



High Performance Computer Architecture (CS60003)

**Dept. of Computer Science & Engineering
Indian Institute of Technology Kharagpur**

Spring 2023



Superscalar Processors

Instruction Level Parallelism

- Branch Prediction is a mechanism to reduce control hazards in the pipeline.
- However, data dependencies can also prevent us from keeping an ideal CPI of 1.
- Instruction Level Parallelism (ILP) tells us how many instructions can be executed in one clock cycle.
 - On an Ideal processor
 - Assume an instruction can be executed in 1 cycle


An Example Ideal Scenario

Instr	1	2	3	4	5
R1=R2+R3	F	D&R	E	M	WB
R4=R1-R5	F	D&R	E	M	WB
R6=R7 \oplus R8	F	D&R	E	M	WB
R5=R8 \times R9	F	D&R	E	M	WB
R4=R8+R9	F	D&R	E	M	WB

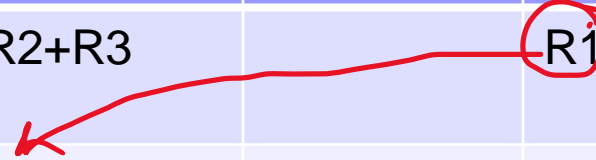
$$CPI_{ideal} = \frac{5}{\infty} \rightarrow 0$$

Real Life Scenario

- We have to respect the dependencies.

$R1 = R2 + R3$

 $R4 = R1 - R5$

	1	2	3	4	5
$R1 = R2 + R3$			$R2 + R3$		$R1$ WB
$R4 = R1 - R5$			$R1 - R5$		

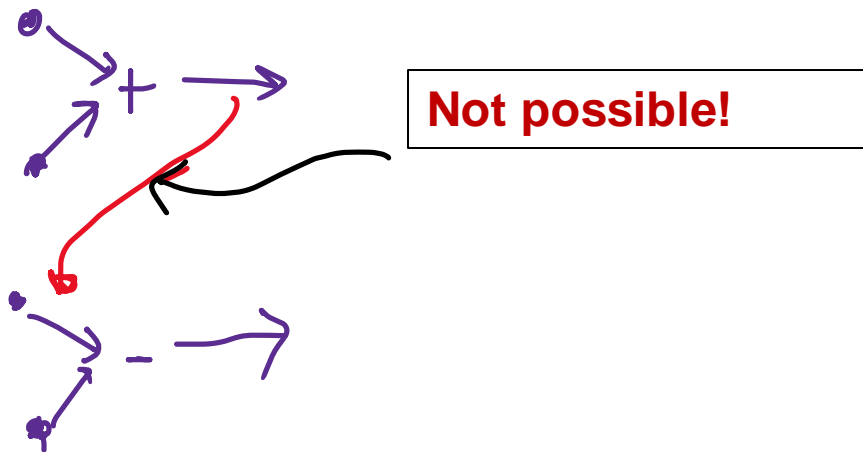


Hence, these 2 instructions cannot execute in the same cycle!

What about Forwarding?

- Forwarding can feed data to the next cycle, and not in the same cycle.
- Lets focus on the execute stage only.

To finish in one cycle we need the following effect of forwarding:



Stalling is Needed

Cycle1	Cycle2
I1	
Stall	I2
I3	
I4	
I5	

Thus, ignoring the time to fill the pipeline,
the ideal CPI is $1/5=0.2$.
But because of the dependency, the effective CPI would be $2/5=0.4$!

Q: Do all kinds of dependencies lead to increase of CPI?


RAW Dependencies

Instructions	Cycle1	Cycle 2	Cycle 3
I1	I1		
I2		I2	
I3	I3		
I4		I4	
I5			I5

$$CPI = 3/5 = 0.6$$

RAW Dependencies

Instructions	Cycle1	Cycle 2	Cycle 3		
I1	I1				
I2		I2			
I3			I3		
I4				I4	
I5					I5



$$CPI = 5/5 = 1.0$$

RAW dependencies affect the CPI

WAW Dependencies

	Ct	Ct+1	Ct+2	Ct+3
R1=R2+R3 <i>RAW</i>	Ex	Mem	WB R1	
R4=R1-R5		Ex	Mem	WB R4 <i>A</i>
R7=... <i>RAW</i>	Ex	Mem	WB R7	
R8=...	Ex	Mem	WB R8	
R4=...	Ex	Mem	WB R4 <i>B</i>	

We want B to be the final result. But now A has the final result!

