P5 Mon Help

P5Mon is a CPU monitoring tool for Pentium based machines running Windows 3.1.and Windows 95. It displays the activity of the CPU as an oscilloscope-like graph of two user-selectable CPU events.

Introduction

How To:

Use the Toolbar.
Change the Events being monitored.
Pause the Graph.
Speed up or Slow down the graph retrace
Change the Vertical or Horizontal scale.
Cut and paste the graph into another document.

Pentium Architecture Overview

About P5Mon

The ToolBar

Click on the toolbar area you wish to find out about. Alternatively, use the buttons marked << and >> above to browse through each help topic about the toolbar.

Introduction

When Intel designed the Pentium™ processor they included hardware support for profiling of software and measuring the performance of the CPU. Central to this support are two event counters which can be used to collect statistics on various parts of the CPU. Unfortunately, when the *Pentium™ Family Users Manual* was published, Intel reserved the details of these counters and other information and placed them into a separate supplement which can only be obtained by signing a Non Disclosure Agreement or NDA. The justification for this is that the information is `considered Intel confidential and proprietary` and non-essential to standard applications.

However Terje Mathisen, a systems architect from Norway, managed to reverse engineer the instruction format for accessing the two event counters by disassembling a commercial profiling tool sold by a company which had signed the NDA with Intel. He published his results in *Byte Magazine* in the July 1994 issue [1]. P5Mon was written using the information in this *Byte* article.

Graph Window

The Graph window displays two traces of different CPU events. One is coloured red and the other is coloured blue. The traces show the changes in the values of two internal Pentium profiling counters. The vertical scale shows the instantaneous number of events per second. Note that this is not the same as the absolute number of events between sample, the height of each plotted point is calculated as the difference between the sampled events divided by the elapsed time between samples. The elapsed time is judged extremely accurately by reading the CPU timestamp counter and dividing by the clock frequency. A grid is optionally displayed which divides the horizontal scale into units of 20 sample points and the vertical scale into tenths. It is toggled by the Show/Hide Grid button.

Changing the Vertical Scale

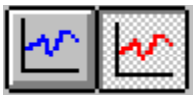
The vertical scale rescales itself automatically to fit the order of the number of events being counted. It always rescales to a multiple of 1, 2 or 5. If the application window is stretched vertically, the numeric value of the top of the scale stays the same but the graph is stretched to fit the new window dimensions.

Changing the Horizontal Scale

The horizontal scale remains constant while the window is stretched horizontally but the depth of the trace changes. Thus if the window is doubled in horizontal width, the red and blue traces will display a particular trace for twice as long before it is deleted by the left to right horizontal retrace. To stretch the traces or more correctly, to change the visible width between samples use the Magnification buttons. To change the speed of the traces, alter the Sampling Interval.

Cutting and Pasting the Graph Window

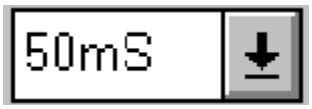
P5Mon does not support Cut and Paste directly but it is still possible to take a screen snapshot by pressing <Alt-PrintScreen>. This copies a bitmap image into the clipboard which can then be pasted into another document.



Select Red/Blue Graph

This two button pair controls which trace is selected by the Events listbox. Whichever color button is pressed then the list box entry shows the event being logged by the corresponding trace which can then be changed using the scrollbar.

Pause
This pauses the trace. The same effect can be achieved by pressing the **Pause** button on the keyboard.



Sampling Interval

This controls the frequency of sampling of the trace. The default is 10 times a second or a 100mS sampling interval.
The shortest interval is 50mS which gives a sample frequency of 20 times a second. The longest interval is 2 seconds

Decrease/Increase Magnification

The two buttons represented by magnifying glasses control the gap between each sample point on the screen. The smallest gap is 1 pixel, the largest is 8 pixels. The magnifying glass with a plus sign doubles the current sample width and the magnifying glass with the minus sign halves it.
The keypad plus and minus buttons have the same effect.

Rescale

The **Rescale** button causes both traces to be rescaled upwards to maximize vertical resolution on the scale. The largest sample value still visible (undeleted by the left to right scan) is used to calculate the new scales. This is useful if the

Pressing <Ctrl-R> has the same effect.

Equate Red & Blue Scales

Pressing this button causes both traces to be rescaled to the larger value of the two. This is useful for judging the relative quantities of two related events.
e.g. the ratio of Instructions Executed to Instructions Executed in V pipe gives a measure of instruction level parallelism.
<Ctrl-E> has the same effect.

Show/Hide Grid Lines

This button toggles the grid lines on the graph window. The grid is divided into tenths on the vertical scale and 20 sample points on the horizontal scale. Changing the magnification will change the grid width correspondingly.
<Ctrl-G> also toggles the grid lines.

