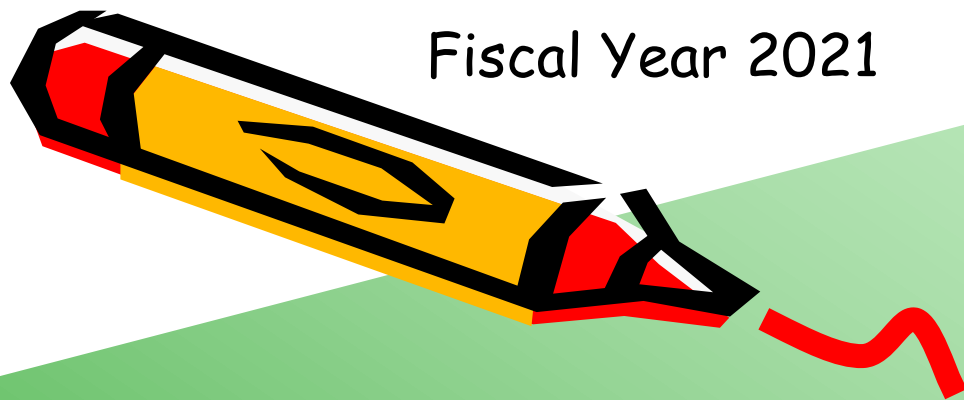


Fiscal Year 2021

Ver. 2021-01-12a



Course number: CSC.T433
School of Computing,
Graduate major in Computer Science

Advanced Computer Architecture

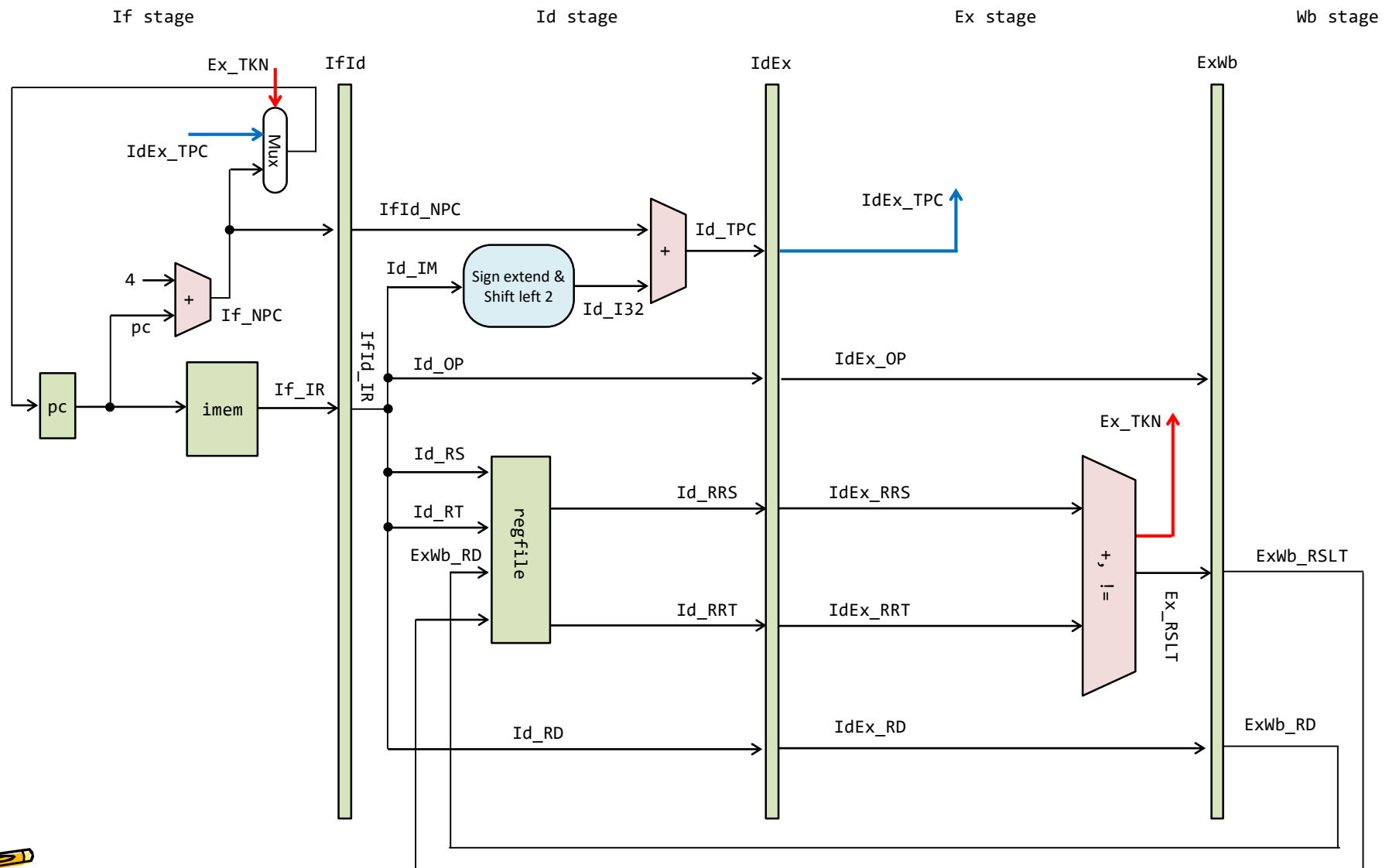
7. Instruction Level Parallelism: Dynamic Scheduling



