Interfacing the 68000 to an AIM 65

Today there are several 16-bit CPUs on the market; for a number of reasons the most famous are only three, the 8086, the Z8000 and the Motorola MC68000. We have been hearing a lot about what they can and cannot do; as a matter of fact, boards employing one of such CPUs did not have too much luck.

This recalls to memory what happened years ago: everybody realized that the Z80 was more powerful than the 8080, but an upgrading kit for 8080 machines simply did not sell. The reason was not the lack of Z80 software, because the Z80 can run 8080 software. It was that the user was not willing to spend money for something maybe better but not "useful."

Nowadays we have to ask ourselves how useful is a 16-bit CPU; we feel it is not for the computer consumer, the one who buys a computer just to play games or little more. There is a range of applications requiring more computational power than what is currently available on 8-bit CPUs. When we say "computational power," we do not only mean an extended instruction set or the capability of running standard programs ten times faster: all those aspects have to be considered as a whole, along with all the hardware facilities. Many concepts developed for the old mainframes are becoming prominent in microsystem design: can you imagine a multi-task, multiprocessor system without the test-and-set instruction and the user-supervisor environment? Even the hobbyist with little background can successfully experiment with concurrent programming on a fairly small system, provided he has the right CPU to start with.

We think that a good 16-bit CPU has enough power to handle fairly sophisticated, concurrent programming. Among the available devices, the MC68000 is a good choice. Besides its nice hardware structure, the following points are to be taken into account:

1) Its instructions are powerful but limited in number. We feel that a large instruction set does not necessarily make a processor more powerful, it may instead confuse the programmer.
2) The instruction set is microprogrammed; it may be improved without changing the overall structure.
3) Its pipelined structure is optimized for speed.
4) It is asynchronous; this feature allows for easy interfacing with all kinds of devices and peripherals.

The 68000 has some drawbacks, too:

1) It does not provide a dynamic memory refresh like the Z8000; this is a really handy feature, even if software refresh is an alternative.
2) Its most interesting supporting chips are not scheduled to be available in the near future.
3) Like most of its competitors, there is not too much software available if we exclude that delivered from Motorola for their boards and development systems (which, in any case, are quite expensive).

Luca Fusina and Claudio Granuzzo

Luca Fusina, Via Mocenigo 8, Verona 37100, Italy.

Nowadays it seems that what really interests the computer consumer is software. He does not care too much about the underlying hardware, he is most concerned about the programs he can run on his machine. At most he considers the interface capabilities of his computer, but usually he even does not know the internal hardware structure. In our opinion there are still some people interested in system design: people who like to experiment with new devices. The tool for this kind of work is called a development system. After the hobbyist eliminates the ones that are too expensive, what is left is an evaluation system which basically is a single board equipped with a CPU and a handful of switches and LEDs. Experimenting with such boards is time consuming and not rewarding at all in any case. There must be a better solution.

The Idea

Almost all of the computer enthusiasts around already own a microcomputer. Why not use our own system to control a 68000? In this article we will describe this hardware and software implementation on a Rockwell AIM 65. For the reader who is unfamiliar with this machine we will summarize its features. It is a single board based on the 6502 with a QWERTY keyboard, a 20-column thermal printer and an alphanumeric display, 4K byte of RAM, an 8K byte ROM monitor, a 4K byte ROM assembler, and 2 Versatile Interface Adapter chips for I/O and expansion. We had to make only a few trivial hardware changes.

We have said, indeed, that we are going to control our 68000, but which way? We want to be able to control it during each read or write cycle (from now on simply cycle), in real time, changing the mode of operation according to necessity among all the ones available to our system. Furthermore, we want to be able to manipulate during each cycle all the 68000 control signals; this way we can simulate interrupts, multi-processing and so on.

The Hardware

All we needed to implement our idea was nine eight-bit I/O ports and a couple of decoders. The 68000 became a peripheral connected to the AIM bus at a certain address. On the AIM expansion connector we found all we needed: the 8-bit bi-directional data lines, the 16-bit address lines, and two control signals: R/W and 02 which are used for synchronizing R/W operations. Figure 1 (page 15) shows the block diagram of the interface; Figure 2 (page 16) is the schematic. For all the 6502 timings, the reader can refer to the 6502 hardware manual.

From now on we assume the reader understands 68000 hardware and software details; refer to the MC68000 user's manual for a complete description of this processor.

As the 68000 is asynchronous, we can control each cycle using the signal $\overline{\text{DTACK}}$, which stands for data transfer acknowledge. When the 68000 wants to perform a read or a write, it asserts the $\overline{\text{AS}}$ signal and waits for $\overline{\text{DTACK}}$. At this point our AIM program can do all it has to, and when finished it asserts $\overline{\text{DTACK}}$ by writing to location $8XXB, signaling the 68000 to continue. IC 20 (a 74LS74 flip-flop) takes care of negating $\overline{\text{DTACK}}$ when 68000 negates $\overline{\text{AS}}$.

All 68000 signals are interfaced to the AIM through six DM81LS95 buffers and three 74LS373 latches. The 68000 interface is seen by the AIM as a 1Kbyte memory. The 6502 can address up to sixty-four 1Kbyte pages and the board can be located anywhere inside them, provided there are no conflicts with the AIM 65 requirements. This is accomplished with a DM8131 six-bit comparator (IC 10). To select the right page the user must supply the appropriate logic levels to the comparator inputs. In our implementation the board is located at $8000. Inside this address space there are sixteen meaningful locations, from address $8XX0 to address $8XXF. Only twelve are, however, actually used and only the first nine are used to interface the 68000, by means of selecting one of the I/O ports. Selection 10 (signal $\overline{\text{Y9}}$ in Figure 2, address $8XX9) is used to control the $\overline{\text{HALT}}$ line. Number eleven is not connected. Number twelve asserts $\overline{\text{DTACK}}$ (signal $\overline{\text{Y11}}$ in Figure 2, address $8XXB). Refer to Figures number 3, 4, 5 (page 17) for a detailed explanation about how the ports are arranged. Note that with this hardware it is not possible to use the M command of the AIM monitor inside the board space.

