IBM CONFIDENTIAL

POWER2 Performance Monitor Definition

Version 1.02

Prepared By:
Edward H. Welbon
E26/904−4D018
678−1827

IBM Corporation
ADVANCED ENGINEERING SYSTEMS
11400 Burnet Road
Austin, Texas 78758

January 18, 1993

TABLE OF CONTENTS

FIGURES

Introduction

The Rios 2 Performance Monitor is a software accessible facility intended to provide detailed information concerning the utilization of Rios 2 instruction execution and storage control. The monitor consists of twenty two 32–bit counters to be used to count CPU/Storage performance related events. In addition to the counters, there is a Monitor Mode Control Register. These counters physically reside on the Storage Control Unit (SCU) chip and are addressable for read or write via Programmed I/O (PIO) operations in manner similar to the current RIOS/1 SCU control registers.

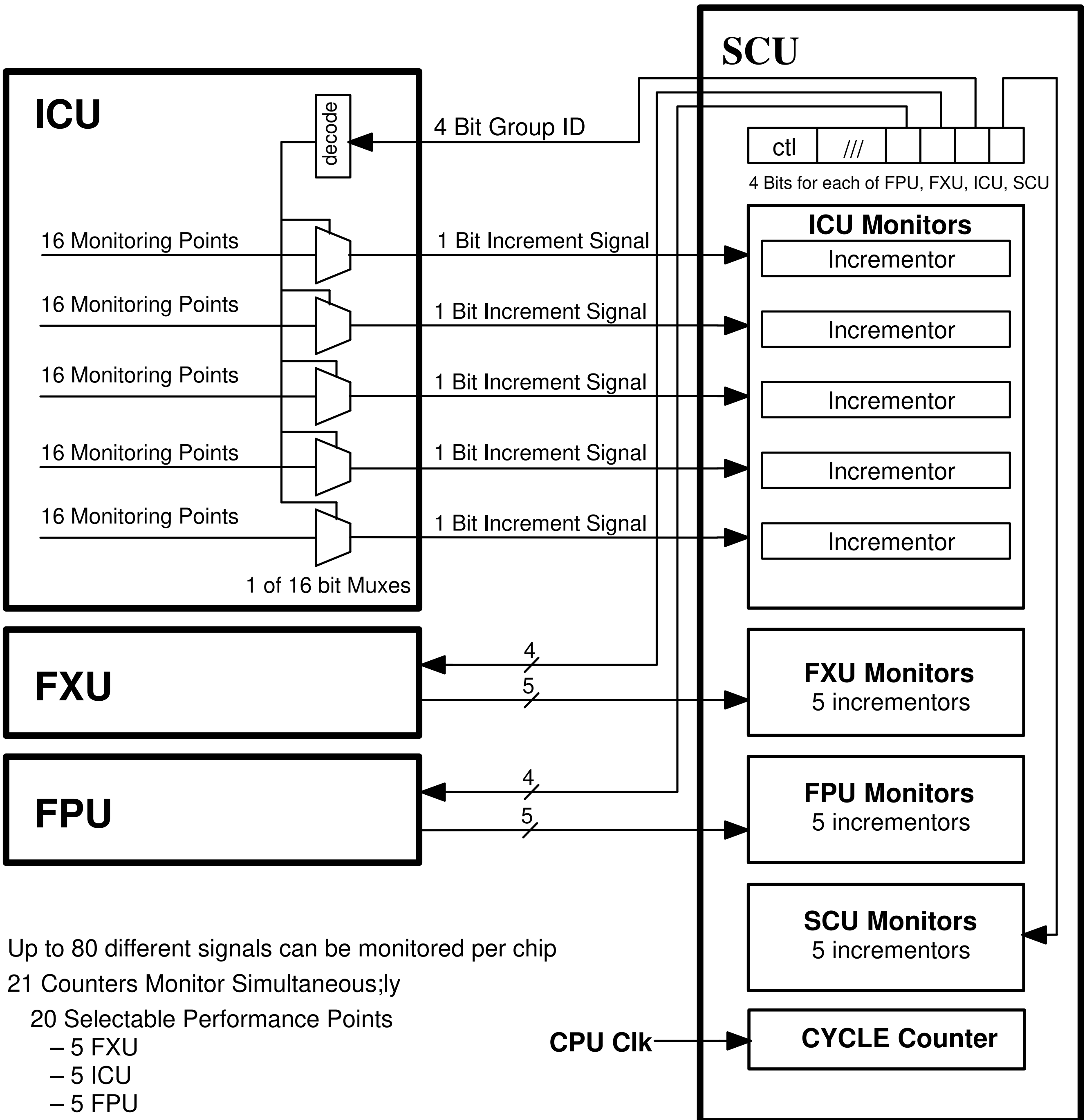
The Rios 2 Central Electronics Complex (CEC) is composed of four chips, the Fixed Point Unit (FXU), the Floating Point Unit (FPU), the Branch Unit (ICU) and the Storage Control Unit (SCU). Each of these four chips has access to five counters for a total of twenty. In addition to these twenty counters, there are two additional counters, a CPU cycle counter to count hardware clock ticks, and a memory system soft error counter for a total of twenty two.

For each of the ICU, FPU, SCU and FXU chips there is a four bit control field in the Monitor Mode Control Register (MMCR) that selects one of sixteen possible sets of five events(signals) to be recorded(counted) by the counters. For each chip, it is possible to choose any one of sixteen groups of five events for monitoring. The hardware event count source selection for any chip is independent of the selection of the other three chips.

All twenty two counters operate concurrently. To reduce the intrusion incurred by reading these counters it is recommended that these registers be read/written with the load/store multiple instruction. In the worst case (ignoring tlb and page miss, but counting cache miss) engineering has estimated that sixty cycles are required to save twenty three GPRs, load twenty two counters (and the control register), store out twenty two counters (and the control register) and then reload twenty three GPRs.

A bit has added to the Rios 2 processor Machine Status Register (MSR) that will control the state of the Monitor. Careful use of this bit will reduce the intrusiveness of the Monitor to negligible levels for most cases of importance. By incorporating the control of the monitoring function in the MSR, this control becomes part of process state. Thus the control of the monitor is handled in an essentially automatic fashion in a manner modelled after the control of the Floating Point Available state.

Organization Of Monitor Hardware



Up to 80 different signals can be monitored per chip

21 Counters Monitor Simultaneous;ly

- 20 Selectable Performance Points
  - – 5 FXU
  - – 5 ICU
  - – 5 FPU
  - – 5 SCU
- 1 Non–Selectable Performance Point
  - – 1 Cycle Counter

Monitor Hardware

Facilities Defined

(1) 1 Monitor Mode Control Register (MMCR).
(2) CPU cycle counter
(3) 5 Counters dedicated to SCU
(4) 5 Counters dedicated to ICU
(5) 5 Counters dedicated to FPU
(6) 5 Counters dedicated to FXU
(7) Soft error counter

These counters/registers are physically located on the SCU, are 32 bits in length and accessible via PIO with the standard SCU BUID. The standard BUID allows privileged accesses only (read/write). Reading these counters/registers does not change their content.

The SCU will provide three count mode busses, one for each of the ICU, FXU and FPU. These count mode busses will select the group of events that the ICU, FXU, and FPU will provide on their respective increment busses. The Monitor Mode Control Register (MMCR) will control the values propagated on the count mode busses.

The ICU, FXU and FPU will each provide a 5–bit increment bus to the SCU. Each bit of a given increment bus will serve to increment one of the five 32–bit counters allocated to that chips counters per system clock. Since all of the counters reside on the SCU, the SCU does not require external control or increment busses.

The SCU, ICU, FXU and FPU counters are incremented under control of the their respective count mode settings as reflected in the MMCR. The Soft Error Counter will be incremented by one when one or more DCU chips reports a memory soft error. The CPU cycle counter will increment on each tick of the CPU clock. The increment signal will be a level to allow counter updates per cycle. Each of the counters will increment by ’1’ only.

Certain events will require that their corresponding counts be incremented by more than ’1’ on some cycles. These events may require more than one counter to allow for correct counting. For example, up to six floating point instructions (including loads and stores) may executed per cycle. To correctly count this case three counters would be required; a counter for the one’s bit, a counter for the two’s bit and the four’s bit of the binary representation of the increment amount.

Suppose that in some cycle that the FPU executes three instructions; corresponding to the value three, the FPU would output the value B’011’ (on the proper three bits of the five bit increment bus). Corresponding to these three bits are three counters in the SCU that logically count the number of floating point instructions executed. Of these, the counters representing the two’s bit and the ones bit would be incremented.

If in some other cycle the FPU were to execute four more instructions (for a total of seven), the FPU would output the value B’100’. In this case the counter corresponding to the four’s bit would increment. If we let c4=four’s bit counter, c2=two’s bit counter and c1=ones bit counter then we would expect that c4=1, c2=1 and c1=1. The total actual count would be computed using total=4*c4 + 2*c2 + 1*c1 = 4*1 + 2*1 + 1*1 = 7 as expected.

Monitor Pio Load/Store Addresses

Adapter Specific	Effective Address	Name	Use
000000	00001100	mmcr	Monitor Mode Control Register
000000	00001104	pmc0	CPU cycle counter
000000	00001108	pmc1	SCU event counter 1
000000	0000110C	pmc2	2
000000	00001110	pmc3	3
000000	00001114	pmc4	4
000000	00001118	pmc5	5
000000	0000111C	pmc6	ICU event counter 1
000000	00001120	pmc7	2
000000	00001124	pmc8	3
000000	00001128	pmc9	4
000000	0000112C	pmc10	5
000000	00001130	pmc11	FPU event counter 1
000000	00001134	pmc12	2
000000	00001138	pmc13	3
000000	0000113C	pmc14	4
000000	00001140	pmc15	5
000000	00001144	pmc16	FXU event counter 1
000000	00001148	pmc17	2
000000	0000114C	pmc18	3
000000	00001150	pmc19	4
000000	00001154	pmc20	5

Monitor Mode Control Register

Organization Of Control Fields

Format Of The Monitor Mode Control Register (MMCR)

MONITOR MODE CONTROL REGISTER

DIS	DP	DU	DMS	DMR	SGA	//	FPU	FXU	ICU	SCU
0	1	2	3	4	5	6	16	20	24	28

This register will be set to zero on power up. Reading this register does NOT change its content. Bits 6 though 15 are reserved, and always read as zeroes. To maintain compatibility with future implementations, software must NOT write nonzero values into bits 6 through 15. The named fields have the following definitions.

