seconds

PICKING A RESOLUTION

One microsecond is a millionth of a second. One millisecond is a thousandth of a second. Let's consider the 1ØØ millisecond resolution setting. 1ØØ milliseconds is .1 seconds. If the interval being timed is any duration less than .1 seconds, MET will store it as taking .1 seconds. Any interval .1 to under .2 seconds in duration will be stored as .2 seconds, and so forth. Since MET uses numbers from 1 to 65,536 to store the duration measurements, while in the 1ØØ millisecond resolution mode it can store durations from .1 (1 times .1) to 6,553.6 (65,536 times .1) seconds. Longer durations will "wrap around", so a 6,553.7 second duration would be stored as .1 seconds, 6,553.8 would be stored as .2, and so forth.

Generally, you'll want to pick the smallest resolution that won't wrap around. For example, if you expect to time events that will take up to 1Ø seconds, you'll want to use the 5ØØ microsecond resolution since it can time up

to 32 seconds. The 1ØØ microsecond resolution is too small since it can only time up to 6 seconds. If resolution is critical, you can use the smallest resolution even though it might wrap around. For example, if you're timing something that takes about 5 seconds, you could use the 5Ø microsecond setting and remember to add 3.2768 seconds to the result obtained.

It is important to avoid resolutions which are too large. Only one event may occur in each resolution interval. For example, with 1ØØ millisecond resolution, if two events are sent to the 8Ø88 in the same 1ØØ millisecond interval, only the last one will be recorded.

TIMING

After MET is set by typing RUN MET, you are ready to begin timing. Each event is signaled to the 8Ø88 by writing a reference number to one of its memory addresses. The memory address to use is SLOT*16-16255 where SLOT is the slot number of the AD8Ø88 card (Ø-7). The first poke to that address begins timing, and each subsequent poke causes MET to store the number poked and the time since the previous poke. (The first number poked, which starts the timing, is not stored.) For example, let's say we want to time how long the statement A=7^2 takes to execute. We'll need the smallest resolution, so we RUN MET and ask for resolution number Ø. If the AD8Ø88 is in slot 3, we need to poke at 3*16-16255, which is -162Ø7. Now, type in this program:

```
FP                              (ready for a new program)
1Ø POKE -162Ø7,Ø                (start timing)
3Ø POKE -162Ø7,1                (stop timing)
RUN
```

Notice line 2Ø is skipped. This is where we will put the statement we want to time, but first we need to know how much overhead there is. MET stored the reference number 1 and the time it took between pokes when we ran the program. It is now timing how long it will be until the next poke. We type:

```
2Ø A=7^2                        (the statement we want to time)
RUN
```

The two pokes are still there, with line 2Ø between them. MET has now stored how long it took us to type the line and type RUN, and this is stored with reference number Ø. The time to run line 2Ø, plus the overhead, is stored with reference number 1. Let's also time A=7*7. We type:

```
2Ø A=7*7                        (new statement to time)
RUN
```

Again, MET times how long it took to type the line and stores that with reference number Ø. The timing for the new line 2Ø is stored with a 1 reference number. Now we can type RUN MET READ to see the results. After asking for the AD8Ø88 slot number, MET READ will read the data from the Processor Card, and then show an = prompt. Type VIEW to see the data. The program shows:

#	VALUE	.Ø5MS	TOTAL TIME
Ø	1	159	159
1	Ø	36967	37126
2	1	115Ø	38276
3	Ø	41348	79624
4	1	236	7986Ø

The lines with VALUE's of Ø represent typing time, so they'll vary quite a bit from one try to another. The other times may vary slightly, and of course the TOTAL TIMEs will change to match the times shown. Event #Ø shows 159 times 5Ø microseconds ("MS" means milliseconds, and .Ø5 millseconds equals 5Ø microseconds), or 7,95Ø microseconds. This is the amount of processing time between the two pokes, mostly spent turning -162Ø7 into binary. Note that the actual time could have been as little as 7,9ØØ microseconds, since MET rounds up to the next 5Ø microsecond interval (when in 5Ø microsecond resolution). Also, the crystal time base used has an accuracy of plus or minus Ø.Ø1%, which in this case would be plus or minus Ø.795 microseconds.

Event #2 shows 115Ø intervals and is the timing of the A=7^2 statement plus the poke processing. Subtracting the 159 intervals of poke processing leaves 991 intervals (49,55Ø microseconds) of processing time in the A=7^2 statement.

Event #4 shows 236 intervals, which minus the 159 poke processing time is 77 intervals (3,85Ø microseconds) for the A=7*7 statement. As you can see, 7*7 is a much faster way to square 7 than 7^2 is. To give you an idea of how many 65Ø2 machine instructions the Apple is running to make the computations, it has an average speed of 1,Ø2Ø,5ØØ cycles per second (.9799 microseconds per cycle). The fastest instruction takes 2 cycles, and the slowest takes 7.

If you used one Processor Card for FTL and another for MET, you could see that with FTL 7*7 takes 3,25Ø microseconds and 7^2 takes 3,6ØØ microseconds. This is because the 8Ø88 is faster than the 65Ø2, and because FTL uses a more intelligent exponentiation algorithm than Applesoft.

THE MET READ PROGRAM

The MET READ program is a convenient way to quickly examine timing data either in numeric or plotted form. The MET READ program also obtains the timing data from the Processor Card automatically, and allows it to be saved to disk and loaded from disk. The program is run simply by typing RUN MET READ (see

the Installation section for disk information). Once run, MET READ will ask for the slot number of the AD8Ø88. Type the slot number, or type 8 if you do not have the AD8Ø88 installed at the moment. A message indicating whether or not data was read from the Processor Card and whether or not data present in memory seems to be in MET format will be displayed. Then an = prompt appears, and the program is ready to accept commands. Generally, numbers may be given in decimal or in hexadecimal. Hex numbers must be preceded by a dollar sign ($).

COMMANDS

The **SAVE** command saves the timing data currently in memory to disk as a "B" file. It is used the same as the SAVE command in BASIC. Examples: SAVE TEST FILE or SAVE TEST FILE,D2.

The **LOAD** command loads timing data previously saved with the SAVE command (above) into memory. It is used the same as SAVE. Examples: LOAD TEST FILE or LOAD TEST FILE,D2.

The **CATALOG** command is used to see a catalog of programs on disk. It is the same as the CATALOG command in DOS 3.2 or DOS 3.3. Examples: CATALOG or CATALOG,D2.

The **DELETE** command is used to delete a program from the disk. It is the same as the DELETE command in DOS 3.2 or DOS 3.3. Examples: DELETE TEST FILE or DELETE TEST FILE,D2.

The **VIEW** command is used to see the timing data in numeric form. It has various commands which are described below.

The **PLOT** command is used to see the timing data in plotted form. It has various commands which are described below.

The **QUIT** command is used to leave MET READ and go back to BASIC. To use MET READ after typing QUIT, it must be loaded from disk again.

The **HELP** command lists the available commands.

VIEW MODE

When in view mode, the top of the screen displays several lines of timing data, and the bottom is used to type in commands. The heading at the top of the screen could read:

```
# VALUE          .Ø5MS       TOTAL TIME
```

In the # column the consecutive event number (starting with Ø) for each event is shown. In the VALUE column is the reference value for each event. The next column is the event duration, in whatever units are shown in the heading. Finally, in the TOTAL TIME column the total of all event durations so far is shown. The following commands are available from view mode:

The **-> key** is used to see one more event. The screen rolls up one line so the new event can appear near the bottom of the screen.

