Instruction-Level Parallelism

Introduction

1. Goal of instruction-level parallelism (ILP) is to finish more than a single instruction per cycle
   - Requires resolving data dependencies

ILP All in the Same Cycle

1. If all instructions can be done in parallel, CPI = 5 / Inf = 0
   - However, dependencies across registers prevents this ideal case from being realized
     - Instructions 1 and 2 below have a RAW dependency for R1

Instructions	1	2	3	4	5
R1 = R2 + R3	F	D	E	M	WB
R4 = R1 - R5	F	D	E	M	WB
R6 = R7 ^ R8	F	D	E	M	WB
R5 = R8 * R9	F	D	E	M	WB
R4 = R8 + R9	F	D	E	M	WB

The Execute Stage

1. Forewarding: Sending values from previous computations ahead prior to the value is written to a register
   - This only works for instructions in the next cycle, not the same cycle as is desired for executing instructions in parallel
2. Instead of forewarding, instructions with dependencies can just stall
   - Instructions without dependencies can still execute in parallel

RAW Dependencies

1. Consider the following program:
   - I2 depends on I1
   - I4 depends on I3
   - I5 depends on I4
   - What is the CPI?
     - It takes 3 cycles to finish 5 instructions due to dependencies, so 0.6
2. RAW dependencies determine the actual CPI that can be realized
   - If all instructions have RAW dependencies, it isn't possible to parallelize

WAW Dependencies

1. When instructions get reordered due to RAW dependencies, it can cause an instruction (I1) that should have completed first to actually complete after a later instruction (I2)
   - This results in the value from I1 persisting in the register even though the result from I2 should be present

Dependency Quiz

1. Processor with typical 5-stage pipeline (Fetch, Decode, Execute, Memory, Write)
2. Processor does forwarding
3. Processor can execute 10 instructions in each stage assuming no dependencies
4. When do we execute writeback for the following 6 instructions?

Instructions	Execute	Writeback
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

Removing False Dependencies

1. RAW: True dependencies
2. WAR: False (name) dependencies
3. WAW: False (name) dependencies
4. True dependencies must delay instructions to satisfy them
5. Name dependencies simply require handling multiple values for the same register

Duplicating Register Values

1. Consider the following program:

```
I1: R1 = R2 + R3
I2: R4 = R1 - R5
I3: R3 = R4 + 1
I4: R4 = R8 - R9
```

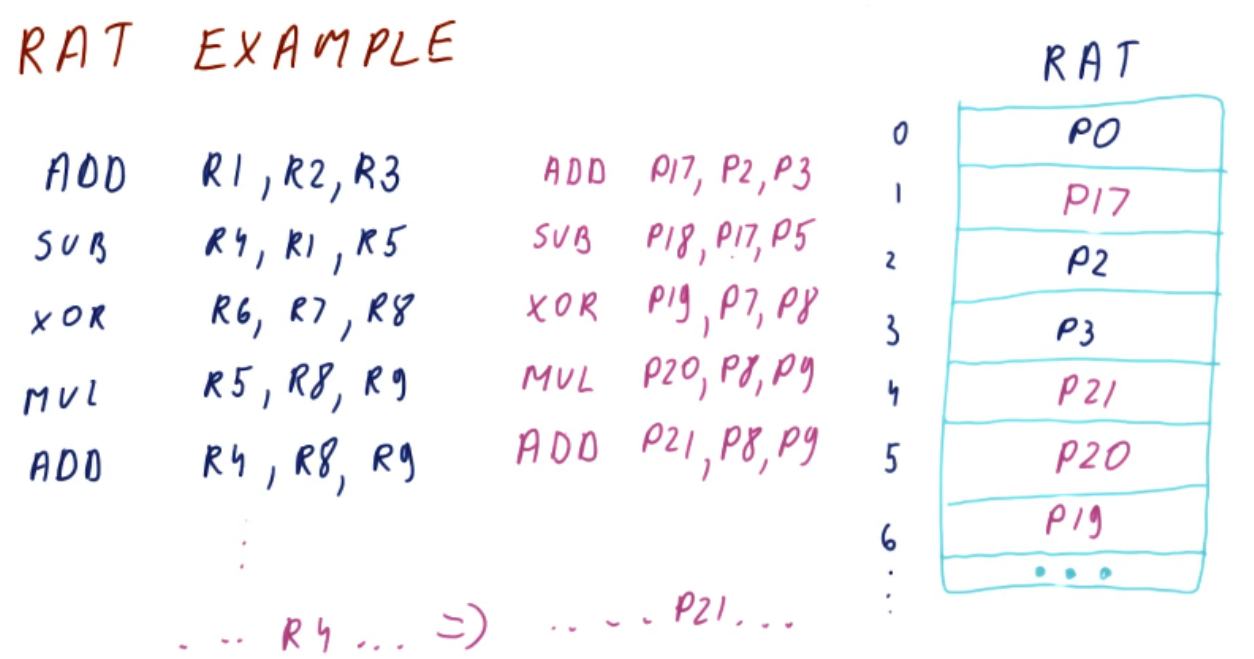
2. There's a WAW dependency between I1 and I4
   - Instead of using R4 for both, store all possible values for R4
   - An instruction that wants to read R4 will have to search for the correct value
   - This is very complicated in practice

Register Renaming

1. Architectural registers: Registers programmer/compiler use
2. Physical registers: All places values can go
3. Rewrite program to use physical registers
4. Register Allocation Table (RAT): Table that says which physical register has a value for which architectural register

