MICROPROCESSOR PROVIDING SELECTABLE ALIGNMENT CHECKING ON MEMORY REFERENCES

FIELD OF THE INVENTION

The invention relates to the field of semiconductor microprocessors.

BACKGROUND OF THE INVENTION

The present invention covers a scheme for having alignment checks controlled by "mode bits" within a microprocessor. Using this concept, the programmer is allowed to select whether a fault will be generated if a memory reference is to a misaligned address. The currently preferred embodiment of the present invention is incorporated in the architecture of the Intel 80486 microprocessor, also known as the 486™ processor. The 486™ microprocessor is an improvement over Intel's 80386 microprocessor, also referred to as the 386™ processor. (Intel, 80386, 386, 80486, and 486 are trademarks of Intel Corporation).

The 80486 microprocessor is a 32-bit, high-performance member of the x86 family of processors. It is object code compatible with the 80386 processor. As a result it can run code developed for the previous members of the family. Generally, it is a reimplementation of the 80386 architecture with the aim of providing at least a 2.5 times performance improvement as measured by the average number of clocks per instruction. The present invention represents one such feature of the 80486 microprocessor which contributes to this performance improvement.

As mentioned above, this invention adds the capability to a microprocessor of being able to select whether a fault is generated if a memory reference in a program is to a misaligned address. Conventionally, an address of a datum is considered aligned if it is a multiple of its length. A one-byte datum is always aligned. A two-byte datum is aligned if its address is a multiple of two. A four-byte datum is aligned if its address is a

multiple of four, and so on. Data, therefore, is generally aligned when the address is a multiple of its size.

The significance of aligned memory references becomes more evident when one considers the internal memory architecture of a microprocessor. In most computers the memory is organized in such a way that the data size is equal to the memory width. If you have a memory organized by 32-bits, this means that you can access a 32-bit item that has an address which is a multiple of four within one memory clock cycle. (A 32-bit word is frequently referred to as a doubleword or dword -- a dword being comprised of four separate bytes, e.g., 0, 1, 2 and 3). Both the 80386 and the 80486 CPUs permit referencing to data at arbitrary addresses. If the reference is not aligned then the microprocessor suffers a performance penalty which is manifested by additional memory cycles needed to access or reference the data.

Consider a memory reference to a 32-bit dword beginning at address 3. In past approaches, the processor would reference 2 dwords; first the dword beginning at address 0 to extract byte 3, then the dword beginning at address 4 to extract the remaining bytes 4, 5 and 6. After accessing the required bytes the processor would then somehow piece them together to reconstruct the data item or memory reference. Thus, prior approaches suffer at least two penalties: (1) the requirement of at least two additional memory cycles, and (2) a reconstructing or "piecing together" of the data item in a new storage location.

To avoid this drawback, several alternative approaches have been attempted. For instance, in the 80386 alignment checks can be performed by the insertion of an in-line code proceeding every memory reference. This code generates an address into a register and then masks out the lower bits of the register. The masking operation is implemented by taking the same register components and adding an aligning displacement. Basically, a series of three separate instructions must be executed proceeding every memory reference using this approach. Since half of the instructions in a

typical program reference memory in some way, this represents an unacceptable overhead -- degrading performance as much as 20 to 30 percent.

Several machines, the most notably the new RISC processors, require all memory references to be aligned. These machines always fault on references to misaligned data. However, the faults are not selectable, so that faults are invariably generated. This presents a problem since many non-artificial intelligence environments (especially COBOL) benefit from the ability to reference misaligned data. Another category of machines permits misaligned references, however, with the performance penalty described above. These machines include the Intel 80386, Digital's VAX (Registered Trade Mark) and the IBM 370 (Registered Trade Mark). Consequently, a new method of providing optional misaligned address faults, which does not suffer from the performance penalties described above, is what is needed in the field.

As will be seen, the present invention provides a means for alignment checking which is selectable by the user and requires no additional instructions. Moreover, the invention supports two levels of masking for the alignment checking procedure; one at the application level and the other at the operating system level. With the ability to mask the trap or the fault itself, this invention offers the user two opportunities to control alignment checking.

