PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 4 : G06F 9/00, 9/30	A1	(11) International Publication Number: WO 89/09962 (43) International Publication Date: 19 October 1989 (19.10.89)

(21) International Application Number: PCT/US89/01393

(22) International Filing Date: 5 April 1989 (05.04.89)

(30) Priority data:
178,382 6 April 1988 (06.04.88) US

(71)(72) Applicant and Inventor: KING, Ed [US/US]; 4945 Norris Road, Fremont, CA 94536 (US).

(74) Agents: HARRIMAN, J., D., II et al.; 2049 Century Park East, Suite 1200, Los Angeles, CA 90067 (US).

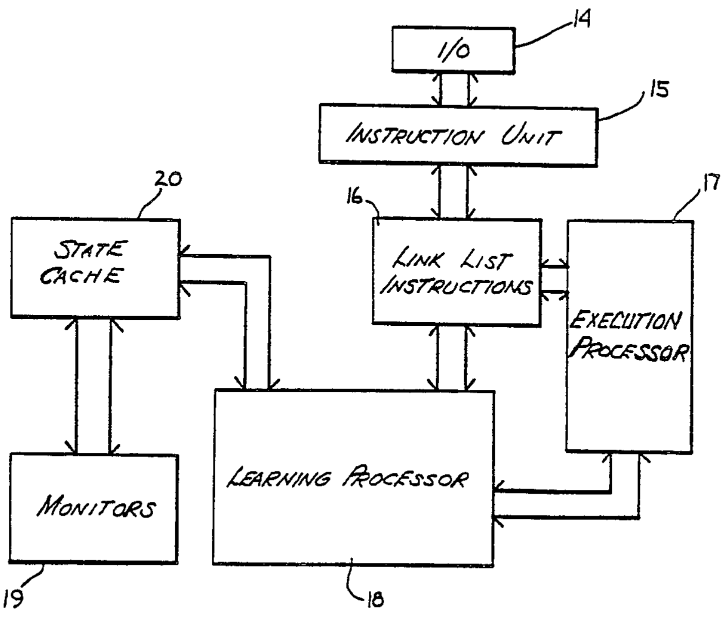
(81) Designated States: AT, AT (European patent), AU, BB, BE (European patent), BF (OAPI patent), BG, BJ (OAPI patent), BR, CF (OAPI patent), CG (OAPI patent), CH, CH (European patent), CM (OAPI patent), DE, DE (European patent), DK, FI, FR (European patent), GA (OAPI patent), GB, GB (European patent), HU, IT (European patent), JP, KP, KR, LK, LU, LU (European patent), MC, MG, ML (OAPI patent), MR (OAPI patent), MW, NL, NL (European patent), NO, RO, SD, SE, SE (European patent), SN (OAPI patent), SU, TD (OAPI patent), TG (OAPI patent).

Published
With international search report.

(54) Title: METHOD AND APPARATUS FOR IMPROVED EXECUTION OF INSTRUCTION SEQUENCES

(57) Abstract

The computer system of the present invention provides a method and apparatus for executing instruction sets. The processor (17) of the present invention creates a link list of instructions (16). The link list includes the origin and destination of the operation to be executed in the particular instruction step. A "learn processor" (18) reviews the link list instruction set and updates pointers to eliminate instruction steps that do not require execution. The learn processor (18) tracks results from instruction sequences, which produce an output. The variable inputs and outputs are stored in a state cache (20). If the variable inputs do not change between one cycle sequence and the next the correct result is simply placed at the proper location in the instruction sequence.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	FR	France	ML	Mali
AU	Australia	GA	Gabon	MR	Mauritania
BB	Barbados	GB	United Kingdom	MW	Malawi
BE	Belgium	HU	Hungary	NL	Netherlands
BG	Bulgaria	IT	Italy	NO	Norway
BJ	Benin	JP	Japan	RO	Romania
BR	Brazil	KP	Democratic People's Republic of Korea	SD	Sudan
CF	Central African Republic			SE	Sweden
CG	Congo	KR	Republic of Korea	SN	Senegal
CH	Switzerland	LI	Liechtenstein	SU	Soviet Union
CM	Cameroon	LK	Sri Lanka	TD	Chad
DE	Germany, Federal Republic of	LU	Luxembourg	TG	Togo
DK	Denmark	MC	Monaco	US	United States of America
FI	Finland	MG	Madagascar		

METHOD AND APPARATUS FOR IMPROVED EXECUTION OF INSTRUCTION SEQUENCES

BACKGROUND OF THE INVENTION

1. FIELD OF THE INVENTION

This invention relates to the field of computing systems, and in particular to a method and apparatus for increasing the speed and efficiency of execution of a sequence of instructions.

2. BACKGROUND ART

Computing systems perform tasks by executing a series of instruction steps known as a program. The performance of a computing system is directly related to the speed with which it can execute these instruction steps. Typically, a processing means, such as a central processing unit (CPU), microprocessor or other processing engine is utilized to execute instruction steps. The processing means is driven by a clock which enables the processing means, allowing the execution of instructions or portions of instructions in fixed increments of time. The clock rate is one factor in determining the performance of a computing system. Other factors include the number of instructions to be executed and the number of instructions which can be executed in a single clock cycle. The prior art has attempted to increase the efficiency of computer systems by reducing the number of instructions and/or increasing the number of instructions which are executed in each clock cycle. In order to increase the performance and speed of computer systems, some manufacturers have employed reduced instruction set computers (RISC) chips and others are implementing a standard complex instruction set (CISC) chip.

One disadvantage of utilizing RISC chips is the requirement of a new architecture and new software, costing time and money. The use of CISC architecture has the disadvantage of narrow applicability to only certain types of applications. In addition, these solutions are limited to chip level improvements and do not provide a method and apparatus for improving the efficiency of all computer systems, regardless of the processing means. Therefore, it would be advantageous to increase the speed of execution of existing software applications and provide a method and apparatus for improved execution of any set of executable instructions.

It would also be advantageous to improve the perfomance of low cost computing systems, such as personal computers, to achieve performance heretofore only available on minicomputers or mainframes. For example, personal computers have not been widely used in certain operating environments and applications because they lack the speed and performance of minicomputers and main frame computers. Work stations, artificial intelligence (AI) applications and other computation intensive applications require specialized and expensive hardware. The specialized nature of such hardware often makes it incapable of running the vast array of the existing software available to the personal computer market. It would be advantageous to combine the speed and performance of a super or super mini computer with the low cost and existing software library of a personal computer.

Personal computers are generally built around a standard microprocessor such as, for example, the Intel 80286 or 80386. Generally, the performance of the microprocessor is a controlling factor in the performance of the machine itself.

There are several factors which control the speed of a standard microprocessor. Initially, the technology of the microprocessor has an effect on the speed of the microprocessor. For example, the technology (CMOS, NMOS, bipolar etc.) determines the clock rate of the microprocessor. The faster the clock rate, the faster the microprocessor. However, clock speed is not the only controlling factor in microprocessor performance. The number of clock cycles per executed instruction, in combination with clock speed, is the principal determinant of microprocessor performance. More simplified, the performance of a microprocessor can be described as follows:

$T=N{\cdot}I{\cdot}C$

T=Time to execute program

N=Number of instructions in program

I=Average number of clock cycles per instruction

C=Basic clock cycle time.

