

Description

This invention relates to data processing systems, and more specifically to a system which provides for execution of nested subroutines and interrupts.

5 The operations carried out in a data processing system and the order in which they occur are determined by means of a program stored in a memory. To alleviate the programmer's problem of completely specifying each series of operations to be performed and to better utilize the capabilities of a data processor, common functions which may be required several times in one program or may be utilized in other programs are usually provided. Each common function may be specified as a subroutine which
10 includes a sequence of instructions.

Subroutines may be utilized for a number of common functions, for example, trigonometric functions using mathematical approximations. Rather than rewrite the series of operations several times, the programmer merely has to write one set of instructions to perform the operation and call it a number of times. Thus, for a routine requiring a value of a trigonometric function, such as cosine, a branch operation to
15 the cosine subroutine in the digital data processor is executed.

It is often convenient for one subroutine to call another subroutine in order to complete its operation. When this situation occurs, complications arise since the return address, i.e., the next address from the branching subroutine, must be stored and recalled by the data processor. Thus while the efficiency of the overall program is enhanced by providing for a particular function in only one place in the data processor, it
20 is often found that inefficiencies arise in calling and returning from the various subroutines both as to time considerations and hardware and/or space limitations.

In the prior art, a number of solutions for transferring to and from subroutines have been provided. One solution involves the transferral of the next instruction location to the first location of the subroutine. In this design, the last subroutine instruction accesses the first location of the subroutine. This first location
25 contains the return address to the next instruction (i.e., one instruction past the instruction that branched to the subroutine) thus enabling the data processor to continue sequencing through the program. This method suffers disadvantages since handling of several common transfers complicates the situation. For example, it is often advantageous to transfer operations from a first subroutine to a second subroutine which utilizes the first subroutine. In other situations, it may be advantageous if the first subroutine recalls itself. These
30 transfers are difficult and sometimes impossible to achieve with data processing systems of the above types without modification or without increasing the number of instructions. When the first subroutine is called for a first time, the address of the next microinstruction in the main routine is transferred to the first subroutine location. When the first subroutine is recalled by an intermediate routine, the address of the next instruction in the intermediate routine is transferred to the same location in the first subroutine thus
35 destroying the original contents. As a result, while the first subroutine can return to the intermediate routine, it cannot return to the main routine.

In data processing systems which permit a first or second subroutine to recall the first subroutine, an instruction may be provided to move the return address to a specified storage location. The last subroutine instruction is then altered so as to include the address of the specified storage location. Although this
40 system permits one subroutine to call another subroutine, i.e., to have the latter subroutine nest, and permit a partially completed subroutine to be subsequently used for other purposes, one reserve memory location and several instructions are required for each nesting level. Increasing the number of these memory locations for each nested subroutine increases the complexity of the control circuitry. In addition, programming complexity is increased because the last subroutine instruction must be modified to address the
45 proper memory location for each subroutine. Therefore, this approach becomes more cumbersome as the number of nesting levels are increased.

Another type of data processing system utilizes vacant locations in memory for storing the return addresses of the subroutine or routine previously executed. The last instruction of the subroutine branches to a designated memory location. In order to exploit this system, however, two registers, one a pointer to
50 the vacant memory locations and another which provides for the current value of the register and the memory address, are required. While this type of data processing unit overcomes some of the previous problems, it still results in supplemental instructions to indicate each new level of subroutine with a concurrent greater execution time required.

A more recent solution to this problem is found in U.S. Patent No. 3,909,797, for "Data Processing
55 System Utilizing Control Store Unit and Push Down Stack for Nested Subroutines". This data processing system has a microprogrammed control store unit which enables the sequencing of the central processing subsystem. This system provides for the nesting of subroutines programmed within the firmware that controls the control store unit, but the principles are equally applicable to software programs or firmware

microprograms. The control store unit provides a branching microinstruction to a microprogram subroutine via a microcommand for enabling the return address of the current operating microprogram routine to be stored. The microcommand also enables a push down stack such that previously stored return microprogram addresses are moved down (pushed) one level. Upon completion of the microprogram subroutine, the control store unit provides a branch field for enabling a multiplexer to select the return address contained on the top of the push down stack and provide it to the current address register in the system. The branch field also raises one level the previously stored return addresses in the push down stack.

Interrupts are used in a data processing system to make it responsive to events that occur asynchronously to the execution of the program. These asynchronous events may be faults within the data processing system or changes of status of devices controlled by the data processing system. For example, by interrupting the data processing system, an input/output (I/O) device can signal that it requires attention or service by the data processing system. Such interrupts can be used to signal the completion of an I/O operation or a system fault such as a memory error. As in the case of subroutines, interrupts can divert the program flow. In the case of interrupts, program flow is diverted to an interrupt service routine which is programmed to handle the type of event that caused the interrupt. Upon completion of the execution of the interrupt service routine, the interrupt service routine must cause the data processor to resume execution of the program at the return address that was stored upon interruption.

One method for storing the return address is to assign fixed locations within memory to each type of interrupt for storing the return address of the program that was interrupted. Using this method, when the interrupt service routine is completed, it can reload the data processor's program counter from the fixed memory location associated with a particular type of interrupt and resume execution of the program that was executing just prior to the occurrence of the interrupt. This method has the disadvantage that interrupt service routines can not easily share common program instructions because in order to return to the interrupted program, the service routine must retrieve the interrupt return address from a fixed memory location that differs with each interrupt.

A different approach to handling interrupt return addresses is found in U.S. Patent No. 4,340,933, for "Data Processing System Having Centralized Nonexistent Memory Address Detection". In this system, which deals with hardware interrupts that interrupt the execution of a firmware microprogram, the microprogram return address is stored in a special register (the hardware interrupt return address register) and further hardware interrupts are prevented until completion of the microprogram hardware interrupt service routine. Inhibiting subsequent interrupts insures that the first microprogram return address stored in the hardware interrupt return address register will not be destroyed by a second return address being stored over it by the occurrence of a second hardware interrupt. This method, although allowing interrupt service routines to share common program instructions because all interrupt services routines restore execution to the interrupted program by reloading the program counter from the one interrupt return address register, still has the disadvantage that interrupts and subroutines can not easily share common program instructions.

Therefore, what is needed is a method of storing return addresses that will allow the nesting of subroutine calls and the servicing of interrupts in a common and efficient manner.

Accordingly the present invention provides in a data processor controlled by a microprogrammed control store, wherein said control store executes different sequences of microinstructions held in addressable locations of a storage unit thereof, and wherein said processor is provided with a pushdown stack for holding a plurality of addresses of said storage unit; apparatus for controlling said data processor to enable intermixed subroutine branching and entry into interrupt service routines from said sequences of microinstructions, comprising:

means controlled while executing a first microinstruction which denotes a transfer to a specified subroutine from the routine/subroutine in execution for incrementing the address of said first microinstruction and pushing such incremented address onto the top of said stack,

means controlled while executing a second microinstruction during which an interrupt occurs for pushing onto the top of said stack the next address being developed during execution of said second microinstruction and for transferring operation to a microinstruction service routine whose first address is determined by the nature of said interrupt, and

means controlled by the last microinstruction in each subroutine and said service routine to pop up said stack and use the address received from the top of said stack as a return address to re-enter the routine/subroutine from which said subroutine or service routine was entered.

A data processing system embodying the present invention will now be described, by way of example, with reference to the drawings, in which:

Figure 1 is a general block diagram of the data processing system;

Figure 2 is a block diagram of a portion of the relevant control area of the microprocessor of the system;
Figure 3 shows the firmware microinstruction word fields of the microprocessor;

Figure 4 is a flow diagram of the execution of microinstructions of a microprogram beginning in a main routine and continuing through two subroutines and two hardware interrupt service routines; and

5 Figure 5 is a logic block diagram of circuitry utilized in the control area of Figure 2.

INTRODUCTORY SUMMARY

The present system provides, in a data processing system microprocessor, a control area which
10 enables the sequencing of the microprocessor. The control area provides a branching microinstruction to the subroutine via a microcommand for enabling the return address of the current operating routine to be stored. The microcommand also enables a push-down stack so that the previously stored return addresses are pushed down one level. The control area further provides for generating vectored interrupt addresses which correspond to the highest priority non-inhibited pending interrupt. These vectored interrupt addresses
15 are provided to the next address register in place of the address generated by the current microinstruction, and the latter address is pushed onto the push-down stack one level upon the occurrence of an interrupt. Upon completion of a subroutine or an interrupt service routine, the control area provides an address field for enabling a multiplexer to select the return address contained in the top of the push-down stack to be provided to the next address register in the control area. This popping of the push-down stack also raises
20 one level of the previously stored return addresses. Means are also provided so that if an interrupt occurs when returning from a subroutine or interrupt service routine, the push-down stack is neither popped nor pushed and the vectored address is provided to the next address register as the address of the next microinstruction to be executed. In the preferred embodiment, the push-down stack is implemented using shift registers.

25

DETAILED DESCRIPTION

Although the above background discussion has been primarily in terms of software and the nesting of software routines and interrupts, the principles are equally applicable to firmware routines and interrupts.
30 The following discussion will be in terms of an embodiment within a microprocessor for use in nesting firmware subroutines and interrupts. For convenience, the embodiment is designed to be capable of nesting four subroutine or interrupt levels. However, in actual practice the number of subroutine or interrupt levels may be much more than four, with an appropriate increase in the hardware which is employed. Alternatively, rather than effecting an enlargement of the hardware, an overflow procedure may be employed. In
35 the latter situation, if it is desired to nest subroutine or interrupt levels in excess of four, the microprocessor could revert to other implemented mechanisms for handling subroutines or interrupts in excess of four.

Figure 1 illustrates the overall system in which the nested subroutine and interrupt implementation of the present system may be used. More particularly, Figure 1 shows a main memory subsystem 10, a
40 central processing unit (CPU) 20 and an input/output (I/O) unit 40. The main memory subsystem 10 consists of modules 12, 14 and 16 interfaced to the central processor bus 20 and the input/output bus 40 via main bus 26, which provides access to and control of all memory modules and input/output units.

The central processing unit 20 executes word oriented instructions that operate on fixed and variable length fields. The basic unit of information in the central processor is a 16-bit word consisting of two 8-bit
45 bytes. These words of information are used in groups of one, two or four for instructions or fixed or floating point operands. Bytes are also used in variable length fields as decimal or alphanumeric data.

