PPE 42
Embedded Processor Core

Microarchitecture

Version 1.4

February 19, 2015

IBM Corporation
Systems & Technology Group
11400 Burnet Road
Austin, Texas 78758
c/o Bishop Brock, bcbrock@us.ibm.com
c/o Michael Floyd, mfloyd@us.ibm.com

IBM Confidential

Disclaimer

This document was a working document during development of the PPE 42 core microarchitecture. It may no longer be accurate or reflect late changes in the design. The definitive microarchitechtural specification is now the semi-formal specification found at the following URL:

https://aussvn.austin.ibm.com/svn/pgpdoc/tp_p9/spec/head/PPE%20Specification

Source Documents

The source documents for this document can be accessed via the following URL:

aussvn.austin.ibm.com/svn/pgpdoc/tp_p9/spec/head/PPE%20Microarchitecture

Note that change bars for sub-documents of Open Office master documents are lost when Open Office master documents are rendered into PDF. In order to view change markings of sub-documents is it necessary to view the Open Office sub-documents directly.

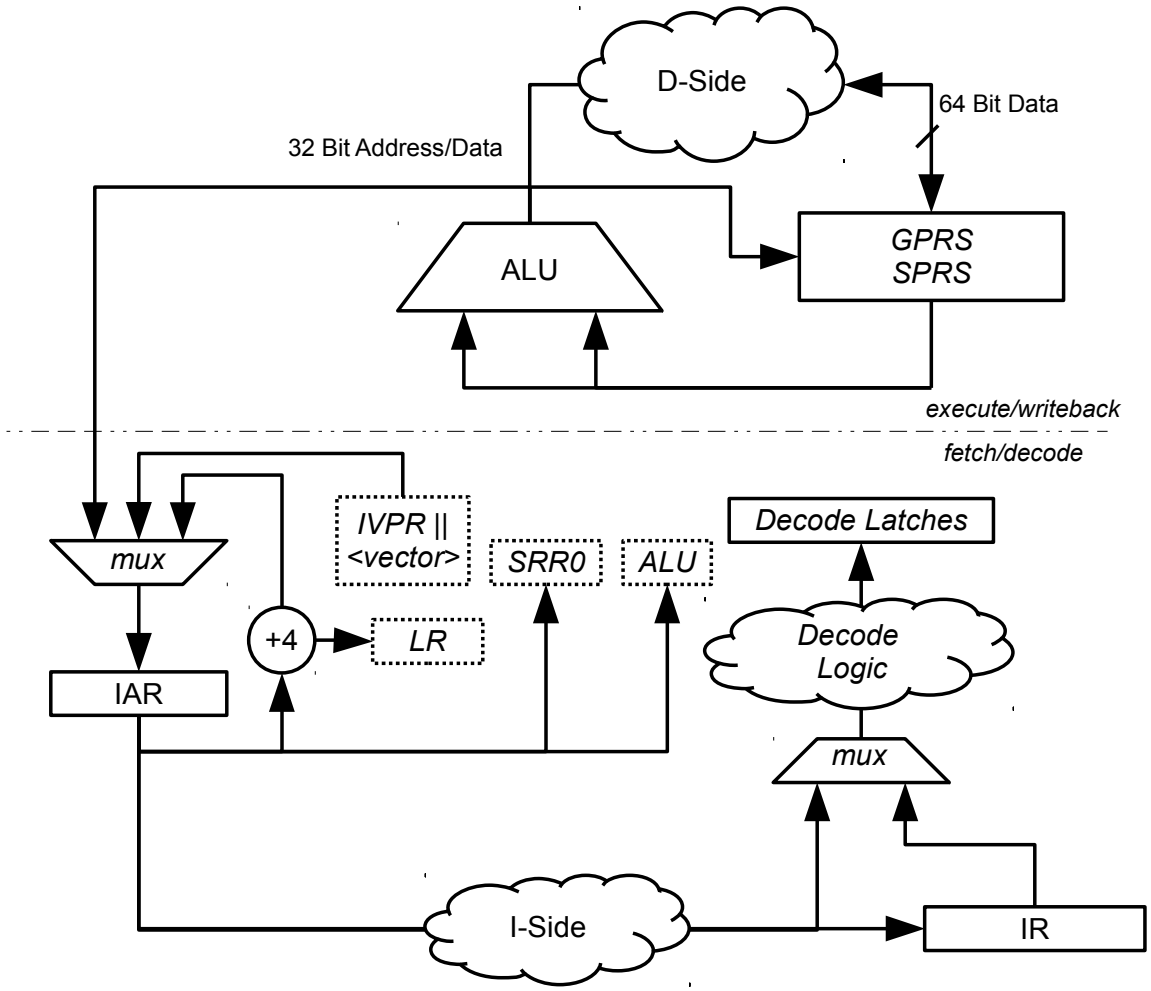
Revision History

Version	Date	SVN Revision	Changes	Author
1.4	02/19/2015	Head	• Clarified asynchronous (including external interrupt) exception processing.	mfloyd
1.3	07/22/2014	Head	• Corrected the initial block diagram • Changed DBSR to ISR • Many edits for specification changes occurring over the last several months.	Bishop Brock
1.2	4/10/2014		• Renamed Class T (Move To) to Class M; Added Class T (trap) • Added column showing clearing condition of hidden state • Reduced the amount of hidden state • State table revisions as discussed in review meetings • Other updates from review meetings	Bishop Brock
1.1	04/04/2014		• Modified definition of the halted state to include not having a pipelined instruction. • Added comments about IR update • Added comments about how IR gets into EDR • Minor editorial changes	Bishop Brock
1.0	04/02/2014		First version of the final plan-of-record detailing that the core will be implemented as a simple 2-phase model.	Bishop Brock

Table of Contents

1 Introduction

Illustration 1: PPE 42 Core: Very High-Level Block Diagram



The illustration on the preceding page is a very high-level block diagram of the PPE microarchitecture.

On the fetch/decode side, the IAR drives out through the I-side of the memory interface, gets an instruction and decodes it in a single cycle. The IR is mainly used for ramming, however we continue to update the IR during normal fetch since if there is a program exception we need to hold the IR until interrupt prioritization, so that it can be copied to the EDR. The IR is written with the instruction returned from the I-side only when the I-side ACKs an instruction fetch request. The IR can also be written directly to initiate ramming.

The IAR is supported by a dedicated incrementer. The IAR is incremented only when non-branching instructions complete. The IAR feeds the ALU to support branch computations. Branches are handled by writing the IAR with the branch target if the branch is decided as taken.

Since the interrupt vector computation is a simple concatenation, and there is otherwise no need for the vector arguments in the data path, the interrupt vector is directly loaded into the IAR to make the interrupt branch. The IAR is also written directly to SRR0 when an interrupt branch occurs.