Because the clock speed of the microprocessor is a function of circuit technology, it cannot be improved by a computer designer. However, the number of instructions in a program and the average number of clock cycles per instruction can be improved by "pipe lining" the architecture of the computer system to maximize efficiency in execution. A typical bottleneck in increasing computer system efficiency is the lack of memory bandwidth. Typical computer buses are too slow to take full advantage of the memory bandwidth of an Intel 30286 microprocessor, for example. An IBM PC/AT, for example, has a memory band width of 4 megabytes per second. The Intel 80286 microprocessor requires 2.87 bytes per op code. Therefore, to achieve performance of ten million instructions per second (mips) would require over 27 megabytes per second of memory bandwidth, not available in the standard IBM PC/AT.

A second bottleneck is the interrupt structure within a computer system in dealing with third party controllers, i.e. peripherals. Other speed bottlenecks occur within the processor itself related to the ability to perform complex instructions in a single clock cycle.

It is therefore an object of the present invention to provide a method and apparatus for improving the effieciency of execution of instruction steps in a computing system.

It is another object of the present invention to provide a computer system having a reduced number of clock cycles per executed instruction.

It is a further object of the present invention to provide a computer system which will operate at high speeds.

It is still another object of the present invention to provide a computer system whose performance is substantially independent of the technology of the processing means.

SUMMARY OF THE PRESENT INVENTION

The present invention provides an improved method and apparatus for executing instruction sets. The present invention utilizes a dual bus architecture to provide greater memory access bandwidth and allowing increased system operation. A high speed access bus is utilized for memory access and a standard bandwidth bus is utilized for third party controllers and peripherals. The processor of the present invention creates a link list of instructions. The link list includes the origin and destination of the operation to be executed in the particular instruction step. A "learn processor" reviews the link list instruction set and updates pointers to eliminate instruction steps that do not require execution. The learn processor tracks result from instruction sequences, which produce an output. The variable inputs and outputs are stored in a state cache. If the variable inputs do not change between one cycle sequence and the next, the correct result is simply placed at the proper location in the instruction stream, eliminating the necessity of executing the instruction sequence. By eliminating all unnecessary instructions, the clock cycles per instruction ratio is improved, leading to more efficient system operation. The "learn processor" also evaluates link list instructions to determine origin points and destination points. Sequential instructions which have identical origin points and destination points may be collapsed to an "ORG" point which is, in the preferred embodiment, a link list instruction location which performs a plurality of instructions simultaneously.

In the preferred embodiment of the present invention, an I/O port couples the present system to an instruction set. The instructions are routed to an instruction unit for conversion to a link list format. Each element in the link list has an execution field and a pointer field. The execution field includes the instruction operand, source and destination. The pointer field includes a pointer to another location in the link list. After conversion to the link list format, the link list instruction elements are stored in a link list memory.

A learning processor and an execution processor are coupled to the link list memory. The execution processor executes the instructions in an element of the link list and proceeds to the next element indicated by the pointer field. The learn processor reviews each link list element to determine if the instruction can be (a) collapsed onto another link list element, (b) omitted from execution in general and (c) omitted from execution at the present time.

Link list elements containing instructions having similar destinations or origins, or which have open descriptors, may be collapsed into a single link list element. Thus, if three consecutive instructions fetched data from similar memory locations, for example, these instructions can be collapsed into a single link list element. Each link list element which cannot be collapsed onto a prior link list element is defined as an "ORG" point. ORG points consist of link list elements having a single instruction or a plurality of instructions.

Certain instructions need never be executed by the execution processor. For example, a no-op (no operation), need not be executed. When a no-op is encountered, the learn processor updates the pointer field of the prior link list element to skip over the no-op and proceed to the next executable link list element.

Often, programs sequences are encountered which manipulate one or more variables to produce a desired result. Each time the sequence is encountered, the same result is produced if the variables have not changed. The learn processor is coupled to a state cache which stores the variables and result. When the sequences are encountered, the learn processor pulls the state cache to see if the variables have been changed. If not, the result is inserted into the proper location and the sequence is not performed. In one embodiment of the invention, if the variables have been changed, the sequence of steps is performed separately from the main sequence program so that when the sequence is encountered, the new value may be automatically inserted at the proper location. Alternatively, the sequence could be performed as part of the main program when the variables have been changed and the program sequence is encountered. The present invention provides a method and apparatus for improving the efficiency of execution of a sequence of instruction steps, regardless of the particular application or environment of the sequence.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram illustrating prior art computer system.

Figure 2 is a block diagram illustrating a computer system of the present invention.

Figure 3 is a block diagram illustrating an embodiment of the present invention.

Figure 4 is a block diagram illustrating the "learning processor" of the present invention.

DETAILED DESCRIPTION OF THE PRESENT INVENTION

Method and apparatus for improved execution of instructions sets is described. In the following description, numerous specific details, such as bus width, memory width, number of fields etc., are set forth in detail in order to provide a more thorough understanding of the present invention. It will be obvious, however, to one skilled in the art, that the present invention may be practiced without these specific details. In other instances, well known features have not been described in detail in order not to unnecessarily obscure the present invention.

GENERAL OVERVIEW

The present invention is directed to a method and apparatus for improving the performance of computer systems in the execution of instruction steps. In one embodiment, the present invention is directed to improving the performance of a computer system running standard microprocessor based software.

The present invention includes an execution processor means capable of executing a number of instructions in a single clock cycle. A learn processor means is provided which eliminates unnecessary program steps, combines selected program steps, and monitors repeated sequences of program steps so that they are executed only if changes in data have occurred.

By eliminating unnecessary program steps, such as no-ops, the number of instructions to be executed is reduced, improving system performance. Program steps having similar sources and/or destinations are combined and executed simultaneously, further reducing the number of program steps and increasing the performance of the system. Sequences of instructions which produce the same result if the variables do not change are monitored, with the result stored in a state cache. If, when the sequence of instructions is next encountered, the variables have not changed, the result is inserted into the program at the appropriate location and the sequence is not executed, further reducing the number of program steps and improving the performance of the machine.

The learn processor of the present invention tracks the history of the instruction steps being executed and eliminates and compresses the steps, reducing the number of instructions and correspondingly the number of clock cycles necessary to execute the program. This differs from prior art schemes which are predictive in nature.

As discussed previously, the time it takes a computer system to execute a program is dependent on N, the number of instructions in the program, I, the average number of clock cycles per instruction, and C, the basic clock cycle time. N and I are functions of architecture while C is a function of circuit technology. For purposes of this application, performance increases are achieved for all values of C. The present invention has equal application to any clock cycle time and or processing technology. In other words, improvements in clock cycle time will result in improvements in performance of the present invention. However, the focus of the present invention is on improving performance by optimizing the values of N and I. Other prior art computer systems view N and I as fixed and rely on improvements in chip technology to increase performance. Therefore, one goal of the present invention is to both reduce the number of instructions and reduce the average number of clock cycles per instruction.