The **<- key** is used to see one previous event. The screen rolls down one line so the previous event can appear near the top of the screen.

Note that the -> and <- keys can be used to scroll the screen only when the flashing cursor is not present. (When the cursor is present, these keys are used as backspace and forward space as usual.) Pressing the return key will remove the flashing cursor and allow the -> and <- keys to be used for scrolling.

The **GO TO** command is used to begin viewing with the desired event. The consecutive event number to begin the display with must be specified. Example: GO TO 25 (the display begins with event # 25, the 26th event).

The **UNITS** command is used to select what unit of time all durations will be shown in. UNITS AUTO selects the resolution the MET program was set for when the data was recorded. UNITS MS selects milliseconds. UNITS SEC selects seconds. UNITS MIN selects minutes.

The **EVENTS** command is used to limit the display to a particular range of consecutive event numbers. Also, TOTAL TIME will be computed starting with the first event in the range selected. A dash is used to separate the first and last event numbers to be displayed. This command also affects the DUMP and PRINTER commands. Examples: EVENTS 5Ø-1ØØ (show only events 5Ø through 1ØØ, inclusive) or EVENTS -1ØØ (show events Ø through 1ØØ) or EVENTS - (show all events) or EVENTS 5Ø- (show all events from # 5Ø on).

The **VALUES** command is used to limit the display to a particular range of reference values. A dash is used to separate the lowest and highest reference values to be used. Only events with reference values in the specified range will be displayed. This command also affects the DUMP and PRINTER commands. Examples: VALUES 5Ø-1ØØ (show only events with values from 5Ø to 1ØØ, inclusive), VALUES -1ØØ (show only events with values Ø through 1ØØ) or VALUES - (show all events) or VALUES 5Ø- (show only events with values 5Ø through 255).

The **BASE** command is used to select the display base for the VALUE column. BASE DEC selects decimal, and BASE HEX selects hexadecimal. All hexadecimal numbers are preceded by a $ when displayed.

Note that the **UNITS, EVENTS, VALUES,** and **BASE** commands will show their current settings if the command is typed alone (just UNITS, EVENTS, VALUES, or BASE and press return).

The **DUMP** command is used to create a text or "execute" file on disk containing the reference values and durations of all events selected for display by the EVENTS and VALUES commands. It is used the same as the SAVE command (above.) The first line in the text file contains a number from Ø to 7 indicating the resolution MET was set for when the data was recorded, a comma, and a number indicating the number of events which follow. Each event line contains a number from Ø to 255 indicating the reference value, a comma, a number from 1 to 65536 indicating the duration since the previous event (in time units as specified by the resolution indication in the first text line), a comma, and a number from 1 to 16416768Ø indicating the "total time" (in the same units as the duration). The last line is -1,-1,-1 to indicate the end of the file. Command examples: DUMP FINAL DATA or DUMP FINAL DATA,D2

The **PRINTER** command is used to print the events selected by the EVENTS and VALUES commands to a PR#-compatible printer. The slot number of the printer must be specified. Each page will begin with two blank lines, the heading, another blank line, 6Ø lines of events (changeable as described below), and two blank lines. Command example: PRINTER 1.

The **EXIT** command is used to exit view mode and return to the main program. To exit to BASIC, type **QUIT.**

PLOT MODE

When in plot mode, the top part of the screen is in high-resolution graphics mode showing a plot of the timing data; and the bottom part of the screen is used for typing commands. The horizontal axis of the plot shows various time durations, and the vertical axis shows the number of events with a given duration. The lines below the graphics area indicate the scale of each axis and the range of the horizontal axis. For example,

1Ø EVENTS/DIV 5Ø SEC/DIV
TIMES .Ø1-655.36

indicates the vertical axis has 1Ø events per division (each division is indicated by a missing dot in the vertical axis), the horizontal axis has 5Ø seconds per division (each division is indicated by a dot below the horizontal axis), and the range of times shown is .Ø1 seconds to 655.36 seconds. The same plot might be shown as:

1Ø EVENTS/DIV 5ØØØ (1ØMS)/DIV
TIMES 1-65536

which means 1Ø events per division, 5ØØØ times 1Ø milliseconds (5Ø seconds) per division, and times from 1 times 1Ø milliseconds (.Ø1 seconds) to 65536 times 1Ø milliseconds (655.36 seconds). Note that the display does not show an integral number of divisions either vertically or horizontally. The following commands are available from plot mode:

The **UNITS, EVENTS, VALUES, EXIT,** and **QUIT** commands are the same as in view mode. Settings made while in view mode will carry over to plot mode and vice versa.

The **TIMES** command selects a range of durations to be displayed. (The current TIMES setting is always displayed on the next-to-top text line.) The current UNITS command setting specifies what units the TIMES range should be given in. Numbers can contain a decimal point, but may not be in hexadecimal. Example commands while in UNITS SEC setting: TIMES 5Ø-1ØØ (shows durations from 5Ø seconds to 1ØØ seconds, inclusive) or TIMES -1ØØ (shows durations up to and including 1ØØ seconds) or TIMES - (shows all durations) or TIMES 5Ø- (shows durations from 5Ø seconds up).

CHANGING PARAMETERS

Both the MET program and the MET READ program have parameters in line 1Ø

that can be changed if desired. This is done by loading the program, listing line 1Ø, retyping the line changing only the appropriate numbers, and saving the program. Neither program should be changed after it has been run. The length of line 1Ø must remain the same. If necessary, leading Ø's should be added to numbers. For example, line 1Ø in MET reads:

1Ø SLOT = 8 : BUFFER = 26624 : TIME = 8

To change buffer to 8192, type:

1Ø SLOT = 8 : BUFFER = Ø8192 : TIME = 8

The SLOT value is the slot number the Processor Card is plugged into, or 8 to have the program ask for the slot number. The BUFFER value is the memory address where the 8Ø88 will write the timing data when commanded to do so, or (for MET READ) the address where the timing data is expected to be. (BUFFER must the set the same in MET and MET READ if MET READ will be used to read the data. 26624 is virtually the only address MET READ can use.) If BUFFER is set to Ø, the program will ask for the buffer address. The TIME value (not present in MET READ) is the desired resolution (Ø to 7) or 8 to have the program ask for the resolution. In MET READ, there is also a line 2Ø which can be changed in a similar fashion. It is 2Ø LINES = 66 and it indicates the desired number of lines per page when the PRINTER command is used. Again, care must be taken that the length of the line is not changed.

READING DATA DIRECTLY

If you do not wish to use MET READ to read the timing data, you can read it yourself. When 248 (hex F8) is poked at location SLOT*16-16256 (where SLOT is the slot number the Processor Card is plugged into), the 8Ø88 will write the timing data into memory, starting at the address indicated by BUFFER in the MET program. (Sending a Ø to this address will cause the data to be discarded and MET to be stopped.) The first byte of data will be the resolution selected plus 48. The next two bytes will be the length of the data (this number divided by 3 minus 1 indicates the number of events which follow). All following bytes are the events, consisting of one byte indicating the reference value and two bytes indicating the time since the previous event (or, for the first event, the time since the "start" command). A 65536 duration is stored as Ø. The time is given in units as specified by the selected resolution. All two byte numbers appear low byte first. There is no end marker in the data.