RAT Example

1. Hardware can use whatever physical register it desires, but this is hidden from the programmer
2. This renaming results in a removal of WAW dependencies
   - Earlier instructions will be stored in different physical registers from later ones, removing the dependence



Register Allocation Table

Register Renaming Quiz

1. Update the RAT and give the renamed instructions for the following program:

Instruction	Fetched	Renamed
I1	MUL R2, R2, R2	MUL P7, P2, P2
I2	ADD R1, R1, R2	ADD P8, P1, P7
I3	MUL R2, R4, R4	MUL P9, P4, P4
I4	ADD R3, R3, R2	ADD P10, P3, P9
I5	MUL R2, R6, R6	MUL P11, P6, P6
I6	ADD R5, R5, R2	ADD P12, P5, P11

Register	RAT
R1	P8
R2	P11
R3	P10
R4	P4
R5	P12
R6	P6

False Dependencies After Renaming

1. True dependencies between I1 -> I2, I3 -> I4, and I5 -> I6
2. Name dependencies between I1 -> I3, I3 -> I5
3. Anti dependencies between I2 -> I3, I4 -> I5
4. After renaming, only the true dependencies remain

- Fetched: CPI = 1 (IPC = 1), must execute sequentially
- Renamed: CPI = 2 / 6 (IPC = 3)
  - I1, I3, I5 can be executed in cycle 1
  - I2, I4, I6 can be executed in cycle 2

Instruction	Fetched	Renamed
I1	MUL R2, R2, R2	MUL P7, P2, P2
I2	ADD R1, R1, R2	ADD P8, P1, P7
I3	MUL R2, R4, R4	MUL P9, P4, P4
I4	ADD R3, R3, R2	ADD P10, P3, P9
I5	MUL R2, R6, R6	MUL P11, P6, P6
I6	ADD R5, R5, R2	ADD P12, P5, P11

Instruction Level Parallelism (ILP)

1. ILP == IPC when:
   - Processor does entire instruction in 1 cycle
   - Processor can do any number of instructions in the same cycle
     - Must obey true dependencies
2. Steps to get ILP
   - Rename registers
   - "Execute"
3. ILP is a property of a program, not a processor (ideal processor)
   - Only consider for an ideal processor, not a specific processor
   - IPC is processor-specific

ILP Example

1. Consider the following program:

```
I1: ADD P10, P2, P3
I2: XOR P6, P7, P8
I3: MUL P5, P8, P9
I4: ADD P4, P8, P9
I5: SUB P11, P10, P5
```

2. True dependency between I3 and I5, so they must execute in separate cycles
3. ILP = 5 instructions / 2 cycles = 2.5
4. Convenient tricks for ILP
   - Don't actually have to rename registers, only need to consider true dependencies

ILP Quiz

1. Consider the following program:

```
ADD R1, R1, R1 (cycle 1)
ADD R2, R2, R1 (cycle 2)
ADD R3, R2, R1 (cycle 3)
ADD R6, R7, R8 (cycle 1)
ADD R8, R3, R7 (cycle 4)
ADD R1, R1, R1 (cycle 2)
ADD R1, R7, R7 (cycle 1)
```

2. ILP = 7 instructions / 4 cycles = 1.75

ILP with Structural and Control Dependencies

1. No structural dependencies
   - Occur when there is insufficient hardware to do instructions in the same cycle
     - ILP is only concerned with an ideal processor
2. Control dependencies
   - Perfect same-cycle branch prediction
   - In the program below, the "MUL R5, R7, R8" instruction can execute in cycle 1 since we assume the branch predictor is perfect and it has no data dependencies
   - Control dependencies are essentially eliminated with our assumption of an ideal processor

Instructions	1	2	3
ADD R1, R2, R3	X		
MUL R1, R1, R1		X	
BNE R1, R1, L1			X
ADD R5, R1, R2			
L1: MUL R5, R7, R8	X		

ILP vs IPC

1. ILP is not necessarily equal to IPC
2. Processor:
   - 2-issue (2 instructions per cycle)
   - Out-of-order superscalar (doesn't need to execute in program order)
   - 1 MUL, 2 ADD/SUB/XOR
3. Consider the following program:

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

4. ILP = 5 / 2 = 2.5
   - ADD, XOR, MUL, ADD execute in the first cycle
   - SUB executes in the second cycle
5. IPC = 5 / 2 = 1.5
   - Last add must execute in the second cycle because both adders are in use
   - If we only had 1 adder, 5 instructions execute in 4 cycles = 1.25
6. IPC can be equal to ILP, but this is not necessarily true
   - ILP is greater than or equal to IPC on any given processor

IPC vs ILP Quiz

1. Processor description:
   - 3-issue in-order
   - 3 ALUs
2. Consider the following program:

```
ADD R1, R2, R3
ADD R2, R3, R4
ADD R3, R1, R2
ADD R7, R8, R9
ADD R1, R7, R7
ADD R1, R4, R5
```

3. ILP = 6 / 2 = 3
4. IPC = 6 / 3 = 2
5. In an in-order processor, once an instruction is skipped, we can no longer execute any other instructions

IPC vs ILP Discussion

1. ILP: Ideal out-of-order processor
2. ILP greater than or equal to IPC
    - Narrow-issue in-order -> Limited mostly by narrow-issue
    - Wide-issue in-order -> Limited by in-order
3. Wide issue processors need to be out-of-order to benefit from ILP
    - Fetch/Execute/... >> 1 instruction / cycle
    - Eliminate false dependencies
    - Reorder instructions

Conclusion

1. Should be able to get good performance even with many data dependencies