Accordingly, the object of the present invention is to provide a means for detecting inadvertent misalignment of data. For purposes of program

debugging, aligning data can increase performance substantially.

Another object of the present invention is to provide a means for generating alignment faults in certain artificial intelligence (A1) programs which use lower order address bits to identify types of pointers, and then use small displacements to "adjust out" these tag bits to get an aligned address. The present invention provides pointer type checking without a substantial performance penalty. Experimental measurements indicate that the present invention realizes a thirty percent performance boost on average A1 programs running on the 80386 microprocessor. other studies have shown a twenty percent increase on other machines.

SUMMARY OF THE INVENTION

A microprocessor which includes a means for detecting misaligned data reference is described. The detecting means is selectable such that when the detecting means is enabled and reference is made to a misaligned data object, a fault is produced which interrupts execution of the current program. The detecting means is part of the segmentation unit of the presently described microprocessor. In the preferred embodiment, the microprocessor also has a protection mechanism for granting access to certain data objects at various levels of privilege.

The detecting means comprises two mode bits stored within the microprocessor. The first mode bit provides masking of the fault at the least privileged level of execution (i.e., the applications level) while the second mode bit provides masking of the fault at the most privileged level (i.e., the operating system level). Thus, access to the mode, or mask, bits is provided at two different levels. Both mode bits must be set to "1" in order for the detecting means to be enabled. The use of two separate mode bits for optionally enabling alignment checking provides optimum programming flexibility.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be understood more fully from the detailed description given below and from the accompanying drawings of the preferred embodiment of the invention, which, however, should not be taken to limit the invention to the specific embodiment but are for explanation and understanding only.

Figure 1 illustrates the four privilege levels provided by the invented microprocessor. The most secure level is shown as level 0 and the least secure level is shown as level 3.

Figure 2 illustrates the EFLAGS register located within the invented microprocessor. The EFLAGS register has an extra bit (the AC bit) defined in the upper 16-bits to support faults on access to misaligned data at privilege level 3.

Figure 3 illustrates the machine status control register (CR0) which contains an alignment masking bit (AM) which controls whether the AC bit in EFLAGS can allow an alignment fault or not.

Figure 4 is a general block diagram of the microprocessor of the present invention.

Figure 5 is a diagram showing how alignment checking is logically implemented in the currently preferred embodiment of the present invention.

DETAILED DESCRIPTION

A microprocessor including a means for selecting whether a fault is to be generated whenever a misaligned memory reference is detected is described. In the following description, numerous specific details are set forth, such as specific bit lengths, register contents, logic diagrams, etc., in order to provide a thorough understanding of the present invention. It will be obvious, however, to one skilled in the art that these specific details need not be used to practice the present invention. In other instances, well-known structures and circuits, have not been shown in detail in order not to unnecessarily obscure the present invention.

Prior to discussing the detailed embodiment, it is beneficial to the understanding of the present invention to first consider several architectural features of the 80486 microprocessor.

BASIC MICROPROCESSOR ARCHITECTURE

As the role of computers in our society becomes increasingly significant, more microprocessors are implementing a protected virtual address mode (protected mode). The protected mode incorporated into the 80486 microprocessor allows multiple applications to run concurrently, but isolates them from one another so that failures in one application do not affect any other application. The central feature of a protection mechanism is the *selector*. Rather than directly accessing any part of the system, a program deals with a selector which grants access to a system object. Associated with each object is information about it, for example its location, size, restrictions on use, etc. In the 80486 microprocessor only the operating system has access to the data referenced by the selector which is called a *descriptor*.

Descriptors describe a system object in detail. Memory segments are one kind of system object. Other system objects include tables that support the protection mechanism, special segments that store the processor state, etc. By examining the selector the hardware determines which descriptor is

associated with the selector and with the object to which the descriptor points. One item that the descriptor indicates is the privilege level of the object. When a program requests access to an object with the selector either access is denied (if the request violates a rule of the protection mechanism control passes from the program to a designated routine in the operating system), access is permitted but not granted (e.g., if the object is not currently in memory, an operating system routine must swap the object into memory and return control to the program), or access is granted at the requested privilege level.

