AGH

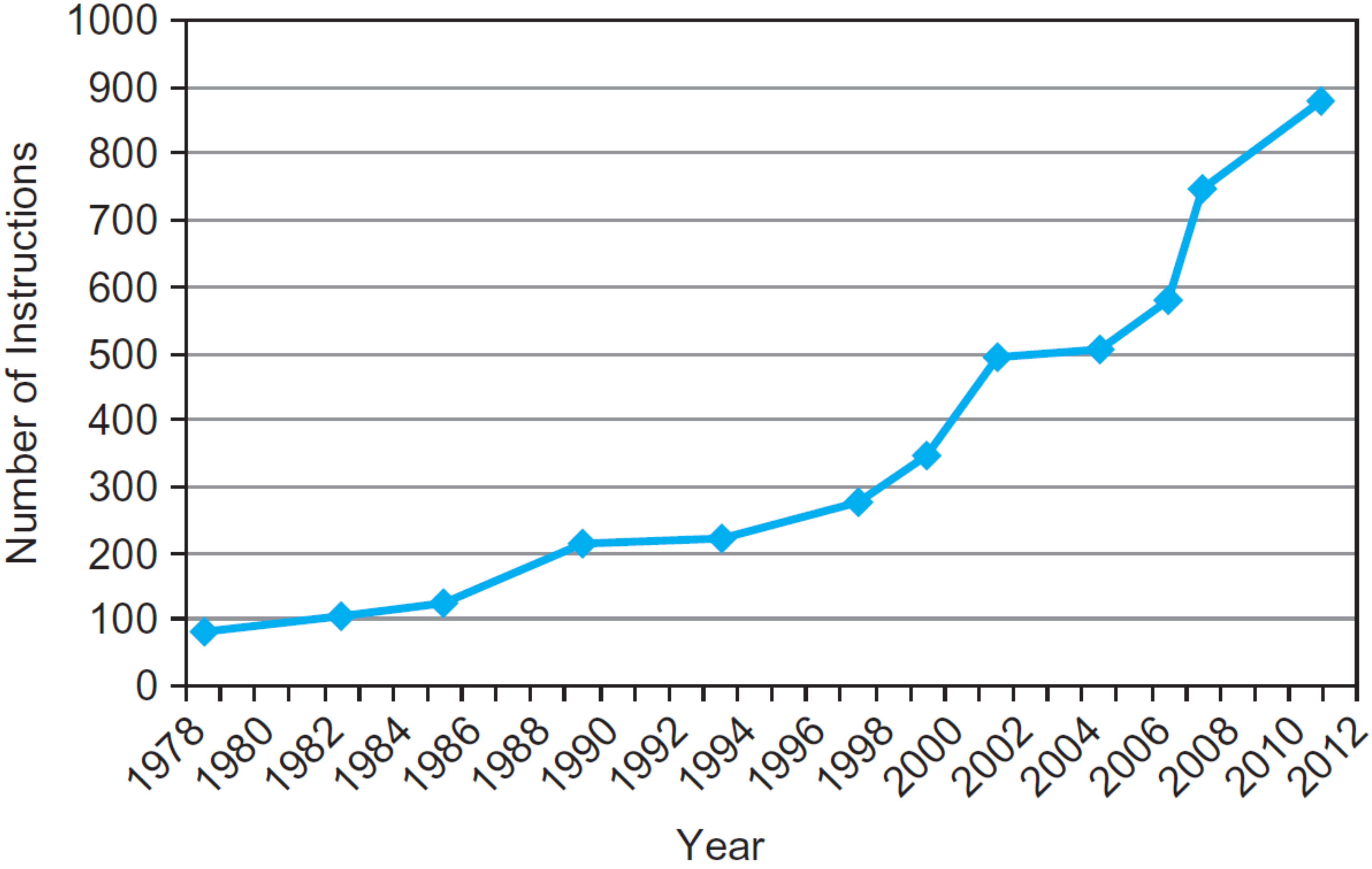
AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Procesory i Architektura Systemów Komputerowych

Historia rozwoju

IET
Katedra Elektroniki
Kraków 2015
dr inż. Roman Rumian

Wzrost liczby instrukcji x86 w czasie



■ **1978**: The Intel 8086 architecture was announced as an assembly language–compatible extension of the then successful Intel 8080, an 8-bit microprocessor. The 8086 is a 16-bit architecture, with all internal registers 16 bits wide. Unlike MIPS, the registers have dedicated uses, and hence the 8086 is not considered a **general-purpose register** architecture.

■ **1980**: The Intel 8087 floating-point coprocessor is announced. This architecture extends the 8086 with about 60 floating-point instructions. Instead of using registers, it relies on a stack (see **Section 2.21** and Section 3.7).

■ **1982**: The 80286 extended the 8086 architecture by increasing the address space to 24 bits, by creating an elaborate memory-mapping and protection model (see Chapter 5), and by adding a few instructions to round out the instruction set and to manipulate the protection model.

■ **1985**: The 80386 extended the 80286 architecture to 32 bits. In addition to a 32-bit architecture with 32-bit registers and a 32-bit address space, the 80386 added new addressing modes and additional operations. The added instructions make the 80386 nearly a general-purpose register machine. The 80386 also added paging support in addition to segmented addressing (see Chapter 5). Like the 80286, the 80386 has a mode to execute 8086 programs without change.

■ **1989–95**: The subsequent 80486 in 1989, Pentium in 1992, and Pentium Pro in 1995 were aimed at higher performance, with only four instructions added to the user-visible instruction set: three to help with multiprocessing (Chapter 6) and a conditional move instruction.

■ **1997**: After the Pentium and Pentium Pro were shipping, Intel announced that it would expand the Pentium and the Pentium Pro architectures with MMX (Multi Media Extensions). This new set of 57 instructions uses the floating point stack to accelerate multimedia and communication applications. MMX instructions typically operate on multiple short data elements at a time, in the tradition of *single instruction, multiple data* (SIMD) architectures (see Chapter 6). Pentium II did not introduce any new instructions.

■ **1999**: Intel added another 70 instructions, labeled SSE (*Streaming SIMD Extensions*) as part of Pentium III. The primary changes were to add eight separate registers, double their width to 128 bits, and add a single precision floating-point data type. Hence, four 32-bit floating-point operations can be performed in parallel. To improve memory performance, SSE includes cache prefetch instructions plus streaming store instructions that bypass the caches and write directly to memory.

