Fiscal Year 2022

Ver. 2022-12-26a

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

Advanced Computer Architecture

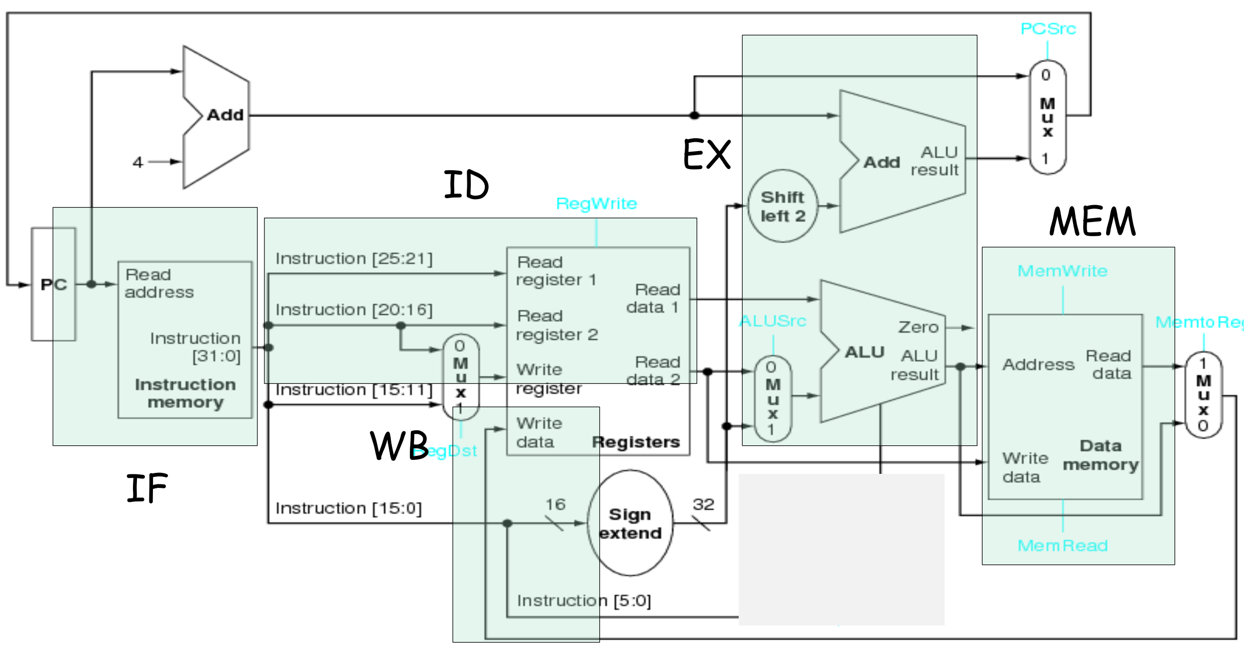
5. Instruction Level Parallelism: Concepts and Challenges

www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W831, HyFlex
Mon 13:45-15:25, Thr 13:45-15:25