There are two 68000 signals not connected to any port: $\overline{\text{VMA}}$ (valid memory address) and E (enable). Motorola implemented these signals to maintain compatibility with existing 6800 peripherals. In our opinion it is better to use them as test points. The user can easily add one additional port to the interface to monitor them under program control.

The network of open collector inverting gates connected to the 555 timer and the $8XX9 selection is used to interface correctly the 68000 $\overline{\text{HALT}}$ and $\overline{\text{RESET}}$ signals, which are bidirectional. Two LEDs can be used to monitor their logic levels. The board performs an automatic power-on reset; a manual reset is also provided, as well as software $\overline{\text{HALT}}$ and $\overline{\text{RESET}}$ control.

The Software

Before entering into program details, it may be a good idea to lay down its objectives:

Run-time control of the dynamic evolution of each 68000 instruction step using the AIM keyboard. It is possible to execute a program stopping the 68000 every cycle, or to execute any number of instructions consecutively; a dynamic switch between these two modes is possible, too.

Run-time control of the 68000 input control signal (IC 19, #74LS374 latch). The AIM 65 keyboard can be used to supply a byte to be stored in that latch. We called this feature "dummy memory."

Output on display/printer of each 68000 cycle including addresses, data and control signals coming out from the processor.

Dynamic allocation of memory. 68000 memory is segmented and each segment base address can be located anywhere inside the AIM free RAM. If the 68000 is doing a read, it is possible to enter data from keyboard; if it is doing a write, it is possible to do a data display. This way no effective memory operations are done.

Use of all AIM 65 peripherals and utilities under 68000 program control. We defined that a 68000 segment cannot be greater than 64Kbytes (in automatic mode). If a write is performed inside the first 256 bytes of the last 1Kbyte page available to a segment (address $00FCXX) it is assumed that a 6502 subroutine call is made. The 68000 lower data byte is loaded in the 6502 accumulator. The 68000 upper data byte is used to index a table of pointers to 6502 subroutines. On return, the 6502 accumulator is copied into two locations, one inside the 68000 user data segment and the other inside its supervisor data segment.

The 68000 is operated cycle by cycle by the program in Listing 1, p. 36. As the program currently running evolves, whatever happens is shown on the display/printer or whatever is connected to the AIM 65. Various instruction combinations can be tried, and the operation of the 68000 becomes clear.

The five objectives that we have so far discussed are implemented in a program 519 bytes long. The user has to select the operating mode by storing an appropriate value in a control byte, CONTRL. Each bit controls a mode as explained below.

If bit 0 is set, after printing the 68000 address the AIM 65 program requests a byte from the keyboard. It will be stored in the latch connected to the 68000 input control signals.

If bit 1 is set, manual mode is selected, otherwise automatic mode is assumed. Manual mode corresponds to the dummy memory R/W mode previously discussed; in the auto mode, R/W is performed from memory.

If bit 2 is set, before issuing the $\overline{\text{DTACK}}$ signal the user is requested to validate all operations performed during the current cycle by entering a carriage return. This is also called step mode. Any other character will repeat the current cycle. No such validation check is made if bit 2 is cleared.

If bit 4 is set, fast mode is selected. When this mode is selected 68000 cycle status output is suppressed. This allows for fast 68000 program running, as a great deal of the AIM housekeeping time is spent reporting things on display/printer.

The previous modes can be mixed together. Fast mode overrides all the others. When it is set, the other modes are used to select the mode that will be entered upon error. Dynamic mode changing can be accomplished by executing a 68000 instruction that stores a new value in the control variable. Thus, a 68000 program can put itself in Step mode.

Memory segmented is implemented using two tables, FCTAB and MAXADD. They specify the AIM 65 base addresses of each 68000 segment and their extension. These latter values must be supplied, and they must be consistent with the former ones. Of course, each 68000 segment starts from location $0000000.

Final Thoughts

We have so far discussed how to build from scratch a small development system for Motorola's 16-bit processor, the MC68000. The underlying concepts are quite general, and it should not be difficult to implement our idea using the reader's own computer instead of our AIM 65. The ones familiar with this machine will have already noted that the accompanying program was not listed with the 20-column thermal printer available on the AIM board, and that the assembler was modified.

In fact, besides using the AIM 65 as an experimental computer, we use it as our "big" system. Anyone who is interested and wants more information about how we did it may write to the authors. Let us now report the impressions about the MC68000 that we gained using the board we have here presented. We appreciated most the power and simplicity of its instruction set. Some things shocked us, though. For example, you can try to execute a CLR to memory and see what happens. Before clearing the desired locations, the processor reads them. This, of course, wastes time and has no usefulness. Any-way, summing up all the againsts and fors, it proved to be a superior processor. After executing just a few programs, the user accustomed to 8-bit microprocessors will no longer be satisfied with them.

The MC68000 architecture is a bit different from standard 8-bit machines. The reader who will use our board might then have some problems. Let us give some hints that may prove useful.

The 68000 has a pipelined structure. This means that it fetches one or more words before actually executing the instruction. These words, which may be subsequent instructions, are printed and displayed. Such a thing may lead one to think that the processor is not executing properly; on the contrary, it is doing its job. Again, during stacking operations, words may not be stacked following the address ordering; the overall stacking procedure is still correct.

The MC68000 is a 16-bit machine, therefore it addresses by words. Instructions must start on even boundaries. If the user specifies the initial PC to be at an odd address, the 68000 will enter an address error exception. If the supervisor stack point also starts at an odd address, well, you will be in trouble. There are, of course, many other things that should be said, but it may be more interesting to explore the 68000 world yourself.

Concluding, we would like to point out that everything we have said so far is not restricted to a particular machine. Some readers will like to experiment with a different CPU, say the Z8000, and we think they will not have too many problems adapting our ideas to their needs. After a few weeks of experimenting with our board, the need for more sophisticated software may arise. It should not be too difficult to write a cross assembler using AIM BASIC. This would eliminate the need for hand compiling all the 68000 instructions. Again, it is possible to slightly modify the hardware to let the 68000 have an independent life, without passing through the AIM for executing its instructions.

