Organisasi Sistem Komputer

OSK 11 – Superscalar

Muh. Izzuddin Mahali, M.Cs.

Pendidikan Teknik Elektronika
FT UNY

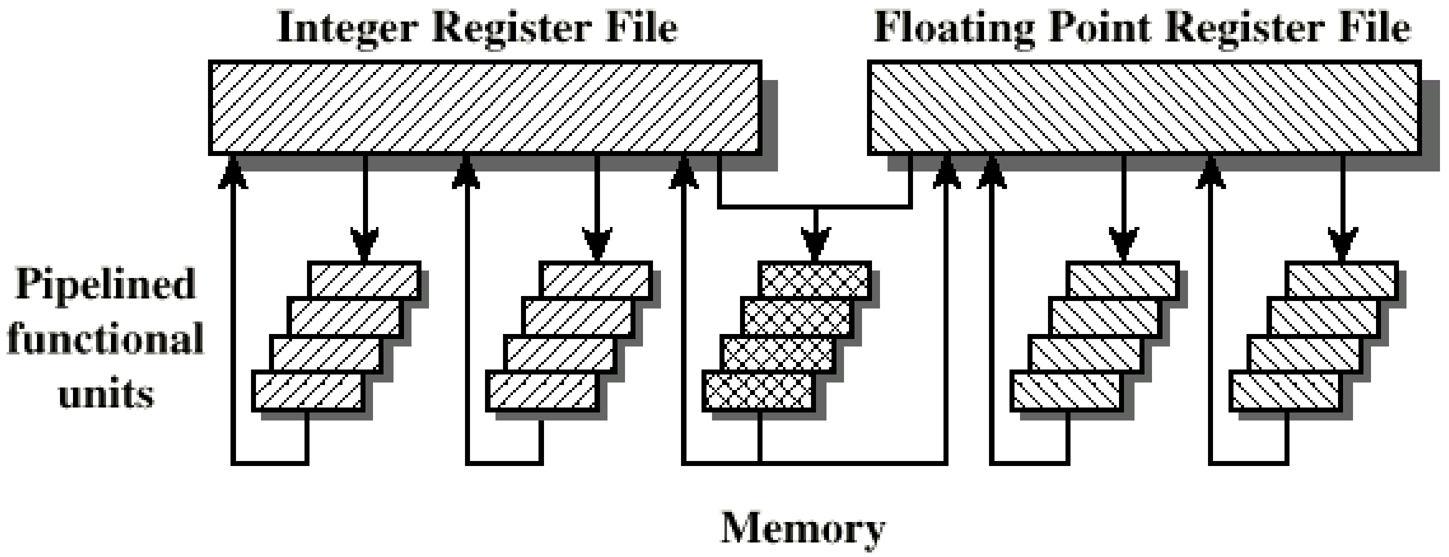
What is Superscalar?

- Common instructions (arithmetic, load/store, conditional branch) can be initiated and executed independently
- Equally applicable to RISC & CISC
- In practice usually RISC

Why Superscalar?

- Most operations are on scalar quantities (see RISC notes)
- Improve these operations to get an overall improvement