SETTING UP MET YOURSELF

You can have your own Applesoft program set up MET automatically. First, poke the slot number of the AD8Ø88 card times 16 into memory location 6. Poke the desired resolution (Ø to 7) into location 7. Poke the buffer address (where the timing data will eventually be stored) into locations 8 (low byte) and 9 (high byte), use 26624 if you will be reading the results with MET READ. Next BLOAD MET.B into memory and call it. For example, in an Applesoft program for a 48K

Apple with the AD8Ø88 card in slot 3, you could use:

```
1Ø POKE 6,48 : POKE 7,Ø : POKE 8,112 : POKE 9,1Ø2
2Ø PRINT CHR$(4);"BLOAD MET.B,A37888" : CALL 37888
```

In this example, your program must not use any string variables before the above lines are executed, because the MET.B program is BLOADed into the string variable area. The MET.B program can be loaded into any memory area that doesn't conflict with anything else and where 5ØØ bytes are available (just substitute the appropriate address after the ",A" in the BLOAD and after the "CALL").

The MET.B program can easily be moved to another disk. Load it into memory by typing BLOAD MET.B,A37888 and then save it on the desired disk by typing BSAVE MET.B,A37888,L5ØØ (48K is required to load at this address). Remember you must have a label that reads "MET © 1982 by ALF" on any disk you put the MET.B program on. If you plan to sell your program and wish to put MET.B on the disks to be sold, contact ALF for a license agreement.

If MET is loaded and timing is stopped (by sending a Ø to location SLOT*16-16256 or by reading the timing data), it can be restarted with any desired resolution by poking the resolution number plus 48 to location SLOT*16-16256. This will work only if MET is still loaded in the 8Ø88's RAM (programs like FTL remove MET).

4
PROM ROUTINES

INTRODUCTION

Communication with the Processor Card is done mainly through 16 I/O ports, at Apple memory addresses CØxp (where x is 8 for slot Ø to F for slot 7, and p is the port number, Ø to F). In decimal, SLOT*16-16256+P (where SLOT is the slot number, Ø to 7, and P is the port number, Ø to 15). Data can be written to any of these ports. When port Ø is read, the most significant bit (bit 7) is Ø if the ports should not be written to or 1 if the Processor Card is ready to receive data. Bits 6 through Ø of the data read from port Ø are random. Ports 1 through 15 should not be read.

The following conventions of the PROM routines are recommended for all 8Ø88 programs. (1) Port Ø is used as the command register. All other ports are used to pass parameters. (2) Sending a Ø to port Ø causes any 8Ø88 program to stop operation and jump to the main PROM idle loop. (3) Whenever the most significant bit of the data read from port Ø is 1 (i.e., the card is ready to accept data), the 8Ø88 is not accessing the 65Ø2's memory (and so speed critical Apple routines can proceed).

The 1-9-1-1 PROM routines use the 15 parameter ports to set 15 stored parameters. Commands sent to the command port can access parameters already set via the 15 parameter ports. The ports are assigned as follows:

Port	Function
Ø	Commands.
1-11	Reserved for the 1Ø-5-8 Graphics Subsystem.
12	Address A (low byte).
13	Address A (high byte).
14	Address B (low byte).
15	Address B (high byte).

The commands are as follows:

Decimal	Hex	Function
Ø	ØØ	Reset to main idle loop.
1-28	Ø1-1C	Reserved for the 1Ø-5-8 Graphics Subsystem.
29-32	1D-2Ø	2-byte integer math functions.
33-47	21-2F	5-byte floating-point math functions.
48-247	3Ø-F7	(Available.)
248	F8	Reserved for MET.
249-25Ø	F9-FA	Reserved for special function.
251-255	FB-FF	Miscellaneous.

MISCELLANEOUS COMMANDS

The **SEQUENCE** command (code 251 or FB) is used to have the Processor Card read and execute a sequence of commands in the Apple's memory. Address B must already be set to point to the command sequence. Each command in the sequence must consist of two bytes. The first byte indicates which port number (Ø-15) the second byte applies to. For example, the bytes ØE 27 in a command sequence would have the same effect as sending a 27 to port E (14). Another SEQUENCE command in the command sequence causes any remaining commands in the sequence to be ignored; processing continues at the address specified by the new SEQUENCE command. A sequence is ended by placing a RESET command in the sequence (ØØ ØØ).

The **RANDOM** command (code 252 or FC) is used to obtain a "random" number. Address B must already be set to the Apple memory address where the 1 byte result will be stored. All 8 bits in the stored result will be "random". The formula used is NEW RND=(OLD RND * 2 + (BIT 2 XOR BIT 3Ø)) MOD 2147483648, where BIT 2 and BIT 3Ø are bits 2 and 3Ø from OLD RND. This formula is run continuously by the main idle loop. The least significant 8 bits of NEW RND are returned by the RANDOM command.

The **SET MEMORY** command (code 253 or FD) is used to set a block of 8Ø88 or Apple memory to a selected value. Address B must already be set to the Apple memory address where the following table is located. The first four bytes in the table are a 4-byte memory reference indicating the start of the memory area to set. The next two bytes are the length of the memory area to be set, low byte first. The last byte in the table is the value to be written into the block of memory.

The **MOVE DATA** command (code 254 or FE) is used to move a block of data from one place in memory to another. Address B must already be set to the Apple memory address where the following table is located. The first four bytes in the table are a 4-byte memory reference indicating the destination memory address. The next four bytes are a 4-byte memory reference indicating the source memory address. (Data is moved from the source area to the destination area.) The source and destination areas must not overlap if the destination address is greater than the source address. The final two bytes indicate the number of bytes to move, low byte first.

The **CALL** command (code 255 or FF) is used to cause the 8Ø88 to call a subroutine from the main idle loop. When the subroutine returns (using an inter-segment return), the main idle loop will continue. Address B must already be set to the Apple memory address where a 4-byte memory reference (the address to call) is stored.

The first two bytes of a **4-byte memory reference** are the address of an Apple memory location or the offset of an 8Ø88 memory location, low byte first. The final two bytes are the 8Ø88 segment number, low byte first. Apple memory is accessed with a segment number of 1ØØØ hex or 4Ø96 decimal.

THE BUSY FLAG (AND RANDOM)

Note that before a command is sent, the busy flag must be examined. A typical assembly language routine to read a random number into location 15 might be:

```
RANDOM   LDX SLOT16    SLOT16 CONTAINS AD8Ø88 SLOT NUMBER * 16.
         JSR BUSY      WAIT UNTIL AD8Ø88 IS READY.
         LDA #15       SEND LOW BYTE OF ADDRESS B.
         STA $CØ8E,X
         JSR BUSY      WAIT UNTIL AD8Ø88 IS READY.
         LDA #Ø        SEND HIGH BYTE OF ADDRESS B.
         STA $CØ8F,X
         JSR BUSY      WAIT UNTIL AD8Ø88 IS READY.
         LDA #252      SEND RANDOM NUMBER COMMAND.
         STA $CØ8Ø,X
         JSR BUSY      WAIT UNTIL RESULT IS IN APPLE MEMORY.
         LDA 15        PUT RANDOM NUMBER IN A.
         RTS           RETURN TO CALLING ROUTINE.
BUSY     LDA $CØ8Ø,X   READ BUSY/READY STATUS.
         BPL BUSY      (AD8Ø88 IS BUSY, SO WAIT.)
         RTS           AD8Ø88 IS READY, SO RETURN.
```