DIS: Disable counting unconditionally: This bit when set to a one, causes the counters to stop counting unconditionally. The counter values are not changed, only suspended.

DP: Disable counting when in privileged state: This bit when set to a one, causes the counters to stop counting when MSR(PR) is set to a zero. The counter values are not changed, only suspended.

DU: Disable counting when in problem state: This bit when set to a one, causes the counters to stop counting when MSR(PR) is set to a one. The counter values are not changed, only suspended.

DMS: Disable counting when PM is set This bit when set to a one, causes the counters to stop counting when MSR(PM) is set to a one. The counter values are not changed, only suspended.

DMR: Disable counting when MSR(PM) is clear This bit when set to a one, causes the counters to stop counting when MSR(PM) is set to a one. The counter values are not changed, only suspended.

SGA: SCU Gets All: This bit when set to a one allows the SCU to control all twenty two counters i.e., the SCU gets all of the counters. This bit over rides the FPU, FXU, ICU and SCU source event set selection.

FPU: PMCR[16:19], 4–bit code selecting FPU source event set.

FXU: PMCR[20:23], 4–bit code selecting FXU source event set.

ICU: PMCR[24:27], 4–bit code selecting ICU source event set.

SCU: PMCR[28:31], 4–bit Code selecting SCU source event set.

Selection Of Monitoring States

In the Rios architecture there is a system wide register known as the **MSR** or Machine State Register (similar to the S370 **PSW)**. An additional bit has been defined in the **MSR.** This bit, number 29, is referred to as **PM** (Performance Monitor). The setting of this bit is preserved (not altered) by an SVC instruction and set to zero when any interrupt is fielded in the same manner as **MSR(FP)** (see figure 34 on page 98 of 0.90 RS/2 Proc. Arch.).

The **MSR(PM)** bit is used to identify a process for monitoring. This would allow a greater flexibility than the problem state bit **MSR(PM)** since a single process could be measured by itself. This measurement could optionally exclude all system interference. Also it would eliminate the need to save and restore the counters of the Performance Monitor since only a specific process (or set of processes) could alter the counter values.

This is true since the MSR is part of process state and is saved and restored when processes go in and out of execution. Thus, the **MSR(PM)** bit will clearly be useful as a non–intrusive method to measure the performance of single benchmarks (e.g., the Spec Suite). In such cases both the benchmark code (without the system) *and* the system code (without the benchmark) could be measured separately.

In an MP environment, **MSR(PM)** should control the per CPU events in the same manner as the uniprocessor case. A uniform definition of a method to control system wide MP measurements is as yet incomplete. The Shadow Directory Unit (SDU) which maintains MP cache coherency control is an example of a system wide resource of this sort. This unit has a preliminary set of counters defined.

The MP monitoring control would minimally provide the ”or” of the **MSR(PM)** bits from each of the MP member CPU’s as an enabling condition. A refinement would provide the ability to eliminate MP member CPU activities that did not have **MSR(PM)** activated unless these activities were directly related to monitored member CPU activities.

Currently there are the following control bits in the Performance Monitor Control Register.

DIS = Disable all counters, no matter what.

DP = Disable all counters if **MSR(PR)** is reset (Privileged)

DU = Disable all counters if **MSR(PR**) is set (Problem state i.e. not Privileged)

DMS = Disable all counters if **MSR(PM**) is Set

DMR = Disable all counters if **MSR(PM)** is Reset

The above bits are intended to provide control of the Performance Monitor by distinguishing between AIX events and otherwise. The following table designates the set of conditions that can be used to qualify the control of the Performance Monitor counters:

Defined Monitoring States

The bit PR in the MSR controls PRoblem state. When PR=0, all instructions may be executed, this condition is privileged state. When PR=1 only non–privileged instructions may be executed, this condition is problem state.

The PM bit in the MSR controls the performance Monitor Mode. When PM=1, the CPU is said to be monitored. When PM=0, the CPU is *not* monitored. The state of the monitor (i.e. counting or not) depends on the setting of PM, PR, DIS, DP, DU, DMS, DMR as defined by the following table.

DIS	DP	DU	DMS	DMR	Disabled	Counted
0	0	0	0	0	nothing	everything
0	0	0	0	1	–PM	monitored (PR=X)
0	0	0	1	0	+PM	not monitored (PR=X)
0	0	1	0	0	+PR	privileged (PM=X)
0	1	0	0	0	–PR	not privileged (PM=X)
0	0	1	0	1	+PR or –PM	privileged AND monitored
0	0	1	1	0	+PR or +PM	privileged AND not monitored
0	1	0	0	1	–PR or –PM	not privileged AND monitored
0	1	0	1	0	–PR or +PM	not privileged AND not monitored
X	X	X	1	1	everything	nothing
X	1	1	X	X	everything	nothing
1	X	X	X	X	everything	nothing

The logic equation for the *disabling* of the counters is:

DISABLE = DIS + DP*(PR=0) + DU*(PR=1) + DMS*(PM=1) + DMR*(PM=0)

Usage For System Measurement

The Rios/2 Performance Monitor function allows monitoring of CPU activity with resorting to sampling. One of the key features of the monitor is the ability to examine particular processes or kernel activity with essentially no overhead. In particular it is important to note that the monitor can be automatically controlled by either a particular process or by supervisory activities. Also of note is that supervisory services can be excluded form the measurement process the same holds true for processes with the MSR(PM) bit as well.

Note that if the MSR(PM) bit was set to a one for all processes that were not ”operating” system (i.e., user processes) then it would be possible to filter out the various combinations of events as described by the section on ”Defined Monitored States”. For example, it would be possible to eliminate all privileged execution if desired.

Also it is of note just a single task could be excluded from measurement. For example, a process whose responsibility it is to manage the monitor could be defined with MSR(PM)=1; If MMCR(DMS)=1 then the out come would be to measure everything except the task that managed the performance monitor. This would eliminate the effect of the monitor management task.

Basic Considerations

At a 20 nanosecond CPU clock rate a 32–bit counter incrementing continuously will over flow in 85.9 seconds. Thus as long as the monitor counters are read at least as once every 85.9 seconds there will be no data lost due to overflow. In particular, if the counters are inspected on every real time clock interrupt and every task switch, no overflow ambiguity would exist

If the counters are managed by the operating system at a users request then it will be possible to allow multiplexing of the monitoring function across multiple processes. Each process could select a set of hardware events independently of all other processes in execution.

It is envisioned that the monitor function would be integrated into programs such as iostat, vmstat and aixtrace. If the monitor function were integrated into aixtrace it could provide additional insight by providing parameters such as cache misses per process, actual instruction path lengths, tlb misses etc.

If the environment being monitored operates in a steady state then it might prove useful to alternate the events to be counted. This is to say that on a periodic basis one could the values of MMCR(FPU), MMCR(FXU), MMCR(ICU) and MMCR(SCU) across the values of 0, 1, 2, ... 15. In this way one could sample all events and thereby obtain information from all of the signal types that are defined.

Monitoring All Events Using A Log

If a trace of counts for all process and operating system activity is desired then the following algorithm may be applied. In this algorithm a log file is constructed for a performance tracing session. This log file will be processed after the tracing session in a manner similar to AIXTRACE/RMAP. In fact it is recommended that this mechanism be integrated into AIXTRACE/RMAP.

This method does not rely on the MSR(PM) for control of the monitor, all events are monitored. Note that the monitor counters are to be considered process state just as the general purpose registers are considered process state. Thus, the monitor counters must be saved and restored.

A buffer is provided to journal the monitor counters as they are saved due to processes coming in and out of execution. Each journal entry would consist of the process id, the MMCR, the counter values, and the real time clock at the time that the counters were read. It would be advantageous for the monitor buffer and the Aixtrace buffer to be one and the same.

As this buffer fills it will be necessary to write it to a mass storage device if the accumulated data is to be post processed by some utility such as AIXTRACE/RMAP.

Note that in the MP case there would be a buffer per CPU, and that after the performance tracing session is over, the journals from the CPU members of the MP would be integrated into a single journal which would show the results of process migration should it occur.

The save restore algorithm is as follows.

1. At the point in time that a monitored process is stopped, record the current monitoring mode, counts and process ID into a system wide buffer in storage designated for this purpose.

2. Immediately before starting the next process, record the mode and counts that the operating system has accumulated. Set the monitoring mode for the process to be started/resumed and monitored (i.e. set the events selected to be monitored by this process) and clear the residual counts in the counters.

3. start/resume the process (usually with an RFI or RFSVC instructions).

Monitoring One Process

Perhaps 95% of all performance study focuses on a single process, usually a benchmark such as "li" form the SPEC benchmark suite. In this case the MMCR and the MSR(PM) bits can be used to great advantage in collecting performance data.

The MSR for the benchmark in question will have PM set to one and the MMCR would have DMR set to one. All other processes executing would have their respective MSR(PM) bits set to zero. As a consequence, only the benchmark with MSR(PM)=1 will be able to alter the performance counters. Thus, the performance counters would not have to be saved or restored, eliminating the attendant overhead.

Since the counters will "roll over" once every 85.9 seconds, it will be necessary for the operating system to schedule an event that will save the performance counters at least that often, but this overhead is negligible. Also the operating system will have to supply the managerial functions for the monitor. This true since reading/writing the MMCR and the MSR is a privileged operation. In general the monitor is a reusable resource and must be managed to insure that it is utilized in a reliable manner.