CPU 20 is comprised of microprocessor 30, monitor logic 22, and Read Only Storage (ROS) 24. Microprocessor 30 is capable of arithmetic, logic, and control operations, driven by a 48-bit external
firmware microinstruction words from ROS 24. The microprocessor 30 design permits the execution of the CPU 20 16-bit software instruction repertoire.

50 Microprocessor 30 is designed to directly control input/output (I/O) and memory operation for ease in integrated system designs. The design permits greater control and integration by use of a 48-bit external firmware microinstruction word that provides true horizontal microprogramming allowing up to 12 simultaneous micro-operations per 48-bit microinstruction word. The design also permits 8 external hardware interrupts which generate vectors to firmware microprogram routines as well as allowing 5 external software
55 interrupts that are handled under firmware control. In addition, microprocessor 30 provides for 10 external monitor bits originated in monitor logic 22 that are sensed and controlled by sophisticated test branch and major branch operations by logic within microprocessor control area 36 which allows for sophisticated branching operations to be performed within the firmware.

Microprocessor 30 is comprised of 5 major internal hardware logic areas as shown in Figure 1. The five major logic areas are the data manipulation area 32 which includes the arithmetic logic unit (ALU), the memory management unit (MMU 34), the control area 36, the processor bus 37, and internal bus 38.

The processor bus 37 consists of 20 address/data lines, one memory address violation line and three general purpose control lines. Processor bus 37 is connected to main bus 26 and is used to provide addresses to the main memory 10 and input/output unit 40 and to receive and send data to main memory 10 and input/output unit 40.

Internal bus 38 is the major path for communications of information between the other four areas of the microprocessor chip. Internal bus 38 is 20-bit wide. There are 12 sources of information to internal bus 38 under control of the 11 micro-ops within the 48-bit microinstruction word. The ALU is the default source to internal bus 38 if none of the eleven defined micro-ops are used.

The data manipulation area 32 performs arithmetic and logic operations on data and does memory address generation. Data manipulation area 32 is comprised of an indicator register, various discrete registers, a register file, an arithmetic logic unit (ALU), and a shifting mechanism. One of these discrete registers is used as a 20-bit memory address register for the CPU. This program counter contains the 20-bit memory address of the software instruction that is being executed by the central processing unit 20. The register file contains seven 16-bit software addressable data address registers, 7 software addressable base registers, and a variety of other registers all of which are associated with the software program being executed by the CPU 20. The register file also contains a variety of word registers which are used by the firmware program from ROS 24 for temporary storage of information during firmware operations. The shifting mechanism is used to perform various shift operations (i.e., open/close, arithmetic/logic, left/right) on either 16-bit or 32-bit operands.

The control area 36 of microprocessor 30 is logically divided into 3 areas: input latches for control, testable registers, and the next address generation. Control area 36 is described in greater detail in reference to Figure 2 below.

The MMU 34 section of microprocessor 30 is comprised primarily of: a register file, a 12-bit address for base relocation, a 9-bit comparator for checking the size of a memory segment, several 2-bit ring comparators for evaluating access rights to a given segment, and storage flip-flops for indicating potential memory violations. During any CPU generated memory address cycle, the MMU 34 translates the software logical address containing a segment number, a block number and an offset value presented by internal bus 38 into a physical address which is placed on processor bus 37 which in turn is transmitted to main memory 10 via main bus 26.

As can be appreciated from the description so far, CPU 20 executes software programs, the instructions of which are fetched from main memory 10, and performs arithmetic and logical operations on data also contained in main memory 10. The software program executed by CPU 20 has the ability to manipulate general and base address registers that are software visible and the current software instruction is pointed to by a program counter. These general registers, base address registers and program counter which are visible to the software being executed by CPU 20 are physically contained within the data manipulation area 32 of microprocessor 30.

Detailed operation of CPU 20 of Figure 1 is controlled by microprocessor 30 under the control of firmware microinstruction stored in ROS 24. Each location in ROS 24 can be interpreted as controlling one microprocessor machine cycle. As each location of ROS 24 is read, the contents are decoded by control area 36 resulting in a specific operation within microprocessor 30. By grouping ROS locations, firmware microinstruction sequences are obtained that can perform a specific operation or software instruction associated with CPU 20. As each software instruction is initiated, certain bits within the operation code field of the software instruction are used to determine the starting address of the firmware microinstruction routine contained within ROS 24. The testing of certain flip-flops which are set or reset by software instruction decoding done by microprocessor 30 allows the microprocessor to branch to a more specific firmware microinstruction sequence within ROS 24 when necessary.

Connected to main bus 26 is a input/output unit 40. The input/output controller 42 is that portion of the input/output unit 40 which completes a data path from a peripheral device 44 to main memory 10 via main bus 26. I/O controller 42 provides a path through which the peripheral commands are initiated, in addition to controlling resulting data transfers.

Figure 2 illustrates the control area 36 of Figure 1 in greater detail, omitting irrelevant logic and circuitry. Figure 2 also illustrates internal bus 38, monitor logic 22 and read only storage (ROS) 24. In Figure 2, the number next to the upper right hand corner of the blocks represents the number of bits of information contained in the register represented by the block.

The ROS 24 contains the firmware microinstructions (control words) which are used by microprocessor

30 to control the operation of central processing unit and more particularly to execute the software instructions of CPU 20. For each microprocessor machine cycle, a control word is fetched out of ROS 24.

ROS 24 is coupled to ROS data register 65 which receives the microinstruction word fetched from read only storage 24. Each microinstruction contains an address portion and a command portion. The address
5 portion in the microinstruction word identifies the address of the next location to be read from read only storage 24, which will be the next microinstruction to be executed by microprocessor 30. The command portion of the microinstruction identifies the operations to be performed by the microprocessor during the execution of the current microinstruction.

The address portion of the microinstruction word is contained in bits 0 to 12 of the microinstruction
10 word (see Figure 3). The command portion of the microinstruction is contained in bits 13 to 47 (see Figure 3). The command portion is further broken down into a number of fields which comprise subcommands of the microinstruction.

Before describing the microinstruction word in greater detail with respect to Figure 3, the other
15 elements of Figure 2 will be described. Monitor logic 22 provides status information with respect to CPU 20 and is loaded into test flip-flops 50 so that the status may be tested by the firmware. In addition to holding ten bits of dynamic status information from monitor logic 22, test flip-flops 50 hold five bits which sample the status of various CPU locations. The CPU option bits should be thought of as static in nature and indicate whether or not a specific hardware option is present or not within the data processing system. In
20 addition, test flip-flops 50 contains four control flip-flops which are available to be set or reset or to have a bit-transferred under control of the firmware. These four control flip-flops are testable by the firmware. There are also ten temporary flip-flops in flip-flops 50 which are loaded during each firmware cycle with dynamic information such as whether there has been a carry or overflow from bit 16 of the ALU or a carry or
25 overflow from bit 20 of the ALU or whether certain bits on the internal bus 38 are equal to 0, etc. These ten temporary flip-flops are also testable by firmware.

The F-register 51 is a 16-bit instruction register that is loaded from internal bus 38. All bits of the F-
register are testable by firmware. The low-order four bits of F-register 51 also constitute the low-order four bits of the five-bit counter F-counter 52. F-counter 52 is a five-bit counter that can be loaded from internal
30 bus 38. F-counter 52 can be incremented or decremented. The four low-order bits of F-counter 52 are also decoded such that a 16-bit mask can be placed on internal bus 38 under firmware control.

There are five possible conditions that can cause a software interrupt. These conditions are latched in
35 software interrupt register 53. Software interrupt prinet (priority network) 54 prioritizes these conditions and generates a vectored address for input into major branch logic 57.

The next address generation section 55 of control area 36 contains the logic necessary for sequencing
40 the read only storage (ROS) 24. Test branch logic 56 is used to test 64 test conditions which can result in a 2-way branch address for ROS address register 63. These 64 test conditions are testable under firmware control and with the output of the test branch logic 56 being one bit of information into address multiplexer 1 60. Inputs to test branch logic 56 are provided by test flip-flops 50, F-register 51 and F-counter 52.

Major branch logic 57 provides 15 major test branch matrixes. The majority of the inputs to these
45 matrixes are from F-register 51 (in various combinations). Other inputs are from the monitor and option bits of test flip-flops 50. The output of major branch logic 57 is four bits of address information which is provided to address multiplexer 1 60.

Register 58 provides the bits of information that correspond to the ten possible conditions that can
50 cause a hardware interrupt. Hardware interrupt prinet 59 prioritizes these ten possible conditions and produces a four-bit output that is used by address multiplexer 2 62 to produce the 12-bit vectored hardware interrupt address when one of these ten possible conditions occur.

The output of address multiplexer 1 60 provides the 12-bit nominal next address which will be loaded
55 into ROS address register 63 and used to fetch the next microinstruction from ROS 24. This 12-bit address is nominal in the sense that this nominal next address will be used as the next address only if a hardware interrupt does not occur. A hardware interrupt will not occur if no hardware interrupts are pending or if pending hardware interrupts are inhibited by the setting of the interrupt inhibit bit within the microinstruction word (see bit 34 in Figure 3).

Address multiplexer 2 62 is used to select between the 12-bit nominal next address generated by
60 multiplexer 1 60 and the vectored hardware interrupt address that is produced by combining the 4 bits from hardware interrupt prinet 59 with 8 leading 0 bits. The output of address multiplexer 2 62 is the 12-bit next address which is loaded into ROS address register (RAR) 63. The output of RAR 63 is used to provide the address of the next microinstruction to be fetched from ROS 24. The output of RAR 63 is also input to ROS address history register 66.

ROS address history register 66 is provided so that early in the execution of the current microinstruc-

tion contained in ROS data register 65, while the next microinstruction address is being developed and transferred to ROS address register 63, ROS address history register holds the address of the current microinstruction. This current microinstruction address is used in developing the next microinstruction address if the current microinstruction calls for its use. The current address from ROS address history register 66 is also used after being incremented by incrementer 64 as the return address from micro-subroutines and hardware interrupt service routines.

Incrementer 64 increments by 1 the address contained in RAR address history register 66, and is a 12-bit incrementer which is used to source the return address stack 70 via return multiplexer 61 during a PUSH microcommand. The output of incrementer 64 is also used to provide the next ROS address value to RAR 63 for INC and INCK microcommands via address multiplexer 1 60 and address multiplexer 2 62. The INC microcommand specifies that the next ROS address is to be the current ROS address incremented by one and the INCK microcommand specifies that the next ROS address value is to be the current address value plus 1 and in addition a constant, as specified in other unused address field bits within the microinstruction, is to be placed on internal bus 38.

