Description

Technical Field

This invention relates to data processors and more specifically to data processors having hierarchical memory arrangements such as cache-oriented processors or virtual memory systems.

Background Art

Many modern data processing systems utilize multiple levels of storage of data and instructions. Cache memories are small high-speed memories located intermediate the larger, slower main memory and the central processing unit (CPU). Cache memories are loaded from the main memory, and if the data required by the CPU is resident in the cache, very rapid access thereto is available. If the particular required information is not resident in the cache, it must be located in the main memory and transferred, with other likely-to-be-needed information, to the cache.

Virtual memory systems may be similar in concept, but are generally employed for other purposes. In a virtual memory system a main memory is provided, and there may or may not be a cache associated with the memory to CPU access path. Because of the finite amount of memory which can be directly addressed by a CPU in a typical virtual system, an extremely large additional memory, frequently in the form of a magnetic disk memory, may be required to accommodate a very large program or data base. Similarly as in the case of the cache memory, data is loaded from the disk into main memory such that the data is quickly accessible to the CPU. If the required data is not resident in main memory, a disk access is made to move a block of data into the main memory.

Depending upon the specific architecture of the data processing system, and specifically the length of each instruction to be executed by the data processing system, when a block of data is transferred either from the disk of the virtual machine into the main memory thereof or from the main memory to the cache of a cache-based machine, a particular instruction may extend across the boundary of the block of data transferred such that only a portion of the instruction actually becomes resident in the cache or in the main memory. For example, if the data word size of a particular processing system is sixteen bits, and various instruction types may have various instruction lengths from 16 bits to, for example, 48 bits or three words, a particular transfer of a block of data from the disk of a virtual machine into main memory may incompletely transfer a 48 bit long or a 32 bit long instruction. Thus when the CPU attempts to execute this incomplete instruction, an error condition will be detected, forcing the machine to execute exception handling routines to recover from that error.

In some central processing system organizations, entire instructions are loaded into the instruction execution unit or the instruction decoder and the invalidity of the instruction can be readily identified. In a CPU which employs a pipelined instruction accessing system, however, the incomplete transfer may not always be immediately detectable upon fetching of the instruction. In such a pipelined system, while an instruction is currently being executed by the instruction execution unit of the CPU, the next instruction to be executed is concurrently being fetched from main memory. In the case where the data path between the main memory and the instruction execution unit is not as wide as the widest possible instruction (i.e. a 16-bit data path and a 48-bit instruction) the first 16 bit word of an incomplete instruction may be fetched and placed in the top of the pipeline, subsequently be advanced in the pipeline while the second 16 bit word of the incomplete instruction is fetched, and finally execution may be begun on the first 16 bit word of the instruction at the time a fetch is attempted for the missing third 16 bit word of the instruction.

One possible solution to this problem of avoiding the beginning of execution of words of incomplete instructions is to identify, upon the fetch of the first 16 bit word, the size of the instruction being fetched. If the instruction is a multiple-word length instruction, a fetch is immediately made of all required words to insure that they are resident in the memory. In such a system, a fault or error condition is generated immediately if the instruction is incomplete. In some instances however it is desirable to defer the taking of the fault until the first word of the invalid instruction is actually ready to execute rather than at the time it is being loaded into the head of the pipeline. For example, if the immediately preceding instruction (i.e. the instruction now under execution) causes a branch to another portion of the program, the invalid instruction being concurrently fetched may well not be required in this sequence of operations since the program will branch to a different instruction stream. Thus it is desirable to defer the commencement of the fault mechanism for as long as possible, which is usually to the point where the invalid instruction is prepared to execute or has just begun execution.

For example, from U.S. Patent No. 4,298,927 there is known a pipelined system wherein the validity of the instructions, which can be of variable length (2, 4 or 6 bytes), is latched when the instructions are fetched into the input of the pipeline. The validity information is advanced along the pipeline together with the corresponding instruction to indicate the validity/invalidity of the instruction in the corresponding stage of the pipeline. This validity information is used only to determine the validity of the instruction(s) in the output stage of the pipeline. Therefore if an instruction's length is such that it 'spans' two stages of the pipeline, since there is no provision for checking the validity of all the bytes, the instruction will begin to be executed and the execution will then be aborted if the remaining bytes, once they are fetched into the input of the pipeline, are subsequently found to be invalid.

Accordingly, it is an object of the present

invention to provide an improved prefetch validation mechanism wherein the above disadvantages may be overcome or at least alleviated.

It is another object of the instant invention to provide an improved prefetch validation mechanism such that in the event of an invalid instruction, the instruction execution unit executes a fault at the latest possible time during the execution of the instruction sequence.

It is a further object of this invention to provide a more efficient instruction execution technique in a pipelined CPU.

Brief Summary of the Invention

These and other objects of the present invention will become apparent to those skilled in the art upon consideration of the accompanying specification, claims and drawings.