Italy is not that far away; anyone who wants to write us to exchange opinions about computers is encouraged to do so.

DDJ

(Figures 2-6 on pages 16 and 17)
(Listing begins on page 36)

68000 Control Board Block Schematics
by Luca Fusina and Claudio Granuzzo

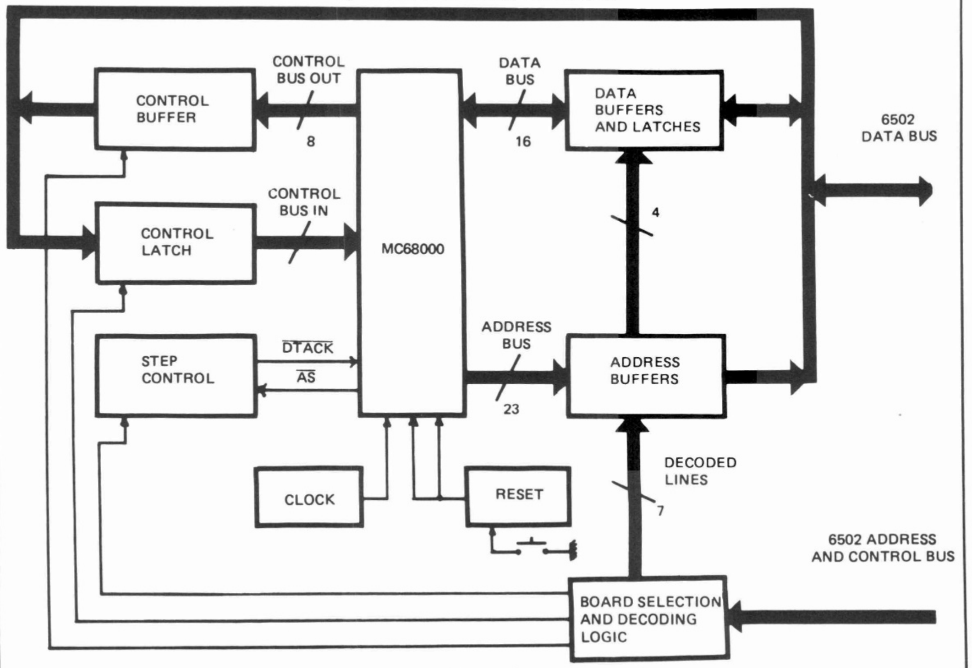
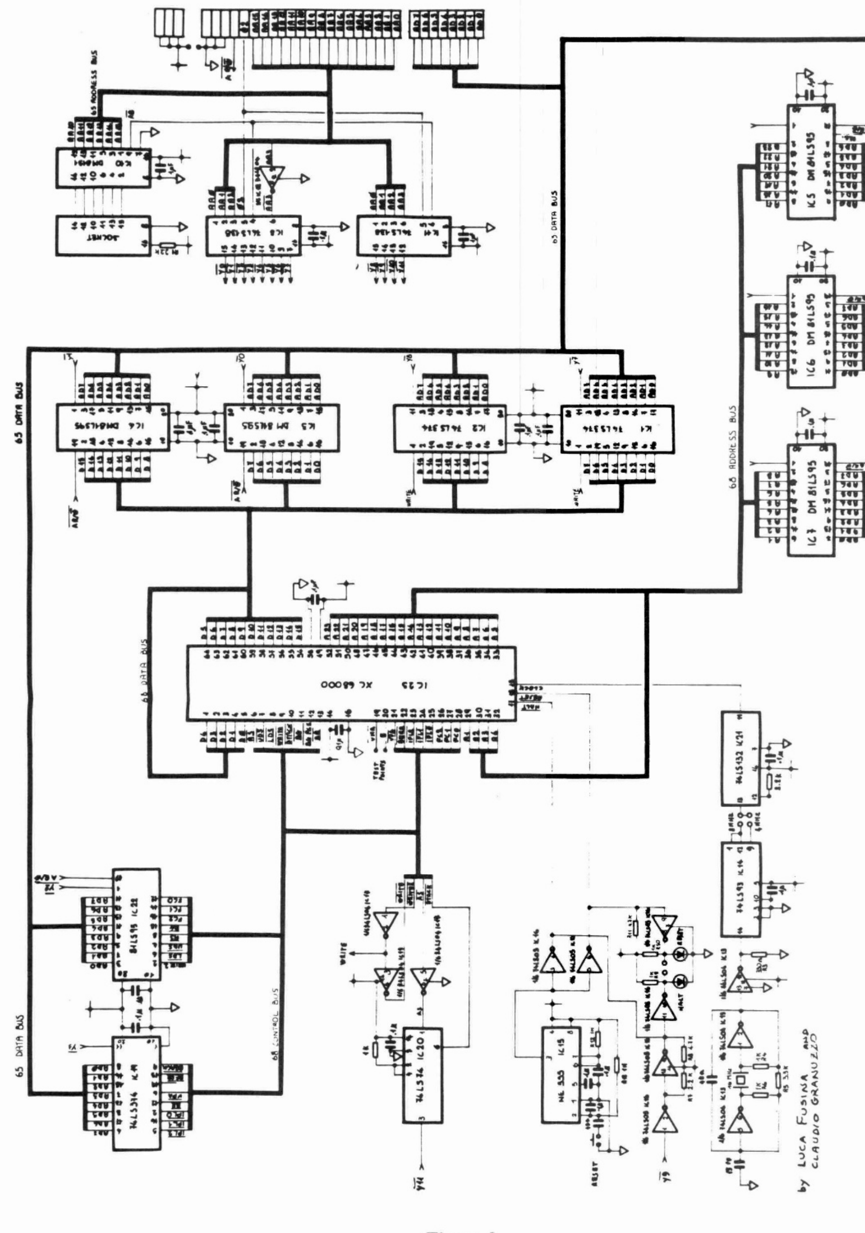


Figure 1



Schematic of MC68000/AIM 65 interface.

