

The Pentium processors

Contents:

- *The main Pentium functional units*
- *Pentium Registers*
- *Pipelining*
- *Instructions*

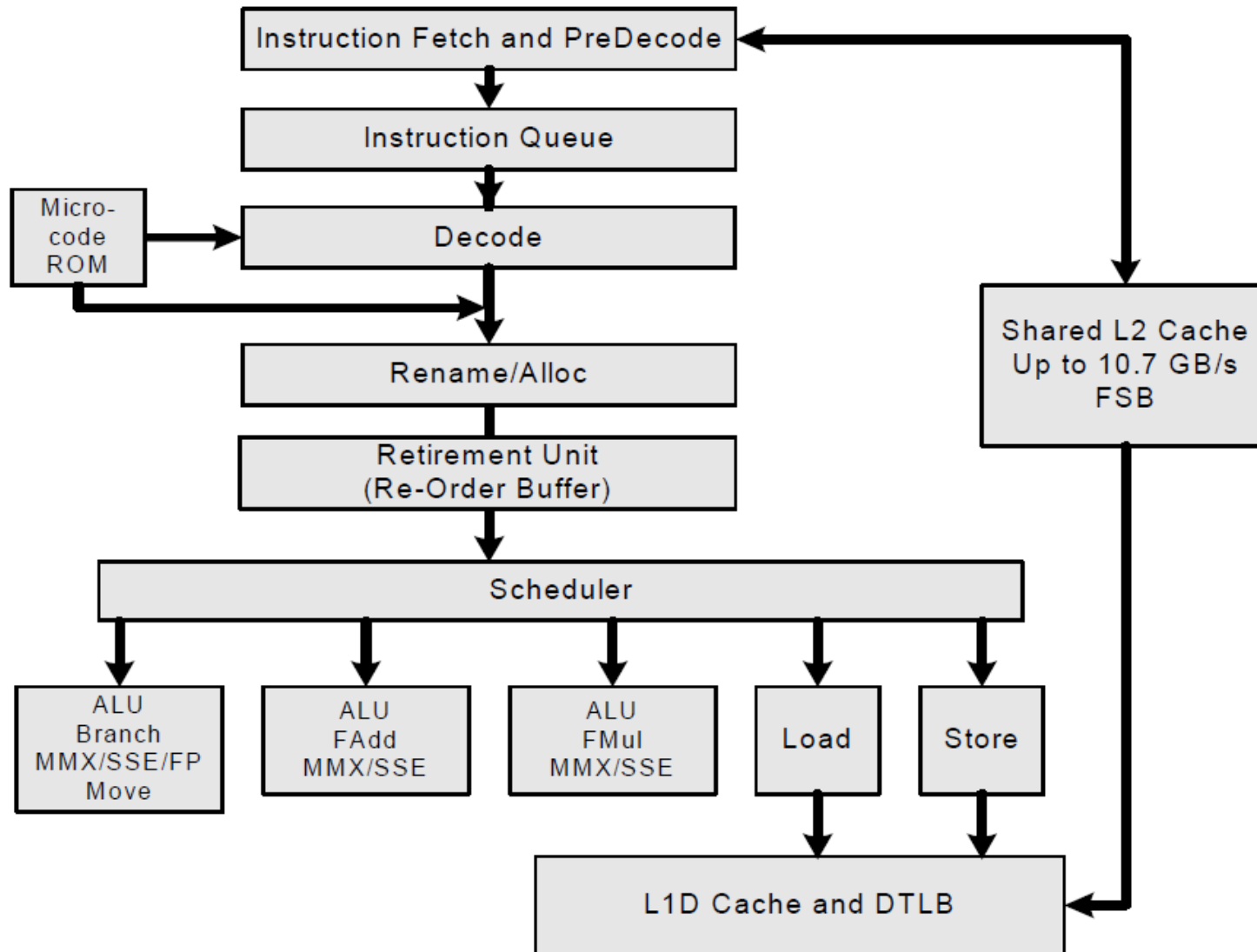
BACKGROUND

- The Pentium family has its origins in the 8086/88 microprocessor that was used by IBM in the original PC of 1980.
- There have been many subsequent generations after the '86: '186, '286, '386, '486, and many versions of the Pentium.
- Each new processor has had to be able to run the software that was written for previous processors; many peculiarities have been inherited.
- Much of this is in Chapter 7 of Williams “Computer System Architecture” .

Execution Core

- The heart of the Intel Core architecture is a complex set of hardware that can work on multiple instructions at the same time.
- Part of this is made possible because each processor core has multiple Arithmetic & Logic Units (ALU). Some of these have specialised uses (e.g. Floating point Add or Multiply).
- An instruction queue can hold multiple instructions to be worked on. Several can be started at the same time: they don't need to finish in the same order.
- To keep the pipeline functioning, there has to be cache on the same chip.

