US 20160202988A1

(19) United States
(12) Patent Application Publication
Ayub et al.
(10) Pub. No.: US 2016/0202988 A1
(43) Pub. Date: Jul. 14, 2016

(54) PARALLEL SLICE PROCESSING METHOD USING A RECIRCULATING LOAD-STORE QUEUE FOR FAST DEALLOCATION OF ISSUE QUEUE ENTRIES

(71) Applicant: **INTERNATIONAL BUSINESS MACHINES CORPORATION**, ARMONK, NY (US)

(72) Inventors: **Salma Ayub**, Austin, TX (US); **Sundeep Chadha**, AUSTIN, TX (US); **Robert Allen Cordes**, Austin, TX (US); **David Allen Hrusecky**, Cedar Park, TX (US); **Hung Qui Le**, Austin, TX (US); **Dung Quoc Nguyen**, AUSTIN, TX (US); **Brian William Thompto**, Austin, TX (US)

(21) Appl. No.: **14/724,268**

(22) Filed: **May 28, 2015**

Related U.S. Application Data

(63) Continuation of application No. 14/595,635, filed on Jan. 13, 2015.

Publication Classification

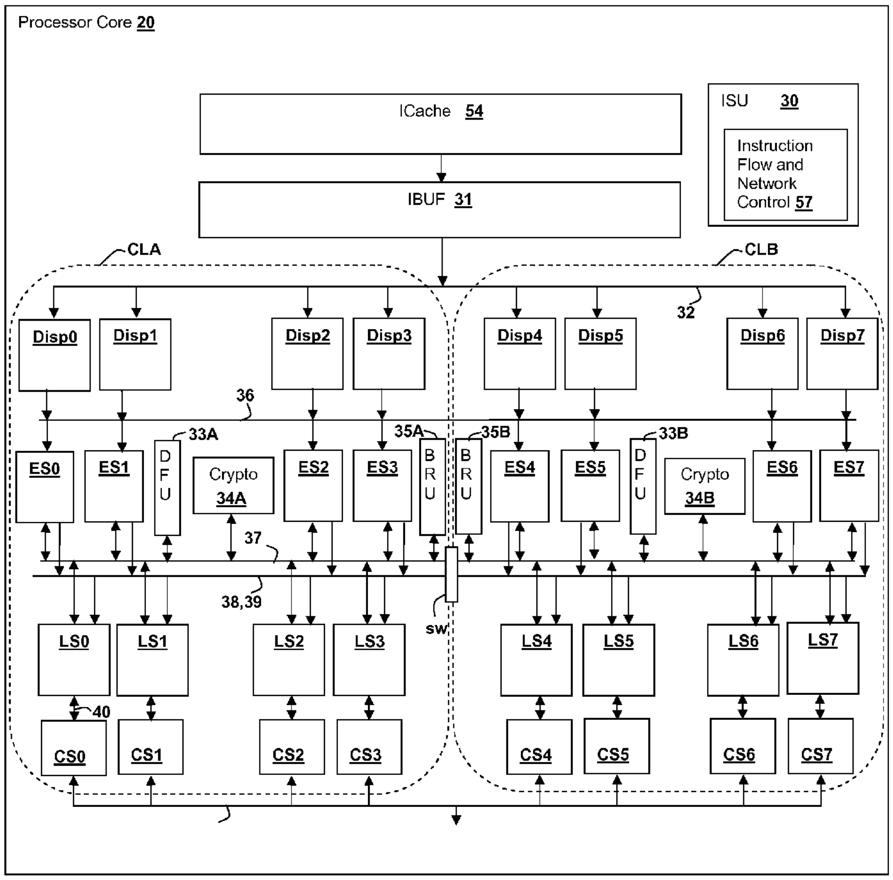
(51) **Int. Cl.**

G06F 9/38	(2006.01)
G06F 9/30	(2006.01)
G06F 12/08	(2006.01)

(52) **U.S. Cl.**
CPC ***G06F 9/3836*** (2013.01); ***G06F 12/0875*** (2013.01); ***G06F 9/30043*** (2013.01); *G06F 2212/1021* (2013.01); *G06F 2212/452* (2013.01); *G06F 2212/608* (2013.01)

(57) ABSTRACT

A method of operation of a processor core execution unit circuit provides efficient use of area and energy by reducing the per-entry storage requirement of a load-store unit issue queue. The execution unit circuit includes a recirculation queue that stores the effective address of the load and store operations and the values to be stored by the store operations. A queue control logic controls the recirculation queue and issue queue so that that after the effective address of a load or store operation has been computed, the effective address of the load operation or the store operation is written to the recirculation queue and the operation is removed from the issue queue, so that address operands and other values that were in the issue queue entry no longer require storage. When a load or store operation is rejected by the cache unit, it is subsequently reissued from the recirculation queue.



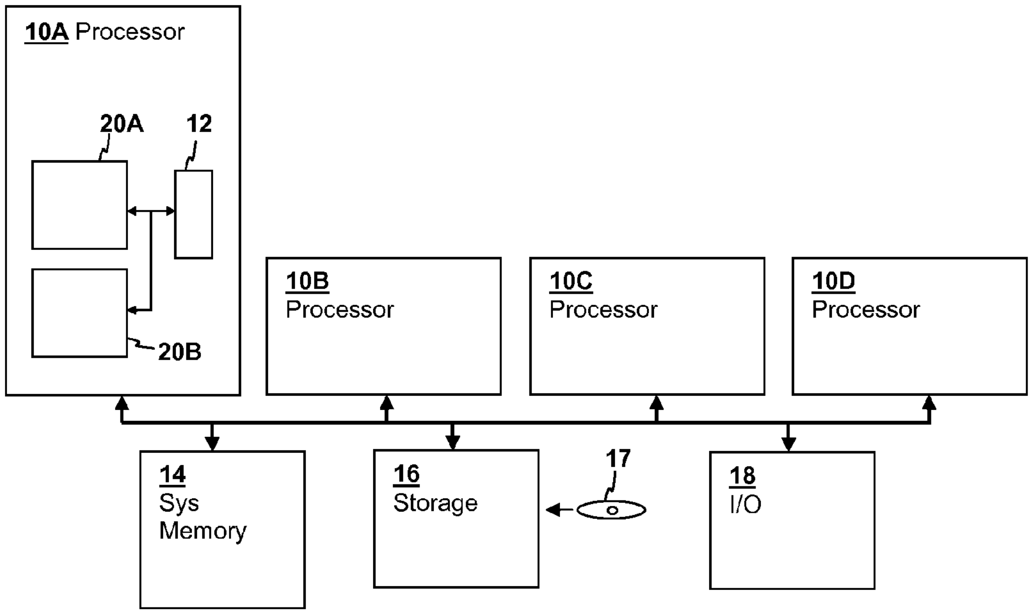

10A Processor
20A
12
20B
10B Processor
10C Processor
10D Processor
14 Sys Memory
16 Storage
17
18 I/O