www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W936
Mon 14:20-16:00, Thr 14:20-16:00

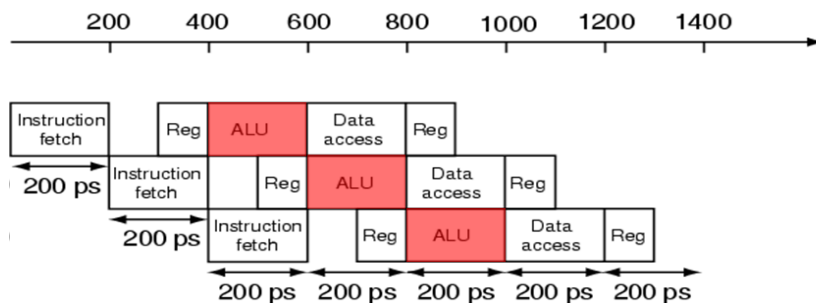
Kenji Kise, Department of Computer Science
kise_at_c.titech.ac.jp

Four stage pipelined processor supporting ADD and BNE, which does not adopt data forwarding (proc08.v, Assignment 5)

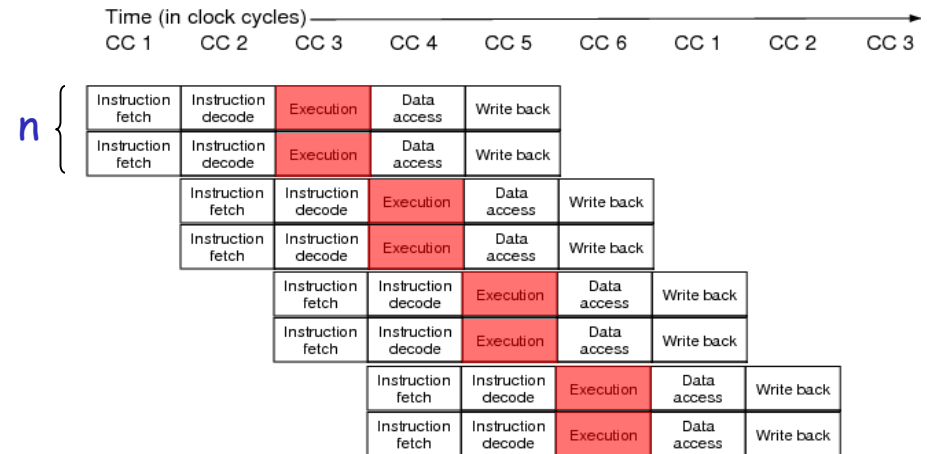


Scalar and Superscalar processors

- Scalar processor can execute at most one single instruction per clock cycle using one ALU.
 - IPC (Executed Instructions Per Cycle) is less than 1.
- Superscalar processor can execute more than one instruction per clock cycle by executing multiple instructions using multiple pipelines.
 - IPC (Executed Instructions Per Cycle) can be more than 1.
 - using n pipelines is called n -way superscalar