Figure 2

68000 On-Board Selections

Address (Hex)	Selection
8XX0	68000 lower 8 data bit (WRITE)
8XX1	68000 upper 8 data bit (WRITE)
8XX2	68000 output control signals (refer to Figure 4)
8XX3	68000 input control signals (refer to Figure 5)
8XX4	68000 lower 8 address bit
8XX5	68000 middle 8 address bit
8XX6	68000 upper 7 address bit
8XX7	68000 lower 8 data bit (READ)
8XX8	68000 upper 8 data bit (READ)
8XX9	$\overline{\text{HALT}}$
8XXA	N.C. (not connected)
8XXB	$\overline{\text{DTACK}}$

Figure 3

68000 Output Control Signals (IC22)

Bit No. on AIM Data Bus	Signal	
AD0	$\overline{\text{WRITE}}$	$\overline{\text{write}}$
AD1	$\overline{\text{LDS}}$	lower data strobe
AD2	$\overline{\text{UDS}}$	upper data strobe
AD3	$\overline{\text{AS}}$	address select
AD4	$\overline{\text{BG}}$	bus grant
AD5	FC2	function code 2
AD6	FC1	function code 1
AD7	FC0	function code 0

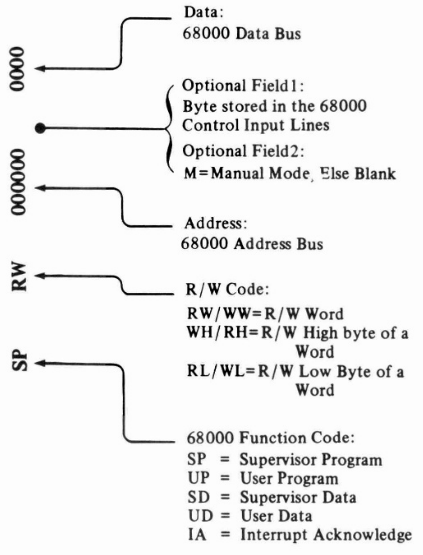
Figure 4

68000 Input Control Signals (IC19)

Bit No. on AIM Data Bus	Signal	
AD0	$\overline{\text{BGACK}}$	bus grant ack
AD1	$\overline{\text{BERR}}$	bus error
AD2	N.C. (not connected)	
AD3	$\overline{\text{VPA}}$	valid peripheral address
AD4	$\overline{\text{BR}}$	bus request
AD5	IPL0	interrupt priority level 0
AD6	IPL1	interrupt priority level 1
AD7	IPL2	interrupt priority level 2

Figure 5

Meaning of the fields in the AIM listing of the 68000 program:



Data:
68000 Data Bus

Optional Field1:
Byte stored in the 68000 Control Input Lines

Optional Field2:
M=Manual Mode, Else Blank

Address:
68000 Address Bus

R/W Code:
RW/WW=R/W Word
WH/RH=R/W High byte of a Word
RL/WL=R/W Low Byte of a Word

68000 Function Code:
SP = Supervisor Program
UP = User Program
SD = Supervisor Data
UD = User Data
IA = Interrupt Acknowledge

Figure 6

(Listing begins on page 36)

