(12) UK Patent Application (19) GB (11) 2 190 521 (13) A

(43) Application published 18 Nov 1987

(21) Application No 8628175

(22) Date of filing 25 Nov 1986

(30) Priority data

(31) 863878 (32) 16 May 1986 (33) US

(71) Applicant
Intel Corporation,

(Incorporated in USA-California),

3065 Bowers Avenue, Santa Clara, California 95051, United States of America

(72) and (74) continued overleaf

(51) INT CL^4
G06F 9/30

(52) Domestic classification (Edition I)
G4A FN PB

(56) Documents cited

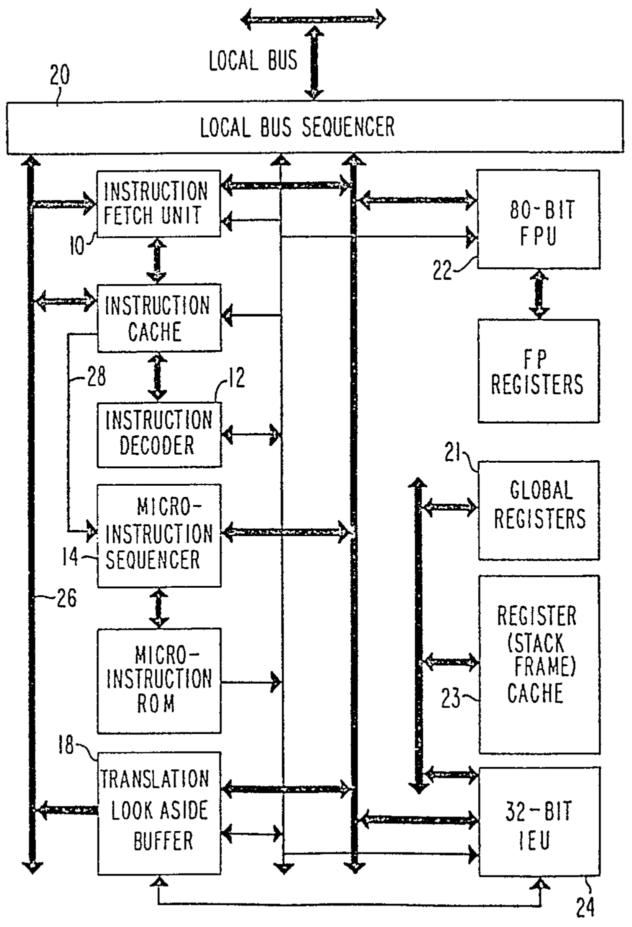
EP A1 0075633	US 4530049
EP A1 0075632	US 4268903

(58) Field of search
G4A
Selected US specifications from IPC sub-class G06F

(54) Stack frame cache on a microprocessor chip

(57) A stack frame cache on a microprocessor chip and a control mechanism therefor. A plurality of global registers 21 are provided on the microprocessor chip. One of the global registers is a frame pointer register containing the current frame pointer, and the remainder of the global registers are available to a current process as general registers. A plurality of floating point registers are also provided for use by the current process in execution of floating point arithmetic operations by processor 22. A register set pool 23 forming an on-chip cache and made up of a plurality of register sets is provided, each register set being comprised of a number of local registers. When a call instruction is decoded, a register set of local registers from the register set pool is allocated to the called procedure, and the frame pointer register is initialized. When a return instruction is decoded, the register set is freed for allocation to another procedure called by a subsequent call instruction. If the register set pool is depleted a register set associated with a previous procedure is saved in the main memory, and that register set is allocated to the current procedure. The local registers in a register set associated with a procedure contain linkage information including a pointer to the previous frame and an instruction pointer, thus enabling most call and return instructions to execute without needing any references to off-chip memory.

FIG. 1



The drawing(s) originally filed was (were) informal and the print here reproduced is taken from a later filed formal copy.

(72) Inventors
Glenford J. Myers,
Konrad Lai,
Michael T. Imel,
Glenn Hinton,
Robert Riches

(74) Agent and/or Address for Service
Potts Kerr & Co., 15 Hamilton Square, Birkenhead,
Merseyside L41 6BR

