Computer Architecture

Prof. Dr. Nizamettin AYDIN

naydin@yildiz.edu.tr
nizamettinaydin@gmail.com

http://www.yildiz.edu.tr/~naydin

1

Computer Architecture

Instruction Level Parallelism and Superscalar Processors

2

Outline

- What is Superscalar?
- Superpipelining
- Limitations
  - Data Dependency
  - Procedural Dependency
  - Resource Conflict
- Effect of Dependencies
- Instruction level parallelizm and machine level parallelism
- Instruction Issue Policy
- Antidependency
- Register Renaming
- Machine Parallelism

3

What is Superscalar?

- A superscalar implementation of a processor architecture is one in which...
  - common instructions (arithmetic, load/store, conditional branch) can be
    - initiated simultaneously and
    - executed independently
- Superscalar approach can be equally applicable to RISC & CISC
- In practice usually RISC

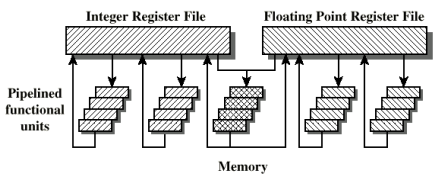
4

Why Superscalar?

- The term superscalar refers to a machine that is designed to improve the performance of the execution of scalar instructions
- In most applications, most operations are on scalar quantities
- Improve these operations to get an overall improvement
- Essence of the superscalar approach is the ability to execute instructions independently in different pipelines

5

General Superscalar Organization - Superpipelining

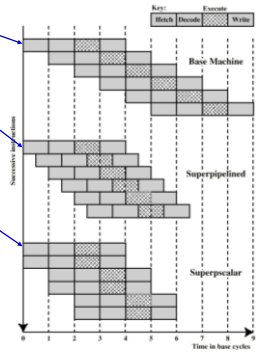


- Superpipelining is an alternative approach to achieving greater performance
- It exploits the fact that many pipeline stages need less than half a clock cycle
- Double internal clock speed gets two tasks per external clock cycle
- Superscalar allows parallel fetch execute

6

Superscalar vSuperpipeline

- Ordinary pipeline
- Superpipelined approach:
  - 2 pipeline stages per clock cycle
- Superscalar implementation
  - Executing two instances of each stage in parallel
- Higher-degree superpipeline and superscalar implementations are possible



7

Limitations

- Superscalar approach depends on the ability to execute multiple instructions in parallel
- Instruction level parallelism refers to the degree to which, on average, the instructions of a program can be executed in parallel
- To maximize instruction-level parallelism:
  - Compiler based optimisation
  - Hardware techniques
- Fundamental limitations to parallelism:
  - True data dependency
  - Procedural dependency
  - Resource conflicts
  - Output dependency
  - Antidependency

8

True Data Dependency

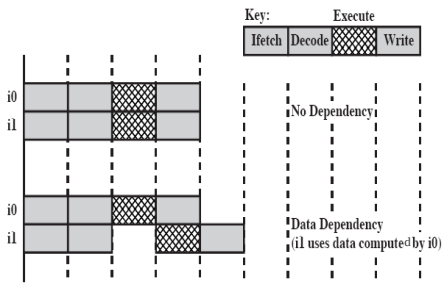
- A true data dependency occurs when an instruction depends on the result of a previous instruction
- Consider the following sequence:

  ADD r1, r2 (r1 := r1+ r2;)

  MOVE r3,r1 (r3 := r1;)
- Can fetch and decode second instruction in parallel with first
  - However, can NOT execute the second instruction until the first one is finished
- Also called flow dependency or write-read dependency

9

True Data Dependency



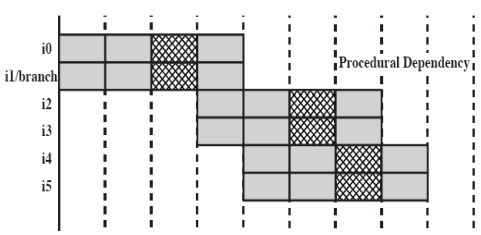
10

Procedural Dependency

- The presence of branches in an instruction sequence complicates the pipeline operation
- The instruction following a branch
  - have a procedural dependency on the branch and
  - can not be executed until the branch is executed
- Also, if instruction length is not fixed, instructions have to be decoded to find out how many fetches are needed
- This prevents simultaneous fetches

