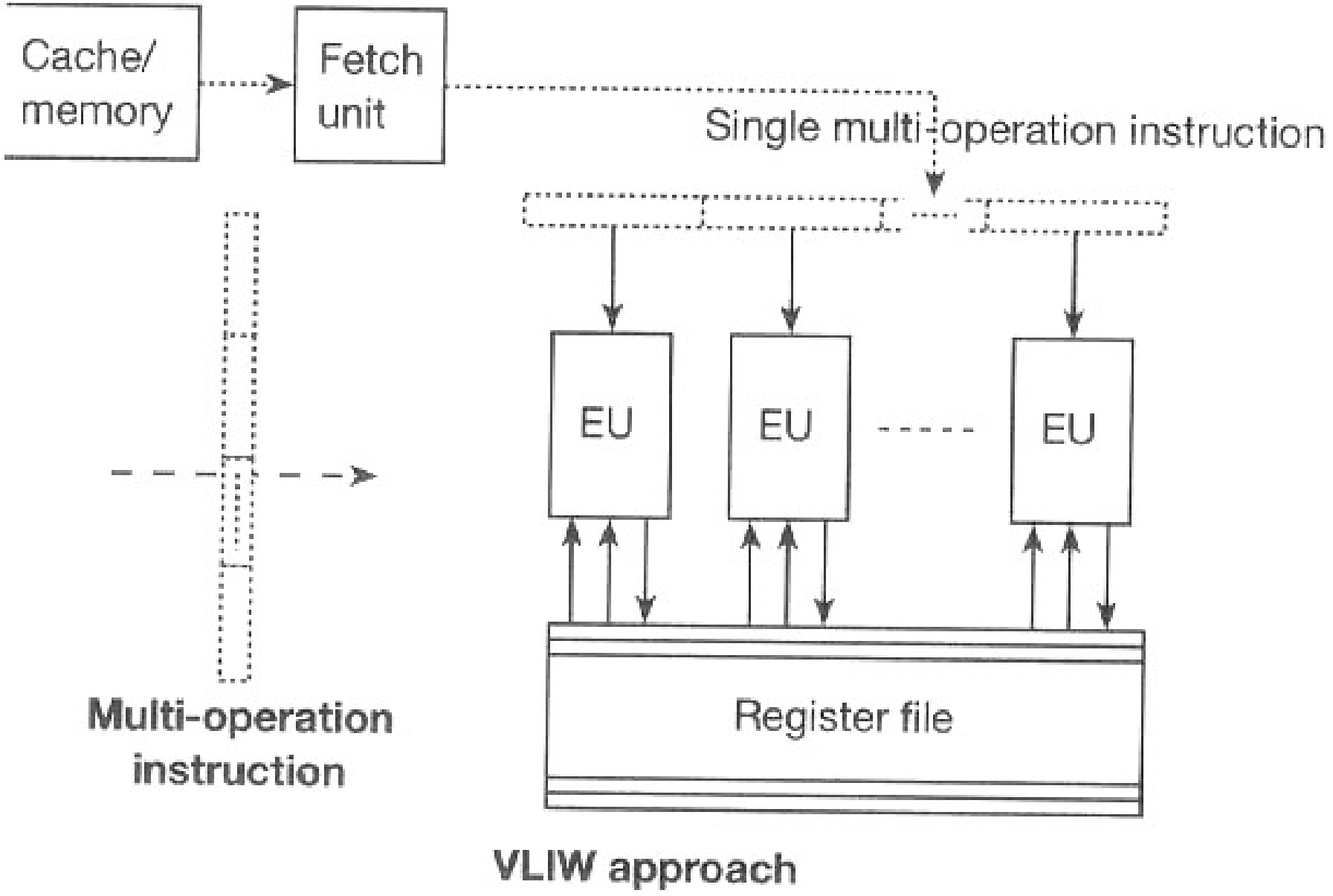
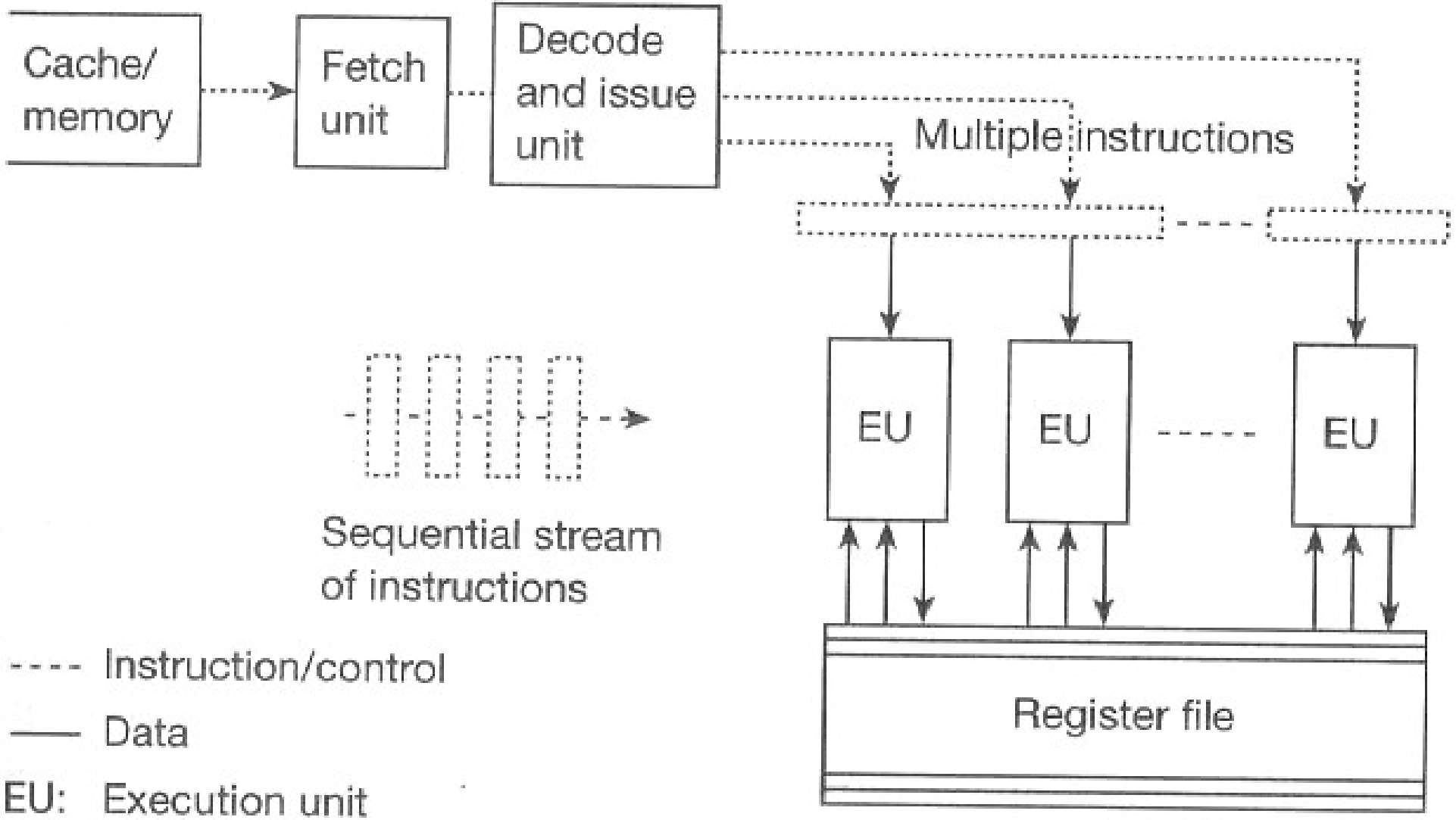
High Performance Architectures

Superscalar and VLIW

VLIW
(very long instruction word,1024 bits!)

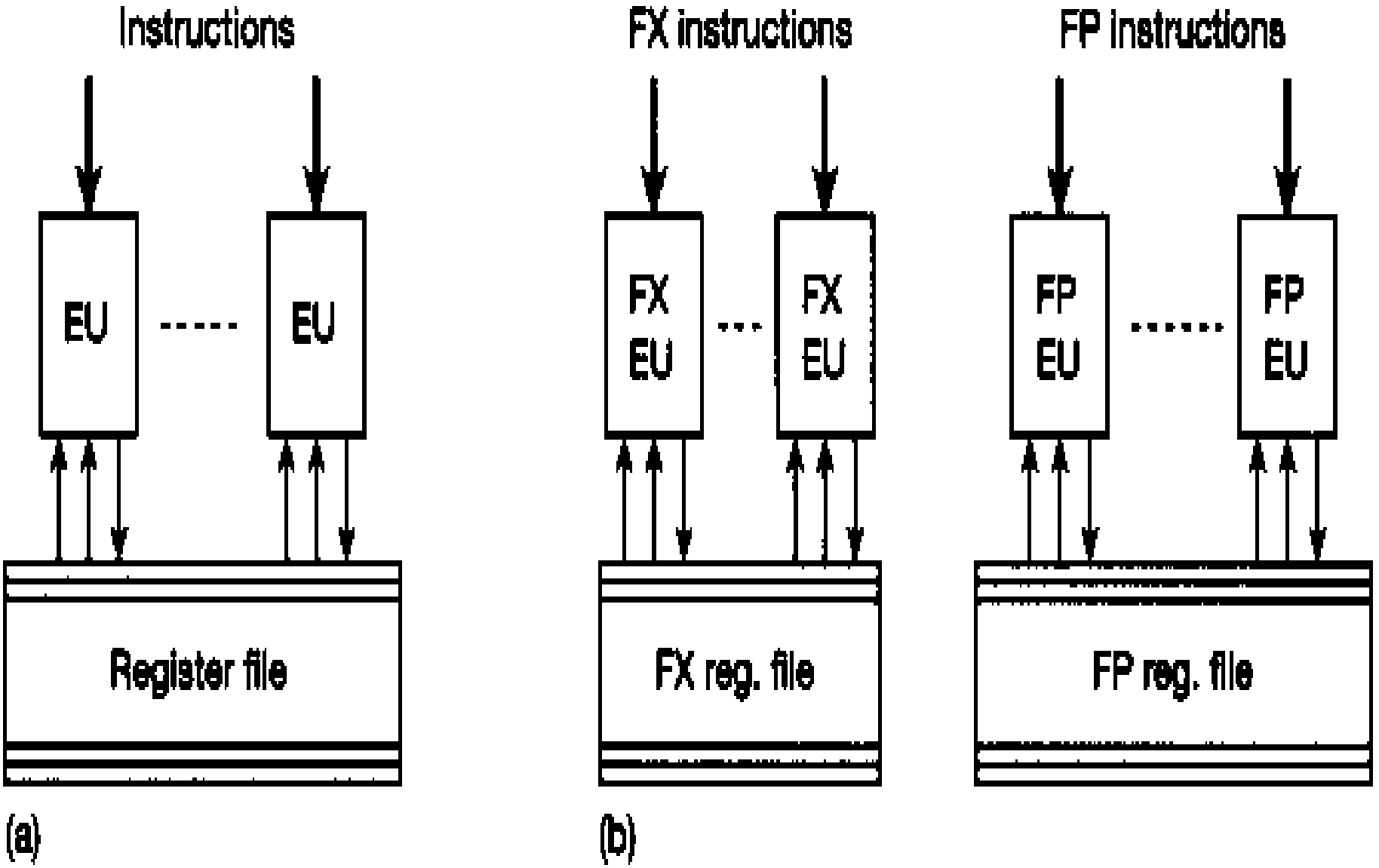


Superscalar (sequential stream of instructions)



