Fiscal Year 2023

Ver. 202-12-25a

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

Advanced Computer Architecture

5. Instruction Level Parallelism: Concepts and Challenges

www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W834, Lecture (Face-to-face)
Mon 13:30-15:10, Thr 13:30-15:10

Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

Performance Factors

CPU execution time for a program = # CPU clock cycles for a program / clock rate

Performance is the inverse of CPU execution time.

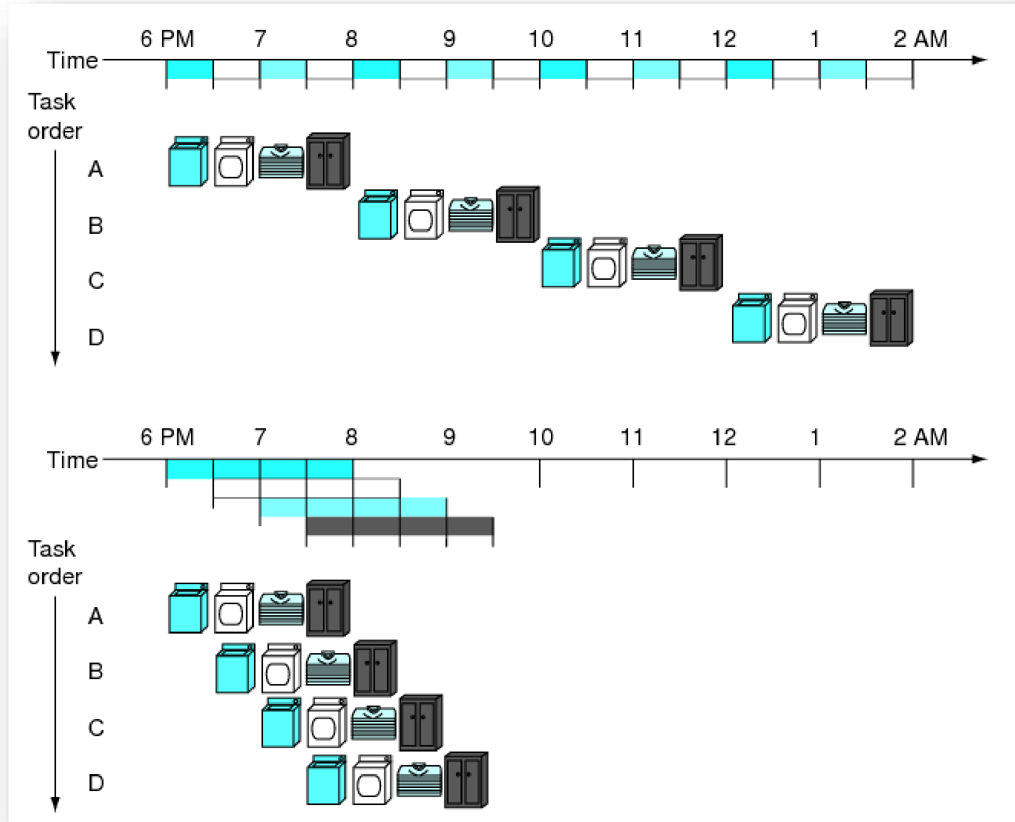
Performance for a program = clock rate x 1 / # CPU clock cycles for a program

- Performance = f x IPC
    - f : frequency (clock rate)
    - IPC : executed (retired) instructions per cycle