Always On Top

When the **Always on Top** button is depressed the P5Mon window will remain visible even when you are using another application. This feature is useful to see the effect of different applications on the CPU without losing the P5Mon window underneath other application windows.
<Ctrl-A> also toggles **Always on Top**

Processor Info

Pressing the **Processor Info** button brings up a dialog with information about the CPU itself, including its clock speed, Model, Family, Type and manufacturing Stepping level. It also tests for the presence of the FDIV instruction bug by checking the equation:
(4195835/3145727)*4195835 - 3145727 = 0

The Pentium Architecture

This guide is not meant to be a definitive reference to the Pentium or its CPU architecture. It is just a brief tour of the main features. For more detailed information and explanation of the terms involved there is a page of references.

Some of the principle features of the Pentium CPU Architecture are described in the help topics listed below. You may also browse through this guide using the << and >> buttons.

Register Set

There are eight 32 bit registers on the Pentium used to hold instruction data and addresses. They are more or less general purpose in the sense that for most instructions any of these registers can be used as an operand. However, each of them has a particular function and some of them are more flexible than others. In 16 bit code only the lower 16 bits are normally useable.The registers are called EAX, EBX, ECX ,EDX, ESI, EDI, EBP and ESP. The register names are derived from their usual functions viz : Accumulator, Base, Count, Data, Source Index, Destination Index, Base Pointer and Stack Pointer. The lower 16 bits of the first four registers can be addressed separately and are called AX, BX, CX, DX. Within the lower 16 bit words of these registers the upper and lower bytes are named AH & AL, BH & BL, CH & CL, DH & DL.

Memory Architecture

A detailed discussion of the x86 memory architecture is beyond the scope of this document, for more information, the reader is referred to one of the many books on this topic.

Segmentation

All members of the x86 family support *segmented* memory. This is a form of addressing memory whereby all addresses are presented in the form of offsets from a segment base address. The address calculated by adding an offset to a segment base is called a *linear address.* For 16 bit code and code that retains compatibility with the Intel 80286 the segment base addresses are held in the segment registers CS, DS, ES, FS and GS. In 32 bit code the segment registers hold *selectors*. Selectors are offsets into arrays of data structures called descriptors. The descriptors hold the base addresses and other information regarding the access rights the page, whether it holds instructions or data etc.

Physical Memory and the Paging Unit

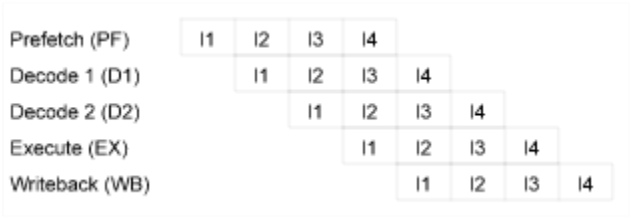
In Windows 3.1, Windows 95, Windows NT, Linux and other operating systems, the linear address is translated into a physical address using the paging unit. This allows the processor to maintain the illusion that there is more physical memory available for programs that actually exists by using hard disk storage for linear memory pages that are not currently needed. When the processor needs a page that is not currently in memory, the processor must fetch it from disk. The paging unit maintains data structures in memory called Page Tables which indicate the state of individual pages. Page Tables are composed of Page Table Entries (PTEs) which are used to hold the physical addresses corresponding to linear ones. Because the page size is 4 kBytes, the lower 12 bits of a linear address are the same as its correponding physical one. The remaining bits of the address are used in a lookup mechanism involving the Page Tables.

Translation Lookaside Buffer

In order to avoid a costly page table lookup for each memory access, the processor keeps a cache of recently used PTEs. This is called the Translation Lookaside Buffer or TLB.

Integer Instruction Pipeline

In common with its predecessor, the 486, Pentium™ instruction execution is pipelined. The structure of the 486 pipeline is shown below.



There are five pipeline stages which function as follows:

PF	Prefetch the next instruction from the on-chip cache or memory.
D1 & D2	Decode the instruction.
EX	Execute instruction in the ALU and access the Data cache
WB	Write back the results and modify the processor state.

Because the Pentium™ is a superscalar processor capable of issuing two instructions in parallel there are two five stage pipelines.

The Pentium has two independent prefetch buffers

Floating Point Pipeline

Pipelining floating point instructions is similar to that for integer instructions for the first five stages:

- PF Prefetch
- D1 Decode
- D2 Generate Address
- EX Read operands from memory and/or registers. Convert internal FP data to external memory format and write to memory.
- X1 FP Execute stage one. Convert external memory data to internal FP format and write to FP register file.
- X2 FP Execute stage two
- WF Round result and write to FP register file.
- ER Report errors and update FP status word.

In addition to these stages, the Pentium can write bypass the register file in stages X1 and WF, passing the result directly to EX stage of a following instruction.

With pipelining it is possible to achieve a throughput of 1 per clock cycle for certain floating point instructions including FMUL, FADD, FSUB, FST(i) and FLD.

Generally speaking, only one floating point instructions can be executed at a time. The second instruction pipe is not used for pairing floating point instructions except under limited circumstances.

See Floating Point Pairing Rules.

