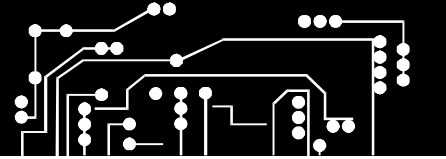


PowerPC Embedded Processors Application Note



401 Interrupt Latency Calculation

IBM Microelectronics
Dept D95/Bldg 060
3039 Cornwallis Road
Research Triangle Park, NC 27709
Version: 1.0

November 4, 1996

This paper outlines a method for determining the number of PowerPC 401 instructions which can be executed between worst case interrupt occurrences from an MPEG2 transport decoder. The method of determining PowerPC 401 CPU core interrupt latencies may be extended to other scenarios.

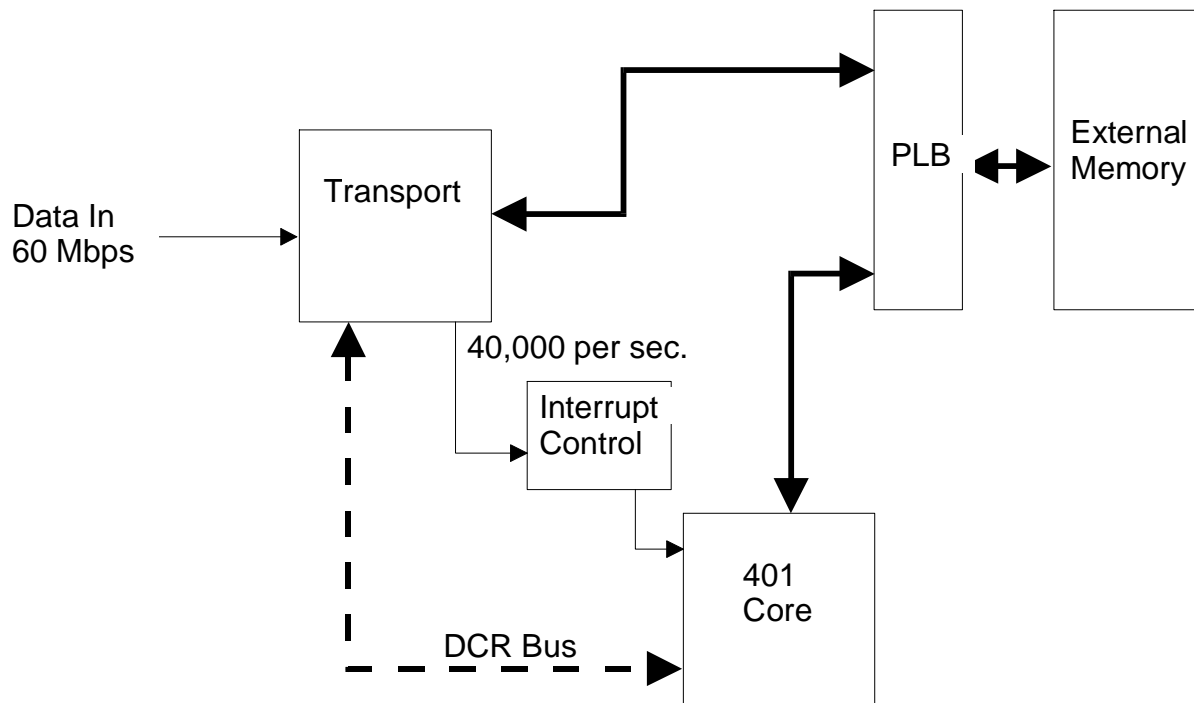


Figure 1: CPU-Transport Interconnection.

Figure 1 shows the relationship between the transport demultiplexor and the 401 core. Two buses are available for transport of data. The device control register bus provides the CPU the ability to directly read and write the transport demultiplexor internal registers. Read or writes to device DCR's require 3 cycles minimum to complete the transaction. The other avenue of data exchange is through system memory via the PLB. For a 32-bit data bus at 40.5 MHz, 4-2-2-2-2 page mode access is assumed using EDO memory. Assume a 60 Mbps data stream input to the transport, then for a

188 byte packet length there are approximately 40,000 packets per second. The transport chip will not interrupt the CPU for every packet, however, it is possible that two back to back interrupts may be generated. Thus the minimum interrupt service time must not exceed

$$\frac{1}{40,000} = 25.00 \mu\text{s}.$$

A 401 operating at 40.5 MHz can theoretically execute one instruction per clock cycle (24.69 ns) if operating totally out of cache. This scenario is generally not the case so the number of cycles per instruction (CPI) is used to determine the raw performance in an actual environment. Table 1 shows the number of instructions that can be executed within a 25.0 μs window for different CPI's.