- The performance can be improved by increasing either f or IPC

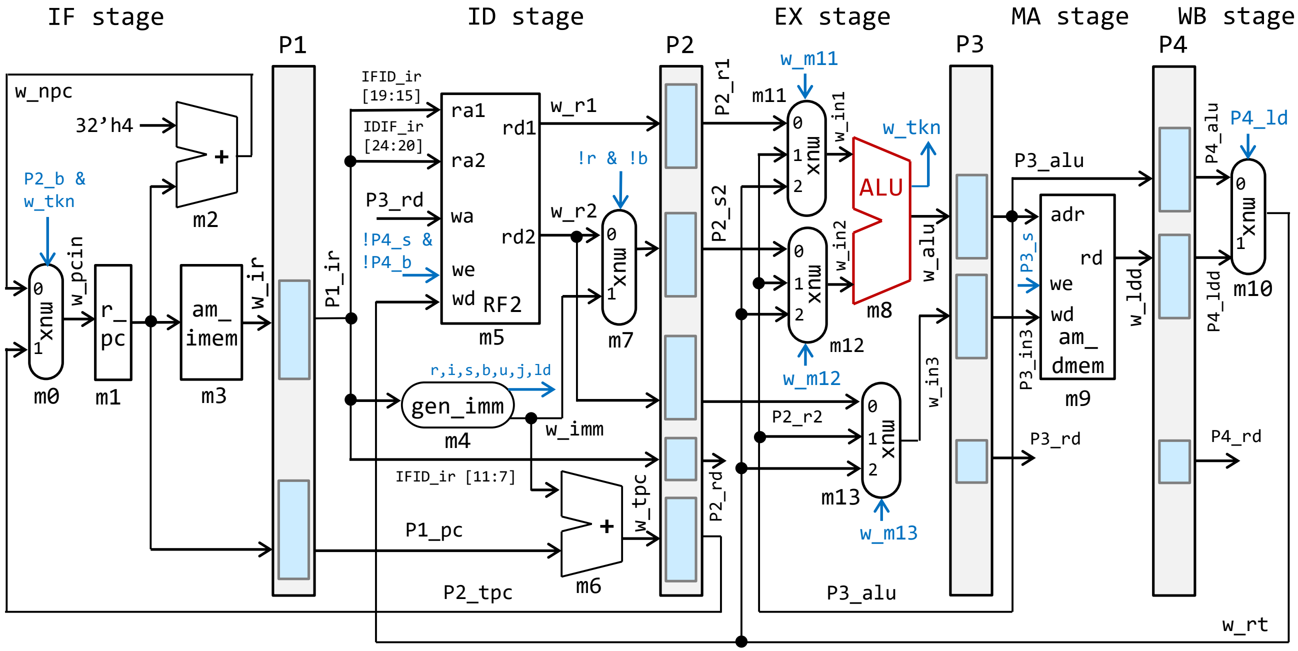
Single-cycle implementation and pipelining

- When the washing of load A is finished at 6:30 p.m., another washing of load B starts.
- Pipelined laundry takes 3.5 hours just using the same hardware resources. The cycle time is 30 minutes.
- What is the latency (execution time) of each load?



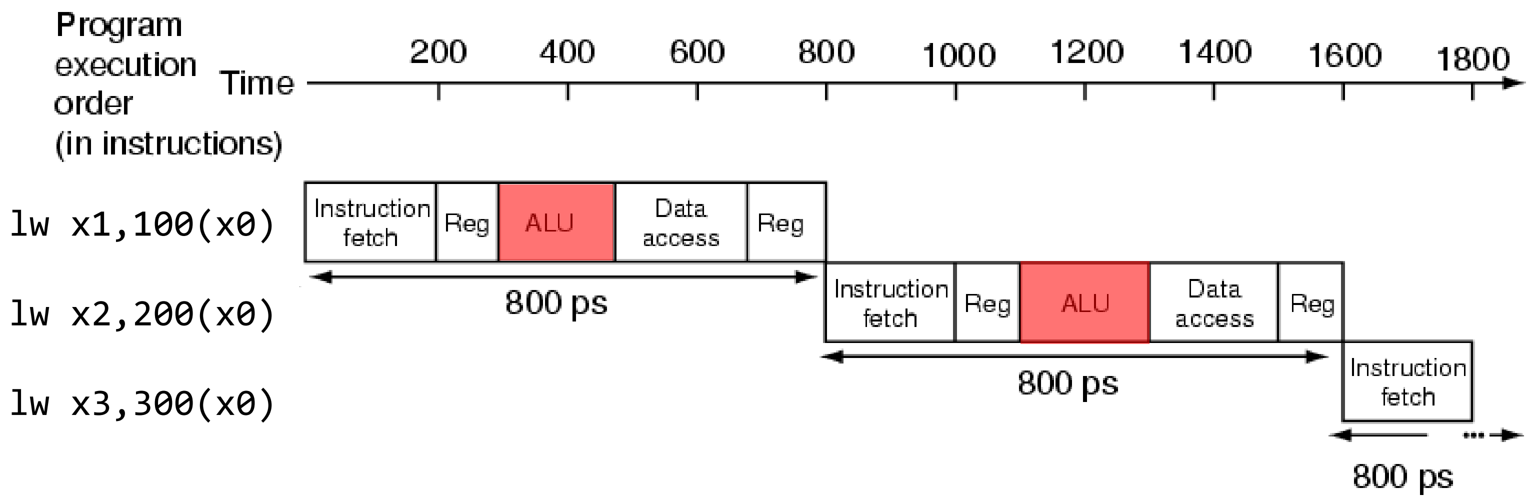
5-stage pipelining RISC-V processor with data forwarding