Integer Instruction Pairing Rules

The Pentium can issue two instructions subject to the following rules:

1) Both Instructions must belong to the following subset of simple instructions:

- mov reg, reg/mem/imm
- mov mem, reg/imm
- alu reg, reg/mem/imm
- alu mem. reg/imm
- inc reg/mem
- dec reg/mem
- push reg/mem
- pop reg
- lea reg/mem
- jmp/call/jcc near

All these instructions are implemented by hardwired logic on the Pentium and thus do not require microcode to execute.

2) There must be no register dependencies between them. There are two types of dependencies, read-after-write (RAW) and write-after-write. Here is an example of a RAW dependency :

```
mov ecx, eax
inc ecx
```

3) In addressing memory, neither instruction can use both a displacement and an immediate value e.g. The following instruction cannot be paired:

```
add [eax +4], 012345678h
```

4) Instructions with prefixes (e.g. segment override, Address size) can only occur in the U pipe. The exception to this is the near conditional jump instruction JCC (prefix 0fh)

Floating Point Instruction Pairing Rules

Floating point instructions cannot generally be paired. The exception to this is with the new FXCH floating point exchange instruction which can be paired if it immediately follows one of the following types of instructions: FLD single/double, FLD ST(i), FADD, FSUB, FMUL, FDIV, FCOM, FUCOM, FTST, FABS, FCHS.

The FXCH instruction can be used to overcome the stack bottleneck caused by the requirement that each floating point instruction have one source operand indexed from the top of the stack.

Floating point instructions cannot be paired with integer instructions. When two floating point instruction are paired the pair should not be followed by an integer instruction. The integer instruction will stall for one clock cycle or four clock cycles if the processor determines that the floating point pair are likely to generate a floating point exception (e.g. overflow, underflow). They should pipeline with integer instructions though, because the FPU logic is separate from the integer execution unit.

Branch Prediction

Every time the Pentium executes a branch instruction it stores the target address in a buffer. If the branch is encountered again, the instruction prefetch logic tries to guess the next instruction address based on the outcome of the last time the branch was executed. If it guesses correctly, the branch will take only one cycle. If it is incorrect, the prefetch queues are flushed and refilled from the correct address. This incurs a penalty of between 3 and 5 clock cycles. The Pentium can store up to 256 branches in its Branch Target Buffer or BTB.

On Chip Cache

A cache is a small area of fast memory for keeping frequently accessed data. The trade-off with normal memory is the high cost of fast memory. In the case of the Pentium, the on chip cache is an expense of chip area and transistors. The cache does not reside in a separately addressed area. Rather there is sophisticated logic used to copy the most frequently (or more *recently*) used regions of memory into the cache and copy them back when they are no longer needed or more likely when the precious resources of the cache are needed to store a more urgent datum.

Cache Terminology

Writethrough & Writeback

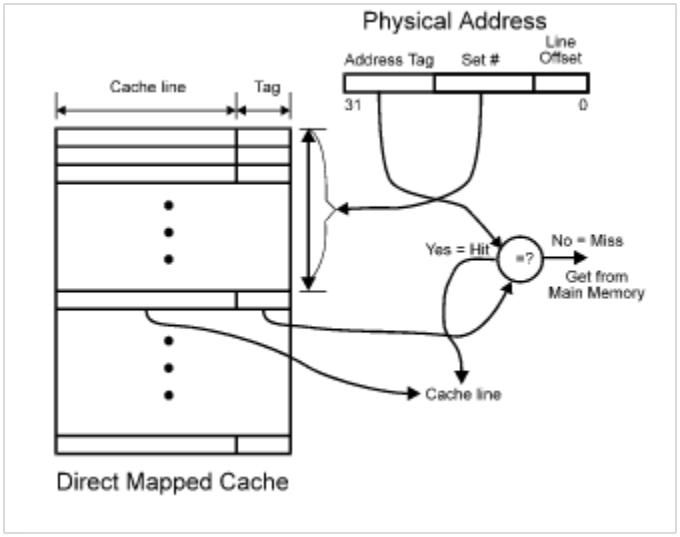
With a Writethrough cache all updates to addresses that hit in the cache are performed on main memory. Depending on the cache implementation, the cache line is either updated or invalidated. With a writeback cache, the updates are performed on the cache line alone. These two terms only apply to data cache lines, code is normally read only.

Cache Line size

This is the minimum unit of memory that the cache loads and stores. On the Pentium this is 32bytes or 4 memory accesses via its 64bit bus. It is always accessed on a cache line boundary which means that for the Pentium, the beginning of the cache line is always accessed with address lines a4 - a0 = 00000bin and the last byte of the cache line is accessed with a4-a0= 11111bin