11

Effect of a branch on a superscalar pipeline of degree 2



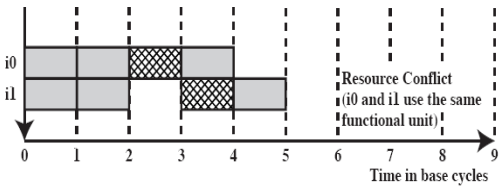
12

Resource Conflict

- A resource conflict is...
  - a competition of two or more instructions requiring access to the same resource at the same time
    - e.g. two arithmetic instructions
- In terms of pipeline it exhibits similar behavior to a data dependency
- However, resource conflict can be overcome by duplication of resources
  - e.g. have two arithmetic units

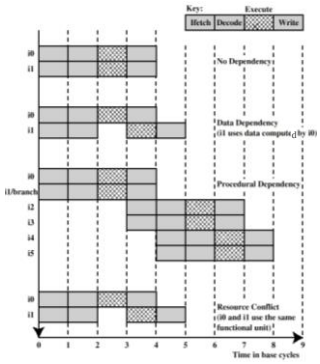
13

Resource Conflict



14

Effect of Dependencies



15

Instruction level parallelizm and machine level parallelizm

- Instruction level parallelism exists when...
  - Instructions in a sequence are independent
- Therefore execution can be overlapped
- Degree of instruction level parallelizm determined by the frequency of true data and procedural dependencies in the code
- For example consider the following codes:

Load	R1 ← R2	Add	R3 ← R3 , "1"
Add	R3 ← R3, "1"	Add	R4 ← R3 , R2
Add	R4 ← R4, R2	Store	[R4] ← R0

- Instructions on the left are independent, so they can be executed in parallel
- Instructions on the right are not independent (data dependency), so they cannot be executed in parallel

16

Instruction level parallelizm and machine level parallelizm

- Machine Parallelism is a measure of the...
  - ability to take advantage of instruction level parallelism
- It is determined by ...
  - the number of instructions that can be fetched and executed at the same time
    - (number of parallel pipelines)
  - the speed and sophistication of the mechanisms that the processor uses to find independent instructions
- Both instruction level and machine level paralellizm are important factors in enhancing performance

17

Instruction Issue Policy

- Instruction issue is the process of initiating instruction execution in the processor's functional units
- Instruction issue policy is the protocol to issue instructions
- The processor is trying to look ahead of the current point of execution to locate instructions that can be brought into the pipeline and executed
- Three types of orderings are important:
  - Order in which instructions are fetched
  - Order in which instructions are executed
  - Order in which instructions change registers and memory

18

Instruction Issue Policy

- In general, instruction issue policies can be devided into the following categories:
  - In-order issue with in-order completion
  - In-order issue with out-of-order completion
  - Out-of-order issue with out-of-order completion

19

In-Order Issue with In-Order Completion

- Issue instructions in the order they occur
- Not very efficient
- May fetch >1 instruction
- Instructions must stall if necessary
- Next slide is an example to this policy
- Assume a superscale pipeline ...
  - capable of fetching and decoding 2 instructions at a time,
  - have 3 seperate functional unit, and
  - two instances of the write-back pipeline stage
- Example assumes the following constraints
  - I1 requires 2 cycles to execute
  - I3 and I4 conflict for the same functional unit
  - I5 depends on the value produced by I4
  - I5 and I6 conflict for a functional unit

20

In-Order Issue with In-Order Completion (Diagram)

Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
I3	I4	I1					3
	I4			I3	I1	I2	4
I5	I6			I4			5
	I6		I5		I3	I4	6
			I6				7
					I5	I6	8