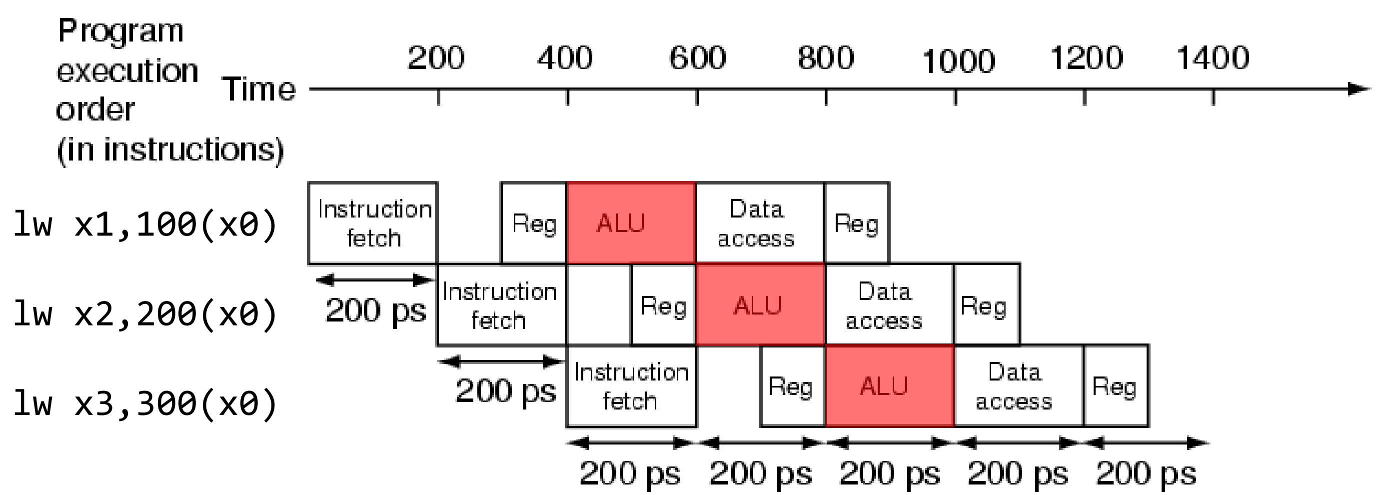
- The strategy is to separate instruction fetch step (IF), instruction decode step (ID), execution step (EX), memory access step (MA), and write back step (WB).
- Use the pipeline registers P1, P2, P3, P4.



Single-cycle and pipelining processors

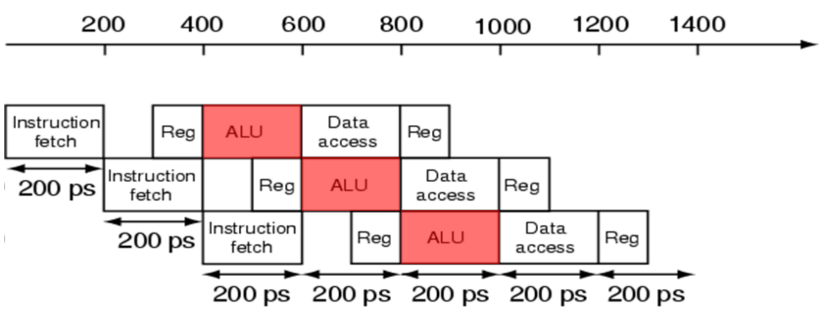
- The pipelining can improve ALU utilization to nearly 100%.