In accordance with the invention there is provided a data processor which executes predetermined sequences of instructions comprising programs stored in a memory, each instruction consisting of at least m instruction units, where m is at least equal to one, the data processor comprising:

an n-stage instruction pipeline for storing in each stage at least one of said instruction units, where n is an integer greater than one;

n serially-connected validation latches, each corresponding to a respective stage of the instruction pipeline and serially advancing in synchronism therewith, for storing a validation bit indicative of the invalidity of any instruction unit stored in the corresponding stage of the instruction pipeline;

instruction execution means, coupled to the instruction pipeline and responsive to one of said instructions stored in the instruction pipeline beginning at an output end thereof, the instruction execution means executing said one of said instructions and providing an instruction prefetch signal during the execution thereof;

instruction prefetch means, coupled to the instruction execution means and responsive to the instruction prefetch signal, for fetching each instruction unit of the next instruction in said predetermined sequence of instructions from said memory;

prefetch error detecting means coupled to the instruction prefetch means and to the validation latch corresponding to the input stage of the instruction pipeline, for detecting errors occurring during the fetch of each instruction unit which indicate the invalidity of that instruction unit, and storing the validity bit as an indication of said detected error in the validation latch corresponding to the input stage of the instruction pipeline into which said invalid instruction unit is stored;

instruction pipeline control means, coupled to the instruction prefetch means and to the instruction pipeline, for storing each instruction unit fetched by the instruction prefetch means into an input stage of the instruction pipeline after selectively advancing the instruction unit(s) stored in each stage of the instruction pipeline into the next higher stage thereof, until the first instruction unit of said next instruction is stored in the output stage of the instruction pipeline; and

instruction validation means, coupled to the instruction pipeline and to the validation latches, for generating an invalid instruction signal if the validation bit stored in the validation latch corresponding to the output stage of the instruction pipeline indicates the invalidity of any instruction unit stored in the corresponding stage of the instruction pipeline;

characterized in that, where m is greater than one and where up to n of said m instruction units comprising said next instruction are stored in a plurality of said stages of said instruction pipeline, the instruction validation means generates the invalid instruction signal if the validation bit stored in the validation latch corresponding to any of said plurality of said stages indicates the invalidity of any instruction unit stored in the corresponding stage of the instruction pipeline.

Brief Description of the Drawings

One data processor in accordance with the present invention will now be described by way of example only with reference to the accompanying drawings, in which:

Figure 1 is a block diagram of a representation of the elements of an embodiment of the invention as employed in a virtual memory oriented processor;

Figure 2 is a representation of instruction words resident in memory at the boundary of the physical address space;

Figure 3 illustrates the fetch and verification patterns for instructions of various lengths;

Figure 4 is a logic diagram representative of circuitry used to implement the verification and fault mechanisms of the instruction execution unit of the instant invention.

Detailed Description of the Invention

Figure 1 is a block diagram representing the elements employed in this invention in a virtual memory oriented data processor.

A central processing unit (CPU) 2 has a plurality of address lines A1-23 connected to a memory management unit (MMU) 4 which provides a mapping function and address translation to translate the sixteen megabyte logical address range of the CPU (23 address lines) to a physical memory address within a much smaller main memory 6. In the particular configuration of Figure 1, the generalized CPU 2 may be, for example, a microprocessor similar to the MC68000 microprocessor unit presently manufactured by Motorola, Inc. Certain logic to be described later would be incorporated into such a processor in order to implement the instant invention. The MMU 4 unit may be an MC68451 memory management unit also manufactured by Motorola, Inc.

In general, the CPU 2 includes a program

counter 8 for providing the logical address to the MMU 4. In response to receiving the logical address, the MMU 4 examines a set of segment descriptors stored therein to determine whether this requested logical address is presently mapped into a physical address within the main memory 6. If the logical address is so mapped, the MMU 4 translates the logical address into the corresponding physical address for output to the main memory 6. In the event the logical address presented by the CPU 2 to the MMU 4 is not mapped into the physical address space of main memory 6, the MMU 4 generates a fault signal to indicate to the CPU 2 that a bus error has occurred. In general, the CPU 2 will respond to the fault signal by initiating a data transfer from a disk 10 or other suitable secondary memory system, sometimes through a direct memory access unit (DMAC) 12, to the main memory 6 in order to load into the main memory 6 the information at the requested logical address originally presented by the CPU 2 to the MMU 4.

In the instant invention, the instruction execution unit of the CPU 2 operates on data words 16 bits in length. Since an instruction, in the instant embodiment, may constitute one, two or three words, the data words are fetched from the main memory 6 in a pipelined fashion. That is, all instruction words are initially loaded from main memory 6 into an Instruction Fetch Buffer 14. A sequencer generates signals to sequence the occurrence of subsequent events. One output signal from the sequencer causes the word in the Instruction Fetch Buffer 14 to be loaded into an Instruction Decode Buffer 16, and the next sequential word in main memory 6, which may be a second instruction or second word of the first instruction, is loaded into this Instruction Fetch Buffer 14. On subsequent signals from the sequencer the word in the Instruction Decode Buffer 16 is partially decoded to begin execution and a fetch is begun of the next sequential word, which may be or the third word of a three word instruction from the main memory 6. The word in the Instruction Decode Buffer also is transferred to the Instruction Register 17 where execution continues after the prefetched word is placed in the Instruction Fetch Buffer 14 and the word previously there enters the Instruction Decode Buffer 16. In this manner, instructions can be fetched from main memory 6 during the execution of a previous instruction to improve the efficiency in the operation of CPU 2. In this embodiment, a pipeline of two stages is used. In other embodiments the pipeline could be of different depth. An Illegal Instruction Decode Register 18 is coupled to the Instruction Register 17 to detect illegal instruction as will be further explained later. The Instruction Register is also coupled to a Microcode ROM Address Generator 19 which is part of the sequencer noted above. Coupled to the MMU 4 is an Instruction Fetch Buffer Valid Latch 20 which is in turn coupled to an Instruction Decode Buffer Valid Latch 22 as will be later discussed.