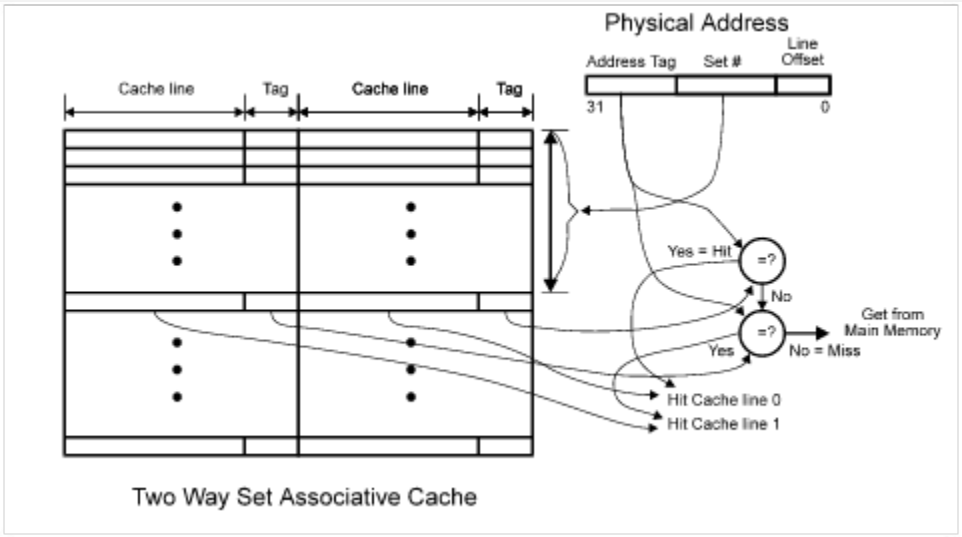
Address Tag and Set number:

When a physical address is presented to the cache, the cache uses the address bits to determine whether or not the cache line resides in the cache or in main memory. It does this by dividing the remaining address bits (a31 - a5 on the Pentium) into two fields. The low order field is called the Set number and this is used as an index into a linear array of cache lines (See diagram below). The upper field is called the Address Tag compared with the Tag stored with the cache line.



Direct Mapped Vs Set Associative Caches

The operation of a *Direct Mapped* cache is shown above. For a *Set Associative* cache the whole system is multiplied by two, three or more. Instead of comparing with one cache tag, the address tag is compared with a number of tags in separate banks. Each bank operates like a direct-mapped cache in its own right. In addition, when a new datum replaces an old one in the same set, it uses a algorithm to determine which of the two (or four in a four -way set associative cache) it should replace. One such algorithm is the Least-Recently-Used or LRU algorithm which tries to replace the cache line which whose last reference is the furthest back in time. A diagram of a two-way set associative cache is shown below.



The Pentium on chip cache is 16Kbytes in size and is divided into an 8Kbytes instruction or code cache and 8Kbytes data cache. Each cache is organized as a two way set associative cache.

Data Cache

The Pentium data cache is 8Kbytes in size and is two-way set associative. There are 128 sets of two 32byte lines. Each cache line has for read/write ports, one for each 4 byte bank. If two paired instructions try to access the same bank, a one clock cycle delay is incurred. This is called a Bank Conflict.

MESI Cache Coherency

The data cache supports the MESI protocol. It stands for Modified, Exclusive, Shared and Invalid which are the names of the four states of a data cache line in a multiprocessor configuration. A cache coherency protocol is an algorithm which ensures that if one CPU reads a datum, subsequent reads or writes to the same datum by other CPUs will access the correct data even if the only copy of the data is in one CPUs cache. Cache coherency protocols are quite involved. For a detailed explanation of MESI and other algorithms see [3] & [4].

Code Cache

The Pentium instruction cache organization is the same as for the data cache, 8Kbytes, two way set associative. Not all instructions fetched are executed due to the effects of branch prediction.

Instruction Set Extensions

A number of new instructions have been added to the 486 instruction set architecture for the Pentium.

CMPXCHG8B Compare and Exchange 8 Bytes

This instruction compares the contents of register EDX:EAX with memory and swaps them if they are equal. It is useful for implementing synchronisation in multiprocessors because it supports the LOCK instruction prefix.

RDTSC Read Timestamp Instruction

The Read Timestamp instruction (RDTSC) copies the contents of a 64 bit clock cycle counter into registers EAX and EDX and is a convenient way of profiling code. The counter is reset on power up and processor reset. The instruction encoding is `0f31h`

The following macro can be used in Windows Protected Mode 16 bit code:

```
#define TIMESTAMP(var)  __asm \
{ \
_asm _emit 0x0f \
_asm _emit 0x31 \
_asm _emit 0x66 \
_asm mov word ptr var, ax \
_asm _emit 0x66 \
_asm mov word ptr var[+4], dx \
}
```

and is used thus:

```
DWORD timst[2];

TIMESTAMP(timst);
```

CPUID CPU Identification Instruction

This useful instruction allows the programmer to identify the make and model of the CPU and the extra features it supports. Its output is controlled by register EAX. If EAX contains 0hex when CPUID is executed, the ASCII string representing the CPU vendor is placed into registers EBX, ECX, EDX. EAX contains an integer representing the highest input value to the instruction. This is currently 1hex. For Intel processors, the vendor string is GenuineIntel. If CPUID is executed with 1hex in EAX details about the CPU model, family, type, stepping level (revision) and features are returned in EAX and EDX.

The instruction encoding is `0fa2hex`. In 16bit code the upper 16 bits of 32 bit registers can be accessed using the ADRSIZE instruction prefix.