So, the writing is getting in a different order wrt. program order.

To, prevent B should write in Ct+4.

Dependence Quiz

- 5 stage pipeline
- Forwarding is enabled
- Wider pipeline, can execute 10 instructions in each stage
- Fill up the blanks to mention cycle counts **assuming fetch starts from cycle 0**
(F=0, DR=1, Exe=2, Mem=3, WB=4)

	Exe	WB
MUL R2, R2, R2	2	4
ADD R1, R1, R2	3	5
MUL R3, R3, R3	2	4
ADD R1, R1, R3	4	6
MUL R4, R4, R4	2	4
ADD R1, R1, R4	5	7

This is ok and in accordance with program order!

Removing False Dependencies

- RAW: True Dependencies, instructions that use values that depend on instructions producing these values.
- WAR, WAW: False/Name Dependencies, they are dependencies because we are using same names for different results.

Q: How do we handle False Dependencies?

Strategy 1: Duplicating Register Values

$R1 = R2 + R3$	EXE			
$R4 = R1 - R5$		EXE	MEM	WB R4
$R3 = R4 + R6$			EXE	
$R4 = R8 - R9$	EXE	MEM	WB R4	
...				
$R10 = R4 + R9$				

Final value is coming from $R4 = R1 - R5$, which is not according to program order!

A way to fix is to keep multiple values of a register. In this case, say we duplicate and store all values of R4. When an instruction needs R4, it has to search among all possible values of R4, and choose the immediate one before it.

This is a complicated method and avoided.

Strategy 2: Register Renaming

- Logical or Architectural Registers: ISA defined registers which are visible to programmers or compilers
- Physical Registers: Total register storage where the values can go.
- The logical registers of an instruction will be renamed, or mapped to physical registers.
- The source operands' names are replaced by the physical addresses to which they map.
- The destination register is given a new physical address name.
- Freelist is a list of physical registers that have not been allocated with first pointing to the next one available
- Register Alias Table (RAT): Table that says which physical register has value for which architectural register.
 - The table is indexed by architectural registers and tells which physical address has current values.

RAT Example

ADD R1, R2, R3	
SUB R4, R1, R5	
XOR R6, R7, R8	
MUL R5, R8, R9	
ADD R4, R8, R9	
... R4 (uses R4)	

0	P0
1	P1
2	P2
3	P3
4	P4
5	P5
6	P6
7	P7

RAT Table

RAT Example

Instructions	After Renaming
ADD R1, R2, R3	ADD P20,P2,P3
SUB R4, R1, R5	SUB P21,P20,P5
XOR R6, R7, R8	XOR P22,P7,P8
MUL R5, R8, R9	MUL P23,P8,P9
ADD R4, R8, R9	ADD P24,P8,P9
... R4 (uses R4)	...P24 (uses P24)

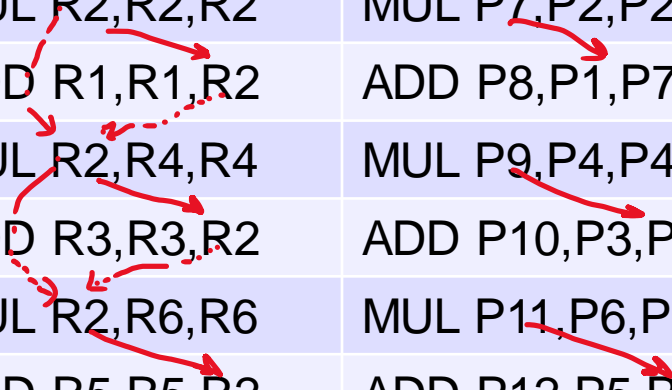
RAT Table	
0	P0
1	P1->P20
2	P2
3	P3
4	P4->P21
5	P5->P23
6	P6->P22
7	P7
...	

RAT Table

There is a name dependency between I2 and I5. However the register renaming eliminates it. Note I2 now writes to P21, and I5 to P24. The next instruction reads from P21, which has the correct value according to program order.