Integer BASIC and Applesoft BASIC are so slow that it is generally not necessary to check the busy flags when using the RANDOM command. (The RANDOM command can be used while FTL is active, but not while timing with MET.) From BASIC, POKE SLOT*16-16242,6 : POKE SLOT*16-16241,Ø is used to set up address B any time before random numbers will be needed. To obtain a random number, use POKE SLOT*16-16256,252 : R=PEEK(6). Note that SLOT must be the AD8Ø88 slot number (Ø to 7) and R will be set to a random integer from Ø to 255. To get n random integers with the smallest result being x, use INT(R/256*N)+X in Applesoft BASIC or R*N/256+X in Integer BASIC. In Integer BASIC, n must be less than 128. When x is Ø, this formula gives the same range as Integer BASIC's RND(N), except Integer BASIC's random algorithm repeats much sooner than the AD8Ø88's algorithm and so the AD8Ø88's numbers may appear more "random". In Applesoft BASIC, n must be less than 257. POKE SLOT*16-16256,252 : R=PEEK(6) : POKE SLOT*16-16256,252 : R=R+256*PEEK(6) can be used to get a random integer from Ø to 65535 and thus INT(R/65536*N)+X can be used for values of n up to 65536 or for more even distribution for small values of n where 256/n is not an integer. Note that the sequence of numbers from the AD8Ø88's random is not repeatable (especially since new random numbers are continuously computed by the main idle loop), whereas Applesoft only has repeatable sequences. The AD8Ø88's random can be a significant advantage where repeatable (or predictable) numbers are not desired.

INTEGER MATH COMMANDS

The **UNSIGNED INTEGER MULTIPLY** command (code 29 or 1D) is used to multiply two unsigned 2-byte integers. Address B must already be set to the Apple memory address where the two multiplicands are stored. Each multiplicand consists of two bytes, stored low byte first. The 4-byte product is stored, low byte first, over the multiplicands.

The **SIGNED INTEGER MULTIPLY** command (code 3Ø or 1E) is used to multiply two signed 2-byte integers. Address B must already be set to the Apple memory address where the two multiplicands are stored. Each multiplicand consists of two bytes, stored low byte first, in two's complement form. The 4-byte two's complement product is stored, low byte first, over the multiplicands.

The **UNSIGNED INTEGER DIVIDE** command (code 31 or 1F) is used to divide a 4-byte unsigned integer by a 2-byte unsigned integer. Address B must already be set to the Apple memory address where the dividend and divisor are stored. The dividend is stored first. The 2-byte quotient will be stored over the first two bytes of the dividend, and the 2-byte remainder will be stored over the last two bytes of the dividend. The divisor is left unmodified. All numbers are stored low byte first. If an overflow occurs, the quotient will be 8ØØØ hex and the remainder is undetermined.

The **SIGNED INTEGER DIVIDE** command (code 32 or 2Ø hex) is used to divide a 4-byte signed integer by a 2-byte signed integer. Address B must already be set to the Apple memory address where the dividend and divisor are stored. The dividend is stored first. The 2-byte quotient will be stored over the first two bytes of the dividend, and the 2-byte remainder will be stored over the last two bytes of the dividend. The divisor is left unmodified. All numbers are two's complement and stored low byte first. If an overflow occurs, the quotient will be 8ØØØ hex and the remainder is undetermined.

FLOATING-POINT MATH COMMANDS

All floating-point numbers are 5 bytes long. The first 4 bytes contain the mantissa in two's complement format, low byte first. The binary point precedes the most significant mantissa bit (bit 7 of the most significant mantissa byte is the mantissa sign, bit 6 is the most significant bit). The fifth byte is the base 2 exponent in two's complement with the sign bit complemented. Zero is represented by all 5 bytes being zero. All non-zero numbers must be normalized (the sign bit of the mantissa must be the complement of the most significant bit of the mantissa).

All floating-point operations work on a "stack" basis. The argument(s) are "popped" from a stack in the Apple's memory, and the result is "pushed" onto the stack. The B address is always the address of the first byte (lowest mantissa byte) of the top item in the stack. (Thus, the B address must always be properly set before using any of the floating-point operations.) The stack expands toward lower-numbered memory addresses.

The **PUSH** command (code 33 or 21 hex) is used to push a floating-point number onto the stack. It decrements the B address by 5. The Apple program must keep its own copy of the B address so the floating-point number can be written into the new top-of-stack location reflected by the new B address value.

The **POP** command (code 34 or 22 hex) is used to pop a floating-point number off the stack. It increments the B address by 5.

The **FLOAT** command (code 35 or 23 hex) is used to convert an integer into a floating point number. The top-of-stack must be a 4-byte signed two's complement integer stored low byte first in the mantissa bytes, and the byte which is normally the exponent must be 159 (9F hex). The FLOAT command consists only of the normalize function.

The **FIX** command (code 36 or 24 hex) computes the greatest integer function for the top-of-stack. The result is not normalized, and the exponent is always 159; thus the mantissa bytes represent a two's complement 4-byte integer.

The following commands perform the indicated function, taking the required argument(s) off the stack, and pushing the result on the stack. Error conditions are not reported. Overflows are returned as (2^32-1)*2^127 or -(2^159), and underflows are returned as Ø. "TOS" is top-of-stack number, and "NOS" is next-to-top-of-stack number.

DECIMAL CODE	HEX CODE	FUNCTION
37	25	TOS=NOS+TOS
38	26	TOS=NOS-TOS
39	27	TOS=NOS*TOS
4Ø	28	TOS=NOS/TOS
41	29	TOS=-TOS
42	2A	TOS=LOG TOS (base 2)
43	2B	TOS=2^TOS
44	2C	TOS=NOS^TOS
45	2D	TOS=SIN TOS (radians)
46	2E	TOS=COS TOS (radians)
47	2F	TOS=ATN TOS (radians)

The following constants may be helpful:

LOW		HIGH	EXP	VALUE
95	1D	55	5C	81	log base 2 of e
FC	ØB	B9	58	8Ø	log base e of 2
42	4D	1Ø	4D	7F	log base 1Ø of 2
51	ED	87	64	82	pi

DIRECT CALLS IN 8088

The floating-point routines can be called directly from 8Ø88 programs. [8Ø88 machine language programming is not described in this manual. The Intel "iAPX 88 Book", available from Intel Corporation (3Ø65 Bowers Avenue; Santa Clara, CA 95Ø51; Attn.: Literature Department) describes the 8Ø88 processor. This book is also available from ALF, order number 11-2-2.] Intra-segment calls must be used. (This can be accomplished by setting the code segment to FFØØ hex and locating your program in the on-board RAM.) The arguments shown as NOS and TOS (above) must be placed in 8Ø88 registers. DI is the two least significant bytes of the mantissa of TOS, BP is the two most significant mantissa bytes, and DL is the exponent. For NOS, these registers are SI, BX, and CL. The addresses are as follows:

FLOAT	ØBFA
FIX	ØB97
ADD	ØBB4
SUBTRACT	ØBB1
MULTIPLY	ØC36
DIVIDE	ØC7F
NEGATE	ØC25
LOGARITHM	ØD3B
ANTI-LOG	ØD94
EXPONENTIATION	ØD82
SINE	ØDFC
COSINE	ØDF1
ARCTANGENT	ØE4E

"AVAILABLE" COMMAND CODES

The available command codes (48-247 or 3Ø-F7) can be used to call 8Ø88 subroutines. Codes must be used from 48 up. With data segment set to ØØØØ, location 2Ø (14 hex) must be set to the first unused command code (this is normally set to 48, of course). Locations 21-24 (15-18 hex) must be set to the 4-byte memory reference (Dword) of the user-provided command address table (containing the 4-byte memory references (Dwords) of each command, starting with code 48). These 5 bytes can be set using the MOVE DATA command. Your command will be called with an inter-segment call. AL will be the command code used. Locations 3Ø-44 (1E-2C) are the parameters sent to ports 1-15. Your command must return with an inter-segment return, and all registers may be changed except SP and SS. RAM locations Ø-511 (Ø-1FF) must not be changed (except 2Ø-24, 14-18 hex). Current stack contents, beginning at 2Ø47 (7FF) and going to SP must not be changed (SS is ØØØØ).