(a) pipeline diagram of scalar processor



(b) pipeline diagram of 2-way superscalar processor

Exploiting Instruction Level parallelism (ILP)

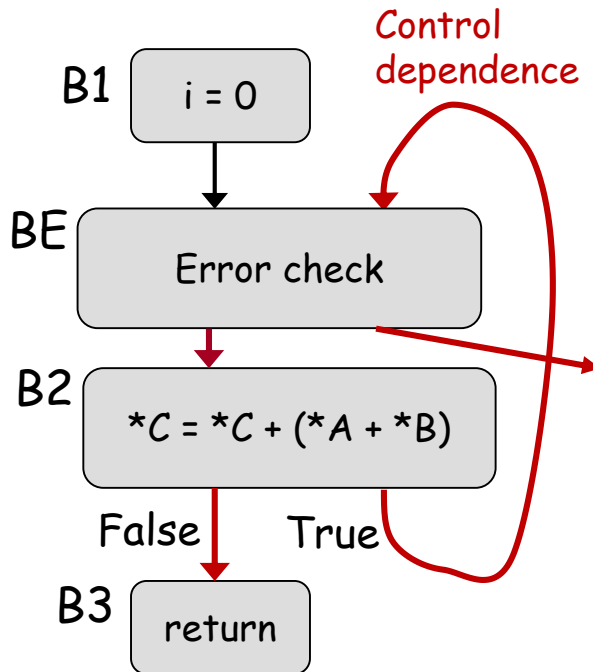
- A superscalar processor has to handle some flows efficiently to exploit ILP
 - **Control flow**
 - To execute n instructions per clock cycle, the processor has to fetch at least n instructions per cycle.
 - The main obstacles are branch instruction (BNE, BEQ)
 - Another obstacle is instruction cache
 - **Register data flow**
 - **Dynamic scheduling**
 - **Memory data flow**



Exploiting Instruction Level Parallelism (ILP)

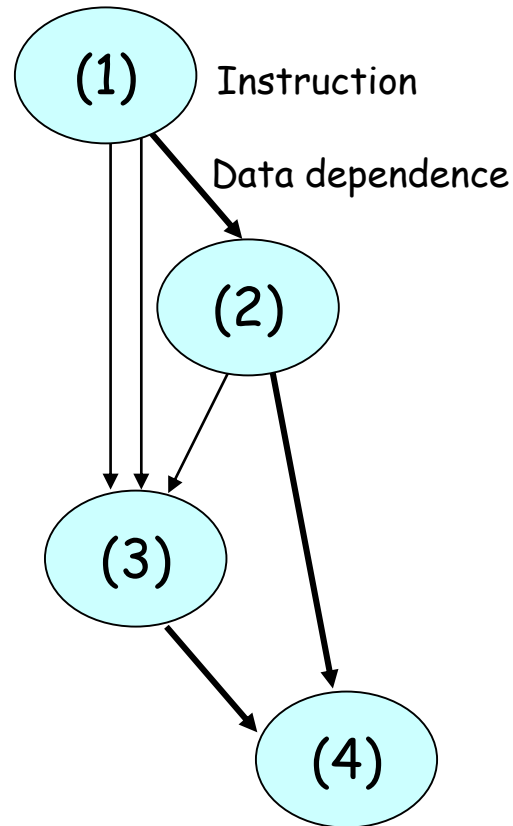
What is the solution?