Fig. 1

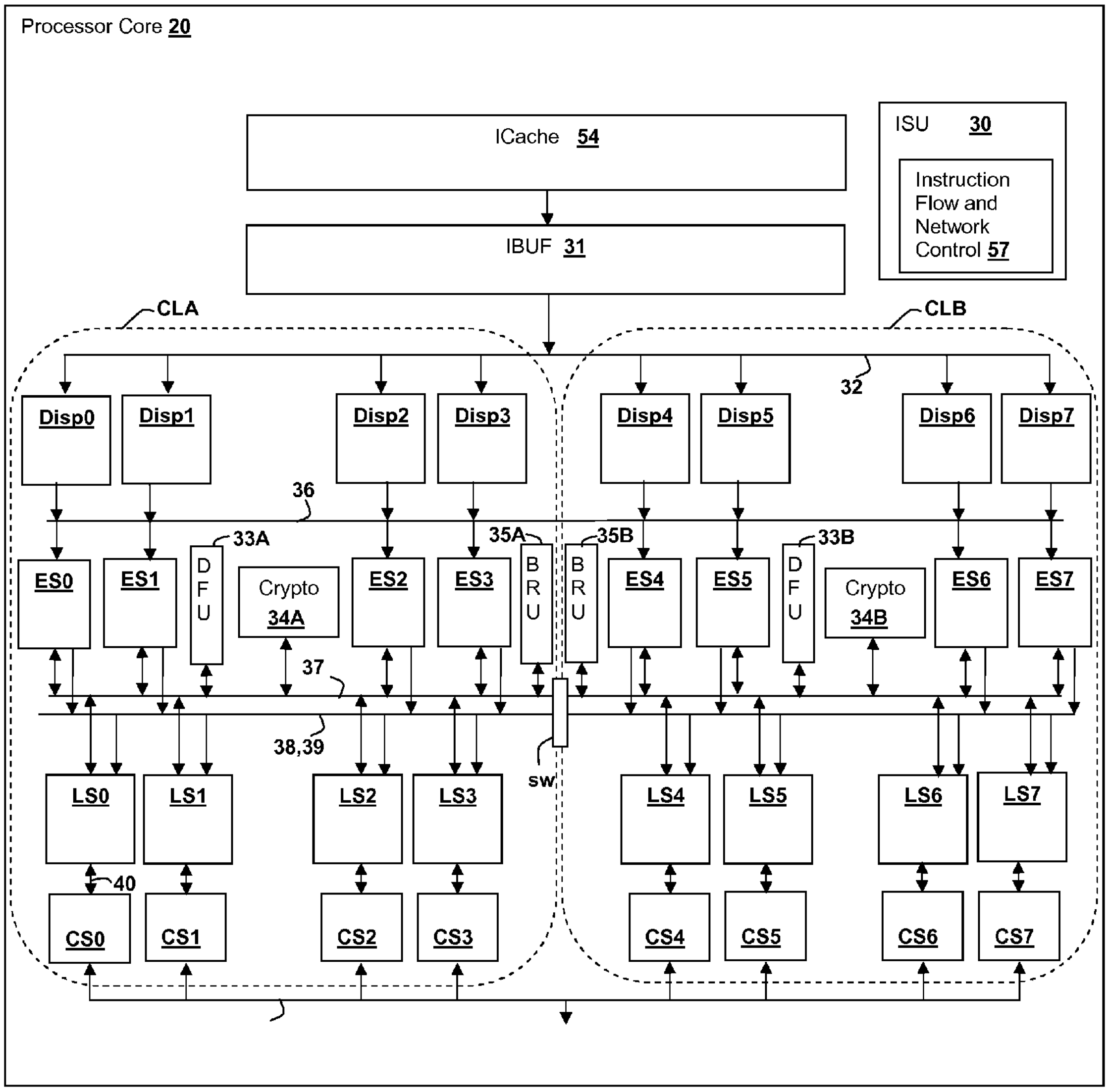

Processor Core 20
ICache 54
IBUF 31
ISU 30
Instruction Flow and Network Control 57
CLA
CLB
32
Disp0
Disp1
Disp2
Disp3
Disp4
Disp5
Disp6
Disp7
36
33A
35A
35B
33B
ES0
ES1
DFU
Crypto 34A
ES2
ES3
BRU
BRU
ES4
ES5
DFU
Crypto 34B
ES6
ES7
37
38,39
SW
LS0
LS1
LS2
LS3
LS4
LS5
LS6
LS7
40
CS0
CS1
CS2
CS3
CS4
CS5
CS6
CS7