■ **2001**: Intel added yet another 144 instructions, this time labeled SSE2. The new data type is double precision arithmetic, which allows pairs of 64-bit floating-point operations in parallel. Almost all of these 144 instructions are versions of existing MMX and SSE instructions that operate on 64 bits of data in parallel.

in parallel. Not only does this change enable more multimedia operations; it gives the compiler a different target for floating-point operations than the unique stack architecture. Compilers can choose to use the eight SSE registers as floating-point registers like those found in other computers. This change boosted the floating-point performance of the Pentium 4, the first microprocessor to include SSE2 instructions.

■ 2003: A company other than Intel enhanced the x86 architecture this time. AMD announced a set of architectural extensions to increase the address space from 32 to 64 bits. Similar to the transition from a 16- to 32-bit address space in 1985 with the 80386, AMD64 widens all registers to 64 bits. It also increases the number of registers to 16 and increases the number of 128- bit SSE registers to 16. The primary ISA change comes from adding a new mode called long mode that redefines the execution of all x86 instructions with 64-bit addresses and data. To address the larger number of registers, it adds a new prefix to instructions. Depending how you count, long mode also adds four to ten new instructions and drops 27 old ones. PC-relative data addressing is another extension. AMD64 still has a mode that is identical to x86 (legacy mode) plus a mode that restricts user programs to x86 but allows operating systems to use AMD64 (compatibility mode). These modes allow a more graceful transition to 64-bit addressing than the HP/Intel IA-64 architecture.

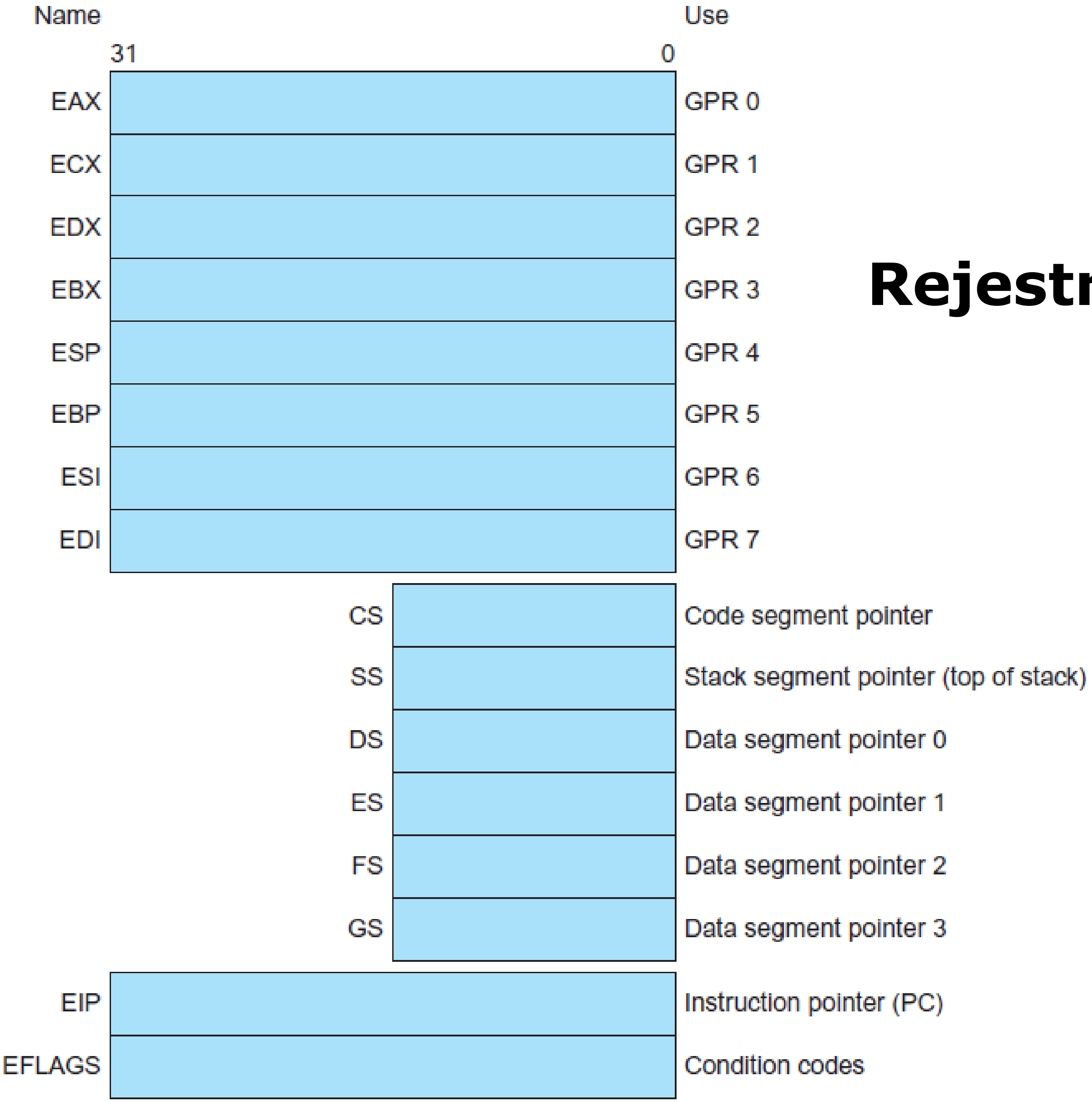
■ 2004: Intel capitulates and embraces AMD64, relabeling it Extended Memory 64 Technology (EM64T). The major difference is that Intel added a 128-bit atomic compare and swap instruction, which probably should have been included in AMD64. At the same time, Intel announced another generation of media extensions. SSE3 adds 13 instructions to support complex arithmetic, graphics operations on arrays of structures, video encoding, floating-point conversion, and thread synchronization (see Section 2.11). AMD added SSE3 in subsequent chips and the missing atomic swap instruction to AMD64 to maintain binary compatibility with Intel.

■ 2006: Intel announces 54 new instructions as part of the SSE4 instruction set extensions. These extensions perform tweaks like sum of absolute differences, dot products for arrays of structures, sign or zero extension of narrow data to wider sizes, population count, and so on. They also added support for virtual machines (see Chapter 5).