- Instructions are fetched in pairs and passed to the decode unit
- Next 2 instructions must wait until the pair of decode pipeline stages has cleared
- When there is a conflict for a functional unit or when a functional unit requires more than 1 cycle to generate a result, issuing of instructions temporarily stalls

21

In-Order Issue with Out-of-Order Completion

- Used in scalar RISC processors to improve the performance of instructions that require multiple cyles
  - next slide illustrates its use on a superscalar processor
- Output dependency (write-write dependency)
  - I1: R3 ← R3 + R5
  - I2: R4 ← R3 + 1
  - I3: R3 ← R5 + 1
  - I4: R7 ← R3 + R4
  - I2 depends on result of I1 - data dependency
  - If I3 completes before I1, the result from I1 will be wrong

22

In-Order Issue with Out-of-Order Completion (Diagram)

Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
	I4	I1		I3	I2		3
I5	I6			I4	I1	I3	4
	I6		I5		I4		5
			I6		I5		6
					I6		7

- Out-of-order completion requires more complex instruction issue logic than in-order completion
- More difficult to deal with instruction interrupts and exceptions
- When an interrupt happens, instruction execution at the current point is suspended, to be resumed later

23

Out-of-Order Issue with Out-of-Order Completion

- To allow out-of-order issue, it is necessary...
  - to decouple decode pipeline from execution pipeline
    - This is done with a buffer referred to as an instruction window
  - After a processor has finished decoding an instruction, it is placed into instruction window
  - Since instructions have been decoded, processor can look ahead
- Processor can continue to fetch and decode new instructions until this buffer is full
- When a functional unit becomes available in the execution stage, an instruction from the instruction window may be issued to the execute stage
- Any instruction may be issued, provided that...
  - it needs the particular functional unit that is available
  - no conflict or dependencies block this instruction
- Next slide illustrates this policy

24

Out-of-Order Issue Out-of-Order Completion (Diagram)

Decode		Window	Execute			Write		Cycle
I1	I2							1
I3	I4	*I1,I2*	I1	I2				2
I5	I6	*I3,I4*	I1		I3	I2		3
		I4,I5,I6		I6	I4	I1	I3	4
		I5		I5		I4	I6	5
						I5		6

- 2 instructions fetched into the decode stage
- 2 instructions move to the instruction window
- It is possible to issue I6 ahead of I5 (I6 does not depend on I5)
- One cycle is saved
- Window implies that the processor has sufficient information about that instruction to decide when it can be issued

25

Antidependency

- An instruction cannot be issued if it violates a dependency or conflict
- In out-of-order issue with out-of-order completion policy, more instructions are available for issuing reducing the possibility that a pipeline stage will have to stall
- Antidepencency (read-write dependency) arises

  I1: R3 ← R3 + R5
  I2: R4 ← R3 + 1
  I3: R3 ← R5 + 1
  I4: R7 ← R3 + R4

  – I3 cannot complete execution before I2 starts as I2 needs a value in R3 and I3 changes R3
  – It is called antidependency because the constraint is similar to that of true data dependency, but reversed,

26

Register Renaming

- Output and antidependencies occur because register contents may not reflect the correct ordering from the program
- May result in a pipeline stall
- One solution is duplication of resources:
  – called register renaming
- Registers allocated dynamically by the processor hardware, and they are associated with the values needed by instructions at various points in time.
  – i.e. registers are not specifically named

27

Register Renaming example

I1: $R3_b \leftarrow R3_a + R5_a$
I2: $R4_b \leftarrow R3_b + 1$
I3: $R3_c \leftarrow R5_a + 1$
I4: $R7_a \leftarrow R3_c + R4_b$

- Without subscript refers to logical register in instruction
- With subscript is hardware register allocated
- In this example, the creation of $R3_c$ in I3 avoids...
  – the antidependency on the 2nd instruction and
  – output dependency on the 1st instruction,

  and it does not interfere with the correct value being accessed by I4.
- The result is that I3 can be issued until the 1st instruction is complete and the 2nd instruction is issued