Prediction & speculation



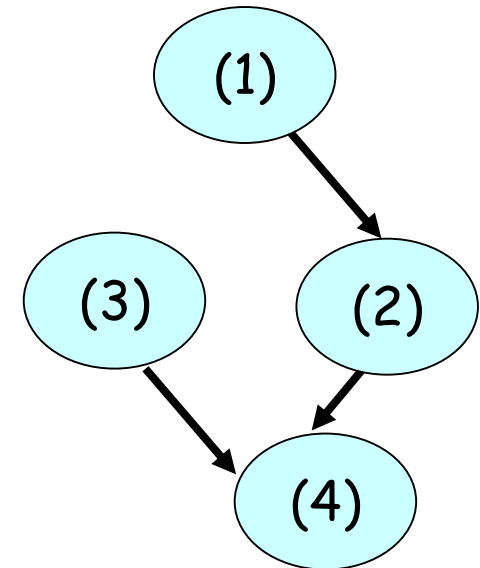
Control flow graph

4 cycles for 4 insns
ILP = 1.0



Data flow graph

3 cycles for 4 insns
ILP = 1.33



Data flow graph



Exercise: what is data dependence

- Draw a data flow graph for each instruction stream

$$R3 = R2 + 1 \quad (1)$$

$$R5 = R4 + 2 \quad (2)$$

$$R7 = R6 + 3 \quad (3)$$

Instruction stream 1

$$R3 = R2 + 1 \quad (1)$$

$$R5 = R4 + 2 \quad (2)$$

$$R7 = R3 + 3 \quad (3)$$

Instruction stream 2

$$R3 = R2 + 1 \quad (1)$$

$$R3 = R4 + 2 \quad (2)$$

$$R7 = R6 + 3 \quad (3)$$

Instruction stream 3

$$R3 = R2 + 1 \quad (1)$$

$$R5 = R4 + 2 \quad (2)$$

$$R4 = R6 + 3 \quad (3)$$

Instruction stream 4



True data dependence

- Insn i writes a register that insn j reads, **RAW (read after write)**
- Program order must be preserved to ensure insn j receives the value of insn i.

$$R3 = R3 \times R5 \quad (1)$$

$$R4 = R3 + 1 \quad (2)$$

$$R3 = R5 + 2 \quad (3)$$

$$R7 = R3 + R4 \quad (4)$$

Assume $R3=10$, $R5=3$

$$20 = 10 \times 2 \quad (1)$$

$$21 = 20 + 1 \quad (2)$$

$$5 = 3 + 2 \quad (3)$$

$$26 = 5 + 21 \quad (4)$$

Assume $R3=10$, $R5=3$

$$20 = 10 \times 2 \quad (1)$$

$$21 = 20 + 1 \quad (2)$$

$$41 = 20 + 21 \quad (4)$$

$$5 = 3 + 2 \quad (3)$$

Output dependence

- Insn i and j write the same register, **WAW** (write after write)
- Program order must be preserved to ensure that the value finally written corresponds to instruction j.

$$\begin{array}{ll} \textcircled{R3} = R3 \times R5 & (1) \\ R4 = R3 + 1 & (2) \\ \textcircled{R3} = R5 + 2 & (3) \\ R7 = R3 + R4 & (4) \end{array}$$

Assume $R3=10$, $R5=3$

$$\begin{array}{ll} \textcircled{20} = 10 \times 2 & (1) \\ 21 = 20 + 1 & (2) \\ \textcircled{5} = 3 + 2 & (3) \\ 26 = 5 + 21 & (4) \end{array}$$

Assume $R3=10$, $R5=3$

$$\begin{array}{ll} \textcircled{5} = 3 + 2 & (3) \\ \textcircled{20} = 10 \times 2 & (1) \\ 21 = 20 + 1 & (2) \\ 41 = 20 + 21 & (4) \end{array}$$



Antidependence

- Insn i reads a register that insn j writes, **WAR (write after read)**
- Program order must be preserved to ensure that i reads the correct value.