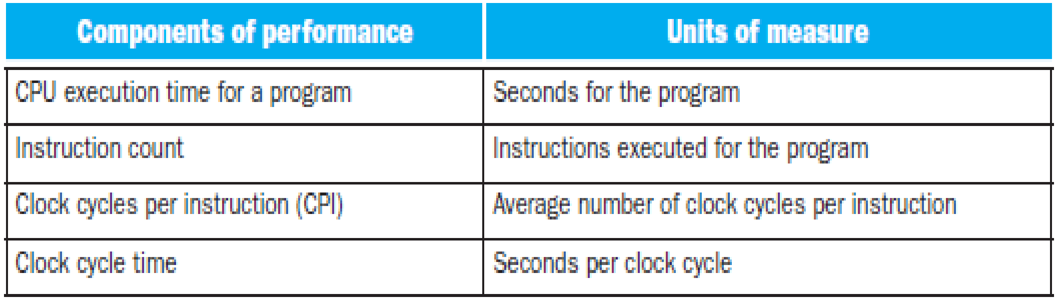
■ 2007: AMD announces 170 instructions as part of SSE5, including 46 instructions of the base instruction set that adds three operand instructions like MIPS.

■ 2011: Intel ships the Advanced Vector Extension that expands the SSE register width from 128 to 256 bits, thereby redefining about 250 instructions and adding 128 new instructions.

Rejestry 80386



Name	Use
EAX	GPR 0
ECX	GPR 1
EDX	GPR 2
EBX	GPR 3
ESP	GPR 4
EBP	GPR 5
ESI	GPR 6
EDI	GPR 7
CS	Code segment pointer
SS	Stack segment pointer (top of stack)
DS	Data segment pointer 0
ES	Data segment pointer 1
FS	Data segment pointer 2
GS	Data segment pointer 3
EIP	Instruction pointer (PC)
EFLAGS	Condition codes

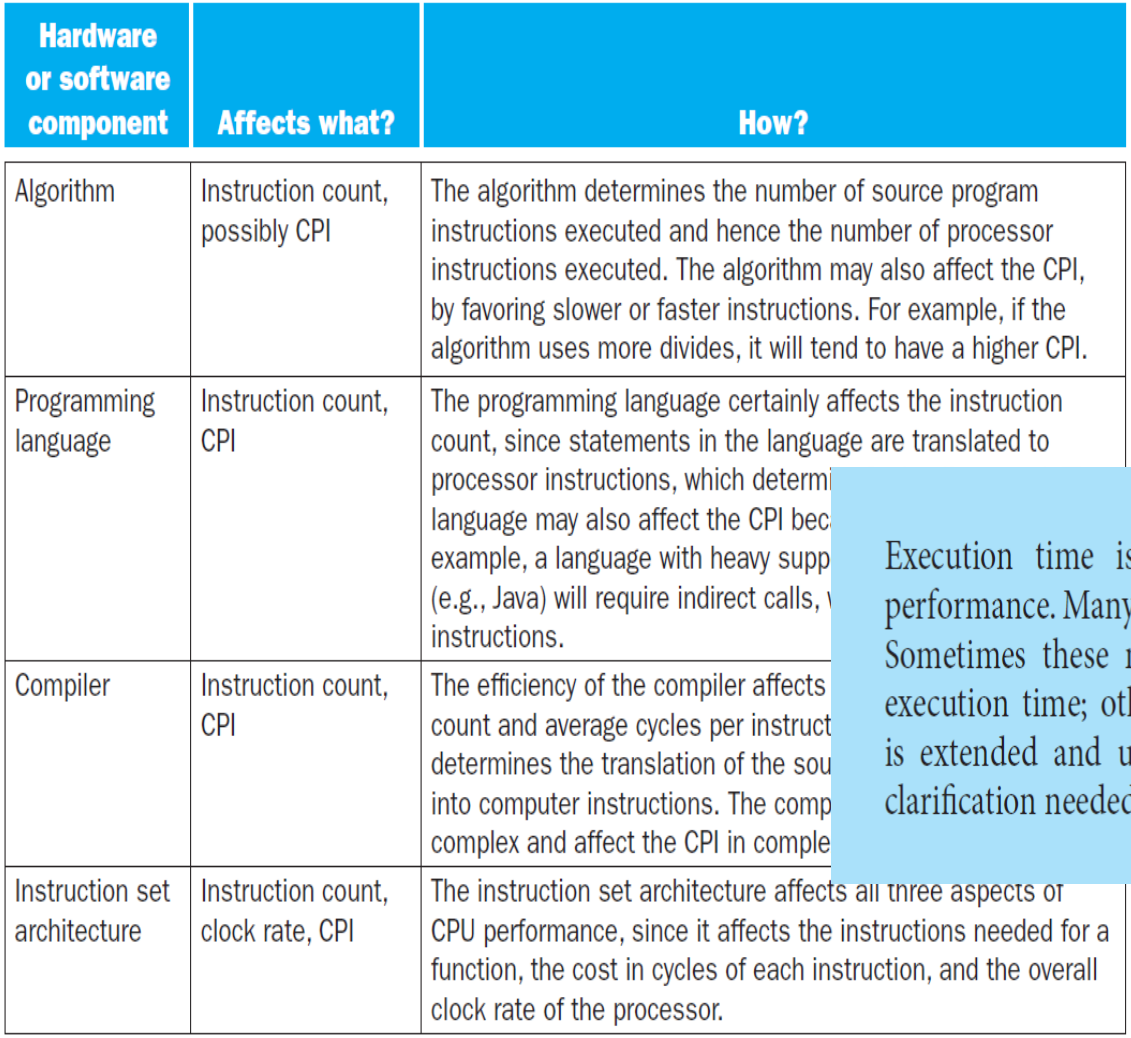


Components of performance	Units of measure
CPU execution time for a program	Seconds for the program
Instruction count	Instructions executed for the program
Clock cycles per instruction (CPI)	Average number of clock cycles per instruction
Clock cycle time	Seconds per clock cycle