68000/AIM 65

(Listing, text begins on page 12)
(See Figure 6, page 17 for meaning of fields in Listing)

```
M)=801 00 00 00 00                           CONTROL MODE TO AUTO AND
(/) 0801 04                                  STEP
*)=200                                       RUN
(G)/
SP RW 000000 0000              SSP FETCH
SP RW 000002 00FE
SP RW 000004 0000              PC FETCH
SP RW 000006 000A
SP RW 000008 46FC    MOVE.W #0,SR    SWITCH TO USER PROGRAM
SP RW 00000A 0000
SP RW 00000C 4E71    NOP   FETCHED BUT NOT EXECUTED (FBNE)
UP RW 00000C 33FC
UP RW 00000E 0100    MOVE.W #$100,$FC00.L   LINK AIM SUBROUTINE READBYTE
UP RW 000010 0000
UP RW 000012 FC00
UP RW 000014 13F9
UD WW 00FC00 05      EXEC READBYTE   THE BYTE SHOWN IN THE DATA FIELD WAS
UP RW 000016 0000                    ENTERED WITH THE AIM KEYBOARD)
UP RW 000018 0000    MOVE.B 0000.L 0001.L
UP RW 00001A 0000
UD RH 000000 0504                    READ LOC. 0000
UP RW 00001C 0001
UD WL 000000 05                      WRITE LOC. 0001 (MODE CONTROL LOC.)
UP RW 00001E 7F4E71  NOP   NOW THE MODE IS CHANGED: IT IS REQUESTED A BYTE TO BE
UP RW 000020 7F4E71  NOP   STORED ON THE 68000 CONTROL INPUT LINES (LEV. 4 INT.)
SD WW 0000FC FF001E            STACK PC LOW (CNTRL BYTE=$FF, INT. CLEARED)
1A RW FBFFFC FFM 0064            IA           AS FUNCTION CODE=IA, WE ARE
SD WW 0000F8 FF0000            STACK STATUS  IN MANUAL MODE: VECT.#=64
SD WW 0000FA                                  ESCAPE FROM THE PROGRAM TO
(M)=801 05 00 00 00                           CHANGE MODE: NOW WITHOUT THE
(/) 0801 04                                    CONTROL BYTE
*)=200
(G)/
SD WW 0000FA 0000              STACK PC HIGH
SD RW 000190 M 0000            FETCH INT. VECT. (WE ARE IN MAN. MODE AS WE
SD RW 000192 M 4000                           ARE OUT OF SUP. DATA SEGMENT
SP RW 004000 M 4E73  RTE       MANUAL MODE AS WE ARE OUT THE SUP. PROG. SEG
SP RW 004002 M 4E71  NOP       FBNE, MANUAL MODE
SD RW 0000F8 0000              UNSTACK PARAMETERS
SD RW 0000FA 0000                 "
SD RW 0000FC 001E                 "
UP RW 00001E 4E71    NOP       BACK TO USER PROGRAM
UP RW 000020 4E71    NOP
UP RW 000022 4E71    NOP
UP RW 000024 4E71    NOP
UP RW 000026 4E71    NOP

M)=801 04 00 00 00                            CONTROL MODE TO MANUAL AND
(/) 0801 06                                    STEP
*)=200                                         RUN
(G)/
SP RW 000000 M 0000            SSP FETCH
SP RW 000002 M 1000
SP RW 000004 M 0000            PC FETCH
SP RW 000006 M 4000
SP RW 004000 M 46FC  MOVE.W #0,SR    SWITCH TO USER PROGRAM
SP RW 004002 M 0000
SP RW 004004 M 4E71  NOP             FETCHED BUT NOT EXECUTED (FBNE)
UP RW 004004 M 42B8  CLR.L $7000.W   CLEAR LOC. $7000-$7003
UP RW 004006 M 7000
UP RW 004008 M 4E71  NOP
UD RW 007000 M FFFF                  READS THE LONG WORD (UNUSEFUL)
UD RW 007002 M FFFF
UP RW 00400A M 4E71  NOP
UD WW 007002 M 0000                  CLEAR LONG WORD
UD WW 007000 M 0000
UP RW 00400C M 4E71  NOP
UP RW 00400E M 303C  MOVE.W #$2000,D0            SET DATA REG. 0
UP RW 004010 M 2000
UP RW 004012 M C0FC  MULU.W #4,D0                16-BIT MULTIPLICATION
UP RW 004014 M 0004
UP RW 004016 M 33C0  MOVE.W D0,$8000.L           SHOW D0 CONTENTS
UP RW 004018 M 0000
UP RW 00401A M 8000
UP RW 00401C M 4E71  NOP
UD WW 008000 M 8000                  THE RESULT OF MULU.W IS $8000
UP RW 00401E M 4E71  NOP
UP RW 004020 M 80FC  DIVU.W D0,#0    ZERO DIVIDE: WHAT HAPPENS NOW?
UP RW 004022 M 0000
UP RW 004024 M 4E71  NOP             FBNE
SD WW 000FFE M 4024            STACK PC: ZERO DIVIDE EXCEPTION
SD WW 000FFA M 0004
SD WW 000FFC M 0000            STACK STATUS
SD RW 000014 M 0000            READS ZERO DIVIDE VECTOR
SD RW 000016 M 9000
SP RW 009000 M 4E73  RTE             RETURN FROM EXCEPTION
SP RW 009002 M 4E71  NOP             FBNE
SD RW 000FFA M 0004            UNSTACK PARAMETERS
SD RW 000FFC M 0000
SD RW 000FFE M 4024
UP RW 004024 M 4E71  NOP             BACK TO USER PROGRAM
UP RW 004026 M 33C0  MOVE.W D0,$6000.L           SHOW D0 CONTENTS AFTER THE
UP RW 004028 M 0000                              ZERO DIVIDE
UP RW 00402A M 6000
UP RW 00402C M 4E71  NOP
UD WW 006000 M 8000                              IT DID NOT CHANGE
UP RW 00402E M 4E71  NOP
```

```
PAGE 01
PASS 2
0000                  ;**************************************************
0000                  ;*  MC68000 INTERFACING PROGRAM                   *
0000                  ;*                                                *
0000                  ;*                                                *
0000                  ;*  WRITTEN BY LUCA FUSINA AND CLAUDIO GRANUZZO   *
0000                  ;**************************************************


0000                  ;IF MORE 6502 USER SUBROUTINES ARE ADDED, THEIR EXACT
0000                  ;NUMBER MUST BE SPECIFIED
0000                  MAXROU=2

0000                  ;BOARD BASE ADDRESS
0000                  BRDADD=$8000

0000                  ;AIM 65 MONITOR SUBROUTINES EQUATES
0000                  CRLOW=$FA13          ;OUTPUT CR,LF
0000                  BLANK=$E83E          ;OUTPUT A BLANK
0000                  NUMA=$EA46           ;OUTPUT A BYTE AS TWO HEX CHAR.
0000                  OUTPUT=$E97A         ;OUTPUT A CHAR.
0000                  RBYTE=$E3FD          ;READ TWO HEX CHAR. FROM KEYBOARD
0000                  ;                     AND PACK THEM IN ONE BYTE
0000                  READ=$E93C           ;READ ONE CHAR.

0000                  ;USER MAY SPECIFY THESE VAR. ADDR.; IF SEGMENT ADDR.
0000                  ;ARE CHANGED, THESE ADDR. MUST BE ACCORDINGLY DEFINED
0000                          *=$800
0800                  SAVEAU                    ;REFER TO THE CAISUB ROUTINE
0800                          *=$500
0500                  SAVEAS                    ;REFER TO THE CALSUB ROUTINE


0500                  ;POINTER TO START ADDR. OF 6502 SUBR. ADDR. TABLE
0500                          *=$FA
00FA                  LOCJMP

00FA                  ;THE FOLLOWINGS ARE USED AS POINTER FOR
00FA                  ;INDIRECT 6502 ADDRESSING
00FA                          *=*+1
00FB                  F0                        ;HIGH BYTE OF A POINTER  USED
00FB                  ;                           FOR 68000 TO MEMORY OPERATIONS
00FB                          *=*+1