Return address stack 70 is a 4 by 12-bit last in first out (LIFO) array comprised of four registers 71 to 74, used for storing the return addresses of subroutines and hardware interrupts. Return address stack 70 is initialized to hexadecimal value of 001 during clear time and its bottom location is set to 001 (hexadecimal) during each POP (return) microcommand. A PUSH microcommand causes the top of return address stack 70 to be sourced by the output of incrementer 64. A hardware interrupt causes the top of stack 70 to be sourced by the output of address multiplexer 1 60, which is the nominal next address. Incrementer 64 transfers to return address stack 70 the incremented address history from ROS address history register 66 when one subfield of the next command portion of the ROS data register 65 specifies a PUSH microcommand. This PUSH microcommand enables the storing of the return address of the microprogram microinstruction that is being executed while branching to a microprogram subroutine.

Return address stack 70 thus stores the incremented current address provided from ROS address history register 66 when the executing microprogram branches to a subroutine. In addition, return address stack 70 stores the nominal next address output by address multiplexer 1 60 whenever a hardware interrupt occurs which vectors the execution of the firmware to a predetermined location within ROS 24 as determined by the particular hardware interrupt that has occurred. These addresses stored in return address stack 70 point to the next step of the microprogram which would have been executed except for the occurrence of a branch to a microprogram subroutine or a branch to a microprogram interrupt handling routine. Since these addresses will be stored when a branch to a microprogram subroutine occurs, or when a hardware interrupt occurs, the addresses in return address stack 70 will, upon the execution of the last microinstruction in a subroutine or hardware interrupt handling routine, return the microprogram to the proper sequence.

Figure 3 shows the firmware microinstruction word fields of microprocessor 30. This microprocessor word is comprised of 48 bits (bits 0-47). Bits 0 to 12 are used as the ROS address field, bits 13 to 17 are used to select registers in the register file, bits 18 to 22 are used to control the arithmetic and logic functions of the ALU and the inputs to its ports, bits 23 to 25 are used as bus control, bits 26 to 30 are used as a register modification field, bits 31 to 33 are used as memory management unit control, bit 34 is used to inhibit the occurrence of a hardware interrupt and bits 35 to 47 are used as a special control field.

The special control field (bits 35 to 47) is used to modify as well as supplement certain of the other fields in the microinstruction firmware word. The special control field provides up to three simultaneous microcommands during a given microcycle. The special control field is divided into 4 subfields (A to D) as illustrated in Figure 3, with the interpretation of some of the subfields dependent upon the contents of other subfields.

The 48 bits of the microinstructions are loaded into the ROS data register 65 at the beginning of the execution of the microinstruction. These 48 bits are referred to as signals RDDT00 to RDDT47.

The ROS address field contains 13 bits (RDDT00 to RDDT12) and is used to generate the address of the next firmware step in a given microprogram sequence. The method for generating this next address is defined by the first five bits of the ROS address field as shown below in Table 1.

Table 1

	RDDT Bits					Operation
	0	1	2	3	4	
5	1	X	X	X	X	Jump
	0	1	X	X	X	Test Branch
	0	0	1	X	X	Major Branch
10	0	0	0	1	X	Increment With Constant
	0	0	0	0	1	Increment Without Constant
	0	0	0	0	0	Return (POP microcommand)

A PUSH microcommand can be used in conjunction with any of the first five operations listed in Table 1. The PUSH microcommand, when used in combination with a jump or branch microcommand, allows the microprogrammer to store away into return address stack 70 a return address to which the microprogrammer will wish to return upon completion of the subroutine which was branched to. To facilitate the storing away of this return address by the person writing a microprogram, the PUSH microcommand pushes the contents of the ROS history address register 66 incremented by 1 by incrementer 64 onto the top of return address stack 70. The return (POP) microcommand is then used by the microprogrammer as the last firmware step of the called subroutine to return to the first location after the microinstruction which called the microprogram subroutine.

One exception to the next address generation being defined by the six operations described in Table 1 is that of a hardware interrupt. When a hardware interrupt is initiated, the next ROS address will be provided as a hardware vector and the ROS address generated by the ROS address field of the current firmware word will be placed on the top of the return address stack 70 by the output of address multiplexer 1 60 being selected as the output of return multiplexer 61 and pushed onto return address stack 70. If a PUSH microcommand (as specified by special control field C in bits RDDT42 to RDDT44 in conjunction with a special coding of subfields A and B) is used in conjunction with one of the first five operations shown in Table 1, then the microprogrammer must also inhibit hardware interrupts by specifying that RDDT bit 34 is a binary ONE in order to prevent a conflicting push operation being performed as the result of the occurrence of a hardware interrupt. Since a hardware interrupt can occur (normally by the occurrence of an external asynchronous event) at any time prior to the completion of a given firmware microprogram sequence, special consideration must be given to allow for the occurrence of a hardware interrupt.

The branching capabilities of the six operations defined in Table 1 are referred to as page branching and bank branching. A page is defined as 64 consecutive memory locations within ROS 24 and a bank is defined as 1024 memory locations within ROS 24 (16 pages). The branch boundaries for the test branch operation are restricted to any location within a page. The branch boundaries for the major branch are restricted to any location within the bank. The remaining four operations of Table 1 are capable of branching or incrementing from one bank to another.

The jump operation is the only next address generation method of Table 1 that allows a branch to any of the possible 4096 locations of ROS 24. This is accomplished by providing, within the ROS address field, a 12-bit direct address of the next firmware microinstruction to be executed.

When RDDT bit 0 equals a binary ONE, RDDT bits 1 to 12 of the present ROS data word (firmware microinstruction) as contained in ROS data register 65 are delivered directly to ROS address register 63 via address multiplexer 1 60 and address multiplexer 2 62 as the next address in the firmware microprogram, assuming of course that no intervening hardware interrupt occurs. Should a hardware interrupt occur, this nominal next address would be pushed onto the top of return address stack 70 and the generated hardware interrupt vector address, as generated by concatenating eight high order binary ZERO bits with the four bit output of hardware interrupt pinet 59, will be loaded into ROS address register 63 via address multiplexer 2 62 as the next ROS address.

A PUSH microcommand (as defined by special control field C) can be used along with the jump operation. Hardware interrupt inhibit bit RDDT 34 must be set to a binary ONE to inhibit hardware interrupts when the PUSH microcommand is used in conjunction with a jump operation. During a PUSH microcommand, the current ROS address held in ROS address history register 66 is incremented by 1 and pushed onto the top of return address stack 70 by return multiplexer 61 before the next address as specified in RDDT bits 1 to 12 is loaded into ROS address register 63.

The test branch operation of Table 1 is a two-way branch using the result of one of 64 test conditions

specified as part of the ROS address field in bits 3 to 8. All test branches are restricted to branching within the current page; that is, the next ROS address generated as a result of the test will always be one of two locations (depending upon the results of the test: i.e., true or false) eight locations apart but within the page of 64 locations currently being addressed by ROS address register 63. In fact, depending upon the results of the test branch, the next microinstruction will be fetched from the location determined by taking the six high-order bits from the current ROS address (from ROS address history register 66) concatenated with the six low-order bits which come from bits 2, 9, 10, 11 and 12 of the ROS address field which are used for bits positions 7, 8, 10, 11 and 12 respectively within the nominal next ROS address and with the result of the test (1 or 0 corresponding to true or false respectively) being used as bit position 9 within the nominal next ROS address.

This nominal next ROS address composed of the bits as described above, generated as the result of the test branch as specified in bits 0 to 13 in the current firmware microinstruction word, is the next ROS address assuming no intervening hardware interrupt occurs. Should a hardware interrupt occur, this address is placed on the top of return address stack 70 and the generated hardware interrupt vector address replaces it as the contents of the next ROS address in ROS address register 63.

The PUSH microcommand can also be used along with test branch operations. As indicated above, hardware interrupts must be inhibited (by setting RDDT bit 34 equal to a binary ONE) if the PUSH microcommand is used. If the PUSH microcommand is used in conjunction with a test branch operation, the current ROS address, which is the address of the test branch microinstruction (from ROS address history register 66) incremented by 1, will be placed on the top of return address stack 70 and the microprogram will branch to the nominal next address as determined by the output of test branch logic 56.

The major branch operation is a 16-way branch using the results of 15 test groups specified as part of the ROS address field in bits 5 to 8. All major branches are restricted to branching within the current bank (1024 locations). That is, the nominal next ROS address generated as a result of the major branch test will always be in one of 16 locations (depending on the output of the major branch matrix) 16 locations apart but within the bank (1024 locations) currently being addressed by ROS address history register 66. The nominal next ROS address is generated by taking bits 0 to 1 from the current ROS address and using them as bits 0 and 1 in the nominal next ROS address and taking bits 3, 4, 9, 10, 11 and 12 from the ROS address field and using them as bits 2, 3, 8, 9, 10 and 11 in the nominal next ROS address respectively. In addition, bits 4 to 7 in the nominal next ROS address are determined by the four-bit output by major branch logic 57.

The nominal next ROS address as described above is generated as the result of the major branch operation specified in bits 0 to 12 of the current firmware microinstruction word contained in ROS data register 65, and is the next ROS address assuming that no intervening hardware interrupt occurs. Should a hardware interrupt occur, this newly generated nominal next ROS address will be placed on top of the return address stack 70 and the generated hardware interrupt vector address replaces it as the next ROS address in RAR 63.

As in the case of test branch operation, the PUSH microcommand can be used along with major branch operations. Again, as indicated above, hardware interrupts must be inhibited by setting bit RDDT 34 to a binary ONE. If a PUSH microcommand is used in conjunction with a major branch operation, the current ROS address, which is the address of the major branch microinstruction itself (from ROS address history register 66) plus 1, will be placed on top of the return address stack 70 and the microprogram will branch to the nominal next address as determined by the output of major branch logic 57.

The incrementing with a constant operation (INCK microcommand) as specified in ROS address field (bits 0 to 3 of the 13-bit field) of the current microinstruction causes the current value of the ROS address history register 66 incremented by 1 to be placed in the ROS address register for the next microcycle. In addition to this next address generation, the remaining 9 bits (RDDT bits 4 to 12) are used to generate an 8-bit constant plus a filler to the 20-bit wide internal bus 38 during the current microcycle.