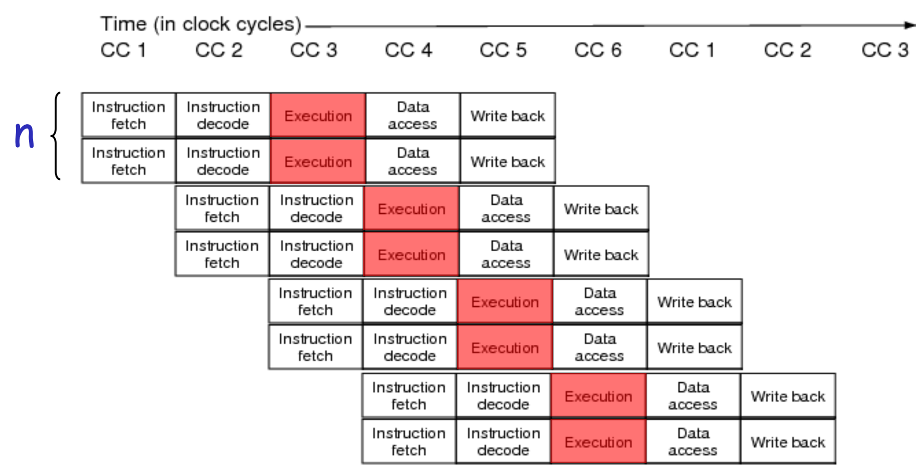


Scalar and Superscalar processors

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



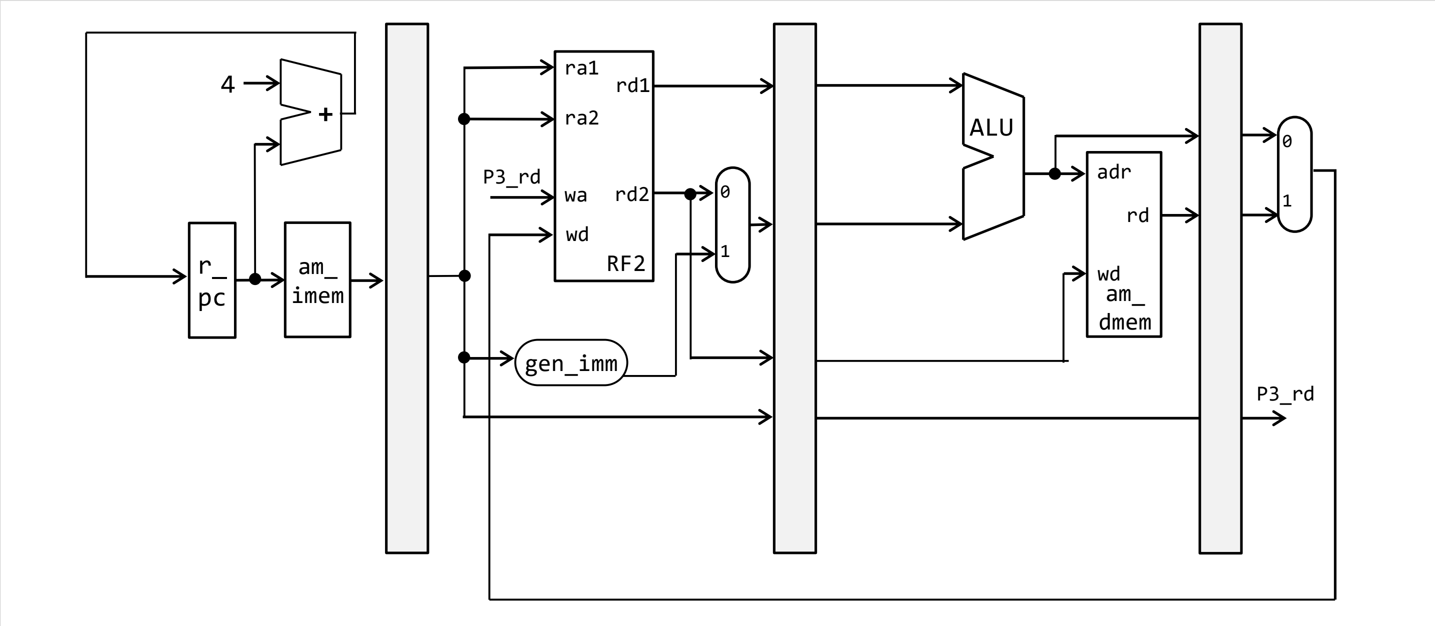
(a) pipeline diagram of scalar processor



(b) pipeline diagram of 2-way superscalar processor

Exercise 1

- Referring to the following diagram, draw a block diagram of a 2-way superscalar processor (4-stage pipelining) supporting add, addi, lw, and sw, which does not adopt data forwarding



A block diagram of a scalar processor (4-stage pipelining) supporting add, addi, lw, and sw, which does not adopt data forwarding