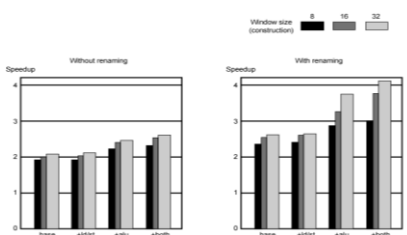
28

Machine Parallelism

- 3 hardware techniques in superscalar processors to enhance performance:
  – Duplication of Resources
  – Out of order issue
  – Renaming
- Next slide shows results of a study, made use of a simulation that modeled a machine with the characteristic of the MIPS R2000, augmented with various superscalar features

29

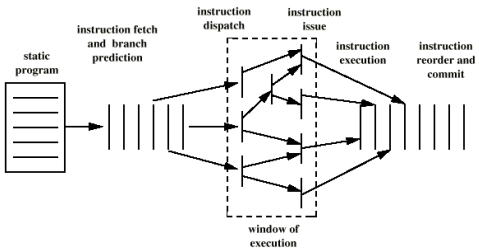
Speedups of Machine Organizations without Procedural Dependencies



- Graphs yield some important conclusions:
  – Not worth duplication functions without register renaming
  – Need instruction window large enough (more than 8)

30

Superscalar Execution



31

Superscalar Implementation

- Simultaneously fetch multiple instructions
- Logic to determine true dependencies involving register values
- Mechanisms to communicate these values
- Mechanisms to initiate multiple instructions in parallel
- Resources for parallel execution of multiple instructions
- Mechanisms for committing process state in correct order

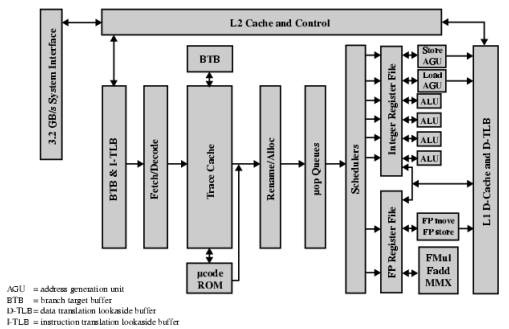
32

Pentium 4

- 80486 - CISC
- Pentium – some superscalar components
  - Two separate integer execution units
- Pentium Pro – Full blown superscalar
- Subsequent models refine & enhance superscalar design

33

Pentium 4 Block Diagram



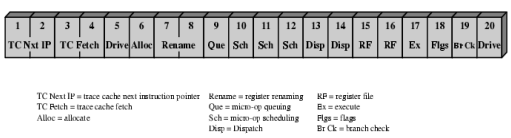
34

Pentium 4 Operation

- Fetch instructions form memory in order of static program
- Translate instruction into one or more fixed length RISC instructions (micro-operations)
- Execute micro-ops on superscalar pipeline
  - micro-ops may be executed out of order
- Commit results of micro-ops to register set in original program flow order
- Outer CISC shell with inner RISC core
- Inner RISC core pipeline at least 20 stages
  - Some micro-ops require multiple execution stages
    - Longer pipeline
  - c.f. five stage pipeline on x86 up to Pentium

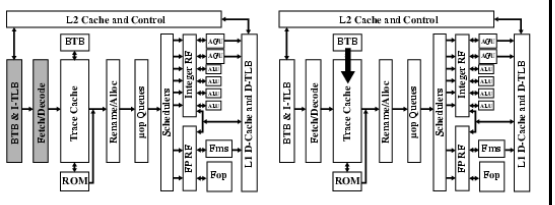
35

Pentium 4 Pipeline



36

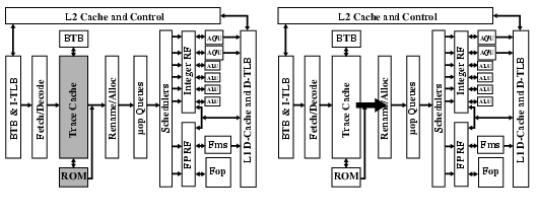
Pentium 4 Pipeline Operation (1)