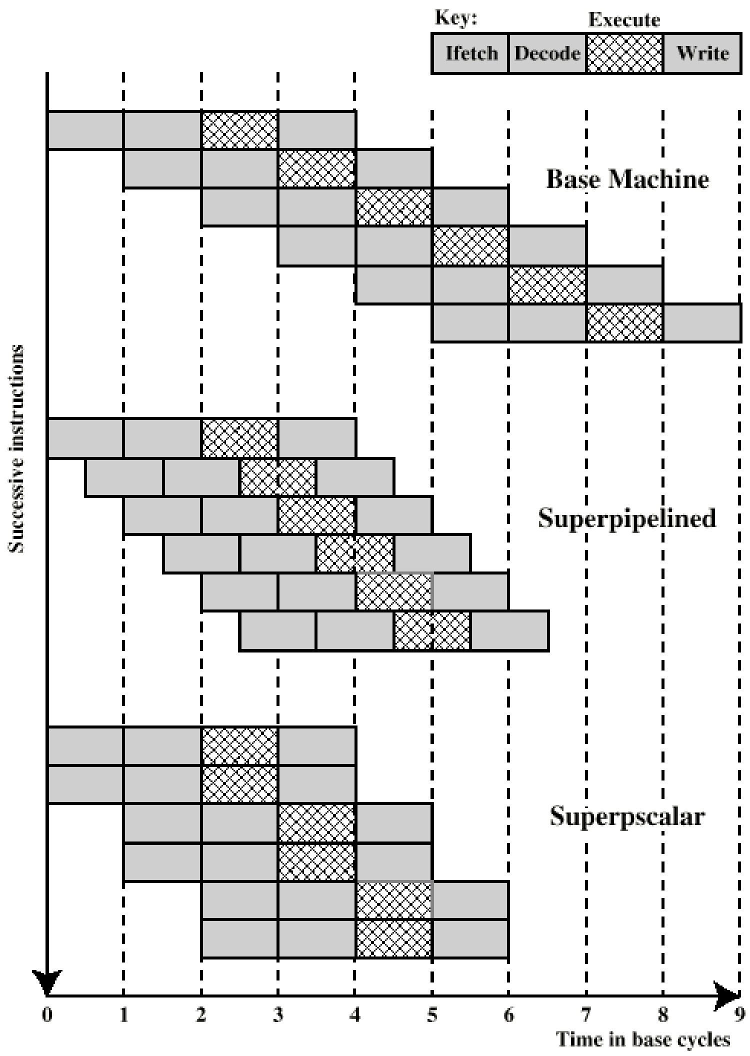
General Superscalar Organization



Superpipelined

- 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

Superscalar v Superpipeline



Limitations

- Instruction level parallelism
- Compiler based optimisation
- Hardware techniques
- Limited by
    - True data dependency
    - Procedural dependency
    - Resource conflicts
    - Output dependency
    - Antidependency

True Data Dependency

- ADD r1, r2 (r1 := r1+r2;)
- MOVE r3,r1 (r3 := r1;)
- Can fetch and decode second instruction in parallel with first
- Can NOT execute second instruction until first is finished