Fig. 2

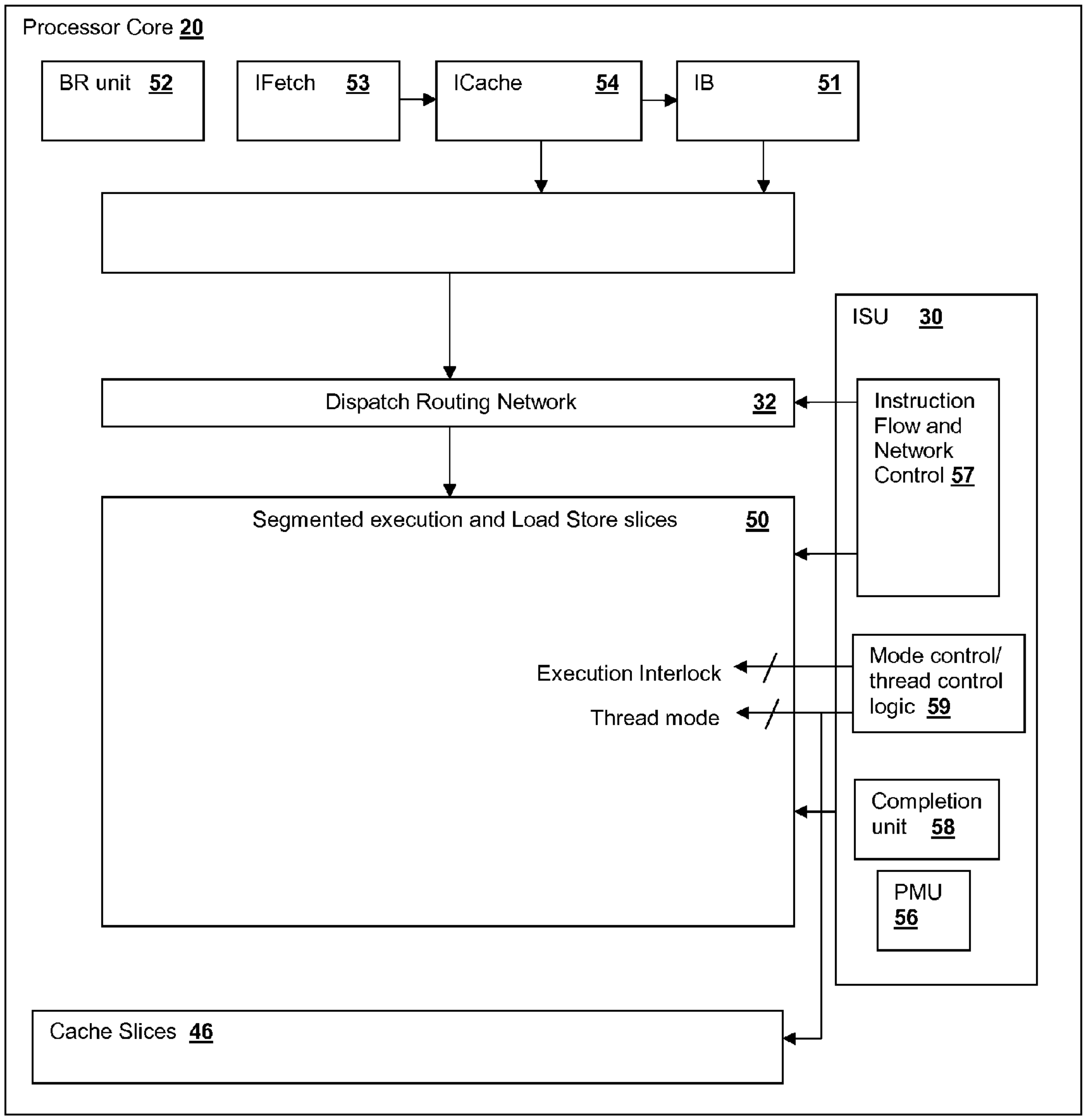

Processor Core 20
BR unit 52
IFetch 53
ICache 54
IB 51
ISU 30
Dispatch Routing Network 32
Instruction Flow and Network Control 57
Segmented execution and Load Store slices 50
Execution Interlock
Thread mode
Mode control/ thread control logic 59
Completion unit 58
PMU 56
Cache Slices 46

Fig. 3

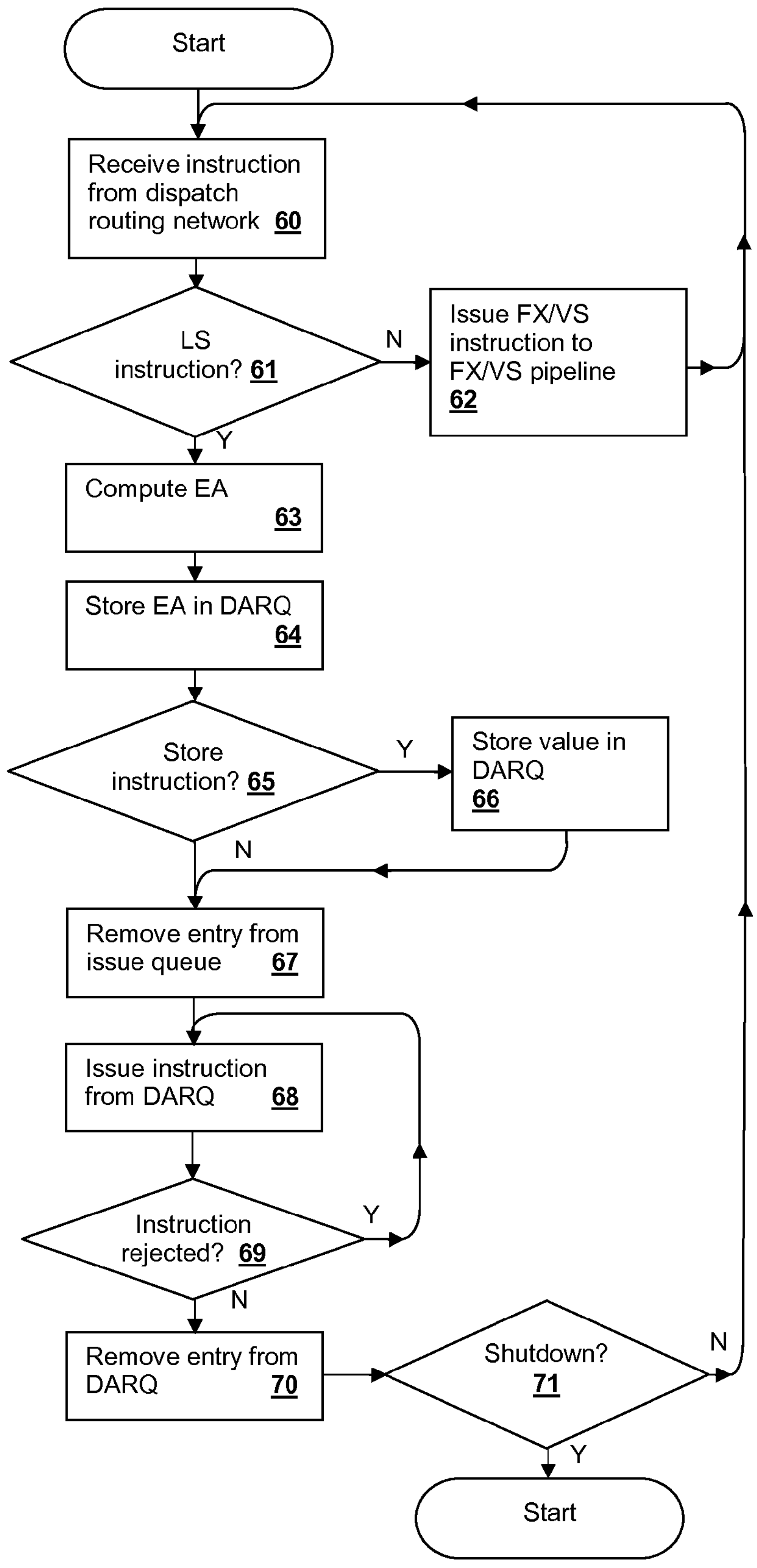

Start
Receive instruction from dispatch routing network 60
LS instruction? 61
N
Y
Issue FX/VS instruction to FX/VS pipeline 62
Compute EA 63
Store EA in DARQ 64
Store instruction? 65
Y
N
Store value in DARQ 66
Remove entry from issue queue 67
Issue instruction from DARQ 68
Instruction rejected? 69
Y
N
Remove entry from DARQ 70
Shutdown? 71
N
Y
Start