ORIGINAL

2190521

FIG. 1

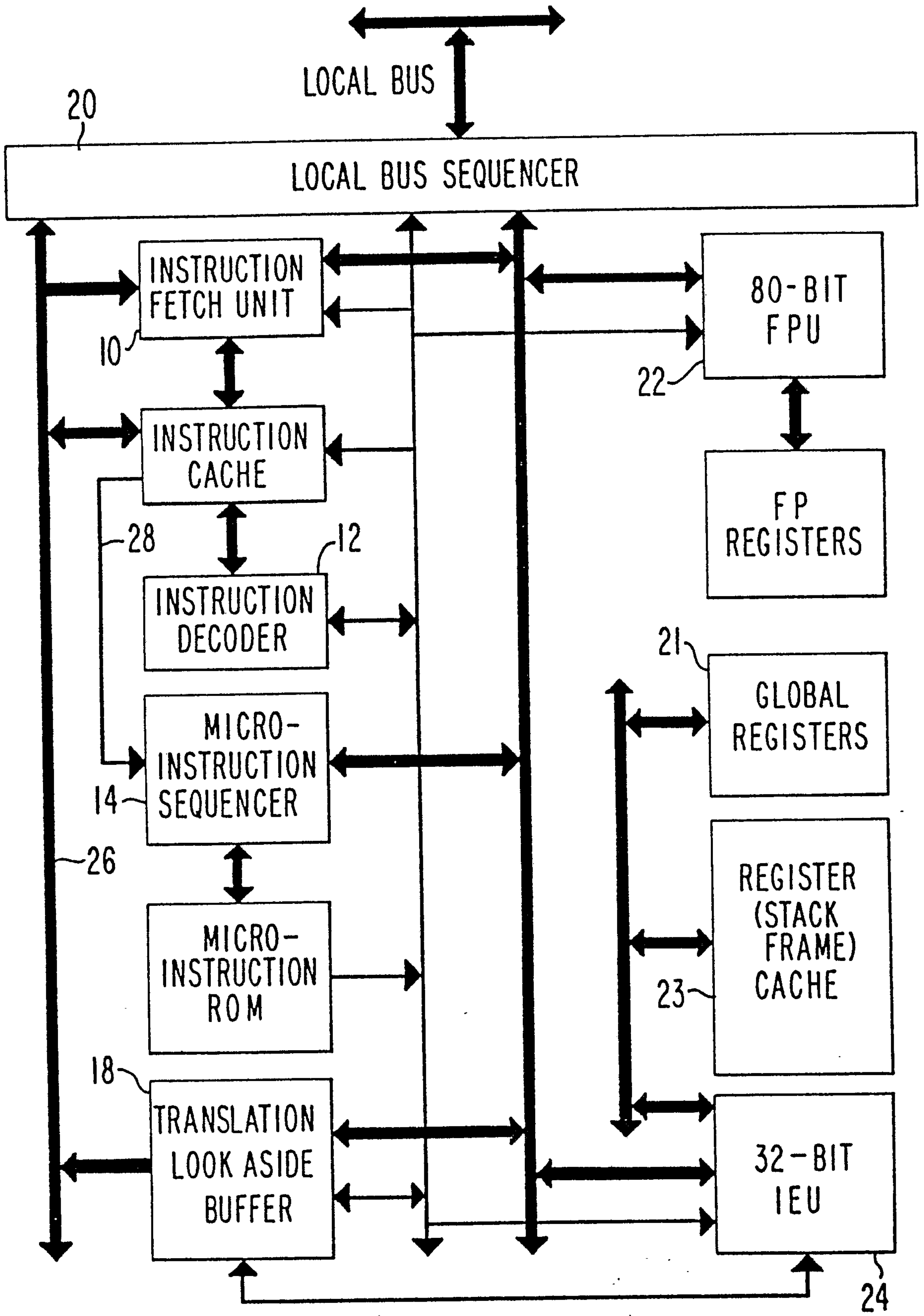


FIG. 2

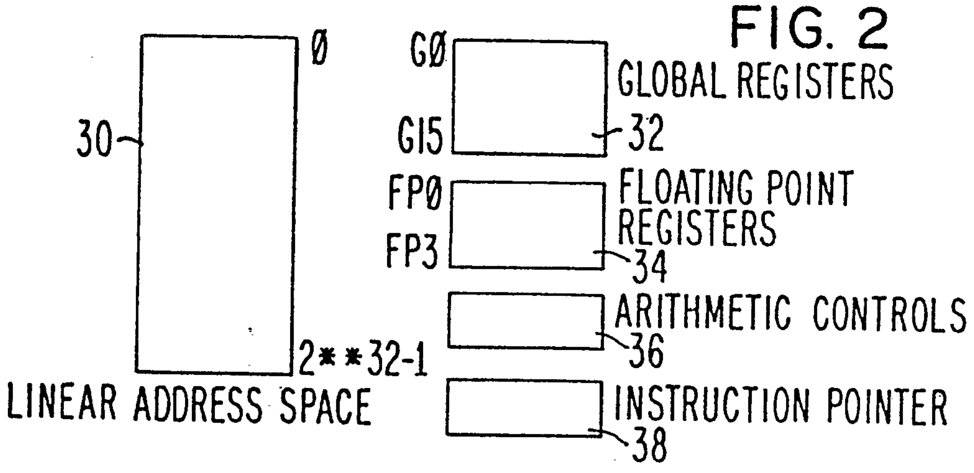


FIG. 3