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

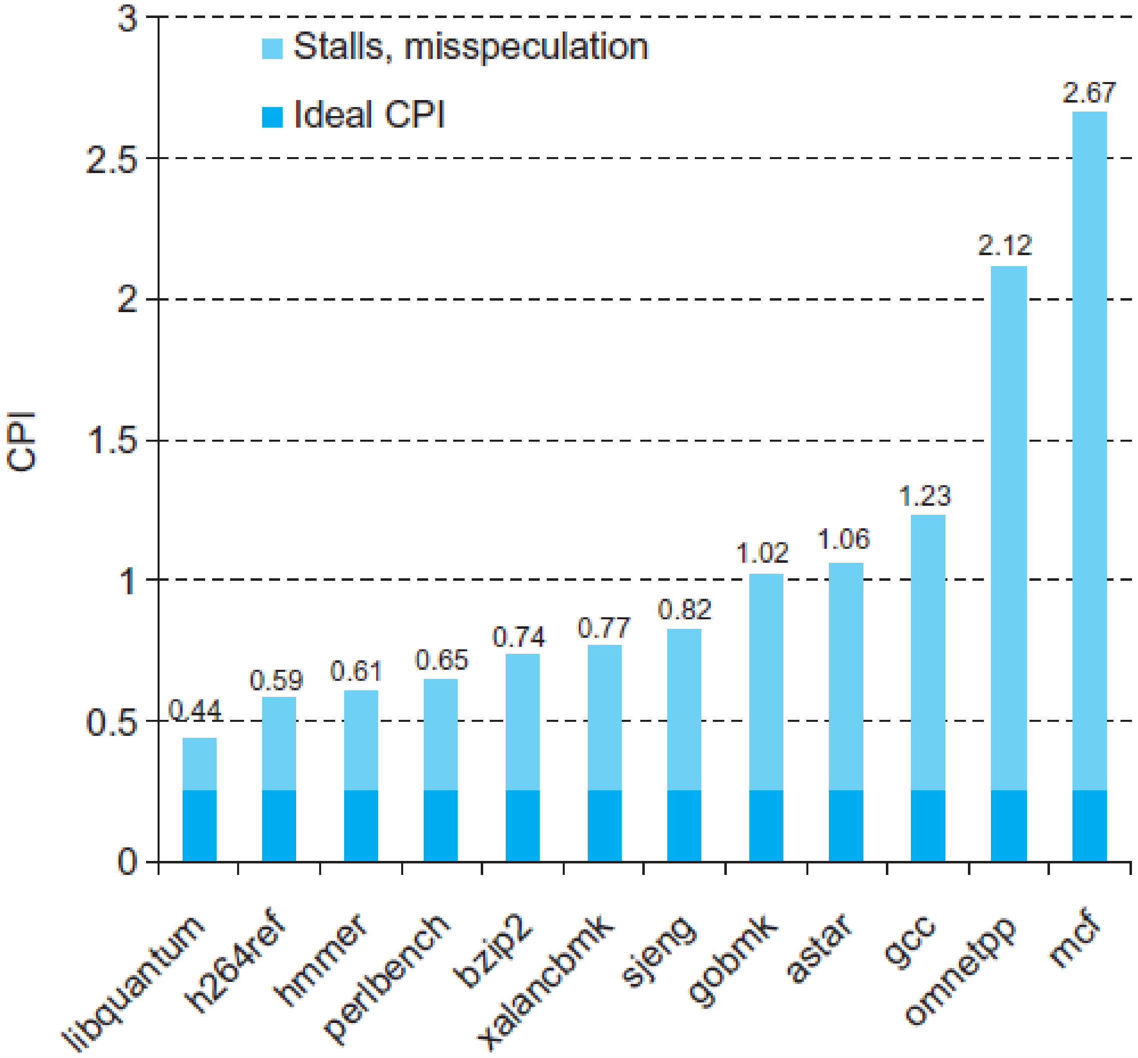
$$\text{MIPS} = \frac{\text{Instruction count}}{\dfrac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$



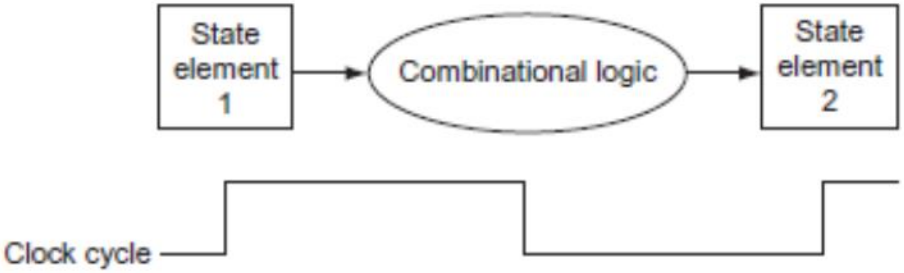
Hardware or software component	Affects what?	How?
Algorithm	Instruction count, possibly CPI	The algorithm determines the number of source program instructions executed and hence the number of processor instructions executed. The algorithm may also affect the CPI, by favoring slower or faster instructions. For example, if the algorithm uses more divides, it will tend to have a higher CPI.
Programming language	Instruction count, CPI	The programming language certainly affects the instruction count, since statements in the language are translated to processor instructions, which determi language may also affect the CPI bec example, a language with heavy supp (e.g., Java) will require indirect calls, instructions.
Compiler	Instruction count, CPI	The efficiency of the compiler affects count and average cycles per instruct determines the translation of the sou into computer instructions. The comp complex and affect the CPI in comple
Instruction set architecture	Instruction count, clock rate, CPI	The instruction set architecture affects all three aspects of CPU performance, since it affects the instructions needed for a function, the cost in cycles of each instruction, and the overall clock rate of the processor.

Execution time is the only valid and unimpeachable measure of performance. Many other metrics have been proposed and found wanting. Sometimes these metrics are flawed from the start by not reflecting execution time; other times a metric that is valid in a limited context is extended and used beyond that context or without the additional clarification needed to make it valid.

CPI of Intel Core i7 920 running SPEC2006 integer benchmarks.



What for is the processor clock?