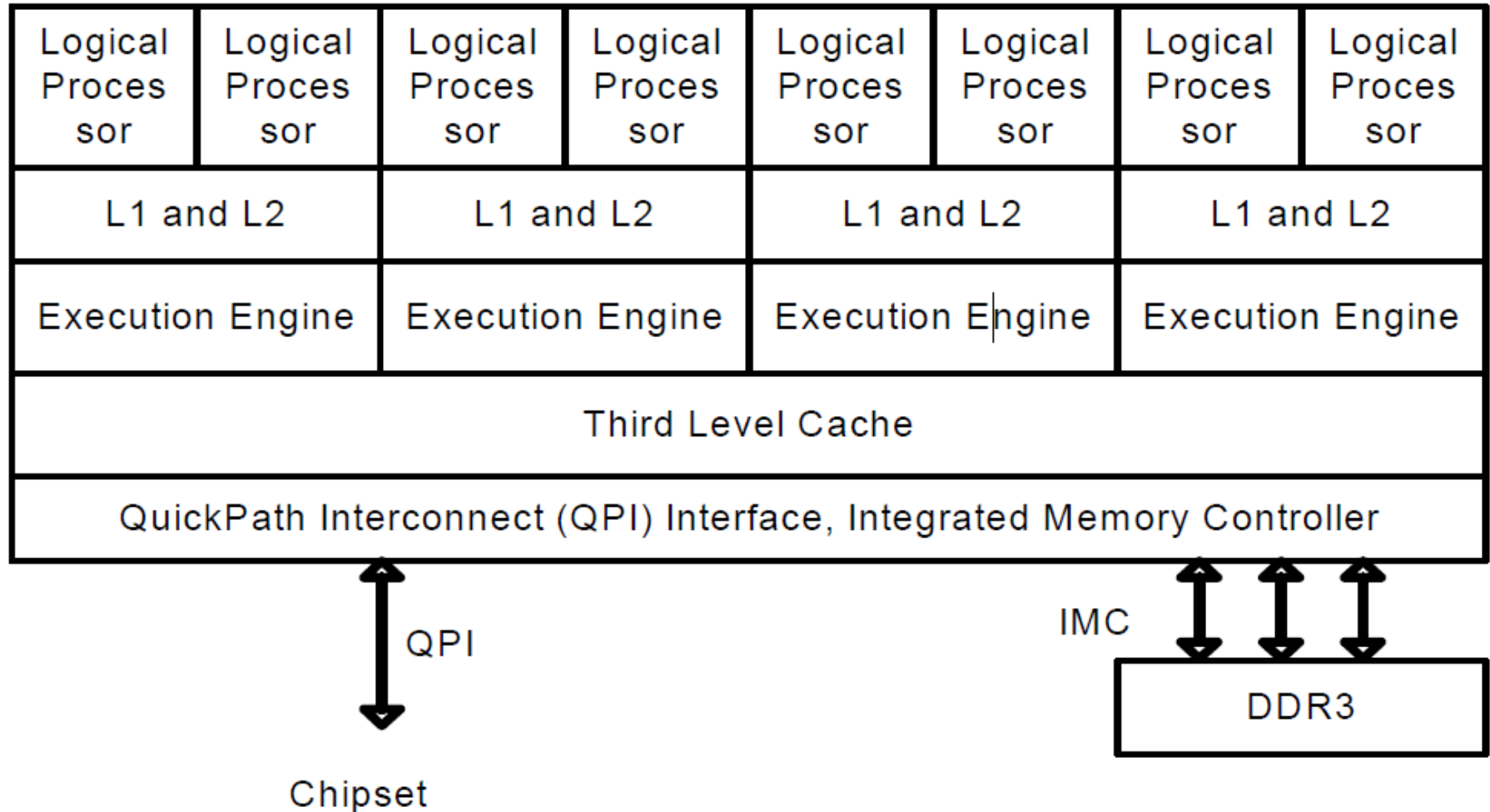
Intel Core pipeline



Multiple processors on one chip

- A high-end processor now has multiple processors on one chip.
- The heart of the processor may well be replicated 2, 4 or more times on a single chip.
- This gives better performance when executing multitasking software. This needs support from the Operating System, but Windows and Linux have had this feature for many years.
- An example is the core i7 processor: quad core, but actually more like 8 'logical processors'.

Intel core i7 processor



Logical processors

- At the heart of the system is the ‘logical processor’. Each one of these is roughly equivalent to an updated version of the original 8086 processor that was in the first PCs.
- The logical processor contains the key features that make up the state of the processor:
 - Registers to hold data while it is being worked on
 - The Program Counter
 - Flags to show the state
- This is the part of the processor that a programmer interacts with, and is the part we will be focussing on next. Intel refers to this as the Basic Execution Environment.

The Registers: Introduction

- This is the bit that the programmer sees. To understand at a fundamental level how to program a microprocessor, it is necessary to become familiar with them.
- In a Complex Instruction Set Computer, such as the Pentium, many have special features:
 - ECX is used as a counter.
 - CS & EIP are used to point to the next instruction.
- The names are often (but not always) abbreviations (EIP Extended Instruction Pointer). When the '386 was introduced, the original 16 bit registers were extended to 32 bits, the extended registers are preceded by 'E'.
- There is some duplication & overlap: AX is just another name for the bottom 16 bits of the EAX register.

Some of the Pentium Registers

EIP

ESP

	IP
	SP

←-----AX-----→

EAX

EBX

ECX

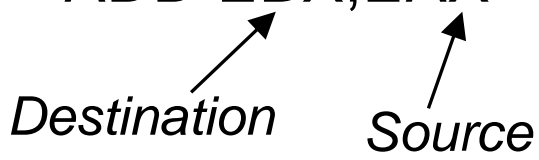
EDX

	AH	AL
	BH	BL
	CH	CL
	DH	DL

EFLAGS

	FLAGS
--	-------

Data Registers

- There are four general purpose registers used to hold data and integers while they are being manipulated.
- They can hold different amounts of data:
 - 8 bits (AL, BL CL, DL, AH, BH, CH, DH) .
 - 16 bits (AX, BX, CX, DX).
 - 32 bits (EAX, EBX, ECX, EDX).
- `MOV AX, 1234H` ; Move (hex) 1234 into AX.
- `ADD EBX,EAX` ;Add the contents of EAX to the contents of EBX & put result in EBX.