The current ROS address contained in ROS address history register 66 is incremented by incrementer 64 and the result is returned to the ROS address register 63 via address multiplexer 1 60 and address multiplexer 2 62 when an increment with constant microcommand is specified in the ROS address field of a microinstruction. Should a hardware interrupt occur, this newly generated next address will be placed on the top of return address stack 70 via return multiplexer 61 and the hardware generated interrupt vector address will be placed in ROS address register 63 to be used as the next ROS address. The PUSH subcommand can be used along with the increment operations. As with all PUSH subcommands, the hardware interrupts must be inhibited by setting RDDT bit 34 to a binary ONE when using a PUSH subcommand in conjunction with an increment operation. If a PUSH microcommand is used in conjunction with an increment operation, the current ROS address incremented by 1 will be placed on the top of return address stack 70, in addition to becoming the next ROS address.

The increment operation (INC) subcommand as specified in the ROS address field of the microinstruction initiates the same operation as described above for the increment with constant (INCK microcommand) operation except that no constant is generated onto internal bus 38.

The return operation (POP microcommand) causes the contents of the top of return address stack 70 to be loaded into ROS address register 63 via address multiplexer 1 60 and address multiplexer 2 62 to be used as the ROS address for the next microcycle. In addition, a ROS address of 001 (hexadecimal) is loaded into the bottom of the stack into register 74 each time return address stack 70 is popped one location. This loading of the bottom of return address stack 70 with the ROS address of 1 is used to detect the case of overpopping of the stack. The overpopping of the stack will result in the microprocessor being vectored to microprogram error sequence which begins at ROS location 1. Because of return operation (POP microcommand) is fully specified by bits 0 to 4 of the ROS address field, bits 5 to 12 of the ROS address field are unused as part of the return operation.

When a return operation is specified in bits 0 to 4 of the ROS address field, ROS address register 63 receives the contents of the top of return address stack assuming no intervening hardware interrupt occurs. Should a hardware interrupt occur, the return (or pop) operation will effectively be bypassed or cancelled. This cancelling of the pop stack operation when a hardware interrupt occurs during a return operation is the logical equivalent of the popping the return address from the top of return address stack 70 and immediately, within the same microcycle, pushing it back onto the top of return address stack 70. It is this cancelling or bypassing of popping return address stack 70 when a hardware interrupt occurs during a return operation that allows the return address stack to be used to contain the return addresses for both microprogram subroutine calls and for hardware interrupts.

By having the push operation onto the stack associated with storing the return address for the hardware interrupt routine cancel the pop operation performed on the stack when returning from a microprogram subroutine or upon completion of a hardware interrupt service routine, the return address stack 70 does not have to be able to simultaneously move in opposite directions or to first pop up and then push down during one microcycle. The method by which this simultaneous movement in opposite directions is prevented (i.e., the method by which the push operation cancels out the pop operation) will be explained in greater detail with respect to Figure 5 below. As will be seen below in the discussion of Figure 4, this cancelling of the stack pop operation associated with a return operation by the occurrence of the push operation associated with the occurrence of a hardware interrupt does not adversely affect the flow of control within the microprogram because the hardware interrupt routine will perform a return operation as the last step in its microprogrammed interrupt service routine.

The PUSH microcommand which is coded in the special control field (RDDT bits 35 to 47) must not be used in the same microinstruction with a return operation which is coded within the ROS address field (RDDT bits 0 to 12) because the results within microprocessor 30 are unspecified.

As described above, a hardware interrupt forces a branch to a fixed ROS address. This ROS address is determined by a priority network (prinet 59) which has various error signals and interrupt requests as inputs from register 58. Hardware interrupts cause the next firmware generated ROS address to be pushed onto the top of return address stack 70. If the next ROS address was to have been generated from the return address stack via a return microcommand, popping of the return address stack 70 is inhibited. Hardware interrupts must be inhibited whenever a PUSH microoperation is performed in order to prevent the requirement to doubly push the stack, the first push being associated with the PUSH microcommand itself and the second push being associated with the saving of the return address for the hardware interrupt.

When the hardware interrupt inhibit field (RDDT bit 34) is a binary ONE, hardware interrupts are inhibited, so that non-error condition hardware interrupts (such as those associated with memory refresh and data request) are inhibited (prevented from intervening between the execution of the current microinstruction and the execution of the next microinstruction). Hardware error condition inputs to prinet 59 (such as system clear, an attempt to access a non-existent resource, an access violation, or a memory parity error) are not under the control of RDDT bit 34 and can intervene between any two given microcycles. The fact that hardware interrupts associated with these error conditions can not be inhibited and therefore could occur during a microinstruction which contains a PUSH microcommand does not cause a problem, because the hardware interrupt service routines associated with these noninhibitable error conditions does not do a return operation at the end and therefore do not depend upon the contents of return address stack 70 being valid. When RDDT bit 34 is a binary ZERO, all hardware interrupts are allowed.

Figure 4 shows an exemplary firmware microinstruction sequence. The main routine 100 in the sequence may be a main microprogram routine or a microsubprogram routine and is any common sequence of microinstructions designed to perform a function. It is desired to be able to perform branches to any common sequence of microinstructions. For exemplary purposes, two microsubroutines indicated at

102 for a first level subroutine and 104 for a second level subroutine are shown. Branches to these subroutines may occur anywhere in the routine or subroutines. In addition, it is desired to be able to perform hardware interrupt service routines to process any hardware interrupts that occur during the execution of microroutine 100 or any of its associated microsubroutines. For exemplary purposes, two hardware interrupt service routines indicated at 106 for a first hardware interrupt service routine and 108 for a second hardware interrupt service routine are shown. In Figure 4, the rectangular boxes in the microprogram indicate microinstructions. The contents of a microinstruction are indicated by the output of ROS 24 as contained in ROS data register 65.

In Figure 4, the microinstruction which determines the jump to the first level subroutine is found at microinstruction P. This microinstruction consists of a ROS address field having a conditional or unconditional branch to microinstruction Q and a special control field indicating a PUSH microcommand along with the hardware interrupt bit being set to inhibit hardware interrupts. This firmware microinstruction branches to microinstruction Q, pushes the return address, and inhibits hardware interrupts. When this microinstruction is reached during the sequencing of the microprogram, the following operations occur. The ROS address field of the microinstruction indicates that a branch operation is to be performed. The special control field indicates that the current ROS address in ROS address history register 66 is to be incremented by incrementer 64 and pushed onto the top of return stack 70 via return multiplexer 61. This is accomplished by incrementer 64 taking the contents of ROS address history register 66, which at this point in time contains the ROS address of instruction P, and incrementing it by 1 so that the output of incrementer 64 is the address associated with instruction P + 1 and storing it via return multiplexer 61 into the top register of return address stack 70, which is register 71. Thus, at this point in time, register 71 will contain the address P + 1. This microinstruction, which resides at address P in ROS 24, also has the hardware interrupt bit set to inhibit the occurrence of a hardware interrupt between the execution of the microinstruction at location P and the first instruction of the first level subroutine, which is the microinstruction at location Q.

Subsequent to the occurrence of the branch, as shown at Q, normal sequencing of the microsubroutine occurs. If the first level subroutine 102 requires another subroutine as shown at R, then another branch to a microsubroutine is executed. Thus, the same steps as occurred at instruction P will be applicable to instruction R when the first level subroutine 102 branches to the second level subroutine 104. Thus, at instruction R, a conditional or unconditional branch is performed to the second level subroutine 104 which begins at address S in ROS 24. When instruction R is executed, the current ROS address in ROS address history register 66 is incremented by 1 by incrementer 64 and is transferred to return address register stack 70 and held in the top register 71 and the previous contents of register 71 are pushed down into register 72. Thus, at this point in the execution of the microprogram, register 71 contains the address R + 1 and the register 72 contains the address P + 1. Address R + 1 is the instruction within the first level subroutine to which the second level subroutine is to return upon its completion and address P + 1 is the address within the main routine to which the first level subroutine is to return upon its completion. The ROS address field of the microinstruction at location R has a value which will cause the microprogram to branch to microinstruction S.

It is assumed in Figure 4 that a first hardware interrupt becomes pending during the execution of the microinstruction R. This first interrupt is of the type that can be inhibited by setting the hardware interrupt inhibit bit (RDDR 34) equal to a binary ONE and therefore the hardware interrupt is not allowed to divert the normal flow of the firmware following microinstruction R. Therefore, upon the completion of the execution of microinstruction R, the branch to location S will be completed and the microinstruction S will be executed. If the first interrupt had been of the error interrupt type, the hardware interrupt would have occurred upon the completion of microinstruction R and the flow of the execution of the microprogram would have been vectored to the location associated with the hardware error interrupt.

During the execution of microinstruction S, the ROS address on the next microinstruction to be fetched from ROS 24 is computed as can be seen from examination of Figure 4. In the absence of the hardware interrupt, which vectors the execution of the firmware to first service routine 106 beginning at location V, the normal next address would be S + 1. This address could be determined by using the ROS address field of the S microinstruction contained in ROS data register 65, which could indicate a jump or branch operation in which case bits within the ROS address field in ROS data register 65 would be multiplexed through address multiplexer 1 60 with or without the addition of bits from test branch logic 56 or major branch logic 57. Alternatively, if an increment operation is indicated in ROS address field of the microinstruction S, the contents of ROS address history register 66 will be incremented by 1 by incrementer 64 and the 12 bit output will be multiplexed through multiplexer 1 60. In any case, the output of address multiplexer 1 60 will be the address S + 1.

Because the microinstruction S does not inhibit interrupts and the hardware interrupt is pending, the

output of address multiplexer 1 60 will be multiplexed through return multiplexer 61 and pushed onto the top of return address stack 70 into register 71 and address $P + 1$ will be pushed from register 72 into register 73 and address $R + 1$ will be pushed from register 71 into register 72. At the same time, address multiplexer 2 62 will multiplex through the output of prinet 59 such that the address associated with the first hardware interrupt will be input into ROS address register 63 such that ROS address register 63 will contain the address V and that microinstruction will be fetched from ROS 24. Upon the completion of the execution of microinstruction S, microinstruction V will have been fetched from ROS 24 and be available for decoding and execution in ROS data register 65.