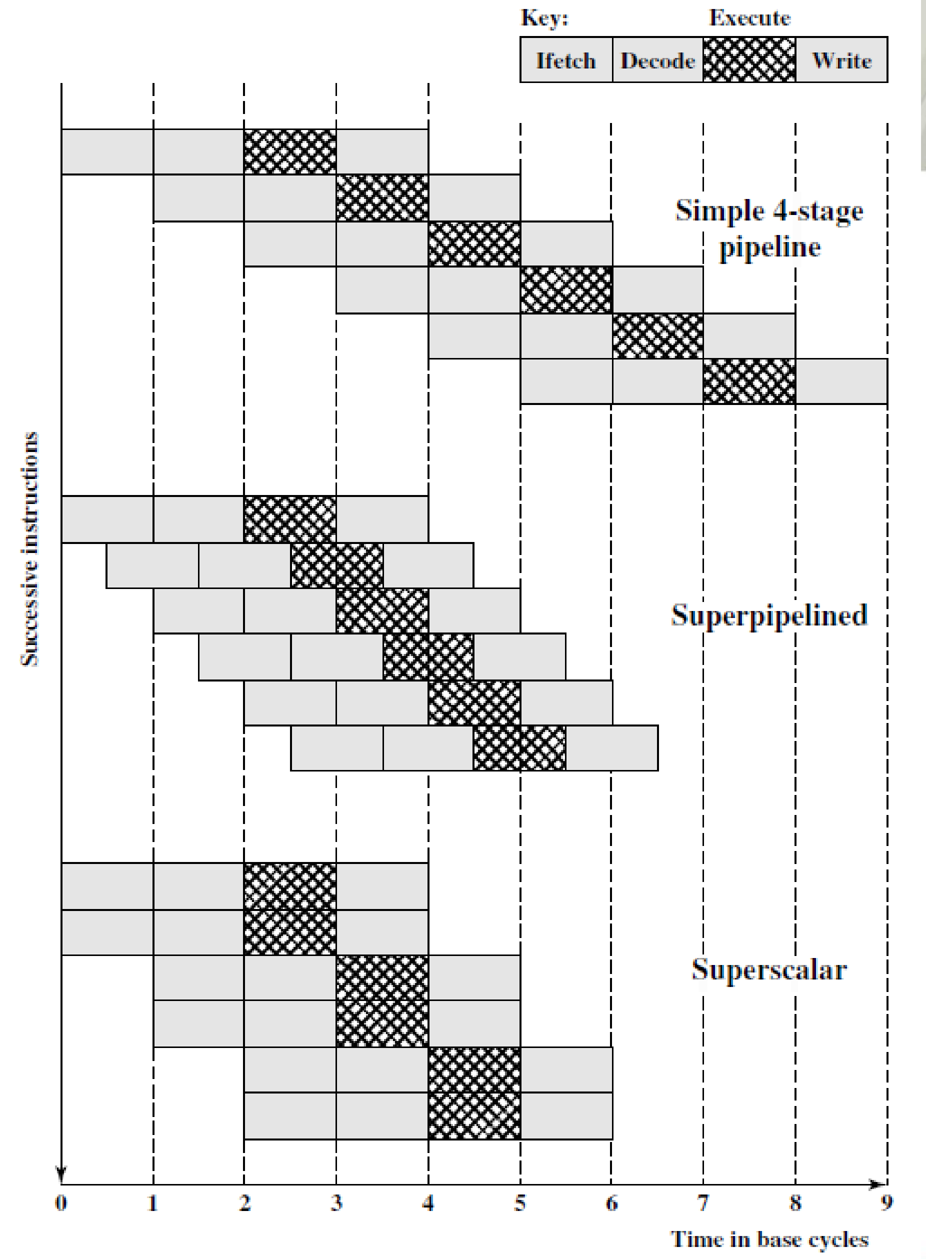
◆ A **superscalar processor** is one in which multiple independent instruction pipelines are used. Each pipeline consists of multiple stages, so that each pipeline can handle multiple instructions at a time. Multiple pipelines introduce a new level of parallelism, enabling multiple streams of instructions to be processed at a time. A superscalar processor exploits what is known as **instruction-level parallelism**, which refers to the degree to which the instructions of a program can be executed in parallel.

◆ A superscalar processor typically fetches multiple instructions at a time and then attempts to find nearby instructions that are independent of one another and can therefore be executed in parallel. If the input to one instruction depends on the output of a preceding instruction, then the latter instruction cannot complete execution at the same time or before the former instruction. Once such dependencies have been identified, the processor may issue and complete instructions in an order that differs from that of the original machine code.

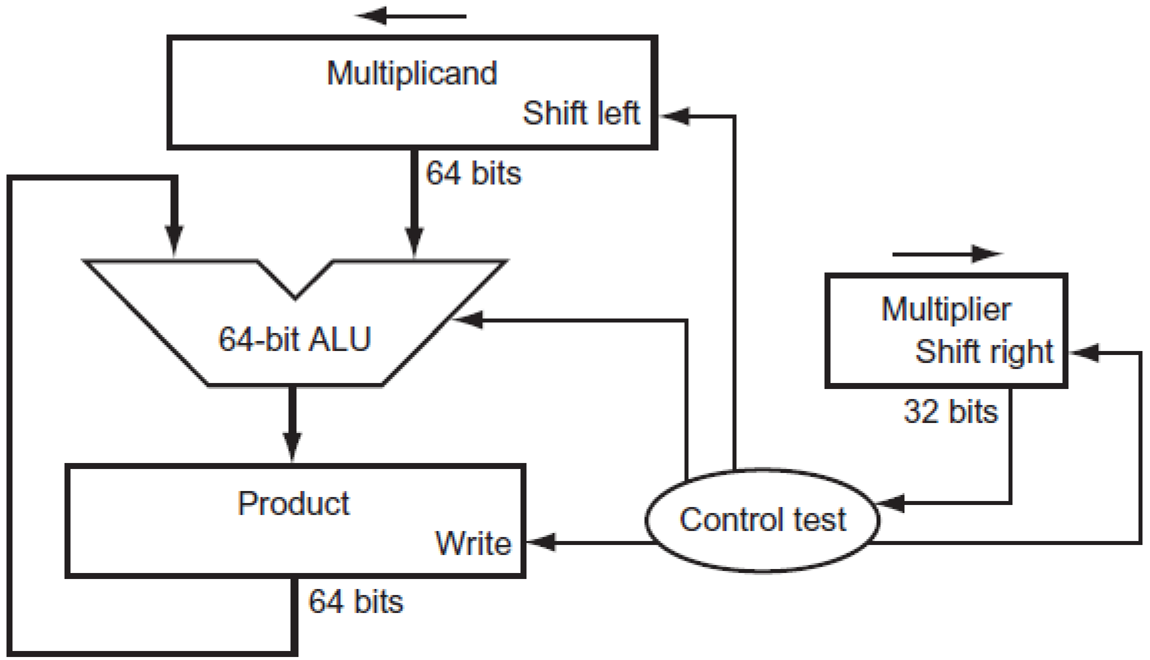
◆ The processor may eliminate some unnecessary dependencies by the use of additional registers and the renaming of register references in the original code.

◆ Whereas pure RISC processors often employ delayed branches to maximize the utilization of the instruction pipeline, this method is less appropriate to a superscalar machine. Instead, most superscalar machines use traditional branch prediction methods to improve efficiency.