Exploiting Instruction Level parallelism (ILP)

- A superscalar has to handle some flows efficiently to exploit ILP
  - Control flow (control dependence)
    - 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)
    - Prediction
    - Another obstacle is instruction cache
  - Register data flow (data dependence)
    - Out-of-order execution
      - Register renaming
      - Dynamic scheduling
  - Memory data flow
    - Out-of-order execution
    - Another obstacle is instruction cache

(5) RISC-V branch if not equal instructions (bne)

- RISC-V conditional branch instructions (bne, branch if not equal) :

```
bne x4, x5, Lbl  # go to Lbl if x4!=x5
```

```
Ex:   if (i==j) h = i + j;

      bne x4, x5, Lbl1   # if (i!=j) goto Lbl1
      add x6, x4, x5     # h = i + j;
Lbl1: ...
```

- Instruction Format (B-type):

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type

- How is the branch destination address specified?

Why do branch instructions degrade IPC?

- Another approach is fetching the following instructions (an instruction at the next address and following ones) when a branch (bne) is fetched.
- When a branch (08 bne) is taken, the wrong instructions fetched are flushed.

	cc1	cc2	cc3	cc4	cc5	cc6	cc7	cc8	cc9	cc10
00 add	IF	ID	EX	MEM	WB					
04 add	IF	ID	EX	MEM	WB					
08 bne		IF	ID	EX	MEM	WB				
0c add		IF	ID	EX	MEM	WB				
10 add			IF	ID	EX	MEM	WB			
14 add			IF	ID	EX	MEM	WB			
18 add				IF	ID	EX	MEM	WB		
1c add				IF	ID	EX	MEM	WB		

2-way superscalar processor executing instruction sequence with a branch

Because of the taken of a branch instruction, only one instruction is executed in cc4 and no instructions are executed in CC6 and CC7. This reduces the IPS.

Deeper pipeline with three ID stages

- Another approach is fetching the following instructions (an instruction at the next address and following ones) when a branch (bne) is fetched.

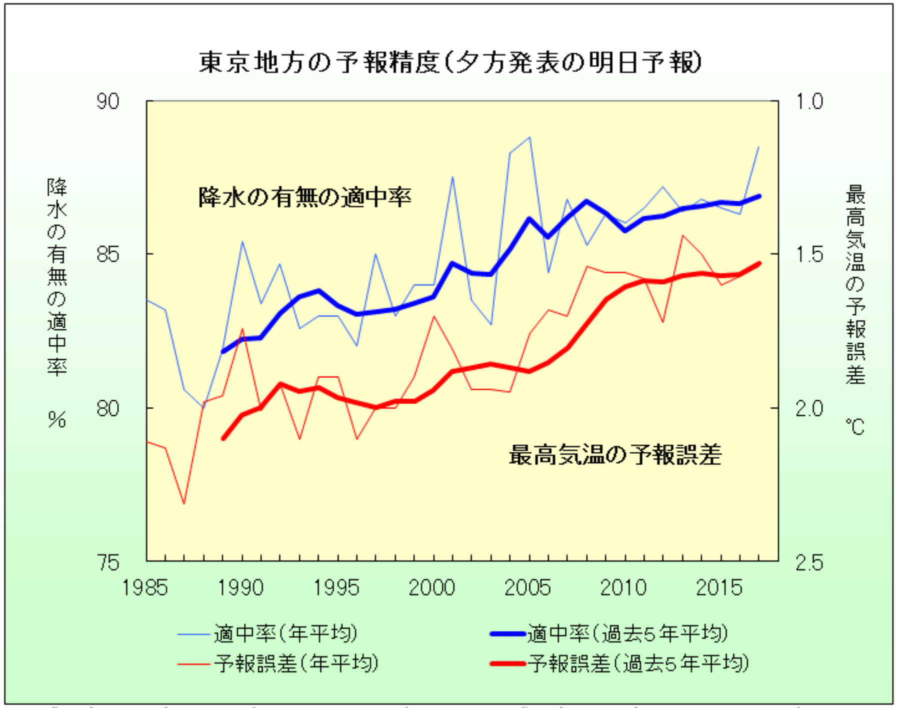
	cc1	cc2	cc3	cc4	cc5	cc6	cc7	cc8	cc9	cc10	cc11	cc12	cc13	cc14
00 add	IF	ID1	ID2	ID3	EX	MEM	WB							
04 add	IF	ID1	ID2	ID3	EX	MEM	WB							
08 bne		IF	ID1	ID2	ID3	EX	MEM	WB						
0c add		IF	ID1	ID2	ID3	EX	MEM	WB						
10 add			IF	ID1	ID2	ID3	EX	MEM	WB					
14 add			IF	ID1	ID2	ID3	EX	MEM	WB					
18 add				IF	ID1	ID2	ID3	EX	MEM	WB				
1c add				IF	ID1	ID2	ID3	EX	MEM	WB				
10 add					IF	ID1	ID2	ID3	EX	MEM	WB			
14 add					IF	ID1	ID2	ID3	EX	MEM	WB			
18 add						IF	ID1	ID2	ID3	EX	MEM	WB		
1c add						IF	ID1	ID2	ID3	EX	MEM	WB		