On the execute/writeback side, GPRs/SPRs drive the ALU and back into the GPRs/SPRs in 1 cycle. For data operations, the address computation (ALU adder) is fast-pathed through the data side to support loading data back into the GPRs in a single cycle. The ALU provides 32 bits for GPR/SPR update, as a data address, and for new IAR computations for branches. The D-side provides a 64-bit interface to the GPRs for loads and stores.

Note that as a simplification, I-side and D-side transactions are mutually exclusive. The PPE core can sustain CPI = 1 for arithmetic and compare instructions with an infinite single-cycle I-side. The PPE core can sustain CPI = 2 for load/store/dcb* in the presence of an infinite single-cycle D-side.

2 Architected State

The following table details the PPE 42 core architected state. This is the PPE 42 core state that is visible to PPE 42 programs, and to external users as External Interface Registers (XIRs).Table 1: PPE 42 Architected State

Table 2: PPE 42 Register State

Register(s)	This Register is Driven by	This Register Drives	Notes
GPRs	ALU	ALU	GPRs can be considered as a register file with 4 32-bit read ports and 3 32-bit write ports. The read ports are used for arithmetic, “move-to” and stores. Four read ports are required to support **stvdx**. One write port is used for arithmetic, “move-from” and update-form addressing. The other 2 write ports are used for loads. The architecture guarantees that the 3 write addresses are always unique.
CR, XER	ALU	GPRs Control Logic	CR and XER are tightly coupled with ALU operations, and can also be read and written as SPRs. CR controls branching.
CTR	ALU Self (-1)	ALU Self (-1)	CTR is a decrementer with a special (CTR == 1) circuit used to decide branches involving CTR.
LR	ALU IAR	ALU	LR is involved with branching and has a special direct connection to be loaded from the IAR (+4) for branch-and-link.
DACR	ALU	ALU Control logic	DACR contents are compared with instruction and data addresses to signal debug events.
DBCR	ALU	ALU Control logic	DBCR controls debug event handling. All debugug events halt the processor.
DEC	ALU Timer inputs	ALU TSR	DEC decrements on every cycle the selected decrementer input is '1'. DEC is read and written as a normal SPR. When DEC_0 transitions from 0 → 1, TSR[DIS] is set.
EDR	ALU IR Memory Interface	ALU External	EDR is loaded from the ALU by **mtspr**. EDR is loaded directly from the IR for program exceptions to facilitate debug. EDR is loaded directly from the memory interface to provide the failing data address for data machine check, data storage and alignment interrupts. EDR is also read-only as an XIR.
IAR	ALU Self (IAR + 4) IVPR || <vector>	ALU SRR0 External Self (IAR + 4)	During execution the IAR is always the NIA. When halted the IAR always points to the instruction to execute next once processing resumes. To simplify exception handling SRR0 is loaded directly from the IAR, and IAR is loaded directly with IVPR || vector.
ISR	Control logic	ALU Control logic	Bits in ISR are set during interrupt processing.
IVPR	External	ALU IAR	
MSR	ALU Control logic Memory Interface	ALU Control Logic Memory Interface	MSR has many special functions and connections. MSR bits are cleared during exception handling, based on the exception. MSR[SEM] and MSR[IPE] are presented to the memory interface. MSR[SIBRC] and MSR[SIBRCA] are updated by the memory interface. MSR bits ME, UE and EE control exception processing. MSR[EE] has a special update from the RDR to implement **wrtee** and **wrteei**.