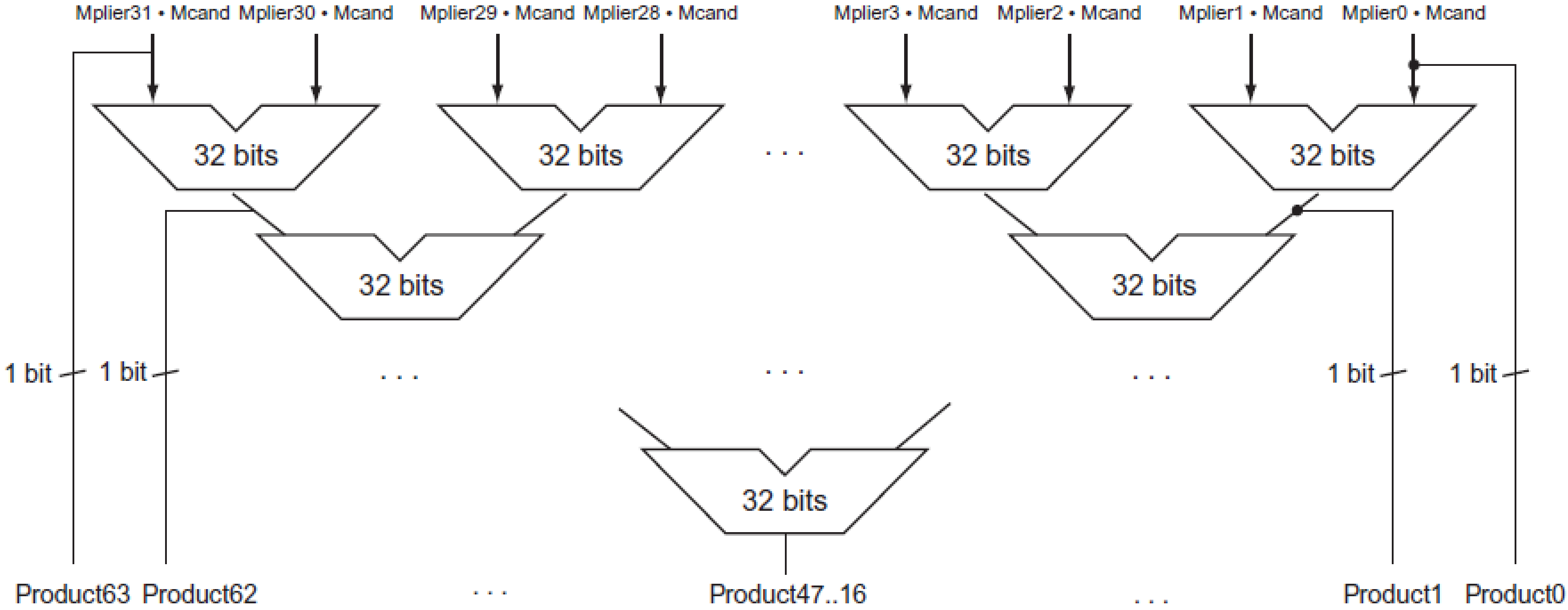
Comparison of Superscalar and Superpipeline Approaches