Superscalar approach

Basic Structure of VLIW Architecture



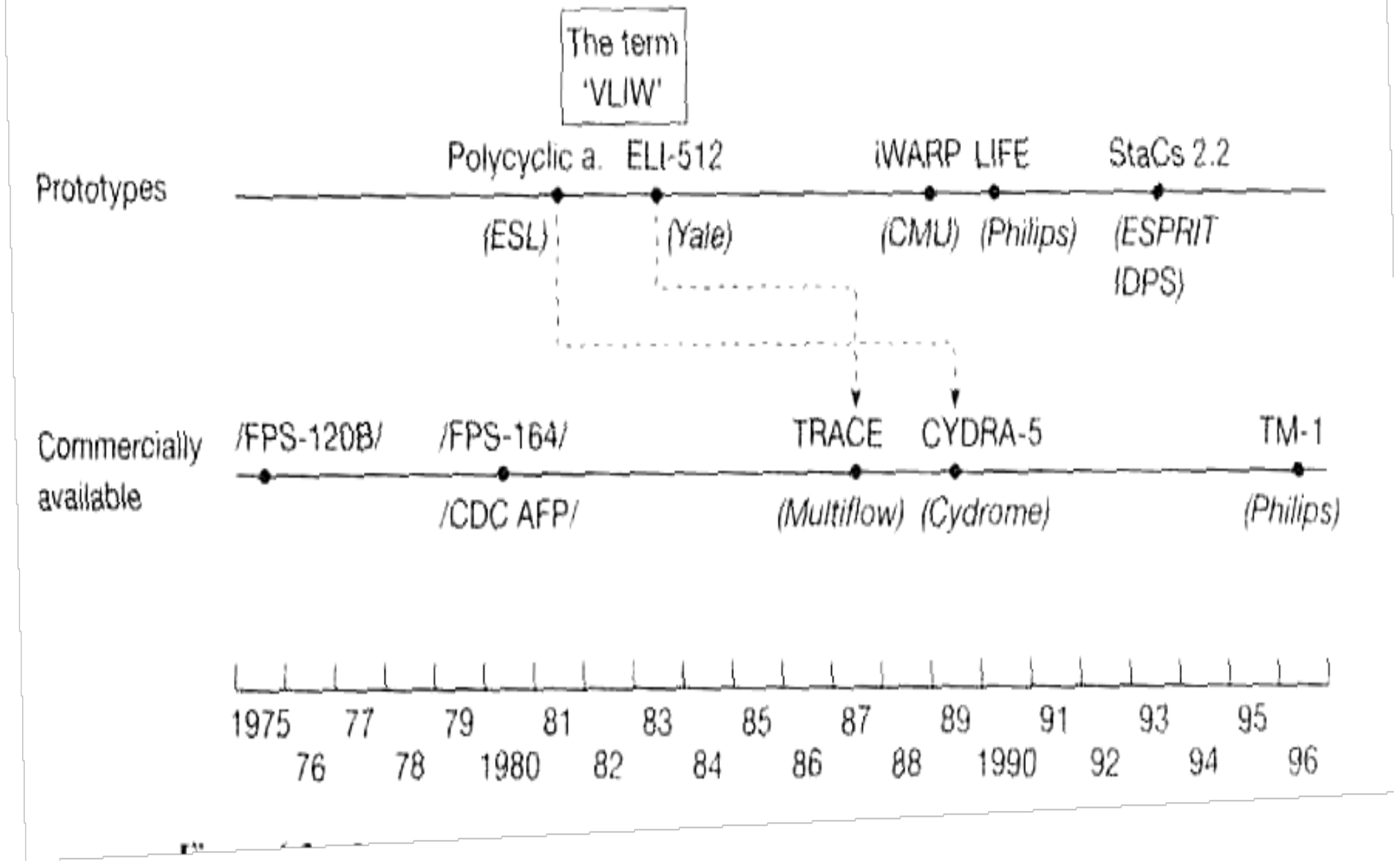
Basic principle of VLIW

- Controlled by very long instruction words
  - comprising a control field for each of the execution units
- Length of instruction depends on
  - the number of execution units (5-30 EU)
  - the code lengths required for controlling each EU (16-32 bits)
  - 256 to 1024 bits
- Disadvantages: on average only some of the control fields will actually be used
  - waste memory space and memory bandwidth
  - e.g. Fortran code is 3 times larger for VLIW (Trace processor)

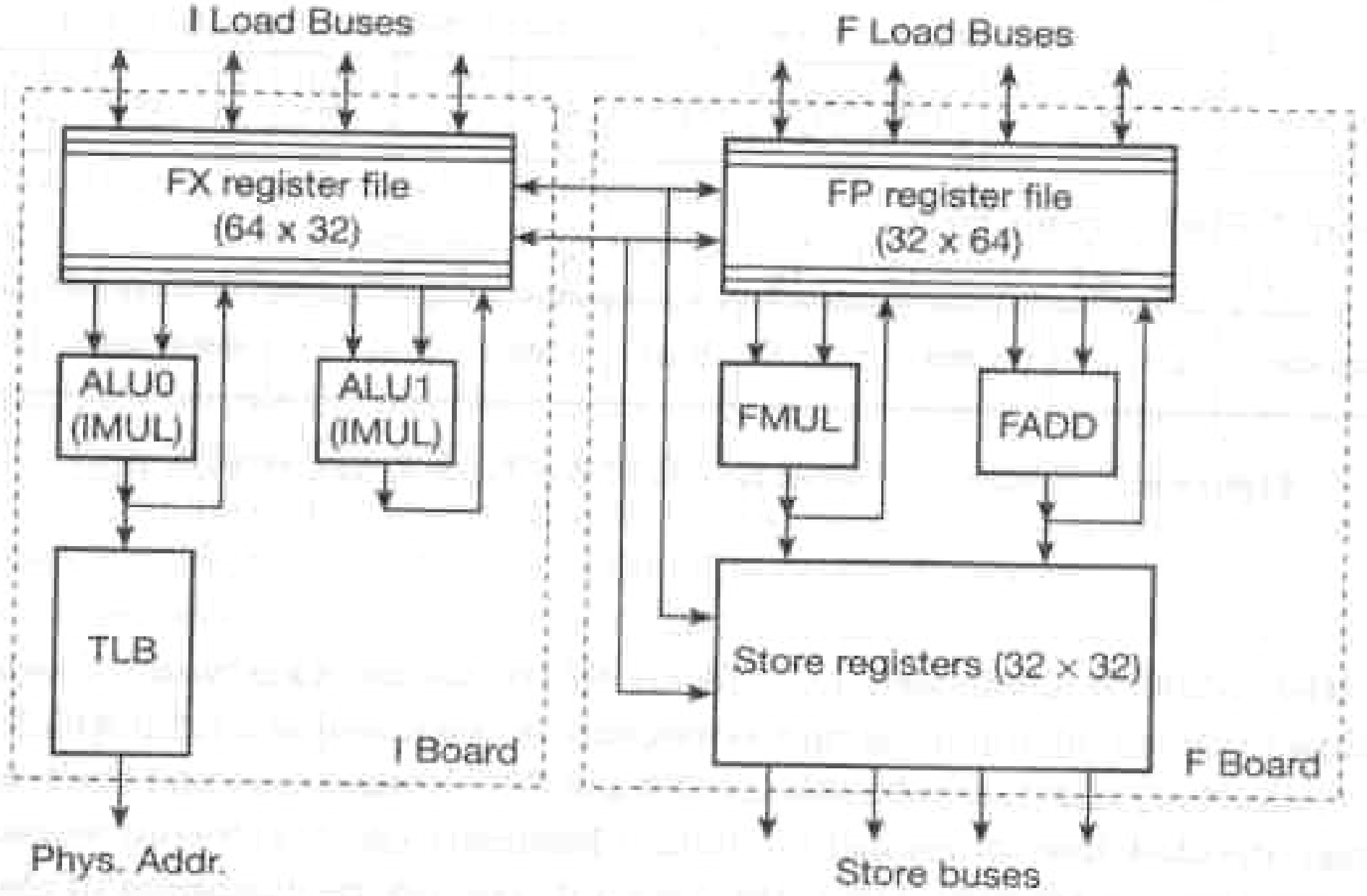
VLIW: Static Scheduling of instructions/

- Instruction scheduling done entirely by [software] compiler
- Lesser Hardware complexity translate to
  - increase the clock rate
  - raise the degree of parallelism (more EU); {Can this be utilized?}
- Higher Software (compiler) complexity
  - compiler needs to aware hardware detail
    - number of EU, their latencies, repetition rates, memory load-use delay, and so on
    - cache misses: compiler has to take into account worst-case delay value
  - this hardware dependency restricts the use of the same compiler for a family of VLIW processors

6.2 overview of proposed and commercial VLIW architectures



6.3 Case study: Trace 200



Trace 7/200

- 256-bit VLIW words
- Capable of executing 7 instructions/cycle
  - 4 integer operations
  - 2 FP
  - 1 Conditional branch
- Found that every 5^{th} to 8^{th} operation on average is a conditional branch
- Use sophisticated branching scheme: multi-way branching capability
  - executing multi-paths
  - assign priority code corresponds to its relative order

Trace 28/200: storing long instructions

- 1024 bit per instruction
- a number of 32-bit fields maybe empty
- Storing scheme to save space
  - 32-bit mask indicating each sub-field is empty or not
  - followed by all sub-fields that are not empty
  - resulting still 3 time larger memory space required to store Fortran code (vs. VAX object code)
  - very complex hardware for cache fill and refill
- Performance data is Impressive indeed!