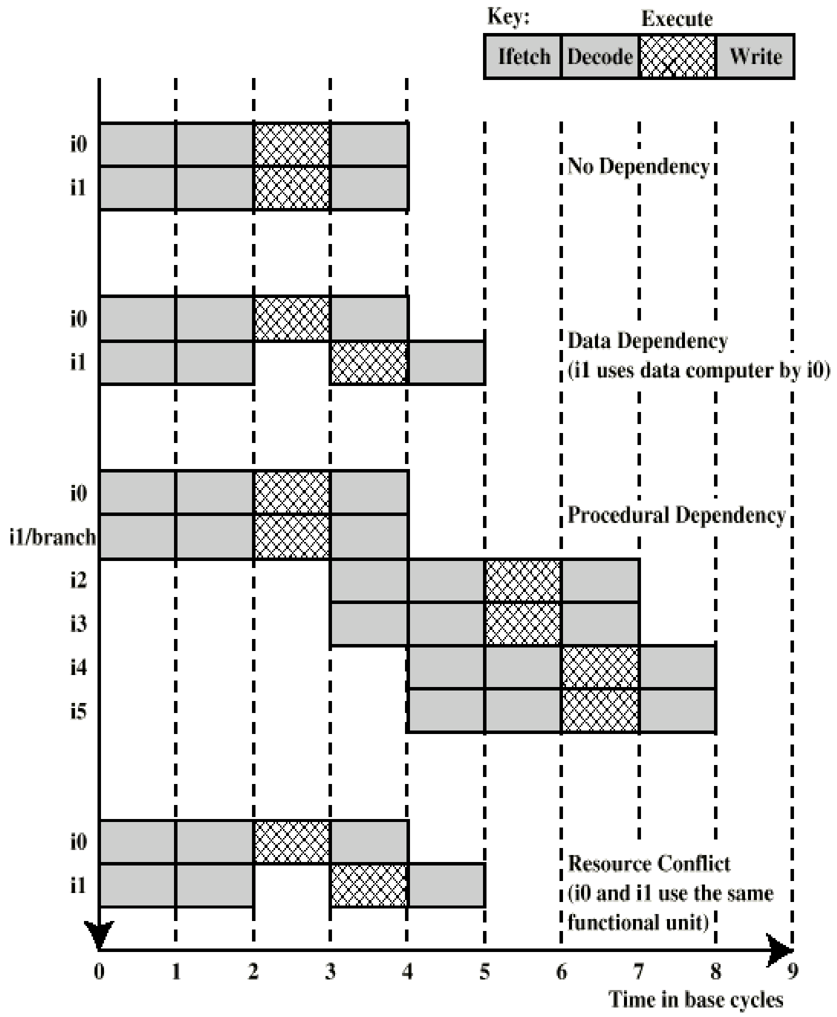
Procedural Dependency

- Can not execute instructions after a branch in parallel with instructions before a branch
- Also, if instruction length is not fixed, instructions have to be decoded to find out how many fetches are needed
- This prevents simultaneous fetches

Resource Conflict

- Two or more instructions requiring access to the same resource at the same time
  - e.g. two arithmetic instructions
- Can duplicate resources
  - e.g. have two arithmetic units

Dependencies



Design Issues

- Instruction level parallelism
  - Instructions in a sequence are independent
  - Execution can be overlapped
  - Governed by data and procedural dependency
- Machine Parallelism
  - Ability to take advantage of instruction level parallelism
  - Governed by number of parallel pipelines

Instruction Issue Policy

- Order in which instructions are fetched
- Order in which instructions are executed
- Order in which instructions change registers and memory

In-Order Issue In-Order Completion

- Issue instructions in the order they occur
- Not very efficient
- May fetch >1 instruction
- Instructions must stall if necessary

In-Order Issue 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

In-Order Issue Out-of-Order Completion

- Output dependency
  - R3:= R3 + R5; (I1)
  - R4:= R3 + 1; (I2)
  - R3:= R5 + 1; (I3)
  - I2 depends on result of I1 - data dependency
  - If I3 completes before I1, the result from I1 will be wrong - output (read-write) dependency

In-Order Issue 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 Issue
Out-of-Order Completion

- Decouple decode pipeline from execution pipeline
- Can continue to fetch and decode until this pipeline is full
- When a functional unit becomes available an instruction can be executed
- Since instructions have been decoded, processor can look ahead

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

Antidependency

- Write-write dependency
  - R3:=R3 + R5; (I1)
  - R4:=R3 + 1;   (I2)
  - R3:=R5 + 1;   (I3)
  - R7:=R3 + R4; (I4)
  - I3 can not complete before I2 starts as I2 needs a value in R3 and I3 changes R3

Register Renaming

- Output and antidependencies occur because register contents may not reflect the correct ordering from the program
- May result in a pipeline stall
- Registers allocated dynamically
  - i.e. registers are not specifically named

Register Renaming example

- R3b:=R3a + R5a (I1)
- R4b:=R3b + 1 (I2)
- R3c:=R5a + 1 (I3)
- R7b:=R3c + R4b (I4)
- Without subscript refers to logical register in instruction
- With subscript is hardware register allocated
- Note R3a R3b R3c

Machine Parallelism

- Duplication of Resources
- Out of order issue
- Renaming
- Not worth duplication functions without register renaming
- Need instruction window large enough (more than 8)