Monitoring Segments Of Code With Well Defined Entries And Exits

If a segment of code has exactly one entry point then it will be possible to monitor the events that occur within the segment. This would be accomplished by placing a call to monitoring at the entry point and placing a trap instruction at the exit point. While it is true that the call to enable monitoring and the trap to disable monitoring will incur some overhead, this overhead will not affect the measurements made on the code segment in question (except for possibly causing extra cache misses).

Fixed Point Unit

This section lists the events that have been selected for monitoring. Software can select a signal from this list only by selecting a group from the next section that contains the desired signal.

Fixed Point Events Monitored

Fixed Point Hold/Idle/Busy Information

Of Fxu Idle/Held Cycles Caused By Fpu Holds
Of Cycles Fixed Point Is Held Off Due To Conditional Branch
Of Cycles Fixed Point Is Held Off Due To D–cache Busy E–unit–0
Of Cycles Fixed Point Is Held Off Due To D–cache Busy E–unit–1
Of Cycles That One E–unit Is Wasted
Of Cycles With Software Programmable Event (SPE) Equal To One

Cache/Tlb Miss Information (See Data Cache/TLB Miss Measurement Timing Chart)

Of D–cache Misses (Loads And Stores Only)
Of D–cache Reload Cycles Total (Regardless Of Hold State)
Of Total–tlb Misses
Of Total–tlb Miss Cycles Total (Forces Hold State)
Of D–Tlb Invalidate Hits (Page Allocations/Destructions)

Instruction Execution Information

The Following Counts **Include** Those Instructions That Are Interrupted.

Of Instructions Executed By E–unit–0
Of Instructions Executed By E–unit–1
Of Instrs Exec By E–unit–0 That Compare To Instruction Match Register
Of Instrs Exec By E–unit–1 That Compare To Instruction Match Register

Of I/O Instructions Executed
Of Cycles Spent Executing I/O Instructions
Of Cycles Fixed Point Is Executing Multi–cycle Ops

The Following Counts **Exclude** Operations That Are Interrupted.

Of Dcache Load Data Commands Executed E–unit–0
Of Dcache Load Data Commands Executed E–unit–1
Of Dcache Store Data Commands Executed E–unit–0
Of Dcache Store Data Commands Executed E–unit–1
Of Dcache Floating Point Load Data Commands Executed E–unit–0
Of Dcache Floating Point Load Data Commands Executed E–unit–1
Of Dcache Floating Point Store Data Commands Executed E–unit–0
Of Dcache Floating Point Store Data Commands Executed E–unit–1

Fixed Point Unit

E–unit Utilization Information (See Table Of Shift Cases)

Of Cycles Have 2 Or More Instrs Available But Only One Instr Shifted (Counter 1, Case 1)
Of Cycles Have 1 Or More Instrs Available But No Instrs Shifted (Counter 2, Cases 0,3)
Of Cycles Have 2 Or More Instrs Available And 2 Are Shifted (Counter 3, Case 2)
Of Cycles With Only 1 Instr Available And That One Instr Shifted (Counter 4 Cases 4,5)
Of Cycles With No Instructions Shifted (Counter 5, Cases 6,7,8)

Instruction Sequence Information (See Table Of Instruction Sequence Information)

Of Cycles That Fixed Shifts 1 Op: *Logic Unit Op* Case 1
Of Cycles That Fixed Shifts 1 Op: *Logic Unit Op* Case 2
Of Cycles That Fixed Shifts 1 Op: *Load Dependent Data*
Of Cycles That Fixed Shifts 1 Op: *Load Dependent Data Execute Bypass*
Of Cycles That Fixed Shifts 1 Op: *Special Case Ops Not Allowed During 3–leg Operations*
Of Cycles That Fixed Shifts 1 Op: *Mq Dependencies*
Of Cycles That Fixed Shifts 1 Op: *Four Op Bypass*
Of Cycles That Fixed Shifts 1 Op: *Execute Bypass Feeding A Mtspr Op – Decode Cycle Only*
Of Cycles That Fixed Shifts 1 Op: *Ca Modifying Ops Or Extended Ops Not Allowed*
Of Cycles That Fixed Shifts 1 Op: *All Other Cases*

Table Of Shift Cases

Case Number	Available to Shift	Could have Shifted	Actualy did Shift	Performance Counter Incremented
0	2	0	0	2
1	2	1	1	1
2	2	2	2	3
3	1	0	0	2
4	1	1	1	4
5	1	2	1	4
6	0	0	0	5
7	0	1	0	5
8	0	2	0	5

Notes:

(1) Available to Shift is the number of instructions that are candidates for execution.

(2) Could have Shifted is the number of instructions that the Fixed Point Unit could have executed had they been available.

(3) Actualy did Shift is the number of instruction that were actualy put into execution.

(4) Performance counter 3 specifies the ideal case where two instructions are moved into execution per cycle.

(5) Performance counters 1 and 2 identify the cases where the Fixed Point is blocking execution. These cases may be further studied with groups ”E” and ”F”.

(6) Performance Counters 4 and 5 identifies those cases where the Fixed Point Unit is starved of instructions.

Table Of Instruction Sequence Information

\# Of Cycles That Fixed Shifts 1 Op: *Logic Unit Op* Case 1
(1) Ex1 Is Dependent On A Lu Op In Ex0

\# Of Cycles That Fixed Shifts 1 Op: *Logic Unit Op* Case 2
(1) Lu Op In Ex1 Is Dependent On Op In Ex0

\# Of Cycles That Fixed Shifts 1 Op: *Load Dependent Data*
(1) Ex1 Is Dependent On A Previous Load & Ex0 Is Not.

\# Of Cycles That Fixed Shifts 1 Op: *Load Dependent Data Execute Bypass*
(1) Ex1 Is Dependent On A Load In Ex0.

\# Of Cycles That Fixed Shifts 1 Op: *Special Case Ops Not Allowed During 3–leg Operations*
(1) Data Dependent Ops Doz, Dozi, Abs, Nabs, Neg
(Logic Ops & Multi–cycle Ops Don't Cared)
(2) Second Unit Includes Mul, Muli, Muls, Div, Divs.

\# Of Cycles That Fixed Shifts 1 Op: *Mq Dependencies*
(1) Mq Modifying Ops Feeding Mq Using Ops
(2) Mq Modifying Op In Ex0 Feeds Mq Using Op In Ex1

\# Of Cycles That Fixed Shifts 1 Op: *Four Op Bypass*

```
Eg.    A   Rt,Ra,Rb
       A   Rt',Rt,Rt
```

\# Of Cycles That Fixed Shifts 1 Op: *Execute Bypass Feeding A Mtspr Op – Decode Cycle Only*

Used To Prevent This Case Since The Hardware Cannot Handle It, For Example:

```
A        Rt,Ra,Rb
Mtspr    Dar,Rt
```

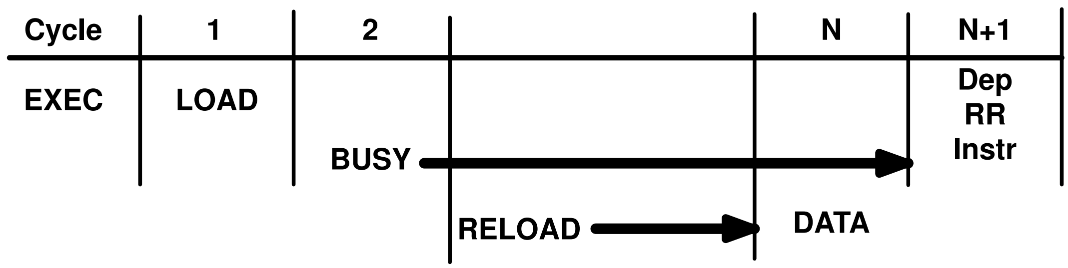
\# Of Cycles That Fixed Shifts 1 Op:
(1) *Ca Modifying Ops Feeding Ca Using Ops*
(1.1) Ca Modifying Op In Ex0 Feeds Ca Using Op In Ex1

(2) *Extended Ops Not Allowed In Ex1 When Bypasses Occur From Ex0*
(2.1) Ae, Ame, Sfe, Sfme, Aze, Sfze
(2.2) Mtspr (Mtspr): Handles Case Of Mtxer

\# Of Cycles That Fixed Shifts 1 Op:
(1) *Two M[T,F]Spr In Decode At Same Time*
(2) *Excepting Op In Ex0, And M[T,F]Spr In Ex1*
(3) *All Other Casses (List To Be Provided By Engineering)*

Data Cache/TLB Miss Measurement Timing Chart

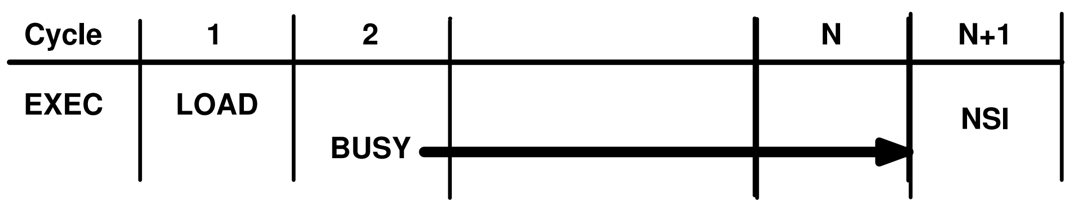
D–Cache Miss Timings



BUSY = Data cache busy, No Data Cache Load or Store commands may be executed to this Data cache port while busy is active.
RELOAD = Data cache reload, Memory system measurement

Note: Store timings identical except for data dependencies.

D–TLB Miss Timings



BUSY = Data Tlb miss
NSI = Next Sequential Instruction

Note: Store timings identical.

Fixed Point Instruction Frequency Function

Special Purpose Register (SPR) number 16 (decimal) has been designated the Instruction Match Register (IMR). This register may be written with the Move To Special Purpose Register (mtspr) and read with the Move From Special Purpose Register (mfspr). Since the SPR affected is greater that decimal 15, both of these instructions are privileged.

