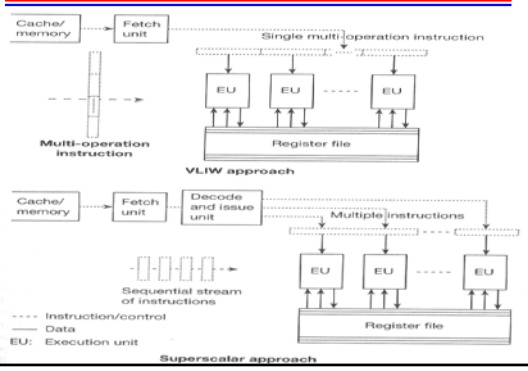
Superscalar Processors

- 7.1 Introduction
- 7.2 Parallel decoding
- 7.3 Superscalar instruction issue
- 7.4 Shelving
- 7.5 Register renaming
- 7.6 Parallel execution
- 7.7 Preserving the sequential consistency of instruction execution
- 7.8 Preserving the sequential consistency of exception processing
- 7.9 Implementation of superscalar CISC processors using a superscalar RISC core
- 7.10 Case studies of superscalar processors

TECH
Computer Science

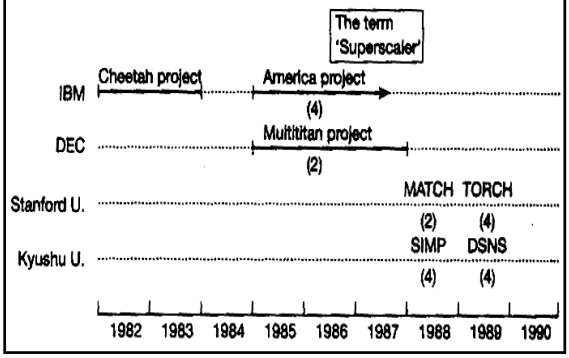
Superscalar Processors vs. VLIW



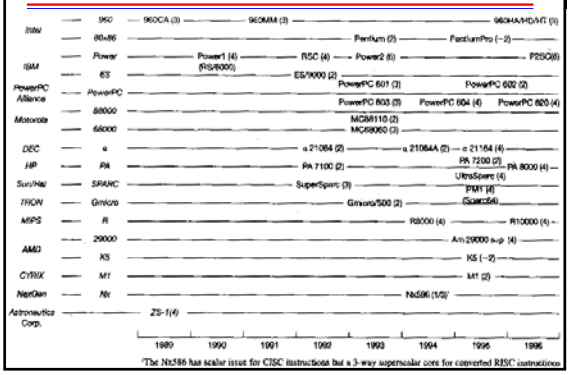
Superscalar Processor: Intro

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

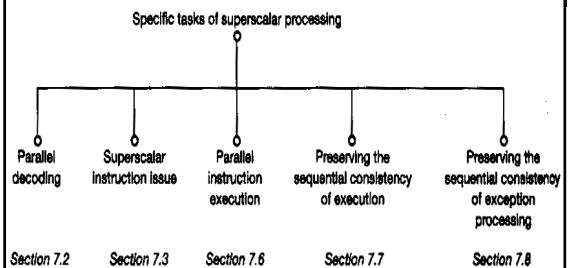
Emergence and spread of superscalar processors



Evolution of superscalar processor

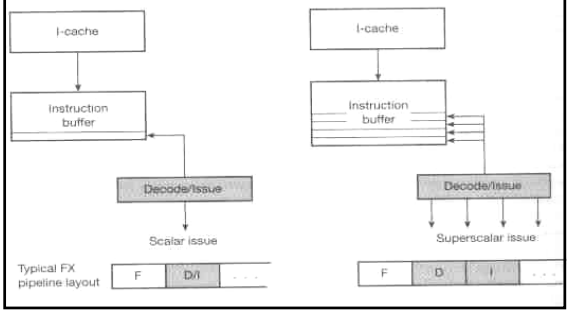


Specific tasks of superscalar processing



Parallel decoding {and Dependencies check}

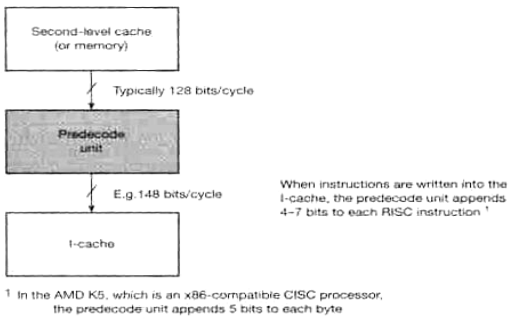
- What need to be done



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 perdecoding



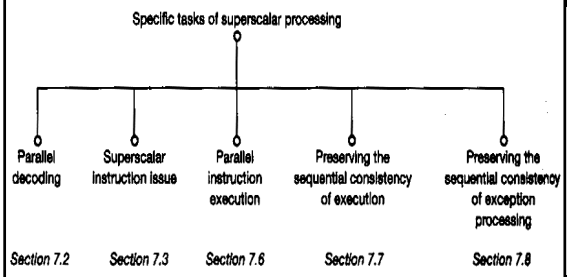
Number of perdecode bits used

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[1]
R10000 (1996)	4

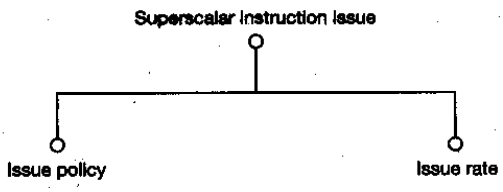
[1] In the K5, 5 predecode bits are added to each byte

Specific tasks of superscalar processing: Issue

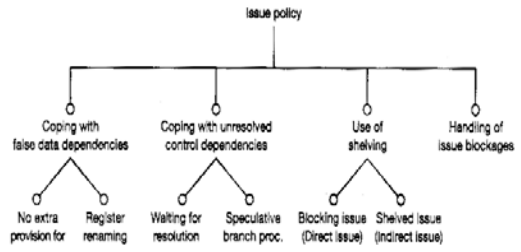


7.3 Superscalar instruction issue

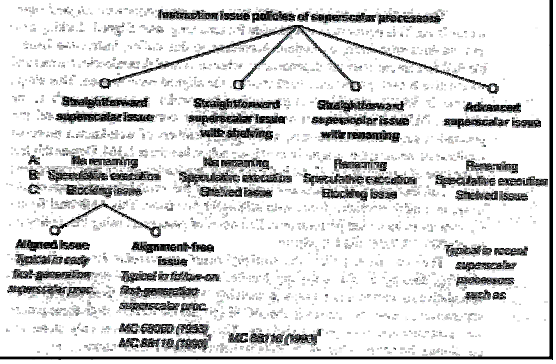
- How and when to send the instruction(s) to EU(s)



Issue policies

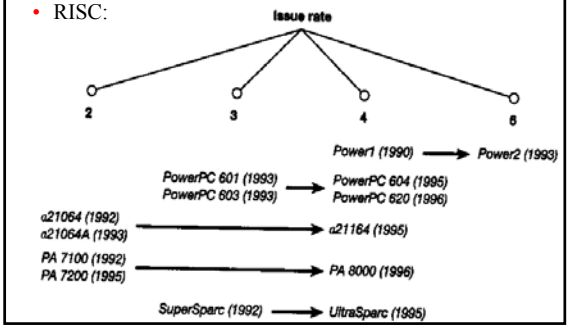


Instruction issue policies of superscalar processors: ---Performance, tread----->