The 80486 processor supports four levels of increasing privilege, numbered 3, 2, 1, and 0 as shown in Figure 1. Privilege level 0 is the most privileged level. The privilege level of the selector in the code segment (CS) register identifies the precedence of the currently executing routine and it is called the current privilege level (CPL). For reliability reasons, only the most trustworthy code in the operating system runs at the most privileged level (CPL = 0). Applications that might fail are run at the lowest level (CPL =3). Various privilege levels are shown as a series of concentric rings in Figure 1. (The word privilege connotates rights or advantages not normally granted. In the 80486 microprocessor, procedures running at the innermost rings can access data objects in the outer rings, but outer-ring procedures cannot access inner-ring objects with greater privilege.)

As will be discussed in more detail later, the present invention allows masking of the alignment check feature at both the application level (e.g., CPL = 3) and at the operating system level (e.g., CPL = 0). This selectability of masking at two different privilege levels allows the present invention to be compatible with those machines which either always trap, or never trap, on a misaligned reference. By way of example, at the operating system level, the alignment check feature may be disabled at reset thereby rendering it compatible with the 80386 microprocessor.

In certain programming languages, such as C or Pascal, when the

data type is declared at compile time, the selector is tagged with that data type forever. In other words, the data type does not change when the program is executing. However, in some AI languages such as LISP or Prolog added flexibility is included in run/time type assignments. In these languages the data type can vary as the program executes. This allows the user to define a function that will operate on multiple types (e.g., $\sqrt{x}$, where x is either an integer, short real, long real, complex, etc.). Thus, depending on the data type at a particular point in the program, multiple functions can be performed.

In LISP or Prolog programs the pointer field indicates both the address and the type of the data. The item that identifies the data type is called the *tag*. The programmer uses the bottom two bits of a 32-bit pointer as the most significant tag bits. In cases in which the data type must be of a certain type or the operation is invalid, or in those cases in which the programmer expects the data type to be a certain type with overwhelming frequency, the user can generate a code that assumes the tag is a given value. This code cancels the lower tag bits. For example, if the tag has a value of 2, then the code would add a value of -2 to cancel out the tag bits. This offset or cancellation is also known as a *displacement*. If the displacement is correct, the memory item will be aligned and the data can be referenced without undergoing a fault or trap.

When executing AI programs using the present invention, memory references will be aligned as long as the displacement matches the tag. However, if the displacement does not match the tag the alignment checker of the present invention optionally generates a trap or fault. When this occurs, the current execution location (CS:EIP) and the contents of the flags register (EFLAGS) are saved on the stack and controls transfers to a software routine known as an interrupt handler. A specific interrupt number is associated with each fault condition. The instruction pointer saved on the stack after a fault occurs points to the instruction that caused the fault. Thus, the operating system can correct the condition and resume executing

the instruction.

OVERALL BLOCK DIAGRAM OF THE MICROPROCESSOR

Referring to Figure 4, the invented microprocessor incorporating the presently described alignment checking feature is shown in general block diagram form. The microprocessor includes a bus interface unit 10 which is coupled to a 32-bit external data bus 30 and additionally is coupled to an address bus 31 and several other control lines. Bus interface unit 10 deals with physical (hardware) addresses only, so operand addresses must first pass through segmentation unit 14 and paging unit 13. Also coupled to bus interface unit 10 are cache unit 12 and prefetcher 11. The prefetch unit continually asks the bus interface unit to fetch the contents of memory of the next instruction address. As soon as the prefetch unit receives the data, it places it in the queue and, if the queue is not full, requests another 32-bit piece of memory.

Cache unit 12 includes a data cache and controller which controls the accessing of the cache memory. Both the prefetcher 11 and cache unit 12 are coupled via a 32-bit bus to segmentation unit 14. Segmentation unit 14 translates segmented addresses into linear addresses.