Fig. 4

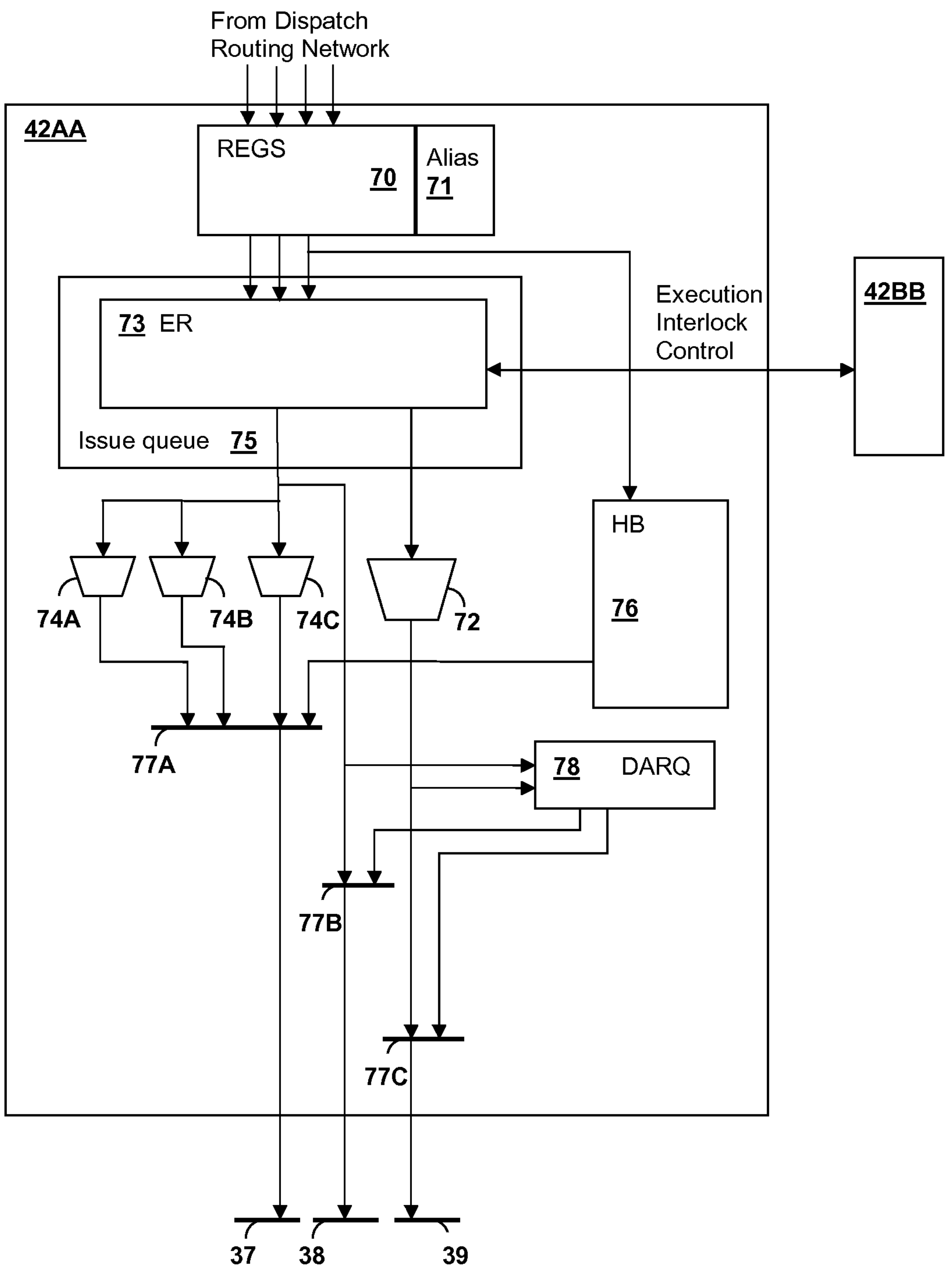

From Dispatch Routing Network
42AA
REGS 70
Alias 71
73 ER
Issue queue 75
Execution Interlock Control
42BB
HB 76
74A
74B
74C
72
77A
78 DARQ
77B
77C
37
38
39

Fig. 5

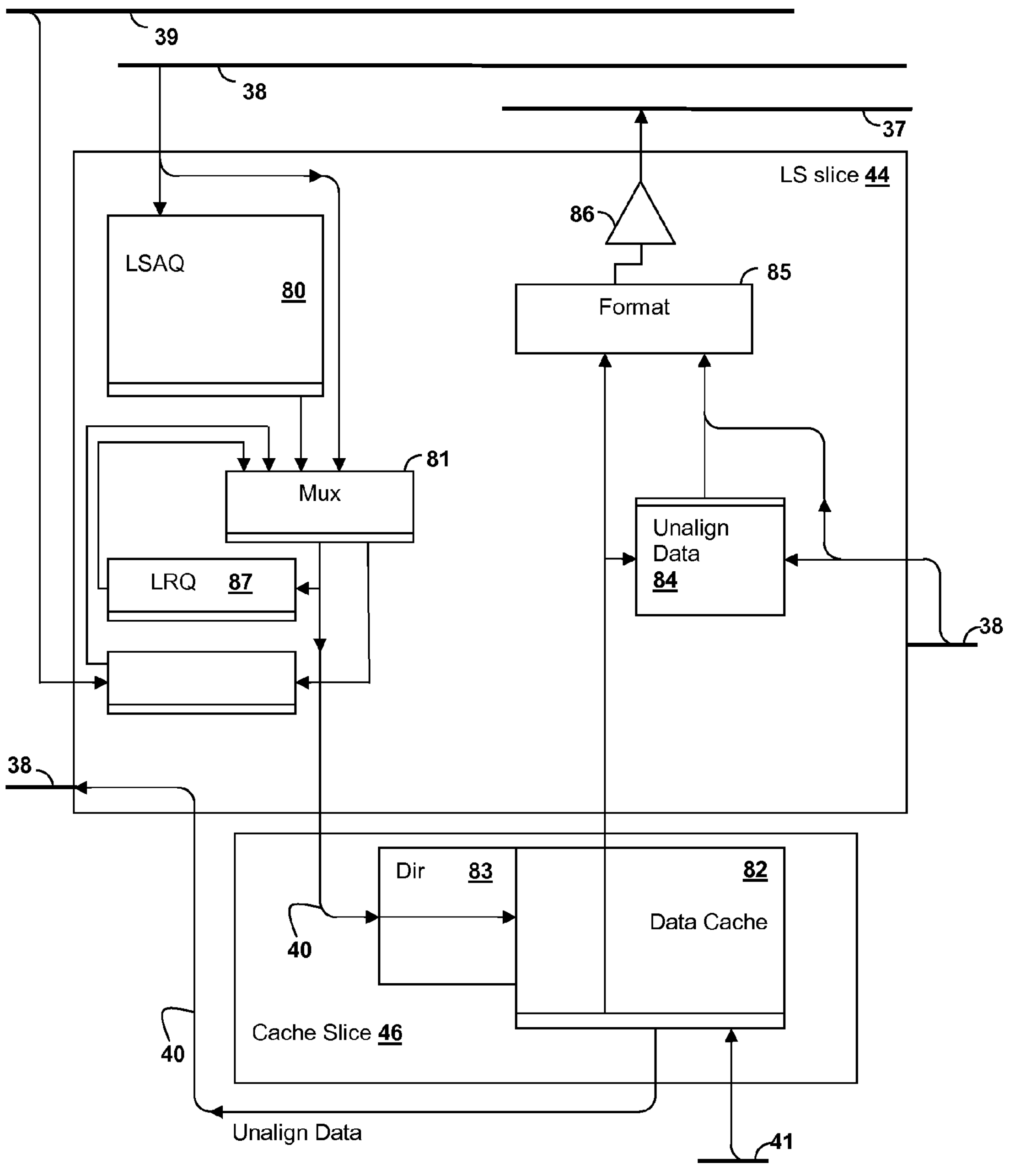

39
38
37
LS slice 44
LSAQ
80
86
85
Format
81
Mux
LRQ 87
Unalign
Data
84
38
38
Dir 83
82
Data Cache
40
40
Cache Slice 46
Unalign Data
41

Fig. 6

PARALLEL SLICE PROCESSING METHOD USING A RECIRCULATING LOAD-STORE QUEUE FOR FAST DEALLOCATION OF ISSUE QUEUE ENTRIES

[0001] The present application is a Continuation of U.S. patent application Ser. No. 14/595,635, filed on Jan. 13, 2015 and claims priority thereto under 35 U.S.C. §120. The disclosure of the above-referenced parent U.S. patent application is incorporated herein by reference.