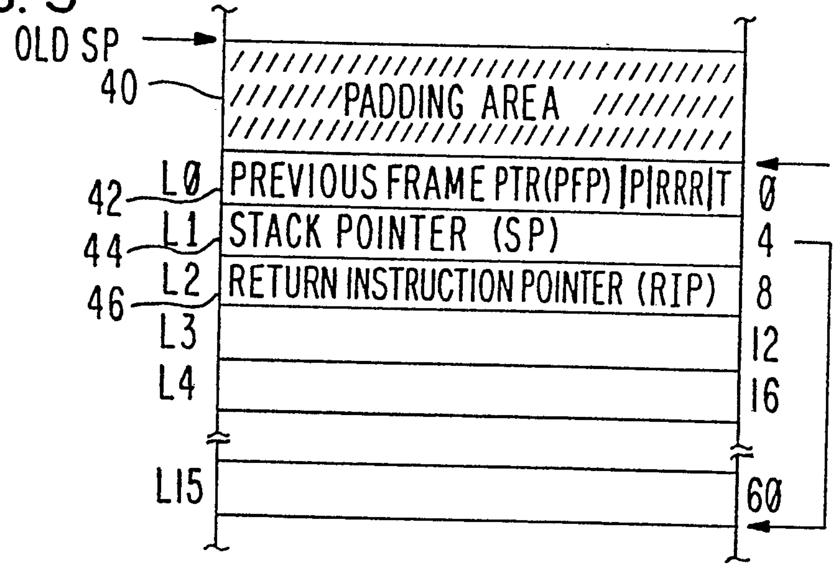


FIG. 4

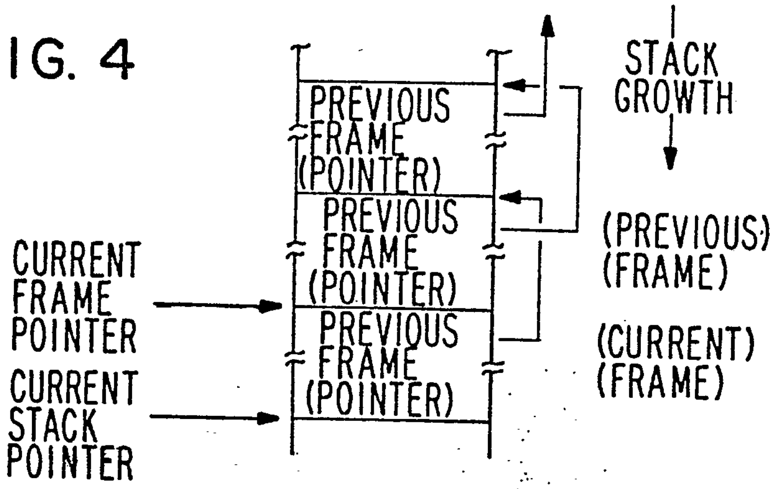
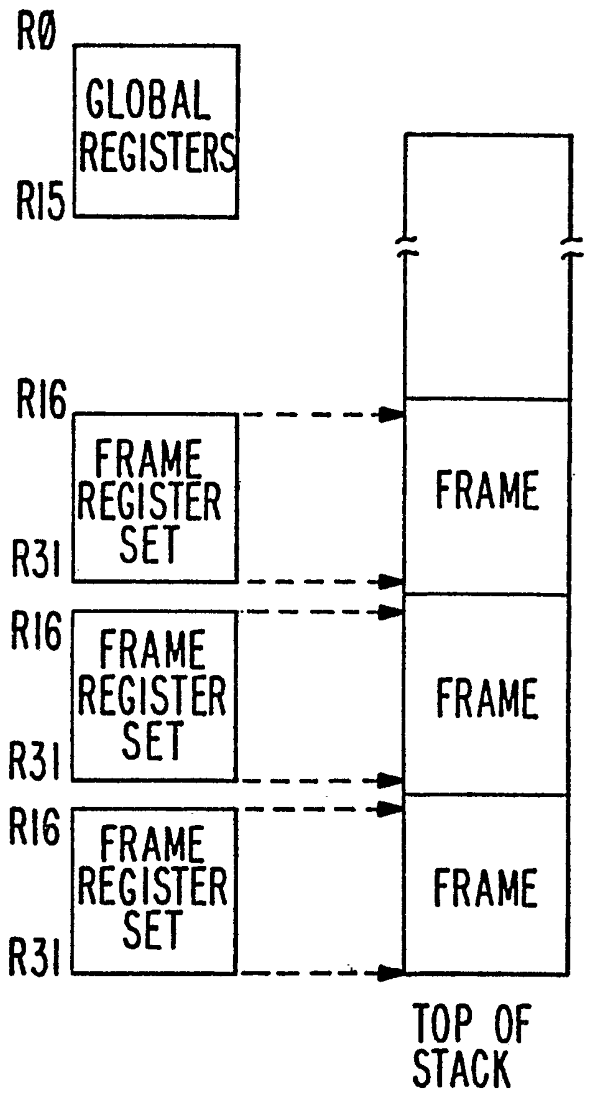