Segmentation unit 14 is coupled to paging unit 13 and cache unit 12 through a 32-bit linear address bus 20, also called the "LA bus" (e.g., Linear Address). Paging unit 13 takes the linear addresses generated by segmentation unit 14 and converts them into physical addresses. If paging is disabled, the linear addresses of the segmentation unit become the physical addresses. When paging is enabled, the linear addresses of the 80486 microprocessor are divided into 4096-byte blocks called pages. Each page can be mapped into an entirely different address. For purposes of understanding the present invention, segmentation unit 14 may be assumed to be the same as that used in the commercially available Intel 80386 microprocessor. The segmentation and paging units for the Intel 80386 microprocessor are described in

U.S. Patent No. 4,972,338, entitled "Memory Management For Microprocessor", which is assigned to the assignee of the present invention.

Within the microprocessor instructions are coupled to the instruction decoder unit 15. The decoder unit operates in a manner similar to that of the prefetch unit. It takes individual bytes from the prefetch queue and determines the number of bytes needed to complete the next instruction. The decoder unit operates with a control unit 19 in which microcode instructions are stored; controller 19 providing sequences of control signals for the microprocessor. The instruction decode unit 15 is shown coupled to control unit 19 while control unit 19 is shown coupled to segmentation unit 14, data unit 18 and floating point unit 17. Data unit 18 is an arithmetic logic unit (ALU) which performs ALU functions in a similar manner to those performed by the Intel 80386 processor.

The microprocessor also includes a floating point unit 17 for performing floating point computations. The precise configuration of floating point unit 17, as well as the other units of the microprocessor, is not critical to the understanding the present invention. Signal flow between the various units of the microprocessor will be discussed only insofar as needed to comprehend the present invention.

DISCUSSION OF THE PREFERRED EMBODIMENT

The selectable alignment checker of the present invention is externally visible to the programmer as two new state (or mode) bits. One bit is located in the EFLAGS register and is called the AC (alignment check) bit. The EFLAGS register holds status information about the current instruction stream (the EIP register contains the address of the currently executing instruction) as well as a number of fields relevant to different instructions. The other bit, called the AM (alignment masking) bit, is located in the control register CR0.

Referring now to Figure 2, the details of the EFLAGS register is

shown. (Note that flag bits 1, 3, 5, 15 and 19 through 31 are "undefined".) The AC bit is located at bit position 18. The AC bit enables the generation of faults if the memory reference is to a misaligned address. For example, this can be caused by a word access to an odd address, a dword access to an address that is not 0 MOD 4, or an 8-byte reference to an address that is not 0 MOD 8. Alignment faults are generated only at level 3, that is, the AC bit setting is ignored (implicitly 0) at levels 0, 1, and 2. References to the descriptor tables (for selector loads) are implicitly level 0 references, even if the instructions "causing" the descriptor table are executed at level 3. Aligned faults are reported in the preferred embodiment through interrupt bit 17, with an error code of 0. (Bit 17 is the virtual mode (VM) bit which indicates that the currently executing instruction stream is x86 code when set to 1.)

The following table lists the alignment required for microprocessor data types:

TABLE 1

ALIGNMENT REQUIREMENTS BY DATA TYPE	
DATA TYPE	**ALIGNMENT**
WORD	2
DWORD	4
SHORT REAL	4
LONG REAL	8
TEMPREAL	8
SELECTOR	2
48-BIT SEGMENTED POINTER	4
32-BIT FLAT POINTER	4
32-BIT SEGMENTED POINTER	2
48-BIT "PSEUDO-DESCRIPTOR"	4
FSTENV/FLDENV SAVE AREA	4/2 (ON OPERAND SIZE)
FSAVE/FRSTOR SAVE AREA	4/2 (ON OPERAND SIZE)
BIT STRING	4

Because the alignment check feature must be separately enabled for each task, it has been included in the EFLAGS register since each task has its own "copy" of EFLAGS.