5
HARDWARE

MEMORY ALLOCATION

ADDRESS	FUNCTION
ØØØØØ-ØØ7FF	On-board RAM.
ØØ8ØØ-Ø1FFF	Optional on-board RAM.
Ø2ØØØ-ØFFFF	Reserved.
1ØØØØ-1FFFF	Apple memory.
2ØØØØ-2FFFF	Expansion port.
3ØØØØ-FEFFF	Reserved.
FFØØØ-FFFFF	On-board PROM.

I/O ALLOCATION

ADDRESS	FUNCTION
ØØ	(write, with any data) Clear busy flag.
ØØ	(read) Data from Apple I/O interface.
Ø1	(read) I/O interface status.
Ø2-7F	Reserved.
8Ø-FF	Expansion port.

I/O INTERFACE STATUS

Most significant bit is Ø when data is present (to be read from I/O address ØØ). Four least significant bits are the address the data was written to by the Apple. See the PROM Routines section for a description of Apple I/O from the Apple side.

EXPANSION PORT

PIN #	NAME	DESCRIPTION
1	A1	Address line 1. Drives 16 LS loads.
2	AØ	Address line Ø. Drives 16 LS loads.
3	A5	Address line 5. Drives 16 LS loads.
4	A4	Address line 4. Drives 16 LS loads.
5	A6	Address line 6. Drives 16 LS loads.
6	A2	Address line 2. Drives 16 LS loads.
7	A7	Address line 7. Drives 16 LS loads.
8	A3	Address line 3. Drives 16 LS loads.
9	DMAE	DMA cycle enable. 8 LS loads.
1Ø	EXTA	Expansion port memory enable. Drives 2Ø LS loads.
11	TRFE	DMA transfer enable. Drives 16 LS loads.
12	DØ	Data line Ø. 3 LS loads, drives 2.

13	ALE	Address latch enable. Drives 4 LS loads.
14	D1	Data line 1. 3 LS loads, drives 2.
15	$\overline{DEN}$	Data enable. Drives 3 LS loads.
16	D2	Data line 2. 3 LS loads, drives 2.
17	DT/$\overline{R}$	Data transmit/receive. Drives 2 LS loads.
18	D3	Data line 3. 3 LS loads, drives 2.
19	IO/$\overline{M}$	Input-output/memory. Drives 3 LS loads.
2Ø	D4	Data line 4. 3 LS loads, drives 2.
21	$\overline{WR}$	Write. Drives 3 LS loads.
22	D5	Data line 5. 3 LS loads, drives 2.
23	GND	Signal ground.
24	D6	Data line 6. 3 LS loads, drives 2.
25	$\overline{RD}$	Read. Drives 3 LS loads.
26	D7	Data line 7. 3 LS loads, drives 2.
27	GND	Signal ground.
28	A8	Address line 8. Drives 2 LS loads.
29	GND	Signal ground.
3Ø	A9	Address line 9. Drives 2 LS loads.
31	RESET	Reset. Drives 1Ø LS loads.
32	A1Ø	Address line 1Ø. Drives 2 LS loads.
33	CLK	Clock. Drives 18 LS loads.
34	A11	Address line 11. Drives 2 LS loads.
35	A15	Address line 15. Drives 2 LS loads.
36	A12	Address line 12. Drives 2 LS loads.
37	XRDY	Expansion port ready. 8 LS loads.
38	A13	Address line 13. Drives 2 LS loads.
39	PCLK	Peripheral clock. Drives 12 LS loads.
4Ø	A14	Address line 14. Drives 2 LS loads.

The following lines are not normal 8Ø88 lines:

DMAE: when held low through the expansion port, it causes the Processor Card to generate an Apple DMA cycle. During a DMA write, the data to be written must be enabled directly to the Apple bus data lines. TRFE indicates when data must be enabled on the Apple bus (or can be read from the Apple bus).

EXTA: goes low when any location from 2ØØØØ-2FFFF is accessed.

TRFE: goes low when the Apple bus is in DMA mode (whether requested with

DMAE or through normal Processor Card operations).

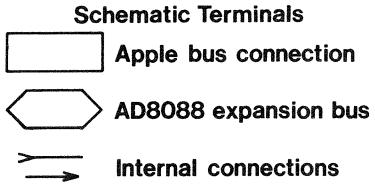
The ribbon cable mating connector for the expansion port is a 3M Scotchflex plug connector, 3M part number 3324-ØØØ1.

PROM SIZE SELECTION

The 2K/4K PROM jumper is used to select a 2K (2716) or 4K (2732) PROM in socket position "d". When a wire is connected from the pad with an arrow to the pad marked "2", a 2K PROM is selected. When a wire is connected from the pad with an arrow to the pad marked "4", a 4K PROM is selected. 35Ø ns PROMS must be used. See the schematic for proper address and data line mapping.

ON-BOARD RAM EXPANSION

The on-board RAM can be expanded from 2K to 8K in 2K increments simply by inserting additional memory chips. To expand the memory from 2K to 4K, a chip is plugged into socket position "c1". To expand to 6K, also insert a chip in position "c2". For 8K, also use position "c3". The Toshiba TMM2Ø16P memory chip should be used. (Other 2K by 8 RAM chips with 15Ø ns access times may be compatible.) Extreme care must be used to protect the memory chip from static electricity once it is removed from its protective packaging. Insert or remove memory chips only when the card is not plugged into the Apple. Care must be taken that the pin 1 indicator is to the top of the circuit card (the same as the factory-installed chip).



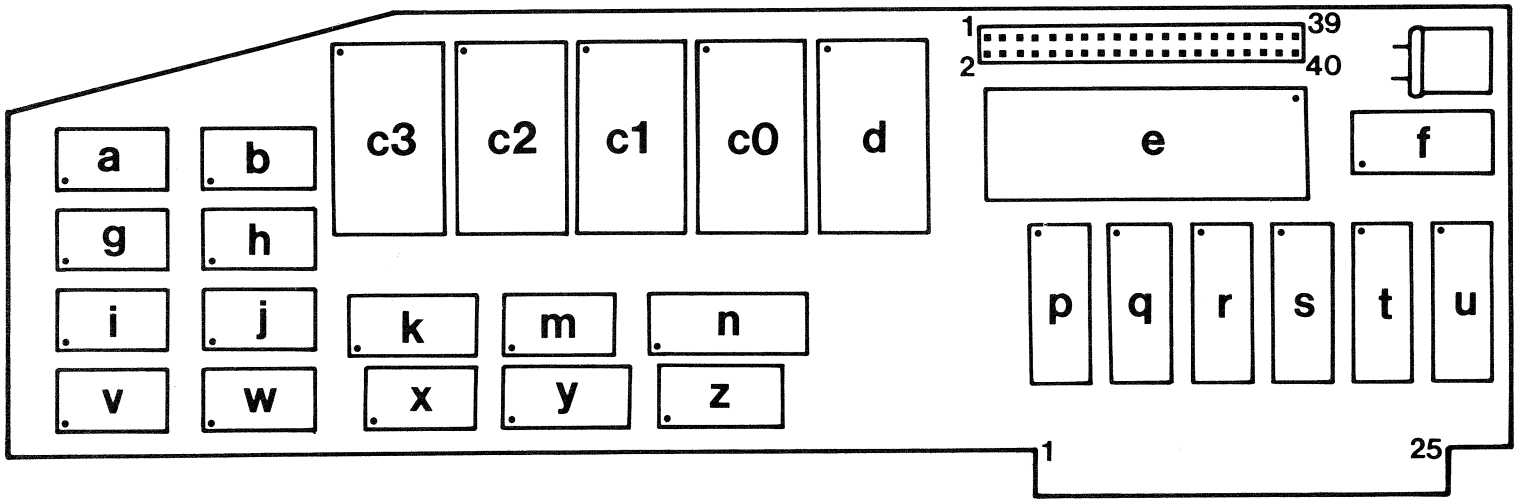
a	b	c3	c2	c1	c0	d	e	f
74LS00	74LS74A	TMM2016	TMM2016	TMM2016	TMM2016	2732A	8088	8284