00FC                  F1                        ;LOW BYTE OF POINTER

00FC                  ;THE FOLL. ARE USED TO SAVE THE 68000 ADDR.
00FC                          *=*+1
00FD                  SAVEL
00FD                          *=*+1
00FE                  SAVEM
00FE                          *=*+1
00FF                  SAVEH
Page 02

00FF                  ;MODE CONTROL VARIABLE. THE ADDR. OF THIS VAR.
00FF                  ;MUST BE INSIDE USER DATA SEGMENT, SO A 68000
00FF                  ;PROGRAM CAN CHANGE IT
00FF                          *=$801
0801                  CONTRL


0801                  ;THE FOLL. ARE THE  BOARD ADDR.; REFER TO THE TEXT
0801                  ;HARDWARE DESCRIPTION FOR THE THEIR MEANING
0801                          *=BRDADD
8000                  DIL
8000                          *=*+1
8001                  DIH
8001                          *=*+1
8002                  CIN
8002                          *=*+1
8003                  COUT
8003                          *=*+1
8004                  ADDL
8004                          *=*+1
8005                  ADDM
8005                          *=*+1
8006                  ADDH
8006                          *=*+1
8007                  DOL
8007                          *=*+1
8008                  DOH
8008                          *=*+1
8009                  HALT
8009                          *=*+1
800A                  RES
800A                          *=*+1
800B                  DTACK



800B                          *=$200
0200  78              MMM68   SEI
0201  A9FF                    LDA #$FF
0203  8D0380                  STA COUT          ;CLEAR 68000 INPUT SIGN.
0206  2013EA          LOOP    JSR CRLOW
0209  AD0280          WAITAS  LDA CIN
020C  2908                    AND #8
020E  D0F9                    BNE WAITAS        ;WAIT FOR ADDR. STROBE
0210  AD0108                  LDA CONTRL
0213  2908                    AND #8            ;SPEEDY MODE CONTROL BIT
0215  F003                    BEQ OUTFC
0217  4C2E03                  JMP FAST

021A                  ;OUTPUT 68000 FUNCTION CODE
021A                  ;U1(UNDEFINED),U2(UNDEFINED),UP(USER
021A                  ;PROGRAM),SP(SUPERVISOR PROGRAM),
021A                  ;UD(USER DATA),SD(SUPERVISOR DATA),
021A                  ;U3(UNDEFINED),IA(INTERRUPT ACK.)
021A  AD0280          OUTFC   LDA CIN

Page 03

021D  29E0                    AND #%11100000
021F  4A                      LSR A
0220  4A                      LSR A
0221  4A                      LSR A
0222  4A                      LSR A
0223  AA                      TAX
0224  BDE803                  LDA TABFC,X
0227  207AE9                  JSR OUTPUT
022A  BDE903                  LDA TABFC+1,X
022D  207AE9                  JSR OUTPUT
0230  203EE8                  JSR BLANK


0233                  ;OUTPUT 68000 R/W CODE
0233                  ;WW(WRITE WORD),RW(READ WORD),WH(WRITE HIGH
0233                  ;BYTE),WL(WRITE LOW BYTE),RL(READ LOW BYTE),
0233                  ;RH(READ HIGH BYTE),UN(UNDEFINED)
0233  AD0280          OUTRWL  LDA CIN
0236  2907                    AND #7
0238  0A                      ASL A
0239  AA                      TAX
023A  BDD803                  LDA TABRWL,X
023D  207AE9                  JSR OUTPUT
0240  BDD903                  LDA TABRWL+1,X
0243  207AE9                  JSR OUTPUT
0246  203EE8                  JSR BLANK


0249                  ;OUTPUT 68000 ADDRESSES
0249                  ;WE MULTIPLY BY TWO TO CONVERT FROM
0249                  ;68000 WORD ADDR. TO BYTE ADDR.
0249  AD0480          OUTADD  LDA ADDL
024C  0A                      ASL A
024D  85FD                    STA SAVEL
024F  AD0580                  LDA ADDM
0252  2A                      ROL A
0253  85FE                    STA SAVEM
0255  AD0680                  LDA ADDH
0258  2A                      ROL A
```

```
0259  85FF              STA SAVEH
025B  2046EA            JSR NUMA
025E  A5FE              LDA SAVEM
0260  2046EA            JSR NUMA
0263  A5FD              LDA SAVEL
0265  2046EA            JSR NUMA
0268  203EE8            JSR BLANK

026B  AD0108  INCMD     LDA CONTRL
026E  2901              AND #1          ;WITH THIS BIT SET A BYTE IS
0270          ;                         ENTERED FROM KEYBOARD
0270          ;                         TO CONTROL THE 68000 INPUT SGN.
0270  F006              BEQ NOCMD
0272  20FDE3  CMD       JSR RBYTE
0275  8D0380            STA COUT
0278  AD0108  NOCMD     LDA CONTRL
027B  2902              AND #2          ;MANUAL OR AUTO MODE CNTR. BIT
027D  F02D              BEQ AUTO

Page 04

027F          ;MANUAL MODE: READ FROM KEYBOARD,
027F          ;MEMORY WRITE TO DISPLAY/PRINTER ONLY
027F  A94D    MANUAL    LDA #$4D
0281  207AE9            JSR OUTPUT      ;OUTPUT 'M'
0284  203EEB            JSR BLANK
0287  AD0280            LDA CIN
028A  2901              AND #1          ;TEST 68000 R/W
028C  D00F              BNE READM

028E          ;IF WRITE OUTPUT 68000 DATA BUS
028E  AD0180  WRITEM    LDA DIH
0291  2046EA            JSR NUMA
0294  AD0080            LDA DIL
0297  2046EA            JSR NUMA
029A  4C1703            JMP TSER

029D          ;IF READ, GET TWO BYTES FROM KEYBOARD AND PUT THEM
029D          ;ON THE 68000 INPUT DATA LATCHES
029D  20FDF3  READM     JSR RBYTE
02A0  8D0880            STA DOH
02A3  20FDE3            JSR RBYTE
02A6  8D0780            STA DOL
02A9  4C1703            JMP TSER

02AC          ;AUTO MODE: R/W FROM MEMORY
02AC  AD0280  AUTO      LDA CIN

02AF          ;MEMORY SEGMENTATION SECTION: IF OVERFLOW
02AF          ;ERROR OR INTA STATE SWITCH TO MANUAL MODE
02AF  29E0    FCADD     AND #%11100000
02B1  4A                LSR A
02B2  4A                LSR A
02B3  4A                LSR A
02B4  4A                LSR A
02B5  4A                LSR A
02B6  AA                TAX
02B7  E007              CPX #7
02B9  F0C4              BEQ MANUAL      ;68000 HAS ACK. AN INTERRUPT
02BB  A5FF              LDA SAVEH
02BD  D0C0              BNE MANUAL      ;SEGMENT GREATER THAN 64K.
02BF  A5FE              LDA SAVEM
02C1  C9FC              CMP #$FC        ;6502 SUBROUTINE CALL IF 68000
02C3          ;                         ADDR. IS EQUAL TO $00FC00
02C3  D003              BNE LABO
02C5  4CA203            JMP CALSUB
02C8  DD0004  LABO      CMP MAXADD,X