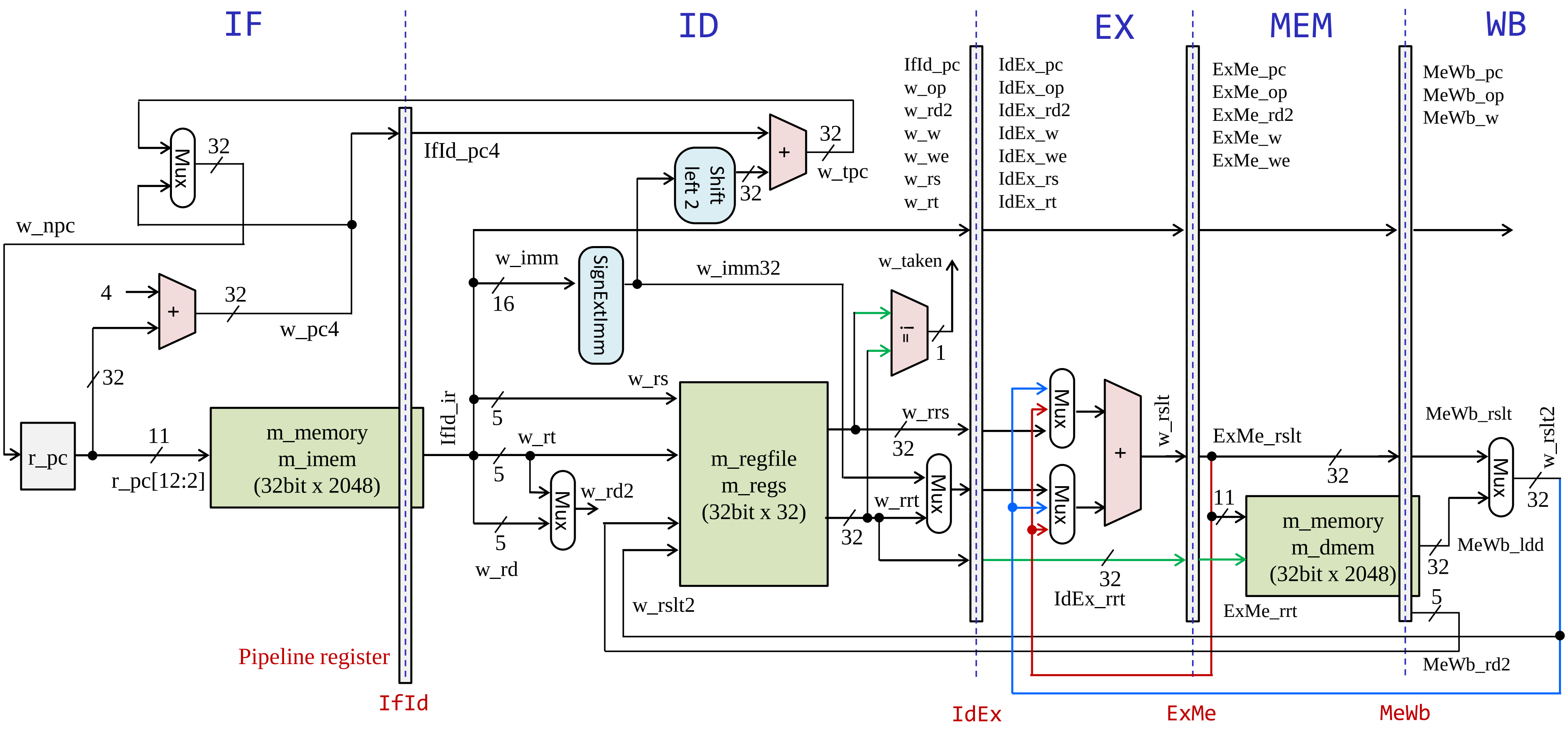
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