FIG. 5



RØ
GLOBAL REGISTERS
R15
R16
FRAME REGISTER SET
R31
R16
FRAME REGISTER SET
R31
R16
FRAME REGISTER SET
R31
FRAME
FRAME
FRAME
TOP OF STACK

SPECIFICATION

Stack frame cache on a microprocessor chip

Technical field

The present invention relates to data processing systems, and more particularly to apparatus for minimizing main memory references initiated during execution of call/return instructions.

Background art

Rapid advances in VLSI technology and design techniques have enabled microcomputers to approach the performance and sophistication of a super minicomputer. As processors become faster, the traffic between the processor and off-chip main memory increases causing a performance bottleneck. In prior systems this bottleneck has been lessened by using a local on-chip memory (called a cache) to store frequently used memory data. If data required by the processor is in the cache, an off-chip memory reference is avoided since the data can be fetched directly from the cache. Further reductions in memory traffic could be achieved if the cache design were expanded to include instruction fetches. For example if informatioin relating to call and return instructions were available locally on the chip, call and return instructions could execute without references to the off-chip memory. The resulting decreased memory bus traffic would also reduce the probability that a load or store instruction will have to wait for the memory bus.

It is therefore an object of the present invention to provide an apparatus for minimizing main memory references occurring during execution of call/return instructions.

Brief description of the invention

Briefly, the above object is accomplished in accordance with the invention by providing a plurality of global registers on the microprocessor chip. One of the global registers is a frame pointer register containing the current frame pointer, and the remainder of the global registers are available to a current process as general registers. A stack frame cache mechanism is provided comprised of a register set pool made up of a plurality of register sets, each register set being comprised of a number of local registers on the chip. When a call instruction is decoded, a register set from the register set pool is allocated to the called procedure, and the frame pointer register is initialized. When a return instruction is decoded, the register set is freed for allocation to another procedure called by a subsequent call instruction. If the register set pool is depleted, the contents of a register set associated with a previous procedure are saved in the main memory, and that register set is allocated to the current procedure.

In accordance with an aspect of the invention, the local registers of a register set associated with a procedure contain linkage information including a pointer to the previous frame and an instruction pointer, thus enabling most call and return instructions to execute without needing any references to off-chip memory.

The invention has the advantage that it significantly reduces the saving and restoring of registers that must be done when crossing subroutine boundaries.

The invention also has the advantage that since the local register sets are mapped into the stack frames, the linkage information that normally appears in stack frames (e.g., pointer to previous frame, saved instruction pointer) is contained in the local registers. This means that most call and return instructions execute without causing any references to off-chip memory.

Description of the drawings

The foregoing and other objects, features, and advantages of the invention will be apparent from the following more particular description of the preferred embodiments of the invention as illustrated in the accompanying drawings, wherein:

Figure 1 is a functional block diagram illustrating each of the major components of the microprocessor in which the invention is embodied;

Figure 2 is a block diagram of an execution environment when executing an instruction on the system shown in Figure 1;

Figure 3 is a diagram of the stack frame structure within the current linear address space of the execution environment shown in Figure 2;

Figure 4 is a diagram of the call stack structure within the current linear address space of the execution environment shown in Figure 2; and,

Figure 5 illustrates the mapping of the microprocessor's register sets into the program's stack in memory.

Description

Referring now to Figure 1, the microprocessor is logically subdivided into seven major units: the Instruction Fetch Unit (10), the Instruction Decoder (12), the Microinstruction Sequencer (14), Translation Lookaside Buffer (18), the Floating Point Unit (22), the Local Bus Sequencer (20), and the Integer Execution Unit (24).

Communication paths between all of these units include a 32-bit data bus, a 29-bit microinstruction bus (26), and microinstruction-valid signal (28). The microinstruction bus controls and synchronizes the activities of the autonomous units. Each of the units is described briefly below.

The instruction Decoder (ID) decodes and controls instruction (macrocode) execution. The ID decodes instructions, performs operand addressing and fetching, handles branch instructions (i.e., instruction pointer manipulation), and either emits execution microinstructions (for simple instructions) or starts microprogram flows (for complex instruction).