The present invention is directed to general algorithm machines. A general algorithm machine is defined, for the purposes of the present application, as any machine, computer system or execution system which executes a sequence of instruction steps. The instruction sequence of a general algorithm machine can be generally described as follows:

TABLE I

1. Fetch instruction
2. Decode instruction
3. (optional) Fetch secondary operands
4. Do address calculation
5. Fetch data (register, memory, or other)
6. Execute instruction
7. Store results
8. Add size of instruction due to current location counter.
9. Check for interrupts
10. Return to One

The goal of prior art computer systems is to execute the above listed ten steps in as few clock cycles as possible. In an ideal "no overhead" machine, one instruction is executed each clock cycle and with pipe lining, all ten of the above general algorithm steps are performed in each clock cycle (at separate locations). However, even in such a "no overhead" machine, all instructions are fetched (whether they need to be executed or not) and only a single instruction is fetched at any one time.

In order to increase the efficiency of the operation of prior art general algorithm machines, many prior art systems have used a solution that is predictive in nature. In other words, some method is implemented to predict which registers, program steps, or data are likely to be utilized by the system. These are then stored in a memory cache for rapid access by the computer system as they are needed. When variations occur, the predictions are updated and revised. The efficiency of such prior art systems is limited by their very predictive nature. By definition, it is impossible for such systems to always provide the correct items at the correct time. Further, such predictive systems do not eliminate unnecessary program steps, nor do they collapse a sequence of instructions into a single step.

LEARN MODE

The present invention, on the other hand, uses a "learning" mode on the entire set of instructions. In a learn mode, the system remembers what is required at each step of the program as well as the next step required at that particular time. In this manner, unnecessary program steps can be eliminated. In addition, the present invention tracks and combines sequences of instructions having similar origin points and similar destination points.

The present invention implements the learn mode by defining and identifying the beginning instruction of a sequence of instructions which can be collapsed into a single step. The instruction set is regrouped into a series of "ORG" points. By definition, an ORG point is the last instruction encountered upon which subsequent instructions can be collapsed. (Accordingly, an ORG point is also any instruction which cannot be collapsed onto a prior instruction.) Correspondingly, an ORG point is an instruction step that must be executed. Instruction sets which need not be executed are not executed in the implementation of the present invention, further saving time. Therefore, two main principles of the present invention are the reduction of number of instructions (N) by eliminating instructions that need not be executed and collapsing sequences of instructions on to ORG points, and reducing the number of clock cycles per instruction (I) by executing a plurality of instructions simultaneously (in the collapsed sequences).

METHOD OF COMBINING INSTRUCTIONS

The learn mode of the present invention combines instructions to promote greater performance and efficiency. Referring again to Table I, the first three steps of a general algorithm machine are made more efficient by the present invention by fetching more than one instruction during a clock cycle whenever possible. Instead of fetching instructions, the present invention fetches ORG points. Because every ORG point consists of one or more instructions (by definition), improved performance is achieved.

However, it is not possible to combine all instruction steps. One limit is determined at step 5 of Table 1, fetch data. If the sources of the data are not similar sources, it is not possible to fetch the data and have it available in a single clock cycle. Therefore, if a sequence of instructions do share similar sources, those instructions are combined in the present invention. For the purpose of the present invention "similar sources" are sources which the computer system can access simultaneously.

In a processor context, for example, similar sources could be memory locations within the "window" of the memory. The window of a memory is generally equal to the width of the memory. If consecutive instructions call memory locations within the memory window and these locations can be accessed simultaneously, the locations are considered to be similar sources. Similarly, if register locations are accessible simultaneously, they are similar sources. Although the above examples relate to processor implementations of the present invention, they are given as examples only. The present invention has equal application to any type of system which executes instructions. Therefore, similar sources are not limited to memory or register locations but relate to any source of data or instructions which can be simultaneously accessed by the system. Further, for systems in which locations not within the memory window may be addressed simultaneously, such locations are considered similar.

At step 7 of table 1, the results of the operation of the instruction on the fetch data are stored. If the results of a sequence of instructions are to be stored in similar destinations, those instructions are susceptible of combination. Therefore, those sequences of instructions having similar destinations may also be combined by the present invention.

As described with respect to similar sources above, similar destinations are destinations (memory locations, registers, or any other type of destination) which can be accessed by the system in question simultaneously. It is not a requirement of the present invention that the destinations be exactly the same location. The present invention is concerned with the ability to move results or other information at the same time to any locations.

Certain instructions have open descriptors, such as JUMP commands and may always be collapsed onto a prior instruction step.

In one embodiment of the present invention, entire sequences of ORG points and/or instructions may be treated as a single step. In certain particular sequences of instructions, certain variables are manipulated to produce a desired result. Often the sequences involve mathematical manipulations of data. If the variables remain constant, the manipulations produce the same result and/or results each time the sequence is encountered. After the sequence has been performed completely once, the variables and results are stored in a state cache and monitored. When next the sequence of steps is encountered, the monitors are polled to see if variables have been changed since the previous execution. If the variables have not changed, the stored result may simply be inserted into the program sequence at the proper location so that the entire sequence of steps is eliminated, improving the efficiency of the system. If the variables have been changed, the particular sequence is executed to produce the new result. As before, these variables and new results are stored and monitored by the learn mode in anticipation of the next time the sequence is encountered. Alternatively, when a variable is changed, the sequence of steps can be executed separately from the main program sequence and the new result may then be inserted at the proper location whenever the sequence is next encountered in the main program.

The following is an example of the application of the learn mode of the present invention to an instruction set which may be found, for example, in a microprocessor environment. The instruction set of the present example is as follows:

TABLE II

1. Move a --> [m]
2. Move b --> [m+2]
3. Move c <-- 0
4. Add [m] + d --> [m]
5. Add d+2 --> d
6. Compare d with immediate value
7. Jump not equal to error
8. Move [m+4] <-- d
9. Jump to 1

When the learn mode of the present invention first encounters the above instruction set, it encounters the instructions consecutively from 1 through 9. When instruction 1 is encountered, the present invention defines that instruction as an ORG point and executes the instruction, generating a result.

When instruction 2 is first encountered, the learn mode analyzes the instruction to see if the instruction may have similar sources or destinations with the present ORG point and therefore be suitable for collapsing onto the present ORG point (instruction 1). In this case, the memory location, (m+2) is within the window area of the first memory location, (m). Therefore, the learn mode collapses these two instructions into a single instruction.

Instruction 3 shares neither similar sources nor destination with the present ORG point (combination of instructions 1 and 2). Therefore, instruction 3 becomes a new ORG point.

When the learn mode of the present invention encounters instruction 4, a comparison is made between the sources and destinations of instruction 4 and the present ORG point, (instruction 3). Since no sources or destinations are shared, instruction 4 is defined as a new ORG point.