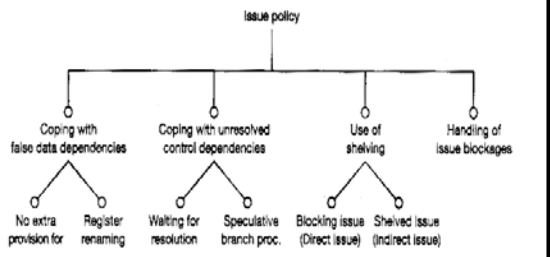


Issue rate {How many instructions/cycle}

- CISC about 2
- RISC:

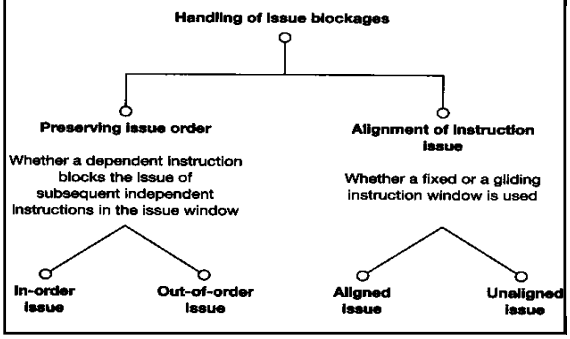


Issue policies: Handing Issue Blockages

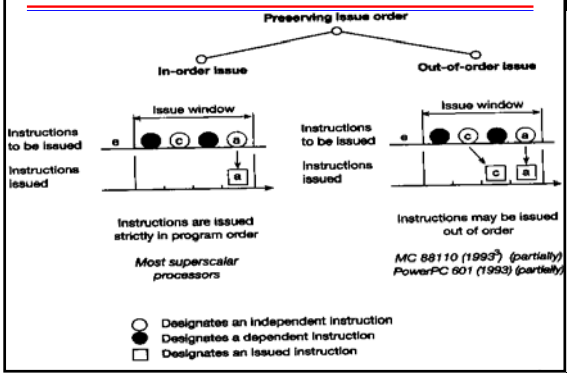


Issue stopped by True dependency

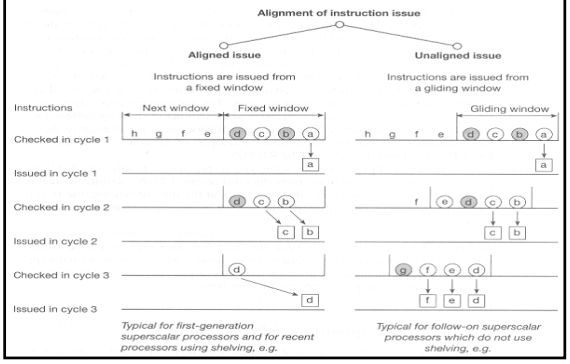
- True dependency → (Blocked: need to wait)



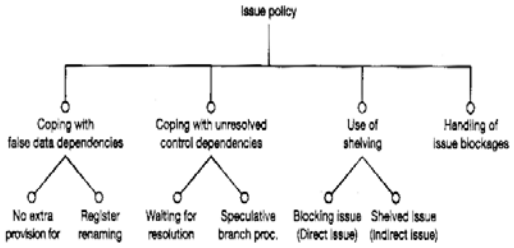
Issue order of instructions



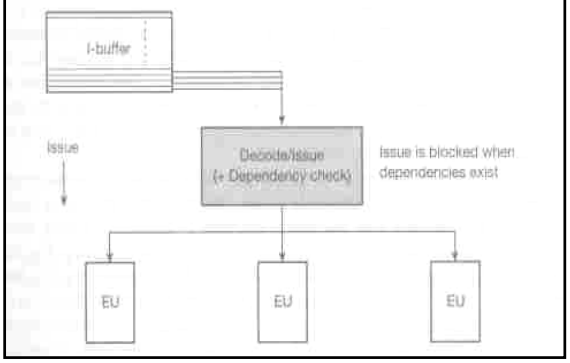
Aligned vs. unaligned issue



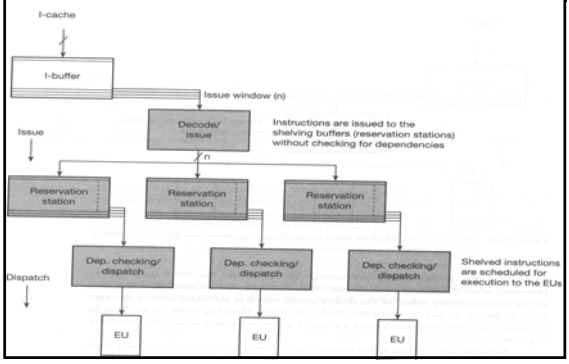
Issue policies: Use of Shelving



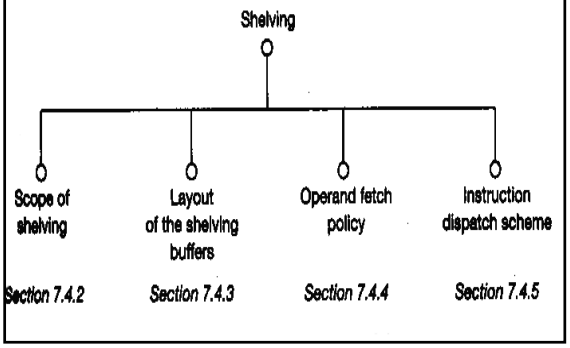
Direct Issue



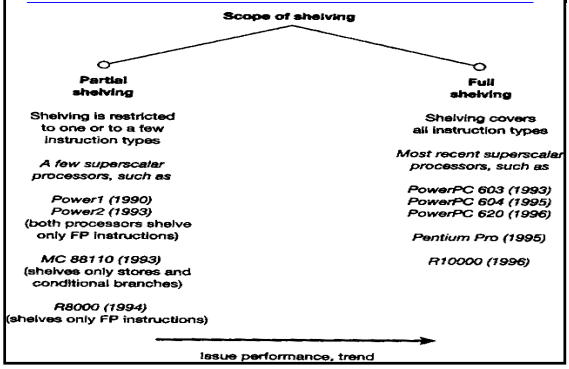
The principle of shelving: Indirect Issue