Figure 2 shows a representation of instruction words as they may be aligned relative to the boundary line between those portions of code resident in main memory 6 and those currently remaining in the disk 10. For example, in Figure 2a, each of the blocks represents one 16-bit instruction word. The denominator of the fraction in each of the blocks indicates number of words in that particular instruction while the numerator indicates which word of the instruction is contained in that particular block. The vertical dashed line between the first two blocks and the third and fourth block represents the boundary between the main memory 6 and the virtual memory or disk-resident data which is not physically present in the main memory 6. In the example shown in Figure 2a, the block or "page" of data which has been transferred from the disk 10 into the main memory 6 contained only the first two words of a three-word instruction. After the contents of the first block (leftmost in Figure 2) has been loaded into the Instruction Decode Buffer 16 of Figure 1, the contents of the second block will be in the Instruction Fetch Buffer 14. At this point, the first word of the instruction is ready to begin execution at the same time the fetch of the third word of the instruction is to take place.

In Figure 2b, the two words within the physical memory depict a two-word instruction which is fully resident in main memory 6. Figure 2c shows two successive one-word instructions which were located adjacent the boundary and thus both of those instructions are completely within main memory 6. FIG. 2d shows a one word instruction followed by the first word of a three word instruction.

In each case, when the processor instruction stream is about to cross the boundary between an instruction word which was actually stored in main memory 6 and an instruction word not so stored, a page fault indication is returned for the next prefetch. In all but the case set forth in FIG. 2a, the current instruction may be fully executed, and the taking of the fault postponed until this instruction has been executed. For example, in FIG. 2b, the first word of the two word instruction is ready to begin execution (i.e., it is in the Instruction Decode Buffer 16), and the second word has been successfully fetched (i.e., certain logic to be described later, has not indicated otherwise). Execution of this instruction may proceed, then, since it is irrelevant as to whether the next prefetch is valid or not.

In FIG. 2c, since the instruction presently in the Instruction Execution Buffer 16 is a one-word instruction, execution of it may proceed. Subsequently the second single-word instruction will also be executed since the success of the next following prefetch is irrelevant to the execution of the second single-word instruction.

In the case of FIG. 2d, the instruction in the Instruction Decode Buffer 16 will execute, but the subsequent instruction will not, since an error will have been noted on the fetch of the second word of the instruction. This is in contrast to the case of

FIG. 2a where the processor must verify that the third instruction word has indeed been fetched successfully before the instruction begins execution.

FIG. 3 illustrates the fetch and verification patterns of instructions of various lengths. In FIG. 3a, in the case of one word instructions, the validity of the instruction words is checked at the beginning of the execution of the instruction, and the instruction must simply prefetch another word and keep track of its success. In the case of a two word instruction, as shown in FIG. 3b, the validity of the instruction word is checked at the beginning of the execution of the instruction since the second word has already been fetched and resides in the Instruction Fetch Buffer 14. Before the second word of the instruction may be used, a microcontrol word causes the CPU 2 to examine the successful fetch flag or the Instruction Fetch Buffer Valid flag 20 associated with the stage of the instruction pipeline where the second instruction word is stored. If the word was successfully fetched, instruction execution may continue, including the prefetch of the next two instruction word. If the word was not successfully fetched, the logic aborts the execution of the instruction and a page fault for the unsuccessful fetch of the second word is then taken. FIG. 3c shows the case of a three word instruction which is an extension of the two word case, but includes a validity check on the instruction word resident in the Instruction Fetch Buffer, as in the case above, but also includes a validity check of the third instruction word as it is being prefetched by the first word of the instruction.

FIG. 4 is a logic diagram representative of circuitry which may be used to implement the verification and fault mechanisms of the instruction execution unit of the instant invention. This logic for purposes of this disclosure utilizes signals generated by the MC68000 MPU, and by other similar devices, the specific logic of FIG. 4 or variations thereof is in addition to the logic presently used in the MC68000. A description of the functional signals of the MC68000 (i.e., pin functions) may be found in the published Data Sheet for the device or in the "16-Bit Microprocessor User's Manual" available from Motorola, Inc. A description of the microcode functions of the MC68000 may be found in U.S. Patent No. 4,342,078, assigned to the assignee of the instant invention. That patent and particularly the discussion of FIGs. 4 and 5 therein also provides an explanation of the structure and operation of sequencer suitable for use in the instant invention.