(a) Generation of micro-ops (b) Trace cache next instruction pointer

37

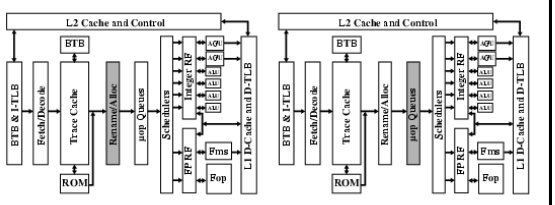
Pentium 4 Pipeline Operation (2)



(c) Trace cache fetch (d) Drive

38

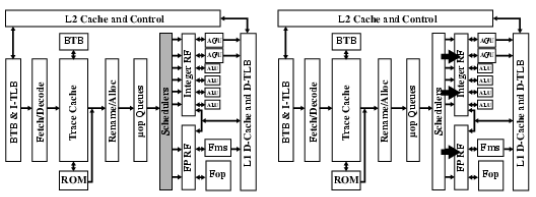
Pentium 4 Pipeline Operation (3)



(e) Allocate; Register renaming (f) Micro-op queuing

39

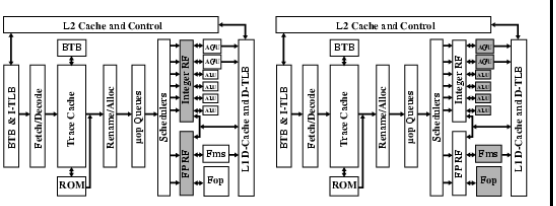
Pentium 4 Pipeline Operation (4)



(g) Micro-op scheduling (h) Dispatch

40

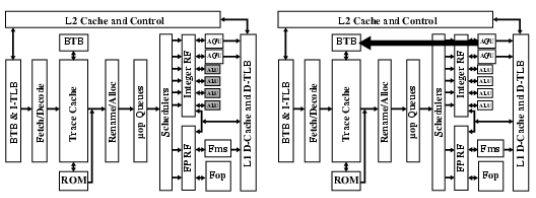
Pentium 4 Pipeline Operation (5)



(i) Register file (j) Execute; flags

41

Pentium 4 Pipeline Operation (6)



(k) Branch check (l) Branch check result

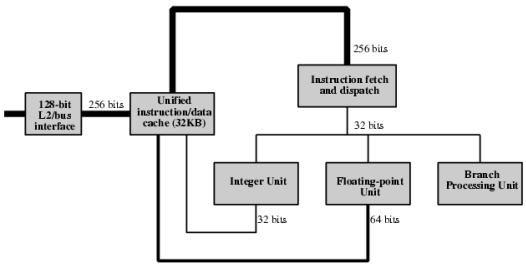
42

PowerPC

- Direct descendent of IBM 801, RT PC and RS/6000
- All are RISC
- RS/6000 first superscalar
- PowerPC 601 superscalar design similar to RS/6000
- Later versions extend superscalar concept

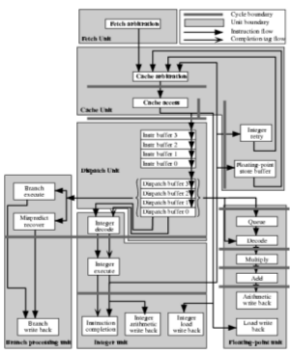
43

PowerPC 601 General View



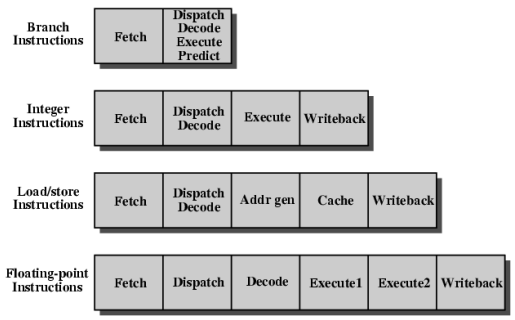
44

PowerPC 601 Pipeline Structure



45

PowerPC 601 Pipeline



46

47