$$R3 = R3 \times R5 \quad (1)$$

$$R4 = R3 + 1 \quad (2)$$

$$R3 = R5 + 2 \quad (3)$$

$$R7 = R3 + R4 \quad (4)$$

Assume $R3=10$, $R5=3$

$$20 = 10 \times 2 \quad (1)$$

$$21 = 20 + 1 \quad (2)$$

$$5 = 3 + 2 \quad (3)$$

$$26 = 5 + 21 \quad (4)$$

Assume $R3=10$, $R5=3$

$$20 = 10 \times 2 \quad (1)$$

$$5 = 3 + 2 \quad (3)$$

$$6 = 5 + 1 \quad (2)$$

$$11 = 5 + 6 \quad (4)$$



Exercise: what is data dependence

- Draw a data flow graph for each instruction stream

$$R3 = R2 + 1 \quad (1)$$

$$R5 = R4 + 2 \quad (2)$$

$$R7 = R6 + 3 \quad (3)$$

Instruction stream 1

$$R3 = R2 + 1 \quad (1)$$

$$R5 = R4 + 2 \quad (2)$$

$$R7 = R3 + 3 \quad (3)$$

Instruction stream 2

$$R3 = R2 + 1 \quad (1)$$

$$R3 = R4 + 2 \quad (2)$$

$$R7 = R6 + 3 \quad (3)$$

Instruction stream 3

$$R3 = R2 + 1 \quad (1)$$

$$R5 = R4 + 2 \quad (2)$$

$$R4 = R6 + 3 \quad (3)$$

Instruction stream 4



Data dependence and renaming

- True data dependence (RAW)
- Name dependences
 - Output dependence (WAW)
 - Antidependence (WAR)

$$R3 = R3 \times R5 \quad (1)$$

$$R4 = R3 + 1 \quad (2)$$

$$\textcolor{red}{R8} = R5 + 2 \quad (3)$$

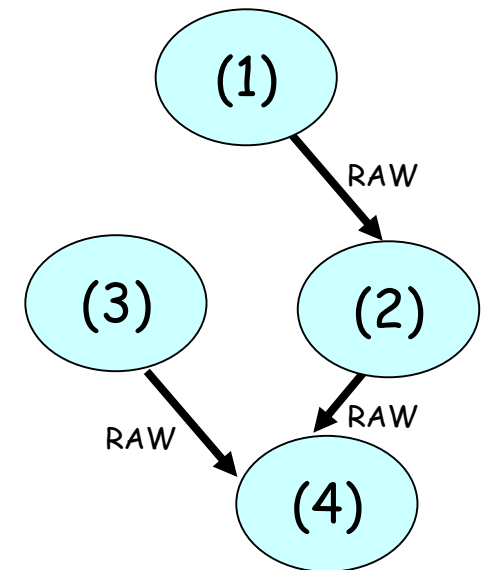
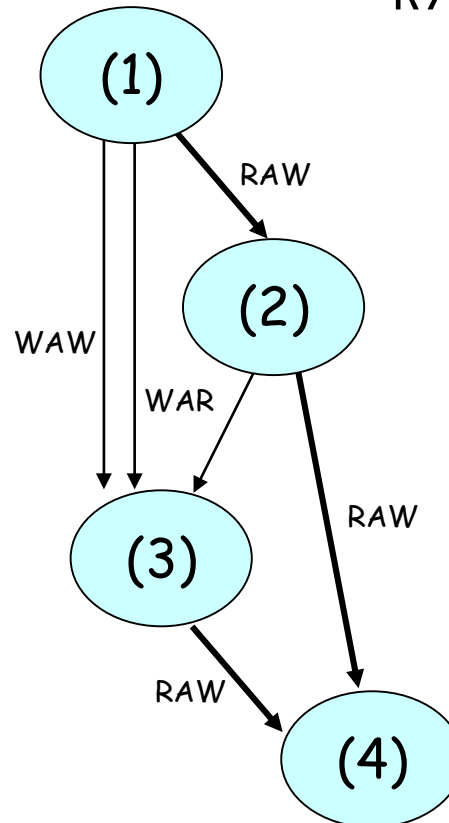
$$R7 = \textcolor{red}{R8} + R4 \quad (4)$$

$$R3 = R3 \times R5 \quad (1)$$

$$R4 = R3 + 1 \quad (2)$$

$$R3 = R5 + 2 \quad (3)$$

$$R7 = R3 + R4 \quad (4)$$





Hardware register renaming

- Logical registers (architectural registers) which are ones defined by ISA
 - \$0, \$1, ... \$31
- Physical registers
 - Assuming plenty of registers are available, p0, p1, p2, ...
- A processor renames (converts) each logical register to a unique physical register dynamically

Typical instruction pipeline of scalar processor



Typical instruction pipeline of high-performance superscalar processor



Exercise: register renaming

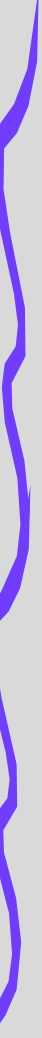
- Rename the following instruction stream using physical registers of p9, p10, p11, and p12

