8232
FLOATING POINT PROCESSING UNIT

- **Compatible with Proposed IEEE Format and Existing Intel Floating Point Standard**
- **Single (32-Bit) and Double (64-Bit) Precision Capability**
- **Add, Subtract, Multiply and Divide Functions**
- **Stack Oriented Operand Storage**
- **General Purpose 8-Bit Data Bus Interface**
- **Standard 24-Pin Package**
- **12V and 5V Power Supplies**
- **Compatible with MCS-80™, MCS-85™ and MCS-86™ Microprocessor Families**
- **Error Interrupt**
- **Direct Memory Access or Programmed I/O Data Transfers**
- **End of Execution Signal**
- **Advanced N-Channel Silicon Gate HMOS Technology**

The Intel® 8232 is a high performance floating-point processor unit (FPU). It provides single precision (32-bit) and double precision (64-bit) add, subtract, multiply and divide operations. The 8232's floating point arithmetic is a subset of the proposed IEEE standard. It can be easily interfaced to enhance the computational capabilities of the host microprocessor.

The operand, result, status and command information transfers take place over an 8-bit bidirectional data bus. Operands are pushed onto an internal stack by the host processor and a command is issued to perform an operation on the data stack. The results of the operation are available to the host processor from the stack.

Information transfers between the 8232 and the host processor can be handled by using programmed I/O or direct memory access techniques. After completing an operation, the 8232 activates an "end of execution" signal that can be used to interrupt the host processor.

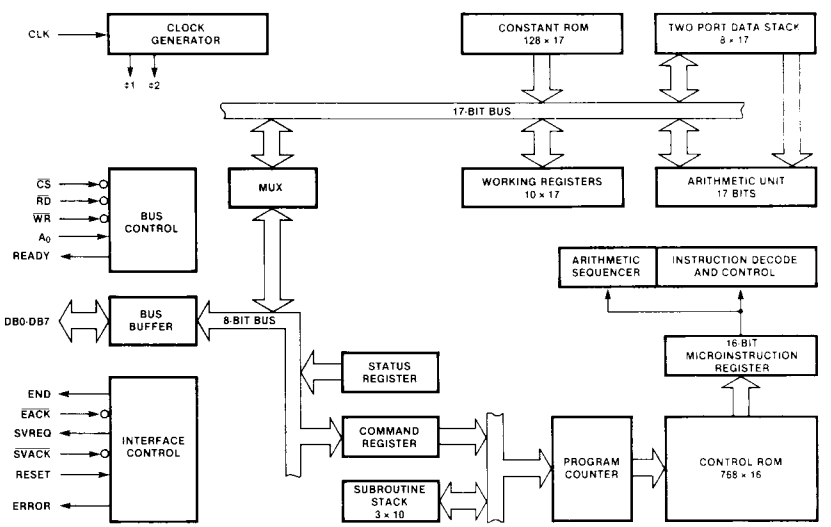


Figure 1. Block Diagram

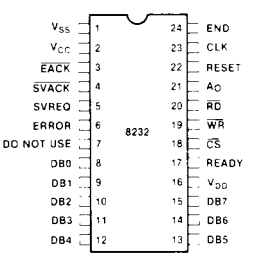


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Description
V_{CC}	2		**POWER SUPPLY:** +5V power supply
V_{DD}	16		**POWER SUPPLY:** +12V power supply
V_{SS}	1		**GROUND**
CLK	23	I	**CLOCK:** An external timing source connected to the CLK input provides the necessary clocking.
RESET	22	I	**RESET:** A HIGH level on this input causes initialization. Reset terminates any operation in progress, and clears the status register to zero. The internal stack pointer is initialized and the contents of the stack may be affected. After a reset the END output, the ERROR output and the SVREQ output will be LOW. For proper initialization, RESET must be HIGH for at least five CLK periods following stable power supply voltages and stable clock.
$\overline{CS}$	18	I	**CHIP SELECT:** input must be LOW to accomplish any read or write operation to the 8232. To perform a write operation, appropriate data is presented on DB0 through DB7 lines, appropriate logic level on the A_0 input and the $\overline{CS}$ input is made LOW. Whenever $\overline{WR}$ and $\overline{RD}$ inputs are both HIGH and $\overline{CS}$ is LOW, READY goes LOW. However, actual writing into the 8232 cannot start until $\overline{WR}$ is made LOW. After initiating the write operation by the HIGH to LOW transition on the $\overline{WR}$ input, the READY output will go HIGH, indicating the write operation has been acknowledged. The $\overline{WR}$ input can go HIGH after READY goes HIGH. The data lines, the A_0 input and the $\overline{CS}$ input can change when appropriate hold time requirements are satisfied. See write timing diagram for details. To perform a read operation an appropriate logic level is established on the A_0 input and $\overline{CS}$ is made LOW. The READY output goes LOW because $\overline{WR}$ and $\overline{RD}$ inputs are HIGH. The read operation does not start until the $\overline{RD}$ input goes LOW. READY will go HIGH indicating that read operation is complete and the required information is available on the DB0 through DB7 lines. This information will remain on the data lines as long as $\overline{RD}$ is LOW. The $\overline{RD}$ input can return HIGH anytime after READY goes HIGH. The $\overline{CS}$ input and A_0 input can change anytime after $\overline{RD}$ returns HIGH. See read timing diagram for details. If the $\overline{CS}$ is tied LOW permanently, READY will remain LOW until the next 8232 read or write access.
A_0	21	I	**ADDRESS:** The A_0 input together with the $\overline{RD}$ and $\overline{WR}$ inputs determines the type of transfer to be performed on the data bus as follows:

A_0	$\overline{RD}$	$\overline{WR}$	Function
0	1	0	Enter data byte into stack
0	0	1	Read data byte from stack
1	1	0	Enter command
1	0	1	Read status

Symbol	Pin No.	Type	Name and Description
$\overline{RD}$	20	I	**READ:** A LOW level on this input is used to read information from an internal location and gate that information onto the data bus. The $\overline{CS}$ input must be LOW to accomplish the read operation. The A_0 input determines what internal location is to be read. See A_0, $\overline{CS}$ input descriptions and read timing diagram for details. If the END output was HIGH, performing any read operation will make the END output go LOW after the HIGH to LOW transition of the $\overline{RD}$ input (assuming $\overline{CS}$ is LOW). If the ERROR output was HIGH, performing a status register read operation will make the ERROR output LOW. This will happen after the HIGH to LOW transition of the $\overline{RD}$ input (assuming $\overline{CS}$ is LOW).
$\overline{WR}$	19	I	**WRITE:** A LOW level on this input is used to transfer information from the data bus into an internal location. The $\overline{CS}$ must be LOW to accomplish the write operation. A_0 determines which internal location is to be written. See A_0, $\overline{CS}$ input descriptions and write timing diagram for details. If the END output was HIGH, performing any write operation will make the END output go LOW after the LOW to HIGH transition of the $\overline{WR}$ input (assuming $\overline{CS}$ is LOW).
$\overline{EACK}$	3	I	**END ACKNOWLEDGE:** When LOW, makes the END output go LOW. As mentioned earlier, HIGH on the END output signals completion of a command execution. The END signal is derived from an internal flip-flop which is clocked at the completion of a command. This flip-flop is clocked to the reset state when $\overline{EACK}$ is LOW. Consequently, if $\overline{EACK}$ is tied LOW, the END output will be a pulse that is approximately one CLK period wide.
$\overline{SVACK}$	4	I	**SERVICE ACKNOWLEDGE:** A LOW level on this input clears SVREQ. If the $\overline{SVACK}$ input is permanently tied LOW, it will conflict with the internal setting of the SVREQ output. Thus, the SVREQ indication cannot be relied upon if the $\overline{SVACK}$ is tied LOW.
END	24	O	**END OF EXECUTION:** A HIGH on this output indicates that execution of the current command is complete. This output will be cleared LOW by activating the $\overline{EACK}$ input LOW or performing any read or write operation or device initialization using RESET. If $\overline{EACK}$ is tied LOW, the END output will be a pulse (see $\overline{EACK}$ description). Reading the status register while a command execution is in progress is allowed. However, any read or write operation clears the flip-flop that generates the END output. Thus, such continuous reading could conflict with internal logic setting of the END flip-flop at the end of command execution.

Table 1. Pin Description (cont'd)

Symbol	Pin No.	Type	Name and Description
SVREQ	5	O	**SERVICE REQUEST:** A HIGH on this output indicates completion of a command. In this sense this output is the same as the END output. However, the SVREQ output will go HIGH at the completion of a command only when the Service Request Enable bit was set to 1. The SVREQ can be cleared (i.e., go LOW) by activating the $\overline{SVACK}$ input LOW or initializing the device using the RESET. Also, the SVREQ will be automatically cleared after completion of any command that has the service request bit as 0.
ERROR	6	O	**ERROR:** Output goes HIGH to indicate that the current command execution resulted in an error condition. The error conditions are: attempt to divide by zero, exponent overflow and exponent underflow. The ERROR output is cleared LOW on a status register read operation or upon RESET. The ERROR output is derived from the error bits in the status register. These error bits will be updated internally at an appropriate time during a command execution. Thus, ERROR output going HIGH may not coincide with the completion of a command. Reading of the status register can be performed while a command execution is in progress. However, it should be noted that reading the status register clears the ERROR output. Thus, reading the status register while a command execution is in progress may result in an internal conflict with the ERROR output.
READY	17	O	**READY:** Output is a handshake signal used while performing read or write transactions with the 8232. If the $\overline{WR}$ and $\overline{RD}$ inputs are both HIGH, the READY output goes LOW with the $\overline{CS}$ input in anticipation of a transaction. If $\overline{WR}$ goes LOW to initiate a write transaction with proper signals established on the DB0-DB7, A_0 inputs, the READY will return HIGH indicating that the write operation has been accomplished. The $\overline{WR}$ can be made HIGH after this event. On the other hand, if a read operation is desired, the $\overline{RD}$ input is made LOW after activating $\overline{CS}$ LOW and establishing proper A_0 input. (The READY will go LOW in response to $\overline{CS}$ going LOW.) The READY will return HIGH, indicating completion of read. The $\overline{RD}$ can return HIGH after this event. It should be noted that a read or write operation can be initiated without any regard to whether a command execution is in progress or not. Proper device operation is assured by obeying the READY output indication as described.
DB0-DB7	8-15	I/O	**DATA BUS:** Bidirectional lines are used to transfer command, status and operand information between the device and the host processor. DB0 is the least significant and DB7 is the most significant bit position. HIGH on a data bus line corresponds to 1 and LOW corresponds to 0. When pushing operands on the stack using the data bus, the least significant byte must be pushed first and the most significant byte last. When popping the stack to read the result of an operation, the most significant byte will be available on the data bus first and the least significant byte will be the last. Moreover, for pushing operands and popping results, the number of transactions must be equal to the proper number of bytes appropriate for the chosen format. Otherwise, the internal byte pointer will not be aligned properly. The single precision format requires 4 bytes and double precision format requires 8 bytes.