Destination *Source*

Pointer Registers

- These are used to hold addresses so that items in memory can be located.
- EIP, the Extended Instruction Pointer, is used to hold the address of the next piece of code to be fetched.
- ESP points to the stack, an area of RAM used as a temporary store.
- DS is used to point to a table that defines the current data segment.

Question

- What would happen if you changed the contents of the instruction pointer?
- The processor would jump to a different bit of program.

Status Word

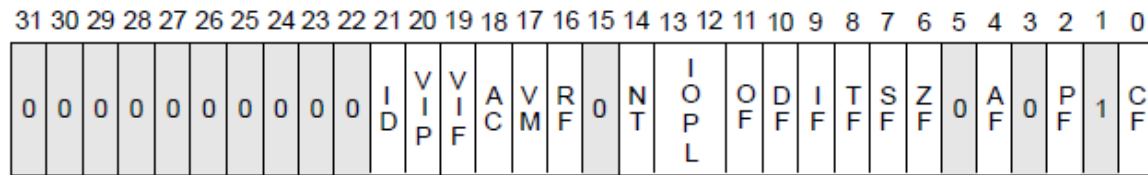
- This is also referred to as the Flags register.
- It contains a collection of individual bits, the status of each bit giving information/control:
 - There is a flag to show if the result of an operation was 0.
 - There is a flag to show if the result was +ve or –ve.

- These are used to control execution:

SUB EAX,EBX ;Subtract two registers.

JZ next_bit ;Jump to 'next_bit' if result is 0

..... ;Otherwise continue at this line.



- X ID Flag (ID) _____
- X Virtual Interrupt Pending (VIP) _____
- X Virtual Interrupt Flag (VIF) _____
- X Alignment Check (AC) _____
- X Virtual-8086 Mode (VM) _____
- X Resume Flag (RF) _____
- X Nested Task (NT) _____
- X I/O Privilege Level (IOPL) _____
- S Overflow Flag (OF) _____
- C Direction Flag (DF) _____
- X Interrupt Enable Flag (IF) _____
- X Trap Flag (TF) _____
- S Sign Flag (SF) _____
- S Zero Flag (ZF) _____
- S Auxiliary Carry Flag (AF) _____
- S Parity Flag (PF) _____
- S Carry Flag (CF) _____

- S Indicates a Status Flag
- C Indicates a Control Flag
- X Indicates a System Flag

Reserved bit positions. DO NOT USE.
Always set to values previously read.