The following code demonstrates how to use CPUID from 16 bit code. It compiles using Microsoft Visual C++ 1.5. For 32bit code, remove the _emit 66h lines and change cx to ecx, ax to eax etc.

```
// First Get Vendor Identification String
// For Intel Pentiums, the string "GeniuneIntel" is
// returned in registers ebx, ecx, edx.
char pszVendor[13];
DWORD dwCPUIDBits;
DWORD dwFeatures;
UINT  nModel;
UINT  nStepping;
UINT  nFamily;
UINT  nType;

__asm
{
    _emit 66h
```

```
    xor ax, ax
    _emit 0fh       ;
    _emit 0a2h      ; CPUID = 0fa2h
    _emit 66h       ; prefix ADRSIZE converts mov from bx to mov from ebx
in 16 bit code
    mov word ptr pszVendor, bx
    _emit 66h       ;
    mov word ptr pszVendor+4, dx
    _emit 66h       ;
    mov word ptr pszVendor+8, cx
    _emit 66h       ;
    xor ax, ax
    mov ax, 1
    _emit 0fh
    _emit 0a2h      ; CPUID
    _emit 66h       ; ADRSIZE
    mov word ptr dwCPUIDBits, ax
    _emit 66h
    mov word ptr dwFeatures, dx
}

pszVendor[12] = 0;
nType = (int)((dwCPUIDBits & 0x00003000) >> 12); // 0 = Normal,
                                                 // 1 = Overdrive
                                                 // 2 = Dual CPU
nFamily = (int)((dwCPUIDBits & 0x00000f00) >> 8);// 5 = Pentium
                                                 // 6 = P6?
nModel = (int)((dwCPUIDBits & 0x000000f0) >> 4);
nStepping = (int)(dwCPUIDBits & 0x0000000f);
```

RDMSR Read Machine Specific Register

In addition to the registers found on the 486, the Pentium includes a number of extra registers (MSRs) which are unique to the Pentium and subject to change in future Intel processors. These registers are all 64bits in width and they are read and written by two new Pentium instructions RDMSR and WRMSR. RDMSR is used to read the registers and is used as follows: RDMSR puts the 64bit value of the MSR selected by the value of ECX into the EDX:EAX registers. Intel documents the values `03h` and `0fh` but the other values are reserved under the terms of the Intel NDA. Some of the other legitimate values of ECX have been determined and published in Byte Magazine [1]. The instructions are privileged and cannot be executed from user mode code. P5MON uses RDMSR and WRMSR to read the CPU profiling counters. Although P5MON.EXE is a standard Windows application, it relies on a VxD, P5MON.386 which performs the actual reading of the counters at a higher privilege level. The instruction encoding is `0f32h`.

WRMSR Write Machine Specific Register

WRMSR writes the contents of registers EDX:EAX to the machine specific register selected by ECX. In common with RDMSR, it is a privileged instruction. The instruction encoding is `0f30h`.

RSM Resume from System Management Mode

System Management Mode is a new operating mode on the Pentium. It is intended for uses such as power management in portables and security functions. SSM is entered due to a hardware signal to the CPU. RSM causes the processor to leave SSM and restore its state.

CPU Profiling Extensions

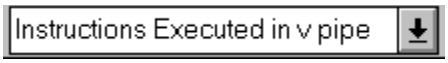
The Pentium includes support for both hardware execution tracing and software profiling. Central to this support is the provision of two profiling counters which can be set to count one of 38 different CPU events.

Profiling Counters

Two of the Machine Specific Registers (MSR 12h & MSR 13h) are dedicated profiling counters whose contents are controlled by MSR 11h. MSRs are accessed using the RDMSR and WRMSR instructions. The least significant 16 bit word of MSR11h controls MSR12h and the next most significant 16bit word controls MSR13h. Each 16bit word breaks into a number of bit fields. Bits 0-5 select which event to select according to the list below. Bit 6 selects enables counting of events while the processor is executing in Rings 1,2 or 3. Bit 7 selects enables counting in Ring 3. If bit 8 is clear, the counter will count events and if it is set it will the count clock cycles the events take. The function of bits 9-15 is unknown. For more information on how to use these counters refer to *Byte Magazine July 1994 [1]*.

Bit 0-5	Event
0	Data Reads
1	Data Writes
2	Data TLB Misses
3	Data Read Misses
4	Data Write Misses
5	Write to E or M lines
6	Data Cache lines written back
7	Data Cache Snoops
8	Data Cache Snoop Hits
9	Memory Accesses
A	Bank Conflicts
B	Misaligned Data Accesses
C	Code Reads
D	Code TLB Misses
E	Code Cache Misses
F	Segment Register Loads10 UNKNOWN
11	UNKNOWN
12	Branches
13	Branch Target Buffer Hits
14	Taken Branches or BTB Hits
15	Pipeline Flushes
16	Instructions Executed
17	nstructions Executed in v pipe
18	Bus Utilization
19	Pipe Stalls - Write Backups
1A	Pipe Stalls - Data Memory Reads
1B	Pipe Stalls - E/M Hits
1C	Locked Bus Cycles
1D	I/O Reads or Writes
1E	Noncacheable Memory References
1F	Address Generation Interlocks
20	UNKNOWN
21	UNKNOWN
22	Floating Point Operations
23	Breakpoint 0 matches
24	Breakpoint 1 matches
25	Breakpoint 2 matches
26	Breakpoint 3 matches

27 Hardware Interrupts
28 Data Reads or Writes
29 Data Cache Misses



Processor Events

There are 38 different types of event that can be monitored.