Continued on next page

Table 3: PPE 42 Register State (Cont.)

Register(s)	This Register is Driven by	This Register Drives	Notes
PIR	External	ALU	A read-only SPR to the core; identifies the specific PPE instance.
PVR	Constant	ALU	A generic constant, read-only to the core.
SPRG0	ALU External	ALU External	SPRG0 is read/write as an XIR.
SRR0	ALU IAR	ALU	SRR0 has a special connection that allows it to be loaded directly from IAR during exception processing. **rfi** decodes similarly to a “branch-to-SRR0” so in this case SRR0 moves through the ALU.
SRR1	ALU MSR	ALU MSR	SRR1 moves through the ALU for **mfsrr1** and **mtsrr1**, however for exception processing and **rfi** SRR1 is loaded from and written to the MSR directly.
TCR, TSR	ALU (Special) Timer inputs	ALU Control logic	Timer control and status. DEC, FIT and WDT events set bits in TSR. TSR implements special write-1-to-clear semantics for status bits, however these bits are directly writable while ramming.
XCR, XSR	External Control Logic	External Control Logic	XCR contains bits controlling reset, halt, single-step and RAM. XSR mirrors DBSR and other status bit externally.

3 Hidden State

Due to the pipelined nature of the PPE 42 core, a great deal of non-architected state is required, especially the ALU decode information. Proper interrupt prioritization and handling requires there to be a bit of state for most of the interrupt types. This hidden state is not cleared until after interrupts are prioritized.

The table below summarizes the type of hidden state required to implement PPE 42 core state machine.

Table 4: PPE 42 Core State Machine Hidden State

State	Associated With or Recognized During	Clearing Condition (If applicable)	Notes
ALU operation	**home**	It should not be necessary to clear all decode latches, as long as any "action" latches are cleared when the action takes place.	Decode latches that record ALU input sources and operations. All ALU function is controlled by these latches; there are no special-case or fast-path overrides.
Writeback operation			Decode latches that record the writeback operation. These latches are set during instruction decode and do not change.
Instruction Type			There are several categories of instructions with different processing requirements. These latches are set during instruction decode and do not change.
pipelined		Always cleared the cycle the action is taken, unless it is also set on that cycle.	The previous fetch decoded as a pipelined arithmetic or branch, which is completed concurrent with the current fetch.
Instruction Machine Check Event		**vector -----> home**	Instruction machine checks are recognized in the **home** state. ISR and EDR are updated immediately, but may be overwritten.
Instruction Storage Event		**vector -----> home**	ISI events are recognized in the **home** state. They may get prioritized out so the hidden state is needed.
External Event		**vector -----> home**	The external event is indicative, but not definitive. For numerous reasons, the external input can glitch. Therefore it would be possible to begin interrupt processing with an external input event present, but find the external status has disappeared at the end of synchronization. In general this can never be avoided except by careful programming. Therefore this bit is latched each time a check is made for pending asynchronous exceptions, and maintains latched until hidden state bits associated with exceptions are cleared after prioritization. Note that the DEC, FIT and WDT status are safe and can not glitch, only the external input can glitch. Therefore we don't need hidden status for the timer interrupts as their presence is easily computed by a simple AND of their status and enable.
Program Event		**vector -----> home**	The program exception is recognized and handled during **home.** We need hidden state bits because we can't update ISR until and unless we actually take the program interrupt after prioritization. We can set ISR[PTR] from these two bits. Note we can update EDR from the instruction interface if a program event happens, and if synchronization uncovers a machine check it will just get overwritten.
trap Event			

Continued next page

Table 5: PPE 42 Core State Machine Hidden State (Continued)

State	Associated With or Recognized During	Clearing Condition (If applicable)	Notes
Data Machine Check Event	**data**	**vector -----> home**	Data machine check events are recognized and handled in the **data** state. ISR and EDR are updated immediately.
Data Storage Event			DSI events are recognized and handled in the **data** state
Alignment Event			Alignment events are recognized and handled in the **data** state
Load/Store indicator			Data storage and alignment need a load/store indicator for ISR, which must be held for prioritization.
System Reset Event	**reset**		Used to prioritize interrupt handling.
Hard Reset			Indicates whether a system reset is “hard” or “soft”

4 Instruction Handling

The majority of instructions require no special handling, and are pipelined with a maximum completion rate of CPI = 1 for arithmetic, compare, **mfspr**, **mtcr0** and **mfcr0**, and CPI = 2 for data operations. The table below details at a high level how each class of instructions is handled.