Conventional five steps (stages) of MIPS

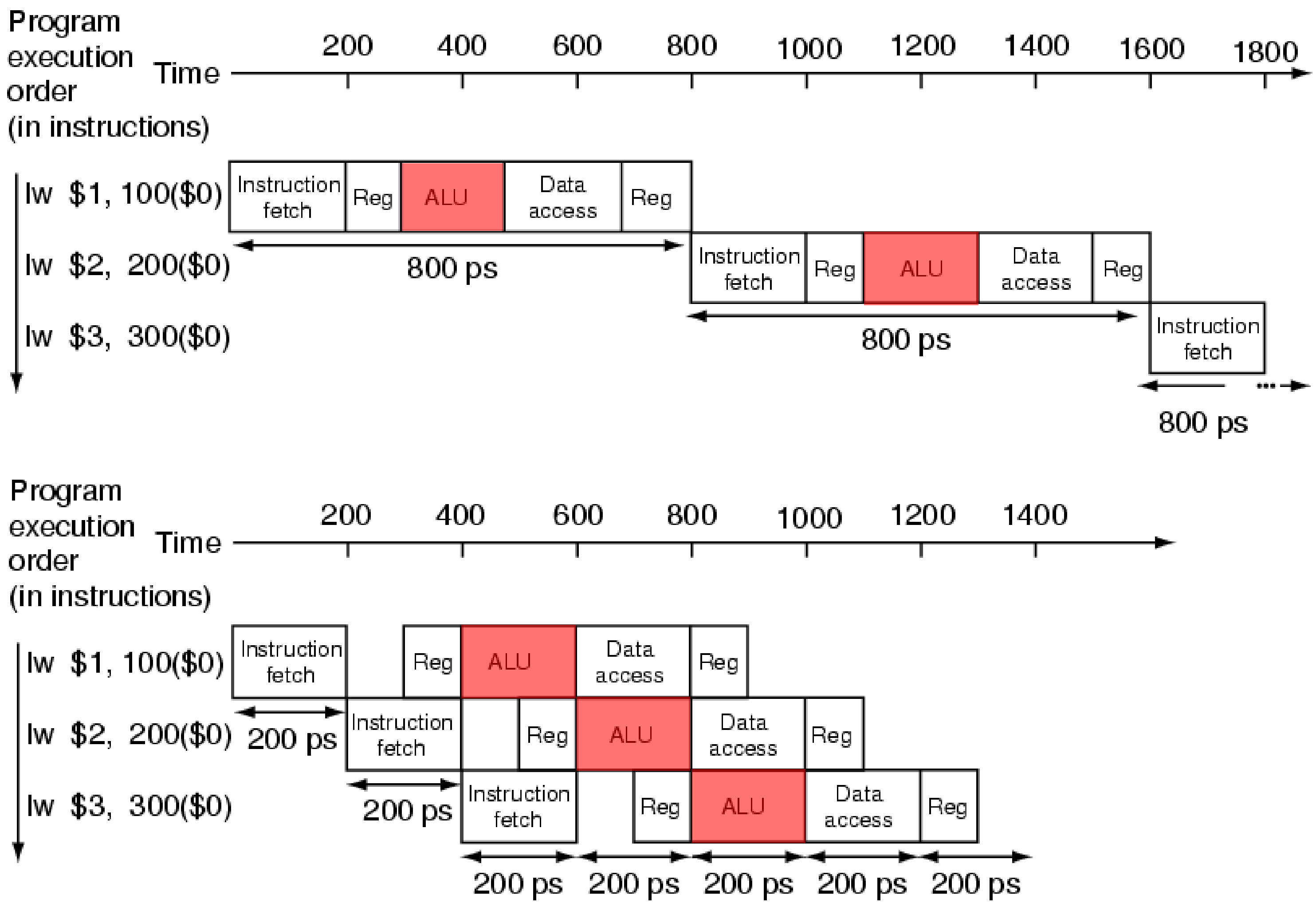
- IF: Instruction Fetch from instruction memory
- ID: Instruction Decode and operand fetch from regfile (register file)
- EX: EXecute operation or calculate address for load/store or calculate branch condition and target address
- MEM (MA): MEMory access for load/store
- WB: Write result Back to regfile