FUNCTIONAL DESCRIPTION

Major functional units of the 8232 are shown in the block diagram. The 8232 employs a microprogram controlled stack oriented architecture with 17-bit wide data paths.

The Arithmetic Unit receives one of its operands from the Operand Stack. This stack is an eight word by 17-bit two port memory with last in-first out (LIFO) attributes. The second operand to the Arithmetic Unit is supplied by the internal 17-bit bus. In addition to supplying the second operand, this bidirectional bus also carries the results from the output of the Arithmetic Unit when required. Writing into the Operand Stack takes place from this internal 17-bit bus when required. Also connected to this bus are the Constant ROM and Working Registers. The ROM provides the required constants to perform the mathematical operations while the Working Registers provide storage for the intermediate values during command execution.

Communication between the external world and the 8232 takes place on eight bidirectional input/output lines, DB0 through DB7 (Data Bus). These signals are gated to the internal 8-bit bus through appropriate interface and buffer circuitry. Multiplexing facilities exist for bidirectional communication between the internal eight

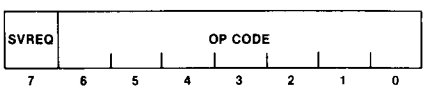
and 17-bit buses. The Status Register and Command Register are also located on the 8-bit bus.

The 8232 operations are controlled by the microprogram contained in the Control ROM. The Program Counter supplies the microprogram addresses and can be partially loaded from the Command Register. Associated with the Program Counter is the Subroutine Stack where return addresses are held during subroutine calls in the microprogram. The Microinstruction Register holds the current microinstruction being executed. The register facilitates pipelined microprogram execution. The Instruction Decode logic generates various internal control signals needed for the 8232 operation.

The Interface Control logic receives several external inputs and provides handshake related outputs to facilitate interfacing the 8232 to microprocessors.

Command Format

The operation of the 8232 is controlled from the host processor by issuing instructions called commands. The command format is shown below.



The command consists of 8 bits; the least significant 7 bits specify the operation to be performed as detailed in Table 1. The most significant bit is the Service Request Enable bit. This bit must be a 1 if SVREQ is to go HIGH at the end of executing a command.

The commands fall into three categories: single precision arithmetic, double precision arithmetic and data manipulation. There are four arithmetic operations that can be performed with single precision (32-bit) or double precision (64-bit) floating-point numbers: add, subtract, multiply and divide. These operations require two operands. The 8232 assumes that these operands are located in the internal stack as Top of Stack (TOS) and Next on Stack (NOS). The result will always be returned to the previous NOS which becomes the new TOS. Results from an operation are of the same precision and format as the operands. The results will be rounded to preserve the accuracy. The actual data formats and rounding procedures are described in a later section. In addition to the arithmetic operations, the 8232 implements eight data manipulating operations. These include changing the sign of a double or single precision operand located in TOS, exchanging single precision operands located at TOS and NOS, as well as pushing and popping single or double precision operands. See also the sections on status register and operand formats.

The execution times of the commands are all data dependent. Table 3 shows one example of each command execution time.

Operand Entry

The 8232 commands operate on the operands located at the TOS and NOS. Results are returned to the stack at NOS and then popped to TOS. The operands required for the 8232 are one of two formats — single precision floating-point (4 bytes) or double precision floating-point (8 bytes). The result of an operation has the same format as the operands. In other words, operations using single precision quantities always result in a single precision result, while operations involving double precision quantities will result in double precision result.

Operands are always entered into the stack least significant byte first and most significant byte last. The following procedure must be followed to enter operands into the stack:

1. The lower significant operand byte is established on the DB0–DB7 lines.
2. A LOW is established on the A_0 input to specify that data is to be entered into the stack.
3. The $\overline{CS}$ input is made LOW. Whenever the $\overline{WR}$ and $\overline{RD}$ inputs are HIGH, the READY output will follow the $\overline{CS}$ input. Thus, READY output will become LOW.
4. After appropriate set up time (see timing diagrams), the $\overline{WR}$ input is made LOW.
5. Sometime after this event, READY will return HIGH to indicate that the write operation has been acknowledged.
6. Any time after the READY output goes HIGH, the $\overline{WR}$ input can be made HIGH. The DB0–DB7, A_0 and $\overline{CS}$ inputs can change after appropriate hold time requirements are satisfied (see timing diagrams).

The above procedure must be repeated until all bytes of the operand are pushed into the stack. It should be noted that for single precision operands 4 bytes should be pushed and 8 bytes must be pushed for double precision. Not pushing all the bytes of a quantity will result in byte pointer misalignment.

The 8232 stack can accommodate four single precision quantities or two double precision quantities. Pushing more quantities than the capacity of the stack will result in loss of data which is usual with any LIFO stack.