As previously noted, instructions of different lengths are handled differently in the instant invention because of the requirement that a fault resulting from an unsuccessful fetch be taken as late as possible. That is, a normally-executing single word instruction is not interrupted because an invalid fetch attempt is made on a subsequent instruction.

Certain function codes may be provided within a central processing unit which indicate the cycle type currently being executed. The MC68000 mentioned above incorporates such a function code. In FIG. 4, function signals (IFC0, IFC1, IFC2), which are generated for presentation to output pins of the MC6800O, are sensed, and the existence of a certain combination of signals indicating that the user program is being run is used to generate a signal to that effect in the logic shown in block 30 of FIG. 4. The signal TSEN is an internal signal which indicates that an external memory access is being performed, i.e., that the access was not an internal register access. If both these conditions are true, the resulting signal from block 30 is applied to the transistor 46 to gate the bus error signal into the Instruction Fetch Buffer Valid latch 20 as previously shown in FIG. 1. This latch will be latched in the high position only upon the occurrence of an active low input on the bus error signal input pin of the CPU at terminal 44. The transistor 42 is a sustaining transistor that is used upon the occurrence of a particular timing or clock phase.

Thus if the instruction currently being executed is a user instruction an external memory access has been performed, and, additionally, a bus error signal has been received from the MMU 4 indicating a fault condition, latch IFBV 20 will be set indicating that the data being fetched and entered into the Instruction Fetch Buffer 20 is invalid data. The output of this latch is applied to circuitry 50, here represented for simplicity by an AND gate, which AND's the IFBV set signal with a signal ADV generated by the executing instruction word to advance the next instruction word in the pipeline. Upon the occurrence of this condition, when the word in the Instruction Fetch Buffer 14 is advanced to the Instruction Decode Buffer 16, the IFBV flag is advanced to the IDBV latch 22. The IDBV latch 22 is in turn coupled to an AND function 52 such that upon beginning of execution of the word residing in the Instruction Decode Buffer 16, an internal interrupt will occur, forcing a jump in the microcode of the processor and causing the processor to enter a special routine to handle an illegal condition. Thus upon the occurrence of the setting of the latch 20 and the loading of the invalid instruction word into the Instruction Fetch Buffer 14, the circuitry 50 will force a fault only upon the complete execution of the instruction presently in the Instruction Decode Buffer, since the internal interrupt is generated after the pipeline has been advanced. If, however, the instruction within the Instruction Decode Buffer 16 is a two or a three word instruction, while the first word of the instruction is apparently valid, the entire instruction is incapable of execution because of an invalid second or third word, as the case may be, of the multi-word instruction. In this case, the output of the latch 20 is also applied to additional circuitry including an AND gate 54 which AND's the set state of the IFBV latch 20 with certain outputs from the microcode ROM (MR1-MR5) which indicate that the instruction presently being executed is a multi-word instruction and

that the processor is using a later word of the instruction. The setting of the IFBV latch and the indication that the processor is using a later word of a multi-word instruction causes the setting of a latch of block 60 which forces an output and causes the immediate abortion of the current microcode sequence and a jump to a bus error sequence to recover from the error.

The microcode ROM outputs that are sensed by the logic 52 are generated by the very first microcode instructions which are executed at the beginning of execution of an instruction word. Thus the fault, even on a multi-word instruction, is taken prior to any destructive transfer of data within the CPU.

The operation of the logic of FIG. 4 will be described using each of the examples of FIG. 3 above.

Recall that in FIG. 3 those blocks to the right of the dashed line represent undefined instruction words, that is, words which are not presently available in main memory 6. The logic of FIG. 4 is provided to determine at what point the crossing of the dashed line by the instruction stream takes place and when it is detected by the processor and acted upon.

In the case of FIG. 3a, which is the case of a one-word instruction, the execution of the first one-word instruction itself does nothing to determine whether the instruction word it is fetching (i.e. the instruction word immediately to the right of the boundary line, the second one-word instruction having been previously fetched) is valid. This is so because the execution of the first one-word instruction which is known to be valid, is not dependent upon whether any other instruction word is valid. It may be recalled that a feature of this invention is that faults are taken as late in the execution of the instruction stream as possible. Of course, since each instruction word has a provision within it to fetch subsequent instruction words, a fetch will be attempted of the word residing in the block immediately to the right of the boundary, and the appropriate flags will be set indicating that the fetch was unsuccessful, but since this is irrelevant to the execution of the first single word instruction itself, no check is made to see whether, in fact, a flag has been set. In this case, since a user program is being executed, and an external memory access is being made, the IFBV latch 20 is set upon the presence on terminal 44 of the Bus Error signal indicating that an unsuccessful fetch was attempted. With the first one-word instruction located in the Instruction Decode Buffer 16, and the second one-word instruction located in the Instruction Fetch Buffer 14, the execution of the first instruction begins by examining the IDBV latch 22 to determine that the instruction itself is valid, then initiating a fetch of the next instruction word (immediately to the right of the boundary), and advancing the pipeline. The first instruction has been copied in part into the Instruction Register 17, which contains that portion of the instruction initiating data transfers or operations within the CPU. Since the sequencing has been initiated, the contents of Instruction Decode Buffer 16 are no longer needed and the second one-word instruction is read into that buffer on the advance command. Simultaneously the third instruction word is fetched and a bus error results since the fetch is unsuccessful. Since a user program is being executed and an external access is being made, the bus error in conjunction therewith sets the IFBV latch 20 indicating that the contents (just fetched) of the Instruction Fetch Buffer 14 is invalid. Note that no need yet exists to take a fault, since the first one-word instruction may execute properly regardless of the contents of the Instruction Fetch Buffer 20.