Pipelined MIPS processor with data forwarding

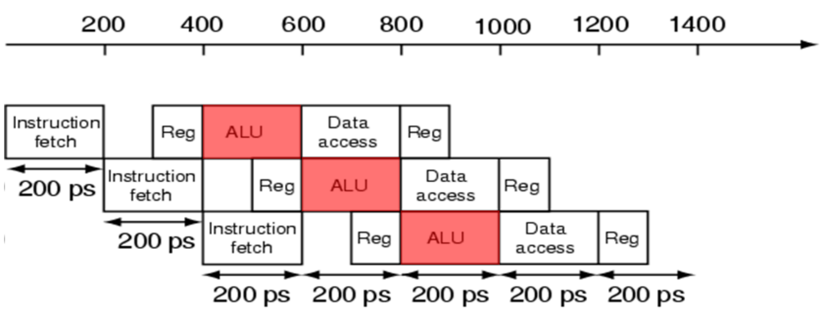


Single-cycle and pipelined processors

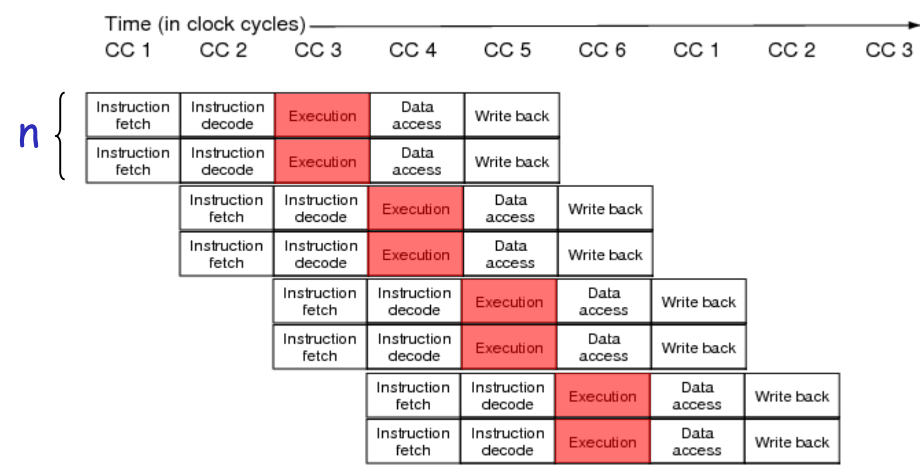


Scalar and Superscalar processors

- 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



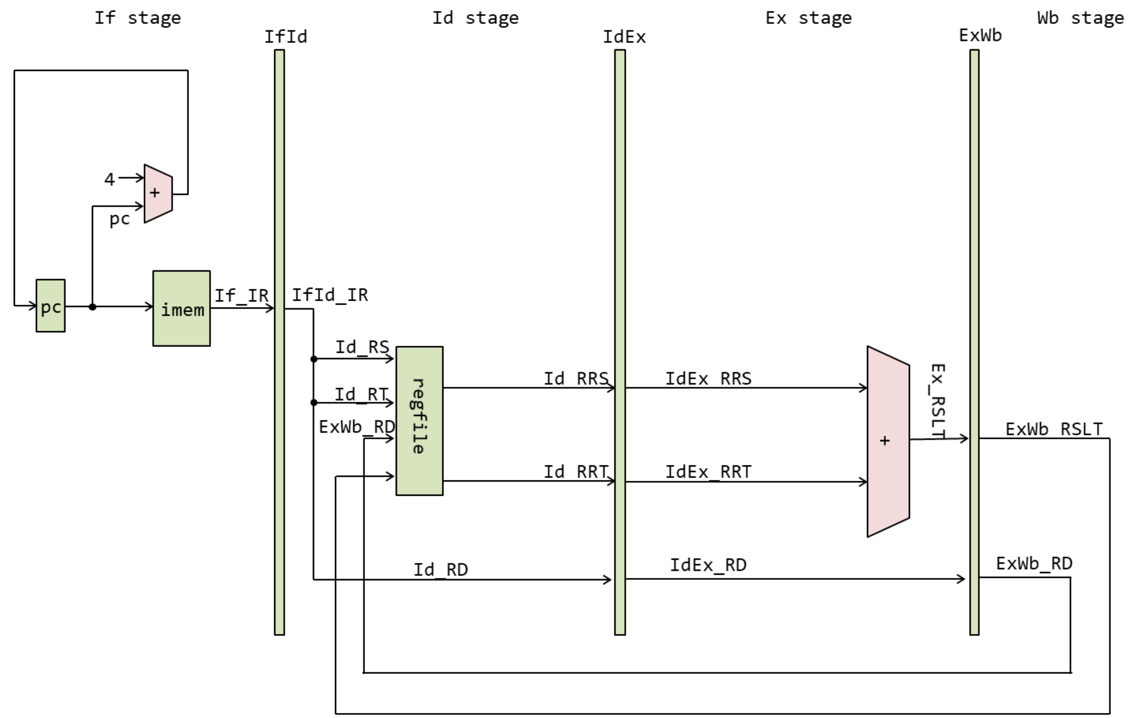
(a) pipeline diagram of scalar processor



(b) pipeline diagram of 2-way superscalar processor

Exercise: datapath of a 2-way superscalar

- Datapath of a 2-way superscalar processor supporting ADD, which does not adopt data forwarding