The stack can be visualized as shown below:

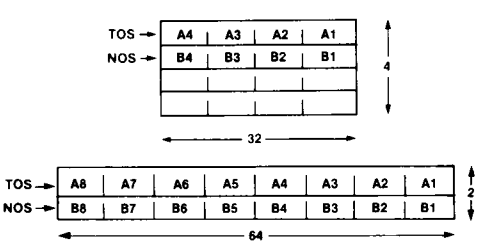


Table 2. 8232 Command Set

Single Precision Instructions

Instruction	Description	Hex[1] Code	Stack Contents[2] After Execution A B C D	Status Flags Affected[4]
SADD	Add A and B	01	R C D U	S, Z, U, V
SSUB	Subtract A from B	02	R C D U	S, Z, U, V
SMUL	Multiply A by B	03	R C D U	S, Z, U, V
SDIV	Divide B by A. If A exponent = 0, then R = B.	04	R C D U	S, Z, U, V, D
CHSS	Change sign of A[5]	05	R B C D	S, Z
PTOS	Push stack[5]	06	A* A B C	S, Z
POPS	Pop stack	07	B C D A	S, Z
XCHS	Exchange	08	B A C D	S, Z

Double Precision Instructions

Instruction	Description	Hex[1] Code	Stack Contents[3] After Execution A B	Status Flags Affected[4]
DADD	Add A and B	29	R U	S, Z, U, V
DSUB	Subtract A from B	2A	R U	S, Z, U, V
DMUL	Multiply A by B	2B	R U	S, Z, U, V
DDIV	Divide B by A. If A = 0, then R = B.	2C	R U	S, Z, U, V, D
CHSD	Change sign of A[5]	2D	R B	S, Z
PTOD	Push stack[5]	2E	A* A	S, Z
POPD	Pop stack	2F	B A	S, Z
CLR	CLR status	00	A B	S, Z, U, V, D

Notes:

1. In the hex code column, SVREQ bit is a 0.
2. The stack initially is composed of four 32-bit numbers (A, B, C, D). A is equivalent to Top Of Stack (TOS) and B is Next on Stack (NOS). Upon completion of a command the stack is composed of: the result (R); undefined (U); or the initial contents (A,B,C, or D).
3. The stack initially is composed of two 64-bit numbers (A, B). A is equivalent to Top Of Stack (TOS) and B is Next On Stack (NOS). Upon completion of a command the stack is composed of: the result (R); undefined (U); or the initial contents (A, B).
4. Any status bit(s) not affected are set to 0. Nomenclature: Sign (S); Zero (Z); Exponent Underflow (U); Exponent Overflow (V); Divide Exception (D).
5. If the exponent field of A is zero, R or A* will be zero.

Table 3. Execution Times

Command	TOS	NOS	Result	Clock Periods
SADD	3F800000	3F800000	40000000	58
SSUB	3F800000	3F800000	00000000	56
SMUL	40400000	3FC00000	40900000	198
SDIV	3F800000	40000000	3F000000	228
CHSS	3F800000	—	BF800000	10
PTOS	3F800000	—	—	16
POPS	3F800000	—	—	14
XCHS	3F800000	40000000	—	26
CHSD	3FF00000 00000000	—	BFF00000 00000000	24
PTOD	3FF00000 00000000	—	—	40
POPD	3FF00000 00000000	—	—	26
CLR	3FF00000 00000000	—	—	4
DADD	3FF00000 0A000000	80000000 00000000	3FF00000 A0000000	578
DSUB	3FF00000 A0000000	80000000 00000000	3FF00000 A0000000	578
DMUL	BFF80000 00000000	3FF80000 00000000	C0020000 00000000	1748
DDIV	BFF80000 00000000	3FF80000 00000000	BFF00000 00000000	4560

Note: TOS, NOS and result are in hexadecimal; clock period is in decimal.

Command Initiation

After properly positioning the required operands in the stack, a command may be issued. The procedure for initiating a command execution is the same as that described above for operand entry, except that the A_0 input is HIGH.

An attempt to issue a new command while the current command execution is in progress is allowed. Under these circumstances, the READY output will not go HIGH until the current command execution is completed.

Removing the Results

Result from an operation will be available at the TOS. Results can be transferred from the stack to the data bus by reading the stack.

When the stack is read for results, the most significant byte is available first and the least significant byte last.

A result is always of the same precision as the operands that produced it. Thus, when the result is taken from the stack, the total number of bytes popped out should be appropriate with the precision — single precision results are 4 bytes and double precision results are 8 bytes. The following procedure must be used for reading the result from the stack:

1. A LOW is established on the A_0 input.
2. The $\overline{CS}$ input is made LOW. When $\overline{WR}$ and $\overline{RD}$ inputs are both HIGH, the READY output follows the $\overline{CS}$ input, thus READY will be LOW.
3. After appropriate set up time (see timing diagrams), the $\overline{RD}$ input is made LOW.
4. Sometime after this, READY will return HIGH, indicating that the data is available on the DB0-DB7 lines. This data will remain on the DB0-DB7 lines as long as the $\overline{RD}$ input remains LOW.
5. Any time after READY goes HIGH, the $\overline{RD}$ input can return HIGH to complete the transaction.
6. The $\overline{CS}$ and A_0 inputs can change after appropriate hold time requirements are satisfied (see timing diagram).
7. Repeat this procedure until all bytes appropriate for the precision of the result are popped out.

Reading of the stack does not alter its data; it only adjusts the byte pointer. Note data must be removed in even byte multiples to avoid a byte pointer misalignment. If more data is popped than the capacity of the stack, the internal byte pointer will wrap around and older data will be read again, consistent with the LIFO stack.

Reading Status Register

The 8232 status register can be read without any regard to whether a command is in progress or not. The only implication that has to be considered is the effect this might have on the END and ERROR outputs discussed in the signal descriptions.

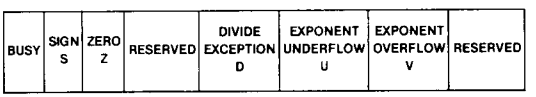
The following procedure must be followed to accomplish status register reading:

1. Establish HIGH on the A_0 input.
2. Establish LOW on the $\overline{CS}$ input. Whenever $\overline{WR}$ and $\overline{RD}$ inputs are HIGH, READY will follow the $\overline{CS}$ input. Thus, READY will go LOW.
3. After appropriate set up time (see timing diagram), $\overline{RD}$ is made LOW.