This register with the logic shown in figure # will provides the ability to measure the number of times any class of instruction is executed. The encodings for these classes are shown in the appendices of this document. There will be two counts provided, one for each E–unit. These counts are used in Group 7 as follows:

Group 7

1 # Of Instrs Exec By E–unit–0 That Compare To Instruction Match Register
2 # Of Instrs Exec By E–unit–1 That Compare To Instruction Match Register
3 # Of Instructions Executed By E–unit–0
4 # Of Instructions Executed By E–unit–1
5 # Of Cycles Fixed Point Is Executing Multi–cycle Ops

Using this group the instruction frequency of any class of fixed point instruction may be determined as well as the overall fixed point CPI. If the ICU, FPU and SCU are left in their default Group 0 state, then overall frequencies of fixed point instructions may be computed.

Format Of The Instruction Match Register

Indtruction Match Register

Reserved (1)	SPE	Reserved (2)	Opcode
0 – 14	15	16 – 21	22 – 31

The field definitions are as follows:

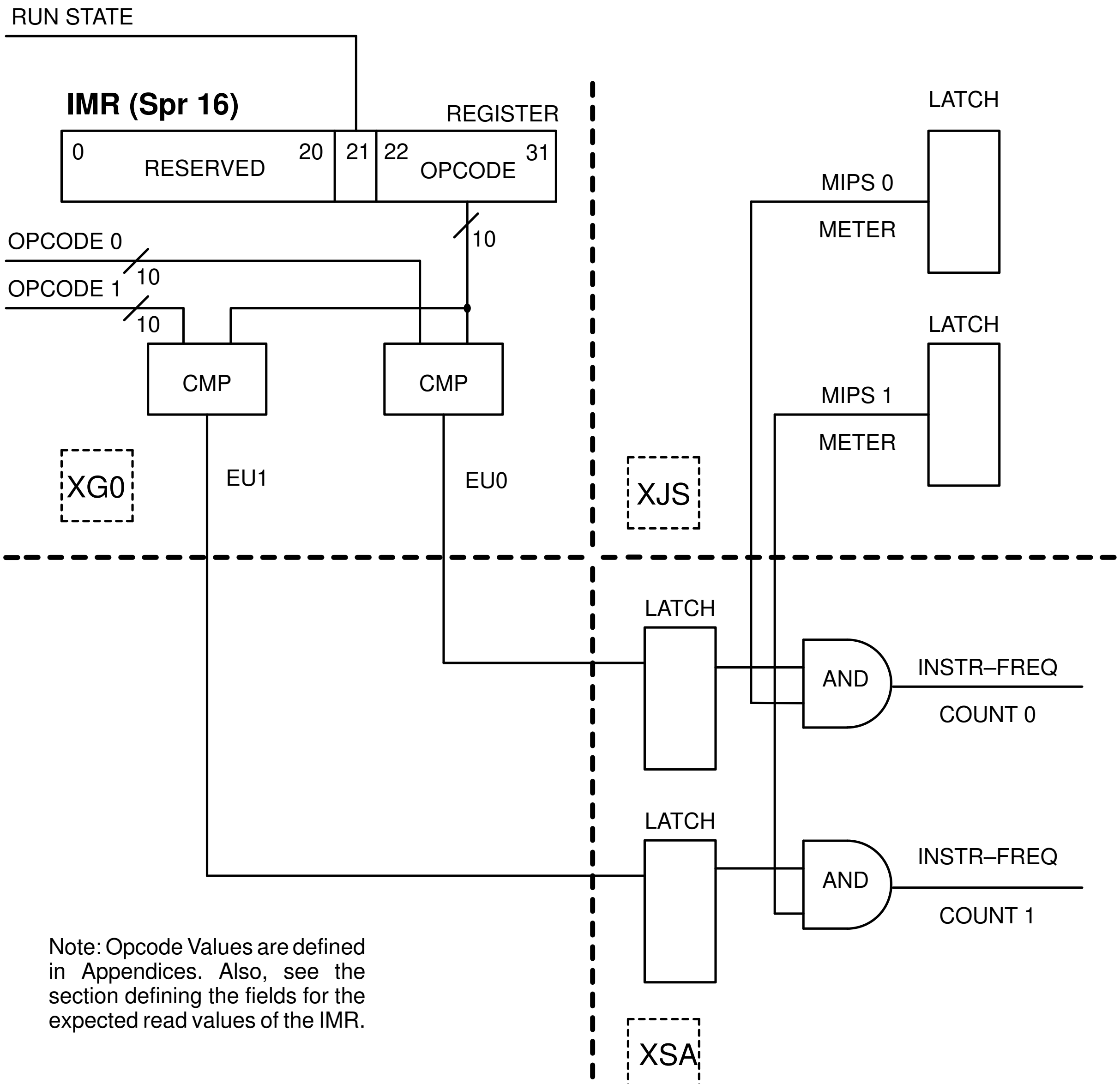
Reserved(1): These bits are NOT currently implemented by the hardware and will always read as zeroes regardless of the pattern written. For compatibility with future designs, software must NOT write any pattern containing other than zeroes into this field unless the pattern is approved by hardware architecture.

SPE: Software Programmable Event – This bit is provided to allow the software to set an event (such) as run state or wait state that can be directly counted by the monitor with the full control of the MMCR conditioning controls. This bit is reflected in group 0 of the FXU.

Reserved(2): These bits ARE currently implemented by the hardware and will always read the value that was written into them. For compatibility with future designs, software must NOT write any pattern containing other than zeroes into this field unless the pattern is approved by hardware architecture.

Opcode: These bits define the fixed point opcode (i.e. instruction) whose frequency is to be monitored. A complete list of these opcodes is given in the appendices of this document. This list is to be considered the definition of the term ”fixed point instruction” (i.e. if an instruction appears in the list then it is considered a fixed point instruction).

Instruction Frequency Count Logic



Note: Opcode Values are defined in Appendices. Also, see the section defining the fields for the expected read values of the IMR.

Fixed Point Event Groupings

This section lists the groupings that have been defined. Keep in mind that *software cannot arbitrarily select any set of five signals to monitor.* Instead, only those sets that follow may be selected. It is also necessary that each chip be represented by exactly one group only. For example, it is not possible to simultaneously select two groups from the Fixed Point.

Group 0 (Default Group)
- 0 # Of Instructions Executed By E–unit–0
- 1 # Of Instructions Executed By E–unit–1
- 2 # Of D–cache Misses (Loads And Stores Only)
- 3 # Of Total–tlb Misses
- 4 # Of Cycles With Software Programmable Event (SPE) Equal To One

Group 1
- 0 # Of Cycles Fixed Point Is Held Off Due To D–cache Busy E–unit–0
- 1 # Of Cycles Fixed Point Is Held Off Due To D–cache Busy E–unit–1
- 2 # Of Fxu Idle/Held Cycles Caused By Fpu Holds
- 3 # Of Cycles Fixed Point Is Held Off Due To Conditional Branch
- 4 # Of Cycles With No Instructions Shifted

Group 2
- 0 # Of D–cache Miss Cycles Total (Regardless Of Hold State)
- 1 # Of D–cache Misses (Loads And Stores Only)
- 2 # Of Total–tlb Miss Cycles Total (Forces Hold State)
- 3 # Of Total–tlb Misses
- 4 # Of D–tlb Invalidate Hits (Page Allocations/Destructions

Group 3
- 0 # Of Loads Executed E–unit–0
- 1 # Of Loads Executed E–unit–1
- 2 # Of Stores Executed E–unit–0
- 3 # Of Stores Executed E–unit–1
- 4 # Of Cycles With No Instructions Shifted

Group 4
- 0 # Of Floating Point Loads Executed E–unit–0
- 1 # Of Floating Point Loads Executed E–unit–1
- 2 # Of Floating Point Stores Executed E–unit–0
- 3 # Of Floating Point Stores Executed E–unit–1
- 4 # Of Cycles With No Instructions Shifted

Group 5
- 0 # Of Instructions Executed By E–unit–0
- 1 # Of Instructions Executed By E–unit–1
- 2 # Of I/O Instructions Executed
- 3 # Of Cycles Spent Executing I/O Instructions
- 4 # Of Cycles Fixed Point Is Executing Multi–cycle Ops

Fixed Point Event Groupings (Continued)

Group 6
- 0 # Of Loads Executed E–unit–0
- 1 # Of Loads Executed E–unit–1
- 2 # Of Stores Executed E–unit–0
- 3 # Of Stores Executed E–unit–1
- 4 # Of Cycles Fixed Point Is Executing Multi–cycle Ops

Group 7
- 0 # Of Instrs Exec By E–unit–0 That Compare To Instruction Match Register
- 1 # Of Instrs Exec By E–unit–1 That Compare To Instruction Match Register
- 2 # Of Instructions Executed By E–unit–0
- 3 # Of Instructions Executed By E–unit–1
- 4 # Of Cycles Fixed Point Is Executing Multi–cycle Ops

Group 8
- 0 # Of Cycles Have 2 Or More Instrs Available But Only One Instr Shifted
- 1 # Of Cycles Have 1 Or More Instrs Available But No Instrs Shifted
- 2 # Of Cycles Have 2 Or More Instrs Available And 2 Are Shifted
- 3 # Of Cycles With Only 1 Instr Available And That One Instr Shifted
- 4 # Of Cycles With No Instructions Shifted

Group 9
- 0 # Cycles Fixed Shifts 1:Logic Unit Op Case 1
- 1 # Cycles Fixed Shifts 1:Logic Unit Op Case 2
- 2 # Cycles Fixed Shifts 1:Load Dependent Data
- 3 # Cycles Fixed Shifts 1:Load Dependent Data Execute Bypass
- 4 # Cycles Fixed Shifts 1:Special Cases Not Allowed During 3–leg Ops

Group A
- 0 # Cycles Fixed Shifts 1:Mq Dependencies
- 1 # Cycles Fixed Shifts 1:Four Op Bypass
- 2 # Cycles Fixed Shifts 1:Execute Bypass Feeds Mtspr Op – Dec. Cycle
- 3 # Cycles Fixed Shifts 1:Ca Modifying Ops Or Extended Ops Not Allowed
- 4 # Cycles Fixed Shifts 1:All Other Cases

Group B
- 0 # Of Cycles That One E–unit Is Wasted
- 1 # Of Cycles Have 1 Or More Instrs Available But No Instrs Shifted
- 2 # Of Cycles Have 2 Or More Instrs Available But Only One Instr Shifted
- 3 # Of Cycles Fixed Point Is Executing Multi–cycle Ops
- 4 # Of Cycles With No Instructions Shifted

Group C
- 0 # Of Instructions Executed By E–unit–0
- 1 # Of Instructions Executed By E–unit–1
- 2 # Of D–cache Miss Cycles Total (Regardless Of Hold State)
- 3 # Of Total–tlb Miss Cycles Total (Forces Hold State)
- 4 # Of Cycles With No Instructions Shifted

Branch Unit

Branch Events Monitored

This section lists the events that have been selected for monitoring. Software can select a signal from this list only by selecting a group from the next section that contains the desired signal.