BACKGROUND OF THE INVENTION

[0002] 1. Field of the Invention

[0003] The present invention is related to processing systems and processors, and more specifically to a pipelined processor core that includes execution slices having a recirculating load-store queue.

[0004] 2. Description of Related Art

[0005] In present-day processor cores, pipelines are used to execute multiple hardware threads corresponding to multiple instruction streams, so that more efficient use of processor resources can be provided through resource sharing and by allowing execution to proceed even while one or more hardware threads are waiting on an event.

[0006] In existing processor cores, and in particular processor cores that are divided into multiple execution slices instructions are dispatched to the execution slice(s) and are retained in the issue queue until issued to an execution unit. Once an issue queue is full, additional operations cannot typically be dispatched to a slice. Since the issue queue contains not only operations, but operands and state/control information, issue queues are resource-intensive, requiring significant power and die area to implement.

[0007] It would therefore be desirable to provide a method of operation of a processor core having reduced issue queue requirements.

BRIEF SUMMARY OF THE INVENTION

[0008] The invention is embodied in a method of operation of a processor core.

[0009] The execution unit circuit includes an issue queue that receives a stream of instructions including functional operations and load-store operations, and multiple execution pipelines including a load-store pipeline that computes effective addresses of load operations and store operations, and issues the load operations and store operations to a cache unit. The execution unit circuit also includes a recirculation queue that stores entries corresponding to the load operations and the store operations and control logic for controlling the issue queue, the load-store pipeline and the recirculation queue. The control logic operates so that after the load-store pipeline has computed the effective address of a load operation or a store operation, the effective address of the load operation or the store operation is written to the recirculation queue and the load operation or the store operation is removed from the issue queue so that if one of the load operations or store operations are rejected by the cache unit, they are subsequently reissued to the cache unit from the recirculation queue.

[0010] The foregoing and other objectives, features, and advantages of the invention will be apparent from the following, more particular, description of the preferred embodiment of the invention, as illustrated in the accompanying drawings.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWING

[0011] The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives, and advantages thereof, will best be understood by reference to the following detailed description of the invention when read in conjunction with the accompanying Figures, wherein like reference numerals indicate like components, and:

[0012] FIG. **1** is a block diagram illustrating a processing system in which techniques according to an embodiment of the present invention are practiced.

[0013] FIG. **2** is a block diagram illustrating details of a processor core **20** that can be used to implement processor cores **20**A-**20**B of FIG. **1**.

[0014] FIG. **3** is a block diagram illustrating details of processor core **20**.

[0015] FIG. **4** is a flowchart illustrating a method of operating processor core **20**.

[0016] FIG. **5** is a block diagram illustrating details of an instruction execution slice **42**AA that can be used to implement instruction execution slices ES**0**-ES**7** of FIGS. **2**-**3**.

[0017] FIG. **6** is a block diagram illustrating details of a load store slice **44** and a cache slice **46** that can be used to implement load-store slices LS**0**-LS**7** and cache slices CS**0**-CS**7** of FIGS. **2**-**3**.

DETAILED DESCRIPTION OF THE INVENTION

[0018] The present invention relates to an execution slice for inclusion in a processor core that manages an internal issue queue by moving load/store (LS) operation entries to a recirculation queue once the effective address (EA) of the LS operation has been computed. The LS operations are issued to a cache unit and if they are rejected, the LS operations are subsequently re-issued from the recirculation queue rather than from the original issue queue entry. Since the recirculation queue entries only require storage for the EA for load operations and the EA and store value for store operations, power and area requirements are reduced for a given number of pending LS issue queue entries in the processor. In contrast, the issue queue entries are costly in terms of area and power due to the need to store operands, relative addresses and other fields such as conditional flags that are not needed for executing the LS operations once the EA is resolved.

[0019] Referring now to FIG. **1**, a processing system in accordance with an embodiment of the present invention is shown. The depicted processing system includes a number of processors **10**A-**10**D, each in conformity with an embodiment of the present invention. The depicted multi-processing system is illustrative, and a processing system in accordance with other embodiments of the present invention include uniprocessor systems having multi-threaded cores. Processors **10**A-**10**D are identical in structure and include cores **20**A-**20**B and a local storage **12**, which may be a cache level, or a level of internal system memory. Processors **10**A-**10**B are coupled to a main system memory **14**, a storage subsystem **16**, which includes non-removable drives and optical drives, for reading media such as a CD-ROM **17** forming a computer program product and containing program instructions implementing generally, at least one operating system, associated applications programs, and optionally a hypervisor for controlling multiple operating systems' partitions for execution by processors **10**A-**10**D. The illustrated processing system also includes input/output (I/O) interfaces and devices **18** such as mice and keyboards for receiving user input and graphical displays for displaying information. While the system of FIG. **1** is used to provide an illustration of a system in which the processor architecture of the present invention is implemented, it is understood that the depicted architecture is not limiting and is intended to provide an example of a suitable computer system in which the techniques of the present invention are applied.

[0020] Referring now to FIG. **2**, details of an exemplary processor core **20** that can be used to implement processor cores **20**A-**20**B of FIG. **1** are illustrated. Processor core **20** includes an instruction cache (ICache) **54** and instruction buffer (IBUF) **31** that store multiple instruction streams fetched from cache or system memory and present the instruction stream(s) via a bus **32** to a plurality of dispatch queues Disp**0**-Disp**7** within each of two clusters CLA and CLB. Control logic within processor core **20** controls the dispatch of instructions from dispatch queues Disp**0**-Disp**7** to a plurality of instruction execution slices ES**0**-ES**7** via a dispatch routing network **36** that permits instructions from any of dispatch queues Disp**0**-Disp**7** to any of instruction execution slices ES**0**-ES**7** in either of clusters CLA and CLB, although complete cross-point routing, i.e., routing from any dispatch queue to any slice is not a requirement of the invention. In certain configurations as described below, the dispatch of instructions from dispatch queues Disp**0**-Disp**3** in cluster CLA will be restricted to execution slices ES**0**-ES**3** in cluster CLA, and similarly the dispatch of instructions from dispatch queues Disp**4**-Disp**7** in cluster CLB will be restricted to execution slices ES**4**-ES**7**. Instruction execution slices ES**0**-ES**7** perform sequencing and execution of logical, mathematical and other operations as needed to perform the execution cycle portion of instruction cycles for instructions in the instruction streams, and may be identical general-purpose instruction execution slices ES**0**-ES**7**, or processor core **20** may include special-purpose execution slices ES**0**-ES**7**. Other special-purpose units such as cryptographic processors **34**A-**34**B, decimal floating points units (DFU) **33**A-**33**B and separate branch execution units (BRU) **35**A-**35**B may also be included to free general-purpose execution slices ES**0**-ES**7** for performing other tasks. Instruction execution slices ES**0**-ES**7** may include multiple internal pipelines for executing multiple instructions and/or portions of instructions.