Before discussing the execution of the first hardware interrupt service routine 106 which contains microinstructions V to X, it should be noted that if microinstruction R did not contain the hardware interrupt inhibit microcommand, then during the execution of microinstruction R, the situation would arise in which a double push of the return address stack 70 would be required in order to preserve the proper execution sequence of the microprogram. That is, both address $R + 1$ as output by incrementer 64 into return multiplexer 61 would have to be pushed onto return address stack 70 followed by address S as output by address multiplexer 1 60 via return multiplexer 61. Because a double push operation is not possible during the execution of one microcycle, the inhibiting of hardware interrupts is required.

If the first interrupt which becomes pending during the execution of microinstruction R was a hardware error interrupt which is not inhibitable by the setting of hardware inhibit bit 34, then after microinstruction R was executed, the microprocessor would branch to microinstruction V and the first hardware interrupt service routine would begin execution without the intervening execution of microinstruction S. In this case, however, the first hardware service 106 would be different from that shown in Figure 4 in that service routines associated with hardware error interrupts do not terminate with a pop operation. Instead, they terminate by branching to a predefined point after doing some system reinitialization function. These hardware error interrupts do not do a pop operation because a pop indicates the intent to resume execution of the microprogram where it was interrupted. These hardware errors that are not inhibitable are not of the type from which a recovery can be done and therefore a return of control to the point of interruption is not done.

Now returning to the first hardware service routine 106 which comprises microinstructions V to X, it will be noted that instruction V and W and all instructions between them have the inhibit hardware interrupt microcode within them (i.e., bit 34) set to a binary ONE. By inhibiting the occurrence of a hardware interrupt during all but the last (POP) microinstruction of a hardware interrupt service routine, the microprogrammer insures that the occurrence of a second or subsequent hardware interrupt will not interrupt the processing of the first hardware interrupt. This insures that any time dependent sequences which are initiated by one microinstruction can be completed by a subsequent microinstruction without any intervening hardware interrupts expanding the time between the initiating and the completing microinstructions. It also has the advantage that because the return address stack is only four deep, three levels of the stack may be used for calling microprogram subroutines and the fourth level can be reserved for hardware interrupts without the possibility that hardware interrupts will have to be nested more than one deep. It will be appreciated that the logic in Figure 2 would in fact permit the nesting of hardware interrupts and if the stack was of sufficient depth to permit all of the hardware interrupts to occur without exceeding the depth of the stack, plus the depth of the maximum nesting depth of microprogram subroutine calls which is permitted, this inhibiting of interrupts during hardware interrupt service routines would not be required (based on the assumption that a given hardware interrupt may not interrupt itself).

During the execution of the first hardware interrupt service routine 106, microinstructions V to W are executed sequentially without interruption. Microinstruction X follows the execution of microinstruction W without interruption because of the inhibit microcommand present in microinstruction W. Of course, microinstructions V to W contain in addition to the INH (inhibit) microcommand, other microcommands which perform operations associated with servicing the particular hardware interrupt which has occurred. Likewise, microinstruction X can contain other microcommands besides the POP microcommand.

During the execution of microinstruction X, ROS data register 65 contains microinstruction X which has within it a return (pop) operation coded in the ROS address field. This return operation causes the popping of return address stack 70 which as indicated above contains address $S + 1$ in address register 71, $R + 1$ in register 72, $P + 1$ in register 73, and the address 001 (hexadecimal) in register 74 prior to the POP microcommand being performed. The POP microcommand in microinstruction X causes the return address stacks 70 to be popped one location. The POP microcommand of microinstruction X causes address $S + 1$ to be popped out of return address stack 70 from register 71 and for the contents of 72 to be loaded into register 71 the contents of register 73 to be loaded into register 72 and the contents of register 74 to be loaded into register 73 and the value 001 (hexadecimal) to be loaded into register 74. Address $S + 1$ is

EP 0 111 407 B1

loaded via address multiplexer 1 60 and address multiplexer 2 62 into ROS address register 63 where it is used to fetch the $S + 1$ microinstruction from ROS 24 which is then loaded into ROS data register 65. Thus, following the execution of microinstruction X, execution of the second level subroutine 104 resumes with the execution of microinstruction $S + 1$.

5 The execution of second level subroutine 104 continues from microinstruction $S + 1$ on an instruction by instruction basis until microinstruction T is reached. After the execution of microinstruction X and prior to the execution of microinstruction T, return address stack 70 contains the address $R + 1$ in register 71, the address $P + 1$ in register 72, and the address 001 (hexadecimal) in registers 73 and 74.

Figure 4 shows a second hardware interrupt becoming pending during the execution of microinstruction 10 T which contains a POP subcommand which, in the absence of the second interrupt, would return the execution of the microprogram to microinstruction $R + 1$ by popping the address $R + 1$ from the top of return address stack 70. However, as indicated above, when a POP microcommand occurs simultaneously with a hardware interrupt command, the popping of the return address stack 70 is cancelled by the implicit pushing associated with responding to the hardware interrupt. Therefore, return address stack 70 is neither 15 popped nor pushed during the execution of microinstruction T and the stack remains unchanged, so that register 71 contains the address $R + 1$, register 72 contains the address $P + 1$ and registers 73 and 74 contain the address 001 (hexadecimal).

During the execution of microinstruction T, pinet 59 generates the vectored address to the second hardware interrupt service routine so that address Y is loaded into ROS address register 63 via address 20 multiplexer 2 62. The execution of the microprogram continues with the execution of microinstruction Y, which is the first instruction in the second hardware interrupt service routine 108, comprising instruction Y and instruction Z. Again, as indicated above, each instruction within a hardware interrupt service routine must inhibit hardware interrupts except for the last microinstruction. Also, as indicated above, instruction Y will contain other microcommands in addition to the inhibit microcommand and microinstruction Z can 25 similarly contain other microcommands besides the POP microcommand.

During the execution of microinstruction Z, ROS data register 65 will contain the return operation in the ROS address field. This return (POP) microcommand causes return address stack 70 to be popped one location thus causing the address $R + 1$ in register 71 to be loaded into ROS address register 61 via 30 address multiplexer 1 60 and address multiplexer 2 62. In addition, the contents of register 72 replace the contents of register 71, and the contents of register 73 replace the contents of register 72, and the contents of register 74 replace the contents of register 73, and the address 001 is loaded into register 74. Thus, at the completion of microinstruction Z, return address stack 70 will contain the address $P + 1$ in register 71, and the address 001 (hexadecimal) in registers 72, 73 and 74.

The popping of return address stack 70 by the POP microcommand in microinstruction Z will cause the 35 microprogram execution to continue with the execution of microinstruction $R + 1$ in the first level subroutine 102. The first level subroutine 102 continues executing its microinstructions until it reaches microinstruction U which contains a POP microcommand which returns the execution of the microprogram to the microinstruction after the microinstruction which branched to the first level subroutine. The POP microcommand of microinstruction U pops return address stack 70 one location and causes the address $P + 1$ to be loaded 40 via address multiplexer 1 60 and address multiplexer 2 62 into ROS address register 63. This will cause the microinstruction located at location $P + 1$ in ROS 24 to be loaded into ROS data register 65 and executed. Thus, execution of the microprogram will continue with microinstruction $P + 1$ in the main routine 100, followed by the next microinstruction which is addressed by the ROS address field of microinstruction $P + 1$.

As can be appreciated from the above discussion of Figure 4, each explicitly microcoded PUSH 45 microcommand associated with the branching to a subroutine causes the return address stack 70 to be pushed down one location, as does each implicitly performed push operation which is associated with responding to a hardware interrupt. From the above example, it can be appreciated that the return address stack was only pushed to a level of three deep, the first push occurring when instruction P branched to the first level subroutine, the second push occurring when instruction R branched to the second level 50 subroutine, and the third push occurring during instruction S in preparation of vectoring to the first hardware interrupt service routine.

In the example in Figure 4, popping of the return address stack 70 occurs three times, the first time during microinstruction X at the completion of the first hardware service routine, the second time occurring at the end of the second hardware interrupt service routine during the execution of microinstruction Z and 55 the third time occurring during the execution of microinstruction U when the first level subroutine returns to the main routine. The pop operation which would normally have been performed at the end of the second level subroutine by microinstruction T was cancelled by the implicit push operation associated with the vectoring to the second hardware interrupt service routine and beginning execution with microinstruction Y.

It should be noted that the microprogram advance from microinstruction T to microinstruction Y is shown in Figure 4 as a change of column because the first column is used for the main routine, the next two columns for subroutines, and the last column for hardware interrupt service routines. This change of column therefore does not indicate a change of nesting level, and the return from microinstruction Z to microinstruction R + 1 is only a single change of level, resulting from a single pop.

Referring now to the detailed circuitry shown in Figure 5, return address stack 70 comprises of twelve 4-bit bi-directional universal shift registers. In Figure 5, only shift register 70A, which holds the most significant bits of the four 12-bit addresses in return address stack 70 and shift register 70K which holds the next to least significant bits of the four 12-bit addresses and shift register 70L which holds the least significant bits of the four 12-bit addresses stored in return address stack 70 are actually illustrated, and for simplicity registers 70B to 70J are not shown. Registers 71 to 74 of return address stack 70 are not shown in Figure 5 because one bit of each of registers 71 to 74 is stored in shift registers 70A to 70L respectively. The method by which the twelve 4-bit shift registers are used as a push down stack will be described in greater detail below.

Figure 5 shows read only storage 24 which stores the microinstructions to be executed by the various units which comprise the microprocessor 30. During a cycle of operation, read only storage 24 is addressed by the contents of ROS address register 63. This causes the contents of the location specified by the address to be read into ROS data register 65.

In Figure 5, ROS data register 65 is illustrated as containing three separate microinstructions which illustrate the three different cases of interest to be described with respect to Figure 5. In actual operation, only one of these microinstructions would be present at any given instance in time within ROS data register 65. The top microinstruction which performs a jump to microinstruction S and also contains a PUSH microcommand. This microinstruction represents microinstruction R of Figure 4 which does the branch to the second level subroutine 104. The middle microinstruction is any microinstruction which does not have hardware interrupts inhibited by the setting of the hardware interrupt (HI) bit within the microinstruction to a binary ONE. The bottom microinstruction is a microinstruction which contains a POP microcommand within the ROS address field and could be any one of the POP instructions of Figure 4 such as microinstruction U, T, X or Z. This POP microinstruction is used to do a return from a subroutine or a hardware interrupt service routine.