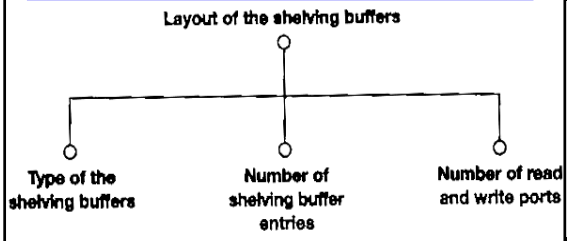
Design space of shelving



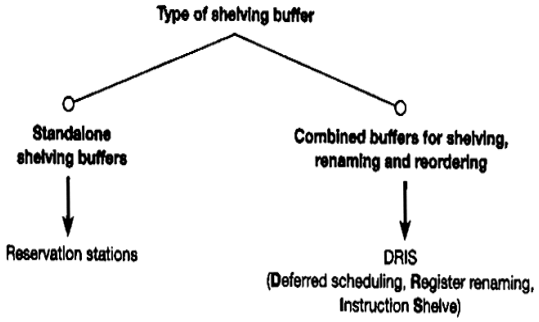
Scope of shelving



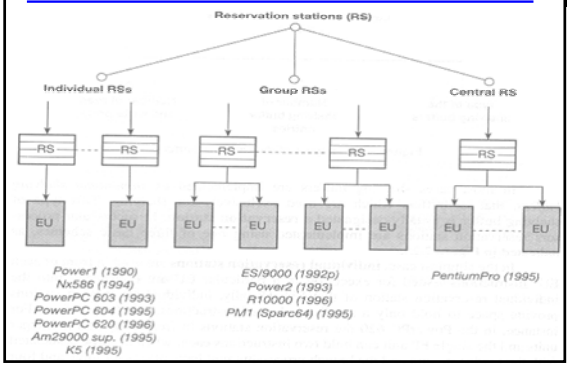
Layout of shelving buffers



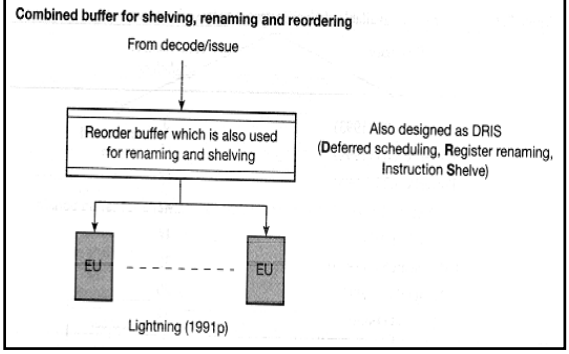
Implementation of shelving buffer



Basic variants of shelving buffers



Using a combined buffer for shelving, renaming, and reordering



Number of shelving buffer entries

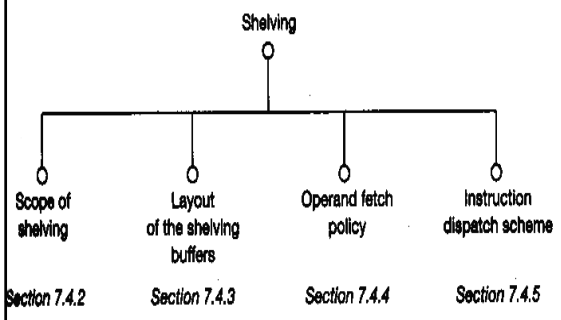
Table 7.2 Comparison of available shelves in recent superscalar processors.

Processor	*Total number of shelves*
PowerPC 603 (1993)	3
PowerPC 604 (1994)	12
PowerPC 620 (1995)	15
Nx586 (1994)	42
K5 (1995)	14
PM1 (Sparc64) (1995)	36
PentiumPro (1995)	20
R10000 (1996)	48

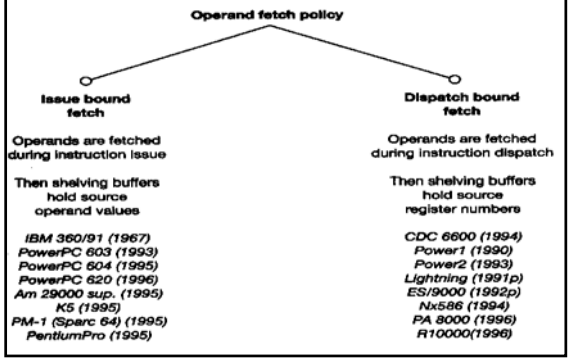
Number of read and write ports

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

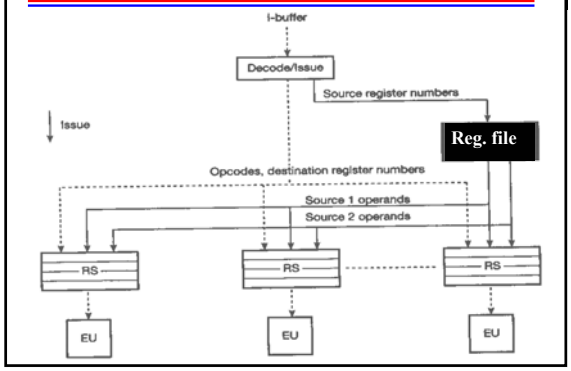
Shelving: Operand fetch policy



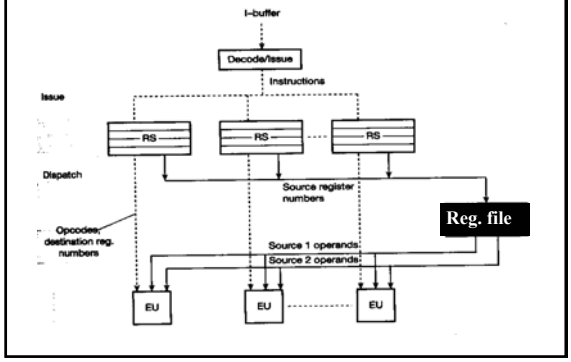
7.4.4 Operand fetch policies



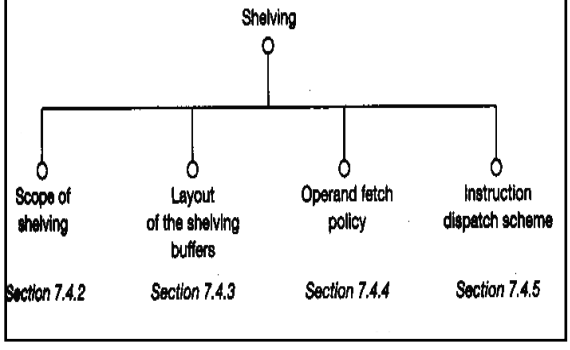
Operand fetch during instruction issue



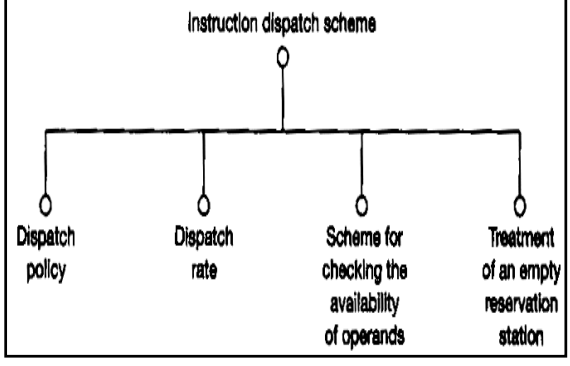
Operand fetch during instruction dispatch



Shelving: Instruction dispatch Scheme //



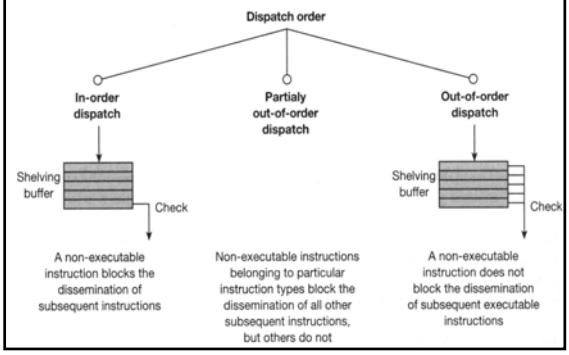
7.4.5 instruction dispatch scheme



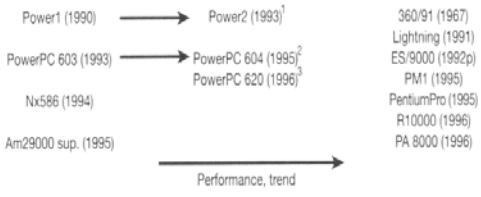
- Dispatch policy

- Selection Rule
  - **Specifies when instructions are considered executable**
  - **e.g. Dataflow principle of operation**
    - Those instructions whose **operands** are available are executable.
- Arbitration Rule
  - **Needed when more instructions are eligible for execution than can be disseminated.**
  - **e.g. choose the 'oldest' instruction.**
- Dispatch order
  - **Determines whether a non-executable instruction prevents all subsequent instructions from being dispatched.**

Dispatch policy: Dispatch order



Trend of Dispatch order

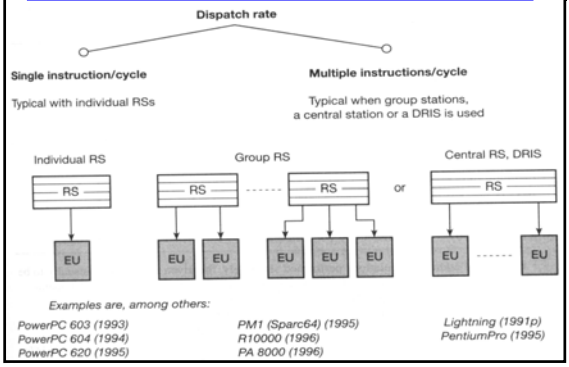


[1] In the Power2, only a single pending (not executable) FP instruction can be skipped.