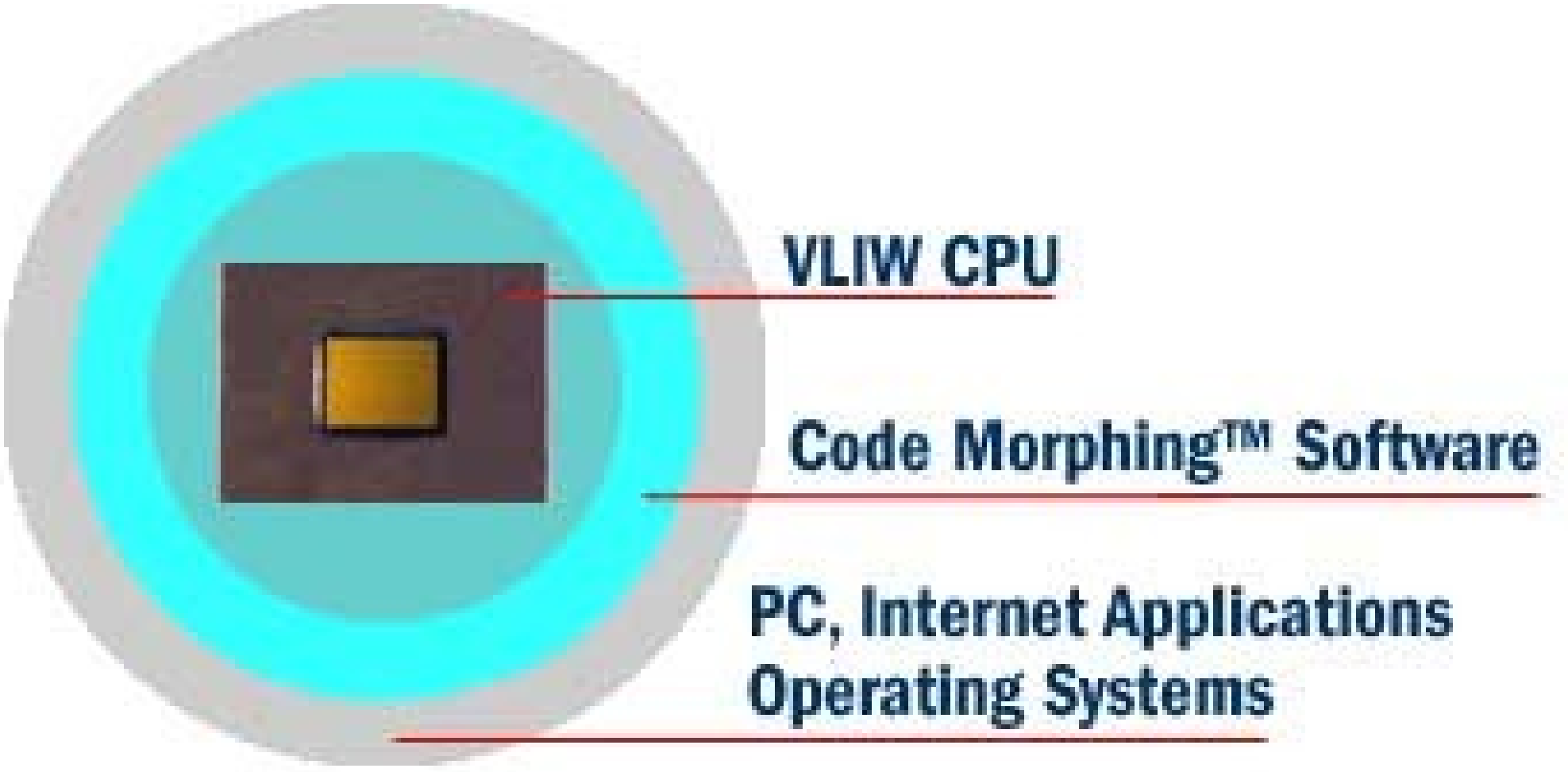
Trace: Performance data

Table 6.1 Performance data for Trace processors compared with that of the DEC 8700 and Cray XMP.

	Instruction issue rate (ns)	*Compiled Linpack (Full precision) (MFLOPS)*
Trace 7/200	130	6
Trace 14/200	130	10
DEC 8700	45	0.97
Cray XMP	8	24

Transmeta: Crusoe

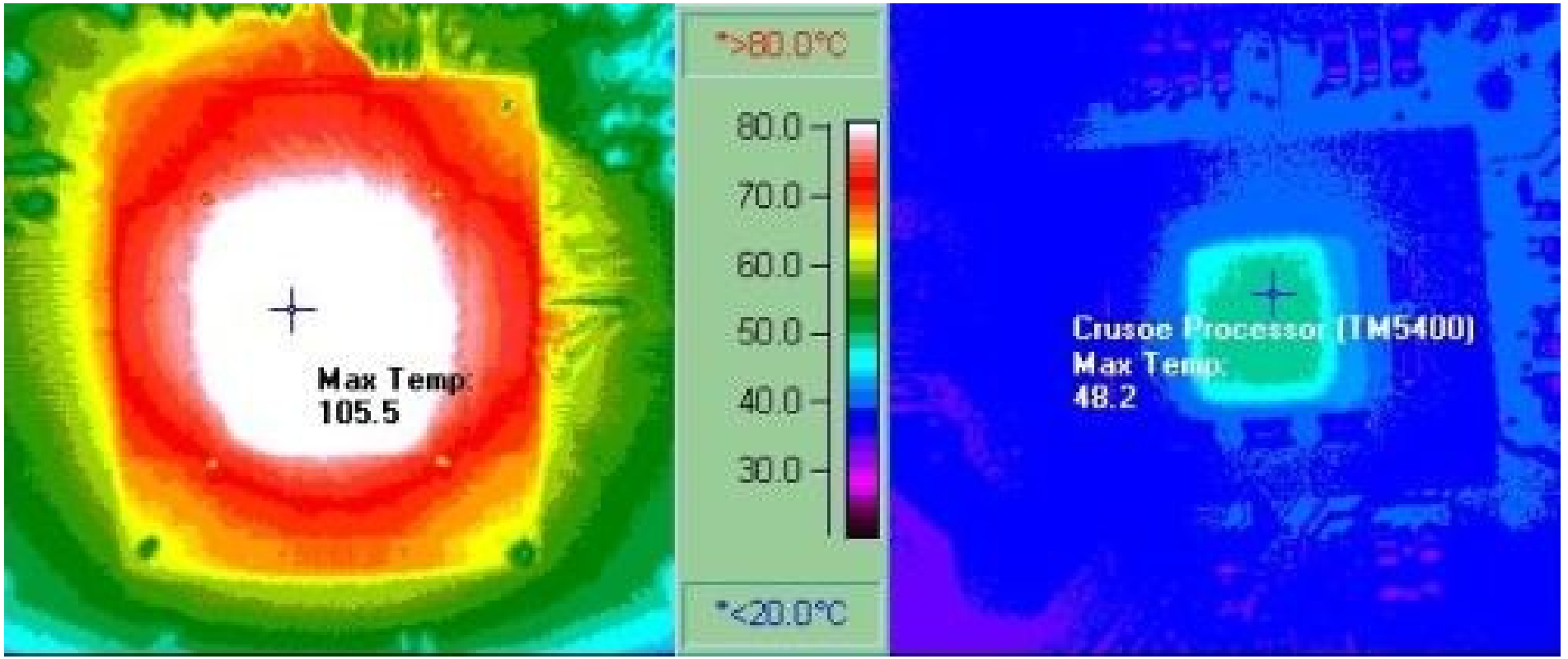
- Full x86-compatible: by dynamic code translation
- High performance: 700MHz in mobile platforms



Crusoe

- “Remarkably low power consumption”
  - “A full day of web browsing on a single battery charge”

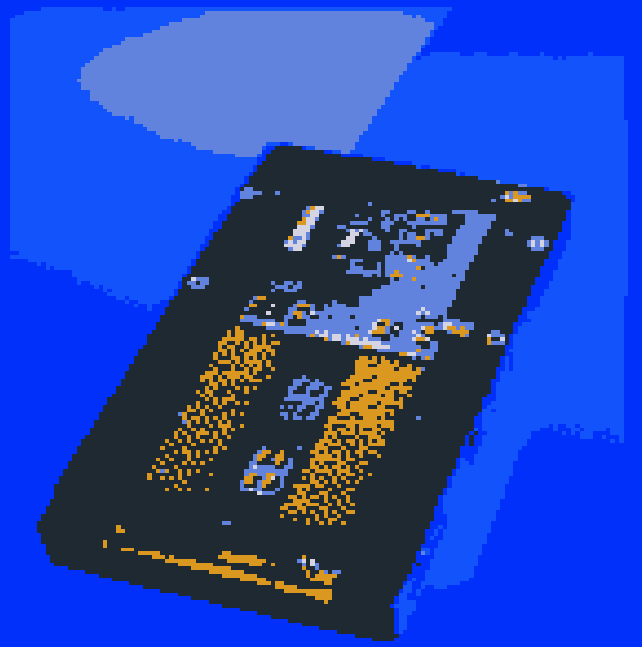
- Playing DVD: Conventional? 105 C vs. Crusoe: 48 C



Intel: Itanium (VLIW) {EPIC} Processor

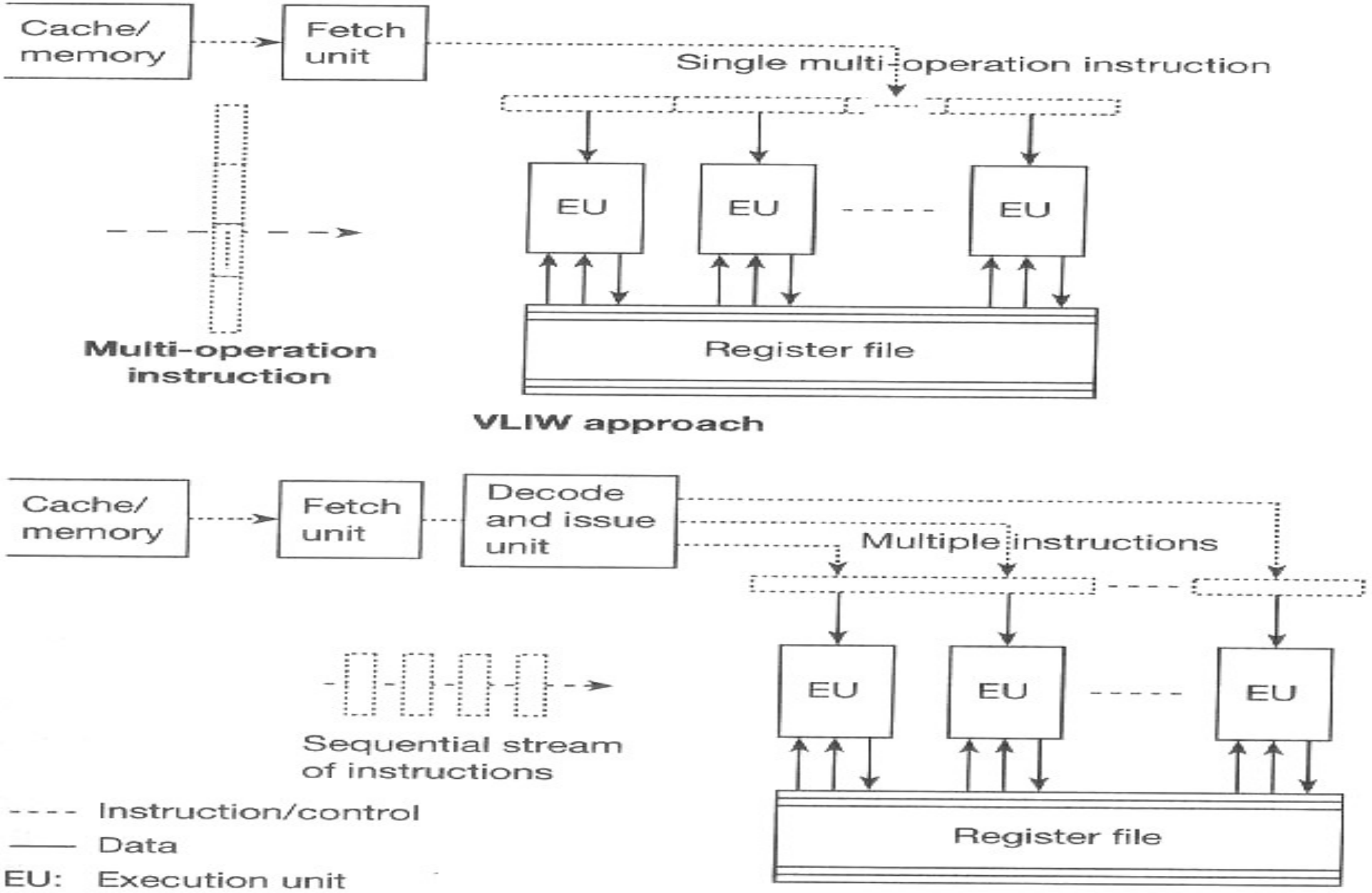
Itanium™ Processor

- 800 MHz production frequency
  - EPIC enables up to 20 operations per clock
- 4 MB high speed on-cartridge L3 cache
- Over 320M transistors
  - 25M in CPU, 295M in L3 cache
- 2.1 GB/s multi-drop system bus
  - Enhanced Defer Mechanism enables high scalability through improved bus efficiency
- Extensive reliability and availability
  - ECC, parity protection, enhanced MCA
- Excellent functionality on initial silicon
  - No architectural or ISA changes required
  - MP functionality on track to plan