Instruction 5 has a matching destination (d) with instruction 4. Therefore, instruction 5 can be collapsed onto the present ORG point, (instruction 4) to form a new ORG point, namely a combination of instructions 4 and 5.

Instruction 6 shares a source (d) with the existing ORG point, and so it too is collapsed into the ORG point. The present ORG point now becomes the combination of instructions 4, 5 and 6.

Instruction 7 is a jump command. Whether or not the command is to be executed at this point is immaterial. The jump command is a decisional command which can be collapsed onto the ORG point and executed at the proper time. The new ORG point then becomes the combination of steps 4 - 7. In the preferred embodiment of the present invention, jump commands are also monitored by a state cache. The parameters making up the decisional aspect of the jump command are constantly monitored and tested outside of the main program. When the jump command is encountered, the state cache is polled to determine if the jump should be executed. If the jump is to be executed, the pointer of the ORG point containing the jump command will have already been updated to the proper location. If the jump command is not to be executed, the pointer will reflect such a state as well.

Instruction 8 shares sources and destinations with the present ORG point. Therefore, instruction 8 is collapsed into the ORG point. Instruction 9 is a jump to the beginning of the loop and is independent of source or destination and can be collapsed into the ORG point. At this point, the instruction set is a series of ORG points as follows.

ORG 1 -Instructions 1, 2

ORG 2 -Instruction 3

ORG 3 -Instructions 4, 5, 6, 7, 8, 9

Using the learn mode of the present invention, an instruction sequence requiring nine clock cycles in a prior art system can be executed in only three clock cycles in the system of the present invention. The method of creating ORG points and the operation of the ORG points in the present invention is described in detail below.

The learn mode of the present invention also takes advantage of the fact that long sequences of instructions may be collapsed if the result of the sequence does not change from one encounter to the next. For example, if a sequence of instructions based on a number of variables yields a result, this result can be stored in a special memory, in the preferred embodiment a "state cache". The variables can also be stored in the state cache. If the variables have not been changed prior to the next encounter of that sequence, there is no need to execute the sequence and the result may be simply inserted into the program at the proper location.

<u>Link List Generation</u>

In order to implement the learn mode of the present invention, instruction sequences are first converted to a link list format. The link listing includes a number of fields including instructions, operands, and a pointer field to point to the next location in the link list to be executed. The link list element consists primarily of an execution field and a pointer field. The execution field stores instructions from the instruction sequence or instruction set to be executed by the system of the present invention. The pointer is a location indicator which directs the execution unit to another location in the link list. The link list is constantly updated during execution of the program. As discussed previously, certain instruction sequences can be collapsed into a single ORG point for execution during a single clock cycle. Referring again to the example list of instructions of Table II above, the generation of the link list is as follows.

When instruction number 1 is read by the learn mode, the learn mode identifies the instruction as an ORG point (because it is the first instruction encountered) and creates a link list element as follows:

Execution field (Instruction 1) Pointer (2)

The pointer points to instruction 2 which is the next instruction in the

instruction sequence. When instruction 2 is read, a comparison of the destination and sources of the instruction is initiated as described above. As noted previously, instruction 2 can be collapsed onto instruction 1. The resulting collapsed instruction is as follows:

Execution field (Instruction 1, Instruction 2) Pointer (3)

The pointer now points to instruction 3, which is the next instruction in sequence. Instruction 3 is read, and a comparison is made between the sources and destination of instruction 3 and the previous ORG point. There is no match between the present ORG point and instruction 3 so the link list now reads as follows:

Execution field (Instruction 1, Instruction 2) Pointer (3)

Execution field (Instruction 3) Pointer (4)

Instruction 4 is encountered and again no match is made between the present ORG point (instruction 3) and instruction 4. Instruction 4 therefore becomes a new ORG point with the link list appearing as follows:

Execution field (Instruction 1, 2) Pointer (3)

Execution field (Instruction 3) Pointer (4)

Execution field (Instruction 4) Pointer (5)

Instruction 5 shares source/destination with instruction 4 and is therefore collapsed onto instruction 4. The link list reads as follows:

Execution field (Instruction 1, 2) Pointer (3)

Execution field (Instruction 3) Pointer (4)

Execution field (Instruction 4, 5) Pointer (6)

Instruction 6 also has a common source/destination with the present ORG point and is therefore collapsed onto the ORG point. The link list now reads as follows:

Execution field (Instruction 1, 2) Pointer (3)

Execution field (Instruction 3) Pointer (4)

Execution field (Instruction 4, 5, 6) Pointer (7)

When reading instruction 7, the learn processor identifies it as an open descriptor and can thus be collapsed into the present ORG point. The link list then reads:

Execution field (Instruction 1, 2) Pointer (3)

Execution field (Instruction 3) Pointer (4)

Execution field (Instruction 4, 5, 6, 7) Pointer (8)

In the preferred embodiment of the present invention, the number of instructions that can be collapsed onto a single ORG point is unlimited. However, depending upon memory configurations, and cost/performance consideration, it may be desirable to limit the number of instructions that can be collapsed into a single ORG point. For purposes of this example, it is assumed that a maximum of 4 instructions may be collapsed into a single ORG point. Therefore, when reading instruction 8, even though it could be collapsed onto the present ORG point due to similar source/destination information, the learn mode labels instruction 8 as a new ORG point. The link list now reads:

Execution field (Instruction 1, 2) Pointer (3)

Execution field (Instruction 3) Pointer (4)

Execution field (Instruction 4, 5, 6, 7) Pointer (8)

Execution field (Instruction 8) Pointer (9)

Since instruction 9 has an open descriptor, it can be collapsed onto instruction 8 so that the final link list of the 9 instructions reads as follows:

Execution field (Instruction 1, 2) Pointer (3)

Execution field (Instruction 3) Pointer (4)

Executed field (Instruction 4, 5, 6, 7) Pointer (8)

Executed field (Instruction 8, 9) Pointer (1)

The pointer of the last ORG point of the link list points back to the first element of the link list and not to any particular instruction. As can be seen, even with a limited number of instructions that may be collapsed into a single ORG point, the present invention results in ORG points in place of the original 9 instruction steps. Because each ORG point may be executed in a single clock cycle, the present invention has reduced the number of clock cycles required to execute the instruction sequence from a minimum of 9 in a no overhead prior art machine to 4 in the present invention.

LANGUAGE PROCESSING

The present invention has application to all types of instruction sets. A micro code instruction set typically begins with a programmer preparing a series of steps in source code (such as fortran, basic etc.). The source code is used to generate "P code" which is an intermediate or source code. Typically, there are two or three source code instructions for each source code instruction. A compiler may be used to generate templets which generate the same sequences of instructions for a particular type of language function. The ratio of templet instructions to P code is greater than 1. The templets are then converted to assembly language and from the assembly language, the hardware generates micro code. The final ratio of micro code instructions to source code instructions can be as large as 200 to 1. By applying the principles of the present invention at any or all stages of code, efficiencies can be achieved prior to execution by a processor. For example, if five source code instructions could be reduced to a single link list element of the present, the resulting micro code may be only 200 lines instead of 1000 lines in the uncollapsed case.