I0: sub \$5,\$1,\$2

I1: add \$9,\$5,\$4

I2: or \$5,\$5,\$2

I3: and \$2,\$9,\$1



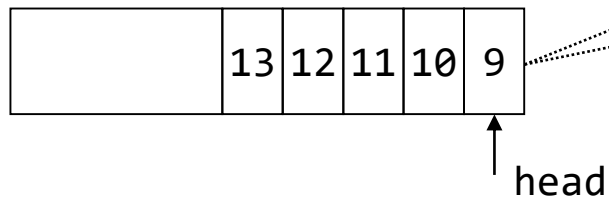
Example behavior of register renaming (1/4)

- Renaming the first instruction I0

Cycle 1

I0: sub \$5,\$1,\$2
I1: add \$9,\$5,\$4
I2: or \$5,\$5,\$2
I3: and \$2,\$9,\$1

Free tag buffer



dst = \$5
src1 = \$1
src2 = \$2

Register map table

0	0
1	1
2	2
3	3
4	4
5	5 -> 9
6	6
7	7
8	8
9	
10	
31	

dst = p9
src1 = p1
src2 = p2

I0: sub p9,p1,p2



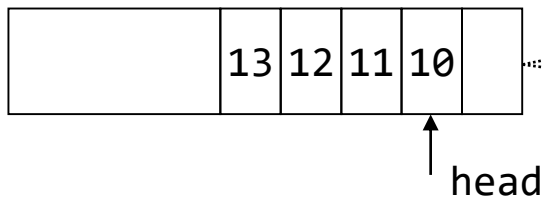
Example behavior of register renaming (2/4)

- Renaming the second instruction I1

Cycle 2

I0: sub \$5,\$1,\$2
I1: add \$9,\$5,\$4
I2: or \$5,\$5,\$2
I3: and \$2,\$9,\$1

Free tag buffer



dst = \$9
src1 = \$5
src2 = \$4

Register map table

0	0
1	1
2	2
3	3
4	4
5	9
6	6
7	7
8	8
9	->10
10	
31	

dst = p10
src1 = p9
src2 = p4

I0: sub p9,p1,p2
I1: add p10,p9,p4



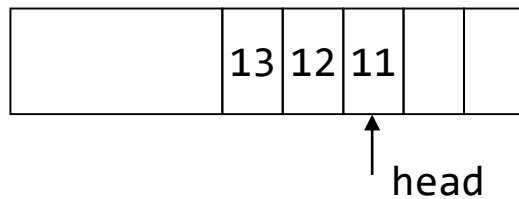
Example behavior of register renaming (3/4)

- Renaming instruction I2

Cycle 3

I0: sub \$5,\$1,\$2
I1: add \$9,\$5,\$4
I2: or \$5,\$5,\$2
I3: and \$2,\$9,\$1

Free tag buffer



dst = \$5
src1 = \$5
src2 = \$2

Register map table

0	0
1	1
2	2
3	3
4	4
5	9->11
6	6
7	7
8	8
9	10
10	
31	

dst = p11
src1 = p9
src2 = p2

I0: sub p9,p1,p2
I1: add p10,p9,p4
I2: or p11,p9,p2

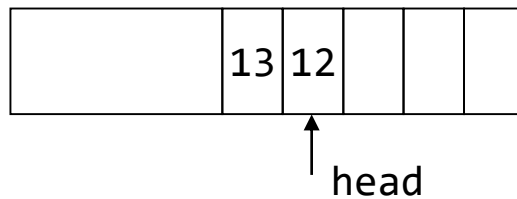
Example behavior of register renaming (4/4)

