PENTIUM SUPERSCALAR PROGRAMMING

Daniel Esteban Fernandez

In 1993 Intel announced the Pentium processor. This new release of the 80x86 family has several major changes that makes it really much faster than the 486. Some features, such as a 64-bit bus, a 8K code cache and 8K data cache, and fewer clock cycles for some instructions (especially floating point) are transparent for the programmer, because we're not able to avoid or potentiate them. They are just there, we like it or not.

But there are many others features which are capable to improve, with our help, the processor's perfomance. Basically they are a dynamic branch prediction logic, a pipelined FPU and the aim of this article, a superscalar pipelined programming architecture, i.e. two integer pipelines able to execute some instructions in parallel.

Superscalar means that the CPU can execute two or more instructions per cycle.

Dual Integer Pipelines

Let's make some concepts clearer; superscalar means that the CPU can execute two or more instructions per cycle (being more precise, the Pentium can generate the results of two instructions in a single clock cycle). A pipelined architecture refers to a CPU that executes each portion of an instruction in different stages. After a stage is completed, the following instruction begins executing this stage while the previous instructions moves to the second stage. The Pentium has two five-stage pipelines, named the U pipe and the V pipe. Here is a brief description of what happens during each pipeline stage:

PF: Prefech.
Instructions are fetched from the cache or memory are stored in the prefetch queue.

D1: Instruction.
The instruction is decoded and broken into components parts, opcode and decode operands. If the instruction contains a prefix an extra cycle is required.

D2: Address Generation.
The effective address of the memory operand, if present, is calculated.

EX: Execute and Cache access.
The processor executes the instruction, including reading and storing results.

WB:Write Back .
Instruction is complete and the write buffer is filled with the data to be written in memory.

Most of the time instructions complete each of this stages in one cycle, except the EX stage. It would be very ambicious, and probably imposible, to pretend to execute any pair of instructions (just think what would happen if the second instruction needs the outcome of the first one). For this and others reasons, Intel has made the U pipe fully capable of executing any (integer) instruction, but the V pipe can only execute simple instructions. If two simple instructions are next in the prefech queue and several rules are met, then the CPU pairs them and begins execution of both at the same time. In fact, what we have called 'simple instructions' are the most used ones. Here we have a complete list of each simple instruction and all their formats:

Simple Instructions

```
MOV   reg,reg
MOV   reg,mem
MOV   reg,imm
MOV   mem,reg
MOV   mem,imm
alu   reg,reg
alu   reg,mem
alu   reg,imm
alu   mem,reg
alu   mem,imm
INC   reg
INC   mem
DEC   reg
DEC   mem
PUSH  reg
POP   reg
LEA   reg,mem
JMP   near
CALL  near
Jxx   near
(conditional/inconditional jump code)
NOP
shift reg
shift mem,1
shift reg,imm
shift mem,imm
```

DANIEL ESTEBAN FERNANDEZ *is, at present, developing his PFC at the Universidad Politécnica de Cataluña (UP C).*

Notes:

alu = add, adc, and, or, xor, sub, sbb, cmp, test
shift = sal, sar, shl, shr, rcl, rcr, rol, ror
mem = memory operand
imm = immediate operand
All memory-immediate (mem, imm) instructions are not pairable with a displacement in the memory operand
reg = register (but not a segment register)

Besides the requeriment that both instructions must be simple, there are some other requeriments. We do have to know them, if we want to use all the Pentium's resources:

1. As shown, both instructions must be simple.

2. ADC, SBB and shift/rotate instructions can only execute in the U pipe. This means that the second instruction must can be executed in the V pipe, if we want them to pair (not a JMP/CALL/Jxx, as we'll see in the next rule).

3. JMP/CALL/Jxx can only execute in V pipe, so the second must be a U-pipe executable instruction (not a ADC or SBB or a shift/rotate instructions, as seen, or a prefixed instruction, as shown in 6).

4. Neither instruction can contain both a displacement and an inmediate operand. This is a Pentium design limitation that must be based on the number of instructions components and operand bytes that the Pentium decoders can process in order to determine if two instructions are pairable.