Note that cycle times for instructions requiring either form of synchronization delay include a minimum of 1 cycle for generating a synchronization transaction. Extra synchronization delay is only required if the memory interface does not respond to the synchronization request in 1 cycle.

Table 6: PPE 42 Core Instruction Handling

Instruction Class	Instruction(s)	Notes
A Arithmetic	Arithmetic, compare, **mfspr, mfcr0, mtcr0**	No special handling, these instructions can not cause exceptions. Single-cycle pipelined execution.
D Data	Load/Store/**dcb***	Potentially require blocking in the **data** state while waiting for memory. May cause DAC debug halts, Data Storage and Alignment Interrupts. May cause or report precise and imprecise data machine check interrupts. 2 cycle + memory-delay.
B Branch	**b, bc, bcctr, bclr**	All branches are fully resolved prior to fetching the next instruction – PPE 42 does not predict branch direction. Branches are decided in the **serial** state. When returning to **home** from **serial**, only taken branches update the IAR with the branch target.
F Fused	Fused compare-branch	The fused compare-branch instructions are not pipelined. They execute serialized in 3 cycles. These branches are decided in the **fused** state.
M move To	**mtspr, wrtee, wrteei**	As a simplification all **mtspr** are handled specially and not pipelined. Many SPRs control exceptions, and we prefer a sequential execution model for instruction flow. The architecture does not require **wrtee** and **wrteei** to be context synchronizing but these may cause exceptions so they are handled here as well. Note that these instructions do not require synchronization. **mtspr**, **wrtee** and **wrteei** are simply delayed to allow any asynchronous exceptions that they may uncover to appear before fetching the next instruction. 2 cycles. **wrteei EE** decodes as **addi 0, 0, EE** with no GPR read or writeback. **wrtee Rn** decodes as **addi 0**, **RN**, **0** with no writeback. In both cases RDR[16] is copied directly to MSR[16] during **serialize**. Note that MSR[SIBRC] and MSR[SIBRCA] continue to update during execution of the **wrteei** and **wrtee** instructions.
C Context synchronizing	**mfmsr**	**mfmsr** is required to be execution synchronizing. In PPE it is context synchronizing as well. Because of the fact that parts of the MSR are updated by the memory interface, the memory interface must insure that the MSR is up-to-date before reporting completion of PIB data operations. 3 cycles + execution synchronization delay.
	mtmsr	**mtmsr** is required to be execution synchronizing. In PPE it is context synchronizing as well. **mtmsr** stalls for execution synchronization then does the normal writeback. 3 cycles + execution synchronization delay
	rfi	**rfi** is required to be context synchronizing. **rfi** is decoded as a branch to SRR0, and SRR1 ← MSR directly. 3 cycles + execution synchronization delay.
S Sync	**sync**	**sync** stalls the pipeline for storage synchronization. For simplicity, **sync** is decoded as a **nop** and completes normally after synchronization. 3 cycles + storage synchronization delay.
T Trap	**trap**	**trap** is never executed, but instead always causes an exception or halt.

5 Exception Handling

Asynchronous and synchronous exception handling are disjoint, and unlike the Power ISA, all asynchronous exceptions have lower priority than synchronous exceptions.

Asynchronous exceptions are recognized only “between” instructions, technically before execution of the next instruction. PPE 42 never partially executes an instruction, and makes every attempt to complete every instruction fetched. If the state machine enters interrupt prioritization with an asynchronous interrupt pending, this can only be overridden by an imprecise machine check uncovered during execution synchronization.

Synchronous exceptions are recognized during instruction execution. If the state machine enters interrupt prioritization with a synchronous exception pending, there is no need to also check asynchronous exceptions, and the synchronous exception can only be overridden by an imprecise machine check uncovered during execution synchronization.

Asynchronous exceptions are only recognized during the **home** state. If an asynchronous event is pending and not overridden by a debug halt, then the fetch is aborted, and hidden state records the event for interrupt prioritization.

The **data** state may cause data storage and precise machine checks, or allow imprecise data machine checks to be reported. The **sync** state will also report any pending imprecise data machine checks. Imprecise data machine checks are only reported during the **data** and **sync** states; the memory interface must hold imprecise machine checks until the next data access or synchronization event from the PPE 42. Note that by definition, all potential imprecise machine check sources must be reported before an execution synchronizing event (any interrupt or synchronizing instruction is allowed to proceed).

5.1.1 EDR and ISR handling

Note that in the event of an instruction or data machine check, the EDR and ISR status bits are updated simultaneously with the reporting of the event by the memory interface. This is valid because machine checks are always the highest priority interrupt outstanding, so these updates will never be incorrect, even though technically interrupt prioritization does not take place until after execution synchronization. If multiple data machine checks are presented the EDR and ISR may be updated again to reflect the latest machine check reported.