Data Reads
Data Writes
Data TLB Misses
Data Read Misses
Data Write Misses
Write to E or M lines
Data Cache lines written back
Data Cache Snoops
Data Cache Snoop Hits
Memory Accesses
Bank Conflicts
Misaligned Data Accesses
Code Reads
Code TLB Misses
Code Cache Misses
Segment Register Loads
Branches
Branch Target Buffer Hits
Taken Branches or BTB Hits
Pipeline Flushes
Instructions Executed
Instructions Executed in v pipe
Bus Utilization
Pipe Stalls - Write Backups
Pipe Stalls - Data Memory Reads
Pipe Stalls - E/M Hits
Locked Bus Cycles
I/O Reads or Writes
Noncacheable Memory References
Address Generation Interlocks
Floating Point Operations
Breakpoint 0 matches
Breakpoint 1 matches
Breakpoint 2 matches
Breakpoint 3 matches
Hardware Interrupts
Data Reads or Writes
Data Cache Misses

Data Reads

All data requests are made through the processors Data Cache. All data requests not found in the cache are satisfied by cache line fills unless the addresses are determined to be Noncacheable by the page table entries or by the hardware memory interface. Cache line fills can be can be monitored by setting one of the traces to count Data Read Misses.

Data Writes

The Pentiums Data Cache can be configured to be writethrough or writeback. In software, setting the Page-WriteThrough (PWT) bit in each page table entry can declare regions of memory to be writethough or writeback on a page by page basis. When the Page Cache Disable (PCD) bit of the page table entry is set then all caching is disabled for that page. In hardware, holding the WB/WT# (cache writeback/writethrough select) pin low during a cache line fill causes subsequent writes to that line to be writethrough. Obviously, a writeback cache yields higher performance but it is more difficult to implement. Some of the earlier Pentium motherboards are writethrough only. You can see if this is the case by setting one trace to count Data Writes and the other to count Data Write Misses. If the graphs are equal then the data cache is hardwired to be writethrough.

Data TLB Misses

This counts the number of accesses to data pages that cause a miss in the page table entry lookup system, specifically in the Translation Lookaside Buffer. The penalty is an extra memory access or two to find the correct page table entry (PTE).

Data Read Misses

Data Read Misses are caused by read accesses to data not resident in the Data Cache. Read misses cause a cache line fill if the line is cacheable.

Data Write Misses

A Data Write Miss is caused by:

1. an update to a datum not resident in the Data Cache .
2. an update to a page declared writethrough by setting the PWT bit in its Page Table Entry.
3. an update to a page declared Noncacheable by setting the PCD bit in its page table entry.
4. An update to a cache line declared writethrough by holding the WB/WT# pin low at the time of cache line fill.

Data Write Misses do not cause cache line fills.

Write to E or M lines

This is only relevant in multiprocessor configurations. The processor drives the E(xclusive) or M(odified) pins in response to an inquiry cycle from another processor that hits a cache line that is modified or is about to be modified. In Windows 3.1 or Windows 95 these events never occur. See MESI cache coherency.

Data Cache lines written back

Data Cache lines are written back in response to one of the following events:

1. A Data Read to a cache line that conflicts with a modified line in the cache.
2. When the processor executed the WBINVD (Write Back and Invalidate) instruction.
3. A Snoop Hit to a modified cache line. This is only relevant in multiprocessor configurations.

Data Cache Snoops

A Data Cache Snoop is a request by another processor to check if a cache line is resident in the Data Cache. See MESI cache coherency.
This is only relevant in multiprocessor configurations.

Data Cache Snoop Hits

When a second CPU requests a cache line that hits in the first processors Data Cache this is called a Snoop Hit. See MESI cache coherency.
This is only relevant in multiprocessor configurations.

Memory Accesses

This is the sum of Data Reads, Data Writes and Code Reads.

Bank Conflicts

A Bank conflict occurs when two instructions, one in each pipeline try to access the same bank in the data-cache. It causes a one clock cycle in the V pipe.

Misaligned Data Accesses

A Misaligned Data Access occurs when the processor requests a datum which crosses an even address boundary. Intel documentation states that a misaligned access incurs a three cycle clock penalty.

Unlike code for the 486, aligning code on cache line boundaries has no great effect on system performance.

Code Reads

This is total number of reads to code pages. Note that this is not the same as the number of instructions executed. Neither does the Pentium execute every instruction that it fetches because it will prefetch code cache lines ahead of the current Program Counter value predicting branches where necessary using the Branch Target Buffer.