2-way superscalar adopting deeper pipeline executing instruction sequence with a branch

Branch predictor

- A branch predictor is a digital circuit that tries to guess or predict which way (taken or untaken) a branch will go before this is known definitively.
  - A random predictor will achieve about a 50% hit rate because the prediction output is 1 or 0.
  - Let's guess the accuracy. What is the accuracy of typical branch predictors for high-performance commercial processors?

Prediction Accuracy of weather forecasts



平成29年(2017年)までを表示しています。次の更新は平成31年(2019年)1月31日頃の予定です。

Tomorrow will be rainy?

年平均	北 海 道	東　北	関東甲信	東　海	北　陸	近　畿	中　国	四　国	九州北部	九州南部	沖　縄	全国平均
明　日	79	81	85	85	84	84	84	84	85	85	79	83
明後日	75	77	81	82	80	80	81	80	81	81	75	79
3日目	71	72	76	77	75	76	76	77	76	76	71	75
4日目	68	70	74	74	72	73	73	74	73	73	69	72
5日目	66	67	72	72	69	71	71	72	71	70	68	70
6日目	65	65	70	70	66	70	69	71	70	68	67	68
7日目	63	64	69	68	64	67	67	69	68	67	65	67
3～7日目平均	67	68	72	72	69	71	71	73	72	71	68	70

MLIT
Ministry of Land, Infrastructure, Transport and Tourism

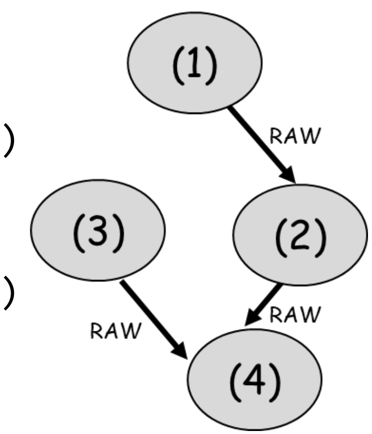
国土交通省
気象庁
Japan Meteorological Agency

Exploiting Instruction Level parallelism (ILP)

- A superscalar has to handle some flows efficiently to exploit ILP
  - Control flow (control dependence)
    - 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)
    - Prediction
    - Another obstacle is instruction cache
  - Register data flow (data dependence)
    - Out-of-order execution
      - Register renaming
      - Dynamic scheduling
  - Memory data flow
    - Out-of-order execution
    - Another obstacle is instruction cache

```
(1) add x5,x1,x2
(2) add x9,x5,x3
(3) lw  x4, 4(x7)
(4) add x8,x9,x4
```

```
(3) lw  x4, 4(x7)
(1) add x5,x1,x2
(2) add x9,x5,x3
(4) add x8,x9,x4
```



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 = 10
R5 = 2
R3 = R3 x R5     (1)
R4 = R3 + 1      (2)
R3 = R5 + 3      (3)
R7 = R3 + R4     (4)
```

```
20 = 10 x 2      (1)
21 = 20 + 1      (2)
5  = 2  + 3      (3)
26 = 5  + 21     (4)
```

wrong sequence