[0021] The load-store portion of the instruction execution cycle, (i.e., the operations performed to maintain cache consistency as opposed to internal register reads/writes), is performed by a plurality of load-store (LS) slices LS**0**-LS**7**, which manage load and store operations as between instruction execution slices ES**0**-ES**7** and a cache memory formed by a plurality of cache slices CS**0**-CS**7** which are partitions of a lowest-order cache memory. Cache slices CS**0**-CS**3** are assigned to partition CLA and cache slices CS**4**-CS**7** are assigned to partition CLB in the depicted embodiment and each of load-store slices LS**0**-LS**7** manages access to a corresponding one of the cache slices CS**0**-CS**7** via a corresponding one of dedicated memory buses **40**. In other embodiments, there may be not be a fixed partitioning of the cache, and individual cache slices CS**0**-CS**7** or sub-groups of the entire set of cache slices may be coupled to more than one of load-store slices LS**0**-LS**7** by implementing memory buses **40** as a shared memory bus or buses. Load-store slices LS**0**-LS**7** are coupled to instruction execution slices ES**0**-ES**7** by a write-back (result) routing network **37** for returning result data from corresponding cache slices CS**0**-CS**7**, such as in response to load operations. Write-back routing network **37** also provides communications of write-back results between instruction execution slices ES**0**-ES**7**. Further details of the handling of load/store (LS) operations between instruction execution slices ES**0**-ES**7**, load-store slices LS**0**-LS**7** and cache slices CS**0**-CS**7** is described in further detail below with reference to FIGS. **4**-**6**. An address generating (AGEN) bus **38** and a store data bus **39** provide communications for load and store operations to be communicated to load-store slices LS**0**-LS**7**. For example, AGEN bus **38** and store data bus **39** convey store operations that are eventually written to one of cache slices CS**0**-CS**7** via one of memory buses **40** or to a location in a higher-ordered level of the memory hierarchy to which cache slices CS**0**-CS**7** are coupled via an I/O bus **41**, unless the store operation is flushed or invalidated. Load operations that miss one of cache slices CS**0**-CS**7** after being issued to the particular cache slice CS**0**-CS**7** by one of load-store slices LS**0**-LS**7** are satisfied over I/O bus **41** by loading the requested value into the particular cache slice CS**0**-CS**7** or directly through cache slice CS**0**-CS**7** and memory bus **40** to the load-store slice LS**0**-LS**7** that issued the request. In the depicted embodiment, any of load-store slices LS**0**-LS**7** can be used to perform a load-store operation portion of an instruction for any of instruction execution slices ES**0**-ES**7**, but that is not a requirement of the invention. Further, in some embodiments, the determination of which of cache slices CS**0**-CS**7** will perform a given load-store operation may be made based upon the operand address of the load-store operation together with the operand width and the assignment of the addressable byte of the cache to each of cache slices CS**0**-CS**7**.

[0022] Instruction execution slices ES**0**-ES**7** may issue internal instructions concurrently to multiple pipelines, e.g., an instruction execution slice may simultaneously perform an execution operation and a load/store operation and/or may execute multiple arithmetic or logical operations using multiple internal pipelines. The internal pipelines may be identical, or may be of discrete types, such as floating-point, scalar, load/store, etc. Further, a given execution slice may have more than one port connection to write-back routing network **37**, for example, a port connection may be dedicated to load-store connections to load-store slices LS**0**-LS**7**, or may provide the function of AGEN bus **38** and/or data bus **39**, while another port may be used to communicate values to and from other slices, such as special-purposes slices, or other instruction execution slices. Write-back results are scheduled from the various internal pipelines of instruction execution slices ES**0**-ES**7** to write-back port(s) that connect instruction execution slices ES**0**-ES**7** to write-back routing network **37**. Cache slices CS**0**-CS**7** are coupled to a next higher-order level of cache or system memory via I/O bus **41** that may be integrated within, or external to, processor core **20**. While the illustrated example shows a matching number of load-store slices LS**0**-LS**7** and execution slices ES**0**-ES**7**, in practice, a different number of each type of slice can be provided according to resource needs for a particular implementation.

[0023] Within processor core **20**, an instruction sequencer unit (ISU) **30** includes an instruction flow and network control block **57** that controls dispatch routing network **36**, write-back routing network **37**, AGEN bus **38** and store data bus **39**. Network control block **57** also coordinates the operation of execution slices ES**0**-ES**7** and load-store slices LS**0**-LS**7** with the dispatch of instructions from dispatch queues Disp**0**-Disp**7**. In particular, instruction flow and network control block **57** selects between configurations of execution slices ES**0**-ES**7** and load-store slices LS**0**-LS**7** within processor core **20** according to one or more mode control signals that allocate the use of execution slices ES**0**-ES**7** and load-store slices LS**0**-LS**7** by a single thread in one or more single-threaded (ST) modes, and multiple threads in one or more multi-threaded (MT) modes, which may be simultaneous multi-threaded (SMT) modes. For example, in the configuration shown in FIG. **2**, cluster CLA may be allocated to one or more hardware threads forming a first thread set in SMT mode so that dispatch queues Disp**0**-Disp**3** only receive instructions of instruction streams for the first thread set, execution slices ES**0**-ES**3** and load-store slices LS**0**-LS**3** only perform operations for the first thread set and cache slices CS**0**-CS**3** form a combined cache memory that only contains values accessed by the first thread set. Similarly, in such an operating mode, cluster CLB is allocated to a second hardware thread set and dispatch queues Disp**4**-Disp**7** only receive instructions of instruction streams for the second thread set, execution slices ES**4**-ES**7** and LS slices LS**4**-LS**7** only perform operations for the second thread set and cache slices CS**4**-CS**7** only contain values accessed by the second thread set. When communication is not required across clusters, write-back routing network **37** can be partitioned by disabling transceivers or switches sw connecting the portions of write-back routing network **37**, cluster CLA and cluster CLB. Separating the portions of write-back routing network **37** provides greater throughput within each cluster and allows the portions of write-back routing network **37** to provide separate simultaneous routes for results from execution slices ES**0**-ES**7** and LS slices LS**0**-LS**7** for the same number of wires in write-back routing network **37**. Thus, twice as many transactions can be supported on the divided write-back routing network **37** when switches sw are open. Other embodiments of the invention may sub-divide the sets of dispatch queues Disp**0**-Disp**7**, execution slices ES**0**-ES**7**, LS slices LS**0**-LS**7** and cache slices CS**0**-CS**7**, such that a number of clusters are formed, each operating on a particular set of hardware threads. Similarly, the threads within a set may be further partitioned into subsets and assigned to particular ones of dispatch queues Disp**0**-Disp**7**, execution slices ES**0**-ES**7**, LS slices LS**0**-LS**7** and cache slices CS**0**-CS**7**. However, the partitioning is not required to extend across all of the resources listed above. For example, clusters CLA and CLB might be assigned to two different hardware thread sets, and execution slices ES**0**-ES**2** and LS slices LS**0**-LS**1** assigned to a first subset of the first hardware thread set, while execution slice ES**3** and LS slices LS**2**-LS**3** are assigned to a second subject of the first hardware thread set, while cache slices CS**0**-CS**3** are shared by all threads within the first hardware thread set. In a particular embodiment according to the above example, switches may be included to further partition write back routing network **37** between execution slices ES**0**-ES**7** such that connections between sub-groups of execution slices ES**0**-ES**7** that are assigned to different thread sets are isolated to increase the number of transactions that can be processed within each sub-group. The above is an example of the flexibility of resource assignment provided by the bus-coupled slice architecture depicted in FIG. **2**, and is not a limitation as to any particular configurations that might be supported for mapping sets of threads or individual threads to resources such as dispatch queues Disp**0**-Disp**7**, execution slices ES**0**-ES**7**, LS slices LS**0**-LS**7** and cache slices CS**0**-CS**7**.