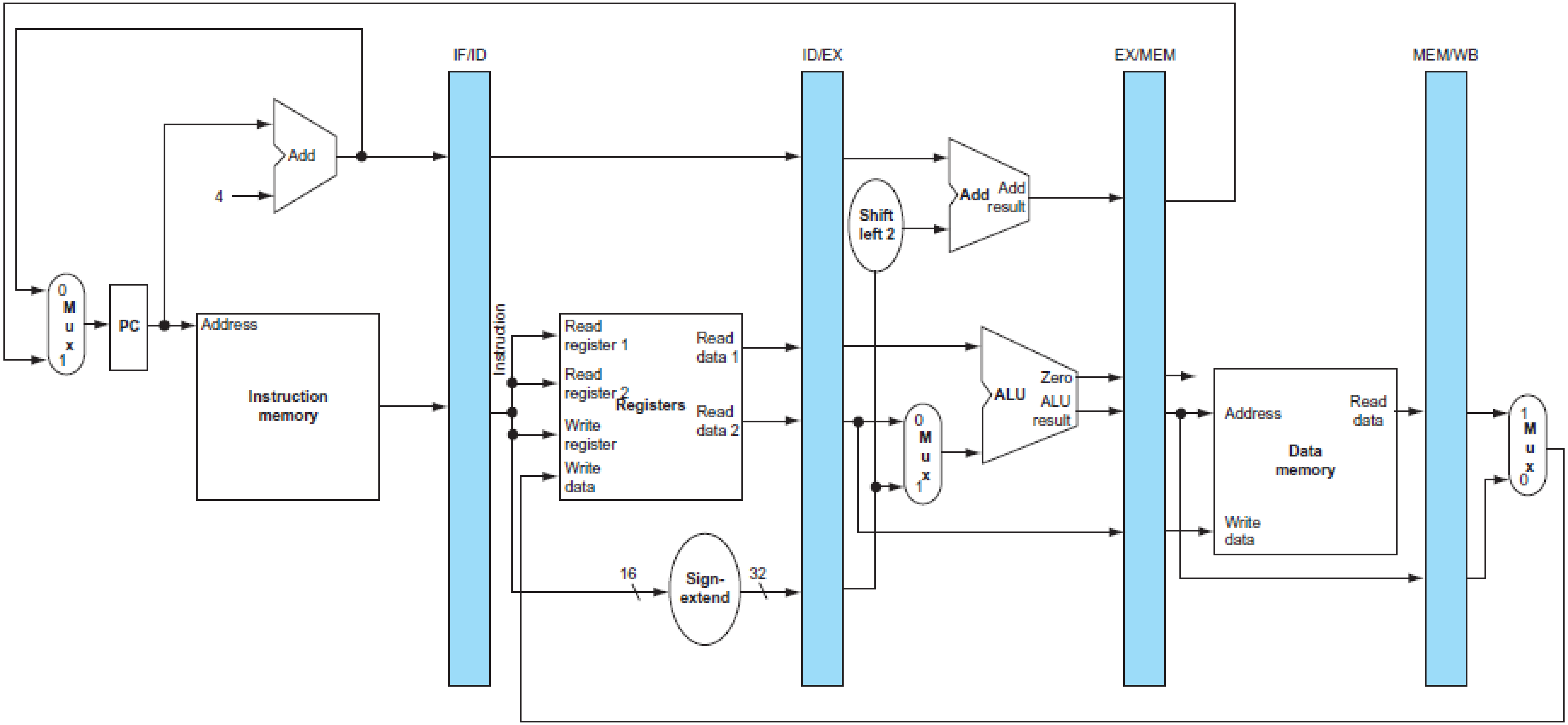
Pierwsza wersja układu mnożącego



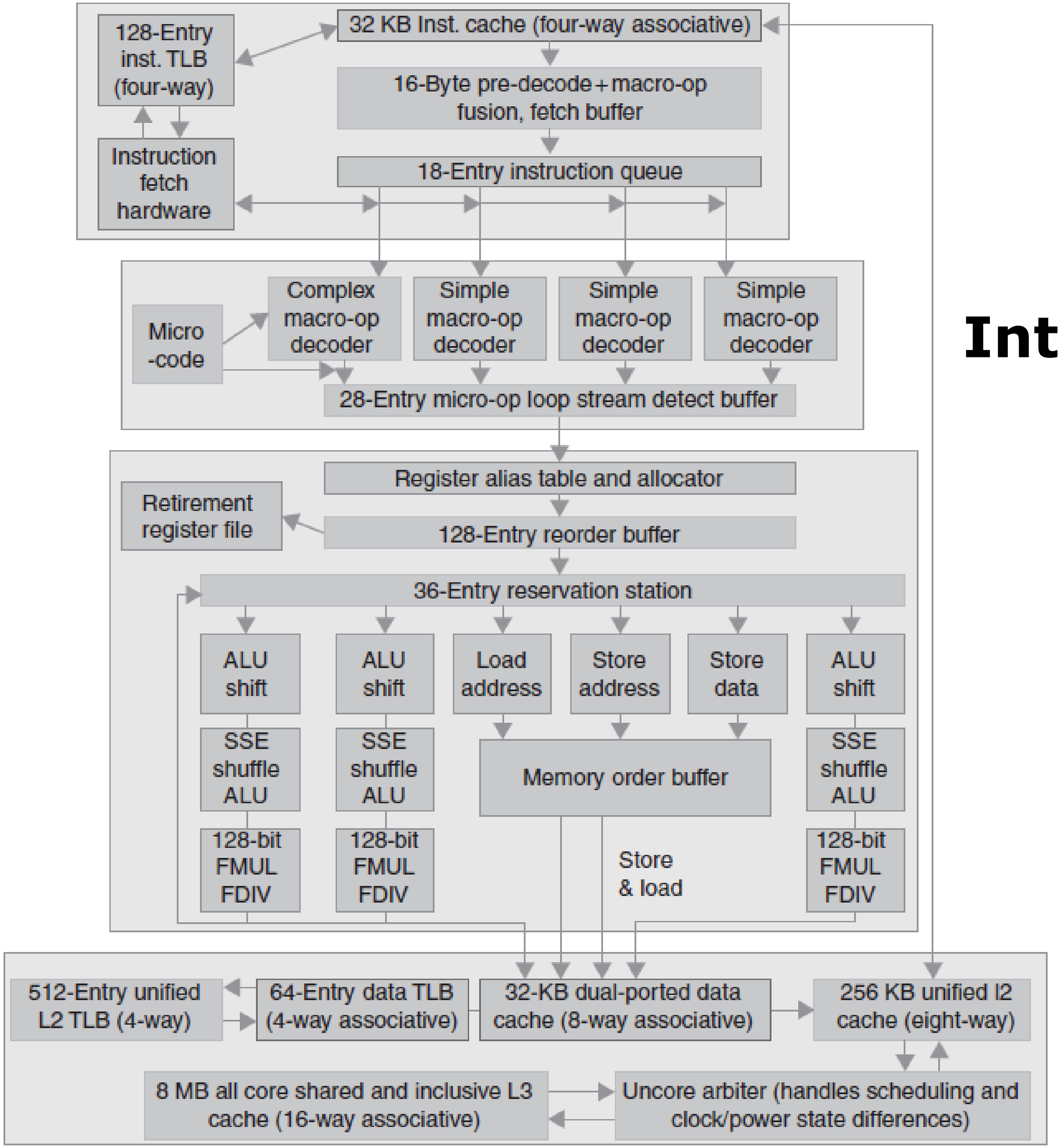
Idea szybkiego układu mnożącego



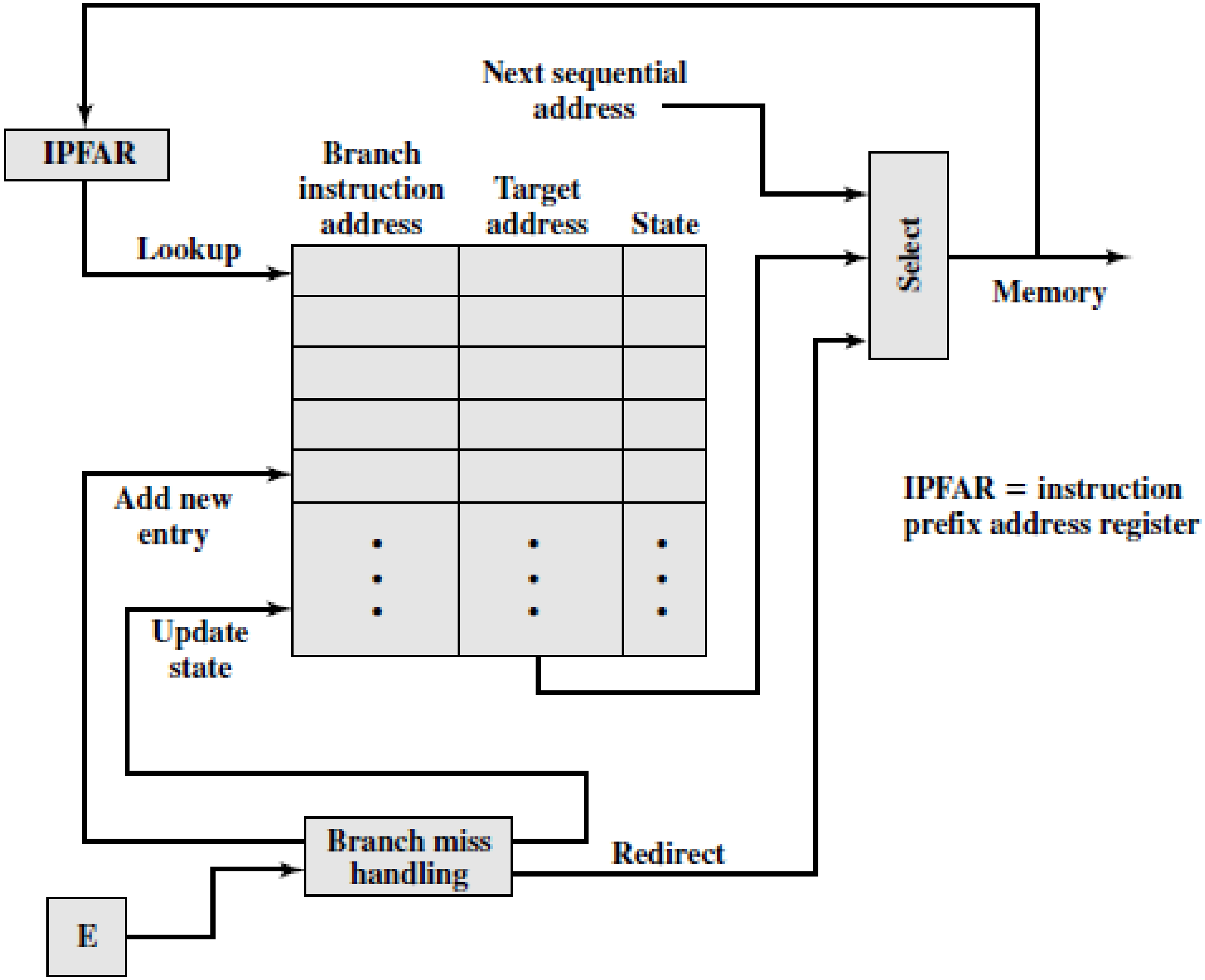
Potok (ang. pipeline)