Fields of the microinstruction word containing the next address to be accessed and the next commands to control the microprocessor 30 subsystem are transferred and decoded by the functional units illustrated in Figures 1 and 2. Each functional unit includes logic circuits for decoding the requisite subcommands specified by the microinstruction word under the control of the system timing operations.

When first level subroutine 102 calls second level subroutine 104 by performing a branch to S microinstruction with a PUSH and inhibit (INH) microcommands, the following sequence occurs. The branch microinstruction is read out from read only store 24 and loaded into ROS data register 65 and the branching takes place in the cycle following the read out and it is developed from the ROS address field having a binary ONE in bit position ZERO. The branch microinstruction word also has a PUSH microcommand and an inhibit hardware interrupt microcommand which stores the return address into return address stack 70 and inhibits hardware interrupts respectively.

More specifically, the JUMP microcommand is decoded by inverter 87 receiving the binary ONE from ROS address field bit position 0 and producing at its output a binary ZERO such that signal JUMP-, which is connected to the output enable (F) input of driver 60A, enables driver 60A's inputs onto its outputs thus making the output of address multiplexer 1 60 to be the bits 1 to 12 of the ROS address field which contain the address of the S microinstruction. Driver 60A is a bus driver of the type that has 3-state outputs such that only one of the bus drivers that comprise address multiplexer 1 60 will have its outputs enabled at any given time. It should be noted that address multiplexer 1 60 in fact comprises more than driver 60A and 60B and has other inputs besides those illustrated in Figure 5.

The output of address multiplexer 1 60 which appears at the A input of address multiplexer 2 62 will be selected by the select input (SEL) being in the binary ONE state such that the output of address multiplexer 2 62 will be the address of microinstruction S which appears in bits 1 to 12 of the ROS address field. This output of address multiplexer 2 62 will be loaded into ROS address register 63 so that instruction S will be addressed in read only storage 24.

The select input signal LDRSAR+ which appears at the select (SEL) input of address multiplexer 2 62 and return multiplexer 61 is used to select between the A and B inputs of multiplexers 62 and 61. Signal LDRSAR+ which appears at the output active (P) output of hardware interrupt priority encoder 54 will be in the binary ZERO state if one of the outputs Q1 to Q8 of encoder 54 is in the binary ZERO state. One of these outputs will be in the binary ZERO state if one of the inputs I0 to I10 is in the binary ZERO state and

input enable signal DISHIP+ which is attached to the enable input (EI) of encoder 54 is in the binary ZERO state.

Hardware interrupt pinnet encoder 54 is comprised of two cascaded 8-line to 3-line priority encoders. The I0 input of encoder 54 is connected to receive signal CLEARX- which is a signal which will be a binary ZERO during a hardware clear condition. The other input signals I1 to I10 are connected to the various signals which indicate a hardware interrupt request or a hardware error condition. Whenever one of these input signals, I0 to I10, is in the binary ZERO condition, encoder 54 will produce on outputs Q1 to Q8, which are binary encoded, a signal corresponding to the highest priority hardware interrupt or hardware error requesting servicing.

Because the first microinstruction illustrated in ROS data register 65 contains a binary ONE in the hardware interrupt inhibit field, signal RDDT34+ will be a binary ONE partially enabling AND gate 84. Signal CLEARX- will also be a binary ONE because it is not a clear condition. The third input of AND gate 84, signal STOPIT+, will also be a binary ONE because if no hardware error condition is present, therefore the output of AND gate 84, signal DISHIP+, will be a binary ONE and disable the outputs of encoder 54 thereby making signal LDRSAR+ a binary ONE.

The PUSH microcommand of the first microinstruction is decoded by NOR gates 80, 81 and 82 along with NAND gate 83 and inverter 85. The PUSH subcommand is specified in the special control field of the microinstruction word by having an A subfield of 000 binary, a B field of X0X0 binary (where X bits are don't care conditions), and a C subfield of 100 binary. The A subfield is decoded by inputting bits 35 to 37 into NOR gate 82, the output of which will be a binary ONE if a PUSH subcommand is specified. The B subfield is decoded by inputting bits 39 and 41 into NOR gate 81, the output of which will be a binary ONE if a PUSH subcommand is specified. The C subfield is decoded by inverter 85 inverting bit 42 and using its output along with bits 43 and 44 as inputs into NOR gate 80. The output of NOR gate 80 will be a binary ONE if a PUSH subcommand is specified. If a PUSH subcommand is specified in the special control field, all three inputs to NAND gate 83 will be a binary ONE and cause the output thereof, signal PUSHOP- to be a binary ZERO.

With the PUSH signal, PUSHOP-, being a binary ZERO at one input to NAND gate 90, and the other input thereto, signal LDRSAR+, being a binary ONE indicating that no hardware interrupt is both pending and enabled, the output of NAND gate 90, signal PUSHNT+ will be a binary ONE indicating that a push operation on return address stack 70 is to be conducted either because a PUSH microcommand is encoded within ROS data register 65 or a hardware interrupt is pending and hardware interrupts are enabled. Signal PUSHNT+ is one input of NAND gate 91. The other input to NAND gate 91 is POPOP+ which is output by inverter 88 which receives as input signal POPOP- which is the output of OR gate 86.

Bits 0 to 4 of the ROS address field are input into OR gate 86 which is used to decode POP microcommands. If bits 0 to 4 are binary ZEROs, the output of OR 86 will be a binary ZERO which indicates that a pop operation is to be performed. Because bit 0 of the JUMP S microinstruction contains a binary ONE, the output of OR gate 86 will be a binary ONE causing the output of inverter 88 to be a binary ZERO.

Signal POPOP+ being a binary ZERO at one input of NAND gate 91 will cause the output thereof, signal PUSPOP-, to be a binary ONE indicating that both a push and pop operation are not being requested simultaneously. With both inputs of AND gate 92 equal to a binary ONE, the output thereof, signal PUSNOT+, will be a binary ONE indicating that a push operation is to be performed on return address stack 70. With signal PUSNOT+ being a binary ONE at one input of OR gate 94, the output thereof, signal PUSNEN+, will be a binary ONE indicating that a push operation is to be performed on return address stack 70.

The pop operation signal, POPOP+, is also one input to AND gate 93. With both inputs to AND gate 93 being a binary ZERO, the output thereof, signal POPNOT+, will be a binary ZERO indicating that a pop operation is not to be performed on return address stack 70. Because a clear operation is not being performed, the output of inverter 89, signal CLEARX+, will be a binary ZERO, and therefore both inputs to OR gate 95 will be binary ZEROs causing the output thereof, signal POPNEN+ to be a binary ZERO indicating that a pop operation is not to be performed on return address stack 70.

Signal PUSNEN+ is connected to the S0 shift mode input of each of the twelve 4-bit bi-directional shift registers 70A to 70L. Similarly, signal POPNEN+ is connected to the S1 shift mode input of the twelve 4-bit bi-directional shift registers 70A to 70L. Shift mode inputs S0 to S1 are binary encoded such that when S0 and S1 are equal to binary ZEROs, no shifting or parallel loading of the shift register occurs when the clocking signal at the clock (C) input of the shift register transitions from the binary ZERO to the binary ONE state. When S0 is a binary ZERO and S1 is a binary ONE, a left shift (pop) operation will be performed when the shift registers are clocked. When S0 is a binary ONE and S1 is a binary ZERO, a

right shift (push) operation will be performed when the shift registers are clocked. When both S0 and S1 are binary ONES, a parallel loading of all four bit positions within each shift register will be done when the clock transitions from the binary ZERO to the binary ONE state. The reset (R) input of each of the twelve 4-bit shift registers is set to a binary ONE because the registers are not reset (cleared) during operation of the system.

Late within the execution of the current microinstruction in ROS data register 65, and after the address of the next microinstruction has been loaded into ROS address register 63, clocking signal PSTROB+ is used to clock return address stack 70 by clocking shift registers 70A to 70L and is also used to clock (load) ROS address history register 66. Therefore, simultaneously with the clocking of the return address into return address stack 70 by signal PSTROB+ transitioning from the binary ZERO to the binary ONE state, the address of the next location in read only storage 24 that is to be fetched using the next address found in ROS address register 63 will be loaded into ROS address history register 66 for use as the current address early in the execution of the next microinstruction.

Early in the execution of the current microinstruction, which is prior to clocking address S from ROS address register 63 into ROS address history register 66, ROS address history register 66 contains the address of the microinstruction in the ROS data register 65 which specified the jump to location S (in Figure 4 this is microinstruction R). Thus, early in the execution of microinstruction R, ROS address history register 66 would contain the current address R and the output of incrementer 64 would be the address of $R + 1$ which appears at the A input of the return multiplexer 61. As seen above, the select signal LDRSAR+ during the JUMP S microinstruction is a binary ONE causing the A output of return multiplexer 61 to be enabled onto the Q output thereof. Thus, signals STKI00 to STKI11 which appear at the shift right serial inputs (RI) of shift register 70A through 70L will correspond to the binary address $R + 1$. Thus, when clocking signal PSTROB+ occurs late during the execution of the JUMP S microinstruction, the address $R + 1$ will be pushed into return address stack 70 by shifting shift register 70A to 70L right one position.

Having described the functioning of the logic of Figure 5 with respect to the JUMP S microinstruction which also inhibits interrupts and does a PUSH microcommand, the execution of the second and third microinstructions illustrated in ROS data register 65 will now be discussed. Both the second and third microinstructions illustrated in Figure 5 do not contain a push operation in the special control field. Therefore, during the execution of both these instructions, the output of NAND gate 83 which is used to decode the PUSH microcommand will be a binary ONE indicating that a push operation is not encoded within the microinstruction.

Turning now to the second microinstruction, we see that the hardware interrupt inhibit bit in bit position 34 is a binary ZERO indicating that hardware interrupts are permitted. This binary ZERO appears as signal RDDT34+ at the middle input of AND gate 84 and causes the output thereof, signal DISHIP+, to be a binary ZERO thereby enabling the inputs to hardware interrupt priet encoder 54. If there are no hardware interrupt signals in the binary ZERO state and no hardware error signals in the binary ZERO state and a clear condition does not exist, all inputs I0 to I10 of encoder 54 will be in the binary ONE state and therefore the output active signal at the P output of encoder 54, signal LRSAR+, will be a binary ONE.