[2] Out-of-order dispatch from the three integer reservation stations, but in-order dispatch from the Branch, Load/Store and FP reservation stations.

[3] Out-of-order dispatch from the three integer and Load/Store reservation stations, in-order dispatch from the Branch and FP reservation stations.

-Dispatch rate (instructions/cycle)



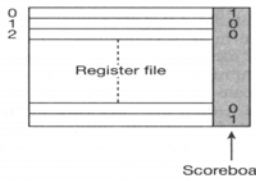
Maximum issue rate <= Maximum dispatch rates
>> issue rate reaches max more often than dispatch rates

Table 7.3 Maximum issue and dispatch rates of superscalar processors with shelving.

Processor/Year of volume shipment	*Maximum issue rate instr./cycle*	*Maximum dispatch rate[1] instr./cycle*
PowerPC 603 (1993)	3	3
PowerPC 604 (1995)	4	6
PowerPC 620 (1996)	4	6
Power2 (1993)	4/6[2]	10
Nx586 (1994)	3/4[3,4]	3/4[3,4]
K5 (1995)	4[4]	5[4]
PentiumPro (1995)	4	5[4]
PM1 (Sparc 64) (1995)	4	8
PA8000 (1996)	4	4
R10000 (1996)	4	5

[1] Because of address calculations performed separately, the given numbers are usually to be interpreted as operations/cycle. For instance, the Power2 performs maximum 10 operations/cycle, which corresponds to 8 instructions/cycle.
[2] The issue rate is 6 for sequential mode and 4 for target mode.
[3] Both rates are 3 without an optional FP unit (labelled Nx587) and 4 with it.
[4] Both rates are related to RISC operations (rather than to the native CISC operations) performed by the superscalar RISC core.

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

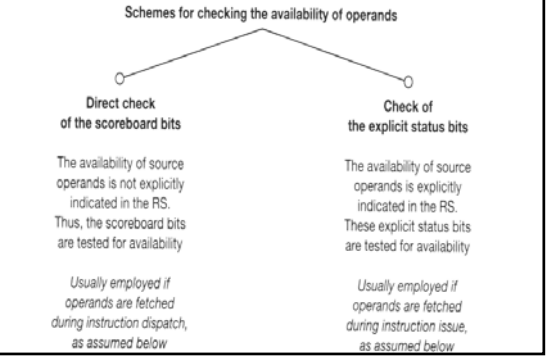


Interpretation:

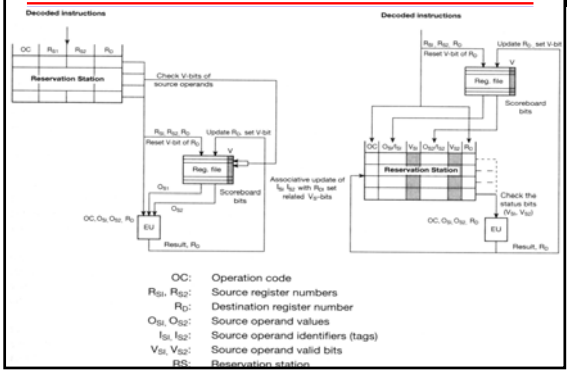
0 ← When an instruction is issued the scoreboard entry corresponding to the destination register is reset to '0'

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'

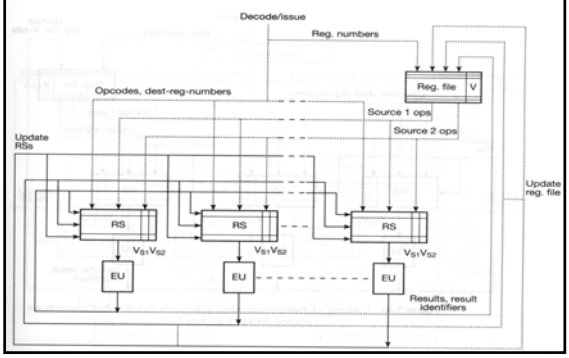
Schemes for checking the availability of operand



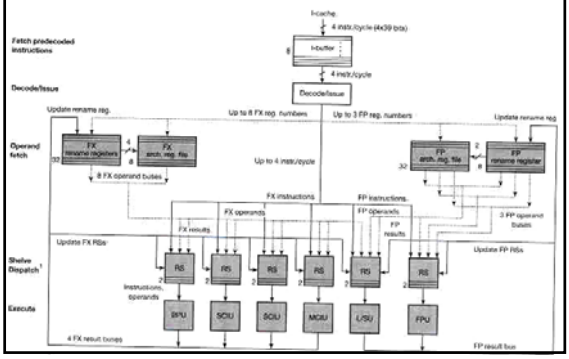
Operands fetched during dispatch or during issue



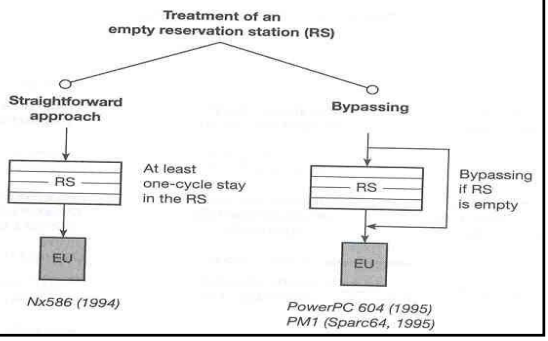
Use of multiple buses for updaing multiple individual reservation strations



Interal data paths of the powerpc 604



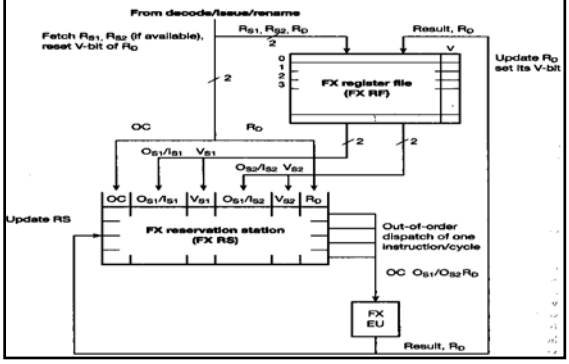
-Treatment of an empty reservation station



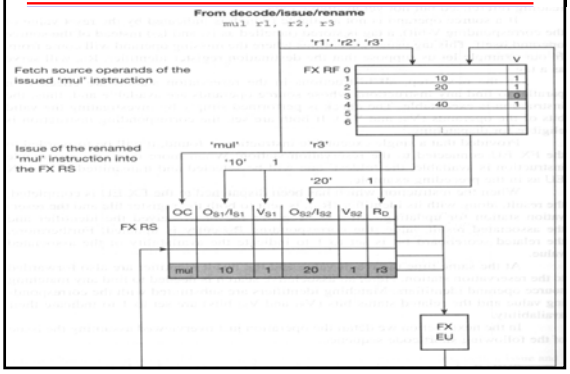
7.4.6 Detail Example of Shelving

- Issuing the following instruction
  - **cycle i: mul r1, r2, r3**
  - **cycle i+1: ad r2, r3, r5**
  - **ad r3, r4, r6**
    - format: Rs1, Rs2, Rd