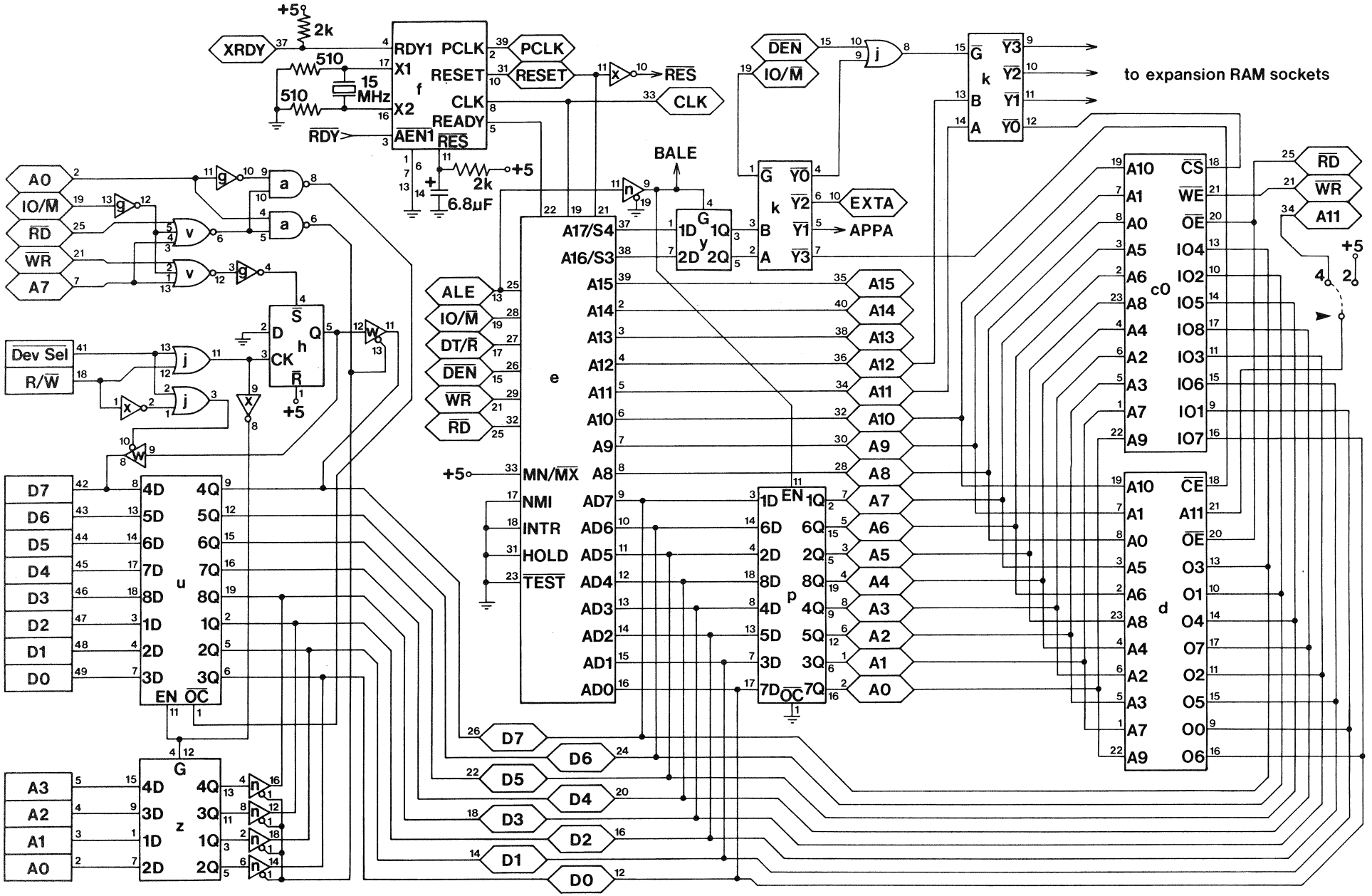
g	h
74LS04	74LS74A

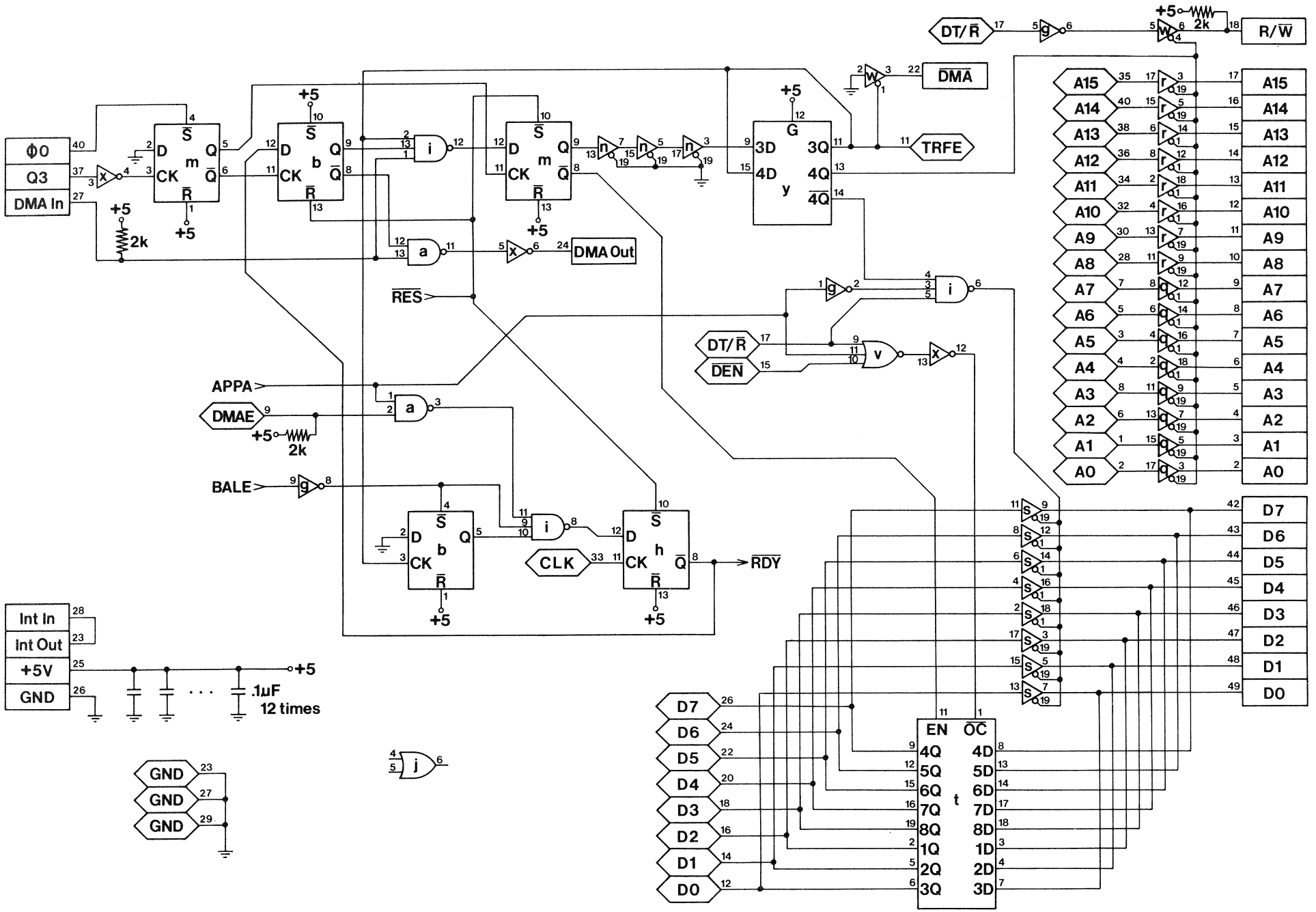
p	r	t
74SC373	74SC244	74SC373

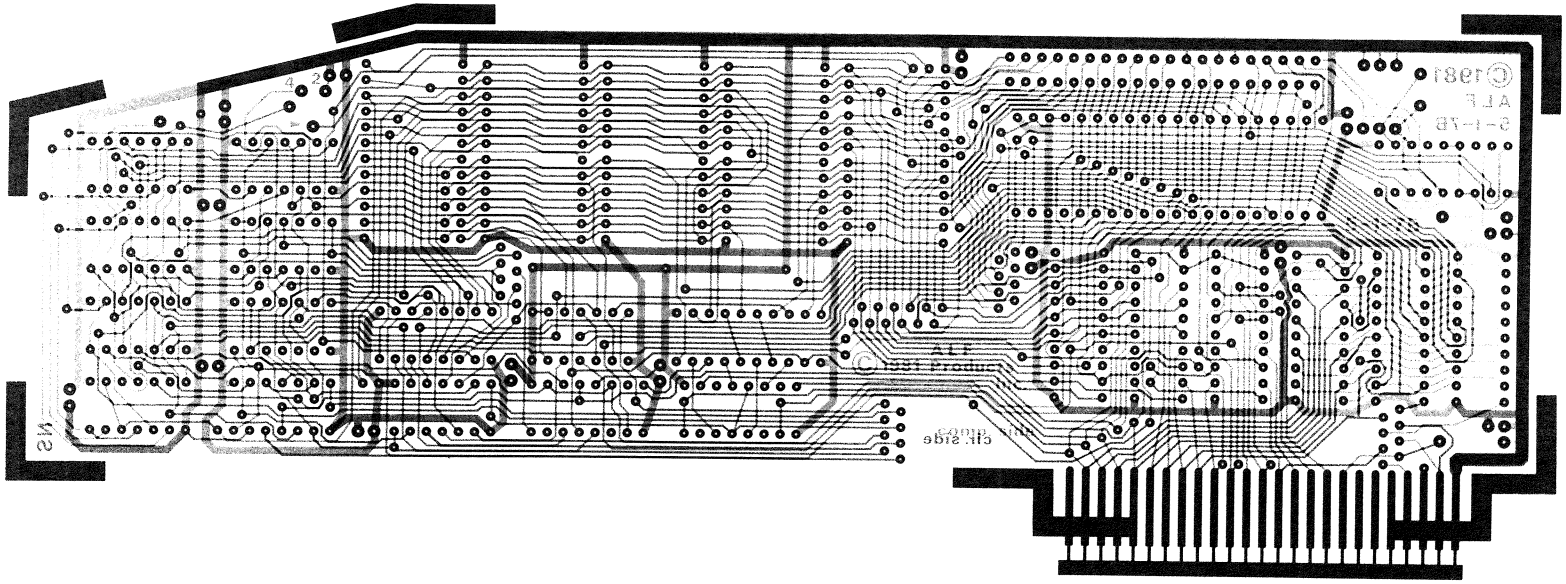