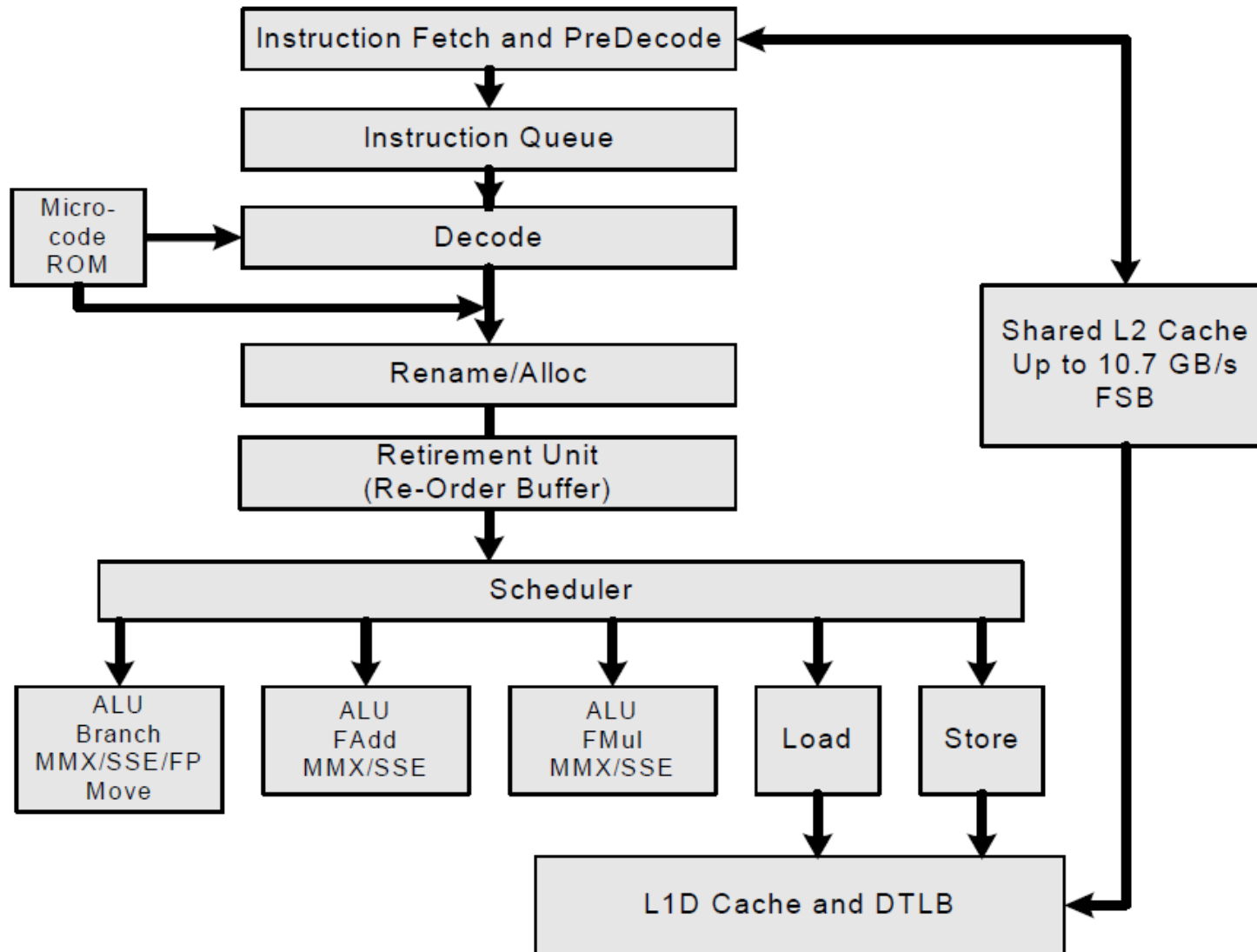
Other Registers

- There are a variety of special purpose registers.
- Some have only one task: CR3 points to the start of page tables.
- There are a group of 64 bit floating point registers; these are used in the manipulation of numbers: 123.456, -0.0034
- In the PII, and later processors, there are also a group of 64 bit registers used in multimedia applications (the MMX registers). These are useful if the same operation has to be done to a number of data items (e.g. all the pixels in an image). (Single Instruction Multiple Data).

Decode Units

- The complex instructions have to be broken into simple operations, e.g.:
- ADD AX, 1234h
- In English: Fetch the number in the AX register, add 1234 to it, store the result back in AX.
- In Pentium Microcode:
 - Fetch instruction into decode unit.
 - Decode it (work out what it will do).
 - Read the Operand (number being operated on).
 - Do the arithmetic (execute the instruction).
 - Store the result in the Destination.

Intel Core pipeline



Pipelining

- The processor is driven by a clock that controls the internal logic; typically about 3 GHz on a new PC (3,000,000,000 ticks per second).
- One instruction is too complex to do in a single cycle, but they can be organised into a pipeline.
- In normal circumstances, an instruction finishes every clock, even though each instruction has taken 5 cycles.
- Needs Parallel hardware, so that one bit can be fetching an instruction, while another is decoding the previous one, and so on.

Pipeline in operation



	Fetch	Decode	Read Op	Exec	Store
Cycle 1	Mov				
Cycle 2	Add	Mov			
Cycle 3	Inc	Add	Mov		
Cycle 4	Mul	Inc	Add	Mov	
Cycle 5	Cmp	Mul	Inc	Add	Mov
Cycle 6	Mov	Cmp	Mul	Inc	Add
Cycle 7	Sub	Mov	Cmp	Mul	Inc

Microarchitecture

Pipeline stages

[P5](#) (Pentium)

5

[P6](#) (Pentium Pro)

14

P6 (Pentium 3)

10

[NetBurst](#) (Willamette)

20

NetBurst (Northwood)

20

NetBurst (Prescott)

31

NetBurst (Cedar Mill)

31

[Core](#)

14

[Bonnell](#)