The Instruction Fetch Unit (IFU) fetches, prefetches, and caches instructions from memory for use by the ID. The IFU also maintains six instruction pointers that track instructions through the pipeline. The IFU caches the most recently used blocks of instructions and keeps the instruction decoder supplied with a stream of instructions. It also contains the instruction pointers and operand reduction logic controlled by the ID.

The Microinstruction Sequencer (MIS) sequences microcode flows to handle chip initialization, macroinstructions that are too complex to handle directly, and exception and interrupt conditions.

The MIS contains a 3K by 42-bit microcode ROM and sequencing logic for microcode flows. The functions that the MIS perform include: fetch the next microinstruction, microprogram branching, handle exception conditions, maintain a scoreboard on the register file, and in conjunction with the ID, detect macroinstruction-boundary and trace events.

The Integer Execution Unit (IEU) executes most of the microinstructions issued by the ID and the MIS. It contains the registers visible to the programmer, the scratch registers used by microcode, the ALU, barrel shifter, and the logic needed to execute its instructions. The IEU contains one-hundred twelve 32-bit registers, a 32-bit ALU, and a 32-bit barrel shifter. It features an ALU bypass path that allows ALU operations to be executed at the rate of one per cycle. It also contains a single-port register file that can be accessed twice in one cycle such that the result from the previous operation can be stored in the same cycle as a new operand is being fetched for the current operation.

The Floating Point Unit (FPU) contains the logic needed to perform floating point operations, and integer multiply and divide. The FPU contains four floating point registers, several temporary registers, a 68-bit shifter that can shift up to 16 bits in either direction, a 69-bit mantissa adder, a significant bit finder, a mantissa ROM, two internal 68-bit data paths, and a separate exponent data path that includes its own 16-bit adder and registers. It executes integer multiply and divide, and all floating point operations, including the cordic algorithms for the transcendental instructions.

The Translation Lookaside Buffer (TLB) performs the address translation needed to implement virtual memory mechanisms. The TLB performs address translation and memory protection using an associative table of storage descriptors and page table entries. It contains a 48-entry address cache, a six-bit address adder, and memory protection checking hardware. Each entry in the address cache contains 27 CAM bits and 38 RAM bits. The TLB supports several address translation mechanisms to allow the user to choose the type of memory protection from a variety of conventional mechanisms (paging or segmentation).

The Local Bus Sequencer pipelines and sequences external bus accesses. The local bus sequencer contains the interface hardware to the external local bus, manages the bus protocol, and recognizes external events (e.g., interrupts, initialization). It contains an outgoing 33-bit wide address and data FIFO, an incoming 33-bit data FIFO, and a sequencer. The outbound FIFO allows up to 3 requests to be queued in the local bus sequencer so that the rest of the processor can proceed with execution, independent of the memory access latency. The inbound FIFO buffers read data returning from external memory until a free cycle is available to transfer the data to its destination.

A plurality of global registers (21) are provided. One of the global registers is a frame pointer register containing the current frame pointer, and the remainder of the global registers are available to a current process as general registers. A register (stack frame) cache (23) is provided comprised of a register set pool made up of a plurality of register sets, each register set being comprised of a number of local registers. When a call instruction is decoded, a register set from the register set pool is allocated to the called procedure, and the frame pointer register is initialized. When a return instruction is decoded, the register set is freed for allocation to another procedure called by a subsequent call instruction. If the register set pool is depleted, the contents of a register set associated with a previous procedure are saved in the main memory, and that register set is allocated to the current procedure. The local registers of a register set associated with a procedure contain linkage information including a pointer to the previous frame and an instruction pointer, thus enabling most call and return instructions to execute without needing any references to off-chip memory.

Instruction set

A process sees a flat linear address space, addressed with 32-b ordinals, out of which it allocates data, instruction, and stack space. A call instruction creates a new stack frame (activation record) on a sequentially allocated stack.

The instruction set of the microprocessor is similar in design to those of RISC (reduced instruction-set computer) machines. All instructions are 32-bits in length and must be aligned on word boundaries, and only load, store, and branching instructions reference memory (all others reference registers).

Refer to Figure 2 which shows the environment when executing. The execution environment consists of a 2**32 byte linear address space (30) and thirty six registers. Of the thirty six registers, 16 are 32-bit global registers (32), sixteen are 32-bit local registers (34), and the remaining four are 80-bit floating-point registers (36). The local registers are associated with a mechanism known as the stack-frame cache. When a procedure is called, a new set of local registers are allocated from a pool of registers on-chip, and are feed by a procedure return. The present embodiment of the invention provides four sets (64) of local registers on-chip, but this number is transparent to the programmer.