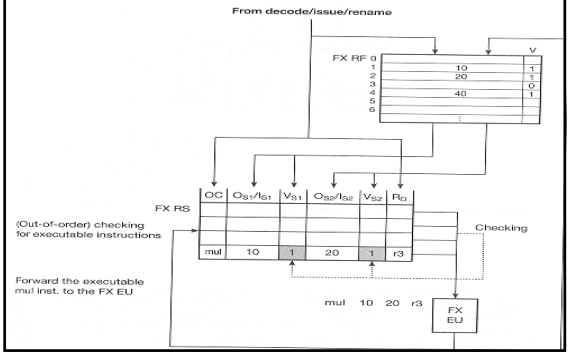
Example overview



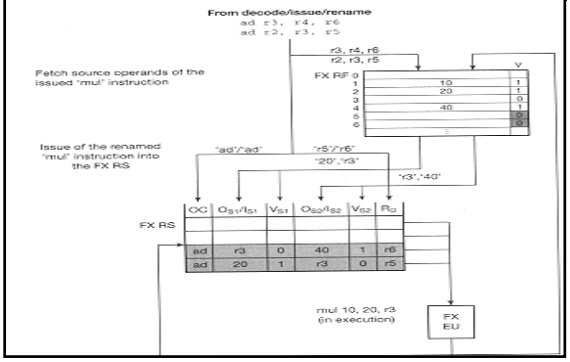
Cycle i: Issue of the 'mul' instruction into the reservation station and fetching of the corresponding operands



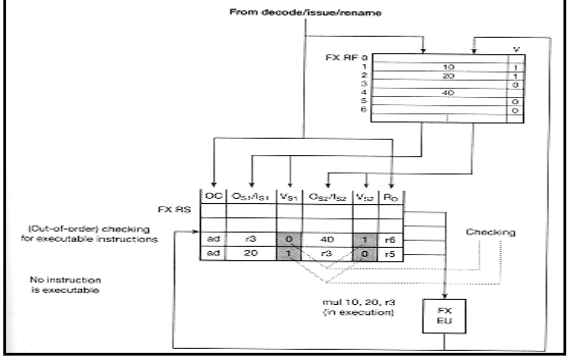
Cycle i+1: Checking for executable instructions and dispatching of the 'mul' instruction



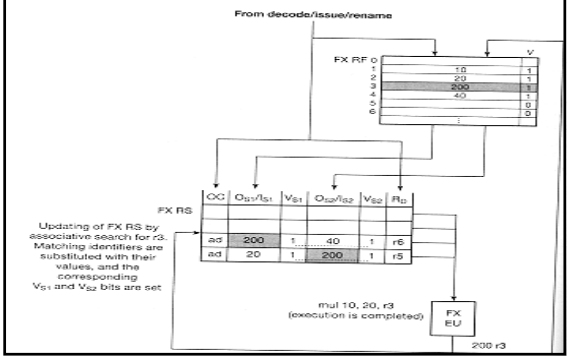
Cycle i+1 (2nd phase): Issue of the subsequent two 'ad' instructions into the reservation station



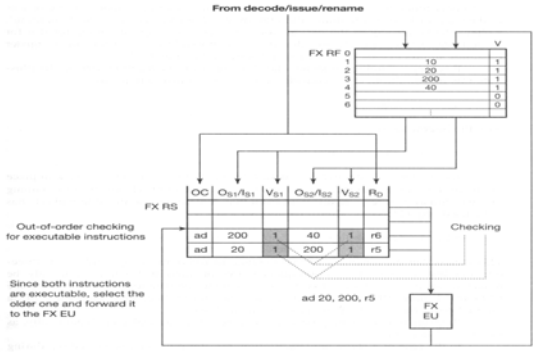
Cycle i+2: Checking for executable instruction (mul not yet completed)



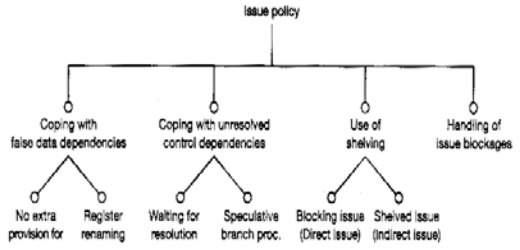
Cycle i+3: Updating the FX register file with the result of the 'mul' instruction



Cycle i+3 (2nd phase): Checking for executable instructions and dispatching the 'older' 'ad' instruction



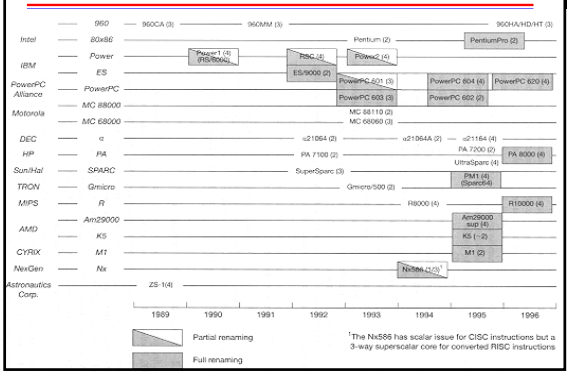
=Instruction Issue policies:Register Renaming



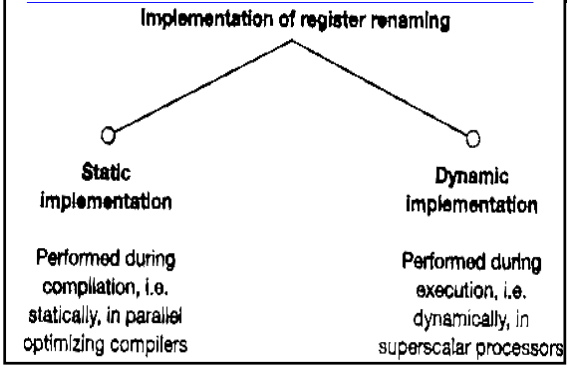
Register Remaining and dependency

- three-operand instruction format
- e.g. Rd, Rs1, Rs2
- False dependency (WAW)
  - **mul r2, …, …**
  - **add r2, …, …**
    - two different rename buffer have to allocated
- True data dependency (RAW)
  - **mul r2, …, …**
  - **ad …, r2, …**
    - rename to e.g.
    - mul p12, …, …
    - ad …, p12, ….

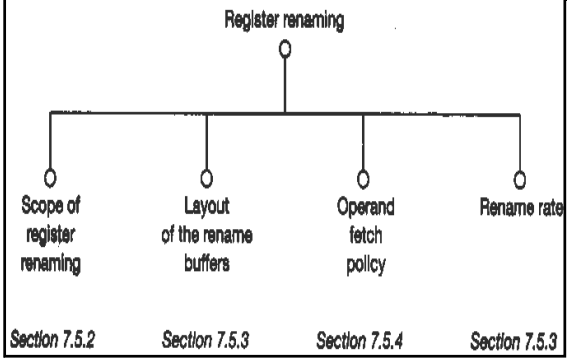
Choronology of introduction of renaming (high complexity, Sparc64 used 371K transistors that is more than i386)