With signal LDRSAR+ being a binary ONE and signal PUSHOP- being a binary ONE indicating that a push operation is not specified, the output of NAND gate 90, signal PUSHNT+, will be a binary ZERO indicating that a push operation is not to be performed because neither a PUSH microcommand is present, nor is a hardware interrupt pending. With one input of AND gate 92 being a binary ZERO, the output thereof, signal PUSNOT+, will be a binary ZERO at one input of OR gate 94. The other input of OR gate 94, signal CLEARX+, will be a binary ZERO if a clear operation is not being performed. Therefore, the output of OR gate 94, signal PUSNEN+, will be a binary ZERO.

Because a pop operation is not specified in the ROS address field of the second microinstruction, the output of inverter 88, signal POPOP+, will be a binary ZERO indicating that a pop operation has not been specified. Signal POPOP+ being a binary ZERO at one input of AND gate 93 will cause the output thereof, signal POPNOT+, to be a binary ZERO. With both inputs to OR gate 95 being binary ZEROS, the output thereof, signal POPNEN+, will be a binary ZERO. Because both shift mode signals, S0 and S1, at the inputs of shift register 70A through 70L are binary ZEROS, the shift registers will not shift nor parallel load when they are clocked by signal PSTROB+ transitioning from the binary ZERO to the binary ONE state and therefore neither a push nor pop nor load operation will be performed during the execution of this microinstruction.

During the execution of this microinstruction, the A inputs to address multiplexer 2 62 will be gated onto the Q output thereof because the selection signal, LDRSAR+, is a binary ONE as seen above. Therefore, the output of address multiplexer 1 60 will be loaded into ROS address register 63. Late in the execution of the current microinstruction when the output of return multiplexer 61 is strobed by signal PSTROB+

transitioning from the binary ZERO to the binary ONE state, the output of ROS address register 63 which contains the next address is loaded into ROS address history register 66 and thereafter incremented by one by incrementer 64. Thereafter, when the next microinstruction is executed as the current microinstruction, ROS address history register 66 will contain the current address and the current address + 1 as generated by incrementer 64 will be at the A input of return multiplexer 61.

If, during the execution of this second microinstruction in which hardware interrupts are enabled by bit 34 being a binary ZERO, a hardware interrupt occurs, the output active signal, LDRSAR+, from the P output of encoder 54 will be a binary ZERO indicating that a hardware interrupt or error condition or clear condition has occurred and the B input of address multiplexer 2 62 will be enabled onto the Q output thereof. The B input to address multiplexer 2 62 is comprised of the binary encoded bits Q1 to Q8 from encoder 54 along with eight leading binary ZEROS in the eight most significant bits such that the Q output of address multiplexer 2 62 will be a 12-bit binary address corresponding to the starting location of a hardware interrupt service routine which is preprogrammed to handle a particular hardware interrupt or hardware error or clear condition.

The output of address multiplexer 2 62 is input into ROS address register 63 and from there output to read only storage 24 so that the first word of the hardware interrupt service routine may be fetched and executed. The address of the first word of the hardware interrupt service routine is also loaded into address history register 66 from ROS address register 63 during the execution of the second microinstruction in ROS data register 65 when signal PSTROB+ clocks (transitions from the binary ZERO to the binary ONE state). At the same time that the address is clocked from ROS address register 63 into ROS address history register 66, the output of return multiplexer 61 is pushed onto return address stack 70.

The output of return multiplexer 61 will be the output of address multiplexer 1 60 which will be the nominal next microinstruction address which appears at the B input of return multiplexer 61. As seen above, the B input of return multiplexer 61 is placed on the Q output thereof because select signal LDRSAR+ is a binary ZERO whenever there is an active output from encoder 54. Shift register 70A to 70L will do a push operation (right shift) because signal PUSNEN+ from OR 94 is a binary ONE and signal POPNEN+ from OR gate 95 is a binary ZERO. Signal PUSNEN+ is a binary ONE and POPNEN+ is a binary ZERO because signal LDRSAR+ is a binary ZERO, signal CLEARX+ is a binary ZERO, signal PUSHOP- is a binary ONE and signal POPOP+ is a binary ZERO at the inputs to gates 90 to 95, the operation of which is described above.

During the execution of the third microinstruction illustrated in ROS data register 65 in Figure 5, a pop operation is performed as specified in the ROS address field. The pop operation indicated above is decoded by OR gate 86, the output of which, signal POPOP-, will be a binary ZERO indicating that a pop operation is to be performed. Signal POPOP- being a binary ZERO will enable the output of driver 60B within address multiplexer 1 60 such that the A input of address multiplexer 2 will be the signals STK000 to STK011 which are the twelve bits which correspond to the QA outputs of shift registers 70A to 70L. These twelve signals correspond to the top address within the return address stack 70. The outputs of driver 60A are disabled by signal JUMP- being a binary ONE. Thus, the output of address multiplexer 1 60 will be the top address in return address stack 70 which corresponds to the contents of register 71 in Figure 4.

No hardware interrupt occurs during the execution of this POP microinstruction. The A inputs of address multiplexer 2 62 will be selected to be output as the Q output and thereby loaded into ROS address registers 63 which specifies the address of the next instruction to be fetched from read only store 24. As in this case of a POP instruction without the occurrence of a hardware interrupt, the next instruction read from read only store 24 will be the instruction which was addressed by the top location within return address stack 70. If no hardware interrupt occurs during the execution of this POP microinstruction, signal LDRSAR+ output by encoder 54 will be a binary ONE indicating that none of its outputs are active. Because a push operation was not specified in the special control field, signal PUSHOP- from NAND gate 83 will be a binary ONE and therefore the output of NAND gate 90, signal PUSHNT+, will be a binary ZERO indicating that no push operation is to be performed. If a clear operation is not taking place as indicated by signal CLEARX+ being a binary ZERO, the output of OR gate 94, signal PUSNEN+, will be a binary ZERO and the output of OR gate 95, signal POPNEN+, will be binary ONE indicating that a left shift (pop) operation is to be performed by shift registers 70A to 70L. This will pop the return address stack 70 one location when signal PUSTROB+ transitions from the binary ZERO to the binary ONE state.

When this left shift operation of shift registers 70A to 70L occurs, the bits at the shift left serial input (LI) of the twelve 4-bit shift registers are loaded into the shift registers. As can be seen in Figure 5, shift registers 70A to 70K contain a binary ZERO at this input, and shift register 70L contains a binary ONE at this input thus resulting in the bottom location within return address stack (register 74) in Figure 4 being loaded with the hexadecimal value 001. This 001 value is used as an address within read only storage 24 of

a routine that is microprogrammed to handle the case of overpopping the stack which will occur if this 001 hexadecimal is ever popped off the top of the stack.

This same ROS address of 001 hexadecimal is simultaneously loaded into registers 71 to 74 of return address stack 70 when a clear operation is performed by parallel loading all four inputs of shift registers 70A to 70K with binary ZEROS by having a binary ZERO at the PIA, PIB, PIC and PID inputs of registers 70A to 70K and binary ONES at the corresponding inputs of register 70L and doing a parallel load by having signals PUSNEN+ and POPNEN+ equal to binary ZEROS when these registers are clocked by PSTROB+ transitioning from the binary ZERO to the binary ONE state during a clear operation. This is accomplished because signal CLEARX+ is a binary ONE during a clear operation forcing the outputs of OR gates 94 and 95 to be binary ONES thereby setting the shift mode select signals S0 and S1 both equal to binary ONES.

As discussed above with respect to Figure 4, if a hardware interrupt occurs during the execution of a microinstruction which contains a pop operation as does the third microinstruction illustrated in Figure 5, the push operation associated with the hardware interrupt cancels the pop operation of the microinstruction and the return address stack is neither pushed nor popped. The method by which this is accomplished will now be discussed with respect to Figure 5.

If the third microinstruction which contains the pop operation in the ROS address field also contains a hardware interrupt inhibit bit of binary ZERO, a hardware interrupt can occur during the execution of a POP microinstruction. As seen above, during the execution of a microinstruction which contains the POP microcommand, the output of driver 60B will be enabled such that the output of address multiplexer 1 60 will be the return address which is contained in the top of return address stack 70. The output of address multiplexer 1 60 is presented to the A input of address multiplexer 2 62 and the B input of return multiplexer 61. As seen above, if a hardware interrupt occurs, signal LDRSAR+ from encoder 54 is a binary ZERO selecting the B input of both address multiplexer 2 62 and return multiplexer 61. Thus, the output of address multiplexer 2 will be the vectored address as determined by encoder 54 at which is located the hardware interrupt service routine. The output of return multiplexer 61 will be the output of driver 60B which is the output of the top of return address stack 70.

Thus, the ROS address register 63 will be loaded with the starting address of the hardware interrupt service routine associated with the highest priority interrupt pending and that location will be fetched from read only storage 24 to begin the execution of the hardware interrupt service routine. With signal LDRSAR+ being a binary ZERO, the output of NAND gate 90, signal PUSHNT+, will be a binary ONE indicating that a push operation should be performed on return address stack 70. Because a POP microcommand is encoded in ROS address field, the output of OR gate 86, signal POPOP-, will be a binary ZERO thereby causing the output of inverter 88, signal POPOP+, to be a binary ONE indicating that a pop operation should be performed on return address stack 70. With both inputs to NAND gate 91 being binary ONES, the output thereof, signal PUSPOP-, will be a binary ZERO indicating that both a push and pop operation is to be performed simultaneously on return address stack 70.

Signal PUSPOP- being a binary ZERO will disable AND gates 92 and 93 and cause signals PUSNOT+ and POPNOT+ to be binary ZEROS indicating that neither a push nor pop operation is to be performed. Because a clear operation is not being performed, signal CLEARX+ will be a binary ZERO and therefore the output of OR gates 94 and 95 will be binary ZEROS indicating that neither a shift nor load operation is to be performed on shift registers 70A through 70L. Therefore, a POP microcommand within the microinstruction cancels the implicit push operation of the hardware interrupt resulting in neither the pushing nor the popping nor the parallel loading of return address stack 70.

From the above discussion it can be appreciated that NAND gate 91 is used to detect the simultaneous occurrence of a push and pop operation and to inhibit the simultaneous pushing and popping of return address stack 70 when this condition occurs. This condition can occur when a POP microcommand is encoded within a microinstruction and a hardware interrupt occurs the combination of which prevents the movement of the stack in either direction. From the above discussion, it can also be appreciated how the stack is loaded at the bottom register, each time it is popped, with an address of a routine which is used to detect the overpopping of the stack and how the stack is initialized during a clear operation so that each register within the stack contains this address of the overpopping service routine.