The register model consists of 16 global registers and 4 floating-point registers that are preserved across procedure boundaries, and multiple sets of 16 local (or frame) registers that are associatively mapped into each stack frame.

At any instant, an instruction can address thirty six of these registers as follows:

Register type	Register name
Global register	G0 ... G15
Floating-point register (floating-point operand)	FP0 .. FP13
Local register	L0 ... L15

At any point in time, one can address thirty-two 32-bit registers, and four 80-bit floating-point registers (the 32 registers can also be used to hold floating-point values). Of the 32 registers, 16 are global registers and 16 are local registers. The difference is that the 16 global registers are unaffected when crossing procedure boundaries (i.e., they behave like "normal" registers in other processors local registers are affected by the call and return instructions.

When a call instruction is executed, the processor allocates to the called procedure a new set of 16 local registers from an on-chip pool of four register sets. If the processor's four-set pool is depleted, the processor automatically reallocates a register set by taking one register set associated with an earlier procedure and saving the contents of that register set in memory. The contents of the earlier procedure's register set are saved in the first 16 words of the procedure's stack frame in memory. Because of this, the mechanism is named the stack frame cache. The return instruction causes the current local register set to be freed (for use by a subsequent call).

There are sixteen global registers (32) associated with a process; they are saved in the process control block when the process is not executing. Global registers are not associatively mapped into the process control block.

Of the sixteen 32-bit registers, G15 contains the current frame point (FP) and G0..G14 are general-purpose registers. The FP contains the linear address (pointer) into the current execution environment for the current (topmost) stack frame. Since stack frames are aligned to 64-byte boundaries, the low-order 6 bits of FP are ignored and always interpreted to be zero. This register is initialized on calls and restored on returns.

A reference to a register as an operand that is bigger than 32 bits uses the registers with consecutive higher register numbers.

Floating-point registers

There are four floating-point registers (34) associated with a process; they are saved in the process control block when the process is not executing. Floating-point registers are not associatively mapped into the process control block.

Floating-point numbers are stored in extended real format in the floating-point registers. Floating-point registers are accessed only as operands of floating-point instructions (but such instructions may also use the 32-bit local and global registers).

Arithmetic controls

The Arithmetic Controls (36) are used to control the arithmetic and faulting properties of the numeric instructions as well as for storing the conditions codes. When a process is suspended, the arithmetic controls information is saved in the process control block.

Instruction pointer

The Instruction Pointer (38) is a linear address (pointer) into the current linear address space to the first byte of the current instruction. Since instruction must begin on word (4-byte) boundaries, the two low-order bits of IP are ignored and assumed to be 0.

Local (or frame) registers

Refer to Figure 3. Registers L0..L15, the local registers, do not denote registers of the conventional variety; they denote the first 16 words of the current frame. Thus, register L0 is mapped into linear address FP+0 to FP+3, register Li is mapped into linear address FP=4i to FP+4i+3, and so on.

A cache of multiple stack frames is provided. There are multiple banks of high-speed registers, one bank per procedure activation. The program does not have to save and restore registers explicitly.

Stack frame

The stack frame, shown in Figure 3, is a contiguous portion of current linear address space, containing data in a stack-like fashion. There is one stack frame per activated procedure, which contains local variables, parameters, and linkage information. A call operation acquires a new stack frame; a return operation releases it. When a new frame is acquired, it is aligned on a 64-byte boundary.

The fields in the stack frame of Figure 3 are defined as follows:

Padding Area. This area (42) is used to align the FP to the next 64-byte boundary. The size of this area varies from 0 to 63 bytes. When a call operation is performed, a padding area is added to round the caller's SP to the next 64-byte boundary to form the FP for this frame. If the caller's SP is already aligned, the padding area is absent.

Frame Status (L0). The frame status (42) records the information associated with the frame, after a call, to be used on a return from the frame. The fields of a frame status are defined as follows:

Trace Enable, T (Bit 0). In a supervisor call, this bit records the trace-enable bit at the time of the call. On return, this bit is used to restore the caller's trace-enable bit in the process if the execution mode of the returning frame is supervisor.

Return Status, RRR (bits 1-3). This 3-bit field records the call mechanism used in the creation of this frame and is used to select the return mechanism to be used on return. The encodings of this field are as follows:

000	Local
001	Supervisor
010	Interrupt
011	Nonsubsystem fault
100	Subsystem
101	reserved
110	Idle/stopped interrupt
111	reserved

Prereturn Trace, P. (bit 4). On a return from a frame when the prereturn trace bit is 1, a prereturn trace event (if enabled) occurs before any actions association with the return operation is performed. This bit is initialized to zero on a call.