4. Sometime after the HIGH to LOW transition of $\overline{RD}$, READY will become HIGH, indicating that status register contents are available on the DB0-DB7 lines. These lines will contain this information as long as $\overline{RD}$ is LOW.
5. The $\overline{RD}$ input can be returned HIGH any time after READY goes HIGH.
6. The A_0 input and $\overline{CS}$ input can change after satisfying appropriate hold time requirements (see timing diagram).

Status Register

The 8232 contains an 8-bit status register with the following format:



All the bits are initialized to zero upon reset. Also, executing a CLR (Clear Status) command will result in all zero status register bits. A zero in bit 7 indicates that the 8232 is not busy and a new command may be initiated. As soon as a new command is issued, bit 7 becomes 1 to indicate the device is busy and remains 1 until the command execution is complete, at which time it will become 0. As soon as a new command is issued, status register bits 0-6 are cleared to zero. The status bits will be set as required during the command execution. Hence, as long as bit 7 is 1, the remainder of the status register bit indications should not be relied upon unless the ERROR occurs. The following is a detailed status bit description.

Bit 0 Reserved.

Bit 1 Exponent overflow (V). When 1, this bit indicates that the result exponent is more positive than +127 (+1023). The exponent is "wrapped" into the negative exponent range, skipping the end values.

Bit 2 Exponent Underflow (U). When 1, this bit indicates that the result exponent is more negative than −126 (−1022). The exponent is "wrapped" into the positive range by the number of underflow bits, skipping −127 (−1023) and +128 (+1024).

Bit 3 Divide Exception (D). When 1, this bit indicates that an attempt to divide by zero is made. Cleared to zero otherwise.

Bit 4 Reserved.

Bit 5 Zero (Z). When 1, this bit indicates that the result returned to TOS after a command is all zeros. Cleared to zero otherwise.

Bit 6 Sign (S). When 1, this bit indicates that the result returned to TOS is negative. Cleared to zero otherwise.

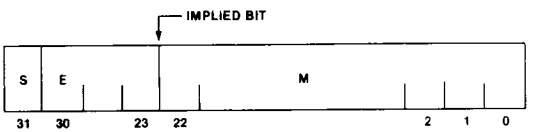
Bit 7 Busy. When 1, this bit indicates the 8232 is in the process of executing a command. It will become zero after the command execution is complete.

All other status register bits are valid when the Busy bit is zero.

Data Formats

The 8232 handles floating-point quantities in two different formats — single precision and double precision. These formats are the same as those used by Intel in other products and those proposed by the IEEE Subcommittee on floating point arithmetic.

The single precision quantities are 32 bits long, as shown below:



Bit 31:

S = Sign of the mantissa. One represents negative and 0 represents positive.

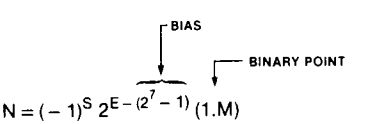
Bits 23-30:

E = These 8 bits represent a biased exponent. The bias is $2^7 - 1 = 127$.

Bits 0-22:

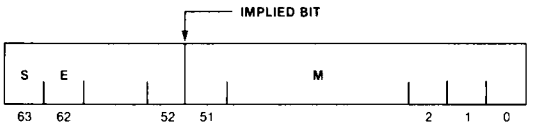
M = 23-bit mantissa. Together with the sign bit, the mantissa represents a signed fraction in sign-magnitude notation. There is an implied 1 beyond the most significant bit (bit 22) of the mantissa. In other words, the mantissa is assumed to be a 24-bit normalized quantity and the most significant bit, which will always be a 1 due to normalization, is implied. The 8232 restores this implied bit internally before performing arithmetic, normalizes the result and strips the implied bit before returning the results to the external data bus. The binary point is between the implied bit and bit 22 of the mantissa.

The quantity N represented by the above notation is



Provided E≠0 (reserved for 0) or all 1's (illegal). The approximate decimal range for this format is $\pm 8.43 \times 10^{-37}$ to $\pm 3.37 \times 10^{38}$. The format supports 7 significant decimal digits.

A double precision quantity consists of the mantissa sign bit, an 11-bit biased exponent (E), and a 52-bit mantissa (M). The bias for double precision quantities is $2^{10}-1$. The double precision format is illustrated below.



Bit 63:

S = Sign of the mantissa. One represents negative and 0 represents positive.

Bits 52-62:

E = These 11 bits represent a biased exponent. The bias is $2^{10}-1=1023$.

Bits 0-51:

M = 52-bit mantissa. Together with the sign bit the mantissa represents a signed fraction in sign-magnitude notation. There is an implied 1 beyond the most significant bit (bit 51) of the mantissa. In other words, the mantissa is assumed to be a 53-bit normalized quantity and the most significant bit, which will always be a 1 due to normalization, is implied. The 8232 restores this implied bit internally before performing arithmetic, normalizes the result and strips the implied bit before returning the result to the external data bus. The binary point is between the implied bit and bit 51 of the mantissa.

The quantity N represented by the above notation is

$$N=(-1)^{S}\,2^{E-\overbrace{(2^{10}-1)}^{\text{BIAS}}}\,\underbrace{(1.M)}_{\text{BINARY POINT}}$$

Provided E ≠ 0 (reserved for 0) or all 1s (illegal). The approximate decimal range is $\pm 4.19\times10^{-307}$ to $\pm 1.67\times10^{308}$. The format supports 16 significant decimal digits.