- Renaming instruction I3

Cycle 4

I0: sub \$5,\$1,\$2
I1: add \$9,\$5,\$4
I2: or \$5,\$5,\$2
I3: and \$2,\$9,\$1

Free tag buffer



dst = \$2
src1 = \$9
src2 = \$1

Register map table

0	0
1	1
2	2->12
3	3
4	4
5	11
6	6
7	7
8	8
9	10
10	
31	

dst = p12
src1 = p10
src2 = p1

I0: sub p9,p1,p2
I1: add p10,p9,p4
I2: or p11,p9,p2
I3: and p12,p10,p1



-

Renaming **two instructions** per cycle for superscalar

- Renaming instruction I0 and I1

Cycle 1

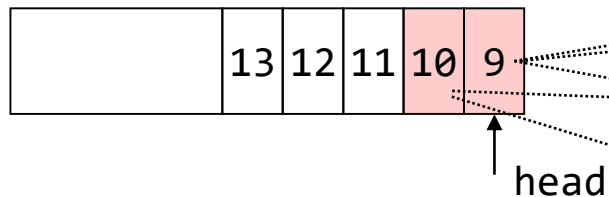
I0: sub \$5,\$1,\$2

I1: add \$9,\$5,\$4

I2: or \$5,\$5,\$2

I3: and \$2,\$9,\$1

Free tag buffer



I0 A_dst = \$5
A_src1 = \$1
A_src2 = \$2

I1 B_dst = \$9
B_src1 = \$5
B_src2 = \$4

Register map table

0	0
1	1
2	2
3	3
4	4
5	5->9
6	6
7	7
8	8
9	->10
10	
31	

A_dst = p9
A_src1 = p1
A_src2 = p2

B_dst = p10
B_src1 = p9
B_src2 = p4

If B_src1==A_dst, use tag from free tag buffer

I0: sub p9,p1,p2
I1: add p10,p9,p4



Pollack's Rule



- Pollack's Rule states that microprocessor "performance increase due to microarchitecture advances **is roughly proportional to the square root of the increase in complexity**". Complexity in this context means processor logic, i.e. its area.



WIKIPEDIA



True data dependence

- Insn i writes a register that insn j reads, **RAW (read after write)**
- Program order must be preserved to ensure insn j receives the value of insn i.

$$R3 = R3 \times R5 \quad (1)$$

$$R4 = R3 + 1 \quad (2)$$

$$R3 = R5 + 2 \quad (3)$$

$$R7 = R3 + R4 \quad (4)$$

Assume $R3=10$, $R5=3$

$$20 = 10 \times 2 \quad (1)$$

$$21 = 20 + 1 \quad (2)$$

$$5 = 3 + 2 \quad (3)$$

$$26 = 5 + 21 \quad (4)$$

Assume $R3=10$, $R5=3$

$$20 = 10 \times 2 \quad (1)$$

$$21 = 20 + 1 \quad (2)$$

$$41 = 20 + 21 \quad (4)$$

$$5 = 3 + 2 \quad (3)$$



Recommended Reading

- Focused Value Prediction

- Sumeet Bandishte, Jayesh Gaur, Zeev Sperber, Lihu Rappoport, Adi Yoaz, and Sreenivas Subramoney, Intel
- ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), pp. 79-91, 2020

- A quote:

"Value Prediction was proposed to speculatively break true data dependencies, thereby allowing Out of Order (OOO) processors to achieve higher instruction level parallelism (ILP) and gain performance. State-of-the-art value predictors try to maximize the number of instructions that can be value predicted, with the belief that a higher coverage will unlock more ILP and increase performance. Unfortunately, this comes at increased complexity with implementations that require multiple different types of value predictors working in tandem, incurring substantial area and power cost. In this paper we motivate towards lower coverage, but focused, value prediction. Instead of aggressively increasing the coverage of value prediction, at the cost of higher area and power, we motivate refocusing value prediction as a mechanism to achieve an early execution of instructions that frequently create performance bottlenecks in the OOO processor. Since we do not aim for high coverage, our implementation is light-weight, needing just 1.2 KB of storage. Simulation results on 60 diverse workloads show that we deliver 3.3% performance gain over a baseline similar to the Intel Skylake processor. This performance gain increases substantially to 8.6% when we simulate a futuristic up-scaled version of Skylake. In contrast, for the same storage, state-of-the-art value predictors deliver a much lower speedup of 1.7% and 4.7% respectively. Notably, our proposal is similar to these predictors in performance, even when they are given nearly eight times the storage and have 60% more prediction coverage than our solution.