Further, the present invention is not limited to coding environments but to any system which executes sequences of instructions. For example, the present invention has particular use in a language processor. A language processor is a machine which directly executes a high level language such as source code or P code. All programs have repeated sequences which may be collapsed by the learn processor or avoided through use of the state cache.

System Layout

Referring to figure 1, a prior art computer system is illustrated. In such a prior art system, a single bus 10 is used to connect a processor 11, memory 12 and peripherals 13. In a prior art "AT" style system, the bus is limited to 4 megabytes per second, too slow to support high performance computing.

The preferred embodiment of the present invention is illustrated in figure 2. Input output (I/O) means 14 connect the present invention to a source of instructions. Instruction unit 15 coupled to I/O 14 converts the instructions to link list format stored in link list instruction memory 16. The instruction unit 15 pre-fetches streams of instructions from an instruction source in the "outside world". The instruction unit uses typical caching methods for efficient pre-fetching of instructions that has the added capability of reconfiguring the reconstruction list into a link list format. As noted previously, the link list format consists of an instruction field and a pointer field. The link list instruction memory 16 is coupled to an execution processor 17 for executing each of the link list elements. Link list memory 16 is also coupled to learning processor 18 which evaluates each link list element for compression to ORG points. The learning processor 18 also is coupled to the execution processor 17 so that cache updates may be maintained.

As noted above, the present invention combines a plurality of operands into single link list elements. For example, a plurality of "add" instructions may be combined in a single link list element. Therefore, the execution processor 17 must be capable of executing a plurality of adds in a single step.

The learning processor 18 is also coupled to state cache 20 for storing the results of instruction sequences. As previously noted, if none of the variables of the instruction sequence are changed, the state value for that sequence is simply inserted at the proper location each time the sequence is encountered. Monitors 19 are used to indicate whether any changes have occurred in the variables. If changes have occurred, the state cache is updated accordingly.

In the preferred embodiment, the present invention utilizes "resettable" RAMs so that, upon initialization, all memory (state cache, link list, etc.) may be cleared in a single clock cycle.

In operation, the learn processor 18 further divides the instruction field into a plurality of subfields. These subfields include a plurality of source fields identifying the source of the operation, destination fields, identifying the destination of the operation, and an operand field also called a "type class" field which stores the operand of the instruction. The learn processor 18 then compares these fields to the fields of existing ORG points to determine if further collapsing of the link list element can be achieved. As noted previously, if source fields are similar, and collapse of the operand is permissible, a link list element may be collapsed onto a prior element or ORG point.

In the preferred embodiment of the present invention, the sources need not be identical but merely similar. For example, referring to Table II, instruction 1 moves a register value A into a memory location M. Instruction 2 involves writing to a different memory location, but one that is within the window of the original memory location M. Thus, the second memory location is similar to the first memory location. Such sources and/or destinations may be combined in the preferred embodiment of the present invention.

If a link list element is collapsed onto a prior ORG point, thus creating a new ORG point, the learn processor 18 updates the link list 16. Each time the link list is encountered, further compression of the link list elements and ORG points may be accomplished. Thus, efficiency and correspondingly speed advantages may be obtained.

Referring to Figure 3, a block diagram illustrating another embodiment of the present invention is illustrated. A plurality of instructions are stored in memory 30 and coupled through bus 34 to instruction fetch 31. Instruction fetch 31 includes format and decode means 33 for providing execution data to the link list block 36, including the source of the instruction and all information needed to execute the instruction. A slot control block 32 assigns a slot number to each instruction to provide an index to each instruction. The output of format and decode block 33 is a formatted instruction with a number of fields, including a PC field, a number of execution fields, state cache slot number, etc.. When a slot is assigned, a look-up table is updated to reflect the assignment so that instruction elements may be found quickly and easily.

The formatted instructions are coupled to instruction link list block 36 and an instruction link list is generated as described previously. The instruction link list is coupled to and stored in register 37, which may be a number of RAM cache memories. A state cache 38 stores data elements which may be called or accessed by particular instructions in the link list. When a sequence of instructions calls a state cache slot, the data is read and a validation is performed and a write operation is performed on the same phase as the validation.

The execution phase unit 39 accesses register 37 and state cache 38 and the instruction link list element is executed. The learn processor 40 is coupled to the execution phase unit 39 and performs compressions and combinations as outlined above. The instruction fetch 31, instruction link list 36, state cache 38, execution phase unit 39 and learn processor 40 are coupled to BRD bus 35. The BRD bus is a synchronous bus.

Learn Processor

The learn mode processor is illustrated in detail in Figure 4. The learn mode processor monitors the link list elements to determine if the element or group of elements should be executed and to update the learn processor with the result. Another aspect of the learn processor is to monitor a series of results to see if a compression of elements may be achieved. An instruction register 50 receives a number of link list element instructions for review by the learn processor. The register 50 is coupled to an execution unit 51, a descriptor capture block 55, a state RAM cache 52 and current ORG storage 66.

As noted previously, the link list element comprise a number of fields. The learn processor divides the link list element into two types of fields, execution fields and state fields. The execution fields are coupled to the execution unit 51 and descriptor capture block 55. The state fields are coupled to state RAM cache 52. The execution unit acts on the execution fields and produces results and destinations 63. The current ORG point is stored in current ORG register 66 which is updated each time the ORG point changes or a new ORG point is selected.

The state cache RAM 52 is addressed by the state fields of the link list elements and pulls out data. The data is provided to the monitor circuit 53, which is a series of tables. The state cache RAM 52, monitors 53 and validate circuitry 54 make up the state cache 20 of Figure 2. When an instruction is first encountered and converted to a link list element, one descriptor field, a state cache bit, is set (high or low) to indicate that the instruction is in the category of state cache control. In other words, the learn processor is designed to assign certain types and classes of instructions as in the category of state cache control. An "add" for example, can be executed faster in the execution processor than by utilizing the state cache method. Instructions which are appropriate for state cache treatment are instructions which are inefficient, such as procedure calls, returns, jumps, state changing instructions, task switching instructions, and memory management checks.

When an instruction is designated as appropriate for state cache treatment, the descriptor or series of descriptors comprising the instruction (or link list element or ORG point) are stored in the state cache along with the result of those descriptors. Each descriptor is coupled to an address monitor to determine if the address has been written to since the last execution. The address monitor stores the address location of the descriptor. The address monitor is a RAM, and, when the stored memory address is written to, a "dirty" bit is set in the monitor, indicating that the address has been written to and that the result is now assumed to be invalid, so that the sequence of instructions must now be executed to obtain a new result.

The field values of the link list elements are provided to descriptor capture block 55. The descriptor capture block 55 provides the field values, such as, for example, source, destination, instruction type (add, subtract, jump, etc.) to the field comparison circuitry 56. The field comparison circuitry 56 compares the descriptor fields of the present link list element to the descriptor fields of the present ORG point stored in current ORG storage

66. If similar sources and or destinations are found, and the instruction is one which is valid to compress, the present link list element is combined with the present ORG point.