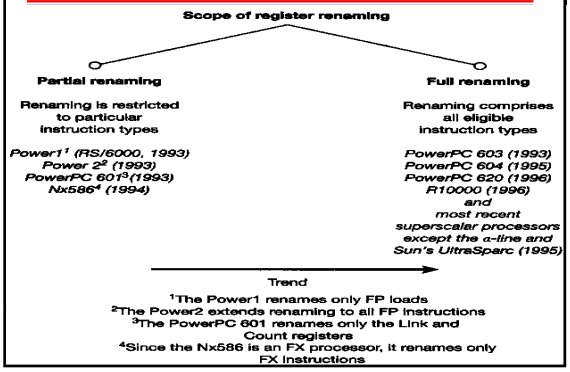
Static or Dynamic Renaming



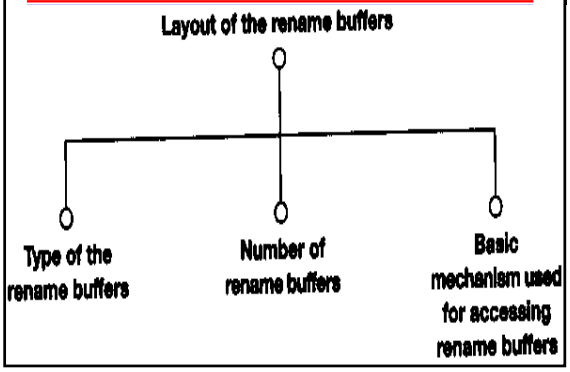
>Design space of register renaming



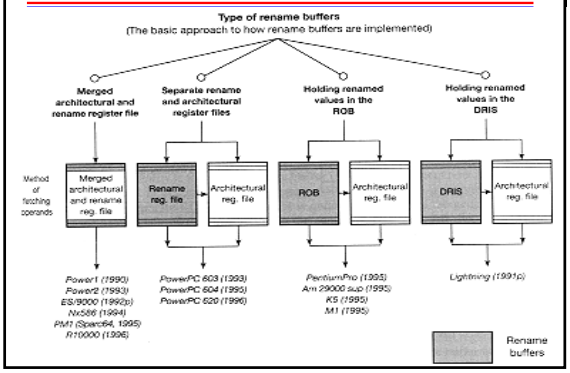
-Scope of register renaming



-Layout of rename buffers



-Type of rename buffers



Rename buffers hold intermediate results

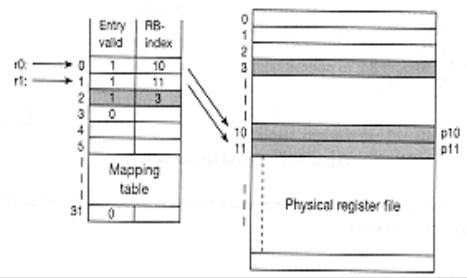
- Each time a Destination register is referred to, a new rename register is allocated to it.
- Final results are stored in the Architectural Register file
- Access both rename buffer and architectural register file to find the latest data,
  - **if found in both, the data content in rename buffer (the intermediate result) is chosen.**
- When an instruction completed (retired),
  - **(ROB) {retire only in strict program sequence}**
  - **the correspond rename buffer entry is writing into the architectural register file (as a result modifying the actual program state)**
  - **the correspond rename buffer entry can be de-allocated**

-Number of rename buffers

Implementation of renaming		*Number of rename buffers*	
Processor type		FX	FP
Merged rename and arch. register file			
Power1	(1990)	–	8 (32 arch. + 8 rename)
Power2	(1993)	–	22 (32 arch. + 22 rename)
ES/9000	(1992p)	16 (16 arch. + 16 ren.)	12 (4 arch. + 12 rename)
PM1	(1995)	38 (78 arch. + 38 ren.)	24 (32 arch. + 24 rename)
R10000	(1996)	32 (32 arch. + 32 ren.)	32 (32 arch. + 32 rename)
Separate rename register file			
PowerPC 603	(1993)	n.a.	4
PowerPC 604	(1995)	12	8
PowerPC 620	(1996)	8	8
Renaming within the ROB			
Am29000 sup	(1995)	10	
K5	(1995)	16	
PentiumPro	(1995)	40	

-Basic mechanisms used for accessing rename buffers

- **Rename buffers with associative access (latter e.g.)**
- **Rename buffers with indexed access**
  - (always corresponds to the most recent instance of renaming)



-Operand fetch policies and Rename Rate

- rename bound: fetch operands during renaming (during instruction issue)
- dispatch bound: fetch operand during dispatching
- Rename Rate
  - **the maximum number of renames per cycle**
  - **equals the issue rate: to avoid bottlenecks.**

7.5.8 Detailed example of renaming

- renaming:
  - **mul r2, r0, r1**
  - **ad r3, r1, r2**
  - **sub r2, r0, r1**
- format:
  - **op Rd, Rs1, Rs2**
- Assume:
  - **separate rename register file,**
  - **associative access, and**
  - **operand fetching during renaming**

Structure of the rename buffers and their supposed initial contents

- **Latest bit: the most recent rename 1, previous 0**

	Entry valid	Dest reg.no.	Value	Value valid	Latest bit
0	1	4	40	1	1
1	1	0	0	1	1
2	1	1	10	1	1
3	0				
4	0				
			Rename registers		

Renaming steps

- Allocation of a free rename register to a destination register
- Accessing valid source register value or a register value that is not yet available
- Re-allocation of destination register
- Updating a particular rename buffer with a computed result
- De-allocation of a rename buffer that is no longer needed.

Allocation of a new rename buffer to destination register (circular buffer: Head and Tail) (before allocation)

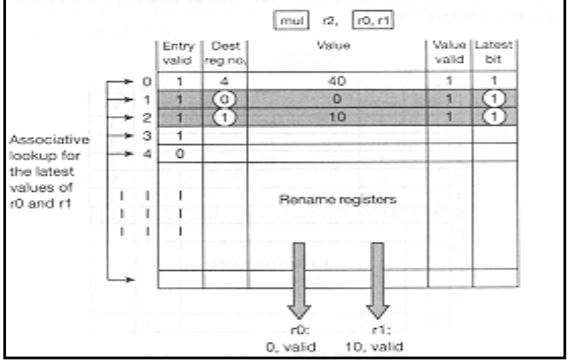
mul r2, r0, r1:

	Entry valid	Dest reg.no.	Value	Value valid	Latest bit
0	1	4	40	1	1
1	1	0	0	1	1
2	1	1	10	1	1
Head → 3	0				
4	0				
			Rename registers		
(i) Tail →	0				

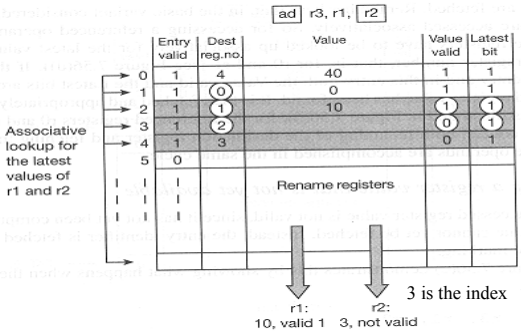
(After allocation) of a destination register

	Entry valid	Dest reg.no.	Value	Value valid	Latest bit
0	1	4	40	1	1
1	1	0	0	1	1
2	1	1	10	1	1
3	(1)	(2)		(0)	(1)
Head → 4	0				
			Rename registers		
(ii) Tail →	0				

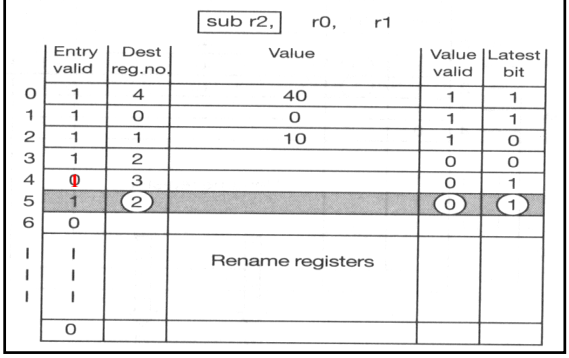
Accessing abailable register values



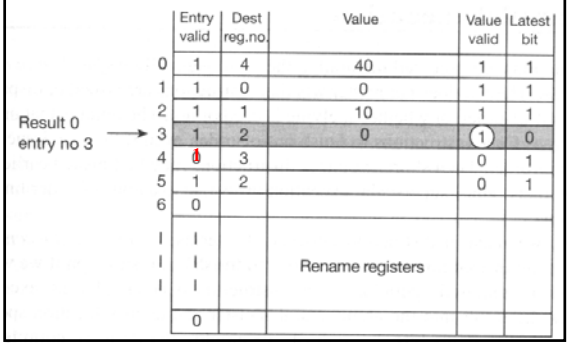
Accessing a register value that is not yet available