Code TLB Misses

This counts the number of accesses to code pages that cause a miss in the page table entry lookup system, specifically in the Translation Lookaside Buffer. The penalty is an extra memory access or two to find the correct page table entry (PTE).

Code Cache Misses

Instruction fetches that miss in the Code Cache cause a cache line fill.

Segment Register Loads

The 486 and the Pentium hold the descriptors referred to by the segment registers in hidden registers on chip to minimize memory accesses. Changing the contents of a segment registor causes the corresponding descriptor to be loaded from memory. Because descriptors can be up to 48 bits in length, this can be an expensive operation. Segment register loads should therefore be minmized. Accessing FAR pointers in 16 bit code for Windows 3.1causes a segment register load.

Branches

A branch is an instruction which transfers execution from one contiguous sequence of instructions to another. Branches can be *conditional*, and are taken depending on the outcome of preceding instructions or *unconditional* which are always taken. The Pentium tries to predict the outcome of conditional branches using its Branch Target Buffer. Branch instructions include: Jcc (Conditional Jump) JMP and CALL.

Branch Target Buffer Hits

This counts the number of Branches predicted (correctly or incorrectly) by the Branch Target Buffer.

Taken Branches or BTB Hits

This is a count of Branches predicted correctly by the Branch Target Buffer.

Pipeline Flushes

A Pipeline Flush is a clearing out of the instruction prefetch buffers. They are caused by certain external events and instruction sequences.

Instructions Executed

The Pentium is a *superscalar* microprocessor which means that it can execute more than one instruction at once. It can do this because it has two instruction pipelines. If the instruction sequences are scheduled to take advantage of this feature, taking into account its limitations, it is possible to achieve an throughput greater than one instruction per clock cycle. In practice, however, this is unlikely to be sustained for more than a few milliseconds.

Instructions Executed in V Pipe

The second instruction pipeline can only execute certain instructions. Intel calls these simple instructions. They are characterised by the fact that they are all implemented by hardwired logic and thus do not require microcode to execute. This makes them easy to pipeline. Intel chose these instructions as the ones which are most often executed and which can be implemented easily to spend only one clock cycle in the Integer Pipeline. See Integer Instruction Pairing Rules for a list of simple instructions.

Bus Utilization

This is a count of clock cycles during which the processors external memory bus is in use. Thus if you have a 60Mhz Pentium and measure Bus Utilization measures at 40Million/second the % Bus utilization is 66%.

Pipe Stalls - Write Backups

The Pentium buffers writes to memory to increase instruction execution. This means that it does not wait for the write to complete before starting a new instruction. If, however, an instruction tries to write a datum when the write buffer is full, it must stall in the WB stage of the pipeline while the write buffer completes any pending memory writes.

Pipe Stalls - Data Memory Reads

If an instruction requests a datum which is not in the data cache Cache, it must stall its pipeline while it waits for that data to read from external memory. In contrast, if it is writing a datum that is not in the cache it buffers the write and continues execution.

Pipe Stalls - E/M Hits

This is only relevant in multiprocessor configurations. If the CPU requests a datum which is in the prefetch stage of another CPUs pipeline, the first CPU stall its pipeline while it waits for the data to be read from the second CPUs data cache. It monitors the Exclusive/Modified pins to determine this. See MESI Cache Coherency.

Locked Bus Cycles

Locked bus cycles occur when the instruction being executed is prefixed by the LOCK instruction. The only instructions for which this is relevant are those that perform a Read-Modify-Write operation. The processor will activate the LOCK# signal ensuring that no other entity can read or write the data before the instruction completes. This feature is useful in multiprocessor configurations where it allows software locks and semaphores to be implemented easily.

I/O Reads or Writes

In common with every member of the x86 family, the Pentium has a separate I/O address space accessed by IN and OUT instructions. There are 65536 I/O address locations. Nearly every PC peripheral card and motherboards is controlled using I/O instructions.

Noncacheable Memory References

Regions of memory can be noncacheable for a variety of reasons. If the region represents a memory mapped I/O device, such as video memory it must be noncacheable.
Authors Note: I have not determined if this event includes references to areas of memory declared uncacheable in software by setting the PCD bit in the page table entry or if the cache is diasabled in CR0

Address Generation Interlocks

An Address Generation Interlock (AGI) occurs when a CPU register is used as a base or an index to for an instruction that accesses memory and was modified by a previous instruction. This effect is caused by the nature of the Pentium instruction pipeline. An AGI can be caused by an instruction which modifies the register up to three instructions behind the current one.

The following instruction sequences illustrate where AGIs do and do not occur:

Example 1:

```
add  ecx, 4
mov  eax, [ecx]
```

In this example, register edx is modified one instruction before it is used as an address register

Example 2:

```
mov  bp,ax
mov  ax,[ebx]
mov  dx, 2[bx]
mov  -30[bp],ax
```

Here, use of the bp register causes an AGI despite the fact that the modifying instruction occurs three instructions further back in the instruction stream.