i	j	k	m	n	q	s	u
74LS10	74LS32	74LS139	74LS74A	74SC244	74SC244	74SC244	74SC373

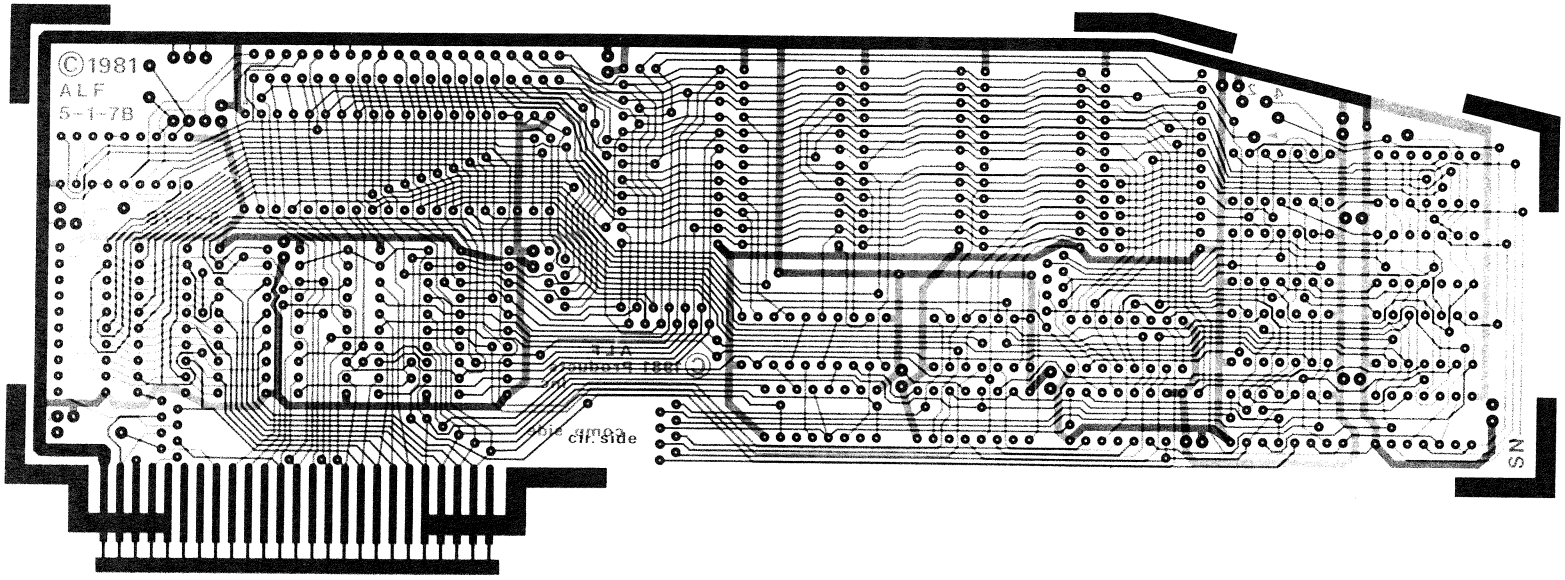
v	w	x	y	z
74LS27	74LS125	74LS04	74LS375	74LS375