Referring now to Figure 3, the control register CR0, containing the aligning mask control bit (AM), is shown. Control register CR0 is one of several control registers which regulate the paging and numeric co-processor operation of the microprocessor. As shown in Figure 3, the alignment mask control bit is at bit position 18 of CR0. The AM bit controls whether the AC bit in EFLAGS can allow an alignment fault or not. AM = 0 disables the check (e.g., 80386 microprocessor compatibility), while AM = 1 enables it. The AM bit can only be accessed at privilege level 0; presumably only by the operating system. The operating system will set this bit to 1 to enable alignment faults only if the operating system includes the appropriate interrupt handler.

Use of the two bits to enable alignment checks provides a great deal of flexibility. The AM bit provides global operating system control, as it is located in register CR0 that is global to all tasks, and can only be referenced by the operating system. It will typically be set when the operating system is initialized, provided that the operating system includes the appropriate interrupt handler (e.g., bit 17 of CR0).

The AC bit provides application level control. It is in the EFLAGS register that is swapped at a task switch and is accessible by an application executing at level 3. Since EFLAGS is swapped on a task switch it is as if each task has a private copy of EFLAGS and, hence, its own copy of the alignment checker. Typically, AI programs need to have alignment checking only at level 3, the applications level. By having access to the two mask bits, the programmer has two opportunities to select whether or not an alignment check is done; one at the applications level (EFLAGS register can be cleared or set by the program), and one at the operating system level (AM bit cleared or set). It should be understood that both bits mask at all levels. However, AC is implicitly masked at all levels except level 3

while the AM bit can only be set/reset (i.e., controlled) at level 0.

The implementation of the invented alignment checker centers around the recognition of misaligned data accesses. This is performed using the lower three bits of the 32-bit linear address bus 20 (see Figure 4) and the length of the bus cycle as ordered by instruction decode unit 15 or the control unit 19. Misaligned data reference faults are generated within segmentation unit 14, which also generates linear addresses. The fault is serviced by control unit 19.

The following table defines a misaligned data access for bus cycles of different lengths in terms of the values of the lower three bits of LA bus 20. Bus cycle lengths are determined by decoding the "bus-request type" field from the instruction decode unit 15 or control unit 19.

TABLE 2

Bus Cycle Length (in bytes)	LA Bus (bits 2, 1, 0)	Comments
1	xxx	No misaligned access possible.
2	xx1	Access to odd address is misaligned.
4	xx1	Access to an address crossing 4-byte
	x1x	boundary is misaligned.
8	xx1	Access to an address crossing 8-byte
	x1x	boundary is misaligned.
	1xx	
4ND	xx1	This is a 4-byte-not-done bus-cycle.
	x1x	It implies that another 4-byte cycle is coming up
	1xx	and is therefore treated like an 8-byte cycle.

Once it is determined that a misaligned access has occurred, it is a relatively simple matter to determine the other conditions necessary to generate a fault. With reference to Figure 5, a diagram of the alignment checking logic circuitry is shown. Logic gates 40-44 and latches 41 and 45 are used to decode signals received from segmentation unit 14 to insure

that unit 14 is driving bus 20 and that the bus cycle will not be aborted. The three lowest-order bits of LA bus 20 appear on lines 51-53. These signals are combined with bus-request-type field signals from either unit 15 or unit 19 at the inputs of gates 46 and 69. Altogether, gates 46, 48, 49, and 67-72 combine to produce a set of signals which indicate the byte-length of the misaligned reference in accordance with Table 2. For example, gate 46 produces a signal on line 80 which signifies a 2-byte misaligned reference; gate 67 produces a signal on line 81 signifying a 4-byte misaligned reference; gate 71 produces a signal on line 82 which signifies an 8-byte misaligned reference; and gate 49 produces a signal on line 83 which signifies a 4ND misaligned reference. Each of the signals 80-83 are ORed by gate 47, with the result being stored in latch 50. (The symbols P1 and P2 associated with the latches of Figure 5 designate the phase in which each is clocked. For instance, latches 41 and 59 are clocked in phase 2 of the system clock, whereas latches 45, 50 and 62 are clocked in phase 1 of the next clock pulse).