[0024] Referring now to FIG. **3**, further details of processor core **20** are illustrated. Processor core **20** includes a branch execution unit **52** that evaluates branch instructions, and an instruction fetch unit (IFetch) **53** that controls the fetching of instructions including the fetching of instructions from ICache **54**. Instruction sequencer unit (ISU) **30** controls the sequencing of instructions. An input instruction buffer (IB) **51** buffers instructions in order to map the instructions according to the execution slice resources allocated for the various threads and any super-slice configurations that are set. Another instruction buffer (IBUF) **31** is partitioned to maintain dispatch queues (Disp**0**-Disp**7** of FIGS. **2**-**3**) and dispatch routing network **32** couples IBUF **31** to the segmented execution and load-store slices **50**, which are coupled to cache slices **46**. Instruction flow and network control block **57** performs control of segmented execution and load-store slices **50**, cache slices **46** and dispatch routing network **32** to configure the slices as illustrated in FIGS. **2**-**3**, according to a mode control/thread control logic **59**. An instruction completion unit **58** is also provided to track completion of instructions sequenced by ISU **30**. ISU **30** also contains logic to control write-back operations by load-store slices LS**0**-LS**7** within segmented execution and load-store slices **50**. A power management unit **56** may also provide for energy conservation by reducing or increasing a number of active slices within segmented execution and cache slices **50**. Although ISU **30** and instruction flow and network control block **57** are shown as a single unit, control of segmented execution within and between execution slices ES**0**-ES**7** and load store slices LS**0**-LS**7** may be partitioned among the slices such that each of execution slices ES**0**-ES**7** and load store slices LS**0**-LS**7** may control its own execution flow and sequencing while communicating with other slices.

[0025] Referring now to FIG. **4**, a method of operating processor core **20** is shown according to an embodiment of the present invention. An instruction is received at one of execution slices ES**0**-ES**7** from dispatch routing network **32** (step **60**), and if the instruction is not an LS instruction, i.e., the instruction is a VS/FX instruction (decision **61**), then FX/VS instruction is issued to the FX/VS pipeline(s) (step **62**). If the instruction is an LS instruction (decision **61**), the EA is computed (step **63**) and stored in a recirculation queue (DARQ) (step **64**). If the instruction is not a store instruction (decision **65**) the entry is removed from the issued queue (step **67**) after the instruction is stored in the DARQ. If the instruction is a store instruction (decision **65**), then the store value is also stored in DARQ (step **66**) and after both the store instruction EA and store value are stored in DARQ, the entry is removed from the issued queue (step **67**) and the instruction is issued from DARQ (step **68**). If the instruction is rejected (decision **69**), then step **68** is repeated to subsequently reissue the rejected instruction. If the instruction is not rejected (decision **69**), then the entry is removed from DARQ (step **70**). Until the system is shut down (decision **71**), the process of steps **60**-**70** is repeated. In alternative methods in accordance with other embodiments of the invention, step **67** may be performed only after an attempt to issue the instruction has been performed, and in another alternative, steps **64** and **66** might only be performed after the instruction has been rejected once, and other variations that still provide the advantage of the reduced storage requirements of an entry in the DARQ vs. and entry in the issue queue.

[0026] Referring now to FIG. **5**, an example of an execution slice (ES) **42**AA that can be used to implement instruction execution slices ES**0**-ES**7** in FIGS. **2**-**3** is shown. Inputs from the dispatch queues are received via dispatch routing network **32** by a register array **70** so that operands and the instructions can be queued in execution reservation stations (ER) **73** of issue queue **75**. Register array **70** is architected to have independent register sets for independent instruction streams or where execution slice **42**AA is joined in a super-slice executing multiple portions of an SIMD instruction, while dependent register sets that are clones in super-slices are architected for instances where the super-slice is executing non-SIMD instructions. An alias mapper **71** maps the values in register array **70** to any external references, such as write-back values exchanged with other slices over write-back routing network **37**. A history buffer HB **76** provides restore capability for register targets of instructions executed by ES **42**AA. Registers may be copied or moved between super-slices using write-back routing network **37** in response to a mode control signal, so that the assignment of slices to a set of threads or the assignment of slices to operate in a joined manner to execute as a super-slice together with other execution slices can be reconfigured. Execution slice **42**AA is illustrated alongside another execution slice **42**BB to illustrate an execution interlock control that may be provided between pairs of execution slices within execution slices ES**0**-ES**7** of FIGS. **2**-**3** to form a super-slice. The execution interlock control provides for coordination between execution slices **42**AA and **42**BB supporting execution of a single instruction stream, since otherwise execution slices ES**0**-ES**7** independently manage execution of their corresponding instruction streams.