02CB  B0B2              BCS MANUAL      ;ERR. IF SEGMENT IS
02CD          ;                          OUT OF ITS RANGE
02CD  18                CLC
02CE  7DF803            ADC FCTAB,X     ;ADD SEG. OFFSET TO EFFECTIVE
02D1          ;                         68000 ADDR., SO TO SELECT THE
02D1          ;                         RIGHT MEMORY LOCATIONS IN THE
02D1          ;                         AIM MEMORY SPACE
02D1  85FC              STA F1
02D3  A5FD              LDA SAVEL

Page 05

02D5  85FB              STA FO
02D7  AD0280            LDA CIN
02DA  2901              AND #1          ;TEST 68000 R/W
02DC  D025              BNE READA

02DE          ;AUTO WRITE IN MEMORY
02DE  AD0280  WRITA     LDA CIN
02E1  2902              AND #2          ;TEST 68000 UPPER DATA STROBE
02E3  D00A              BNE NOUDS

02E5          ;WRITE UPPER DATA BYTE
02E5  A000    WRIUDS    LDY #00
02E7  AD0180            LDA DIH
02EA  91FB              STA (FO),Y
02EC  2046EA            JSR NUMA
02EF  AD0280  NOUDS     LDA CIN
02F2  2904              AND #4          ;TEST 68000 LOWER DATA STROBE
02F4  D00A              BNE NOLDS

02F6          ;WRITE 68000 LOWER DATA BYTE
02F6  A001    WRILDS    LDY #1
02F8  AD0080            LDA DIL
02FB  91FB              STA (FO),Y
02FD  2046EA            JSR NUMA
0300  4C1703  NOLDS     JMP TSER

0303          ;READ A WORD FROM MEMORY
0303  A000    READA     LDY #0
0305  B1FB              LDA (FO),Y
0307  8D0880            STA DOH
030A  2046EA            JSR NUMA
030D  A001              LDY #1
030F  B1FB              LDA (FO),Y
0311  8D0780            STA DOL
0314  2046EA            JSR NUMA

0317          ;IF STEP MODE IS SELECTED, WAIT FOR A CHAR. FROM
0317          ;KEYBOARD: IF IT IS NOT A CR DO NOT ISSUE
0317          ;DTACK, AND REPEAT THE CICLE
0317  AD0108  TSER      LDA CONTRL
031A  2904              AND #4          ;STEP MODE CONTROL BIT
031C  F00A              BEQ DTA
031E  203CE9            JSR READ
0321  C90D              CMP #$0D
0323  F003              BEQ DTA
0325  4C0602            JMP LOOP

0328          ;ISSUE DTACK
0328  AD0B80  DTA       LDA DTACK
032B  4C0602            JMP LOOP

032E          ;FAST AUTO MODE: NO OUTPUT IS PERFORMED

032E          ;SAVE 68000 ADDR.
032E  AD0480  FAST      LDA ADDL        ;MULTIPLY BY TWO
0331  0A                ASL A

Page 06

0332  85FB              STA FO
0334  AD0580            LDA ADDM
0337  2A                ROL A
0338  85FE              STA SAVEM
033A  AD0680            LDA ADDH
033D  2A                ROL A
033E  F003              BEQ FAST1       ;SKIP IF SEG. IS LESS THAN 64K
0340  4C1A02  OUTF1     JMP OUTFC       ;ERR.: OUTPUT CYCLE STAT.

0343          ;MEMORY SEGMENTATION SECTION: ON ERROR
0343          ;STANDARD CYCLE HANDLING IS ENTERED, AND
0343          ;FAST MODE IS RE-ENTERED NEXT CYCLE
0343  AD0280  FAST1     LDA CIN
0346  29E0              AND #$E0        ;68000 FUNCTION CODE
0348  4A                LSR A
0349  4A                LSR A
034A  4A                LSR A
034B  4A                LSR A
034C  4A                LSR A
034D  AA                TAX
034E  E007              CPX #7
0350  F0EE              BEQ OUTF1       ;SKIP IF INTA
0352  A5FE              LDA SAVEM
0354  C9FC              CMP #$FC        ;6502 SUBROUTINES CALL
0356  F04A              BEQ CALSUB
0358  A5FE              LDA SAVEM
035A  DD0004            CMP MAXADD,X
035D  B0E1              BCS OUTF1       ;OUT OF SEG. RANGE ERR.
035F  18                CLC
0360  7DF803            ADC FCTAB,X     ;ADD SEG. OFFSET
0363  85FC              STA F1
0365  AD0280            LDA CIN
0368  2901              AND #1          ;TEST 68000 R/W
036A  D022              BNE READF

036C          ;FAST MEMORY WRITE
036C  AD0280  WRITF     LDA CIN
036F  2902              AND #2          ;68000 UDS
0371  D007              BNE NOUDSF
0373  A000    WRFUDS    LDY #0
0375  AD0180            LDA DIH
0378  91FB              STA (FO),Y
037A  AD0280  NOUDSF    LDA CIN
037D  2904              AND #4          ;68000 LDS
037F  D007              BNE NOLDSF
0381  A001    WRFLDS    LDY #1
0383  AD0080            LDA DIL
0386  91FB              STA (FO),Y
0388  AD0B80  NOLDSF    LDA DTACK
038B  4C0902            JMP WAITAS

038E          ;FAST MEMORY READ
038E  A000    READF     LDY #0
0390  B1FB              LDA (FO),Y
0392  8D0880            STA DOH
0395  A001              LDY #1
0397  B1FB              LDA (FO),Y
0399  8D0780            STA DOL

Page 07

039C  AD0B80            LDA DTACK
039F  4C0902            JMP WAITAS

03A2          ;6502 SUBROUTINES HANDLER.  68000 UPPER
03A2          ;DATA BYTE IS THE INDEX TO THE SUBROUTINES
03A2          ;ADDRESS TABLE: 68000 LOWER
03A2          ;DATA BYTE IS LOADED IN THE 6502 ACC.
03A2          ;ON RETURN THE 6502 ACC. IS COPIED IN
03A2          ;TWO LOCATIONS SPECIFIED BY THE USER:
03A2          ;SAVEAU IS LOCATED IN THE USER DATA SEGMENT;
03A2          ;SAVEAS IS LOCATED IN THE SUPERVISOR
03A2          ;DATA SEGMENT
03A2  AD0280  CALSUB    LDA CIN
03A5  2901              AND #1          ;TEST 68000 R/W BECAUSE
03A7  F003              BEQ CALSU1      ;ONLY WRITE IS ALLOWED
03A9  4C7F02  ERR       JMP MANUAL
03AC  AD0180  CALSU1    LDA DIH
03AF  C902              CMP #MAXROU     ;SUBR. ADDR. TABLE OVERFLOW
03B1  B0F6              BCS ERR
03B3  0A                ASL A
03B4  AA                TAX
03B5  BDD403            LDA TABSUB,X
03B8  85FA              STA LOCJMP
03BA  BDD503            LDA TABSUB+1,X
03BD  85FB              STA LOCJMP+1
03BF  AD0080            LDA DIL
03C2  20D103            JSR JMPSUB
03C5  8D0008            STA SAVEAU
03C8  8D0005            STA SAVEAS
03CB  AD0B80            LDA DTACK
03CE  4C0602            JMP LOOP

03D1  6CFA00  JMPSUB    JMP (LOCJMP)

03D4          ;6502 SUBR. TABLE
03D4  46EA    TABSUB    .WORD NUMA
03D6  FDE3              .WORD RBYTE

03D8          ;68000 R/W CODES
03D8  5757    TABRWL    .BYTE 'WWRWWLRLWHRHUNUN'

03E8          ;68000 FUNCTION CODES
03E8  5531    TABFC     .BYTE 'U1U2UPSPUDSDU3IA'

03F8          ;SEGMENT START ADDR., EXPRESSED IN PAGES OF 256
03F8          ;BYTES EACH. USER MAY CHANGE THESE VALUES; IF
03F8          ;HE DOES SO HE HAS TO CHANGE ALL THE RELATED
03F8          ;VARIABLE ADDRESSES (SEE TOP OF PROGRAM).
03F8          ;HE MAY HAVE TO CHANGE VALUES IN THE MAXADD
```