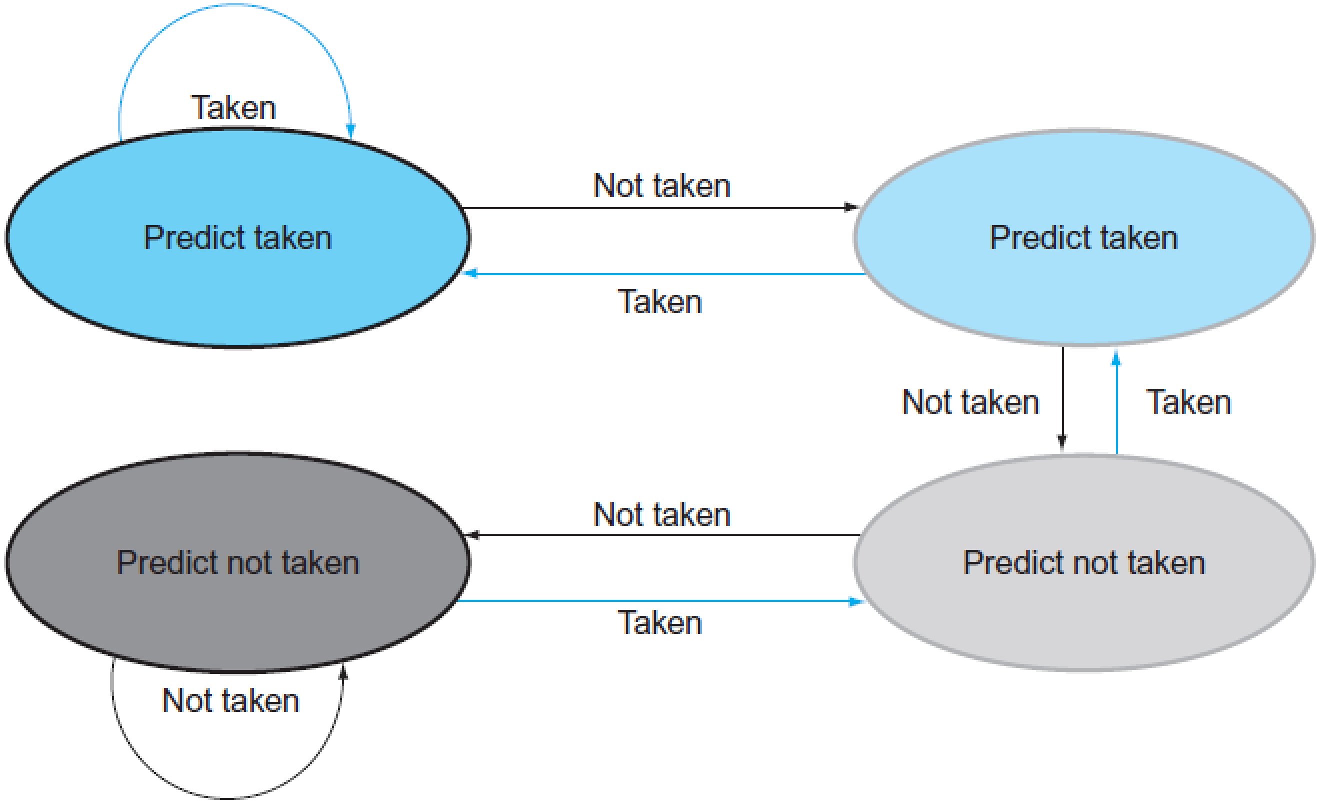
Intel Core i7


128-Entry inst. TLB (four-way)
32 KB Inst. cache (four-way associative)
16-Byte pre-decode + macro-op fusion, fetch buffer
Instruction fetch hardware
18-Entry instruction queue
Micro -code
Complex macro-op decoder
Simple macro-op decoder
Simple macro-op decoder
Simple macro-op decoder
28-Entry micro-op loop stream detect buffer
Register alias table and allocator
Retirement register file
128-Entry reorder buffer
36-Entry reservation station
ALU shift
ALU shift
Load address
Store address
Store data
ALU shift
SSE shuffle ALU
SSE shuffle ALU
Memory order buffer
SSE shuffle ALU
128-bit FMUL FDIV
128-bit FMUL FDIV
Store & load
128-bit FMUL FDIV
512-Entry unified L2 TLB (4-way)
64-Entry data TLB (4-way associative)
32-KB dual-ported data cache (8-way associative)
256 KB unified l2 cache (eight-way)
8 MB all core shared and inclusive L3 cache (16-way associative)
Uncore arbiter (handles scheduling and clock/power state differences)

Branch history table strategy



The states in a 2-bit prediction scheme



Taken
Not taken
Predict taken
Predict taken
Taken
Not taken
Taken
Not taken
Predict not taken
Predict not taken
Taken
Not taken