[0027] Execution slice **42**AA includes multiple internal execution pipelines **74**A-**74**C and **72** that support out-of-order and simultaneous execution of instructions for the instruction stream corresponding to execution slice **42**AA. The instructions executed by execution pipelines **74**A-**74**C and **72** may be internal instructions implementing portions of instructions received over dispatch routing network **32**, or may be instructions received directly over dispatch routing network **32**, i.e., the pipelining of the instructions may be supported by the instruction stream itself, or the decoding of instructions may be performed upstream of execution slice **42**AA. Execution pipeline **72** is a load-store (LS) pipeline that executes LS instructions, i.e., computes effective addresses (EAs) from one or more operands. A recirculation queue (DARQ) **78** is controlled according to logic as illustrated above with reference to FIG. **4**, so execution pipeline **72** does not have to compute the EA of an instruction stored in DARQ **78**, since the entry in DARQ **78** is the EA, along with a store value for store operations. As described above, once an entry is present in DARQ **78**, the corresponding entry can be removed from an issue queue **75**. DARQ **78** can have a greater number of entries, freeing storage space in issue queue **75** for additional FX/VS operations, as well as other LS operations. FX/VS pipelines **74**A-**74**C may differ in design and function, or some or all pipelines may be identical, depending on the types of instructions that will be executed by execution slice **42**AA. For example, specific pipelines may be provided for address computation, scalar or vector operations, floating-point operations, etc. Multiplexers **77**A-**77**C provide for routing of execution results to/from history buffer **76** and routing of write-back results to write-back routing network **37**, I/O routing network **39** and AGEN routing network(s) **38** that may be provided for routing specific data for sharing between slices or operations, or for load and store address and/or data sent to one or more of load-store slices LS**0**-LS**7**. Data, address and recirculation queue (DARQ) **78** holds execution results or partial results such as load/store addresses or store data that are not guaranteed to be accepted immediately by the next consuming load-store slice LS**0**-LS**7** or execution slice ES**0**-ES**7**. The results or partial results stored in DARQ **78** may need to be sent in a future cycle, such as to one of load-store slices LS**0**-LS**7**, or to special execution units such as one of cryptographic processors **34**A,**34**B. Data stored in DARQ **78** may then be multiplexed onto AGEN bus **38** or store data bus **39** by multiplexers **77**B or **77**C, respectively.

[0028] Referring now to FIG. **6**, an example of a load-store (LS) slice **44** that can be used to implement load-store slices LS**0**-LS**7** in FIG. **2** is shown. A load/store access queue (LSAQ) **80** is coupled to AGEN bus **38**, and the direct connection to AGEN bus **38** and LSAQ **80** is selected by a multiplexer **81** that provides an input to a cache directory **83** of a data cache **82** in cache slice **46** via memory bus **40**. Logic within LSAQ **80** controls the accepting or rejecting of LS operations as described above, for example when a flag is set in directory **83** that will not permit modification of a corresponding value in data cache **82** until other operations are completed. The output of multiplexer **81** also provides an input to a load reorder queue (LRQ) **87** or store reorder queue (SRQ) **88** from either LSAQ **80** or from AGEN bus **38**, or to other execution facilities within load-store slice **44** that are not shown. Load-store slice **44** may include one or more instances of a load-store unit that execute load-store operations and other related cache operations. To track execution of cache operations issued to LS slice **44**, LRQ **87** and SRQ **88** contain entries for tracking the cache operations for sequential consistency and/or other attributes as required by the processor architecture. While LS slice **44** may be able to receive multiple operations per cycle from one or more of execution slices ES**0**-ES**7** over AGEN bus **38**, all of the accesses may not be concurrently executable in a given execution cycle due to limitations of LS slice **44**. Under such conditions, LSAQ **80** stores entries corresponding to as yet un-executed operations. SRQ **88** receives data for store operations from store data bus **39**, which are paired with operation information such as the computed store address. As operations execute, hazards may be encountered in the load-store pipe formed by LS slice **44** and cache slice **46**, such as cache miss, address translation faults, cache read/write conflicts, missing data, or other faults which require the execution of such operations to be delayed or retried. In some embodiments, LRQ **87** and SRQ **88** are configured to re-issue the operations into the load-store pipeline for execution, providing operation independent of the control and operation of execution slices ES**0**-ES**7**. Such an arrangement frees resources in execution slices ES**0**-ES**7** as soon as one or more of load-store slices LS**0**-LS**7** has received the operations and/or data on which the resource de-allocation is conditioned. LSAQ **80** may free resources as soon as operations are executed or once entries for the operations and/or data have been stored in LRQ **87** or SRQ **88**. Control logic within LS slice **44** communicates with DARQ **78** in the particular execution slice ES**0**-ES**7** issuing the load/store operation(s) to coordinate the acceptance of operands, addresses and data. Connections to other load-store slices are provided by AGEN bus **38** and by write-back routing network **37**, which is coupled to receive data from data cache **82** of cache slice **46** and to provide data to a data un-alignment block **84** of a another slice. A data formatting unit **85** couples cache slice **44** to write-back routing network **37** via a buffer **86**, so that write-back results can be written through from one execution slice to the resources of another execution slice. Data cache **82** of cache slice **46** is also coupled to I/O routing network **41** for loading values from higher-order cache/system memory and for flushing or casting-out values from data cache **82**. In the examples given in this disclosure, it is understood that the instructions dispatched to instruction execution slices ES**0**-ES**7** may be full external instructions or portions of external instructions, i.e., decoded "internal instructions." Further, in a given cycle, the number of internal instructions dispatched to any of instruction execution slices ES**0**-ES**7** may be greater than one and not every one of instruction execution slices ES**0**-ES**7** will necessarily receive an internal instruction in a given cycle.

[0029] While the invention has been particularly shown and described with reference to the preferred embodiments thereof, it will be understood by those skilled in the art that the foregoing and other changes in form, and details may be made therein without departing from the spirit and scope of the invention.

What is claimed is:

1. A method of executing program instructions within a processor core, the method comprising:

receiving a stream of instructions including functional operations and load-store operations at an issue queue;

computing effective addresses of load operations and store operations;

issuing the load operations and store operations to a cache unit;

storing entries corresponding to the load operations and the store operations at a recirculation queue;

removing the load operations and store operations from the issue queue; and

subsequently reissuing one of the load operations or store operations to the cache unit from the recirculation queue if the one of the load operations or store operations is rejected by the cache unit.

2. The method of claim **1**, wherein the storing entries stores only the effective address of the load operations or store operations and for store operations, the value to be stored by the store operation.

3. The method of claim **2**, further comprising:

removing load operations from the issue queue once the effective address is written to the recirculation queue; and

removing store operations from the issue queue once the effective address and the values to be stored by the store operations are written to the recirculation queue.

4. The method of claim **1**, further comprising:

removing load operations from the issue queue once the effective address is written to the recirculation queue; and

issuing the store operations and the values to be stored by the store operations to the cache unit before removing the store data from the issue queue.

5. The method of claim **1**, wherein the issuing issues the load and store operations to the cache unit in the same processor cycle as the storing entries stores the effective address of the load or store operation in the recirculation queue.

6. The method of claim **1**, wherein the cache unit is implemented as a plurality of cache slices to which the load and store operations may be routed via a bus, and wherein the reissuing of the load operations or the store operations is directed to a different cache slice than another cache slice that has previously rejected the load operations or the store operations.

* * * * *