CPI	# instructions = $\frac{\text{Frequency}}{\text{CPI}} 25.00e - 6$
1	1012.5
1.5	675.0
2	506.3
2.5	405.0
3	337.5
3.5	289.3
4	253.1
4.5	225.0
5	202.5
5.5	184.1
6	168.8

Table 1: CPI vs. # of executable instructions at 40.5 MHz

There are 3 questions which must be answered in order to determine whether the CPU can handle the interrupt in the given amount of time.

1. How many cycles elapse between an active signal on the interrupt line and the execution of the first instruction of the interrupt subroutine?
2. How many cycles does it take to save and restore the GPR's?
3. How many instructions are executed in the interrupt handler?

The answers to these questions cannot be determined precisely, however, they can be estimated to provide general boundaries for code size and the number of GPR's saved/restored.

The first step is to determine the boundaries for interrupt vector load. The worst case scenario is an interrupt occurring while an instruction cache line fill is beginning, a data

cache line load from guarded storage is being decoded in the pipeline, a data cache flush is pending, the interrupt vector is in DRAM, and a memory refresh occurs. All this works out to 49 cycles for 4-2-2-2-2 DRAM timing and 42 cycles for 3-2-2-2-1 timing. The best case scenario is: an interrupt occurs while the processor is executing out of I-cache, the pipe is not locked into a load/store miss, and the interrupt vector is in cache. This takes 7 cycles. For analysis purposes, the worst case timing is used, although the occurrence of this scenario will be rare. A more typical case would be an interrupt occurring while the processor is executing out of I-cache, but the interrupt vector is not in cache which gives an average interrupt to execution latency of 16 cycles (3 cycle latency, average of 6 cycles from previous fill, 7 cycles to get 1st instruction to EXE). Appendix A in the back of this note shows the breakdown of the cycles.

The second step is to determine the save/restore time for the general purpose registers. Worst case timing occurs when none of the storage addresses for 32 GPR's are in cache (a D-side fill must be performed first) and each fill causes a flush. Upon receiving the command to store the 32 GPR's, the 401 begins a series of cache line fills and flushes in the following order: fill, flush, fill, flush, fill, flush,, fill flush.

It is easiest to break up a series of GPR stores into fill, flush-fill, flush-fill,, flush because the flush-fills represent an easily distinguished repeating pattern. The first fill requires 2 cycles for data cache access then N cycles (N is external memory dependent) for a D-side fill before the next store can be executed. Once the fill is complete, the following 3 GPR stores are executed at 2 cycles per store since the line is now in cache. The total for the first 4 GPR stores store is $8+N$ cycles. For 4-2-2-2-2 timing $N = 12$ and for 3-2-2-2-1 timing $N = 10$. The length of a flush-fill sequence is dependent upon memory timing. For 4-2-2-2-2 DRAM timing, 24 cycles are required and 3-2-2-2-1 requires 21 cycles. In a fill-flush, the 3 following GPR stores overlap with 6 cycles of the flush. So in a flush-fill, those 6 cycles are not counted again. As in previous fill-flushes, the final flush is partially overlapped by the last 3 GPR stores so only 6 cycles (for 4-2-2-2-2) or 5 cycles (for 3-2-2-2-1) need to be added. The total time required to store 32 GPR's (using 4-2-2-2-2 timing) is $20 + 7*24 + 6 = 194$ cycles. A 32 GPR restore will require $32*2 = 64$ cycles since these memory addresses are usually not modified by the ISR. The pessimistic delay scenario for a 32 GPR save/restore is 258 cycles. Likewise, using 3-2-2-2-1 results in 236 cycles for a 32 GPR save/restore. This delay can be reduced by storing less registers to a dedicated section of locked D-cache.

The third step is to determine the instruction execution time for the ISR. This is the most difficult parameter to estimate. It requires knowledge of the number of instructions in the handler, the number of I and D cache misses and the number of flushes caused by misses. To make the discussion more general and the equations less complex, assume linear execution of code. This is not unreasonable for an ISR where branching can be sparse.