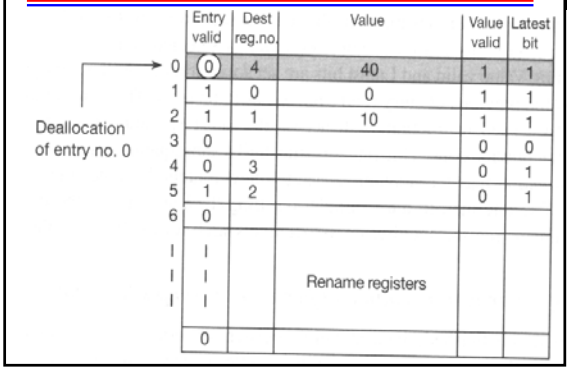
Re-allocate of r2 (a destination register)



Updating the rename buffers with computed result of {mul r2, r0, r1} (register 2 with the result 0)



Deallocation of the rename buffer no. 0 (ROB retires instructions) (update tail pointer)



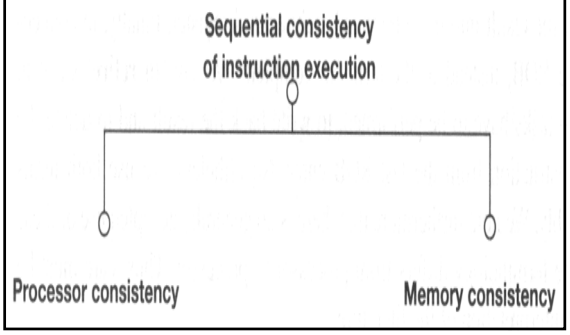
7.6 Parallel Execution

- Executing several instruction in parallel
  - → **instructions will generally be finished in out-of-program order**
- to finish
  - → **operation of the instruction is accomplished,**
  - → **except for writing back the result into**
    - the architectural register or
    - memory location specified, and/or
    - updating the status bits
- to complete
  - → **writing back the results**
- to retire (ROB)
  - → **write back the results, and**
  - → **delete the completed instruction from the last ROB entry**

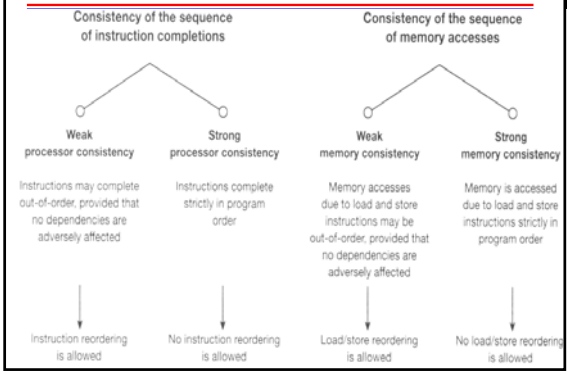
7.7 Preserving Sequential Consistency of instruction execution //

- Multiple EUs operating in parallel, the overall instruction execution should
  >> mimic sequential execution
  - **the order in which instruction are completed**
  - **the order in which memory is accessed**

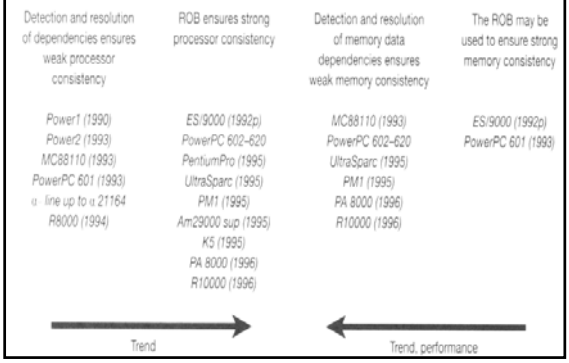
Sequential consistency models



Consistency relate to instruction completions or memory access



Trend and performance



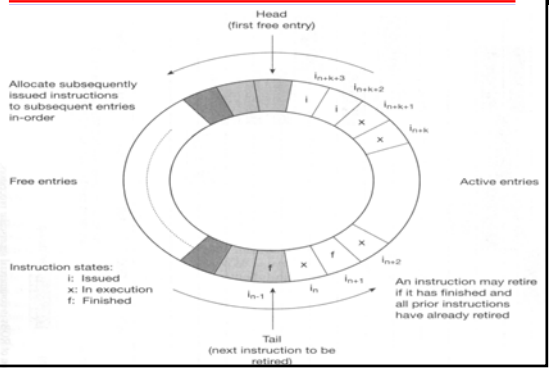
Allows the reordering of memory access

- it permits load/store reordering
  - **either loads can be performed before pending stores, or vice versa**
  - **a load can be performed before pending stores only IF**
  - **none of the preceding stores has the same target address as the load**
- it makes Speculative loads or stores feasible
  - **When addresses of pending stores are not yet available,**
  - **speculative loads avoid delaying memory accesses, perform the load anywhere.**
  - **When store addresses have been computed, they are compared against the addresses of all younger loads.**
  - **Re-load is needed if any hit is found.**
- it allows cache misses to be hidden
  - **if a cache miss, it allows loads to be performed before the missed load; or it allows stores to be performed before the missed store.**

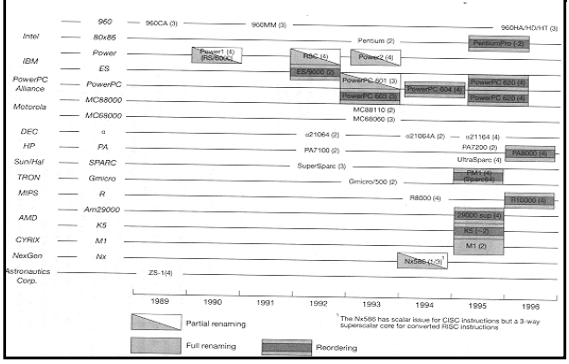
Using Re-Order Buffer (ROB) for Preserving: The order in which instruction are <completed>

- 1. Instruction are written into the ROB in strict program order:
  - **One new entry is allocated for each active instruction**
- 2. Each entry indicates the status of the corresponding instruction
  - **issued (i), in execution (x), already finished (f)**
- 3. An instruction is allowed to retire only if it has finished and all previous instruction are already retired.
  - **retiring in strict program order**
  - **only retiring instructions are permitted to complete, that is, to update the program state:**
    - by writing their result into the referenced architectural register or memory

Principle of the ROB {Circular Buffer}



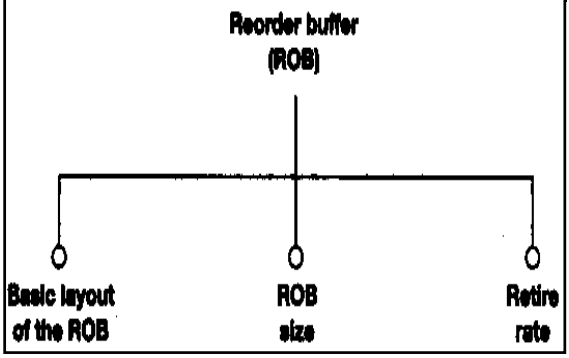
Introduction of ROBs in commercial superscalar processors