Cache/Tlb Miss Information

Of I–cache Misses
Of I–tlb Misses

Dispatch/Fetch Information

Of Instructions Fetched Is B"1000" (8 I.E. Max)
Of Instructions Fetched Is B"01xx" (4,5,6 Or 7)
Of Instructions Fetched Is B"0x1x" (2,3,6 Or 7)
Of Instructions Fetched Is B"0xx1" (1,3,5 Or 7)
(May Be Up To 8/Cycle => 4 Counters Required)

Of Target Instructions Dispatched Is B"1xx" (4,5 Or 6)
Of Target Instructions Dispatched Is B"x1x" (2,3 Or 6)
Of Target Instructions Dispatched Is B"xx1" (1,3 Or 5)
(May Be Up To 6/Cycle => 3 Counters Required)

Of Sequential Instructions Dispatched Is B"1xx" (4,5 Or 6)
Of Sequential Instructions Dispatched Is B"x1x" (2,3 Or 6)
Of Sequential Instructions Dispatched Is B"xx1" (1,3 Or 5)
(May Be Up To 6/Cycle => 3 Counters Required)

Of Target Prefetches
Of Cycles No Dispatch
Of Cycles No Dispatch Due To I–cache Miss And Buffer Empty
Of Cycles No Dispatch Due To I–tlb Miss And Buffer Empty
Of Cycles No Dispatch Due To Cr Dependencies

Instruction Execution Information

Of Instructions Executed By Icu Is B"x1" (1 Or 3)
Of Instructions Executed By Icu Is B"1x" (2 Or 3)
(May Execute Up To Three Instructions Cycle)

Of Lcr Instructions Executed
Of Svc Instructions Executed
Of Interrupts Taken (Any Kind)
Of Mtmsr Instructions Executed
Of Instruction Cache Synchronize Instructions Executed
Of Rfsvc Instructions Executed
Of Rfi Instructions Executed

Branch Events Monitored (Continued)

Instruction Execution Information (Continued)

Of Total Executed Branches Is B”x1” (I.E. 1)
Of Total Executed Branches Is B”1x” (I.E. 2)
(May Execute Up To Two Of These Per Cycle)

Of Not Taken Conditional Branches Is B”x1” (I.E. 1)
Of Not Taken Conditional Branches Is B”1x” (I.E. 2)
(May Execute Up To Two Of These Per Cycle)

Of Taken Conditional Branches
Of Unconditional Branches
Of Branch On Counter Taken (resolved by counter value only)
Of Branch On Counter Not–taken (resolved by counter value only)

Of Pending Conditional Branches Resolved By Fixed
Of Pending Conditional Branches Resolved By Float
Of Pending Conditional Branches Resolved By Branch

Branch Event Groupings

Group 0 (Default Group)
- 0 # Of Instructions Executed: Total By Icu Is B”1x” (2 Or 3)
- 1 # Of Instructions Executed: Total By Icu Is B”x1” (1 Or 3)
- 2 # Of Cycles No Dispatch
- 3 # Of I–cache Misses
- 4 # Of I–tlb Misses

Group 1
- 0 # Of Instructions Fetched: Total Is B”01xx” (4,5,6 Or 7)
- 1 # Of Instructions Fetched: Total Is B”0x1x” (2,3,6 Or 7)
- 2 # Of Instructions Fetched: Total Is B”0xx1” (1,3,5 Or 7)
- 3 # Of Instructions Fetched: Total Is B”1000” (8 I.E. Max)
- 4 # Of Target Prefetches

Group 2
- 0 # Of Branches: Total Executed Is B”1x” (I.E. 2)
- 1 # Of Branches: Total Executed Is B”x1” (I.E. 1)
- 2 # Of Branches: Pending Conditional Resolved By Branch
- 3 # Of Branches: Pending Conditional Resolved By Fixed
- 4 # Of Branches: Pending Conditional Resolved By Float

Group 3
- 0 # Of Branches: Total Executed Is B”1x” (I.E. 2)
- 1 # Of Branches: Total Executed Is B”x1” (I.E. 1)
- 2 # Of Branches: Not Taken Conditional Is B”1x” (I.E. 2)
- 3 # Of Branches: Not Taken Conditional Is B”x1” (I.E. 1)
- 4 # Of Branches: Taken Conditional Branches

Group 4
- 0 # Of Branches: Total Executed Is B”1x” (I.E. 2)
- 1 # Of Branches: Total Executed Is B”x1” (I.E. 1)
- 2 # Of Branches: On Counter Not–taken (Non–conditional)
- 3 # Of Branches: On Counter Taken (Non–conditional)
- 4 # Of Branches: Unconditional Branches

Group 5
- 0 # Of Branches: Total Executed Is B”1x” (I.E. 2)
- 1 # Of Branches: Total Executed Is B”x1” (I.E. 1)
- 2 # Of Cycles No Dispatch
- 3 # Of I–cache Misses
- 4 # Of I–tlb Misses

Group 6
- 0 # Of Instructions Executed: Interrupts Taken (Any Kind)
- 1 # Of Cycles No Dispatch
- 2 # Of Cycles No Dispatch Due To Cr Dependencies
- 3 # Of Cycles No Dispatch Due To I–cache Miss And Buffer Empty
- 4 # Of Cycles No Dispatch Due To I–tlb Miss And Buffer Empty

Branch Event Groupings (Continued)

Group 7
0 # Of Cycles No Dispatch
1 # Of Cycles No Dispatch Due To I–cache Miss And Buffer Empty
2 # Of Cycles No Dispatch Due To I–tlb Miss And Buffer Empty
3 # Of I–cache Misses
4 # Of I–tlb Misses

Group 8
0 # Of Branches: Total Executed Is B”1x” (I.E. 2)
1 # Of Branches: Total Executed Is B”x1” (I.E. 1)
2 # Of Instructions Dispatched: Sequential Is B”1xx” (4,5 Or 6)
3 # Of Instructions Dispatched: Sequential Is B”x1x” (2,3 Or 6)
4 # Of Instructions Dispatched: Sequential Is B”xx1” (1,3 Or 5)

Group 9
0 # Of Branches: Total Executed Is B”1x” (I.E. 2)
1 # Of Branches: Total Executed Is B”x1” (I.E. 1)
2 # Of Instructions Dispatched: Target Is B”1xx” (4,5 Or 6)
3 # Of Instructions Dispatched: Target Is B”x1x” (2,3 Or 6)
4 # Of Instructions Dispatched: Target Is B”xx1” (1,3 Or 5)

Group A
0 # Of Instructions Fetched: Total Is B”01xx” (4,5,6 Or 7)
1 # Of Instructions Fetched: Total Is B”0x1x” (2,3,6 Or 7)
2 # Of Instructions Fetched: Total Is B”0xx1” (1,3,5 Or 7)
3 # Of Instructions Fetched: Total Is B”1000” (8 I.E. Max)
4 # Of I–cache Misses

Group B
0 # Of Instructions Fetched: Total Is B”01xx” (4,5,6 Or 7)
1 # Of Instructions Fetched: Total Is B”0x1x” (2,3,6 Or 7)
2 # Of Instructions Fetched: Total Is B”0xx1” (1,3,5 Or 7)
3 # Of Instructions Fetched: Total Is B”1000” (8 I.E. Max)
4 # Of I–tlb Misses

Group C
0 # Of Cycles No Dispatch Due To Cr Dependencies
1 # Of Instructions Executed: Lcr
2 # Of Branches: Not Taken Conditional Is B”1x” (I.E. 2)
3 # Of Branches: Not Taken Conditional Is B”x1” (I.E. 1)
4 # Of Branches: Taken Conditional Branches

Group D
0 # Of Instructions Executed: Total By Icu Is B”1x” (2 Or 3)
1 # Of Instructions Executed: Total By Icu Is B”x1” (1 Or 3)
2 # Of Cycles No Dispatch
3 # Of Cycles No Dispatch Due To I–cache Miss And Buffer Empty
4 # Of Cycles No Dispatch Due To I–tlb Miss And Buffer Empty

Branch Event Groupings (Continued)

Group E
- 0 # Of Cycles No Dispatch
- 1 # Of Instructions Executed: Mtmsr
- 2 # Of Instruction Cache Synchronize Instructions Executed
- 3 # Of Instructions Executed: Lcr
- 4 # Of Instructions Executed: Svc

Group F
- 0 # Of Svc Instructions Executed
- 1 # Of Rfsvc Instructions Executed
- 2 # Of Interrupts Taken (Any Kind)
- 3 # Of Rfi Instructions Executed
- 4 # Of Branches: Unconditional Branches

Floating Point Unit

This section lists the events that have been selected for monitoring. Software can select a signal from this list only by selecting a group from the next section that contains the desired signal.