Once the instruction passes from the input point of the execution unit to the output point, multiplex and merge circuitry 58 is provided to either merge the link list element with the existing ORG point or to replace the existing ORG point with the present link list element, forming a new ORG point. The multiplexers 58 are coupled to result register 59, along with validate circuitry 54.

The logic array 61 is coupled to the result register to analyze the results of the field comparison block 56. After the instruction is provided to the result register 59, the link list is either updated, indicating that no more compression can take place, or state cache values are assigned to the state cache RAM 52. The result register provides the results to logic array 61 to look at the results of the field comparisons and state cache finalization. The logic array accesses a state machine 60. The state machine 60 controls BRD bus 35 leading to link list 35. If no match of similar sources or destinations has been made, the logic array 61 signals the state machine 60 to write an ORG point to the BRD bus 35. The logic array 61 may comprise a PLA, PAL, AND/OR gates, etc. The state machine 60 saves the current result stored in the result out register 59 and updates it after each instruction or link list element has been compared to the current ORG point. When no further compression can occur, the logic array signals the state machine to save what it has accumulated thus far and send it to the BRD bus to be stored as a new ORG point in the link list 36.

For example, if three instructions are suitable for compression into a single ORG point, the state machine stores the merged instruction from the result register as the current ORG point and outputs that value to the current ORG register 66. If the fourth instruction is not suitable for compression, the logic array instructs the state machine to send the current ORG point and send it to the BRD bus. The fourth instruction, stored in the result register 59, is then provided to the state machine as a new ORG point.

Although the present invention has been described in terms of a processor environment, it has equal application to any system which executes instructions. The method of the present invention can be used to reduce, eliminate and combine any sequence of instructions in any application.

Thus, a method and apparatus for improved execution of instruction sequences has been described.

CLAIMS

1. A method of defining a plurality of instructions for execution in a computer system comprising the steps of:

converting each instruction to a link list element having and execution field and a pointer for indicating a next link list element to be executed;

determining a source and a destination for each link list element;

combining as a single element, consecutively executable link list elements having similar sources and destinations;

defining each element which cannot be combined with prior elements as an ORG point;

defining said pointers to indicate a next element requiring execution;

whereby said plurality of instructions is contained in a plurality of ORG points fewer in number than said plurality of instructions.

2. The method of claim 1 wherein said execution field contains at least one instruction.

3. The method of claim 1 wherein said combined link list elements have a single execution field having a plurality of instructions and a single pointer.

4. The method of claim 1 wherein said instructions comprise a microprocessor instruction set.

5. The method of claim 1 further including the step of storing each link list element in a first memory means.

6. The method of claim 5 wherein a first processor is coupled to said first memory means and is used to determine sources, destinations and necessity of execution of each link list element.

7. The method of claim 1 further including the steps of:

monitoring sequences of elements whose execution produces a result in response to at least one variable;

storing said variable and said result in a second memory means;

inserting said result at a desired location when said sequence is next encountered in place of executing said sequence when said variable has not changed.

8. The method of claim 7 further including the step of executing said sequence when said sequence is next encountered and said variable has changed.

9. The method of claim 8 wherein a result is stored each time said sequence is executed and inserted at said desired location when said sequence is encountered and said variable has not changed.

10. Apparatus for improved execution of a plurality of instructions comprising:

storage means for storing said plurality of instructions;

transformation means coupled to said storage means for transforming each of said instructions into a plurality of fields;

comparison means coupled to said transformation means for comparing said fields of one of said plurality of instructions with a next sequential of said instructions, said comparison means providing a first signal when said fields are similar and a second signal when said fields are not similar;

combining means coupled to said comparison means for combining said one instruction with said next instruction to form a combined instruction when said comparison means provides said first signal;

execution means coupled to said combining means for executing said combined intruction.

11. The apparatus of claim 10 wherein said transformation means comprises a link list generator for transforming said instructions into execution fields and pointer fields.

12. The apparatus of claim 11 wherein said execution fields includes a source field, destination field and an instruction field.

13. The apparatus of claim 12 wherein said comparison means outputs said first signal when said source field of said one instruction is similar to said source field of said next instruction.

14. The apparatus of claim 13 wherein said comparison means outputs said first signal when said destination field of said one instruction is similar to said destination field of said next instruction.

15. The apparatus of claim 14 further including a state storage means coupled to said transformation means and said execution means for storing a result of a sequence of said instructions and providing said result to said execution means when said sequence is encountered.

16. A method for improved execution of a plurality of instructions comprising the steps of:

determining a source and destination for each of said plurality of instructions;

comparing said source and destination of one instruction with said source and destination of a next sequential instruction;

combining said one instruction with said next instruction into a combined instruction when said source of said one instruction is similar to said source of said next instruction and combining said one instruction with said next instruction into a combined instruction when said destination of said one instruction is similar to said destination of said next instruction;

combining said combined instruction with a next sequential instruction when said source of said combined instruction is similar to said source of said next instruction and combining said combined instruction with said next instruction when said destination of said combined instruction is similar to said destination of said next instruction;

repeating the previous step for each sequential instruction.

17. The method of claim 16 wherein said sources comprise memory locations.

18. The method of claim 17 wherein said memory locations are similar when within a predetermined memory window.

19. An apparatus for executing a plurality of instructions comprising:

an instruction unit for converting said instructions to a plurality of link list elements including a source field, destination field, instruction field and pointer field;

a first storage means coupled to said instruction unit for storing said link list elements;

an execution processor coupled to said first storage means for executing said link list elements;

a learn processor coupled to said first storage means and said execution processor, said learn processor comparing said fields of sequential of said link list elements and combining sequentially executable link list elements having similar sources and destinations into an ORG point and providing said ORG point to said first storage means as a link list element.

20. The apparatus of claim 19 wherein said learn processor comprises a second storage means for storing said source and destination fields, a field comparison means coupled to said second storage means for comparing said source and destination fields of sequential of said link list elements, multiplexing means for combining a link list element with a next sequential link list element having similar sources and destinations and for combining previously combined link list elements with a next sequential link list element having similar sources and destinations.