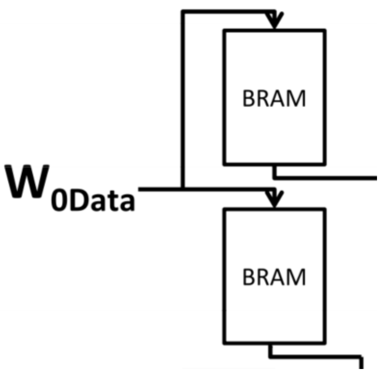


this slide is to be used as a whiteboard

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



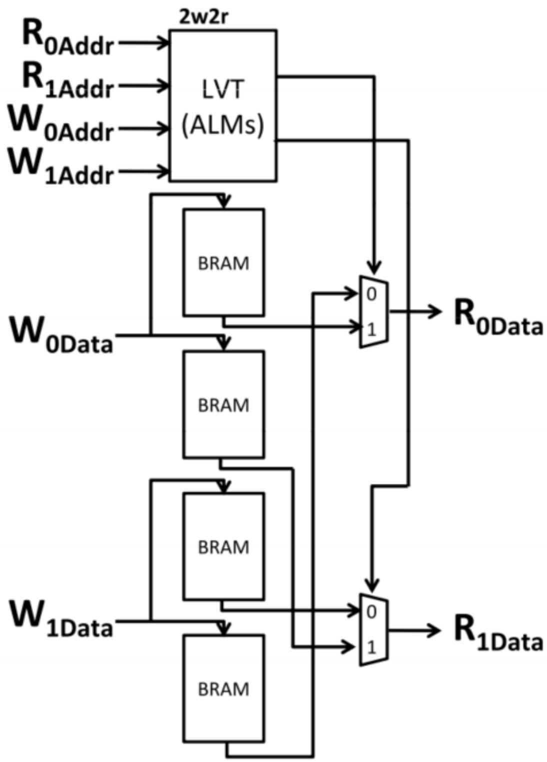


Figure 1: A 2W/2R Live Value Table (LVT) design.

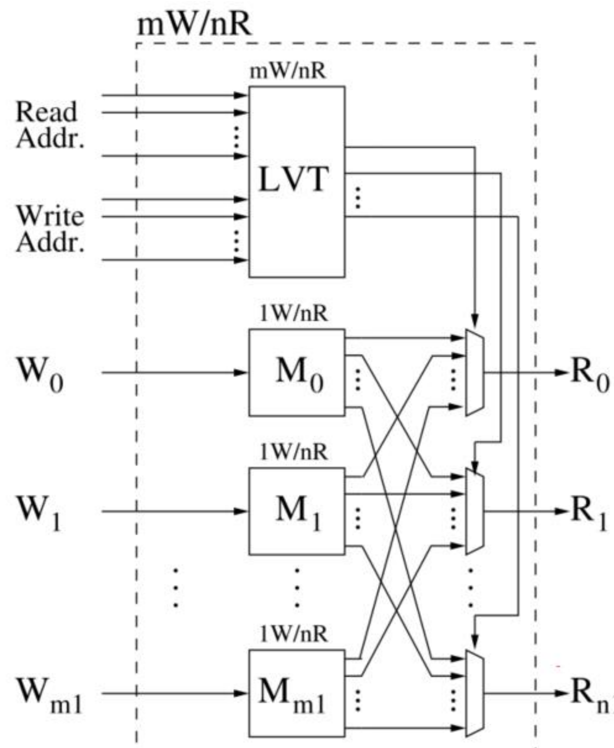


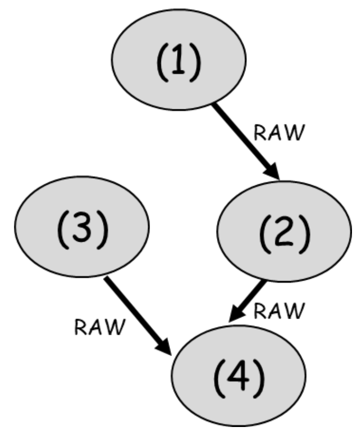
Figure 2: A generalized mW/nR memory implemented using a Live Value Table (LVT)

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, BEQ)
    - 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 $5,$1,$2
(2) add $9,$5,$3
(3) lw  $4, 4($7)
(4) add $8,$9,$4
```