Floating Point Events Monitored

Execution Idle/Hold Off Conditions

Of Cycles Both Aq0 And Aq1 data flow is busy
Of Cycles Aq0 data flow is busy
Of Cycles Aq1 data flow is busy

Of Hold Cycles Math0 Given That There Are Valid & Sunk Instructions To Be Done
Of Hold Cycles Math1 Given That There Are Valid & Sunk Instructions To Be Done
Of Hold Cycles Math0 Due To Load/Store Instructions That Are Valid But Not Sunk
Of Hold Cycles Math1 Due To Load/Store Instructions That Are Valid But Not Sunk

Execution Idle/Hold Off Conditions

Of Hold Cycles Caused By Data Dependencies: Loads: Math0
Of Hold Cycles Caused By Data Dependencies: Loads: Math1
Of Hold Cycles Caused By Data Dependencies: Stores: Math0
Of Hold Cycles Caused By Data Dependencies: Stores: Math1
Of Hold Cycles Caused By Data Anti–Dependencies: Stores: Math0
Of Hold Cycles Caused By Data Anti–Dependencies: Stores: Math1

Of Hold Cycles Caused By Data Dependencies: Math0 On Math0
Of Hold Cycles Caused By Data Dependencies: Math0 On Math1
Of Hold Cycles Caused By Data Dependencies: Math1 On Math0
Of Hold Cycles Caused By Data Dependencies: Math1 On Math1

Of Cycles One Load Waits Due To Avrq Empty
Of Cycles Two Loads Wait Due To Avrq Empty
Of Hold Cycles Caused By Inability To Use Data Bypass On A Op: Math0
Of Hold Cycles Caused By Inability To Use Data Bypass On A Op: Math1

Of Hold Cycles Caused By Synchronizing Instrs: Math0 (Fpscr, Rc Bit Etc.)
Of Hold Cycles Caused By Synchronizing Instrs: Math1 (Fpscr, Rc Bit Etc.)

Of Hold Cycles Caused By Normalization Math0
Of Hold Cycles Caused By Normalization Math1
Of Hold Cycles Caused By Underflow/Overflow Math0
Of Hold Cycles Caused By Underflow/Overflow Math1
Of Hold Cycles Caused By Special Numbers Math0
Of Hold Cycles Caused By Special Numbers Math1
(May Be Up To Two Of These Per Cycle)

Floating Point Events Monitored (Continued)

Instruction Execution Information

Of Arithmetic Instructions Executed Math0
Of Arithmetic Instructions Executed Math1
Of Load Instructions Executed Ldex0
Of Load Instructions Executed Ldex1
Of Store Instructions Executed Stex0
Of Store Instructions Executed Stex1

Instruction Frequency Information

Of Convert Instructions Math0
Of Convert Instructions Math1
Of Fa Instructions Math0
Of Fa Instructions Math1
Of Fcmp Instructions Math0
Of Fcmp Instructions Math1
Of Fd Instructions Math0
Of Fd Instructions Math1
Of Fm Instructions Math0
Of Fm Instructions Math1
Of Fma Instructions Math0
Of Fma Instructions Math1
Of Fpscr Instructions Math0
Of Fpscr Instructions Math1
Of Frsp Instructions Math0
Of Frsp Instructions Math1
Of Move Instructions Math0
Of Move Instructions Math1
Of Square Root Instructions Math0
Of Square Root Instructions Math1

Queue Utilization Information

Of Cycles Aq (Arith Queue) Is Empty
Of Cycles Aq (Arith Queue) Is Full
Of Cycles Avrq Is Empty
Of Cycles Avrq Has One, Two Or Three Entries
Of Cycles Ldq (Load Queue) Is Empty
Of Cycles Ldq (Load Queue) Is Full
Of Cycles Stq (Store Queue) Is Empty
Of Cycles Stq (Store Queue) Is Full

Of Cycles Ipb (Instr–queue) Is Empty
Of Entries In Ipb (Instr–queue) Is B”1000” (8 I.E. Full)
Of Entries In Ipb (Instr–queue) Is B”01xx” (4,5,6 Or 7)
Of Entries In Ipb (Instr–queue) Is B”0x1x” (2,3,6 Or 7)
Of Entries In Ipb (Instr–queue) Is B”0xx1” (1,3,5 Or 7)
(May Be Up To 8/Cycle => 4 Counters Required)

Floating Point Event Groupings

Group 0 (Default Group)

0 # Of Arithmetic Instructions Executed Math0
1 # Of Arithmetic Instructions Executed Math1
2 # Of Cycles Aq0 Data Flow Is Busy
3 # Of Cycles Aq1 Data Flow Is Busy
4 # Of Cycles Both Aq0 And Aq1 Data Flow Are Busy

Group 1

0 # Of Entries In Ipb (Instr–queue) Is B”01xx” (4,5,6 Or 7)
1 # Of Entries In Ipb (Instr–queue) Is B”0x1x” (2,3,6 Or 7)
2 # Of Entries In Ipb (Instr–queue) Is B”0xx1” (1,3,5 Or 7)
3 # Of Entries In Ipb (Instr–queue) Is B”1000” (8 I.E. Full)
4 # Of Cycles Ipb (Instr–queue) Is Empty

Group 2

0 # Of Cycles Ldq (Load Queue) Is Full
1 # Of Cycles One Load Waits Due To Avrq Empty
2 # Of Cycles Two Loads Wait Due To Avrq Empty
3 # Of Cycles Avrq Has One, Two Or Three Entries
4 # Of Cycles Avrq Is Empty

Group 3

0 # Of Hold Cycles Caused By Normalization Math0
1 # Of Hold Cycles Caused By Special Numbers Math0
2 # Of Hold Cycles Caused By Underflow/Overflow Math0
3 # Of Hold Cycles Caused By Synchronizing Instrs: Math0 (Fpscr, Rc Bit Etc.)
4 # Of Hold Cycles Caused By Inability To Use Data Bypass On A Op: Math0

Group 4

0 # Of Hold Cycles Caused By Data Anti–dependencies: Stores: Math0
1 # Of Hold Cycles Caused By Data Dependencies: Stores: Math0
2 # Of Hold Cycles Caused By Data Dependencies: Loads: Math0
3 # Of Hold Cycles Math0 Due To Load/Store Instrs That Are Valid But Not Sunk
4 # Of Hold Cycles Math0 Given That There Are Valid & Sunk Instrs To Be Done

Group 5

0 # Of Hold Cycles Caused By Normalization Math1
1 # Of Hold Cycles Caused By Special Numbers Math1
2 # Of Hold Cycles Caused By Underflow/Overflow Math1
3 # Of Hold Cycles Caused By Synchronizing Instrs: Math1 (Fpscr, Rc Bit Etc.)
4 # Of Hold Cycles Caused By Inability To Use Data Bypass On A Op: Math1

Group 6

0 # Of Hold Cycles Caused By Data Anti–dependencies: Stores: Math1
1 # Of Hold Cycles Caused By Data Dependencies: Stores: Math1
2 # Of Hold Cycles Caused By Data Dependencies: Loads: Math1
3 # Of Hold Cycles Math1 Due To Load/Store Instrs That Are Valid But Not Sunk
4 # Of Hold Cycles Math1 Given That There Are Valid & Sunk Instrs To Be Done

Floating Point Event Groupings (Continued)

Group 7
- 0 # Of Hold Cycles Caused By Data Dependencies: Math0 On Math0
- 1 # Of Hold Cycles Caused By Data Dependencies: Math0 On Math1
- 2 # Of Hold Cycles Caused By Data Dependencies: Math1 On Math0
- 3 # Of Hold Cycles Caused By Data Dependencies: Math1 On Math1
- 4 # Of Cycles Both Aq0 And Aq1 Data Flow Are Busy

Group 8
- 0 # Of Cycles Aq (Arith Queue) Is Empty
- 1 # Of Cycles Aq (Arith Queue) Is Full
- 2 # Of Cycles Aq0 Data Flow Is Busy
- 3 # Of Cycles Aq1 Data Flow Is Busy
- 4 # Of Cycles Both Aq0 And Aq1 Data Flow Are Busy

Group 9
- 0 # Of Cycles Ldq (Load Queue) Is Full
- 1 # Of Cycles Ldq (Load Queue) Is Empty
- 2 # Of Cycles Stq (Store Queue) Is Empty
- 3 # Of Cycles Stq (Store Queue) Is Full
- 4 # Of Cycles Both Aq0 And Aq1 Data Flow Are Busy

Group A
- 0 # Of Arithmetic Instructions Executed Math0
- 1 # Of Fa Instructions Math0
- 2 # Of Fm Instructions Math0
- 3 # Of Fd Instructions Math0
- 4 # Of Fma Instructions Math0

Group B
- 0 # Of Arithmetic Instructions Executed Math0
- 1 # Of Load Instructions Executed Ldex0
- 2 # Of Store Instructions Executed Stex0
- 3 # Of Fcmp Instructions Math0
- 4 # Of Move Instructions Math0

Group C
- 0 # Of Arithmetic Instructions Executed Math0
- 1 # Of Fpscr Instructions Math0
- 2 # Of Frsp Instructions Math0
- 3 # Of Convert Instructions Math0
- 4 # Of Square Root Instructions Math0

Group D
- 0 # Of Arithmetic Instructions Executed Math1
- 1 # Of Fa Instructions Math1
- 2 # Of Fm Instructions Math1
- 3 # Of Fd Instructions Math1
- 4 # Of Fma Instructions Math1

Floating Point Event Groupings (Continued)

Group E
- 0 # Of Arithmetic Instructions Executed Math1
- 1 # Of Load Instructions Executed Ldex1
- 2 # Of Store Instructions Executed Stex1
- 3 # Of Fcmp Instructions Math1
- 4 # Of Move Instructions Math1

Group F
- 0 # Of Arithmetic Instructions Executed Math1
- 1 # Of Fpscr Instructions Math1
- 2 # Of Frsp Instructions Math1
- 3 # Of Convert Instructions Math1
- 4 # Of Square Root Instructions Math1

Note that the signal ”# Of Fpscr Instructions Math1” is always zero and need not be provided. This signal is always zero since the Floating Point Unit does not execute Fpscr instructions in Math1.