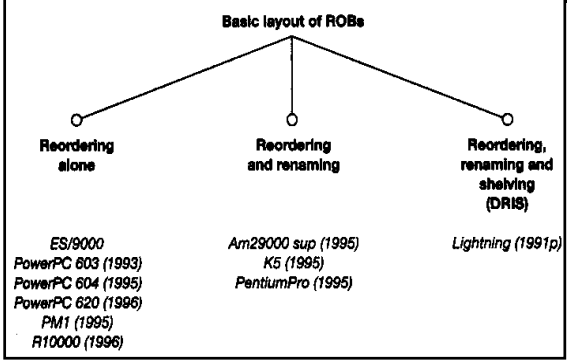
Use ROB for speculative execution

- Guess the outcome of a branch and execution the path
  - → **before the condition is ready**
- 1. Each entry is extended to include a speculative status field
  - → **indicating whether the corresponding instruction has been executed speculatively**
- 2. speculatively executed instruction are not allow to retire
  - → **before the related condition is resolved**
- 3. After the related condition is resolved,
  - → **if the guess turn out to be right, the instruction can retire in order.**
  - → **if the guess is wrong, the speculative instructions are marked to be cancelled.**
    Then, instruction execution continue with the correct instructions.

Design space of ROBs



Basic layout of ROBs



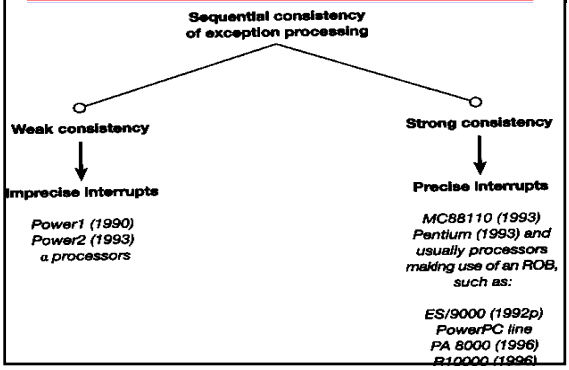
ROB implementation details

	ROB size	*Issue rate*	*Retire rate*	*Intermediate results stored*	*Designation*
ES/9000 (1992p)	32	2	2	No	Completion control logic
PowerPC 602 (1995)	4	2	1	n.a.	Completion unit
PowerPC 603 (1993)	5	3	2	No	Completion buffer
PowerPC 604 (1995)	16	4	4	No	ROB
PowerPC 620 (1996)	16	4	4	No	ROB
PentiumPro (1995)	40	3	3	Yes	ROB
Am29000 sup (1995)	10	4	2	Yes	ROB
K5 (1995)	16	4	4	Yes	ROB
PM1 (Sparc64, 1995)	64	4	4	No	Precise state unit
UltraSparc (1995)	n.a.	4	n.a.	n.a.	Completion unit
PA 8000 (1996)	56	4	4	Yes	Instruction reorder buffer
R10000 (1996)	32	4	4	No	Active list

7.8 Preserving the Sequential consistency of exception processing

- When instructions are executed in parallel,
  - **interrupt request, which are caused by exceptions arising in instruction <execution>,**
  - **are also generated out of order.**
- If the requests are acted upon immediately,
  - **the requests are handled in different order than in a sequential operation processor**
  - **called imprecise interrupts**
- Precise interrupts: handling the interrupts in consistent with the state of a sequential processor

Sequential consistency of exception processing



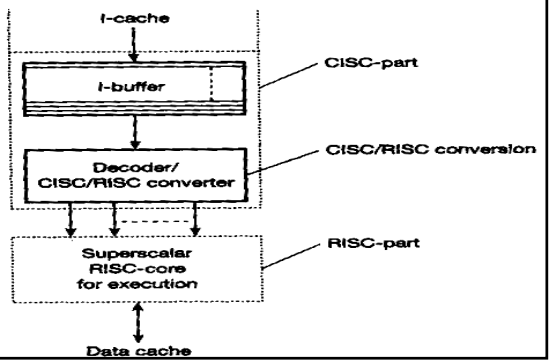
Use ROB for preserving sequential order of interrupt requests

- Interrupts generated in connection with instruction execution
  - **can handled at the correct point in the execution,**
  - **by accepting interrupt requests only when the related instruction becomes the next to retire.**

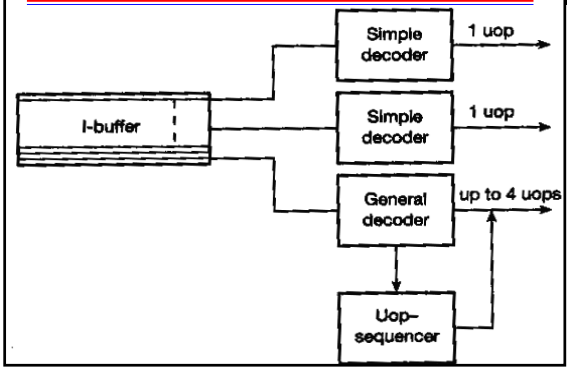
7.9 Implementation of superscalar CISC processors using superscalar RISC core

- 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

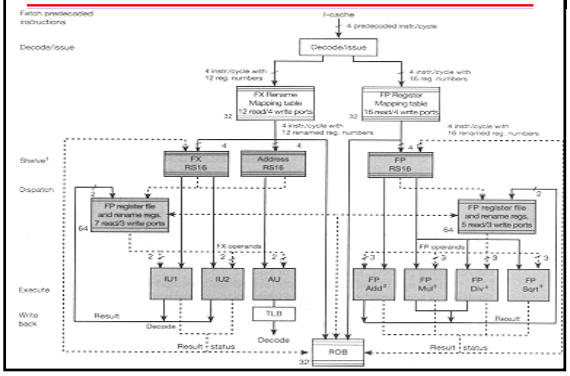
The peinciple of superscalar CISC execution using a superscalar RISC core



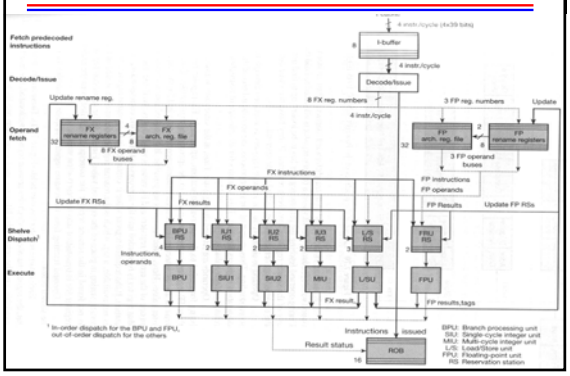
PentiumPro: Decoding/converting CISC instructions to RISC operations (are done in program order)



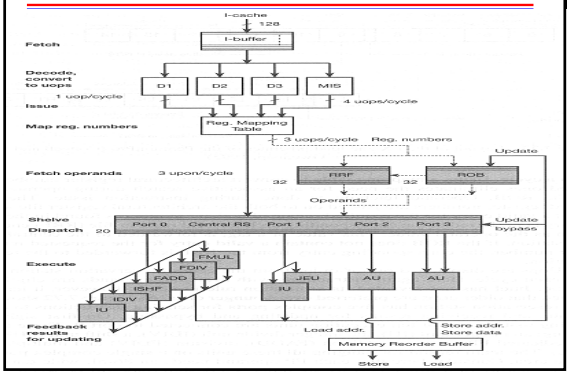
Case Studies: R10000
Core part of the micro-architecture of the R10000



Case Studies: PowerPC 620



Case Studies: PentiumPro
Core part of the micro-architecture



PentiumPro Long pipeline:
Layout of the FX and load pipelines