Quiz

Instructions	After Renaming
MUL R2,R2,R2	MUL P7,P2,P2
ADD R1,R1,R2	ADD P8,P1,P7
MUL R2,R4,R4	MUL P9,P4,P4
ADD R3,R3,R2	ADD P10,P3,P9
MUL R2,R6,R6	MUL P11,P6,P6
ADD R5,R5,R2	ADD P12,P5,P11



CPI before renaming=1, IPC=1
After renaming, CPI=2/6=0.33,
IPC=3

Logical Register	Physical Register
R1	P1->P8
R2	P2->P7->P9->P11
R3	P3->P10
R4	P4
R5	P5
R6	P6

Instruction Level Parallelism (ILP)

- ILP can be defined as the IPC on an ideal processor:
 - When the processor does execute entire instruction in 1 cycle.
 - Processor can execute any number of instructions in the same cycle
 - Theoretically can execute the entire program in one cycle
 - But has to obey true dependencies
- ILP is a property of the program and not of any chosen processor
- Steps to obtain ILP:
 - Rename registers
 - Execute to compute the IPC on an hypothetical idea machine

Example

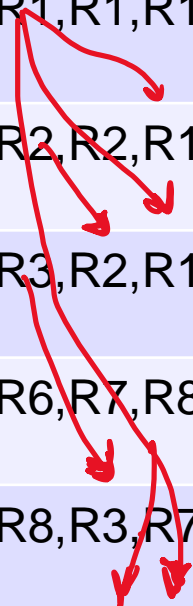
True dependencies

Instructions	Clock Cycle 1	Clock Cycle 2
ADD P10, P2,P3	✓	
XOR P6,P7,P8	✓	
MUL P5,P8,P9	✓	
ADD P4,P8,P9	✓	
SUB P11,P10,P5		✓

Thus, 2 cycles are needed. Thus, $ILP = 5/2 = 2.5$
So, while calculating ILP just neglect the dependencies except for RAW.

Quiz

Instructions	Clock Cycles
ADD R1,R1,R1	1
ADD R2,R2,R1	2
ADD R3,R2,R1	3
ADD R6,R7,R8	1
ADD R8,R3,R7	4
ADD R1,R1,R1	2
ADD R1,R7,R7	1



$$ILP = 7/4 = 1.75$$

What about structural & control dependencies?



ILP with structural & control dependencies

- No structural dependencies:
 - We assume an ideal processor with no hardware limitations
- Perfect same cycle branch prediction
 - 100% correct prediction during fetch

Example: ILP calculation with control dependency

	Cycle1	Cycle2	Cycle3
ADD R1,R2,R3	✓		
MUL R1,R1,R1		✓	
BNE R5,R1,label			✓
ADD R5,R1,R2			
...			
MUL R5,R7,R8	✓		

Not only the instruction at label is executed, but all following instructions will execute without any delays due to branch!

We are assuming perfect branch prediction:

- Even before branch is executed in the fetch cycle itself we know the outcome of the branch.
- Branch itself may be delayed due to true dependencies, but is not causing any delay in the following instructions.

We know that the branch will be taken at the fetch stage

Control dependency of the branch has no effect on ILP.

ILP≠IPC on real processors

- Processor: 3 Issue, out of order, 1 mult, 2 ADD/SUB/XOR

	CycleNo	CycleNo	Cycle No	Cycle No	Cycle No	Cycle No
ADD R1,R2,R3	✓					
SUB R4,R1,R5		✓				
XOR R6,R7,R8	✓					
MUL R5,R8,R9	✓					
ADD R4,R8,R9		✓				

ILP≠IPC on real processors

- Processor: 3 Issue, out of order, 1 Mult, 2 ADD/SUB/XOR

	CycleNo	CycleNo
ADD R1,R2,R3	✓	
SUB R4,R1,R5		✓
XOR R6,R7,R8	✓	
MUL R5,R8,R9	✓	
ADD R4,R8,R9		✓

$$IPC=ILP=5/2=2.5$$

ILP≠IPC on real processors

- Processor: 3 Issue, out of order, 1 Mult, 1 ADD/SUB/XOR

	CycleNo	CycleNo	Cycle No	Cycle No
ADD R1,R2,R3	✓			
SUB R4,R1,R5		✓		
XOR R6,R7,R8			✓	
MUL R5,R8,R9	✓			
ADD R4,R8,R9				✓

$$IPC = 5/4 = 1.25$$

Generally,
 $IPC \leq ILP$

ILP & IPC Quiz

- Processor: 3 Issues, in-order, 3 ALUs

	CycleNo	CycleNo	CycleNo
ADD R1,R2,R3	✓		
ADD R2,R3,R4	✓		
ADD R3,R1,R2		✓	
ADD R7,R8,R9	✓		
ADD R1,R7,R7		✓	
ADD R1,R4,R5	✓		

$$ILP = \frac{6}{2} = 3$$

ILP & IPC Quiz

- Processor: 3 Issues, in-order, 3 ALUs

		CycleNo	CycleNo	CycleNo
I1	ADD R1,R2,R3	✓		
I2	ADD R2,R3,R4	✓		
I3	ADD R3,R1,R2		✓	
I4	ADD R7,R8,R9		✓	
I5	ADD R1,R7,R7			✓
I6	ADD R1,R4,R5			✓

$$\begin{aligned}
 \text{ILP} &= \frac{6}{2} \\
 &= 3
 \end{aligned}$$

(*) Since processor is in-order, we cannot execute this. As I3 is not executed yet, I4 cannot be executed now.