A pessimistic scenario would be all instructions are in memory (25% I cache misses, 75% I cache hits since the following three words in a fill are considered a hit), 40% of all instructions are load/stores, 20% of the load/stores miss in the D cache and 25% of the misses initiate a cache flush. A typical scenario is all ISR instructions are locked in cache (100% I cache hits), 30% of the instructions are load/stores, 15% of the load/stores miss the D cache and 25% of the misses initiate a flush. The total number of cycles required to execute N instructions for the pessimistic scenario using 4-2-2-2-2 timing is:

$$N(7.5) + (0.25)(N) * 2 + (0.40)(0.40)(0.20)(12) + (0.40)(0.20)(0.25)(12) = 5.1N$$

The total number of instructions that can be executed in 25.0 μ s, given a maximum latency and storing 32 GPR's, is

$$N_c = \frac{1012 - 49 - 258}{51} = 138.$$

The total number of cycles required to execute N instructions for the typical scenario is:

$$N(0.61225100) + N = 1.975N$$

The total number of instructions that can be executed in 25.0 μ s with this scenario is

$$N_t = \frac{1012 - 49 - 258}{1.975} = 357.$$

So, given the time window for the interrupt, the number of executed instructions can be estimated by finding the Cycles per Instruction (CPI) value using a given hit/miss scenario. Table 2 describes several scenarios for instruction cache hit/miss ratios.

Table 2: Options for hit/miss ratios.

Instruction Scenario	Definition	ICache Hit Ratio	ICache Miss Ratio
1	100% instructions in cache	1.0	
2	75% of instructions in cache	0.94	0.06
3	50% of instructions in cache	0.88	0.12
4	25% of instructions in cache	0.82	0.18
5	0% of instructions in cache	0.75	0.25
6	0% of instructions in cache w/ branching	0.7	0.3

The CPI for each scenario is determined by the following generalized equation:

$$\begin{aligned} \text{CPI} * N = & N * (\% \text{Icache hit}) + \\ & N * (\% \text{Icache miss}) (X + 3Y + Z - 1) + \\ & N * (\% \text{load/store}) + \\ & N * (\% \text{load/store}) (\% \text{Dcache miss}) (X + 3Y + Z) + \\ & N * (\% \text{load/store}) (\% \text{Dcache miss}) (\% \text{Flush}) (X + 3Y + Z) \end{aligned}$$

where the DRAM cycle time is X-Y-Y-Y-Z for flushes and fills. The cycle time section of the second term in the above equation is defined as follows:

$$X + 1 + 1 + 1 + 3(Y-1) + Z - 1 = X + 3Y + Z - 1$$

where

- X is the DRAM first access delay.
- 1 cycle is added for BIU arbitration.
- 1 cycle is added for instruction transfer through the BIU.
- 1 cycle is added for instruction decode.
- 1 cycle is subtracted for each instruction following the first access since they are considered cache hits and accounted for in the first term.
- 1 cycle is subtracted from the precharge delay (Z) since a cycle is hidden by the BIU arbitration term.

The cycle times for the D side terms represent the scenario of each miss requiring a random access to memory. This means that you pay the full fill/flush latency (X + 3Y + Z) for each D side miss. Better performance is generally expected since a D side load can be accomplished target word first, in which case, the required data is sent directly to the execution unit. The bus is tied up while loading the rest of the cache line, however, instructions may be executed out of I cache. For stores, the 401 has a feature which allows I-side fetches to have priority over random D-side stores. This is accomplished by queuing the store until either instructions are running from cache or another store comes along. If another store request arrives, the priority of the queued

Scenario	% I Hit	% I Miss	0.3	0.3	0.4	0.4	0.5	0.5	% L/S
			0.15	0.2	0.2	0.4	0.4	0.6	% D miss
1	1	0	1.98	2.20	2.60	3.80	4.50	6.00	
2	0.94	0.06	2.64	2.86	3.26	4.46	5.16	6.66	
3	0.88	0.12	3.30	3.52	3.92	5.12	5.82	7.32	
4	0.82	0.18	3.96	4.18	4.58	5.78	6.48	7.98	
5	0.75	0.25	4.73	4.95	5.35	6.55	7.25	8.75	
6	0.7	0.3	5.28	5.50	5.90	7.10	7.80	9.30	

Table 3: Cycles per Instruction for different scenarios. (Assumes 25% flush to fill ratio)

D-side store is elevated above I-side requests guaranteeing that the store will be the next operation completed. The second store is then put in the queue, with the lower priority, until one of the two above events occurs. This enhancement can reduce the CPI by as much as 6% thus increasing the number of executable instructions. Table 3 shows the results of this general equation for multiple instruction scenarios and load/store ratios. These results are 'worst case' in the sense that they are derived using the worst case latency and storing 32 GPR's to memory. Note the effect that load/store and data cache miss ratio has on the CPI. Table 4 shows the number of instructions possible for a 25 μ s window given the CPI's generated in table 3.