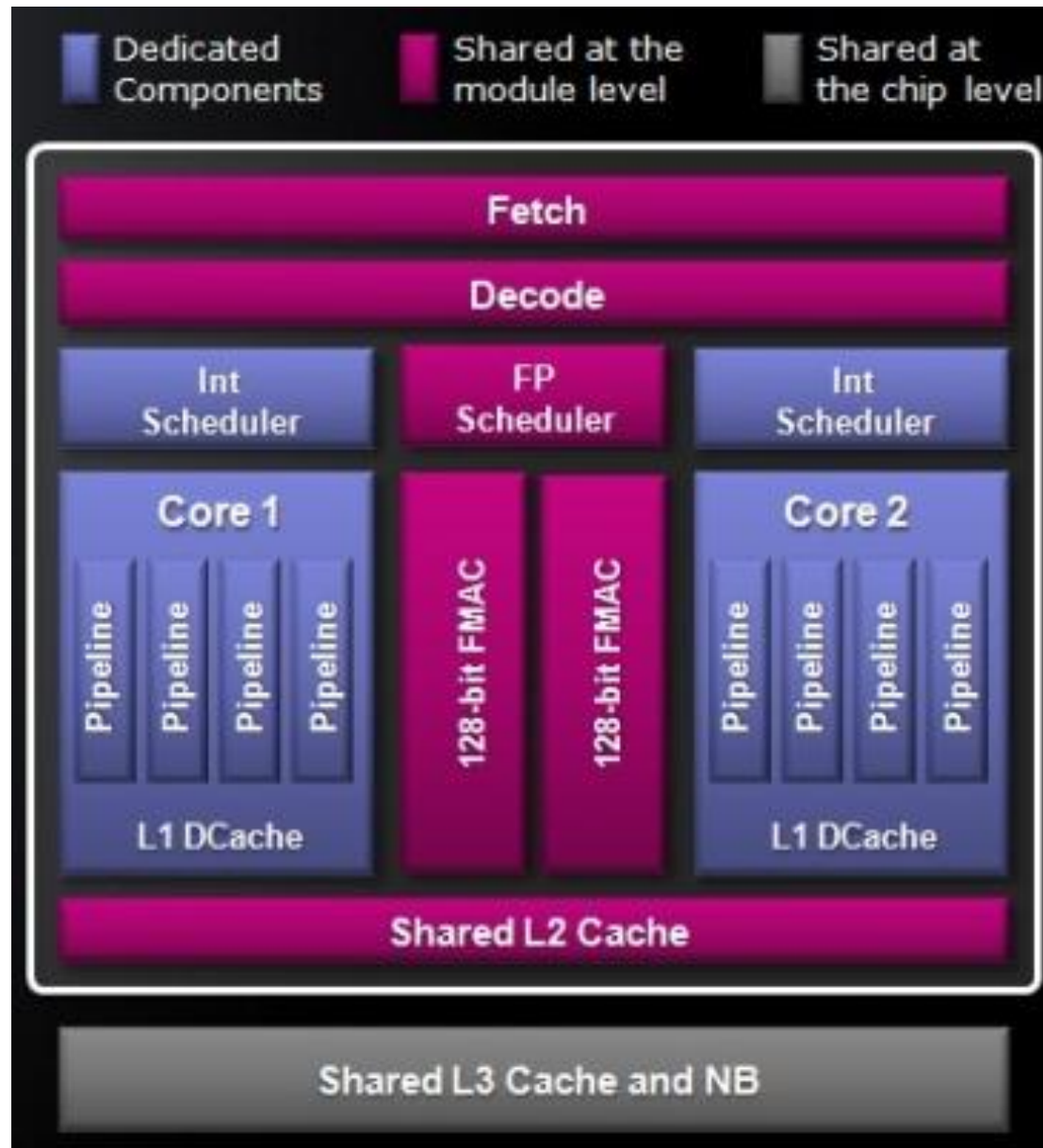
16

Pipeline Problems

- When code is executed in normal order, the pipeline speeds up the process. But, there are problems with branches:

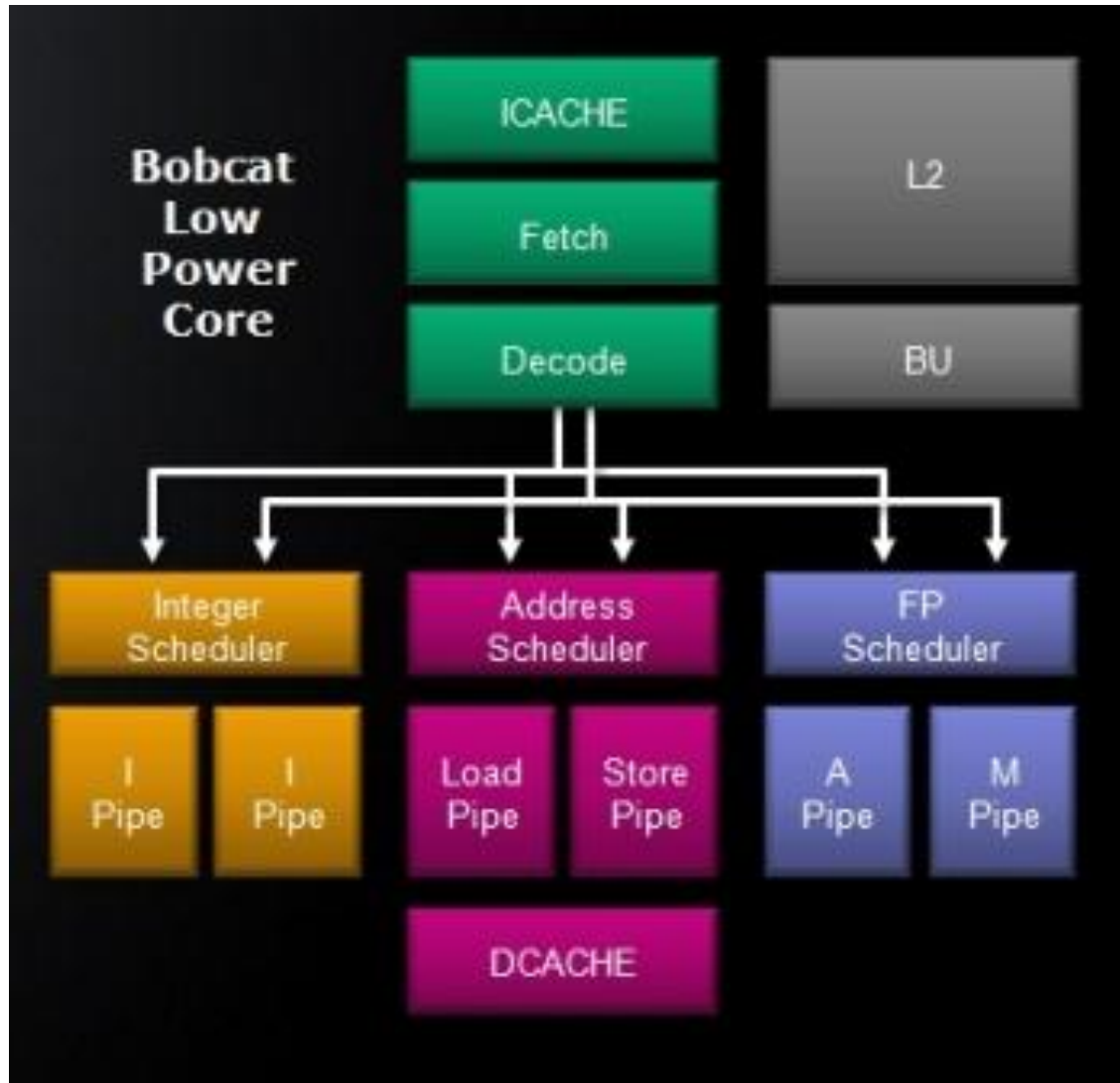
```
CMP EAX, EBX  
JE Next_bit
```

- Depending on the numbers in EAX and EBX, execution after these two instructions could go one of two ways. Until the instruction is executed, it is impossible to say which way. The pipeline may contain half decoded instructions that are not needed.