Summary

- ILP: Ideal Out of Order Processor (O-O Processor)
 - With Perfect Branch-predictor
- $ILP \geq IPC$
 - Narrow Issue (1/2/3 instructions per cycle), In-order
 - Wide Issue, In-order
 - Wide Issue, Out-of-Order
- Our processor should be:
 - Fetch/Exec/... >> 1 instructions/cycle (say 4)
 - Eliminate False dependencies
 - Reorder Instructions

From Scalar to Superscalar

- Our 5-stage pipeline was based on the idea that the instruction execution cycle could be decomposed into non-overlapping stages with the instruction passing through every stage at every cycle.
 - This is called a scalar processor
 - Ideal throughput of 1, $IPC=1$
- With the same ISA, the possibility to improve the execution time, thus depends on:
 - Increasing the ideal IPC of 1 by radically modifying the structure of the pipeline to allow more than one instruction to be in each stage at a given time.
 - These are called as super-scalar processors.

Does this effect the pipeline?

- From the micro-architecture point of view, the pipeline is called wide pipeline.
- It brings several fundamental changes:
 - We need several functional units.
 - The pipeline also does not remain linear.
 - Each stage of the pipeline gets affected.
- Also hand-in-hand, to decrease execution time, by increasing clock frequency, the processor needs to do less job per cycle.
 - This is achieved by deeper pipelines.
- Modern processors are therefore both deeper and wider.

Viewing the pipelines in super-scalar processors

- To study the design decisions that are needed to implement concurrency caused by the superscalar effect, and the consequences of the deeper pipeline, we view the same in two parts:
 - Front end: Corresponds to the FETCH and DECODE stages
 - The front-end must now fetch and decode several instructions at once.
 - m-way superscalar: The number of instructions brought ideally into the pipeline at each cycle is m.
 - Back-end: Comprises of Execution, Memory, and Write Back stages.

In-order and Out-of-order Superscalar Processors

- In both cases, the instructions proceed in the front-end in program order.
- In-order or static processors: Instructions leave the front-end in program order and all data-dependencies are resolved before they go to the back-end.
- Out-of-order or dynamic processors: Instructions can leave the front-end and execute in the back-end before some of their program order predecessors.
 - The Write Back stage needs to be designed so that the semantics of the program are respected.
 - Such processors are more complex to build.
 - Require 30% more logic for the same number of functional units

Finer Points

- Theoretically, an m -way superscalar processor with k -times more clock frequency should provide mk -times speedup.
- However there are several bottlenecks:
 - We need to discover the amount of ILP in the program.
 - Very difficult in in-order processors as instructions that have RAW dependencies cannot leave the front-end until dependencies are resolved.
 - An out-of-order processor though can exploit the ILP better but is very complex to design.
 - Hardly more than 6-way.
 - Intel's 12th generation processors are 5-way decode.

Finer Points (Contd.)

- Second factor that limits m is that the back-end requires m functional units, and the number of forwarding units increase quadratically.
- The increase in clock frequency is also capped due to power dissipations: hovers around 3-5 GHz.
- Second constraint is the pipelined registers, which needs to be written and read in every cycle.
 - This provides a lower bound to the cycle time.
- Deep pipelines also have problems due to mispredictions.
 - Intel core suggested 14 stage pipeline which reversed the trend of Pentium D with 31 stages vs 10 stages in Pentium III

Finer Points (Contd.)

- In-order processors were the first successors of scalar processors
- They were easier to implement compared to the dynamic superscalar processors then due to the higher clock speeds
 - Though out of order processors had better IPC.
- Today high performance single processor chips are out of order.
 - The speed advantages of static processors are not existent now due to cap for controlling power dissipations
- Because of the development of Moore's law, which allows more chip logic, single processors are replaced by multi-processors on a chip (CMP).
- The in-order to out-of-order transition requires to introduce concepts of Tomasulo's algorithm introduced for IBM Systems 360/91.



Thank You!