Data storage and alignment interrupts also immediately update EDR to avoid having to hold 32 bits of state for interrupt prioritization. However if a data machine check is uncovered during prioritization, the EDR will be overwritten with the machine check address.

Program interrupts also immediately update EDR to avoid having to hold 32 bits of state for interrupt prioritization. However if a data machine check is uncovered during prioritization, the EDR will be overwritten with the machine check address.

The ISR status bits associated with instruction storage, data storage, alignment and program events are not updated until after interrupt prioritization however, so hidden state is required to hold this status.

6 Halting

The core is said to be in the halted state whenever

- State == **home**
- XSR[HCP] = '1'
- XSR[SIP] = '0'
- XSR[RIP] = '0'
- Hidden state bit “pipelined” is '0'.

Halting never synchronizes the processor core. An interrupt halt will synchronize the core as a natural part of interrupt processing, but the other halts stop the core immediately and present the core for debugging with the exact architected state that existed at the completion of the last completed instruction.

The core state machine only checks for a halted processor in the **home** state. If the processor is halted and we are not single stepping (XSR[SIP]] = '0') or ramming (XSR[RIP] = '0'), the fetch is aborted and the machine remains in the **home** state.

The single-step and ramming state is cleared before completion of the single-stepped or rammed instruction, so that when the state machine returns to the **home** state it will remain **home** (because XSR[HCP] = '1' when single stepping or ramming). Specifically, the XSR[SIP] bit is cleared when the fetch of the single-stepped instruction is ACKed by the I-side. The XSR[RIP] bit is cleared on the cycle that the instruction would normally update the IAR to IAR + 4. When ramming XSR[RIP] = '1' always suppresses this update

IAC and DAC debug halts are ignored when ramming.

The IAR reported in the halted state depends on why the processor halted:

- If the processor halts due to an IAC or DAC debug event, the IAR addresses the instruction causing the event.
- If the processor is halted by external command, of if single-stepping, the IAR addresses the next sequential instruction.
- If the processor halts due to a machine check while MSR[ME] = '0', an unmaskable interrupt while MSR[UE] = '0', the IAR addresses the first instruction of the associated interrupt handler.

6.1 Force Halting

Force halting clears all hidden state,

Force halting trumps even reset.

7 System Reset

System reset is treated like an interrupt that is taken immediately, regardless of the current state of the PPE 42 core. Conceptually there is an arc from every state (including the **reset** state itself!) to the **reset** state.

Reset is a special transaction type to the memory interface. Two types of reset are supported: A “soft” reset that instructs the memory interface to allow pending transactions to complete before ACKing, and a “hard” reset that forces an immediate reset.

System reset is implemented by a single state in the state machine; soft vs. hard is represented by a hidden state bit.

Reset behavior is fully specified in the PPE 42 User's Manual.

8 State Encoding

Number of bits and encoding is TBD during logic design. The only requirement is that state 0 is illegal.

9 References

The *PPE 42 Core User's Manual* contains detailed user-level descriptions of the following behavior:

- Exception and interrupt handling
- Single-stepping and ramming procedures
- Behavior during ramming
- Synchronization
- Reset state changes and sequencing

To avoid possible inconsistencies if these descriptions were duplicated, those sections are referred from this document as the documentation of record.