```
R3 = 10
R5 = 2
R3 = R3 x R5     (1)
R4 = R3 + 1      (2)
R7 = R3 + R4     (4)
R3 = R5 + 2      (3)
```

```
20 = 10 x 2      (1)
21 = 20 + 1      (2)
41 = 20 + 21     (4)
 5 = 2  + 3      (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.

```
R3 = 10
R5 = 2
R3 = R3 x R5     (1)
R4 = R3 + 1      (2)
R3 = R5 + 3      (3)
R7 = R3 + R4     (4)
```

```
20 = 10 x 2      (1)
21 = 20 + 1      (2)
5  = 2  + 3      (3)
26 = 5  + 21     (4)
```

wrong sequence

```
R3 = 10
R5 = 2
R3 = R5 + 3      (3)
R3 = R3 x R5     (1)
R4 = R3 + 1      (2)
R7 = R3 + R4     (4)
```

```
5  = 2  + 3      (3)
20 = 10 x 2      (1)
21 = 20 + 1      (2)
41 = 20 + 21     (4)
```

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 = 10
R5 = 2
R3 = R3 x R5        (1)
R4 = R3 + 1         (2)
R3 = R5 + 3         (3)
R7 = R3 + R4        (4)
```

```
20 = 10 x 2         (1)
21 = 20 + 1         (2)
5  = 2  + 3         (3)
26 = 5  + 21        (4)
```

wrong sequence

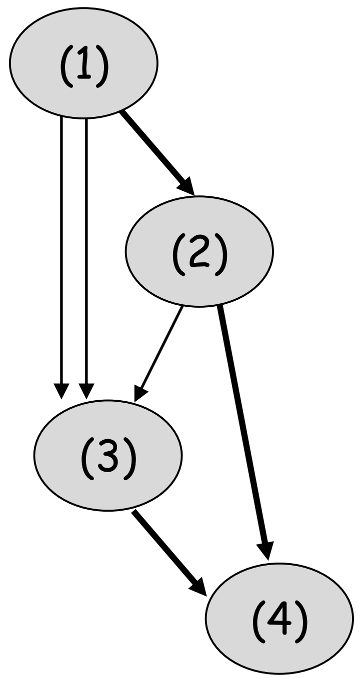
```
R3 = 10
R5 = 2
R3 = R3 x R5        (1)
R3 = R5 + 3         (3)
R4 = R3 + 1         (2)
R7 = R3 + R4        (4)
```

```
20 = 10 x 2         (1)
5  = 2  + 3         (3)
6  = 5  + 1         (2)
11 = 5  + 6         (4)
```

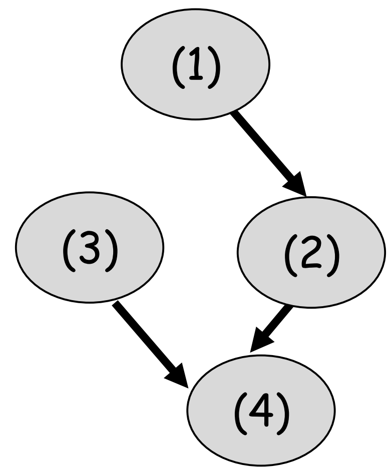
Data dependence and renaming

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

```
R3 = R3 x R5     (1)
R4 = R3 + 1      (2)
R3 = R5 + 3      (3)
R7 = R3 + R4     (4)
```




```
R3 = R3 x R5     (1)
R4 = R3 + 1      (2)
R8 = R5 + 3      (3)
R7 = R8 + R4     (4)
```



Hardware register renaming

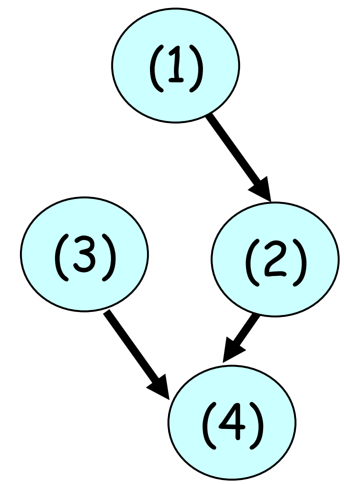
- Logical registers (architectural registers) which are ones defined by ISA
  - x0, x1, … x31
- Physical registers
  - Assuming plenty of registers are available, p0, p1, p2, …
- A processor renames (converts) each logical register to a unique physical register dynamically in the renaming stage

IF	ID	EX	MEM	WB

IF	ID	Renaming	Dispatch	Issue	Execute	Complete	Commit/ Retire

In-order and out-of-order (OoO) execution

- In in-order execution model, all instructions are executed in the order that they appear.
  This can lead to unnecessary stalls.
  - Instruction (3) stalls waiting for insn (2) to go first, even though it does not have a data dependence.
- With out-of-order execution,
  - Using register renaming to eliminate output dependence and antidependence, just having true data dependence
  - Dynamic scheduling: insn (3) is allowed to be executed before the insn (2)
    - Tomasulo algorithm
      (IBM System/360 Model 91 in 1967)



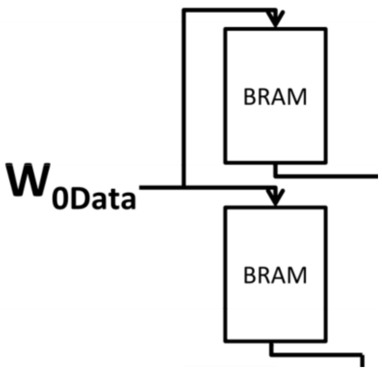