```
(3) lw  $4, 4($7)
(1) add $5,$1,$2
(2) add $9,$5,$3
(4) add $8,$9,$4
```



MIPS Control Flow Instructions

- MIPS conditional branch instructions:

```
bne $s0, $s1, Lbl   # go to Lbl if $s0≠$s1
beq $s0, $s1, Lbl   # go to Lbl if $s0=$s1
```

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

```
    bne $s0, $s1, Lbl1
    add $s3, $s0, $s1
Lbl1:  ...
```

- Instruction Format (I format):

op	rs	rt	16 bit offset

- How is the branch destination address specified?

Why do branch instructions degrade IPC?

- The branch taken / untaken is determined in execution stage of the branch.
- The conservative approach of stalling instruction fetch until the branch direction is determined.

		cc1	cc2	cc3	cc4	cc5	cc6	cc7	cc8	cc9	cc10
1.	add	IF	ID	EX	MEM	WB					
2.	add	IF	ID	EX	MEM	WB					
3.	bne		IF	ID	EX	MEM	WB				
4.	add					IF	ID	EX	MEM	WB	
5.	add					IF	ID	EX	MEM	WB	
6.	add						IF	ID	EX	MEM	WB
7.	add						IF	ID	EX	MEM	WB

Control dependency

2-way superscalar processor executing instruction sequence with a branch

Note that because 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

- In conservative approach, IPC degradation will be significant by deeper pipeline

		cc1	cc2	cc3	cc4	cc5	cc6	cc7	cc8	cc9	cc10	cc11	cc12	cc13	cc14
1.	add	IF	ID1	ID2	ID3	EX	MEM	WB							
2.	add	IF	ID1	ID2	ID3	EX	MEM	WB							
3.	bne		IF	ID1	ID2	ID3	EX	MEM	WB						
4.	add							IF	ID1	ID2	ID3	EX	MEM	WB	
5.	add							IF	ID1	ID2	ID3	EX	MEM	WB	
6.	add								IF	ID1	ID2	ID3	EX	MEM	WB
7.	add								IF	ID1	ID2	ID3	EX	MEM	WB

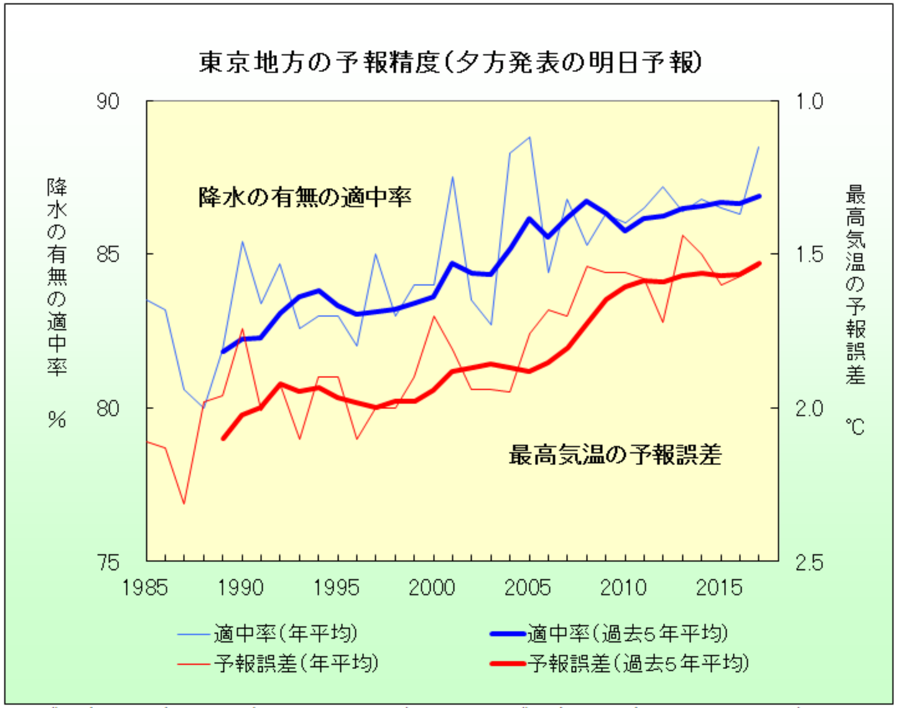
Control dependency

2-way superscalar adopting deeper pipeline executing instruction sequence with a branch this slide is to be used as a whiteboard

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

2018/05/16 17:46:57

印刷