Upon completion of the execution of the first one-word instruction, the second one-word instruction, now in the Instruction Decode Buffer may begin execution by beginning its fetch of the next word (fourth word) and initiation of a pipeline advance whereby the second one-word instruction is copied into the Instruction Register Buffer 17 for completion of execution, the invalid word in the Instruction Fetch Buffer is advanced to the Instruction Decode Buffer, and another unsuccessful fetch is attempted.

Since IFBV latch 20 was previously set, the pipeline advance signal sets the IDBV latch 22. The illegal fetch, of course, ensures that the IFBV latch 20 remains set. Again no fault is yet taken, since the currently executing instruction does not require further instruction words for execution. Since IDBV latch 22 is set however, as the second one-word instruction completes execution and the word in the Instruction Decode Buffer 16 begins execution, a signal indicating the start of a new cycle NCE is generated, and since IDBV latch 22 is set an internal interrupt is generated causing a microcode jump to an error handling routing.

In terms of the actual microinstruction sequence, when an instruction word is read into the Instruction Register 17 to begin execution thereof, it is decoded and a portion of the decoded instruction is used, together with other conditionals, to select the first microinstruction from the microROM and nanoROM. The first microinstructions cause the next instruction word to be fetched and the pipeline advanced. Since the executing instruction is a one-word instruction, the microinstructions do not validate the instruction words in the Instruction Decode Buffer 16 or the Instruction Fetch Buffer 14. The sequencer simultaneously performs the remainder of the operations specified by the instruction, for example, a register-to-register data transfer. Lastly the sequencer causes the instruction in the Instruction Decode Buffer 16 to be advanced into the Instruction Register.

In the case of FIG. 3b, the first word of a two-word instruction is transferred to the Instruction Decode Buffer 16, the second word is fetched and loaded into the Instruction Fetch Buffer 14. As the first word begins execution, it examines the IFBV latch 20, since it is a two-word instruction. Since IFBV latch 20 is not set, the first word initiates the

next fetch and a pipeline advance. The first word enters the Instruction Register 17, the second word is transferred to the Instruction Decode Buffer 16 and a fetch is attempted of the third word, which attempt fails. IFBV latch 20 is set and invalid data is transferred to the Instruction Fetch Buffer. As the second word begins execution, it initiates a pipeline advance, is loaded into Instruction Register 17 and initiates another fetch, which again is unsuccessful. The pipeline advance sets the IDBV latch 22 and the first invalid word is transferred to the Instruction Decode Buffer 16. As the invalid instruction is prepared to begin execution, the NCE signal is generated causing an internal interrupt as above.

In terms of the microinstruction sequence, when the first word of a two-word instruction is read into the Instruction Register 17, the first selected microinstruction may cause the next instruction word to be fetched to the Instruction Fetch Buffer 14 and the pipeline advanced. Since the executing instruction is a two-word instruction, a subsequent microinstruction causes the sequencer to examine the validity of the next instruction word, i.e., the word residing in the Instruction Decode Buffer 16, by examining the Instruction Decode Buffer Valid Latch 22. If the latch is set, a fault is taken. If the latch is not set, the sequencer subsequently performs the remainder of the operations specified by the executing instruction word and then causes the word in the Instruction Decode Buffer 16 to be advanced into the Instruction Register 17.

The case of FIG. 3c is a three-word instruction case, where a fetch and validation must occur simultaneously. The first word of the three-word instruction is in the Instruction Decode Buffer 16 and the second word is in the Instruction Fetch Buffer 14. Both having been successfully fetched, neither the IFBV latch 20 nor the IDBV latch 22 is set.

As the first word begins execution, it initiates a fetch of the third word and a pipeline advance. Since the fetch is unsuccessful, IFBV latch 20 is set and the first word is transferred to the Instruction Register 17. The second word, following the pipeline advance is in the Instruction Decode Buffer 16 and the invalid third word is in the Instruction Fetch Buffer 14 with IFBV latch 20 set.

As the initial sequencing begins, bits MR1-MR5 of the Control ROM are examined. The presence of these bits indicates a three word instruction in Logic 52 and, through AND gate 54 a latch 60 is set which immediately generates an Illegal Instruction signal which is provided to the exception handling logic of the ROM Address generation circuitry (See U.S. Patent No. 4,342,078) prior to the transfer of any register data under control of the contents of the Instruction Register 17 (the first instruction word).

Thus in each case, the fault due to an unsuccessful fetch has been taken at the latest possible time, i.e. any instruction capable of execution is executed even if a subsequent fetch is known to be erroneous.