Example 3:

```
add esp, 4
pop ebx
```

Here the stack pointer (esp) is implicitly read and written by the pop instruction.
Push and Pop cause AGIs if they occur after an instruction that modifies register esp. However, the reverse is not true:

Example 4

```
push ebx
mov  eax, esp   // No stall occurs here
```

Floating Point Operations

Floating point instructions are only rarely used by standard Windows applications but are essential for any engineering design or CAD work. With proper FP instruction scheduling Floating Pairing it is possible to attain a sustained throughput comparable to RISC processors. See [7]. Setting one of the profiling counters to count Floating Point Operations does not count all FP instructions, only those which require numeric calculation. Thus FLD, FSTP, FXCH and other FP instructions which do not perform any numeric calculation are not counted. Only FADD, FSUB, FMUL, FDIV, FCOM and other `true` FP instructions are counted.

Breakpoint 0-3 matches

The Pentium has support for for hardware breakpoints, registers that hold linear addresses of points in code where the CPU will trap to a debugger. These can be enabled or disabled. If disabled, the profiling registers can be set to count breakpoint matches without stopping processor execution thus counting the how often a sequence of instructions is executed.

Hardware Interrupts

Hardware interrupts can come from a number of sources:
1) Clock ticks, once every 20 milliseconds in Windows 3.1
2) Mouse, Keyboard. Parallel and serial ports. (see the effect of moving the mouse while monitoring Hardware Interrupts.)
3) Disk Drive reconnects and other I/O involving DMA.

Data Reads or Writes

This is the sum of Data Memory accessed via the Data Cache. If one trace is set to count Data Reads or Writes and the other is set to count Data Cache Misses, the Data Cache hit rate can be estimated.

Data Cache Misses

This is the sum of Data Read Misses and Data Write Misses. Data read misses cause Data Cache line fills but data write misses are directed to memory without updating the cache. If one trace is set to count Data Reads or Writes and the other is set to count Data Cache Misses, the Data Cache hit rate can be estimated.

References:

[1] Terje Mathisen, *Pentium Secrets: Byte Magazine July 1994* pp191-192,

[2] Hennessy and Patterson 1990, *Computer Architecture: A Quantitative approach*, Morgan Kaufman.

[3] Archibald, J and JL Baer 1986, *Cache Coherency Protocols:Evaluation using a multiprocessor simulation model*, ACM Trans. on Computer Systems 4:4 (November) 273-298.

[4] Intel 1994, *Pentium Processor Data Book*, P/N 241428

[5] Intel 1994, *Pentium Processor Family Users Manual Volume 3: Architecture and Programming Manual*, P/N 241430

[6] Intel 1989, *Intel486 Microprocessor Data Book*, P/N 240440

[7] Steve S. Fried *Pentium Optimization and Numeric Performance, Dr Dobbs January 1995*, pp 18-29

[8] Michael Schmit 1995 *Pentium Processor Optimization Tools*, AP Professional.

About P5Mon

The P5Mon application is in two parts. The upper part is the P5MON.EXE Windows application which includes the user interface, toolbar, graph window etc. The lower part is P5MON.386, a Windows Virtual Device Driver (VxD) which performs the actual reading of the profiling counters. A VxD is necessary because the Pentium profiling counters can only be accessed from Ring 0 (Kernel) level code.

Uncertainty Principle - Impact of P5Mon on its own measurements
P5Mon is relatively unintrusive on CPU activity. The core of the program is a loop which processes WM_TIMER messages and on a Gateway P5-90 it takes about 130,000 clock cycles to process each message with the Grid visible and little less if the grid is not shown. For the worst case, testing a 60mhz CPU with a sample interval of 50 mS the load imposed by P5Mon would be about 5%.

Using P5Mon with other Intel Processors
This program is uses Pentium CPU specific registers and will not work on a 386 or a 486. It has not been tested on the Nexgen 586, AMD K5 or Cyrix M1 CPUs but it is highly unlikely that it will work. It will warn the user and prompt her to continue if it detects a processor that is not a Pentium family processor but supports the CPUID instruction and might be a Pentium successor such as the P6. If this is the case and if Intel preserves the same mechanisms for accessing the profiling counters then P5Mon might display meaningful information. Owing to advances in CPU architecture, it is doubtful that all the event selections will map to the same CPU events in future Intel processors. The user proceeds at her own risk.

Windows NT
Because P5MON.EXE depends on a Windows 3.1/Windows 95 specific device driver, it cannot be run on Windows NT. A Windows NT device driver is the subject of future work.

P5MON.386
P5MON.386 is a Windows VxD which allows protected mode programs running under Windows 3.1 and Windows 95 to access the Pentiums profiling counters. If you would like to use P5MON.386 to profile your own Windows application software, email me at *poc@maths.tcd.ie* for the API. There is no fee, but I would like some feedback.

Licensing
P5Mon is Freeware (Smile!). No license is required to use or distribute P5Mon. You must, distribute it unaltered, including the following files: P5MON.EXE, P5MON.386, P5MON.HLP (this file) and README.TXT. You may not sell P5Mon or distribute it for commercial gain. If you find this program useful or if you find any bugs please let me know by email.

poc@maths.tcd.ie
Philip OCarroll
22 April 1995