Storage Control Unit

This section lists the events that have been selected for monitoring. Software can select a signal from this list only by selecting a group from the next section that contains the desired signal. Note though the exceptions shown by the ”Special Group”.

Storage Control Events Monitored

Cache Reload/Storeback Information

Of I Cache Reload Requests
Of D Cache Reload Requests
Of D Cache Storeback Requests
Of Cycles Latency To Start Of I Cache Reload Requests
Of Cycles Latency To Start Of D Cache Reload Requests
Of Cycles Latency To Start Of D Cache Storeback Requests
Of Cycles To Complete I Cache Reload Requests
Of Cycles To Complete D Cache Reload Requests
Of Cycles To Complete D Cache Storeback Requests

Dma–I/O Information

Of Dma Read Requests(Operations)
Of Dma Write Requests(Operations)
Of Cycles Latency To Start Of Dma Read
Of Cycles Latency To Start Of Dma Write
Of Dma Prefetch Requests
Of Dma Next Prefetch Requests
Of Dma Next Prefetch (Data Invalidated)
Of Dma Read–modify–write Requests

Of Interrupts Handled (External)
Of Cycles Iocu A Request Latency
Of Cycles Iocu B Request Latency
Of Cycles Iocu C Request Latency
Of Cycles Iocu D Request Latency
Of Cycles Iocu A, B, C, or D Request Latency (Any One Active).

Memory System Information

Of Cycles Memory Address Bus Is Busy
Of Cycles That Memory Bus Hold (From Memory System) Is Asserted
Of Cycles That Data Is Being Moved On Memory Bus
Of Cpu Memory Read Requests (Operations) Total
Of Memory Requests (Operations) Total
Of Cycles Memory Request Waits For Memory Bus (Latency)

Storage Control Events Monitored (Continued)

Sio Bus Information

Of Sio Bus Requests
Of Cycles Sio Bus Is Busy (Utilization, Includes Dead Cycles)
Of Cycles That Data Is Being Moved On Sio Bus By Dma

Storage Control Event Groupings

Group 0 (Default Group)
0 # Of I Cache Reload Requests
1 # Of D Cache Reload Requests
2 # Of D Cache Storeback Requests
3 # Of Dma Read Requests(Operations)
4 # Of Dma Write Requests(Operations)

Group 1
0 # Of Cycles That Data Is Being Moved On Memory Bus
1 # Of Cpu Memory Read Requests (Operations) Total
2 # Of D Cache Storeback Requests
3 # Of Dma Read Requests(Operations)
4 # Of Dma Write Requests(Operations)

Group 2
0 # Of Cycles Iocu A Request Latency
1 # Of Cycles Iocu B Request Latency
2 # Of Cycles Iocu C Request Latency
3 # Of Cycles Iocu D Request Latency
4 # Of Cycles Iocu A, B, C, or D Request Latency (Any One Active).

Group 3
0 # Of Dma Prefetch Requests
1 # Of Dma Next Prefetch Requests
2 # Of Dma Next Prefetch (Data Invalidated)
| 3 # Of Cycles That Data Is Being Moved On Memory Bus
4 # Of Cycles That Data Is Being Moved On Sio Bus By Dma

Group 4
0 # Of Dma Prefetch Requests
1 # Of Dma Next Prefetch Requests
2 # Of Dma Next Prefetch (Data Invalidated)
| 3 # Of Dma Read Requests(Operations)
| 4 # Of Dma Write Requests(Operations)

Storage Control Event Groupings (Continued)

Group 5
- 0 # Of Cycles Memory Address Bus Is Busy
- 1 # Of Cycles That Memory Bus Hold (From Memory System) Is Asserted
- 2 # Of Cycles That Data Is Being Moved On Memory Bus
- 3 # Of Memory Requests (Operations) Total
- 4 # Of Cycles Memory Request Waits For Memory Bus (Latency)

Group 6
- 0 # Of Sio Bus Requests
- 1 # Of Cycles Sio Bus Is Busy (Utilization, Includes Dead Cycles)
- 2 # Of Cycles That Data Is Being Moved On Sio Bus By Dma
- 3 # Of Dma Read Requests(Operations)
- 4 # Of Dma Write Requests(Operations)

Group 7
- 0 # Of I Cache Reload Requests
- 1 # Of Cycles Latency To Start Of I Cache Reload Requests
- 2 # Of Cycles To Complete I Cache Reload Requests
- 3 # Of Cycles That Memory Bus Hold (From Memory System) Is Asserted
- 4 # Of Cycles That Data Is Being Moved On Memory Bus

Group 8
- 0 # Of D Cache Reload Requests
- 1 # Of Cycles Latency To Start Of D Cache Reload Requests
- 2 # Of Cycles To Complete D Cache Reload Requests
- 3 # Of Cycles That Memory Bus Hold (From Memory System) Is Asserted
- 4 # Of Cycles That Data Is Being Moved On Memory Bus

Group 9
- 0 # Of D Cache Storeback Requests
- 1 # Of Cycles Latency To Start Of D Cache Storeback Requests
- 2 # Of Cycles To Complete D Cache Storeback Requests
- 3 # Of Cycles That Memory Bus Hold (From Memory System) Is Asserted
- 4 # Of Cycles That Data Is Being Moved On Memory Bus

Group A
- 0 # Of Interrupts Handled (External)
- 1 # Of Cycles Iocu A, B, C, or D Request Latency (Any One Active).
- 2 # Of Sio Bus Requests
- 3 # Of Cycles Sio Bus Is Busy (Utilization, Includes Dead Cycles)
- 4 # Of Cycles That Data Is Being Moved On Sio Bus By Dma

Group B
- 0 # Of Sio Bus Requests
- 1 # Of Cycles Sio Bus Is Busy (Utilization, Includes Dead Cycles)
- 2 # Of Cycles That Data Is Being Moved On Sio Bus By Dma
- 3 # Of Cycles Latency To Start Of Dma Read
- 4 # Of Cycles Latency To Start Of Dma Write

Storage Control Unit Special Group

MONITOR REGISTER	STORAGE CONTROL SIGNAL COUNTED	NOTES
PMC00	# Of CPU Clock Cycles	
PMC01	# Of I Cache Reload Requests	
PMC02	# Of D Cache Reload Requests	
PMC03	# Of D Cache Storeback Requests	
PMC04	# Of Cycles Latency To Start Of I Cache Reload Requests	
PMC05	# Of Cycles Latency To Start Of D Cache Reload Requests	
PMC06	# Of Cycles Latency To Start Of D Cache Storeback Requests	
PMC07	# Of Cycles To Complete I Cache Reload Requests	
PMC08	# Of Cycles To Complete D Cache Reload Requests	
PMC09	# Of Cycles To Complete D Cache Storeback Requests	
PMC10	# Of Cycles That Memory Address Bus Is Busy	
PMC11	# Of Cycles That Memory Bus Hold (From Memory System) is asserted	
PMC12	# Of Cycles That Data Is Being Moved On Memory Bus	
PMC13	# Of Sio Bus Requests	
PMC14	# Of Cycles Sio Bus Is Busy (Utilization, includes dead cycles)	
PMC15	# Of Cycles That Data Is Being Moved On Sio Bus By Dma	
PMC16	# Of Dma Read Requests	
PMC17	# Of Dma Write Requests	
PMC18	# Of Cycles Of Dma Read Latency	
PMC19	# Of Cycles Of Dma Write Latency	
PMC20	# Of External Interrupts	
PMC21	# Of Cycles Iocu A, B, C, or D Request Latency (Any One Active)	

Appendices

A.1 List Of Fixed Ops 10 Bit Decode (Sorted By Mnemonic)

OPCODE	BIN	DEC	HEX
a	0000000000001010	10	00a
abs	0000000101101000	360	168
ae	0000000010001010	138	08a
ai	0000000110001111	399	18f
air	0000000110101111	431	1af
ame	0000000011101010	234	0ea
and	0000000000011100	28	01c
andc	0000000000111100	60	03c
andil	0000000110011111	415	19f
andiu	0000000110111111	447	1bf
aze	0000000011001010	202	0ca
cal	0000000111001111	463	1cf
cau	0000000111101111	495	1ef
cax	0000000100001010	266	10a
clcs	0000001000010011	531	213
clf	0000000001110110	118	076
cli	0000000111110110	502	1f6
cmp	0000000000000000	0	000
cmpi	0000000101101111	367	16f
cmpl	0000000000100000	32	020
cmpli	0000000101001111	335	14f
cntlz	0000000000011010	26	01a
dclst	0000001001110110	630	276
dclz	0000001111110110	1014	3f6
dcs	0000001001010110	598	256
div	0000000101001011	331	14b
divs	0000000101101011	363	16b
doz	0000000100001000	264	108
dozi	0000000100101111	303	12f
eqv	0000000100011100	284	11c
exts	0000001110011010	922	39a
l	0000000000010111	23	017
lbrx	0000001000010110	534	216
lbz	0000000001010111	87	057
lbzu	0000000001110111	119	077
lbzux	0000000001110111	119	077
lbzx	0000000001010111	87	057
lfd	0000001001010111	599	257
lfdu	0000001001110111	631	277
lfdux	0000001001110111	631	277
lfdx	0000001001010111	599	257
lfs	0000001000010111	535	217
rlimi	0000000010011111	159	09f

A.1 List Of Fixed Ops 10 Bit Decode (Sorted By Mnemonic Continued)