5. Prefixed instructions can only execute in the U pipe. This is important when using segment overrides and when you are writing mixed 16-bit and 32-bit code. The REP/E/NE prefixes cannot be used on any simple instructions. LOCK is allowed, however.

6. The U pipe instructions must be only 1 byte in length or it will not pair until the second time it executes from the cache.This means that the only instructions that will pair on the first execution are INC/DEC reg, PUSH/POP reg and NOP (!) Don't get nervious, because worrying about optimizing code that only executes one time (per cache fill) is of little value.

7. There can be no read-after-write or write-after-write register dependencies between the instruction except for special cases for the flag register and the stack pointer. This is because each 32-bit register is an entity. Therefore, reading/writing in BL involucrates the entire EBX register. The flag register exception allows a CMP or TEST instruction to be paired with a Jxx even thought CMP/TEST writes the flags and Jxx reads the flags. The stack pointer exception allows two PUSHes or two POPs to be paired even thought they both read and write to the SP (or ESP) register. This brings up an optimization: you should use CMP or TEST (if possible) to set the flags, because this instruction only write to the flags register (so it's better TESTAX,AX than OR AX,AX for pairing reasons. CMP is one byte longer). Let's see some examples:

pairing:

write-after-read

```
MOV CX,BX
INC BX
```

read-after-read

```
MOV CX,BX
ADD AX,BX
```

not pairing (must be avoided):

read-after-write

```
MOV BH,1
ADD CL,BL
```

write-after-write

```
MOV EBX,1
ADD BX,AX
```

Branch Prediction Logic

If the execution unit is pipelined, is obvious that the prefecher queue treates branches more ineffectively, because when the branch instruction is reached the processor has to stall and flush the conveyor. This was solved by Pentium's designers implementing a branch prediction logic. It keeps track of the last 256 branches in the Branch Target Buffer (BTB) and tries to predict the destination for each call/jmp. If the prediction is correct, then the number of cycles taken to execute are considerably reduced.

The branch prediction takes place in the second stage, and after predicting whether a branch will be taken or not, a second new prefetch queue (Pentium has two) begins fetching instructions. If the prediction is incorrect both queues are flushed and prefetching is restarted.

Cycle Times Optimizing

You probably think that your old 8086 source code will run automatically faster in your new Pentium. Well, yes and no. Of course it will, but you can surprise yourself if the speed increase is not as much as you expected. Instruction combination or just the use of some instructions instead of other ones that were the fastest in the past may not be as fast as other combinations on the Pentium. Here is a list of the most significants changes of the cycle times from the 386 to Pentium:

Cycle Time Changes

	386	486	Pentium
ADD reg, reg	2	1	1
ADD mem,reg	2	1	1
INC/DEC reg	2	1	1
INC/DEC mem	2	1	1
MOV reg,reg	2	1	1
MOV mem,reg	2	1	1
MUL	9-41	13-42	10-11
NOP	3	3	3
POP/PUSH reg	4/2	1	1
POPA	24	9	5
PUSHA	18	11	5
RET	11	5	2
JMP near	8/9	3/5	1
CALL near	8	3	1
LOOP	13	6/9	7/8
REP MOVS	4	3	1
REP STOS	5	4	1
FADD	23-72	8-32	1-3
FMUL	28-82	11-16	1-3
FCOS, FSIN	23-772	257-354	16-126
FDIV	88-128	73-89	39

Addres Generation Interlock (AGI)

There are conditions that can cause an instruction to take more than one cycle for any stage. Let's see it with an example:

```
MOV ECX,1
LEA BX, MY_ARRAY
MOV DX,[BX]
ADD DX,CX
```

If the processor pairs instructions 2 and 3, it would need the address for the 3rd instruction (which would must be DS*16 + BX), but this address would be incorrect, because the BX register is being updated by the previous instruction. Pentium detects this and generates an AGI (Address Generation Interlock). An AGI is generated if a register is used as a component of an effective address and is also the destination register of an instruction in the previous cycle. This can occur for two reasons: when a register that is updated is used in an effective address calculation in the next cycle (as we have seen) or when an instruction in one cycle changes ESP and the next instruction relies on ESP. However, if both instructions use ESP implicity, such as PUSH or POP, there is no AGI.