The AM and AC bits appearing on lines 54 and 55 are input to AND gate 57. Both must be "1" to generate a misaligned fault. The output of gate 57 is ANDed at gate 63 with the outputs of latches 45 and 50 along with the privilege level of the program provided on line 56. The privilege level in the currently preferred embodiment must be at level 3 as determined by CPL bits in control unit 19. Gates 58-62 also provide an input to AND gate 63 to insure that the bus cycle type is one in which AC violations are permitted. (AC violations are not allowed on certain cycles. For instance, they are not permitted on branch cycles since, by definition, it is a single byte cycle.)

Gate 63's output is coupled to one input of AND gate 65. The other input to gate 65 is line 66 which provides a signal indicating that the microprocessor's pipeline is free before the fault signal is generated. Altogether, in addition to ANDing the AC and AM bits, the logic of Figure 5 is designed to insure that no access to any of the system segments is being

made and that a legitimate bus-cycle is being performed. AND gate 65 produces a signal called SINTR (segmentation unit interrupt) on line 67, which is coupled to control unit 19, whenever all of the above conditions are met.

On receiving SINTR, the control unit immediately halts all execution and takes steps to service this alignment check fault. It inserts two no operation (NOP) clocks and then forces the microcode to the entry-point for alignment check faults. From then on, the microcode takes over doing whatever is necessary to store the return address and inform the user program of the fault.

Thus, a microprocessor providing selectable alignment checking has been disclosed.

CLAIMS

1. A microprocessor comprising:

a memory for storing data;

a control unit for executing a program;

a means for referencing data stored within said memory at arbitrary addresses; and

means for detecting misaligned data references, said detecting means being selectable such that when said detecting means is enabled and said referencing means references misaligned data, said detecting means generates a fault which interrupts said program.

2. The microprocessor according to Claim 1 further comprising a protection mechanism for granting access to data at a requested privilege level which identifies the precedence of said program, said requested privilege level being one of a plurality of levels ranging from the least privileged to the most privileged level.

3. The microprocessor according to Claim 2 wherein said detecting means comprises a first mode bit and a second mode bit, said first mode bit being accessed at any privilege level and said second mode bit being accessed only at said most privileged level.

4. The microprocessor according to Claim 3 wherein said detecting means is enabled only if both of said mode bits are set.

5. The microprocessor of Claim 4 wherein said detecting means is enabled only at said least privileged level.

6. A microprocessor comprising:

a memory for storing data;



a control unit for executing a program;

means for referencing data stored within said memory at arbitrary addresses;

a protection mechanism for granting selected access to data objects according to the privilege level of said program;

a segmentation unit for translating segmented addresses into linear addresses, said segmentation unit comprising a means for detecting misaligned data references, said detecting means being optionally enabled such that when said detecting means is enabled and said referencing means references misaligned data, said detecting means generates a fault which interrupts said program.

7. The microprocessor according to Claim 6 wherein said detecting means comprises a first mode bit and a second mode bit said first mode bit controlling said fault at the least privileged level and said second mode bit controlling said fault at the most privileged level of said protection mechanism.

8. The microprocessor according to Claim 7 wherein said fault is generated only if both of said mode bits are set, thereby enabling said detecting means.

9. The microprocessor of Claim 8 wherein said fault is generated only when no access is being made to the system segments within said segmentation unit.

10. The microprocessor according to Claim 9 further comprising a paging unit coupled to said segmentation unit by a bus, said paging unit converting said linear addresses into physical addresses, and wherein certain bits of said bus identify a misaligned data access to said detecting means for bus cycles of different lengths.

11. A microprocessor permitting access to data objects at arbitrary addresses, said microprocessor including a protection mechanism for granting access to data objects at a certain privilege level, means for detecting misaligned data references, said detecting means being selectable such that when said detecting means is enabled and reference is made to a misaligned data object said detecting means generates a fault, and control means for enabling and disabling said detecting means, said control means comprising a first mode bit for masking said fault at a least privileged level and a second mode bit for masking said fault at a most privileged level.

12. A microprocessor substantially as hereinbefore described with reference to the accompanying drawings.