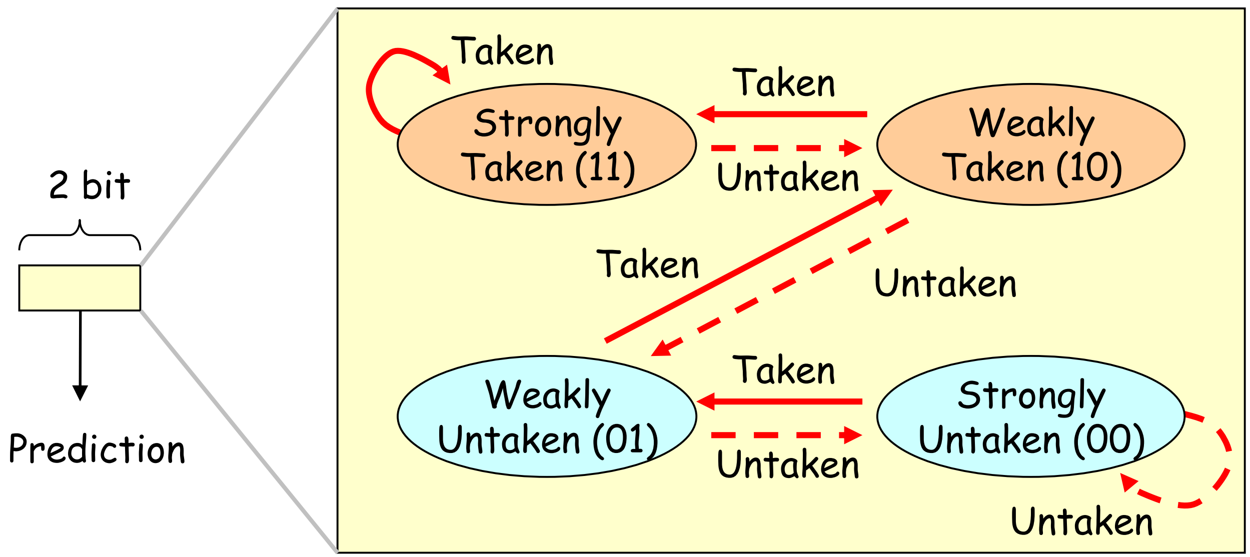
天気予報の予測精度向上に期待 - 気象庁が新スパコンを6月より稼動

マイナビニュース

国土交通省
気象庁
Japan Meteorological Agency

Simple branch predictor: 2bit counter

- It uses two bit register or a counter.
- How to update the register
  - If the branch outcome is taken and the value is not 3, then increment the register.
  - If the branch outcome is untaken and the value is not 0, then decrement the register.
- Hot to predict
  - It predicts as 1 if the MSB of the register is one, otherwise predicts as 0.



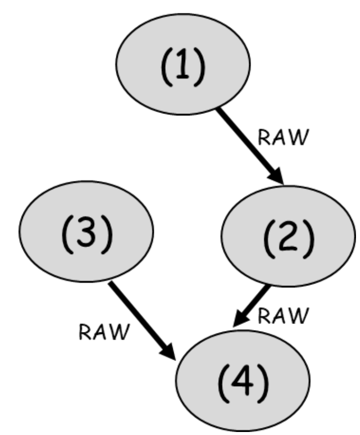
this slide is to be used as a whiteboard

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, BEQ)
    - 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 $5,$1,$2
(2) add $9,$5,$3
(3) lw  $4, 4($7)
(4) add $8,$9,$4
```

```
(3) lw  $4, 4($7)
(1) add $5,$1,$2
(2) add $9,$5,$3
(4) add $8,$9,$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 x R5     (1)
R4 = R3 + 1      (2)
R3 = R5 + 2      (3)
R7 = R3 + R4     (4)
```

```
Assume R3=10, R5=3
20 = 10 x 2      (1)
21 = 20 + 1      (2)
5  = 3  + 2      (3)
26 = 5  + 21     (4)
```

```
Assume R3=10, R5=3
20 = 10 x 2      (1)
21 = 20 + 1      (2)
41 = 20 + 21     (4)
   = 3  + 2      (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 = R3 x R5      (1)
R4 = R3 + 1       (2)
R3 = R5 + 2       (3)
R7 = R3 + R4      (4)
```

```
Assume R3=10, R5=3           Assume R3=10, R5=3
20 = 10 x 2       (1)        5  = 3  + 2       (3)
21 = 20 + 1       (2)        20 = 10 x 2       (1)
5  = 3  + 2       (3)        21 =    + 1       (2)
26 = 5  + 21      (4)        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 = R3 x R5     (1)
R4 = R3 + 1      (2)
R3 = R5 + 2      (3)
R7 = R3 + R4     (4)
```

```
Assume R3=10, R5=3
20 = 10 x 2      (1)
21 = 20 + 1      (2)
5  = 3  + 2      (3)
26 = 5  + 21     (4)
```