10 State Machine

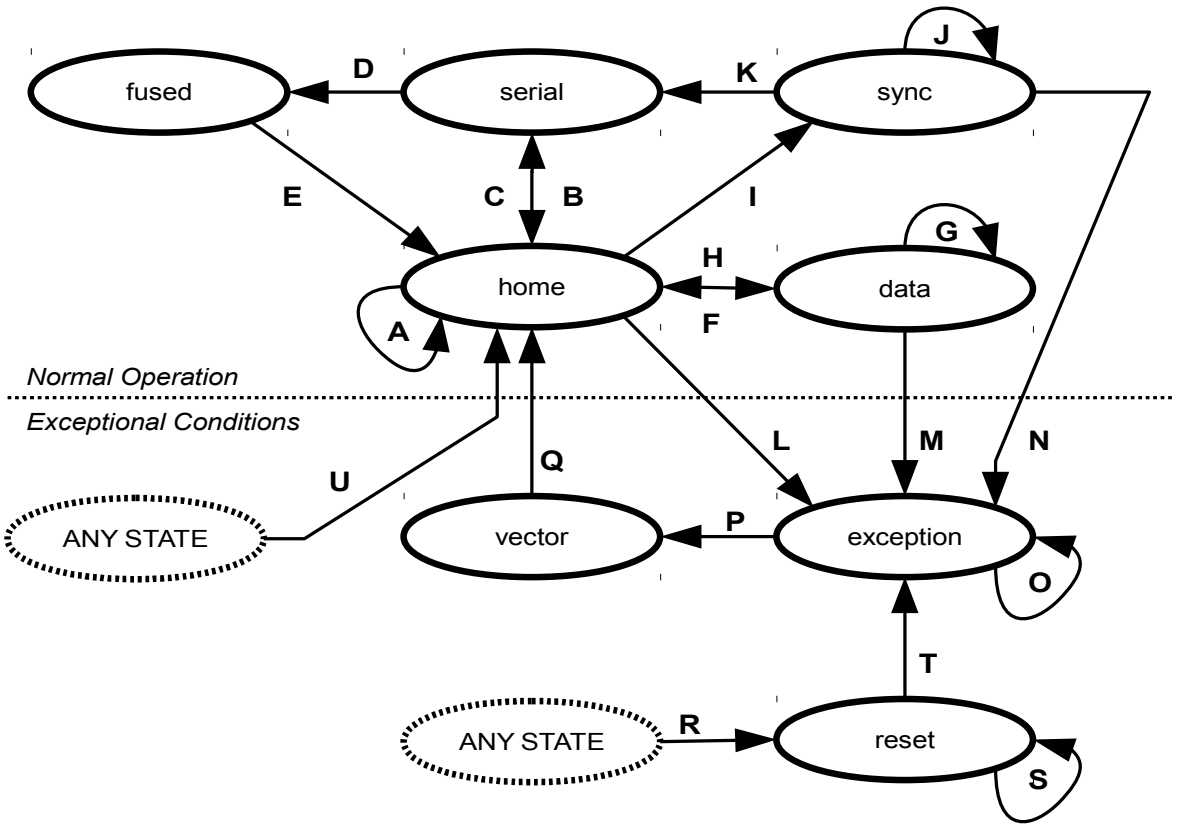
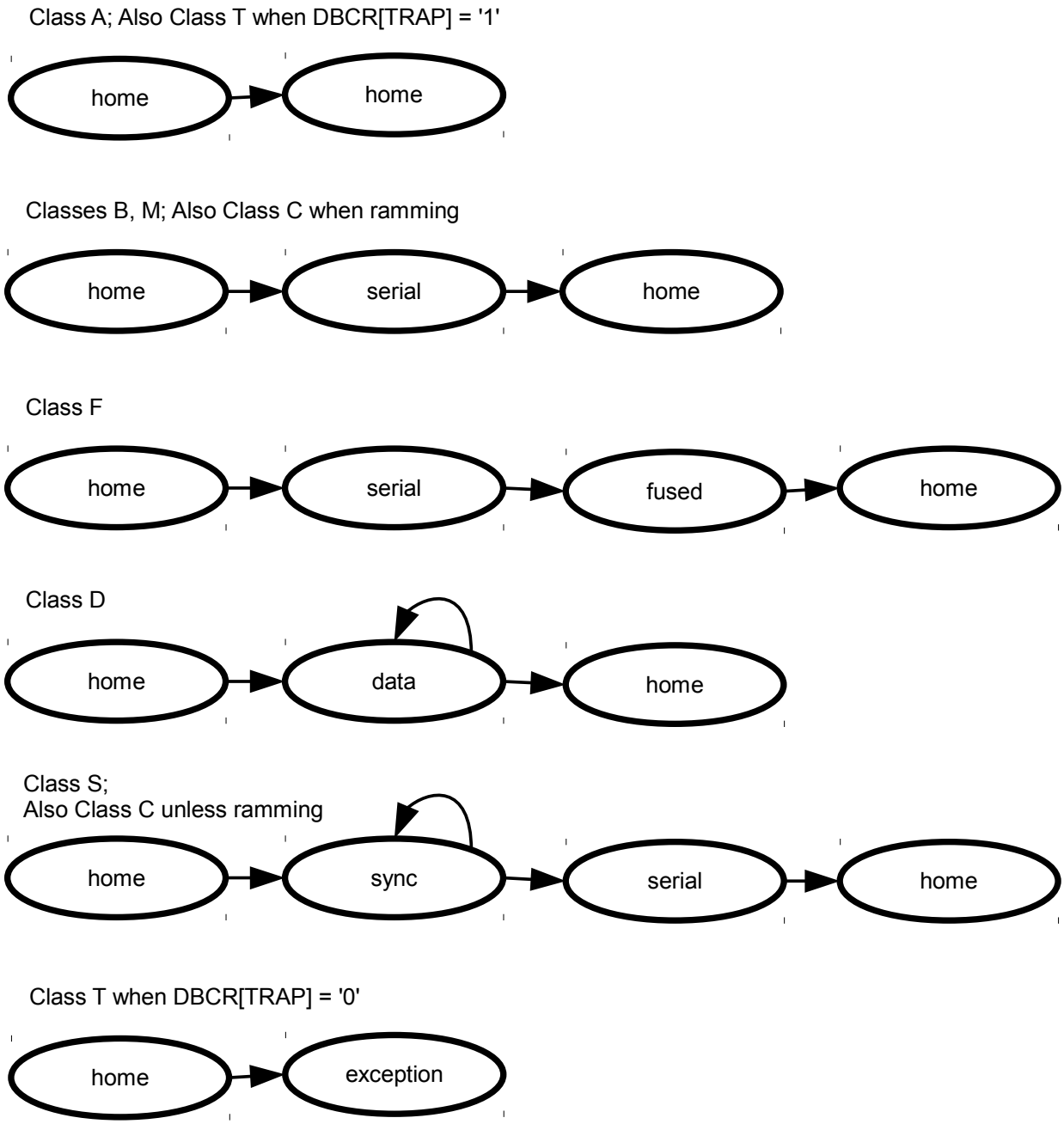