AMD Bulldozer

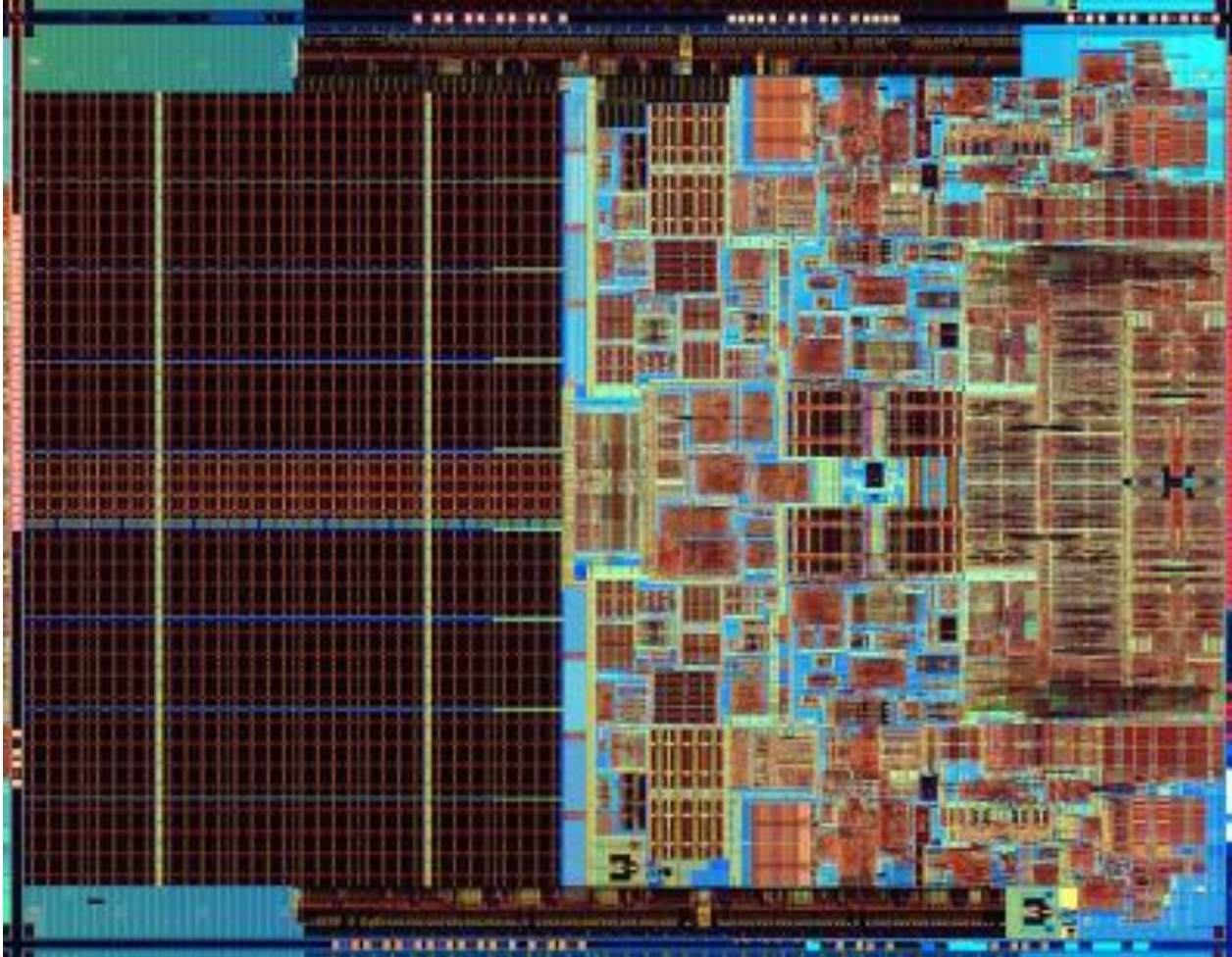
Source: EE Times
23/8/2010



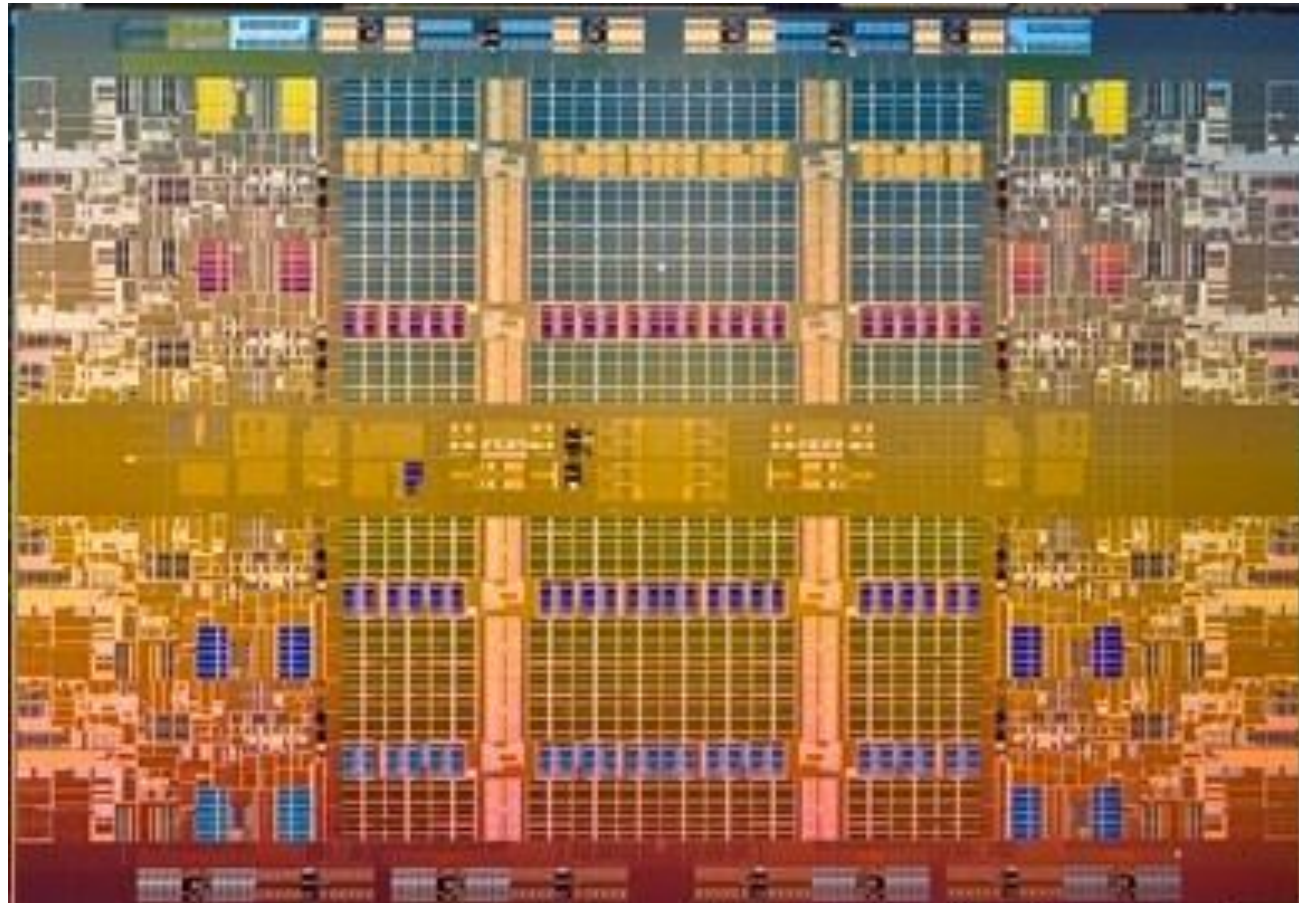
AMD Bobcat

Source: EE Times
23/8/2010

Core 2 duo chip



Intel Xeon 7500 Eight-core



The Structure of Instructions



- I've used Assembler Mnemonics (ADD AX, BX). These are relatively easy to understand.
- The computer actually stores, and uses, binary numbers (machine code). The instruction ADD AX, BX is actually 03C3 in hexadecimal.
- 0 0 0 0 0 0, 1, 1, 1 1, 0 0 0, 0 1 1
 - 000000 Is the code for an ADD instruction.
 - 1 to use a register as the destination.
 - 1 to use a word register (i.e. a 16 bit register).
 - 11 to get both starting numbers from registers.
 - 000 & 011 are the codes for the AX and BX registers.

Another Example

MOV EAX, 12345678h

- Move the hex number 12345678 into EAX.
- This instruction would be 48 bits long, too long to write in binary. In hex it appears as:

66 B8 78 56 34 12

- 66 is a prefix that means use 32 bit numbers.
- B8 = 10111000; 10111 for MOV, 000 for EAX.
- Note that the number itself appears in the instruction, but lowest byte first. This is called 'Little Endian'; other systems (Motorola, Sun) use Big Endian, a source of confusion.

Addressing Modes

1. MOV AX, 1234h, is an example of Immediate addressing: the number is part of the instruction.
2. Transfer between registers (direct): MOV AL,BL
3. Moving data directly to and from memory:
 - MOV AX, [1000] ;Move into AX the data in RAM at location 1000.