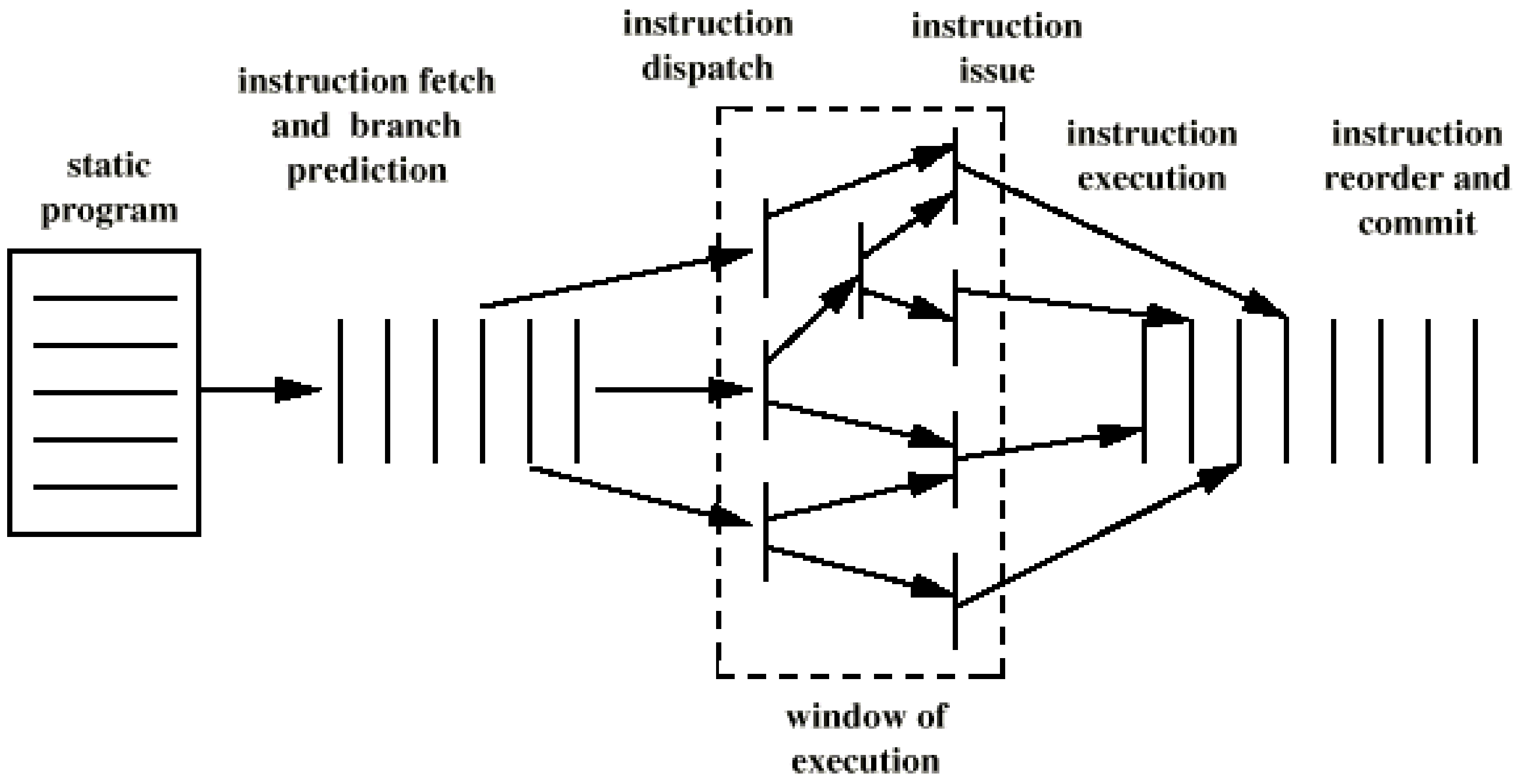
Branch Prediction

- 80486 fetches both next sequential instruction after branch and branch target instruction
- Gives two cycle delay if branch taken

RISC - Delayed Branch

- Calculate result of branch before unusable instructions pre-fetched
- Always execute single instruction immediately following branch
- Keeps pipeline full while fetching new instruction stream
- Not as good for superscalar
  - Multiple instructions need to execute in delay slot
  - Instruction dependence problems
- Revert to branch prediction

Superscalar Execution



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

Required Reading

- Stallings chapter 13
- Manufacturers web sites
- IMPACT web site
  - research on predicated execution