```
R3 = R3 x R5   (1)
R4 = R3 + 1    (2)
R3 = R5 + 2    (3)
R7 = R3 + R4   (4)
```

Data flow graph

Multi-Ported Memories (for FPGAs)

[8] C. E. LaForest and J. G. Steffan. Efficient Multi-ported Memories for FPGAs. In *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '10, pages 41–50, New York, NY, USA, 2010. ACM.

LVT (Live Value Tabele) design



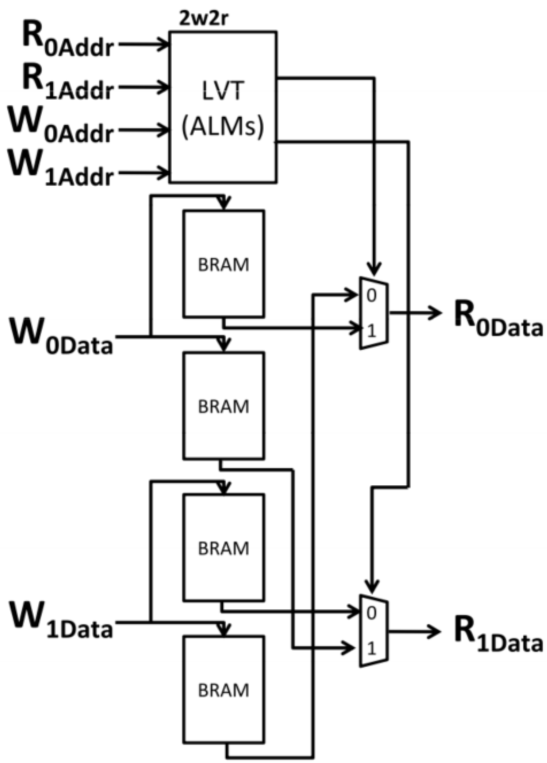


Figure 1: A 2W/2R Live Value Table (LVT) design.

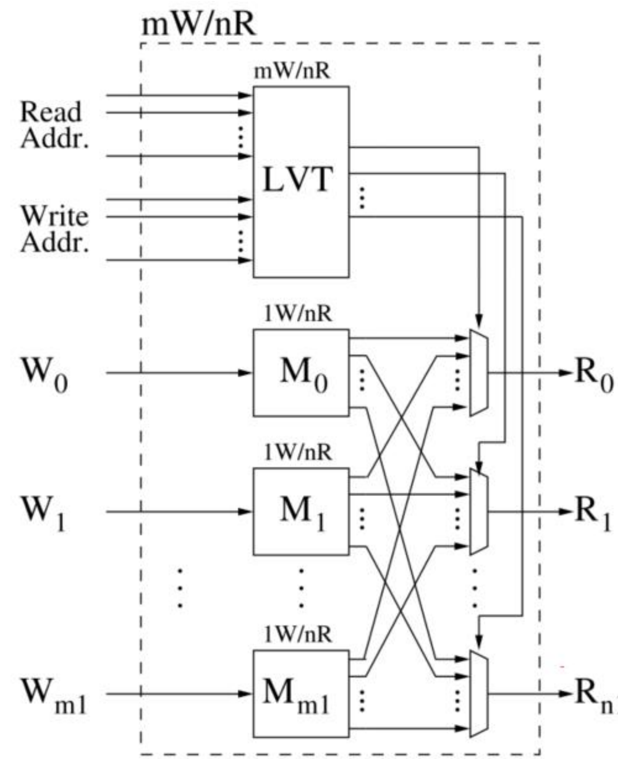


Figure 2: A generalized mW/nR memory implemented using a Live Value Table (LVT)