(Continued on next page)

68000/AIM 65

(Listing continued, text begins on page 12)

```
03F8               ;TABLE,TOO
03FB   00          FCTAB   .BYTE 0         ;UNDEFINED
03F9   00                  .BYTE 0         ;UNDEFINED
03FA   0C                  .BYTE $0C       ;USER PROGRAM
03FB   06                  .BYTE 6         ;SUPERVISOR PROGRAM
03FC   08                  .BYTE 8         ;USER DATA

Page 08

03FD   05                  .BYTE 5         ;SUPERVISOR DATA
03FE   00                  .BYTE 0         ;UNDEFINED
03FF   00                  .BYTE 0         ;(INTA)


0400               ;NUMBER OF PAGES AVAILABLE TO EACH SEGMENT.
0400               ;USER MAY CHANGE THESE VALUES; HOWEVER THEY
0400               ;MUST BE CONSISTENT WITH THE SEGMENTS START
0400               ;ADDRESSES JUST DEFINED
0400   00          MAXADD  .BYTE 0         ;UNDEFINED
0401   00                  .BYTE 0         ;UNDEFINED
0402   04                  .BYTE 4         ;USER PROGRAM (1K)
0403   02                  .BYTE 2         ;SUP. PROGRAM (512 BYTES)
0404   04                  .BYTE 4         ;USER DATA (1K)
0405   01                  .BYTE 1         ;SUP. DATA (256 BYTES)
0406   00                  .BYTE 0         ;UNDEFINED
0407   00                  .BYTE 0         ;(INTA)


0408               ;END OF PROGRAM

0408                       .END

ERRORS- 0000



MAXROU  A: 0002     BRDADD  A: 8000     CRLOW   A: EA13     BLANK   A: E83E
NUMA    A: EA46     OUTPUT  A: E97A     RBYTE   A: E3FD     READ    A: E93C
SAVEAU  A: 0800     SAVEAS  A: 0500     LOCJMP  A: 00FA     F0      A: 00FB
F1      A: 00FC     SAVEL   A: 00FD     SAVEH   A: 00FE     SAVEH   A: 00FF
CONTRL  A: 0801     DIL     A: 8000     DIH     A: 8001     CIN     A: 8002
COUT    A: 8003     ADDL    A: 8004     ADDM    A: 8005     'ADDH   A: 8006
DOL     A: 8007     DOH     A: 8008     HALT    A: 8009     RES     A: 800A
DTACK   A: 8008     MMM68   A: 0200     LOOP    A: 0206     WAITAS  A: 0209
OUTFC   A: 021A     OUTRWL  A: 0233     OUTADD  A: 0249     INCMD   A: 0268
CMD     A: 0272     NOCMD   A: 0278     MANUAL  A: 027F     WRITEM  A: 028E
READM   A: 029D     AUTO    A: 02AC     FCADD   A: 02AF     LABO    A: 02C8
WRITA   A: 02DE     WRIUDS  A: 02E5     NOUDS   A: 02EF     WRILDS  A: 02F6
NOLDS   A: 0300     READA   A: 0303     TSER    A: 0317     DTA     A: 0328
FAST    A: 032E     OUTF1   A: 0340     FAST1   A: 0343     WRITF   A: 036C
WRFUDS  A: 0373     NOUDSF  A: 037A     WRFLDS  A: 0381     NOLDSF  A: 0388
READF    : 038E     CALSUB  A: 03A2     ERR     A: 03A9     CALSU1  A: 03AC
JMPSU      03D1     TABSUB  A: 03D4     TABRWL  A: 03D8     TABFC   A: 03E8
FCTAB   ... 03F8    MAXADD  A: 0400
```

End Listing