 - MOV Total, BH ;Move the contents of BH into the variable Total.

The Pentium has many other modes used to access data structures, such as arrays. These often use a register as an index: MOV EAX, table[ESI]

Questions

- What happens to the original numbers in an instruction such as **ADD EAX, EBX** ?
- EAX will be overwritten, EBX won't change
 $A = A + B.$
- What would the instruction **ADD EAX, [2000]** mean?
- Go to memory location 2000 (the 'source').
- Fetch the contents and add this number to the contents of EAX.
- Store the result in EAX.

High Level Languages

- Fortunately, High Level Languages (C++, Java..) shield us from the difficulties of low level code. The compiler translates each HLL instruction into appropriate Assembler/Machine Code.

NewTotal = OldTotal + Sum;

MOV EAX, Oldtotal

MOV EBX, Sum

ADD EAX, EBX

MOV NewTotal, EAX

64 Bit processing

- AMD was first to add 64 bit support to their processors. The AMD64 processors enable larger amounts of data to be manipulated at any one time, and also larger amounts of memory to be addressed.
- Intel followed with their EM64T. This is compatible with the AMD system.
- Both systems only work properly if the operating system supports it: Linux and more recent versions of Windows. Any compiler used also has to support 64 bit processing.
- AX = 16 bits, EAX = 32 bits, RAX = 64 bits.
- The 64 bit Instruction pointer, RIP, gives access in principle to programs of 2^{64} bytes (roughly 16 million Terabytes).