Of course, AGIs must be avoided because they generate an extra delay time. Usually, this is not a difficult task, just incorporate another instruction between them (if possible).

Floating-Point Pipeline

We'll finish describing the FPU pipeline on the Pentium. The FPU has eight stages, the first five of which are shared with the integer unit. The other stages are, briefly, after the EX one:

X1: FP Execute stage one. Conversion of FP data to internal data format.

X2: FP Execute stage two.

WF: Perform rounding and write FP results to a register file.

ER: Error reporting. Status word update.

Floating-point instructions cannot be paired with integer instructions, but can be executed in parallel because the integer unit and the FPU are separate; moreover, some FP instructions can be paired together, if the U pipe instruction is a simple FPU intruction and the V pipe instruction is a FXCH. The simple FPU instructions are :

Pentium Pro systems will run existing 16-bit applications slower than a high-end Pentium machine .

Simple FPU instructions.

FABS (absolute value),
ADD (add),
ADDP (add and pop),
ACHS (change sign),
FCOM (compare real),
COMP (compare real and pop),
FDIV (divide),
FDIVP(divide and pop),
FDIVR (reverse divide),
FDIVRP (reverse divide and pop),
FLD (load real, single or double or st(i)),
FMUL (multiply),
FMULP (multiply and pop), FSUB (substract),
FSUBP (substract and pop), FSUBR (reverse substract),
FSUBRP (reverse substract and pop),
FTST (test),
FUCOM (unordered compare real),
FUCOMP (unordered compare real and pop),
FUCOMPP (unordered compare real and pop twice).

Pentium PRO : The Next Generation

With this processor, Intel has implemented a superescalar and super-pipelined design, out-of-order execution, register renaming, advanced branch prediction, and speculative execution.Intel condenses this features in a single marketing term: dinaminc execution, that refers to the processor's ability to optimize program execution by predictiong program flow, choosing the best ordering of instruction execution.

Initially the Pentium Pro CPU, with 5.5 million transistors, will be fabricated on the same 0.6-micron BiCMOS process used to build most Pentiums. In a near future, Intel will move to a 0,35-micron process. The chip contains two chips in a single package: the CPU proper - with 64 bits data bus, 36 bits address bus, 8K data cache and 8K first level code cache (L1) - and either 256K (15.5 millions of transistors) or 512K (31 millions of transistors) of second level (L2) non-blocking high-speed cache memory, all running at clock speeds of 133, 150, 180 and 200 MHz.

The processor's design was optimized from the start for 32-bit applications, and as a result early Pentium Pro systems will run existing 16-bit applications slower than a high-end Pentium machine. With these reasons in the mind, if the Pentium's code optimization was important, now becomes essential. The Pentium Pro achieves high clock speed by virtue of a technique called super-pipelining, that extends the basic pipelining concept that was first introduced. Pipelined microprocessors execute instructions in an «assembly-line» fashion: each instruction takes multiple clock cycles to process in full, but by breaking the processing into multiple stages and beginning to process the next instruction as soon as the previous instruction completes

the first stage, a series of completed instructions can be produced rapidly.

Superpipelining divides the standard pipeline stages further; with more stages, each individual stage does less work and thereby needs less associated hardware logic.

Reducing the complexity of the logic reduces propagation delay, so faster clock speeds are possible. There is also a serious downside: instructions that force the Pentium Pro to flush its pipelines, which include mispredicted branches and operations such as segment-register loads can dramatically reduce perfomance.

Super-Pipeline

We found now 14 stages divided into three sections: the in-order front end, responsible for decoding and issuing instructions, comprises 8 stages. The out-of-order core, which actually executes the instructions, has 3 stages. And the in-order retirement pipeline has 3 stages. All these sections operate somewhat independently. We had seen that the Pentium had two pipelines, but there were severe restrictions on the conditions in which two instructions could be issued simultaneously. The Pentium Pro goes beyond with its three-issue superscalar model.