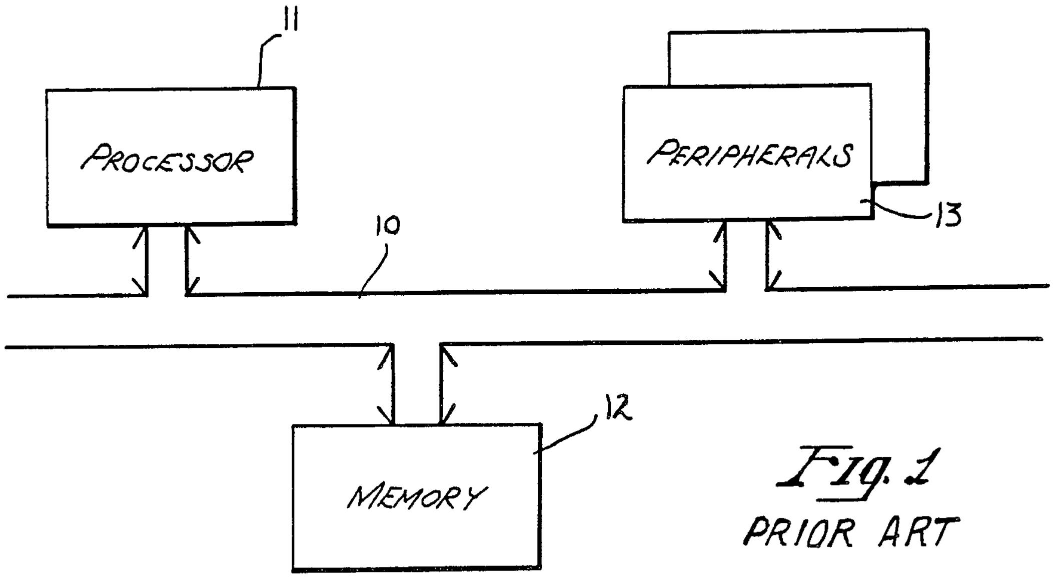


Fig. 1
PRIOR ART

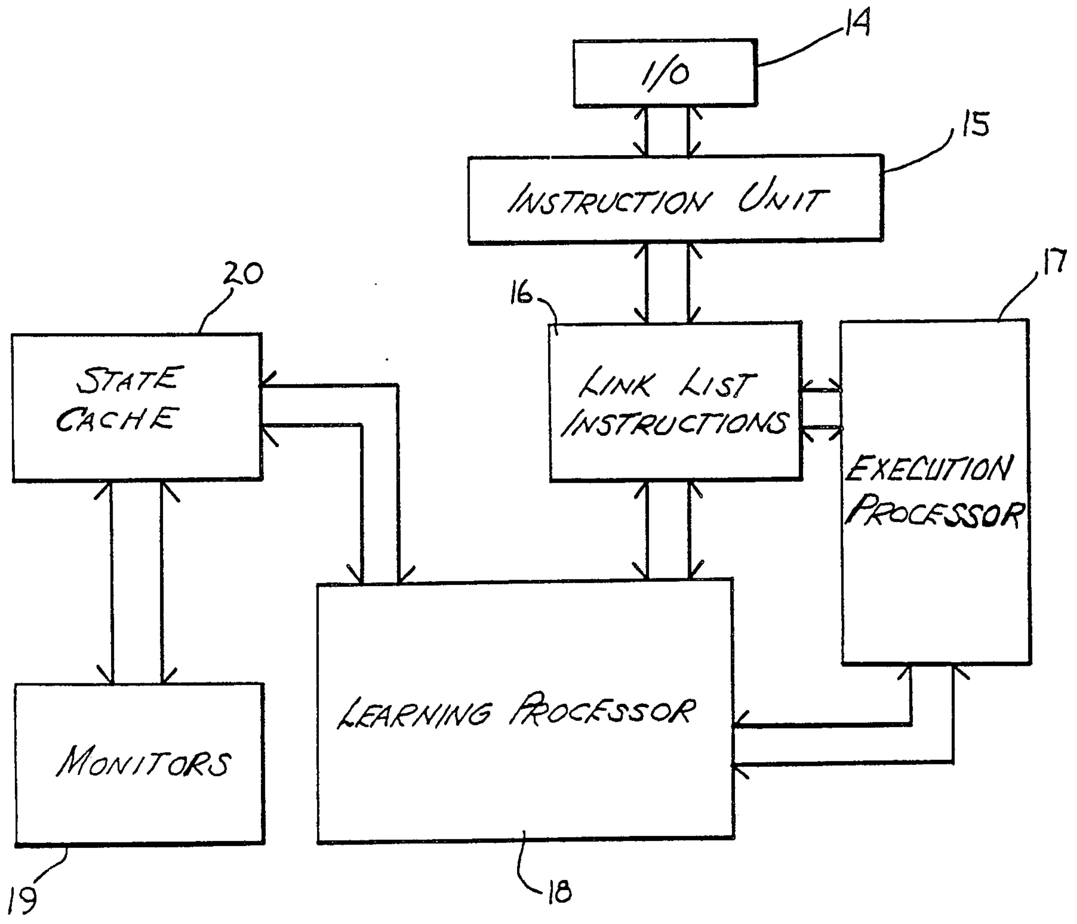


Fig. 2

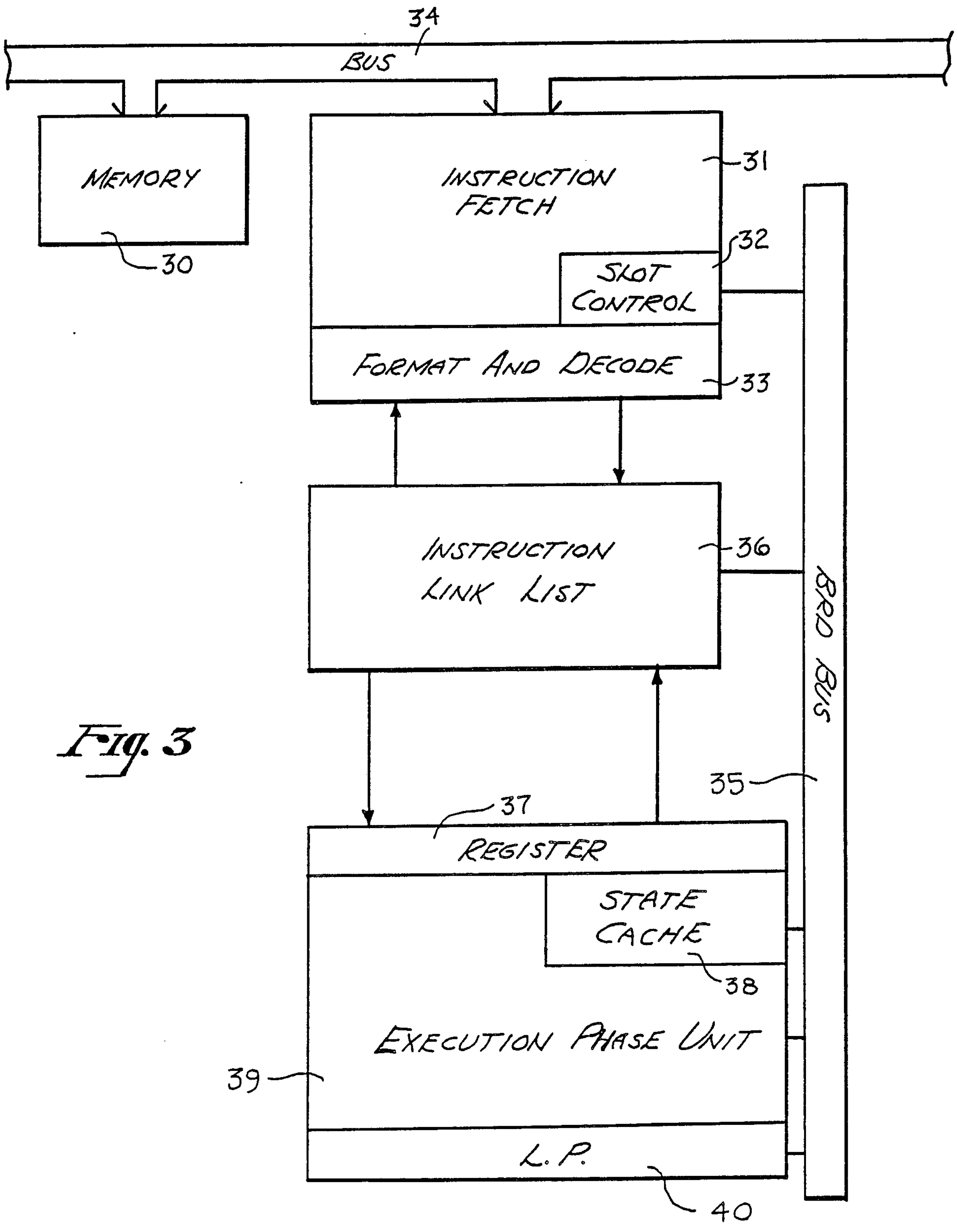

34
BUS
MEMORY
30
INSTRUCTION FETCH
31
SLOT CONTROL
32
FORMAT AND DECODE
33
INSTRUCTION LINK LIST
36
BRD BUS
35
37
REGISTER
STATE CACHE
38
EXECUTION PHASE UNIT
39
L.P.
40

Fig. 3

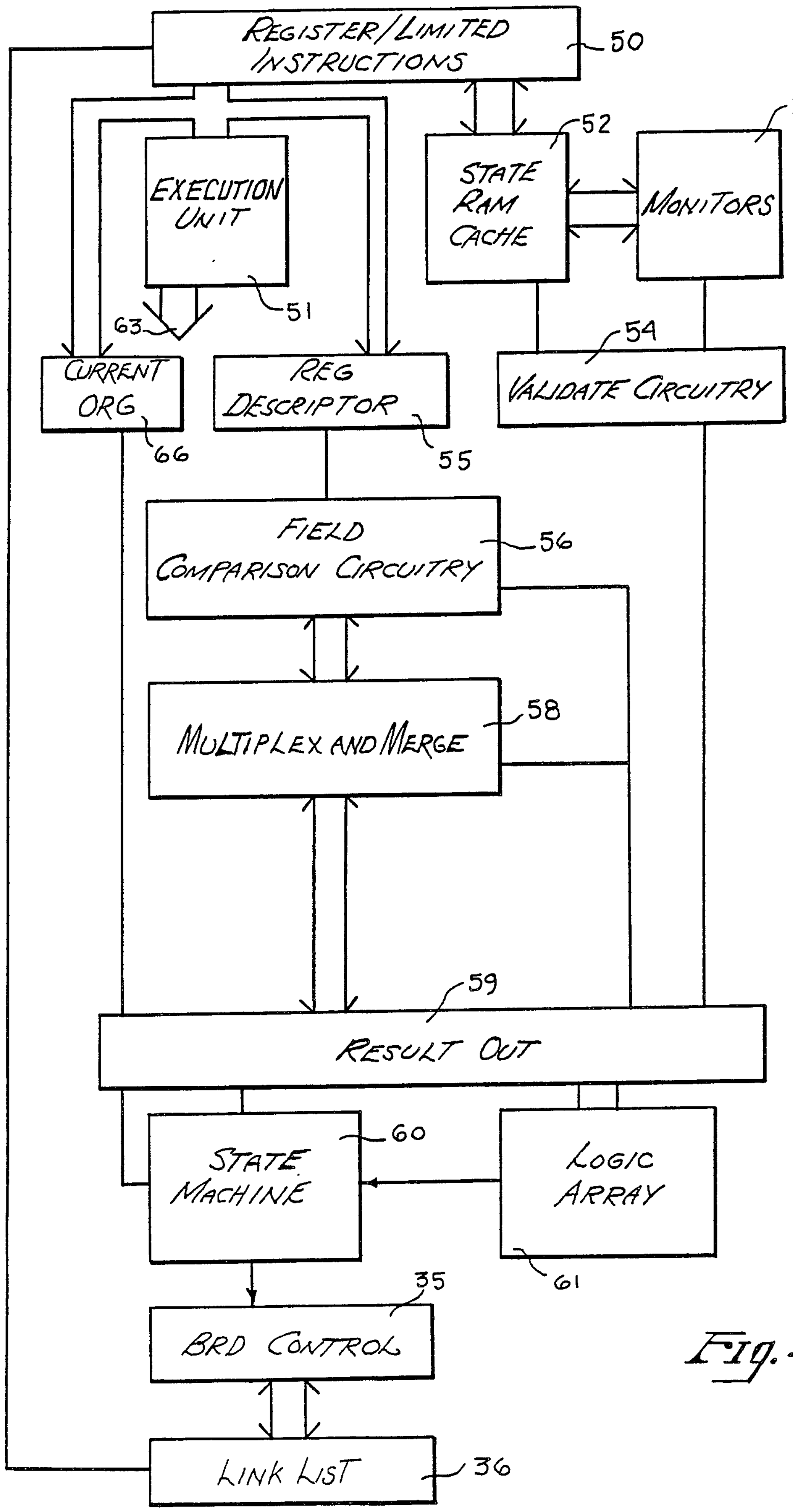

REGISTER/LIMITED INSTRUCTIONS
50
EXECUTION UNIT
51
63
STATE RAM CACHE
52
MONITORS
53
CURRENT ORG
66
REG DESCRIPTOR
55
VALIDATE CIRCUITRY
54
FIELD COMPARISON CIRCUITRY
56
MULTIPLEX AND MERGE
58
RESULT OUT
59
STATE MACHINE
60
LOGIC ARRAY
61
BRD CONTROL
35
LINK LIST
36

Fig. 4

INTERNATIONAL SEARCH REPORT

International Application No. PCT/US89/01393

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all) [6]

According to International Patent Classification (IPC) or to both National Classification and IPC

IPC(4): G06F 9/00, 9/30
U.S. Cl.: 364/300, 200

II. FIELDS SEARCHED

Minimum Documentation Searched [7]

Classification System	Classification Symbols
U.S. Cl.	364/200, 900, 300

Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched [8]

III. DOCUMENTS CONSIDERED TO BE RELEVANT [9]

Category *	Citation of Document, [11] with indication, where appropriate, of the relevant passages [12]	Relevant to Claim No. [13]
A	US, A, 4,567,574 (SAAD'E) 28 January 1986 see entire document.	1-20
A	US, A, 4,454,579 (PILAT) 12 June 1984 see abstract.	1-20
A	US, A, 4,435,753 (RIZZI) 6 March 1984 see abstract.	1-20

* Special categories of cited documents: [10]

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

IV. CERTIFICATION

Date of the Actual Completion of the International Search: 30 MAY 1989	Date of Mailing of this International Search Report: 22 JUN 1989
International Searching Authority: ISA/US	Signature of Authorized Officer: [illegible] DAVID Y. ENG