The following are some examples of single precision floating point representations:

Decimal	S	E	M	Binary Floating Point
0	0	0	0	0000 0000H
1	0	127	0	3F80 0000H
−1.	1	127	0	BF80 0000H
255	0	134	.9922	437F 0000H
π	0	128	.5708	4049 0FDBH

Rounding

One of the main objectives in choosing the 8232's Intel/IEEE proposed floating point arithmetic was to provide maximum accuracy with no anomalies. This means that a mathematically unsophisticated user will not be "surprised" by some of the results. It is probably possible for a sophisticated user to obtain reliable results from almost any floating point arithmetic. However, in that case there will be an additional burden on the software.

The best example of what might be called the 8232's "safety factor" is the inclusion of guard bits for rounding. The absence of guard bits leads to the problem demonstrated by the following four-bit multiplication:

$$\begin{array}{l} .1111\times2^{0} \\ .1000\times2^{1} \\ \hline .01111000\times2^{1} \end{array}$$

Since the last four bits are lost, the normalized result is:

$$.1110\times2^{0}$$

and the identify function is not valid. In the past this problem has been avoided (hopefully) by relying on excess precision.

Instead the 8232 uses a form of rounding known as "round to even." There are other types of rounding provided for in the proposed IEEE standard, but "round to even," an unbiased rounding scheme, is required. "Round to even" comes into play when a result is exactly halfway between two floating point numbers. In this case the arithmetic produces the "even" number, the one whose last mantissa bit is zero. The 8232 uses three additional bits—the Guard bit (G), the Rounding bit (R), and the "Sticky" bit (S)—to do the rounding. These are bits which hold data shifted out (right) of the accumulator. Rounding is carried out by the following rules, as shown in the following figure, after the result is normalized.

Bit			
G	R	S	Rule
0	0	0	No Round
0	0	1	Round Down
0	1	0	
0	1	1	
1	0	0	Round to Even
1	0	1	Round Up
1	1	0	
1	1	1	

APPLICATIONS INFORMATION

The diagram in Figure 3 represents the minimum configuration of an 8232 system. The CPU transfers data to and from the 8232 Floating Point Processor using the READY line. The 8232 status is checked using polling by the CPU.

In a high performance configuration (Figure 4), interrupts are used in place of polling. The interrupts are generated for an error condition and to signal the end of execution. Operand transfers are handled by the DMA controller.