Tuning / testing for production this year

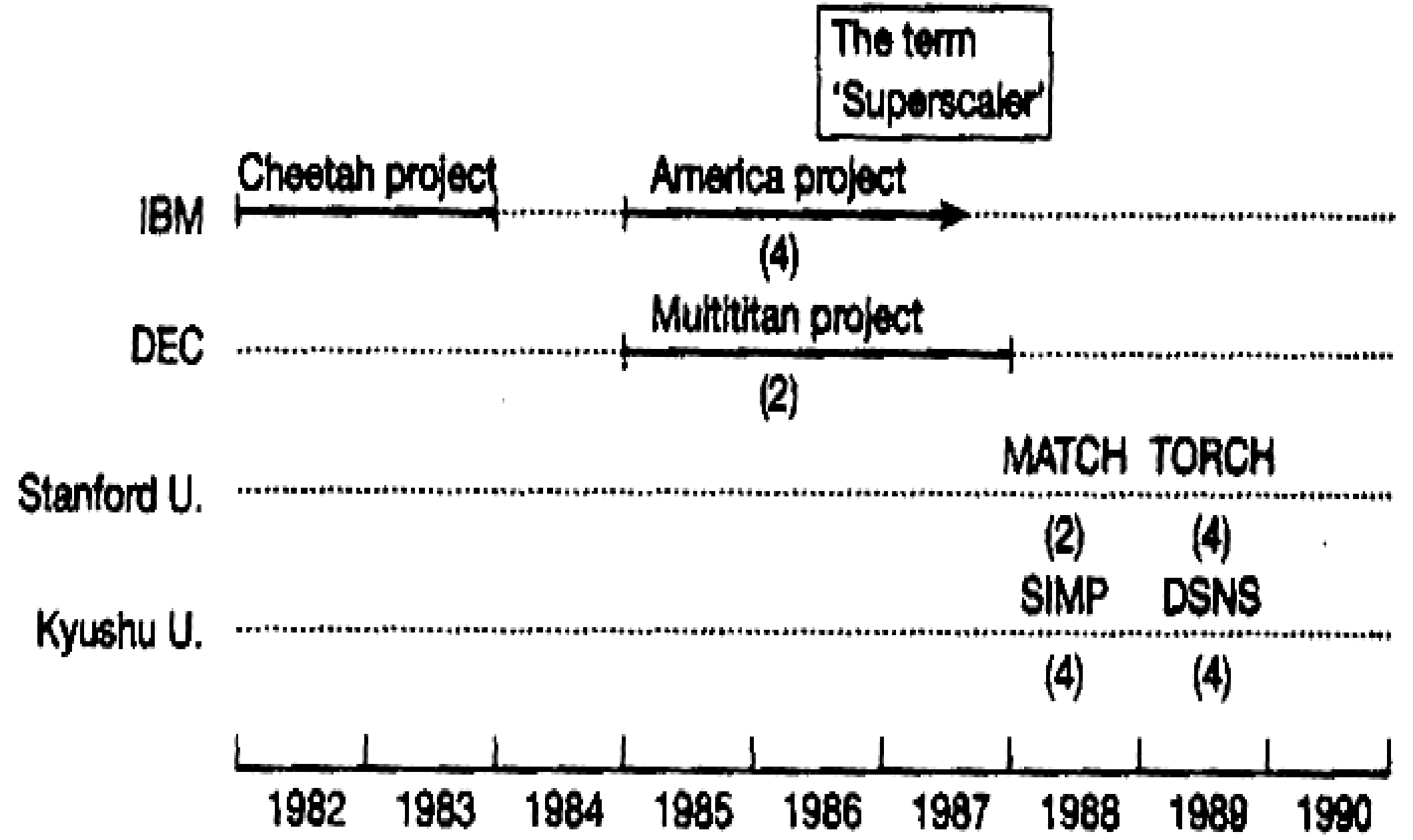
Superscalar Processors vs. VLIW



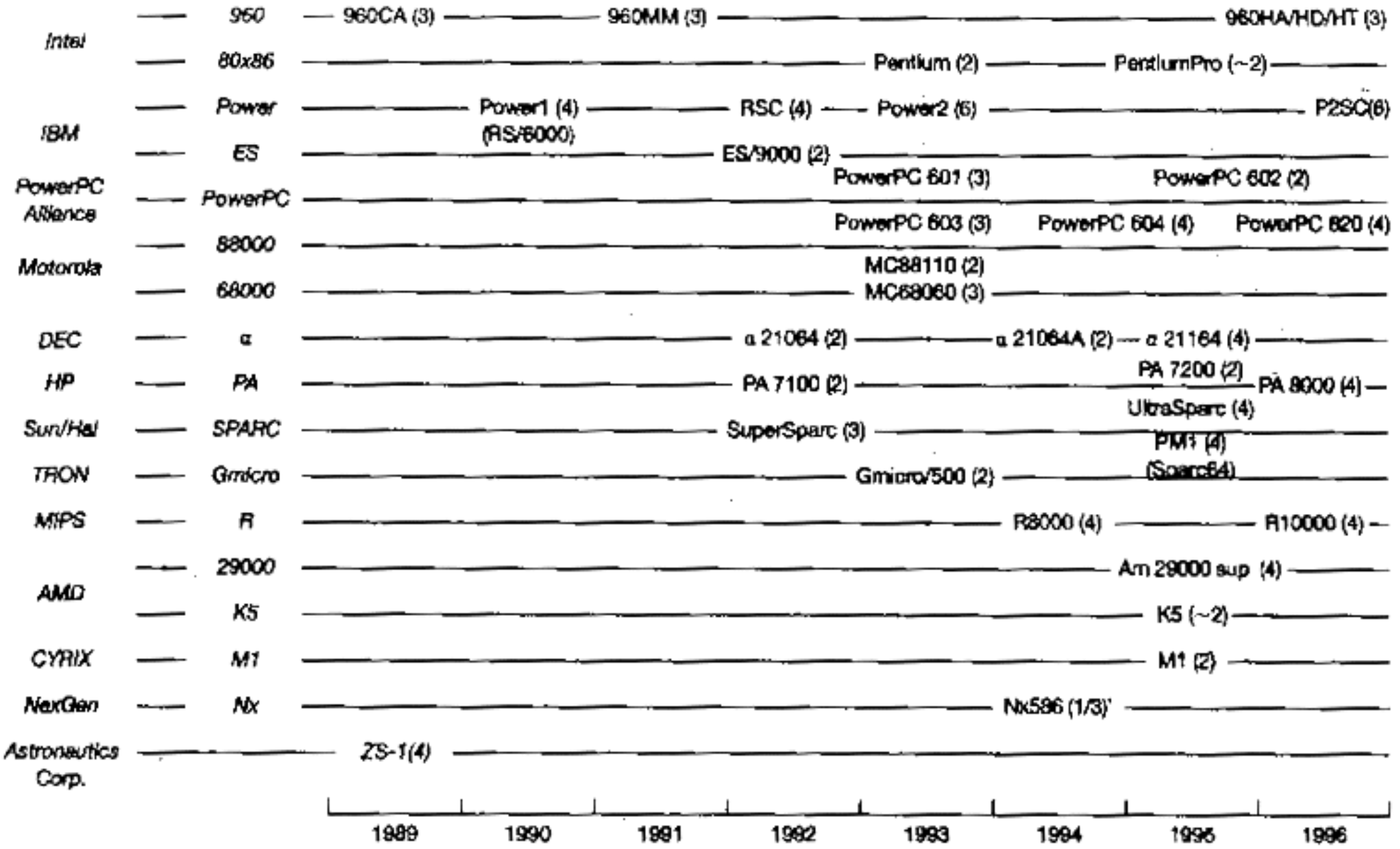
Superscalar Processor: Intro

- Parallel Issue
- Parallel Execution
- {Hardware} Dynamic Instruction Scheduling
- Currently the predominant class of processors
  - Pentium
  - PowerPC
  - UltraSparc
  - AMD K5-
  - HP PA7100-
  - DEC α