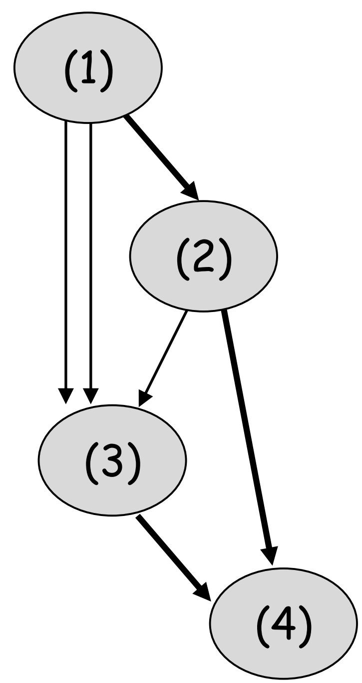
```
Assume R3=10, R5=3
20 = 10 x 2      (1)
5  = 3  + 2      (3)
6  = 5  + 1      (2)
11 = 5  + 6      (4)
```

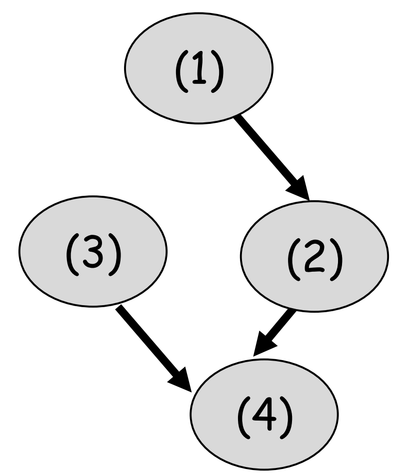
Data dependence and renaming

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

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

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





Hardware register renaming

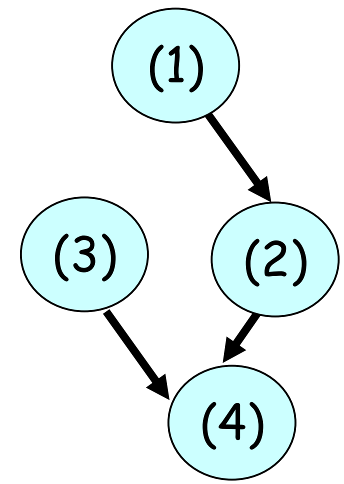
- Logical registers (architectural registers) which are ones defined by ISA
  - $0, $1, … $31
- Physical registers
  - *Assuming plenty of registers are available, p0, p1, p2, …*
- *A processor renames (converts) each logical register to a unique physical register dynamically in the renaming stage*

IF	ID	EX	MEM	WB

IF	ID	Renaming	Dispatch	Issue	Execute	Complete	Commit/ Retire

Out-of-order 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
  - insn (3) is allowed to be executed before the insn (2)
    - Scoreboarding (CDC6600 in 1964)
    - 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

Dynamic scheduling

this slide is to be used as a whiteboard