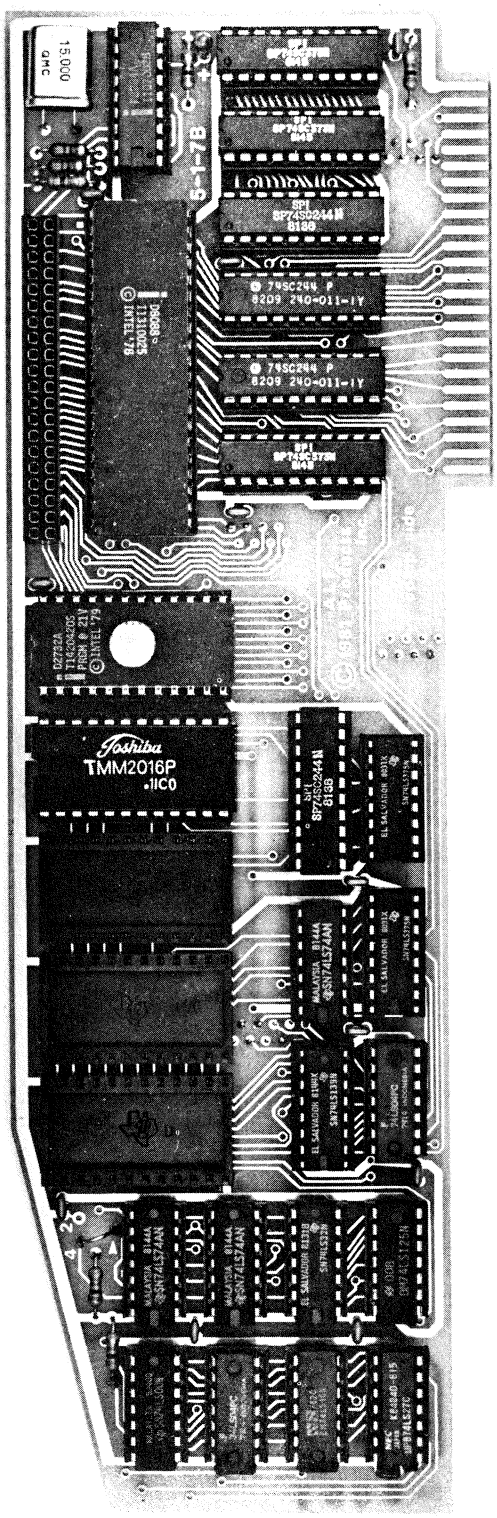



+5
2k
XRDY
510
15 MHz
510
RDY1
X1
X2
AEN1
RDY
PCLK
RESET
CLK
READY
RES
PCLK
RESET
RES
CLK
6.8µF
BALE
DEN
IO/M
EXTA
APPA
to expansion RAM sockets
A0
IO/M
RD
WR
A7
Dev Sel
R/W
ALE
DT/R
MN/MX
NMI
INTR
HOLD
TEST
A17/S4
A16/S3
A15
A14
A13
A12
A11
A10
A9
A8
AD7
AD6
AD5
AD4
AD3
AD2
AD1
AD0
D7
D6
D5
D4
D3
D2
D1
D0
A3
A2
A1
A0
CS
WE
OE
IO4
IO2
IO5
IO8
IO3
IO6
IO1
IO7
CE
O3
O1
O4
O7
O2
O5
O0
O6


Φ0
Q3
DMA In
+5
2k
DMA Out
RES
APPA
DMAE
BALE
CLK
RDY
DT/R
DEN
DMA
TRFE
R/W
A15
A14
A13
A12
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
D7
D6
D5
D4
D3
D2
D1
D0
Int In
Int Out
+5V
GND
.1µF
12 times
EN
OC




©1981
ALF
5-1-7B
cir side


Toshiba
TMM2016P
-11C0

INDEX

8Ø88 conventions: 4-1
8Ø88 direct calls: 4-6

Busy flag: 4-1, 4-3

Constants: 4-5

DOS 3.2: 1-2, 3-4
DOS 3.3: 1-2, 2-1 to 2-2, 3-4

Expansion port: 5-1 to 5-3

FTL.B: 2-2 to 2-3
FTL, auto slot: 2-2
FTL introduction: 2-1
FTL, is it in memory?: 2-2
FTL, losing: 2-1 to 2-2
FTL, setting up yourself: 2-2 to 2-3
FTL, stays in memory: 2-1
FTL, using: 2-1

I/O allocation: 5-1
I/O interface status: 5-1
I/O ports: 4-1, 4-3, 5-1
iAPX 88 book: 4-6
Installing the card: 1-1
Integer BASIC: 2-1 to 2-2

Language card: 1-2, 2-1

MET.B: 3-7 to 3-8
Memory allocation: 5-1
Memory reference, 4-byte: 4-2
MET and FTL: 3-1, 3-3
MET buffer variable: 3-7
MET, changing parameters: 3-6 to 3-7
MET data, reading directly: 3-7
MET introduction: 3-1
MET, picking a resolution: 3-1 to 3-2
MET, restarting: 3-8
MET, setting up: 3-1
MET, setting up yourself: 3-7 to 3-8
MET timing: 3-2 to 3-3
MET READ: 3-3 to 3-7
MET READ buffer variable: 3-7
MET READ, changing parameters: 3-6 to 3-7
MET READ commands: 3-4
MET READ lines variable: 3-7
MET READ plot mode: 3-6
MET READ view mode: 3-4 to 3-6

Parameter ports: 4-1
Photo: 5-8
Preparing the disk: 1-2
PROM commands: 4-1 to 4-5
PROM commands, "available": 4-6
PROM commands, integer math: 4-4
PROM commands, floating point: 4-4 to 4-5
PROM commands, miscellaneous: 4-2
PROM routines introduction: 4-1
PROM size selection: 5-3
Protected DOS 3.3 disks: 2-1 to 2-2

Radio-TV interference: 1-2
RAM card: 1-2, 2-1
RAM expansion: 5-3
Random: 4-2, 4-3
Repair illustration: 5-7

Schematic: 5-4 to 5-6
Slot number: 2-1, 2-2, 3-2, 3-3, 3-4, 3-7, 4-1, 4-3

Tips: 1-1
Typing "FP": 1-2

USING TWO OR MORE CARDS

When two or more AD8Ø88 Processor Cards are used in one Apple, the DMA daisy chain must run through each card. This allows the cards to access the Apple's memory one at a time in a controlled fashion. The daisy chain is "broken" by an empty slot or a card which doesn't connect DMA IN to DMA OUT (the edge contacts on each side of the card, second set from the back).

If all cards in your Apple have the DMA lines properly connected, just make sure there are no empty slots between Processor Cards (that is, all empty slots are at the far left or right). If you suspect the DMA lines may not be properly connected on some of your peripheral cards, just be sure to plug all Processor Cards in adjacent slots.

DMA TECHNICAL DETAILS

The Processor Card is not necessarily compatible with other products that use DMA because Apple has not selected a standard DMA procedure. The rules for compatibility with the system used in the Processor Card are as follows:

1. A card must not change its DMA OUT line while Q3 is low and phase Ø is high. (A flip-flop clocked by a negative transition of Q3 can be used to generate DMA OUT.)
2. A card must pull the DMA line low only when phase Ø goes low and only if its DMA IN line is high at that time.
3. A card must stop pulling the DMA line low at the next negative transition of phase Ø (unless its DMA IN line is still high).

The AD8Ø88 will hold DMA low for only one cycle of phase Ø. However, it does not return DMA OUT to high until Q3 goes low just before the negative transition of phase Ø in the cycle following the DMA cycle. This allows the Apple to run for one cycle following any DMA access (or group of accesses from multiple cards), thus preventing loss of register contents.