Illustration 2: PPE 42 State Machine

Table 7: PPE 42 State Transitions

#	From	To	Notes
A	home	home	I-fetch nack, or ack a Class A instruction. Also spin here while halted or MSR[WE] = '1' Also return here for TRAP, IAC, DAC debug halts
B	home	serial	I-fetch ack Class B, F, or M instruction When ramming, Class C goes this way as well
C	serial	home	Class B, M, C and S complete
D	serial	fused	Compute branch address for Class F
E	fused	home	Class F completes
F	home	data	Fetch ack a Class D instruction
G	data	data	Class D waits for memory
H	data	home	Class D completes
I	home	sync	Fetch ack Class C or S When ramming, only class S goes this way, Class C goes to **serial**
J	sync	sync	Class C or S waits for synchronization
K	sync	serial	Go complete Classes C and S
L	home	exception	Fetch or decode phase exception, including non-halting **trap**
M	data	exception	Memory exception
N	sync	exception	Imprecise data machine check uncovered by synchronization
O	exception	exception	Waiting for synchronization
P	exception	vector	Sync done; Prioritize interrupts and compute vector
Q	vector	home	Resume after exception
R	ANY State	reset	Reset. This arc is trumped by a force halt.
S	reset	reset	Wait for reset ack from MIB
T	reset	exception	Take the SRESET exception
U	ANY State	Home	Force halt. This arc trumps arc R (reset).



State traversals by instruction class, assuming a fetch-acknowledgment in the first home state

11 State Tables

The tables in this section detail the low-level operations of the core. Conditions and state transitions are prioritized, and must be processed in priority order. Mutually exclusive conditions are always considered at the same priority.

11.1 home

At entry to the **home** state a hidden state bit may indicate that a pipelined Class A instruction was decoded on the previous cycle. If so, the pipelined instruction is always completed concurrent with the fetch or other state transition.

The **home** state effectively has 3 “substates”:

- **default** : Normal execution
- **step** : Single stepping, i.e., XSR[SIP] = '1'. XSR[SIP] is always cleared once the single-step fetch completes or takes an error.
- **ram** : XSR[RIP] is set when the IR is written. XSR[RIP]is cleared here for instructions that complete without exception, however for other instructions XSR[RIP] remains set to squash the subsequent IAR update, and is cleared when the instructions complete.

Ramming ignores most conditions and simply decodes and executes the instruction held in the IR.

The IAR is only updated for **default** and **step** fetches when the instruction will complete without an exception. For Class D and S instructions we wait until they complete without exception before updating the IAR. This eliminates the need for IAR rollback.

There is no “wait” state. If MSR[WE] = '1' then under normal conditions we spin here waiting for something that allows the state machine to continue. If single stepping we never leave home.

Table 8: The **home** state

Priority	Substates	Condition	Squash Fetch?	Next State	Updates / Special	Notes
ANY					Any pipelined instruction is completed.	
1	**ram**	Illegal decode, or **trap** with DBCR[TRAP] = '0'	Yes	**exception**	Exception hidden state; EDR ← instruction;	
	ram	**trap** with DBCR[TRAP] = '1'	Yes	**home**	XSR[HCP] ← '1' XSR[HC] ← '100' XSR[TRAP] ← '1'	
	ram	IR decodes as Class A	Yes	**home**	Set “pipelined” hidden state XSR[RIP] ← '0'	
	ram	IR decodes as Class B, F, M, C	Yes	**serial**	XSR[RIP] ← '0'	
	ram	IR decodes as Class D	Yes	**data**	XSR[RIP] unchanged	
	ram	IR decodes as class S	Yes	**sync**	XSR[RIP] unchanged	
2	**default**	XCR[HR] = '1'	Yes	**home**		Halted and not single-stepping
3	**default step**	Asynchronous exception pending	Yes	**exception**	Set asynchronous exception hidden state	All of these transitions clear XSR[SIP]
4	**default step**	MSR[WE] = '1'	Yes	**home**		
5	**default step**	IAC debug Halt	Yes	**home**	XSR[IAC] ← '1' XSR[HCP] ← '1' XSR[HC] ← '100'	
6	**default step**	I-fetch NACK	No	**home**		
	default step	I-fetch error	No	**exception**	Set hidden state bits for exception; ISR updates for machine check.	
	default step	Illegal decode, or **trap** with DBCR[TRAP] = '0'	No	**exception**	Set hidden state bits for exception; EDR ← instruction	
	default step	**trap** with DBCR[TRAP] = '1'	No	**home**	XSR[HCP] ← '1' XSR[HC] ← '100' XSR[TRAP] ← '1'	
	default step	I-fetch ACK decodes as class A	No	**home**	IAR += 4; Set “pipelined” hidden state bit	
	default step	I-fetch ACK decodes as class B, F, M	No	**serial**	IAR += 4	
	default step	I-fetch ACK decodes as class D	No	**data**	IAR unchanged	
	default step	I-fetch ACK decodes as class C, S	No	**sync**	IAR unchanged	