Claims

1. In a data processor (2) which executes predetermined sequences of instructions comprising programs stored in a memory (6), each instruction consisting of at least m instruction units, where m is at least equal to one, the data processor comprising:

an n-stage instruction pipeline (14, 16) for storing in each stage at least one of said instruction units, where n is an integer greater than one;

n serially-connected validation latches (20, 22), each corresponding to a respective stage of the instruction pipeline and serially advancing in synchronism therewith, for storing a validation bit indicative of the invalidity of any instruction unit stored in the corresponding stage of the instruction pipeline;

instruction execution means (19), coupled to the instruction pipeline and responsive to one of said instructions stored in the instruction pipeline beginning at an output end (16) thereof, the instruction execution means executing said one of said instructions and providing an instruction prefetch signal during the execution thereof;

instruction prefetch means (8), coupled to the instruction execution means and responsive to the instruction prefetch signal, for fetching each instruction unit of the next instruction in said predetermined sequence of instructions from said memory;

prefetch error detecting means (30, 46) coupled to the instruction prefetch means and to the validation latch corresponding to the input stage of the instruction pipeline, for detecting errors occurring during the fetch of each instruction unit which indicate the invalidity of that instruction unit, and storing the validity bit as an indication of said detected error in the validation latch corresponding to the input stage of the instruction pipeline into which said invalid instruction unit is stored;

instruction pipeline control means (19), coupled to the instruction prefetch means and to the instruction pipeline, for storing each instruction unit fetched by the instruction prefetch means into an input stage of the instruction pipeline after selectively advancing the instruction unit(s) stored in each stage of the instruction pipeline into the next higher stage thereof, until the first instruction unit of said next instruction is stored in the output stage of the instruction pipeline; and

instruction validation means (54, 60), coupled to the instruction pipeline and to the validation latches, for generating an invalid instruction signal if the validation bit stored in the validation latch corresponding to the output stage of the instruction pipeline indicates the invalidity of any instruction unit stored in the corresponding stage of the instruction pipeline;

characterized in that, where m is greater than one and where up to n of said m instruction units comprising said next instruction are stored in a plurality of said stages of said instruction pipeline, the instruction validation means (50, 52, 54,

next fetch and a pipeline advance. The first word enters the Instruction Register 17, the second word is transferred to the Instruction Decode Buffer 16 and a fetch is attempted of the third word, which attempt fails. IFBV latch 20 is set and invalid data is transferred to the Instruction Fetch Buffer. As the second word begins execution, it initiates a pipeline advance, is loaded into Instruction Register 17 and initiates another fetch, which again is unsuccessful. The pipeline advance sets the IDBV latch 22 and the first invalid word is transferred to the Instruction Decode Buffer 16. As the invalid instruction is prepared to begin execution, the NCE signal is generated causing an internal interrupt as above.

In terms of the microinstruction sequence, when the first word of a two-word instruction is read into the Instruction Register 17, the first selected microinstruction may cause the next instruction word to be fetched to the Instruction Fetch Buffer 14 and the pipeline advanced. Since the executing instruction is a two-word instruction, a subsequent microinstruction causes the sequencer to examine the validity of the next instruction word, i.e., the word residing in the Instruction Decode Buffer 16, by examining the Instruction Decode Buffer Valid Latch 22. If the latch is set, a fault is taken. If the latch is not set, the sequencer subsequently performs the remainder of the operations specified by the executing instruction word and then causes the word in the Instruction Decode Buffer 16 to be advanced into the Instruction Register 17.

The case of FIG. 3c is a three-word instruction case, where a fetch and validation must occur simultaneously. The first word of the three-word instruction is in the Instruction Decode Buffer 16 and the second word is in the Instruction Fetch Buffer 14. Both having been successfully fetched, neither the IFBV latch 20 nor the IDBV latch 22 is set.

As the first word begins execution, it initiates a fetch of the third word and a pipeline advance. Since the fetch is unsuccessful, IFBV latch 20 is set and the first word is transferred to the Instruction Register 17. The second word, following the pipeline advance is in the Instruction Decode Buffer 16 and the invalid third word is in the Instruction Fetch Buffer 14 with IFBV latch 20 set.

As the initial sequencing begins, bits MR1-MR5 of the Control ROM are examined. The presence of these bits indicates a three word instruction in Logic 52 and, through AND gate 54 a latch 60 is set which immediately generates an Illegal Instruction signal which is provided to the exception handling logic of the ROM Address generation circuitry (See U.S. Patent No. 4,342,078) prior to the transfer of any register data under control of the contents of the Instruction Register 17 (the first instruction word).

Thus in each case, the fault due to an unsuccessful fetch has been taken at the latest possible time, i.e. any instruction capable of execution is executed even if a subsequent fetch is known to be erroneous.