Assembler and Security/forensics

- One area where assembler is still used is in security and forensics.
- When analysing a virus or rootkit, the source code isn't available. It may well have been written in C++, but all the analyst has available is the actual code of the virus (perhaps running on a machine or in an .EXE file).
- Tools like OllyDbg can capture running code or code on a disk.
- The display will show the assembler mnemonics.

OllyDbg - Snippy.exe

File View Debug Plugins Options Window Help

File View Debug Plugins Options Window Help

CPU - main thread, module Snippy

```

00401000 56 PUSH ESI
00401001 8BF1 MOV ESI,ECX
00401003 8B4E 04 MOV ECX,DWORD PTR DS:[ESI+4]
00401006 85C9 TEST ECX,ECX
00401008 74 0F JE SHORT Snippy.00401019
0040100A 8B01 MOV EAX,DWORD PTR DS:[ECX]
0040100C 8B10 MOV EDX,DWORD PTR DS:[EAX]
0040100E 6A 01 PUSH 1
00401010 FFD2 CALL EDX
00401012 C746 04 000000 MOV DWORD PTR DS:[ESI+4],0
00401019 8B06 MOV EAX,DWORD PTR DS:[ESI]
0040101B 85C0 TEST EAX,EAX
0040101D 74 0D JE SHORT Snippy.0040102C
0040101F 50 PUSH EAX
00401020 FF15 24204100 CALL DWORD PTR DS:[<&GDI32.DeleteObject
00401026 C706 00000000 MOV DWORD PTR DS:[ESI],0
0040102C 5E POP ESI
0040102D C3 RETN
0040102E CC INT3
0040102F CC INT3
00401030 53 PUSH EBX
00401031 55 PUSH EBP
00401032 56 PUSH ESI
00401033 57 PUSH EDI
00401034 8BF1 MOV ESI,ECX
00401036 E8 C5FFFFFF CALL Snippy.00401000
00401038 8B6C24 14 MOV EBP,DWORD PTR SS:[ESP+14]
0040103F 55 PUSH EBP
00401040 FF15 60224100 CALL DWORD PTR DS:[<&USER32.GetDC>]
00401046 8B5C24 18 MOV EBX,DWORD PTR SS:[ESP+18]

```

0040BA2F=Snippy.0040BA2F

Registers (FPU)

EAX	00000000
ECX	0012FFB0
EDX	7C90E514 ntdll.KiFastSystemCallRet
EBX	7FFDA000
ESP	0012FFC4
EBP	0012FFFF
ESI	FFFFFFFF
EDI	7C910228 ntdll.7C910228
EIP	0040597A Snippy.<ModuleEntryPoint>
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 1	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FFDF000(FFF)
T 0	GS 0000 NULL
D 0	
O 0	LastErr ERROR_NO_IMPERSONATION_TOKEN (0000051D)
EFL	00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0	empty -UNORM BDEC 01050104 00000000
ST1	empty 0.0
ST2	empty 0.0
ST3	empty 0.0
ST4	empty 0.0
ST5	empty 0.0
ST6	empty 1.000000000000000000000000
ST7	empty 1.000000000000000000000000
FST	4020 Cond 1 0 0 0 Err 0 0 1 0 0 0 0 (EQ)
FCW	027F Prec NEAR,53 Mask 1 1 1 1 1 1

Address	Hex dump	ASCII
004160C8	00 00 00 00 2E 3F 41 56?AU
004160D0	4D 75 6C 74 69 4D 6F 6E	MultiMon
004160D8	69 74 6F 72 44 69 73 70	itorDisp
004160E0	6C 61 79 40 40 00 00 00	lay@....
004160E8	F4 27 41 00 00 00 00 00	?'A.....
004160F0	2E 3F 41 56 43 61 63 68	.?AUCach
004160F8	65 64 42 69 74 6D 61 70	edBitnap
00416100	40 47 64 69 70 6C 75 73	@Gdiplus
00416108	40 40 00 00 F4 27 41 00	@@..?'A.
00416110	00 00 00 00 2E 3F 41 56?AU
00416118	53 65 6C 65 63 74 69 6F	Selectio
00416120	6E 42 61 73 65 40 40 00	nBase@.
00416128	F4 27 41 00 00 00 00 00	?'A.....
00416130	2E 3F 41 56 46 72 65 65	.?AUFree
00416138	68 61 6E 64 53 65 6C 65	handSele
00416140	63 74 69 6F 6E 40 40 00	ction@.
00416148	F4 27 41 00 00 00 00 00	?'A.....
00416150	2F 3F 41 56 52 65 63 74	?AUReot

Address	Hex dump	ASCII
0012FFC4	7C817077	RETURN to kernel32.7C817077
0012FFC8	7C910228	ntdll.7C910228
0012FFCC	FFFFFFFF	
0012FFD0	7FFDA000	
0012FFD4	8054B6ED	
0012FFD8	0012FFC8	
0012FFDC	899D6190	
0012FFE0	FFFFFFFF	
0012FFE4	7C839AD8	End of SEH chain
0012FFE8	7C817080	SE handler
0012FFEC	00000000	kernel32.7C817080
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	0040597A	Snippy.<ModuleEntryPoint>
0012FFFC	00000000	

Analysing Snippy: 349 heuristical procedures, 310 calls to known, 56 calls to guessed functions

Paused