Emergence and spread of superscalar processors

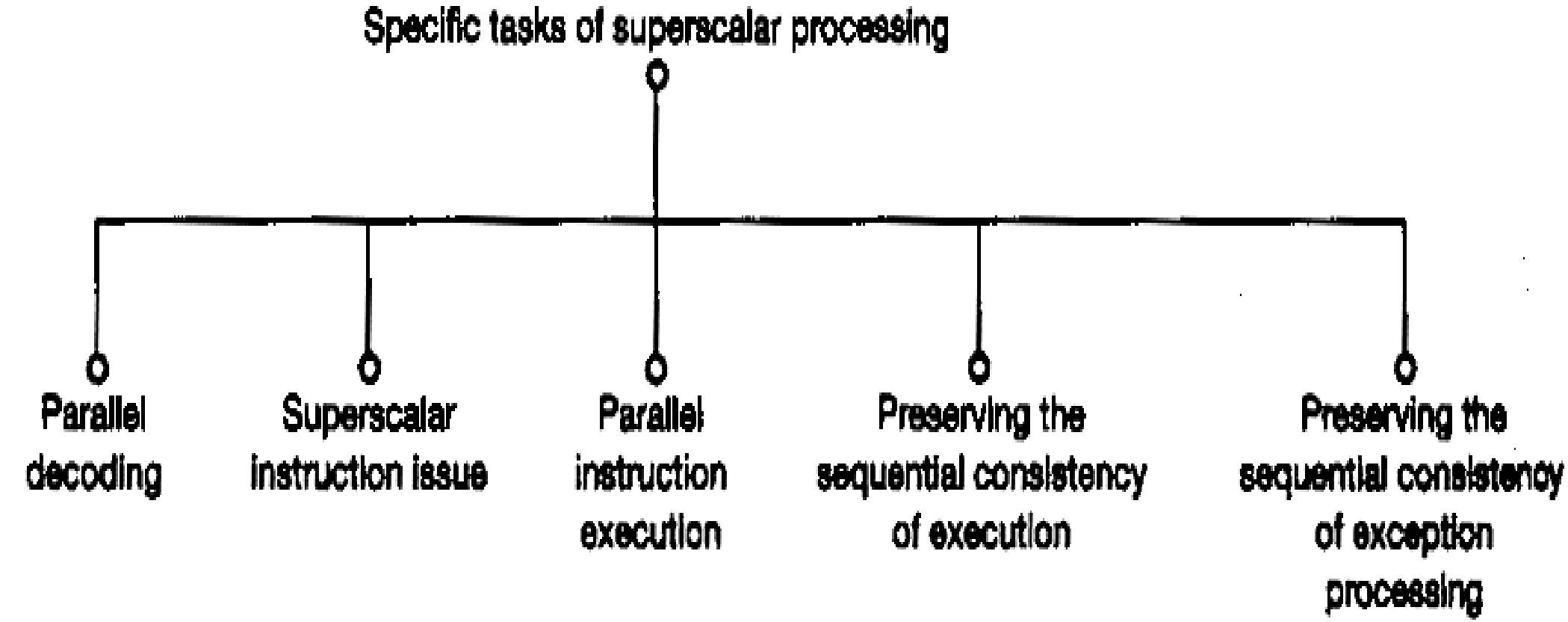


Evolution of superscalar processor



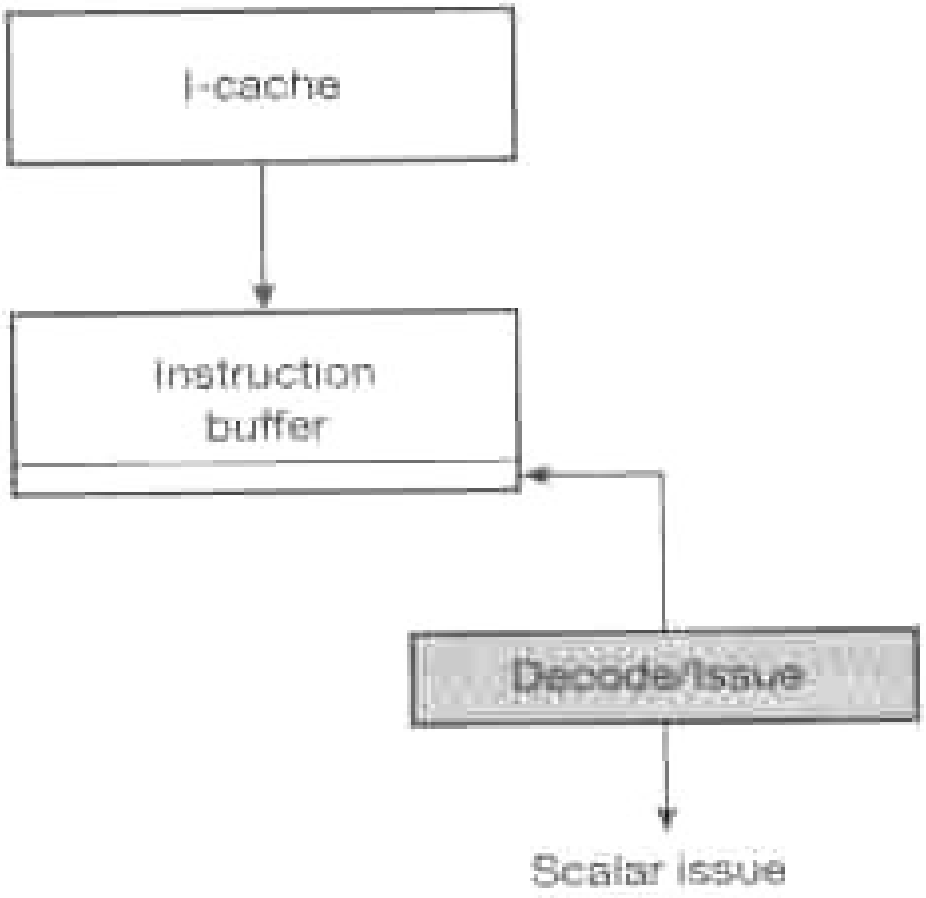
'The Nx586 has scalar issue for CISC instructions but a 3-way superscalar core for converted RISC instructions

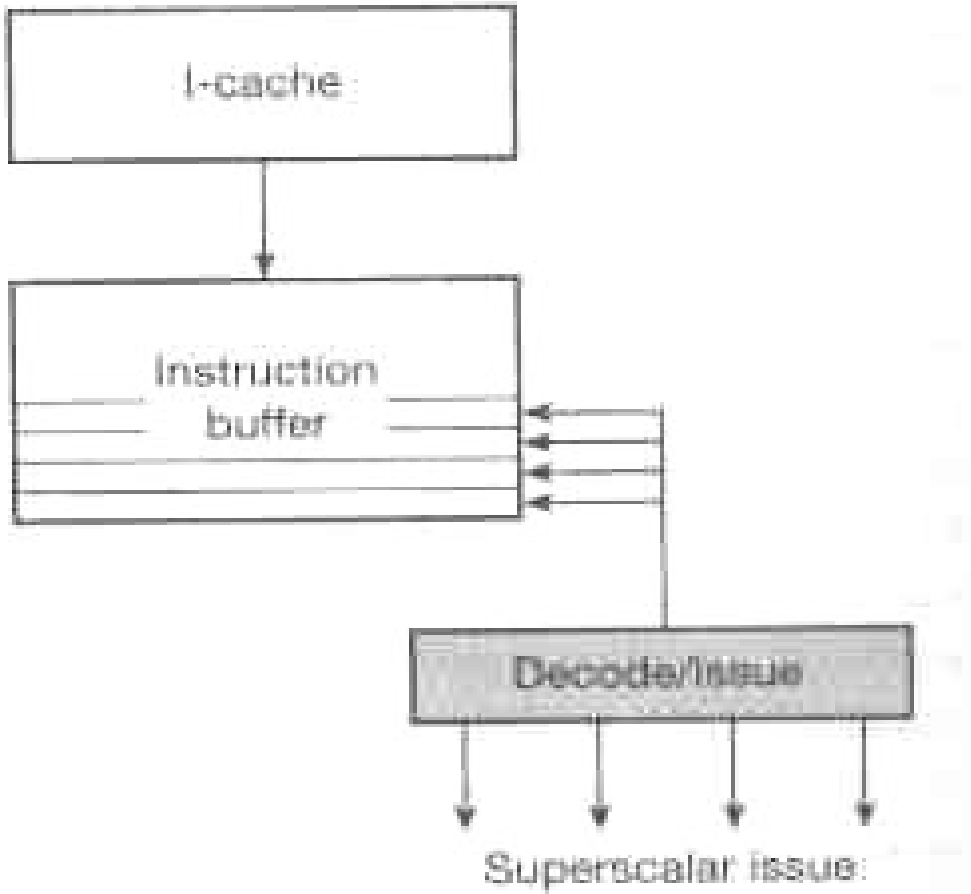
Specific tasks of superscalar processing



Parallel decoding {and Dependencies check}

- What need to be done



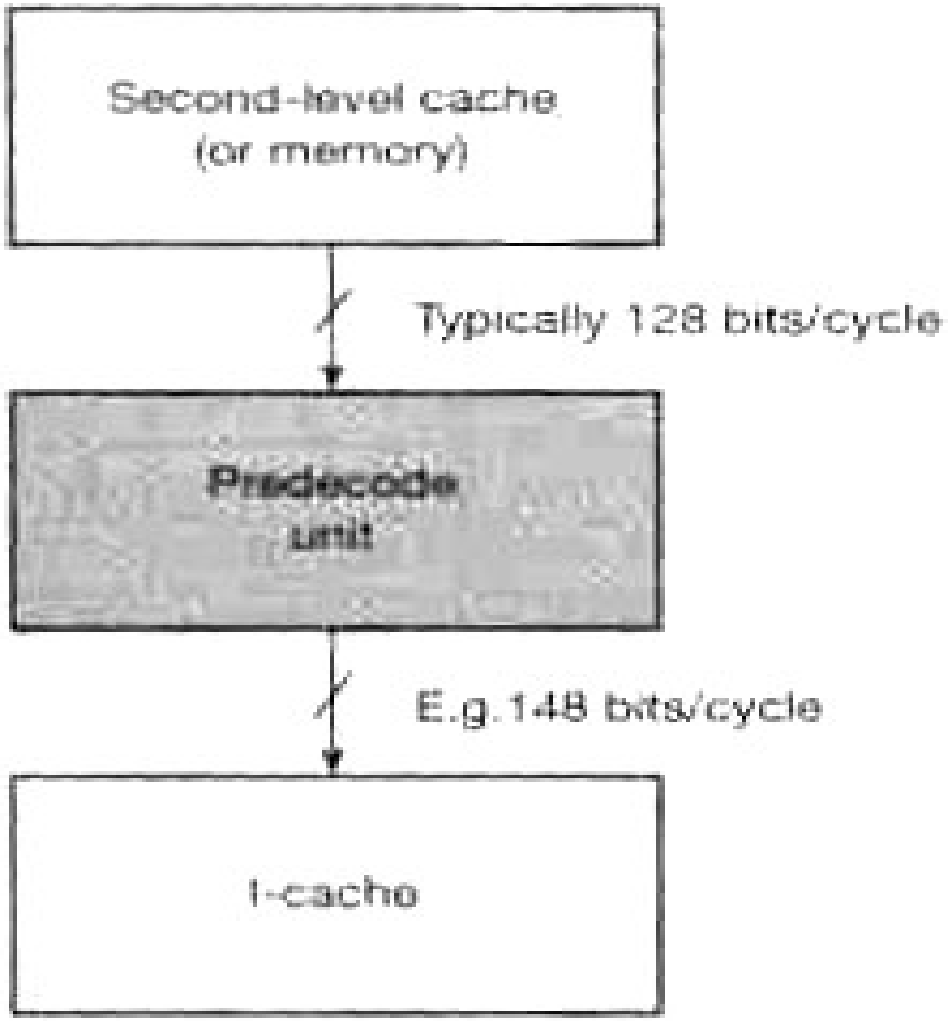


Typical FX pipeline layout

Decoding and Pre-decoding

- Superscalar processors tend to use 2 and sometimes even 3 or more pipeline cycles for decoding and issuing instructions
- >> Pre-decoding:
  - shifts a part of the decode task up into loading phase
  - resulting of pre-decoding
    - the instruction class
    - the type of resources required for the execution
    - in some processor (e.g. UltraSparc), branch target addresses calculation as well
  - the results are stored by attaching 4-7 bits
- + shortens the overall cycle time or reduces the number of cycles needed

The principle of predecoding



When instructions are written into the I-cache, the predecode unit appends 4–7 bits to each RISC instruction [1]

[1] In the AMD K5, which is an x86-compatible CISC processor, the predecode unit appends 5 bits to each byte

Number of predecode bits used

Table 7.1 Number of predecode bits used.

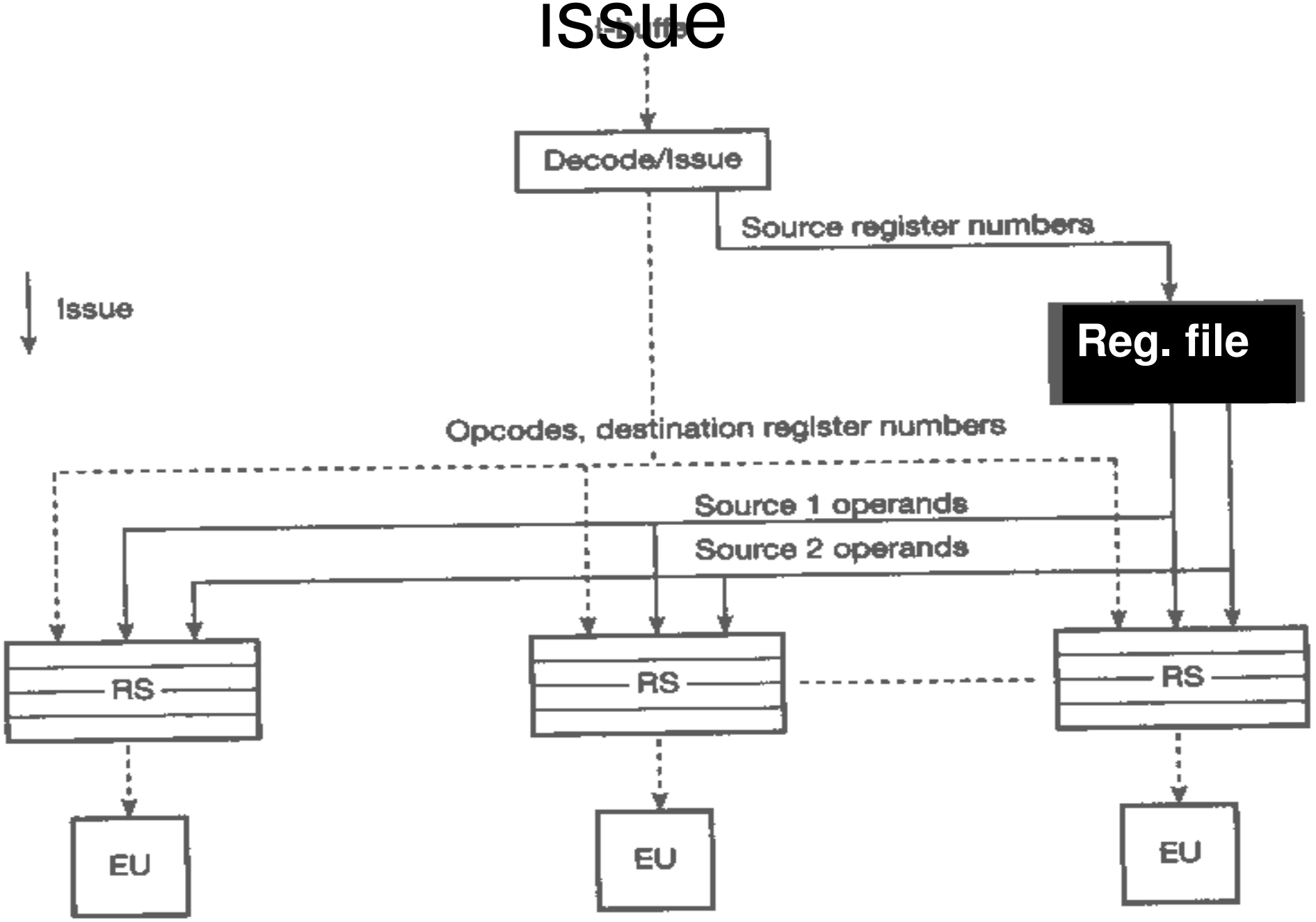
Type/year of first volume shipment	*Number of predecode bits appended to each instruction*
PA 7200 (1995)	5
PA 8000 (1996)	5
PowerPC 620 (1996)	7
UltraSparc (1995)	4
HAL PM1 (1995)	4
AMD K5 (1995)	5†
R10000 (1996)	4