Claims

1. In a data processor (2) which executes predetermined sequences of instructions comprising programs stored in a memory (6), each instruction consisting of at least m instruction units, where m is at least equal to one, the data processor comprising:

an n-stage instruction pipeline (14, 16) for storing in each stage at least one of said instruction units, where n is an integer greater than one;

n serially-connected validation latches (20, 22), each corresponding to a respective stage of the instruction pipeline and serially advancing in synchronism therewith, for storing a validation bit indicative of the invalidity of any instruction unit stored in the corresponding stage of the instruction pipeline;

instruction execution means (19), coupled to the instruction pipeline and responsive to one of said instructions stored in the instruction pipeline beginning at an output end (16) thereof, the instruction execution means executing said one of said instructions and providing an instruction prefetch signal during the execution thereof;

instruction prefetch means (8), coupled to the instruction execution means and responsive to the instruction prefetch signal, for fetching each instruction unit of the next instruction in said predetermined sequence of instructions from said memory;

prefetch error detecting means (30, 46) coupled to the instruction prefetch means and to the validation latch corresponding to the input stage of the instruction pipeline, for detecting errors occurring during the fetch of each instruction unit which indicate the invalidity of that instruction unit, and storing the validity bit as an indication of said detected error in the validation latch corresponding to the input stage of the instruction pipeline into which said invalid instruction unit is stored;

instruction pipeline control means (19), coupled to the instruction prefetch means and to the instruction pipeline, for storing each instruction unit fetched by the instruction prefetch means into an input stage of the instruction pipeline after selectively advancing the instruction unit(s) stored in each stage of the instruction pipeline into the next higher stage thereof, until the first instruction unit of said next instruction is stored in the output stage of the instruction pipeline; and

instruction validation means (54, 60), coupled to the instruction pipeline and to the validation latches, for generating an invalid instruction signal if the validation bit stored in the validation latch corresponding to the output stage of the instruction pipeline indicates the invalidity of any instruction unit stored in the corresponding stage of the instruction pipeline;

characterized in that, where m is greater than one and where up to n of said m instruction units comprising said next instruction are stored in a plurality of said stages of said instruction pipeline, the instruction validation means (50, 52, 54,

* C L M S *

60) generates the invalid instruction signal if the validation bit stored in the validation latch corresponding to any of said plurality of said stages indicates the invalidity of any instruction unit stored in the corresponding stage of the instruction pipeline.

2. A data processor as set forth in claim 1 wherein the means for prefetching instruction units comprises a microinstruction sequencer.

Patentansprüche

1. Datenpozessor (1), der vorbestimmte Befehlsfolgen ausführt, die Programme enthalten, die in einem Speicher (6) gespeichert sind, wobei jeder Befehl aus wenigstens m Befehlseinheiten besteht, wobei m wenigstens gleich 1 ist, enthaltend:

eine n-stufige Befehlspipeline (14, 16) zum Speichern wenigstens einer der Befehlseinheiten in jeder Stufe, wobei n eine ganze Zahl größer als

n in Serie geschaltete Bestätigungsverriegelungsschaltungen (20, 22), die jeweils einer entsprechenden Stufe der Befehlspipeline entsprechen und seriell synchron damit fortschalten, um ein Bestätigungsbit zu speichern, das für die Ungültigkeit einer in der entsprechenden Stufe der Befehlspipeline gespeicherten Befehlseinheit kennzeichnend ist;

eine Befehlsausführungseinrichtung (19), die mit der Befehlspipeline verbunden ist und auf einen der in der Befehlspipeline gespeicherten Befehle anspricht, beginnend mit einem Ausgangsende (16) derselben, wobei die Befehlsausführungseinrichtung diesen Befehl ausführt und während der Ausführung ein Befehlsvorausholsignal abgibt;

eine Befehlsvorausholeinrichtung (8), die mit der Befehlsausführungseinrichtung verbunden ist und auf das Befehlsvorausholsignal anspricht, um jede Befehlseinheit des nächsten Befehls in vorbestimmten Befehlsfolge aus dem Speicher herauszuholen;

eine Vorausholfehlerermittlungseinrichtung (30, 46), die mit der Befehlsvorausholeinrichtung und mit der Bestätigungsverriegelungsschaltung entsprechend der Eingangsstufe der Befehlspipeline verbunden ist, um Fehler zu ermitteln, die während des Herausholens einer jeden Befehlseinheit auftreten und die die Ungültigkeit jener Befehlseinheit anzeigen, und zum Speichern des Bestätigungsbits als ein Kennzeichen des ermittelten Fehlers in der Bestätigungsverriegelungsschaltung entsprechend der Eingangsstufe der Befehlspipeline, in der die ungültige Befehlseinheit gespeichert ist;

eine Befehlspipelinesteuereinrichtung (19), die mit der Befehlsvorausholeinrichtung und mit der Befehlspipeline verbunden ist, um jede Befehlseinheit, die von der Befehlsvorausholeinrichtung erfaßt worden ist, in eine Eingangsstufe der Befehlspipeline einzuspeichern, nachdem selektiv die in jeder Stufe der Befehlseinheit gespeicherte(n) Befehlseinheit(en) in die nächsthöhere Stufe derselben vorgeschoben worden ist (sind), bis die erste Befehlseinheit des nächsten Befehls in der Ausgabestufe der Befehlspipeline gespeichert ist; und