Scenario	% ICache hit ratio	ICache miss ratio	0.3	0.3	0.4	0.4	0.5	0.5	load/store ratio
			0.15	0.2	0.2	0.4	0.4	0.6	D Cache miss ratio
1	1	0	357	321	271	186	157	118	
2	0.94	0.06	268	247	216	158	137	106	
3	0.88	0.12	214	200	180	138	121	96	
4	0.82	0.18	178	169	154	122	109	88	
5	0.75	0.25	149	143	132	108	97	81	
6	0.7	0.3	134	128	120	99	90	76	

Table 4: Number of instructions possible within 25 μ s window.

OPTIMIZATIONS

Typically, an ISR is not guaranteed to be in cache so scenario 5 generally applies. There are several ways to increase the number of instructions available during the interrupt execution window. The most obvious is to use the cache locking feature of the 401 to lock in segments of the ISR into the instruction cache. This can result in a 50-100% increase in the number of executable instructions. Likewise, frequently used data can be locked into the data cache to reduce load/store latency resulting in a 100% increase in the number of executable instructions. Another technique is to store less registers. For instance, storing 8 GPR's rather than 32 results in a 23.4% increase in the number of instructions. Also, saving the GPR's to locked cache rather than DRAM memory results in an 18% increase in executable instructions for 32 GPRs (4.6% for 8 GPR's). Tables 5 and 6 show the CPI and number of instructions possible if all the load stores are to a locked section of data cache and only 8 GPR's are saved to cache. Note the dramatic improvements as higher percentages of code and data are locked into cache. The downside to locking everything in cache is the possible loss of performance. Each cache line locked up for an ISR takes that line away from normal execution. IBM's cache sizes are modular so the system designer can increase cache sizes to negate possible performance impacts. Once the ISR code is known, programmers can tune the routines to achieve the required CPI while minimizing the amount of dedicated cache space. For example: code which is executed every interrupt can be locked in while conditional code remains in DRAM, unnecessary fills

can be eliminated from GPR saves by using the data cache block zero (DCBZ) instruction on the memory save area, and data can be locked into data cache to eliminate unnecessary flushing at the end of the ISR. Tables 3 and 4 provide a rough guide for estimating performance. A spreadsheet is recommended for analysis since a number of variables are involved.

Scenario	ICache hit ratio	ICache miss ratio	0.3	0.4	0.5	load/store ratio
1	1	0	1.30	1.40	1.50	
2	0.94	0.06	1.96	2.06	2.16	
3	0.88	0.12	2.62	2.72	2.82	
4	0.82	0.18	3.28	3.38	3.48	
5	0.75	0.25	4.05	4.15	4.25	
6	0.7	0.3	4.60	4.70	4.80	

Table 5: CPI calculations using available optimizations.

Scenario	ICache hit ratio	ICache miss ratio	0.3	0.4	0.5	load/store ratio
1	1	0	716	665	621	
2	0.94	0.06	475	452	431	
3	0.88	0.12	355	342	330	
4	0.82	0.18	284	275	267	
5	0.75	0.25	230	224	219	
6	0.7	0.3	202	198	194	

Table 6: Number of instructions possible in 25 us window using optimizations.