11.2 data

The **data** state implements memory transactions. At entry the ALU has been programmed to compute the data address.

The **data** state effectively has 2 “substates”:

- **default** : Normal execution
- **ram** : XSR[RIP] is set when the IR is written. XSR[RIP] is cleared here when the instruction completes.

The IAR is updated here only if the instruction completes without an exception. This eliminates the need for IAR rollback.

Table 9: The **data** state

Priority	Substates	Condition	Squash Data Transaction ?	Next State	Updates / Special	Notes
1	**default**	DAC compare debug halt	Yes	**home**	XSR[HCP] = '1' XSR[HC] ← '100' XSR[DACR, DACW] updates	
2	**default** **ram**	D-side NACK	No	**data**		
	default **ram**	D-side error	No	**exception**	Set exception hidden state; ISR and EDR updates for machine checks; EDR update for data storage and alignment.	
	default **ram**	D-side ACK	No	**home**	Commit store data and/or GPR update to GPRs; If ~XSR[RIP] then IAR += 4 XSR[RIP] ← '0'	

11.3 serial

The serial state is used to serialize non-arithmetic, non-data instructions that can not be pipelined. XSR[RIP, SIP] are cleared here for instructions that complete here.

Table 10: The **serial** state

Priority	Condition	Next State	Updates / Special	Notes
1	Class F instruction	**fused**	Update CR[CR0] from ALU. Set ALU decode to compute the fused compare-branch target.	
	Class B instruction	**home**	Update IAR from ALU if taken	
	Class M, S instructions	**home**	Commit GPR/SPR updates from the ALU	
	Class C other than **rfi**	**home**	Commit GPR/SPR updates from the ALU	
	rfi	**home**	Update IAR from ALU; MSR ← SRR1	**rfi** moves SRR0 through the ALU.

11.4 fused

The **fused** state exists to implement the branch address computation for fused compare-branch instructions.

Table 11: The **fused** state

Priority	Condition	Next State	Updates / Special	Notes
1	All	**home**	Update IAR if branch taken	

11.5 sync

The **sync** state implements the special synchronizing memory transactions. The **sync** state does a “heavy-sync” for Class S and a “light-sync” for Class C.

Table 12: The **sync** state

Priority	Condition	Next State	Updates / Special	Notes
1	D-side NACK	**sync**		
	D-side error	**exception**	Set exception hidden state; ISR and EDR updates for machine checks.	
	D-side ACK	**serial**		

11.6 exception

The **exception** state exists to perform execution synchronization for interrupt handling. The **exception** state does a “light-sync” transaction.

Table 13: The **exception** state

Priority	Condition	Next State	Updates / Special	Notes
1	D-side NACK	**exception**		
	D-side error	**vector**	Set exception hidden state; ISR and EDR updates for machine checks	
	D-side ACK	**vector**		

11.7 vector

The **vector** state exists to prioritize all pending interrupts and update the IAR with the interrupt vector. SRR0 gets IAR directly, and the MSR updates for the taken interrupt are also programmed here. The table below discusses conditions *after* interrupt prioritization

Table 14: The **exception** state

Priority	Condition	Next State	Special	Updates
1	Machine check interrupt and MSR[ME] = '0', or other unmaskable interrupt and MSR[UE] = '0'.	**home**	XSR[HCP] ← '1' XSR[HC] ← '011'	Any ISR updates from hidden state. All interrupt hidden state cleared. SRR0 ← IAR IAR ← IVPR || <vector>. MSR cleared except for MSR[ME], machine check also clears ME.
	Otherwise	**home**		

11.8 reset

There is an implicit transition to reset from every state. There are 2 types of reset, soft and hard, which are presented externally as special memory transactions.

The following reset actions must occur on the transition into the **reset** state from any other non-**reset** state:

- ISR[SRSMS] is updated from the current state machine state.

For further information on the **reset** state and the data updates see the PPE 42 User's Manual.

Table 15: The **reset** state

Priority	Condition	Next State	Updates
1	Reset NACK	**reset**	
	Reset ACK	**exception**	Set reset exception hidden state bit