†In the K5, 5 predecode bits are added to each byte

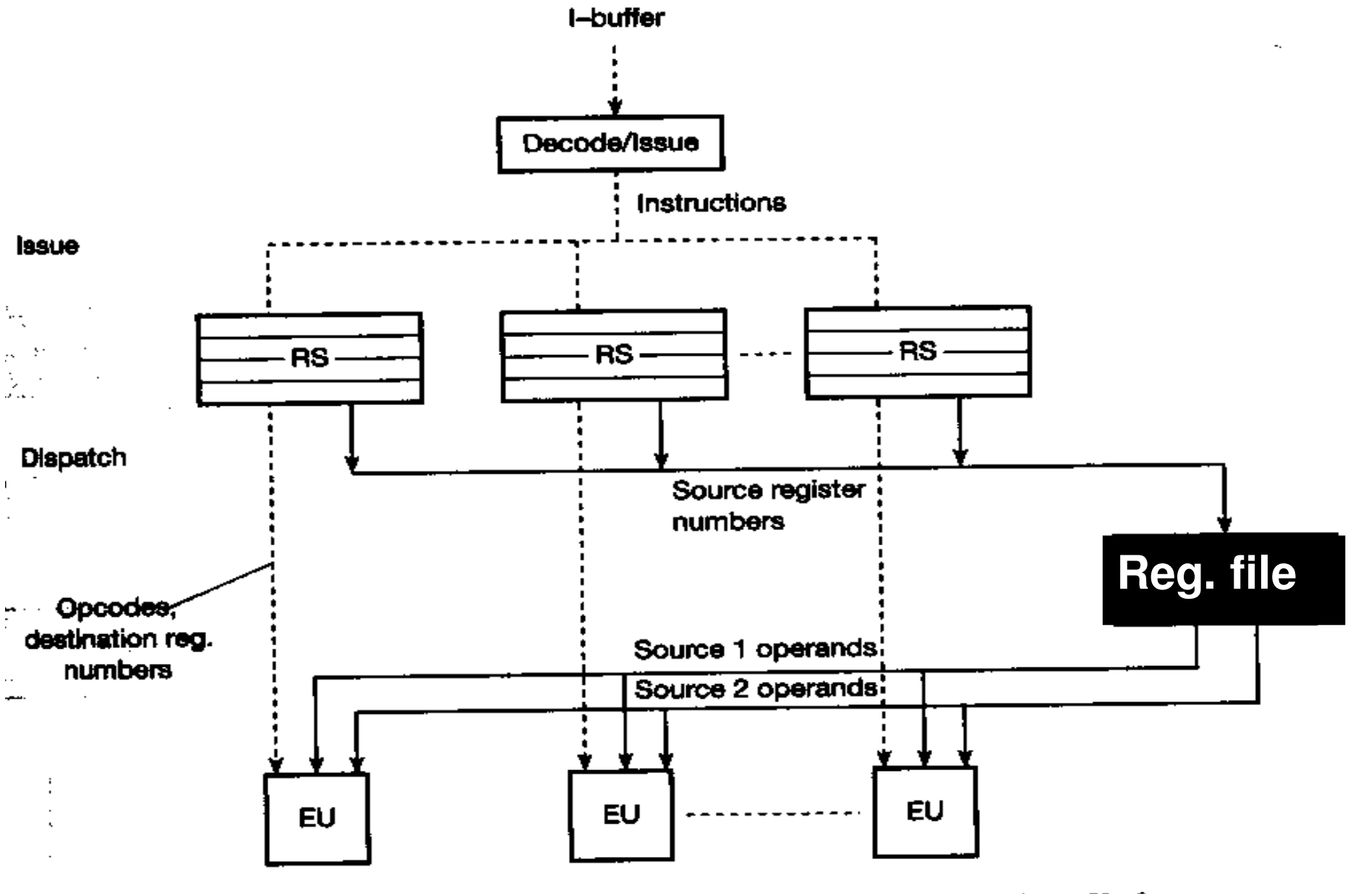
Number of read and write ports

- how many instructions may be written into (input ports) or
- read out from (output ports) a particular shelving buffer in a cycle
- depend on individual, group, or central reservation stations

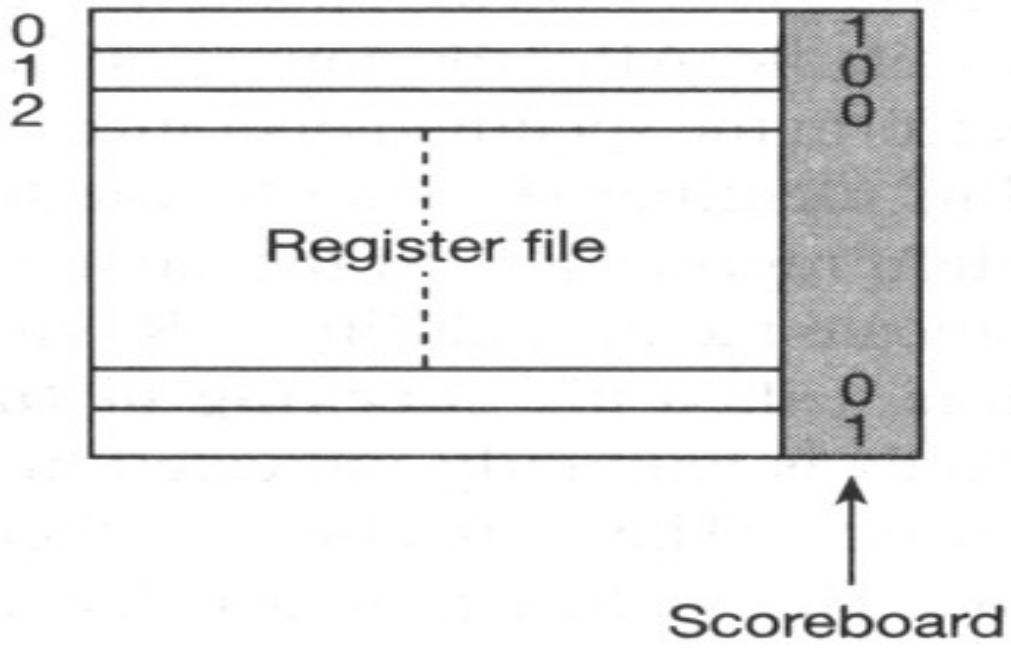
Operand fetch during instruction issue



Operand fetch during instruction dispatch



- Scheme for checking the availability of operands: The principle of scoreboarding

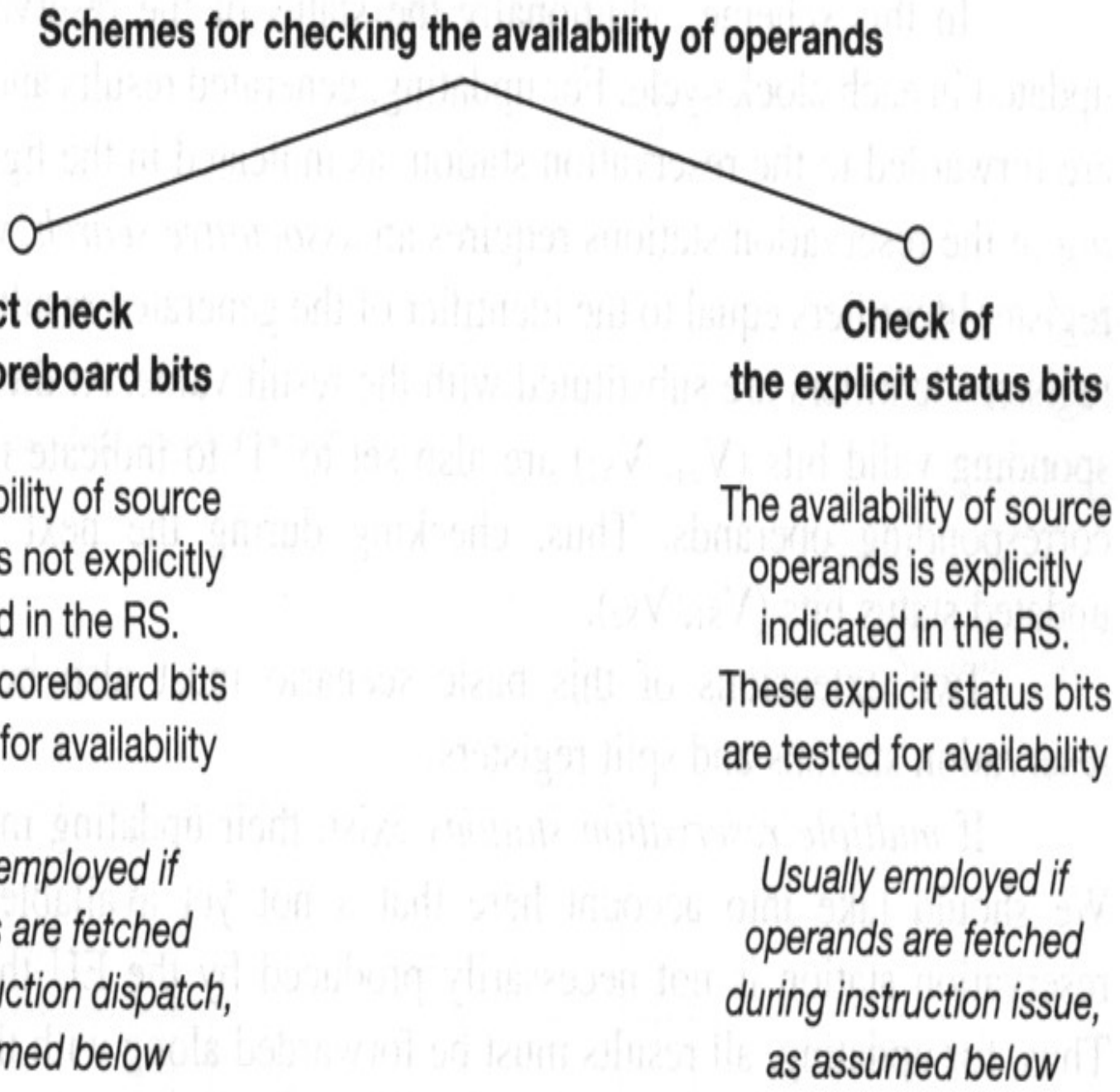


Interpretation:

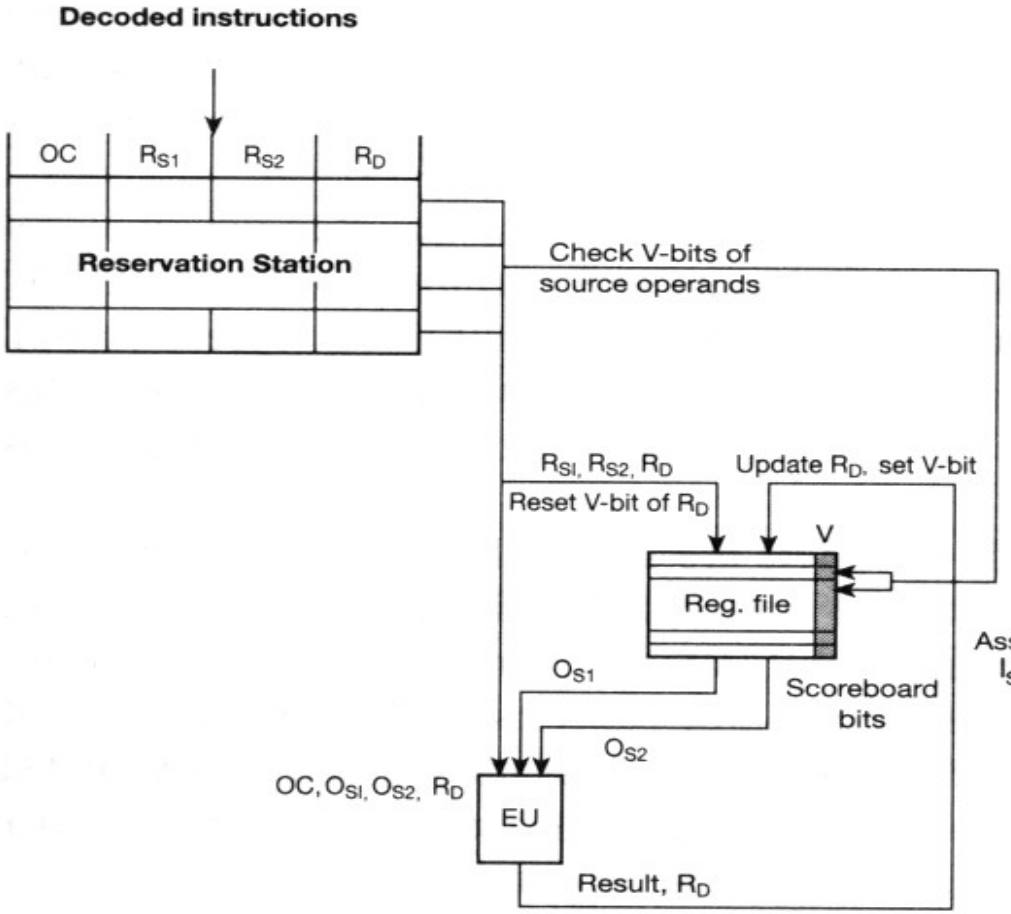
0 ← When an instruction is issued the scoreboard entry corresponding to the destination register is reset to '0'

1 ← When the execution of an instruction is completed, and the result is written into the destination register, the corresponding scoreboard entry is set to '1'

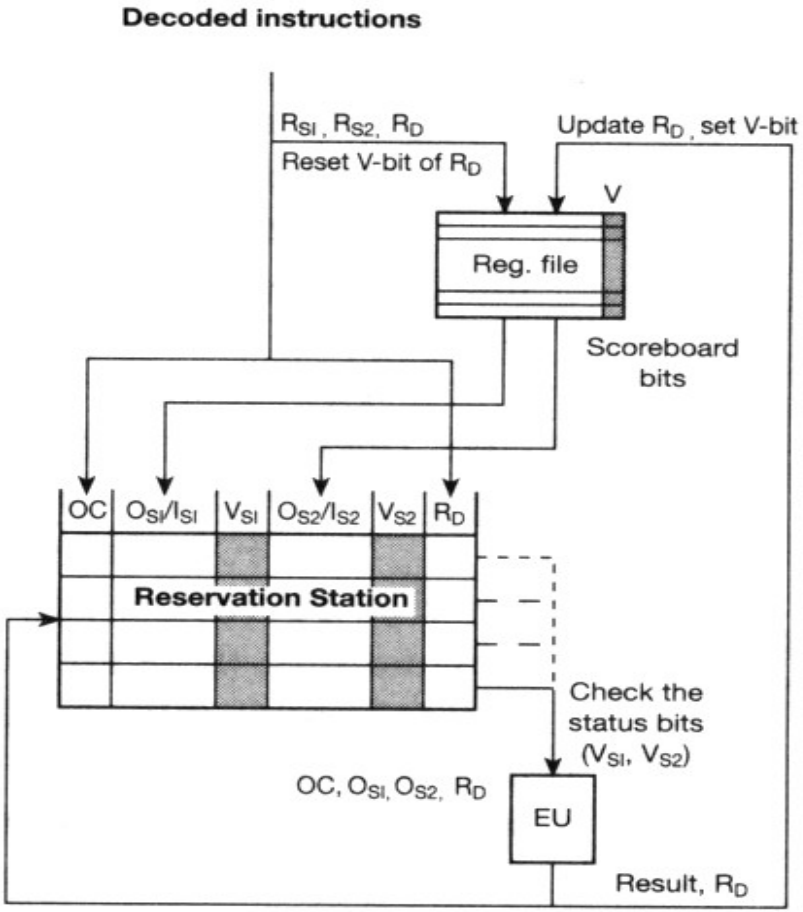
Schemes for checking the availability of operand