OPCODE	BIN	DEC	HEX
lfsu	0000001000110111	567	237
lfsux	0000001000110111	567	237
lfsx	0000001000010111	535	217
lha	0000000101010111	343	157
lhau	0000000101110111	375	177
lhaux	0000000101110111	375	177
lhax	0000000101010111	343	157
lhbrx	0000001100010110	790	316
lhz	0000000100010111	279	117
lhzu	0000000100110111	311	137
lhzux	0000000100110111	311	137
lhzx	0000000100010111	279	117
lm	0000000111010111	471	1d7
lscbx	0000000100010101	277	115
lsi	0000001001010101	597	255
lsx	0000001000010101	533	215
lu	0000000000110111	55	037
lux	0000000000110111	55	037
lx	0000000000010111	23	017
maskg	0000000000011101	29	01d
maskir	0000001000011101	541	21d
mcrxr	0000001000000000	512	200
mfcr	0000000000010011	19	013
mfmsr	0000000001010011	83	053
mfspr	0000000101010011	339	153
mfsr	0000001001010011	595	253
mfsri	0000001001110011	627	273
mtcrf	0000000010010000	144	090
mtmsr	0000000010010010	146	092
mtspr	0000000111010011	467	1d3
mtsr	0000000011010010	210	0d2
mtsri	0000000011110010	242	0f2
mul	0000000001101011	107	06b
muli	0000000011101111	239	0ef
muls	0000000011101011	235	0eb
nabs	0000000111101000	488	1e8
nand	0000000111011100	476	1dc
neg	0000000001101000	104	068
nor	0000000001111100	124	07c
or	0000000110111100	444	1bc
orc	0000000110011100	412	19c
oril	0000000100011111	287	11f
oriu	0000000100111111	319	13f

A.1 List Of Fixed Ops 10 Bit Decode (Sorted By Mnemonic Continued)

OPCODE	BIN	DEC	HEX
rac	0000001100110010	818	332
rlinm	0000000010111111	191	0bf
rlmi	0000000011011111	223	0df
rlnm	0000000011111111	255	0ff
rrib	0000001000011001	537	219
sf	0000000000001000	8	008
sfe	0000000010001000	136	088
sfi	0000000100001111	271	10f
sfme	0000000011101000	232	0e8
sfze	0000000011001000	200	0c8
sl	0000000000011000	24	018
sle	0000000010011001	153	099
sleq	0000000011011001	217	0d9
sliq	0000000010111000	184	0b8
slliq	0000000011111000	248	0f8
sllq	0000000011011000	216	0d8
slq	0000000010011000	152	098
sr	0000001000011000	536	218
sra	0000001100011000	792	318
srai	0000001100111000	824	338
sraiq	0000001110111000	952	3b8
sraq	0000001110011000	920	398
sre	0000001010011001	665	299
srea	0000001110011001	921	399
sreq	0000001011011001	729	2d9
sriq	0000001010111000	696	2b8
srliq	0000001011111000	760	2f8
srlq	0000001011011000	728	2d8
srq	0000001010011000	664	298
st	0000000010010111	151	097
stb	0000000011010111	215	0d7
stbrx	0000001010010110	662	296
stbu	0000000011110111	247	0f7
stbux	0000000011110111	247	0f7
stbx	0000000011010111	215	0d7
stfd	0000001011010111	727	2d7
stfdu	0000001011110111	759	2f7
stfdux	0000001011110111	759	2f7
stfdx	0000001011010111	727	2d7
stfs	0000001010010111	663	297
stfsu	0000001010110111	695	2b7
stfsux	0000001010110111	695	2b7
stfsx	0000001010010111	663	297
sth	0000000110010111	407	197

A.1 List Of Fixed Ops 10 Bit Decode (Sorted By Mnemonic Continued)

OPCODE	BIN	DEC	HEX
sthbrx	0000001110010110	918	396
sthu	0000000110110111	439	1b7
sthux	0000000110110111	439	1b7
sthx	0000000110010111	407	197
stm	0000000111110111	503	1f7
stsi	0000001011010101	725	2d5
stsx	0000001010010101	661	295
stu	0000000010110111	183	0b7
stux	0000000010110111	183	0b7
stx	0000000010010111	151	097
t	0000000000000100	4	004
ti	0000000001101111	111	06f
tlbi	0000000100110010	306	132
xor	0000000100111100	316	13c
xoril	0000000101011111	351	15f
xoriu	0000000101111111	383	17f

There are a total of 146 FXU ops

A.2 List Of Fixed Ops 10 Bit Decode (Sorted By Decode)

OPCODE	BIN	DEC	HEX
cmp	0000000000000000	0	000
t	0000000000000100	4	004
sf	0000000000001000	8	008
a	0000000000001010	10	00a
mfcr	0000000000010011	19	013
l	0000000000010111	23	017
lx	0000000000010111	23	017
sl	0000000000011000	24	018
cntlz	0000000000011010	26	01a
and	0000000000011100	28	01c
maskg	0000000000011101	29	01d
cmpl	0000000000100000	32	020
lu	0000000000110111	55	037
lux	0000000000110111	55	037
andc	0000000000111100	60	03c
mfmsr	0000000001010011	83	053
lbz	0000000001010111	87	057
lbzx	0000000001010111	87	057
neg	0000000001101000	104	068
mul	0000000001101011	107	06b
ti	0000000001101111	111	06f
clf	0000000001110110	118	076
lbzu	0000000001110111	119	077
lbzux	0000000001110111	119	077
nor	0000000001111100	124	07c
sfe	0000000010001000	136	088
ae	0000000010001010	138	08a
mtcrf	0000000010010000	144	090
mtmsr	0000000010010010	146	092
st	0000000010010111	151	097
stx	0000000010010111	151	097
slq	0000000010011000	152	098
sle	0000000010011001	153	099
rlimi	0000000010011111	159	09f
stu	0000000010110111	183	0b7
stux	0000000010110111	183	0b7
sliq	0000000010111000	184	0b8
rlinm	0000000010111111	191	0bf
sfze	0000000011001000	200	0c8
aze	0000000011001010	202	0ca
mtsr	0000000011010010	210	0d2
stb	0000000011010111	215	0d7
stbx	0000000011010111	215	0d7
sllq	0000000011011000	216	0d8

A.2 List Of Fixed Ops 10 Bit Decode (Sorted By Decode Continued)

OPCODE	BIN	DEC	HEX
sleq	0000000011011001	217	0d9
rlmi	0000000011011111	223	0df
sfme	0000000011101000	232	0e8
ame	0000000011101010	234	0ea
muls	0000000011101011	235	0eb
muli	0000000011101111	239	0ef
mtsri	0000000011110010	242	0f2
stbu	0000000011110111	247	0f7
stbux	0000000011110111	247	0f7
slliq	0000000011111000	248	0f8
rlnm	0000000011111111	255	0ff
doz	0000000100001000	264	108
cax	0000000100001010	266	10a
sfi	0000000100001111	271	10f
lscbx	0000000100010101	277	115
lhz	0000000100010111	279	117
lhzx	0000000100010111	279	117
eqv	0000000100011100	284	11c
oril	0000000100011111	287	11f
dozi	0000000100101111	303	12f
tlbi	0000000100110010	306	132
lhzu	0000000100110111	311	137
lhzux	0000000100110111	311	137
xor	0000000100111100	316	13c
oriu	0000000100111111	319	13f
div	0000000101001011	331	14b
cmpli	0000000101001111	335	14f
mfspr	0000000101010011	339	153
lha	0000000101010111	343	157
lhax	0000000101010111	343	157
xoril	0000000101011111	351	15f
abs	0000000101101000	360	168
divs	0000000101101011	363	16b
cmpi	0000000101101111	367	16f
lhau	0000000101110111	375	177
lhaux	0000000101110111	375	177
xoriu	0000000101111111	383	17f
ai	0000000110001111	399	18f
sth	0000000110010111	407	197
sthx	0000000110010111	407	197
orc	0000000110011100	412	19c
andil	0000000110011111	415	19f
air	0000000110101111	431	1af
sthu	0000000110110111	439	1b7

A.2 List Of Fixed Ops 10 Bit Decode (Sorted By Decode Continued)

OPCODE	BIN	DEC	HEX
sthux	0000000110110111	439	1b7
or	0000000110111100	444	1bc
andiu	0000000110111111	447	1bf
cal	0000000111001111	463	1cf
mtspr	0000000111010011	467	1d3
lm	0000000111010111	471	1d7
nand	0000000111011100	476	1dc
nabs	0000000111101000	488	1e8
cau	0000000111101111	495	1ef
cli	0000000111110110	502	1f6
stm	0000000111110111	503	1f7
mcrxr	0000001000000000	512	200
clcs	0000001000010011	531	213
lsx	0000001000010101	533	215
lbrx	0000001000010110	534	216
lfs	0000001000010111	535	217
lfsx	0000001000010111	535	217
sr	0000001000011000	536	218
rrib	0000001000011001	537	219
maskir	0000001000011101	541	21d
lfsu	0000001000110111	567	237
lfsux	0000001000110111	567	237
mfsr	0000001001010011	595	253
lsi	0000001001010101	597	255
dcs	0000001001010110	598	256
lfd	0000001001010111	599	257
lfdx	0000001001010111	599	257
mfsri	0000001001110011	627	273
dclst	0000001001110110	630	276
lfdu	0000001001110111	631	277
lfdux	0000001001110111	631	277
stsx	0000001010010101	661	295
stbrx	0000001010010110	662	296
stfs	0000001010010111	663	297
stfsx	0000001010010111	663	297
srq	0000001010011000	664	298
sre	0000001010011001	665	299
stfsu	0000001010110111	695	2b7
stfsux	0000001010110111	695	2b7
sriq	0000001010111000	696	2b8
stsi	0000001011010101	725	2d5
stfd	0000001011010111	727	2d7
stfdx	0000001011010111	727	2d7
srlq	0000001011011000	728	2d8

A.2 List Of Fixed Ops 10 Bit Decode (Sorted By Decode Continued)

OPCODE	BIN	DEC	HEX
sreq	0000001011011001	729	2d9
stfdu	0000001011110111	759	2f7
stfdux	0000001011110111	759	2f7
srliq	0000001011111000	760	2f8
lhbrx	0000001100010110	790	316
sra	0000001100011000	792	318
rac	0000001100110010	818	332
srai	0000001100111000	824	338
sthbrx	0000001110010110	918	396
sraq	0000001110011000	920	398
srea	0000001110011001	921	399
exts	0000001110011010	922	39a
sraiq	0000001110111000	952	3b8
dclz	0000001111110110	1014	3f6

There are a total of 146 FXU ops