eine Befehlsbestätigungseinrichtung (54, 60), die mit der Befehlspipeline und mit den Bestätigungsverriegelungsschaltungen verbunden ist, um ein "Ungültiger Befehl"-Signal zu erzeugen, wenn das Bestätigungsbit, das in der der Ausgabestufe der Befehlspipeline entsprechenden Bestätigungsverriegelungsschaltung gespeichert ist, die Ungültigkeit irgendeiner Befehlseinheit anzeigt, die in der entsprechenden Stufe der Befehlspipeline gespeichert ist;

dadurch gekennzeichnet daß wenn m größer als ist und bis zu n der m Befehlseinheiten, die der nächste Befehl enthält, in einer Mehrzahl dieser Stufen der Befehlspipeline gespeichert sind, die Befehlsbestätigungseinrichtung (50, 52, 54, 60) das "Ungültiger Befehl"-Signal erzeugt, wenn das Bestätigungsbit, das in der Bestätigungsverriegelungsschaltung gespeichert ist, die einer der genannten Stufen entspricht, die Ungültigkeit der in der entsprechenden Stufe der Befehlspipeline gespeicherten Befehlseinheit anzeigt.

2. Datenprozessor nach Anspruch 1 bei dem die Einrichtung zum Vorausholen von Befehlseinheiten einen Mikrobefehlszuordner enthält.

Revendications

1. Un processeur de données (2) qui exécute des séquences d'instructions prédéterminées constituant des programmes enregistrés dans une mémoire (6), chaque instruction étant constituée par au moins m éléments d'instruction, avec m au moins égal à un, ce processeur de données comprenant:

un pipeline d'instructions à n étages (14, 16) destiné à enregistrer dans chaque étage l'un au moins des éléments d'instruction, n étant un entier supérieur à un;

n bascules de validation (20, 22) connectées en série, chacune d'elles correspondant à un étage respectif du pipeline d'instructions et l'information contenue dans ces bascules avançant en série en synchronisme avec l'avance de l'information dans le pipeline, ces bascules ayant pour fonction d'enregistrer un bit de validation qui indique l'invalidité d'un élément d'instruction quelconque qui est enregistré dans l'étage correspondant du pipeline d'instructions;

des moyens d'exécution d'instructions (19), qui sont connectés au pipeline d'instructions et qui réagissent à l'une des instructions enregistrées dans le pipeline d'instructions, en commençant à une extrémité de sortie (16) de ce dernier, les moyens d'exécution d'instructions exécutant l'instruction précitée et produisant un signal de pré-extraction d'instructions pendant son exécution;

des moyens de pré-extraction d'instructions (18) qui sont connectés aux moyens d'exécution d'instructions et qui réagissent au signal de pré-extraction d'instructions en extrayant de la

* C L M S *

mémoire chaque élément d'instruction de l'instruction suivante dans la séquence d'instructions prédéterminée;

des moyens de détection d'erreur de pré-extraction (30, 46) qui sont connectés aux moyens de pré-extraction d'instructions et à la bascule de validation qui correspond à l'étage d'entrée du pipeline d'instructions, pour détecter des erreurs qui apparaissent pendant l'extraction de chaque élément d'instruction et qui indiquent l'invalidité de cet élément d'instruction, et pour enregistrer le bit de validité, à titre d'indication de l'erreur détectée, dans la bascule de validation qui correspond à l'étage d'entrée du pipeline d'instructions dans lequel l'élément d'instruction invalide est enregistré;

des moyens de commande de pipeline d'instructions (19), connectés aux moyens de pré-extraction d'instructions et au pipeline d'instructions, pour enregistrer dans un étage d'entrée du pipeline d'instructions chaque élément d'instruction qui est extrait par les moyens de pré-extraction d'instructions, après avoir fait avancer sélectivement jusqu'à l'étage immédiatement supérieur du pipeline d'instructions le ou les éléments d'instruction qui sont enregistrés dans chaque étage du pipeline d'instructions, jusqu'à ce que le premier élément d'instruction de l'instruction suivante soit enregistré dans l'étage de sortie du pipeline d'instructions; et

des moyens de validation d'instructions (54, 60), connectés au pipeline d'instructions et aux bascules de validation, pour produire un signal d'instruction invalide si le bit de validation qui est enregistré dans la bascule de validation correspondant à l'étage de sortie du pipeline d'instructions, indique l'invalidité d'un élément d'instruction quelconque qui est enregistré dans l'étage correspondant du pipeline d'instruction;

caractérisé en ce que, lorsque m est supérieur à un et lorsque jusqu'à n des m éléments d'instruction qui constituent l'instruction suivante sont enregistrés dans un ensemble des étages du pipeline d'instructions, les moyens de validation d'instructions (50, 52, 54, 60) produisent le signal d'instruction invalide si le bit de validation qui est enregistré dans la bascule de validation correspondant à l'un quelconque des étages de l'ensemble d'étages, indique l'invalidité d'un élément d'instruction quelconque qui est enregistré dans l'étage correspondant du pipeline d'instructions.

2. Un processeur de données selon la revendication 1, dans lequel les moyens destinés à pré-extraire des éléments d'instruction comprennent un séquenceur de micro-instructions.

* C L M S *