The process begins when the Pentium Pro reads 64 bytes of code (two cache lines) from its L1 instruction cache as directed by the branch target buffer (BTB). The instruction-fetch unit looks at the current instruction pointer to locate the first x86 instruction and then takes the 16 bytes beginning with that location, aligns them, and passes them to three parallel decoders.

One may think: Why fetch 64 bytes when only 16 are used? Because the instruction cache is organized in 32-byte lines, and a line is the smallest unit of information that the processor can fetch from the cache. If the needed instruction appears near the end of the first cache line, the second cache line provides the remaining bytes necessary to fill the 16-byte buffer without delay.

The decoders proceed with the conversion of the x86 instructions into micro-ops. The Pentium Pro contains three decoders that operate in parallel, two of them «simple» and the other «complex». The simple decoders can handle x86 instructions that translate into a single micro-op. The complex decoder handles instructions that translate into from one to four micro-ops. Some particularly complex instructions cannot be directly decoded even by the complex decoder and are passed off to a microcode instruction sequencer (MIS), which generates as many micro-ops as necessary.

After the instructions are decoded and converted into micro-ops, the seventh pipeline stage sends them to the register alias table (RAT) for register renaming. Register renaming helps mitigate false dependencies that can sap perfomance in an out-of-order execution model. For instance, two instructions may need to write values to the same register; without register renaming, they could not be executed out of order, because the later instruction could not be processed until the earlier instruction was completed.

The Pentium Pro has 40 physical registers; in essence, the processor clones the limited number of real, architectural registers and keeps track of which clones contains the most current values. This prevents delays that false dependencies would otherwise impose because of conflicting demands for a register.

Execution out-of-order

After register renaming is complete, the micro-ops are sent to a structure called the reorder buffer (ROB) and also queued up in a special instruction buffer called a reservation station, which is located between the decode and execute stages. Acting much like a reservoir, the reservation station holds a handful of decoded instructions so that the execution units can keep busy even if the decoders stall. Essentially, the processor is freed from having to execute every instruction in serial order; it can instead evaluate multiple pending micro-ops and determine which are best suited for execution at a given time.

Although this implies that operations aren't completed strictly in the order a programmer intended, this isn't actually the case: out-of-order results are calculated and stored in temporary buffers on the chip and are always written to architectural registers and system memory in program order.

This is were the retirement stages of the pipeline come into the picture. The ROB retains execution status and results of each micro-op. A micro-op is retired and the results are written to architectural registers and memory only when preceding micro-ops are known to have been completed.

We haven't still spoken about a great deal of things Pentium Pro incorporates: a sophisticated branch-prediction, floating-point improvements, speculative execution, etc. But, from a programmer's point of view, this is enough. Let's try to sumarize:

- Avoid reading a large register (such as EAX) after writing a smaller version of the same register (such as AL), or this will cause the processor to stall the issuing of instructions that reference the full register and all subsequent instructions until after the partial write has retired,wich can noticeably reduce perfomance.

-Avoid self-modify code.Code that alters itself can cause the Pentium Pro to flush the processor's pipelines and can invalidate codes resident in caches.

-Avoid branches wherever possible,exploting the Pentium Pro return stack using a RET rather than a JMP at the end of a subroutine.

-Aling data properly.This optimization will benefit any processor.

That's all . From a programmer's perspective , not much has changed from the Pentium to the Pentium Pro . It has no new general-purpose registers and only one significant new instruction:CMOV , or conditional move,allowing a compiler to minimize the number of perfomance sapping branch instructions.

Further Reading

IGOR CHEBOTKO,PETER KALATCHIN,YURI KISELEV & OTHERS:*Master Class Assembly language* , Ed.Wrox,1995

MICHAEL ABRASH:*Zen of Code Optimization*,Ed.AP Profesional,1995

MICHAEL L. SCHMIT:*Pentium Processor Optimization Tools*,Ed.AP Profesional,1995