Previous Frame Pointer, PFP (bit 6-31). A linear, address (42) to the first byte of the previous frame. Since frames are aligned to 64-byte boundaries, only the most-significant 26 bits of the FP are saved. If the return status indicates subsystem transfer, this field contains the most-significant 26 bits of the linear address of the top-most (last)frame in this call stack before the call. Otherwise, the top-most frame is the calling frame.

During a call, the lower five bits of the frame status are initialized as follows:

0 000-	--	Local call, or supervisor call from supervisor state
0 001T	--	Supervisor call from user mode
0 010-	--	Interrupt call
0 011-	--	Nonsubsystem fault call
0 100-	--	Subsystem call
0 110-	--	Interrupt call from idle or stopped state

T is the value of the trace bit defined above. "-" indicates a reserved bit, while "x" indicates a don't-care bit.

On all returns, the bits are interpreted as follows:

1 xxxx	--	Generate a prereturn trace
0 000x	--	Perform a local return
0 001T	--	In supervisor mode, perform a supervisor return. The T bit is assigned to the trace-enable bit in the process controls, and the execution-mode bit is set to user. Otherwise, perform a local return.
0 010x	--	Perform an interrupt return
0 011x	--	Perform a fault return
0 100x	--	Perform a subsystem return
0 101x	--	OPERATION. RETURN fault
0 110x	--	Perform an idle/stopped-interrupt return
0 111x	--	OPERATION. RETURN fault

Stack Pointer, S P (Li). A linear address (44) to the first free byte of the stack, that is, the address of the last byte in the stack plus one. SP is initialized by the call operation to point to FP plus 64.

Return Instruction Pointer, RIP (L2). When a call operation is performed to a new frame, the return IP (46) is saved here. When the process is suspended, the instruction pointer of the next instruction is stored here. It contains a 32-bit linear address to which control is returned after a return to this frame.

A procedure call saves the IP in a register of the current frame. Since implicit procedure calls can occur (due to faults and interrupts), programs do not use this register for other purposes.

The Stack grows (Figure 4) from low addresses to high addresses.

Figure 5 illustrates the mapping of the microprocessor's register sets into the program's stack in memory.
The page, or simple object, into which the first 64 bytes of a frame are mapped must be of local lifetime. The lifetime of the page or simple object is checked during a call. This restriction is necessary to ensure efficient manipultion of ADs in the local registers.

Linear address space structure

As shown in Figure 2, each execution environment defines a 32-bit linear address space. The linear address space is partitioned into four regions. The first three regions of an execution environment are specific to the current process (i.e., defined by the process control block). The composition of the process specific regions can be changed by a subsystem call/return. The fourth region of an execution environment is shared by all processes (i.e., defined by the processor control block). There are no restrictions on where instructions, stack frames, or data are located int he linear address space.

Local procedure mechanism

A procedure begins at any arbitary word address in a linear-address space. Procedure calls and return use a stack in the linear address space.

Instructions

CALL
CALL-EXTENDED

CALL and CALL-EXTENDED invoke the procedure at the address specified. CALL specifies the procedure as IP plus a 24-bit signed displacement. CALL-EXTENDED specifies the procedure using a general memory effective address. CALL-EXTENDED also contains an operand which becomes AP in the new frame.
A new stack frame is allocated during the call operation and the control flow is transferred to the specified procedures. The execution environment remains unchanged.

RETURN

The RETURN instruction transfers control back to the calling procedure's addressing environment and releases the called procedure's stack frame. Instruction execution is continued at the instruction pointed to by the RIP in the calling procedure's frame.

MODIFY-AC
CONVERT-ADDRESS

MODIFY-AC is used to read or modify the current arithmeter controls. Because the region ADS are not directly accessible, the CONVERT-ADDRESS instruction can be used to convert a linear address into a virtual address.

Process management

A software process or task, is represented by a process control block. Two means are provided for the control of process switching. One is via two instructions (save-process and resume-process), which allow an operating system to switch processes explicitly. Another is a priority-based process scheduling and dispatching function that is built into the processor. Using the latter mechanism, the processor will automatically dispatch processes from a queue in memory.
The processor keeps track of the cumulative execution time of each process, and also provides optional time-slice management. For the latter, whenever a process executes for longer than a prescribed amount of time, the processor will generate a fault, or enqueue the process on the queue of available processes and dispatch another process.
When automatic process dispatching is used, a set of interprocess communication instructions are provided, which are similar to services normally provided in software operating-system kernels. They provide support for the communication of messages among processes.