Operands fetched during dispatch or during issue

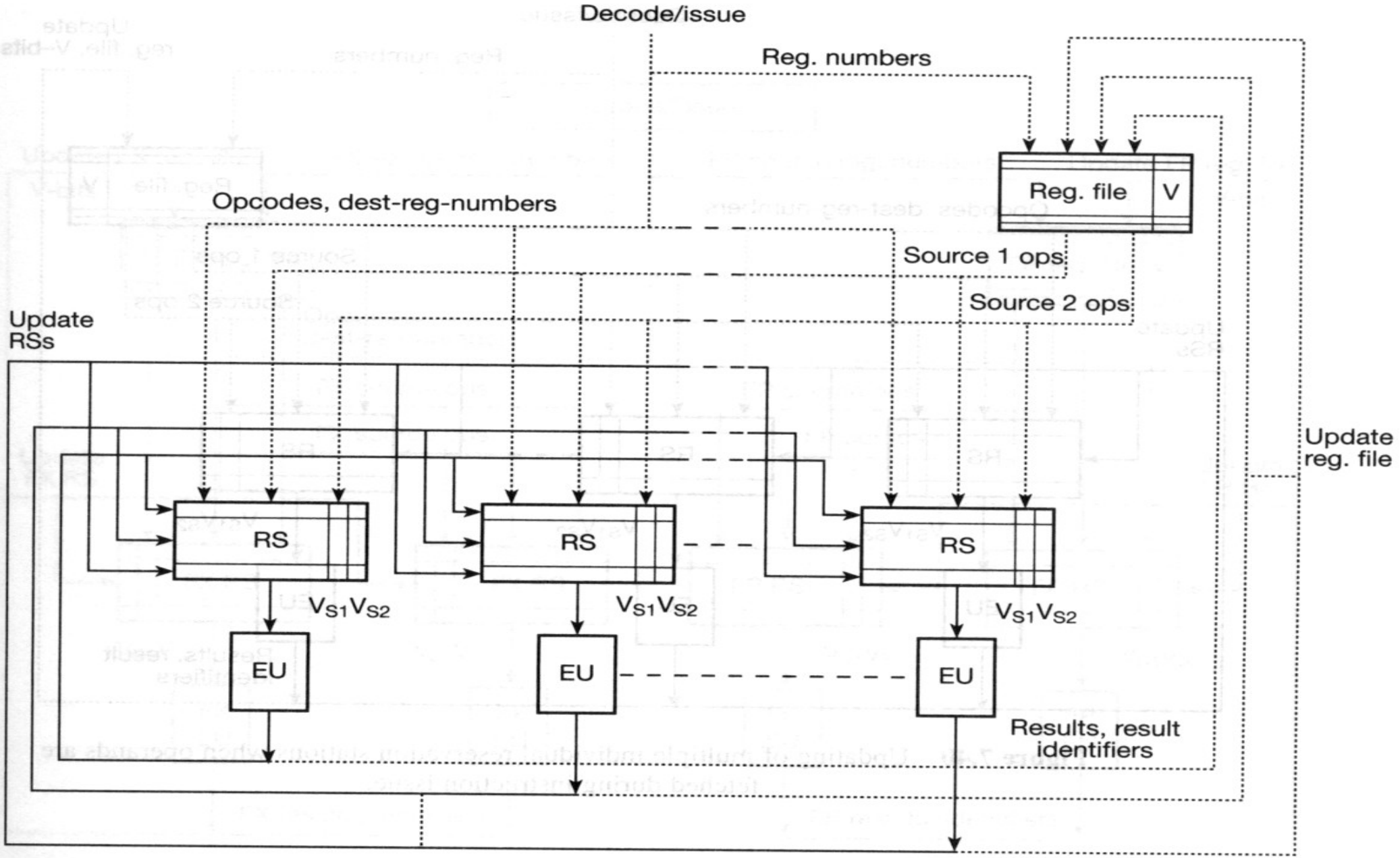


Associative update of I_{SI} I_{S2} with R_{DI} set related V_S–bits

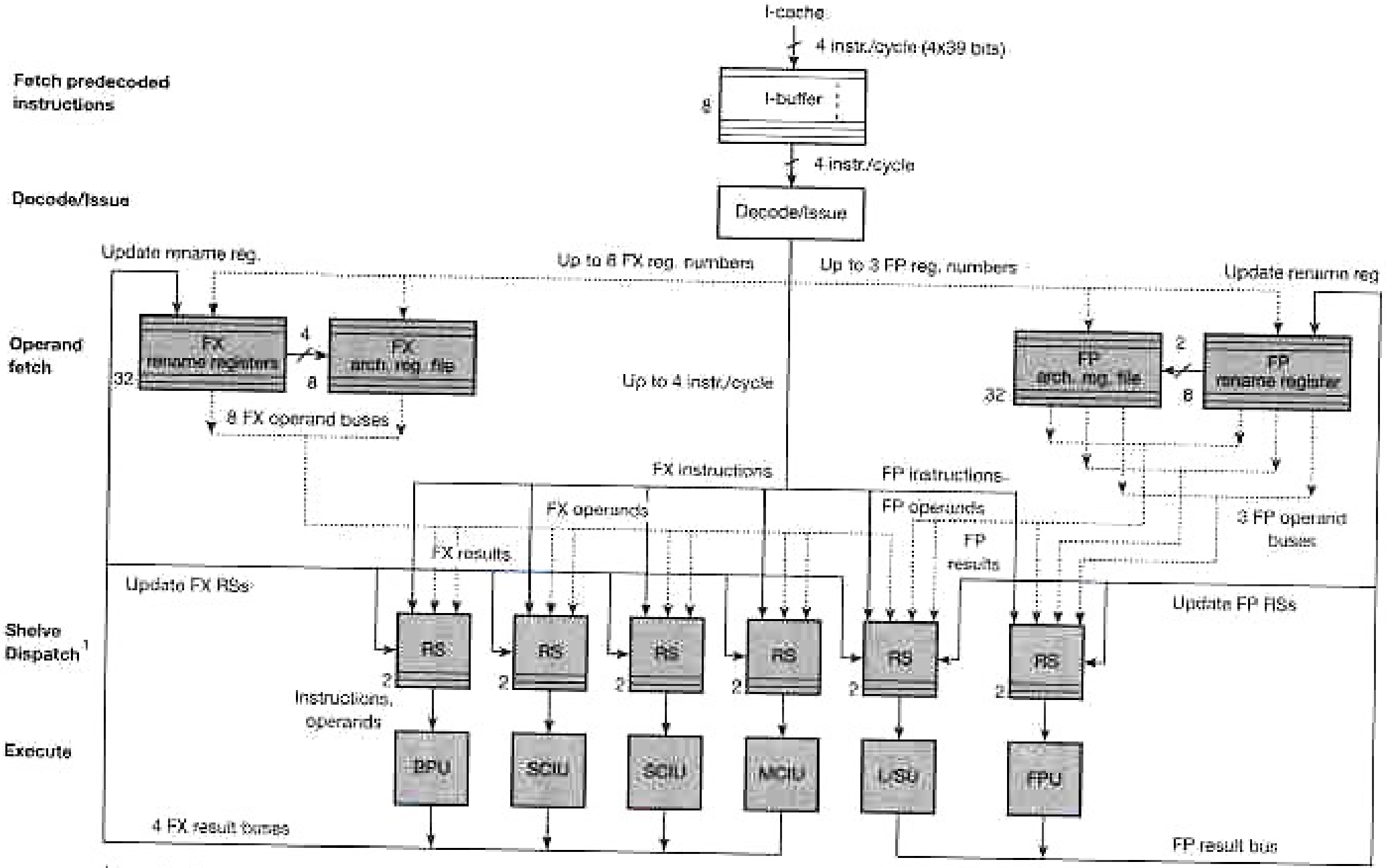


OC:	Operation code
R_{SI}, R_{S2}:	Source register numbers
R_D:	Destination register number
O_{SI}, O_{S2}:	Source operand values
I_{SI}, I_{S2}:	Source operand identifiers (tags)
V_{SI}, V_{S2}:	Source operand valid bits
RS:	Reservation station

Use of multiple buses for updating multiple individual reservation strations



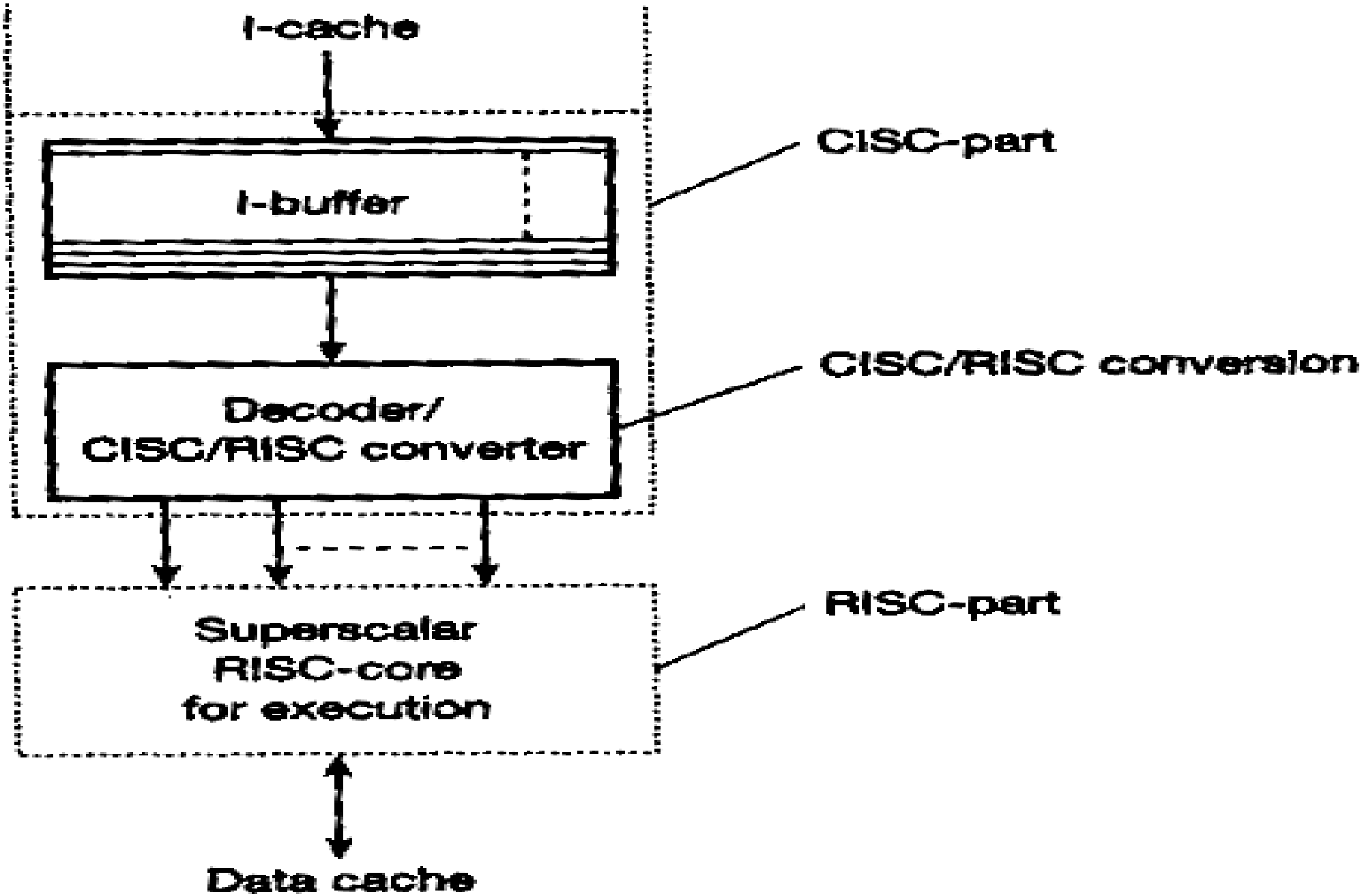
Interal data paths of the powerpc 604



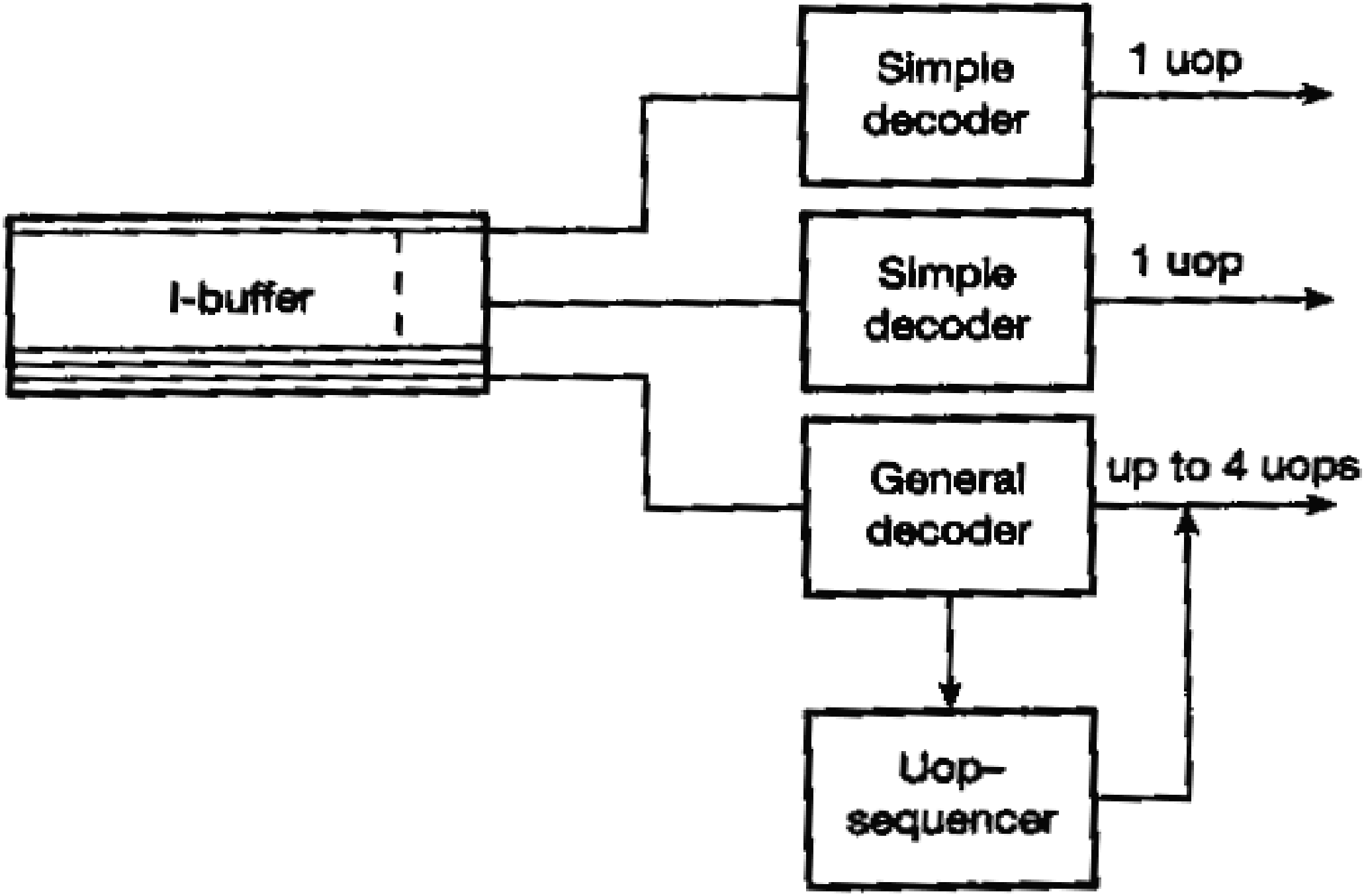
Implementation of superscalar CISC processors using superscalar RISC core

- CISC instructions are first converted into RISC-like instructions <during decoding>.
  - Simple CISC register-to-register instructions are converted to single RISC operation (1-to-1)
  - CISC ALU instructions referring to memory are converted to two or more RISC operations (1-to-(2-4))
    - SUB EAX, [EDI]
      - converted to e.g.
    - MOV EBX, [EDI]
    - SUB EAX, EBX
  - More complex CISC instructions are converted to long sequences of RISC operations (1-to-(more than 4))
- On average one CISC instruction is converted to 1.5-2 RISC operations

The principle of superscalar CISC execution using a superscalar RISC core



PentiumPro: Decoding/converting CISC instructions to RISC operations (are done in program order)



Case Studies: R10000

Core part of the micro-architecture of the R10000