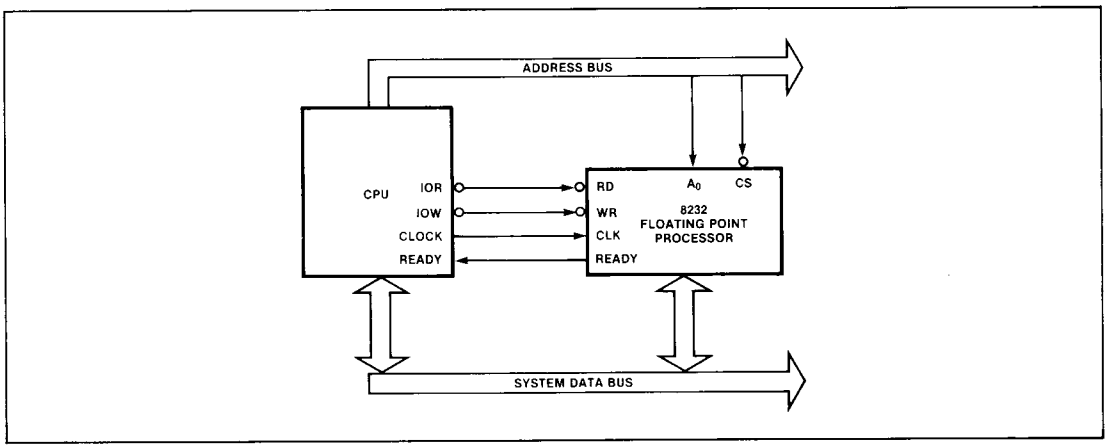


Figure 3. Minimum Configuration Example

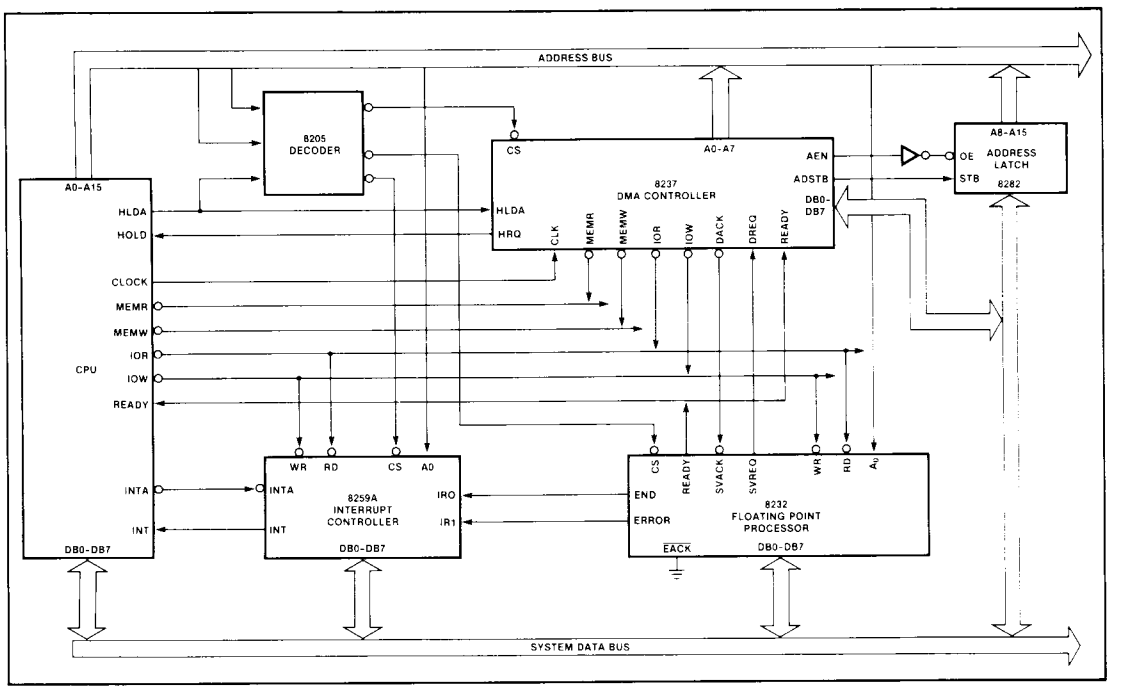


Figure 4. High Performance Configuration Example

ABSOLUTE MAXIMUM RATINGS*

Storage Temperature ..... −65°C to +150°C
Ambient Temperature Under Bias ..... 0°C to +70°C
V_{DD} with Respect to V_{SS} ..... −0.5V to +15.0V
V_{CC} with Respect to V_{SS} ..... −0.5V to +7.0V
All Signal Voltages with Respect to V_{SS} ..... −0.5V to +7.0V
Power Dissipation 2.0W

***Notice:** Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. AND OPERATING CHARACTERISTICS (Note 1) $T_A = 0°C$ to $70°C$, $V_{SS} = 0V$, $V_{CC} = +5V \pm 10\%$, $V_{DD} = +12V \pm 10\%$

Symbol	Parameter	Min.	Typ.	Max.	Units	Test Conditions
V_{OH}	Output HIGH Voltage	3.7			V	$I_{OH} = -200\mu A$
V_{OL}	Output LOW Voltage			0.4	V	$I_{OL} = 3.2mA$
V_{IH}	Input HIGH Voltage	2.0		V_{CC}	V	
V_{IL}	Input LOW Voltage	−0.5		0.8	V	
I_{IL}	Input Load Current			±10	μA	$V_{SS} \leqslant V_{IN} \leqslant V_{CC}$
I_{OZ}	Data Bus Leakage			±10	μA	$V_{SS} + 0.4 \leqslant V_{IN} \leqslant V_{CC}$
I_{CC}	V_{CC} Supply Current		50	95	mA	
I_{DD}	V_{DD} Supply Current		50	95	mA	
C_O	Output Capacitance		8		pF	$f_C = 1.0MHz$, Inputs = 0V
C_I	Input Capacitance		5		pF	
C_{IO}	I/O Capacitance		10		pF	

A.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$, $V_{SS} = 0V$, $V_{CC} = +5V \pm 10\%$, $V_{DD} = +12V \pm 10\%$

READ OPERATION

Symbol	Parameter		8232		8232-3		8232-8		Units
			Min.	Max.	Min.	Max.	Min.	Max.	
t_{AR}	A_0, $\overline{CS}$ Setup to $\overline{RD}$		0		0		0		ns
t_{RA}	A_0, $\overline{CS}$ Hold from $\overline{RD}$		0		0		0		ns
t_{ARY}	READY↓ from A_0, $\overline{CS}$↓ Delay (Note 2)			100		100		150	ns
t_{YR}	READY↑ to $\overline{RD}$↑		0		0		0		ns
t_{RRR}	READY Pulse Width (Note 3)	Data	3.5 t_{CY} + 50		3.5 t_{CY} + 50		3.5 t_{CY} + 50		ns
		Status	1.5 t_{CY} + 50		1.5 t_{CY} + 50		1.5 t_{CY} + 50		ns
t_{RDE}	Data Bus Enable from $\overline{RD}$↓		50		50		50		ns
t_{DRY}	Data Valid to READY↑		0		0		0		ns
t_{DF}	Data Float after $\overline{RD}$↑		20	100	20	150	20	200	ns

WRITE OPERATION

Symbol	Parameter	8232		8232-3		8232-8		Units
		Min.	Max.	Min.	Max.	Min.	Max.	
t_{AW}	A_0, $\overline{CS}$ Setup to $\overline{WR}$	25		25		25		ns
t_{WA}	A_0, $\overline{CS}$ Hold after $\overline{WR}$	30		30		60		ns
t_{AWY}	READY↓ from A_0, $\overline{CS}$↓ Delay (Note 2)		100		100		150	ns
t_{YW}	READY↑ to $\overline{WR}$↑	0		0		0		ns
t_{RRW}	READY Pulse Width		t_{AW} + 50		t_{AW} + 50		t_{AW} + 50	ns
t_{DW}	Data Setup to $\overline{WR}$↑	100		100		150		ns
t_{WD}	Data Hold after $\overline{WR}$↑	20		20		20		ns