Appendix A: Worst Case Interrupt Latency

Circumstance	Result	Cycles 4-2-2-2-2 Memory	Cycles 3-2-2-2-1 Memory
<ul style="list-style-type: none"> Interrupt pin is active on the input 		0	0
<ul style="list-style-type: none"> Interrupt makes its way through the first metastability latch A guarded D-side load operation is in the 1st cycle of the EXE stage. 		1	1
<ul style="list-style-type: none"> Interrupt makes its way through the second metastability latch. ICU makes a BIU request for an I-side line fill. D-side flush buffer currently full from a previous operation causing a pending flush. The guarded D-side load moves to the 2nd cycle of the EXE stage. 	<ul style="list-style-type: none"> BIU accepts ICU request for an I-side line fill. 	1	1
<ul style="list-style-type: none"> Interrupt makes its way through the internal core interrupt latch. The guarded D-side load moves to the 3rd cycle of the EXE stage. It is at this point where the instruction can no longer be aborted by the interrupt. The DCU resolves the pending D-side flush currently in the flush buffer and the guarded load miss by granting priority to the pending D-side flush and issues a request to the BIU (the flush buffer must be available for the current guarded load miss.) The ICU has not yet presented the interrupt vector line fill request to the BIU due to the fact that the ICU is currently engaged in the previous I-side line fill. The previous line fill must complete before a new request can be presented to the BIU. 	<ul style="list-style-type: none"> I-side line fill takes place. Pending D-side flush request currently in the flush buffer gets accepted by the BIU during the pre-charge cycle of the fill operation. 	4-2-2-2-2	3-2-2-2-1
<ul style="list-style-type: none"> The DCU is presenting the line fill request to the BIU. The ICU is presenting the vector miss line fill request to the BIU. The DRAM refresh timer times out and issues a DRAM refresh request to the BIU. 	<ul style="list-style-type: none"> Pending D-side flush takes place. DRAM refresh request gets accepted by the BIU during the pre-charge cycle of the flush operation. 	4-2-2-2-2	3-2-2-2-1
<ul style="list-style-type: none"> The DCU is still presenting the line fill request to the BIU. The ICU is still presenting the vector miss line fill request to the BIU. 	<ul style="list-style-type: none"> DRAM refresh takes place. The D-cache line fill gets accepted by the BIU. 	4	4
<ul style="list-style-type: none"> The ICU is still presenting the vector miss line fill request to the BIU. 	<ul style="list-style-type: none"> D-cache line fill takes place. I-cache vector miss line fill gets accepted by the BIU during the pre-charge cycle of the line fill. 	4-2-2-2-2	3-2-2-2-1
<ul style="list-style-type: none"> I-cache vector miss line fill takes place. First instruction is fetched by the BIU and forwarded via the cache bypass path (assume target word first). 		4	3
<ul style="list-style-type: none"> First instruction of the ISR reaches the BIU Data Register. 		1	1
<ul style="list-style-type: none"> First instruction of the ISR reaches the DEC stage of the pipeline. 		1	1
<ul style="list-style-type: none"> First instruction of the ISR reaches the EXE stage of the pipeline. 		1	1
Total		49	42

Appendix A: Typical Interrupt Latency

Circumstance	Result	Cycles 4-2-2-2-2 Memory	Cycles 3-2-2-2-1 Memory
• Interrupt pin is active on the input		0	0
• Interrupt makes its way through the first metastability latch		1	1
• Interrupt makes its way through the second metastability latch.		1	1
<ul style="list-style-type: none"> Interrupt makes its way through the internal core interrupt latch. Can not access the I-cache at this point due to the fact that the processor is busy executing out of cache or busy doing a line fill. 	<ul style="list-style-type: none"> Assume an average amount of cycles. Can be in the middle of a line fill (12 cycles max) to executing out of cache (1 cycle min) 	6	6
• I-fetch misses in the I-cache. The ICU presents the vector miss line fill request to the BIU.	• I-cache vector miss line fill gets accepted by the BIU.	1	1
•	• I-cache vector miss line fill takes place. First instruction is fetched by the BIU and forwarded via the cache bypass path (assume target word first).	4	3
•	• First instruction of the ISR reaches the BIU Data Register.	1	1
•	• First instruction of the ISR reaches the DEC stage of the pipeline.	1	1
•	• First instruction of the ISR reaches the EXE stage of the pipeline.	1	1
Total		16	15

Appendix A: Best Case Interrupt Latency

Circumstance	Result	Cycles 4-2-2-2-2 Memory	Cycles 3-2-2-2-1 Memory
• Interrupt pin is active on the input		0	0
• Interrupt makes its way through the first metastability latch		1	1
• Interrupt makes its way through the second metastability latch.		1	1
• Interrupt makes its way through the internal core interrupt latch. • Can not access the I-cache at this point due to the fact that the processor is busy executing out of I-cache.		1	1
• Interrupt vector address sent to I-cache.	• I-cache hit	1	1
• Cache outputs first ISR instruction		1	1
• First instruction of the ISR reaches the DEC stage of the pipeline.		1	1
• First instruction of the ISR reaches the EXE stage of the pipeline.		1	1
Total		7	7

IBM will continue to enhance products and services as new technologies emerge. Therefore, IBM reserves the right to make changes to its products, other product information, and this publication without prior notice. Please contact your local IBM Microelectronics representative on specific standard configurations and options.

IBM assumes no responsibility or liability for any use of the information contained herein. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. NO WARRANTIES OF ANY KIND, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE ARE OFFERED IN THIS DOCUMENT.