Modifications of this embodiment will be obvious. For example, if hardware interrupts are not inhibited during a hardware interrupt service routine, the stacking mechanism can be used to nest hardware interrupts. In addition, a stack mechanism comprised of registers comprised of flip-flops as described in U.S. Patent No. 3,909,797, could be used in lieu of the present shift registers, provided that the logic of the present system which is used to detect an attempt to do a simultaneous push and pop operation is utilized so that neither a push nor pop operation is done.

It should be noted that the ROS address field to the subroutine can be either a conditional or

unconditional branch. This is especially advantageous with the system described since the next address is continually provided by the ROS address field of microinstructions from read only storage 24. Nevertheless, it would be equally applicable in those addressing methods in which a sequence counter is used. Both addressing methods require the utilization of the PUSH microcommand to enable the storing of the address of the next instruction which would otherwise be executed if a branch was not made to the subroutine.

Claims

1. In a data processor controlled by a microprogrammed control store, wherein said control store executes different sequences of microinstructions held in addressable locations of a storage unit thereof, and wherein said processor is provided with a pushdown stack for holding a plurality of addresses of said storage unit; apparatus for controlling said data processor to enable intermixed subroutine branching and entry into interrupt service routines from said sequences of microinstructions, comprising:
 - means controlled while executing a first microinstruction which denotes a transfer to a specified subroutine from the routine/subroutine in execution for incrementing the address of said first microinstruction and pushing such incremented address onto the top of said stack, and being characterized by:
 - means controlled while executing a second microinstruction during which an interrupt occurs for pushing onto the top of said stack the next address being developed during execution of said second microinstruction and for transferring operation to a microinstruction service routine whose first address is determined by the nature of said interrupt, and
 - means controlled by the last microinstruction in each subroutine and said service routine to pop up said stack, and use the address received from the top of said stack as a return address to re-enter the routine/subroutine from which said subroutine or service routine was entered.
2. The apparatus as claimed in claim 1 further including means enabled by the concurrent execution of said last microinstruction in a subroutine and the occurrence of an interrupt for inhibiting the popping up of said stack.
3. The apparatus as claimed in claim 1 further including means responsive to said first microinstruction for inhibiting said apparatus from responding to certain interrupts.

Revendications

1. Dans un processeur de données commandé par une mémoire de commande microprogrammée, où ladite mémoire de commande exécute différentes séquences de micro-instructions maintenues à des emplacements adressables d'une unité de mémoire de celle-ci, et où ledit processeur est pourvu d'une pile inversée pour maintenir une pluralité d'adresses de ladite unité de mémoire; un dispositif pour commander ledit processeur de données pour la validation d'un branchement à des sous-programmes mélangés et d'une entrée dans des programmes de traitement d'interruptions à partir desdites séquences de micro-instructions, comprenant:
 - un moyen commandé pendant l'exécution d'une première micro-instruction qui indique un transfert jusqu'à un sous-programme spécifié à partir du programme/sous-programme en cours d'exécution pour incrémenter l'adresse de ladite première micro-instruction et pousser cette adresse incrémentée jusqu'en haut de ladite pile, et étant caractérisé par:
 - un moyen commandé pendant l'exécution d'une deuxième micro-instruction pendant laquelle se produit une interruption pour pousser en haut de ladite pile l'adresse suivante développée pendant l'exécution de ladite deuxième micro-instruction et pour transférer le déroulement jusqu'à un programme de traitement de micro-instructions dont la première adresse est déterminée par la nature de ladite interruption, et
 - un moyen commandé par la dernière micro-instruction de chaque sous-programme et dudit programme de traitement pour remonter ladite pile et mettre en oeuvre l'adresse reçue du haut de ladite pile comme adresse de retour pour relancer le programme/sous-programme à partir duquel une entrée a été faite dans ledit sous-programme ou ledit programme de traitement.
2. Dispositif selon la revendication 1, incluant en outre un moyen validé par l'exécution en cours de ladite dernière micro-instruction d'un sous-programme et l'apparition d'une interruption pour interdire de remonter ladite pile.



unconditional branch. This is especially advantageous with the system described since the next address is continually provided by the ROS address field of microinstructions from read only storage 24. Nevertheless, it would be equally applicable in those addressing methods in which a sequence counter is used. Both addressing methods require the utilization of the PUSH microcommand to enable the storing of the address of the next instruction which would otherwise be executed if a branch was not made to the subroutine.

Claims

1. In a data processor controlled by a microprogrammed control store, wherein said control store executes different sequences of microinstructions held in addressable locations of a storage unit thereof, and wherein said processor is provided with a pushdown stack for holding a plurality of addresses of said storage unit; apparatus for controlling said data processor to enable intermixed subroutine branching and entry into interrupt service routines from said sequences of microinstructions, comprising:
 - means controlled while executing a first microinstruction which denotes a transfer to a specified subroutine from the routine.subroutine in execution for incrementing the address of said first microinstruction and pushing such incremented address onto the top of said stack, and being characterized by:
 - means controlled while executing a second microinstruction during which an interrupt occurs for pushing onto the top of said stack the next address being developed during execution of said second microinstruction and for transferring operation to a microinstruction service routine whose first address is determined by the nature of said interrupt, and
 - means controlled by the last microinstruction in each subroutine and said service routine to pop up said stack, and use the address received from the top of said stack as a return address to re-enter the routine/subroutine from which said subroutine or service routine was entered.
2. The apparatus as claimed in claim 1 further including means enabled by the concurrent execution of said last microinstruction in a subroutine and the occurrence of an interrupt for inhibiting the popping up of said stack.
3. The apparatus as claimed in claim 1 further including means responsive to said first microinstruction for inhibiting said apparatus from responding to certain interrupts.

Revendications

1. Dans un processeur de données commandé par une mémoire de commande microprogrammée, où ladite mémoire de commande exécute différentes séquences de micro-instructions maintenues à des emplacements adressables d'une unité de mémoire de celle-ci, et où ledit processeur est pourvu d'une pile inversée pour maintenir une pluralité d'adresses de ladite unité de mémoire; un dispositif pour commander ledit processeur de données pour la validation d'un branchement à des sous-programmes mélangés et d'une entrée dans des programmes de traitement d'interruptions à partir desdites séquences de micro-instructions, comprenant:
 - un moyen commandé pendant l'exécution d'une première micro-instruction qui indique un transfert jusqu'à un sous-programme spécifié à partir du programme.sous-programme en cours d'exécution pour incrémenter l'adresse de ladite première micro-instruction et pousser cette adresse incrémentée jusqu'en haut de ladite pile, et étant caractérisé par:
 - un moyen commandé pendant l'exécution d'une deuxième micro-instruction pendant laquelle se produit une interruption pour pousser en haut de ladite pile l'adresse suivante développée pendant l'exécution de ladite deuxième micro-instruction et pour transférer le déroulement jusqu'à un programme de traitement de micro-instructions dont la première adresse est déterminée par la nature de ladite interruption, et
 - un moyen commandé par la dernière micro-instruction de chaque sous-programme et dudit programme de traitement pour remonter ladite pile et mettre en oeuvre l'adresse reçue du haut de ladite pile comme adresse de retour pour relancer le programme sous-programme à partir duquel une entrée a été faite dans ledit sous-programme ou ledit programme de traitement.
2. Dispositif selon la revendication 1, incluant en outre un moyen validé par l'exécution en cours de ladite dernière micro-instruction d'un sous-programme et l'apparition d'une interruption pour interdire de remonter ladite pile.



3. Dispositif selon la revendication 1, incluant en outre un moyen réagissant à ladite première microinstruction pour interdire audit dispositif de réagir à certaines interruptions.

Patentansprüche

5

1. In einem Datenprozessor, der durch einen Steuerspeicher mit Mikroprogramm gesteuert wird, wobei der Steuerspeicher unterschiedliche Mikrobefehlsabfolgen ausführt, die an adressierbaren Orten einer Speichereinheit davon gehalten werden, und wobei der Prozessor mit einem Rückstellstapel zum Halten einer Vielzahl von Adressen der Speichereinheit versehen ist, ein Gerät zum Steuern des Datenprozessors zum Ermöglichen eines dazwischengeschobenen Unterprogramms, das verzweigt, und zum Eintreten in Eingriffs-Service-Programme von Mikrobefehlsabfolgen, das aufweist:

10

- eine Einrichtung, die gesteuert wird, während sie einen ersten Mikrobefehl ausführt, der einen Übergang zu einem bestimmten Unterprogramm von dem ausgeführten Programm/Unterprogramm bezeichnet, zum Inkrementieren der Adresse des ersten Mikrobefehls und zum Rückstellen einer derartigen inkrementierten Adresse an die Spitze des Stapels, und

15

gekennzeichnet ist durch:

- eine Einrichtung, die gesteuert wird, während sie einen zweiten Mikrobefehl ansführt, während dem eine Unterbrechung auftritt zum Rückstellen der nächsten Adresse, die beim Ausführen des zweiten Mikrobefehls entwickelt worden ist, an die Spitze des Stapels und zum Übertragen einer Operation zu einem Mikrobefehls-Service-Programm, dessen erste Adresse durch die Natur der Unterbrechung bestimmt ist, und
- eine Einrichtung, die durch den letzten Mikrobefehl in jedem Unterprogramm und dem Service-Programm gesteuert wird, zum schnellen Raufbringen des Stapels und durch die Verwendung der Adresse, die von der Spitze des Stapels als eine Rückkehradresse empfangen ist, zum Wiedereintreten in das Programm/Unterprogramm, von dem aus in das Unterprogramm oder das Service-Programm eingetreten wurde.

20

2. Gerät nach Anspruch 1, das weiterhin eine Einrichtung enthält, die durch die gleichzeitige Ausführung des letzten Mikrobefehls in einem Unterprogramm und dem Auftreten einer Unterbrechung das schnelle Raufbringen des Stapels verhindern kann.

3. Gerät nach Anspruch 1, das weiterhin eine Einrichtung enthält, die auf den ersten Mikrobefehl anspricht zum Abhalten des Geräts von einem Ansprechen auf bestimmte Unterbrechungen.

35

40

45

50

55