Tracing and ICE Support

Software debugging and tracing is provided by means of a trace-controls register that is part of each process. The trace controls allow detection of any combination of the following events:
Instruction execution (i.e., single step)
Execution of a taken branch instruction
Execution of a call instruction
Execution of a return instruction
Detection that the next instruction is a return instruction
Execution of a supervisor or subsystem call
Breakpoint (hardware breakpoint or execution of a breakpoint instruction)

When a trace event is detected, the processor generates a trace fault to give control to a software debugger or monitor. The processor contains two instruction breakpoint registers, into which a debugger can place the addresses of two instructions.

External bus

The microprocessor's bus is a 32-bit multiplexed bus with burst-transfer capability. The burst-transfer mechanism (which allows multiple words to be transferred in successive cycles) allows the bus to be defined as multiplexed. Burst transfers can occur for 1, 2, 3, or 4 words. During the address cycle, the processor indicates the number of words in the request in the low-order two address bits. For instance, if the processor wishes to read four words, the bus operation isn't terminated until four READY's are received. Burst-transfer operations are used often by the processor for instruction-cache fills, stack-frame-cache saves and restores, multiword loads and stores, string operations, and so on.

The microprocessor is highly pipelined. There are normally five instructions in different stages of execution in the pipeline at any given moment. In any given cycle, the instruction pointer to instruction n+4 is computed, instruction n+3 is read from the instruction cache, instruction n +2 is decoded and issued to the microinstruction bus, instruction n+1 is being executed, and the result of instruction n is being stored into the register file.

While the invention has been particularly shown and described with reference to preferred embodiments thereof, it will be understood by those skilled in the art that the foregoing and other changes in form and detail may be made therein without departing from the spirit and scope of the invention.

CLAIMS

1. For use in a data processor connectable to a main memory, a stack frame cache in said data processor and a control mechanism therefor comprising:

a plurality of global registers, one of said global registers being a frame pointer register containing a current frame pointer which points to a current frame, and the remainder of said global registers being available as general registers to a current process;

a register set pool made up of a plurality of register sets, each register set being comprised of a number of local registers;

first control means activated upon the decoding of a call instruction for allocating, to the called procedure, a register set of local registers from said register set pool and for initializing one of said frame pointer registers to create a current frame; and,

second control means activated upon the decoding of a return instruction for freeing said register set for allocation to another procedure called by a subsequent call instruction.

2. The combination in accordance with claim 1 wherein said first control means includes means operable upon the condition that said register set pool is depleted for saving, in said main memory, the contents of a register set associated with a previous procedure and for allocating said register set to said current procedure.

3. The combination in accordance with claim 1 wherein said first control means includes return status bits for recording the call mechanism used in the creation of said frame and for selecting the return mechanism to be used on return.

4. The combination in accordance with claim 1 wherein said local registers in a register set associated with a procedure contain linkage information including a pointer to the previous frame and an instruction pointer.

5. The combination in accordance with claim 2 wherein said local registers in a register set associated with a procedure contain linkage information including a pointer to the previous frame and an instruction pointer.

6. The combination in accordance with claim 1 wherein said local registers in a register set associated with a procedure contain a stack frame including frame status bits for recording the information associated with the frame after a call, to be used on a return from the frame, said information including return status bits for recording the call mechanism used in the creation of said frame and for selecting the return mechanism to be used on return.

7. The combination in accordance with claim 1 wherein said local registers in a register set associated with a procedure contain a stack frame including previous frame pointer bits comprised of a linear address to the first byte of the previous frame.

8. The combination in accordance with claim 1 wherein said local registers in a register set associated with a procedure contain a stack frame including a return instruction pointer field for storing the return instruction pointer upon the condition that a call operation is performed to a new frame and for storing the instruction pointer of the next instruction upon the condition that the process is suspended, said field being a linear address to which control is returned after a return to this frame.

9. The combination in accordance with claim 2 wherein said local registers in a register set associated with a procedure contain a stack frame including frame status bits for recording the information associated with the frame after a call, to be used on a return from the frame, said information including return status bits for recording the call mechanism used in the creation of said frame and for selecting the return mechanism to

be used on return.

10. The combination in accordance with claim 2 wherein said local registers in a register set associated with a procedure contain a stack frame including previous frame pointer bits comprised of a linear address to the first byte of the previous frame.

11. The combination in accordance with claim 2 wherein said local registers in a register set associated with a procedure contain a stack frame including a return instruction pointer field for storing the return instruction pointer upon the condition that a call operation is performed to a new frame and for storing the instruction pointer of the next instruction upon the condition that the process is suspended, said field being a linear address to which control is returned after a return to this frame.

12. A stack frame cache for use in a data processor connectable to a main memory substantially as hereinbefore described with reference to the accompanying drawings.

Printed for Her Majesty's Stationery Office by Croydon Printing Company (UK) Ltd, 9/87, D8991685.
Published by The Patent Office, 25 Southampton Buildings, London WC2A 1AY, from which copies may be obtained.