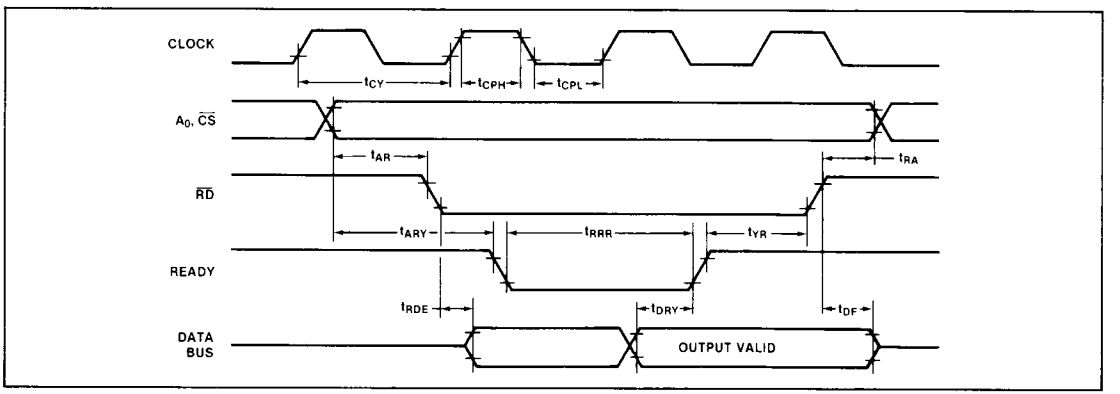
OTHER TIMINGS

Symbol	Parameter	8232		8232-3		8232-8		Units
		Min.	Max.	Min.	Max.	Min.	Max.	
t_{CY}	Clock Period	250	2500	320	3300	480	5000	ns
t_{CPH}	Clock Pulse HIGH Width	100		140		200		ns
t_{CPL}	Clock Pulse LOW Width	120		160		240		ns
t_{EE}	END Pulse Width (Note 4)	200		300		400		ns
t_{EAE}	$\overline{EACK}$↓ to END↓ Delay		150		175		200	ns
t_{AA}	$\overline{EACK}$ Pulse Width	50		75		100		ns
t_{SA}	$\overline{SVACK}$↓ to SVREQ↓ Delay		100		200		300	ns
t_{SS}	$\overline{SVACK}$ Pulse Width	50		75		100		ns

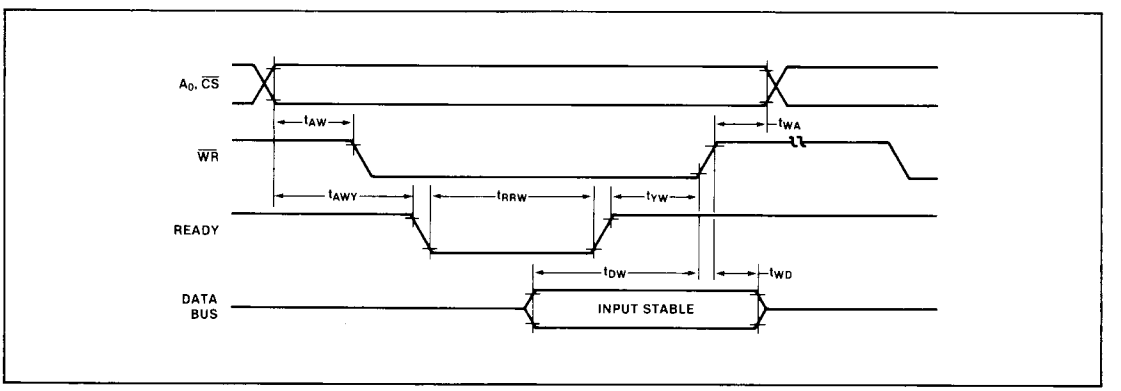
Notes:

1. Typical values are for $T_A = 25°C$, nominal supply voltages and nominal processing parameters.
2. READY is pulled low for both command and data operations.
3. Minimum values shown assume no previously entered command is being executed for the data access. If a previously entered command is being executed, READY low pulse width is the time to complete execution plus the time shown. Status may be read at any time without exceeding the time shown.
4. END high pulse width is specified for EACK tied to V_{SS}. Otherwise t_{EAE} applies.

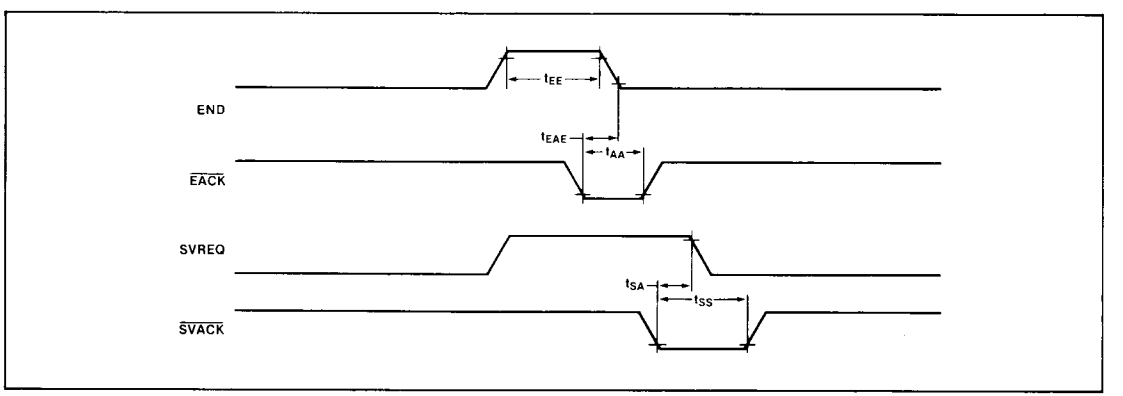
WAVEFORMS



READ OPERATION



WRITE OPERATION



INTERRUPT OPERATION