

# **Seminar 2**

- 1. Instruction Pipelining**
- 2. Superscalars**
- 3. Parallel Architectures**

# Problem 1

A nonpipelined processor has a clock rate of 2.5 GHz. An upgrade to the processor introduces a five-stage pipeline. However, due to internal pipeline delays, the clock rate of the new processor has to be reduced to 2 GHz.

What is the speedup achieved for a sequence of 100 instructions?

# Problem 1

A nonpipelined processor has a clock rate of 2.5 GHz. An upgrade to the processor introduces a five-stage pipeline. However, due to internal pipeline delays, the clock rate of the new processor has to be reduced to 2 GHz.

What is the speedup achieved for a sequence of 100 instructions?

Remember from Lecture 4-5:

- $\tau$ : duration of one cycle
- $n$ : number of instructions to execute
- $k$ : number of pipeline stages
- $T_{k,n}$ : total time to execute  $n$  instructions on a pipeline with  $k$  stages
- $S_{k,n}$ : (theoretical) speedup produced by a pipeline with  $k$  stages when executing  $n$  instructions

$$T_{k,n} = [k + (n - 1)] \times \tau$$

On a non-pipelined processor each instruction takes  $k \times \tau$ ,  
and  $n$  instructions take  $T_n = n \times k \times \tau$

# Problem 1

A nonpipelined processor has a clock rate of 2.5 GHz. An upgrade to the processor introduces a five-stage pipeline. However, due to internal pipeline delays, the clock rate of the new processor has to be reduced to 2 GHz.

What is the speedup achieved for a sequence of 100 instructions?

Remember from Lecture 4-5:

- $\tau$ : duration of one cycle
- $n$ : number of instructions to execute
- $k$ : number of pipeline stages
- $T_{k,n}$ : total time to execute  $n$  instructions on a pipeline with  $k$  stages
- $S_{k,n}$ : (theoretical) speedup produced by a pipeline with  $k$  stages when executing  $n$  instructions

$$T_{k,n} = [k + (n - 1)] \times \tau$$

On a non-pipelined processor each instruction takes  $k \times \tau$ ,  
and  $n$  instructions take  $T_n = n \times k \times \tau$

$$S_{k,n} = \frac{T_n}{T_{k,n}} = \frac{n \times k \times \tau}{[k + (n - 1)] \times \tau} = \frac{n \times k}{k + (n - 1)}$$

# Problem 1

A nonpipelined processor has a clock rate of 2.5 GHz. An upgrade to the processor introduces a five-stage pipeline. However, due to internal pipeline delays, the clock rate of the new processor has to be reduced to 2 GHz.

What is the speedup achieved for a sequence of 100 instructions?

## Solution

$$S_{5,100} = \frac{100 \times 5}{5 + (100 - 1)} = 4,8$$

We would have a speedup of 4.8 if the pipelined processor would work at the same clock rate as the initial one!

But the clock rate of the pipelined processor is reduced by a factor of  $2/2.5 = 0.8$



$$S = 4.8 \times 0.8 = 3.8$$

# Problem 2

Consider the following assembly language program:


1: Move R3, R7	$R3 \leftarrow R7$
2: Load R8, (R3)	$R8 \leftarrow (R3)$
3: Add R3, R3, 4	$R3 \leftarrow R3 + 4$
4: Load R9, (R3)	$R9 \leftarrow (R3)$
5: BLE R8, R9, L3	Branch if $R9 > R8$

Show the dependencies.

# Problem 2

Consider the following assembly language program:

1: Move R3, R7	R3 ← R7
2: Load R8, (R3)	R8 ← (R3)
3: Add R3, R3, 4	R3 ← R3 + 4
4: Load R9, (R3)	R9 ← (R3)
5: BLE R8, R9, L3	Branch if R9 > R8



Show the dependencies.

# Problem 2

Consider the following assembly language program:



1: Move R3, R7	R3 ← R7
2: Load R8, (R3)	R8 ← (R3)
3: Add R3, R3, 4	R3 ← R3 + 4
4: Load R9, (R3)	R9 ← (R3)
5: BLE R8, R9, L3	Branch if R9 > R8

Show the dependencies.



# Problem 2

Consider the following assembly language program:



1: Move R3, R7	R3 ← R7
2: Load R8, (R3)	R8 ← (R3)
3: Add R3, R3, 4	R3 ← R3 + 4
4: Load R9, (R3)	R9 ← (R3)
5: BLE R8, R9, L3	Branch if R9 > R8

Show the dependencies.

# Problem 2

Consider the following assembly language program:

1: Move R3, R7	R3 ← R7	
2: Load R8, (R3)	R8 ← (R3)	
3: Add R3, R3, 4	R3 ← R3 + 4	↘
4: Load R9, (R3)	R9 ← (R3)	
5: BLE R8, R9, L3	Branch if R9 > R8	

Show the dependencies.

# Problem 2

Consider the following assembly language program:



Show the dependencies.

# Problem 2

Consider the following assembly language program:



Show the dependencies.

True data dependency (RAW): 1 - 2, 1 - 3, 2 - 5, 3 - 4, 4 - 5

Output dependency (WAW): 1 - 3

Antidependency (WAR): 2 - 3

# Problem 3

a. Identify the dependencies in the following code:

```
1: R1 ← 100
2: R5 ← R1 + R2
3: R7 ← R5 + 1
4: R1 ← R2 + R4
5: R5 ← 0
6: R2 ← R4 - 25
7: R3 ← R7 - 2
8: R4 ← R1 + R3
9: R10 ← 0
10: R1 ← R1 + 30
```

- b. Rename the registers in the above sequence to prevent, where possible, dependency problems.
- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:
- c1) in order execution;
  - c2) out of order execution before renaming;
  - c3) out of order execution after renaming.

# Problem 3

a. Identify the dependencies in the following code:

1:  $R1 \leftarrow 100$   
2:  $R5 \leftarrow R1 + R2$   
3:  $R7 \leftarrow R5 + 1$   
4:  $R1 \leftarrow R2 + R4$   
5:  $R5 \leftarrow 0$   
6:  $R2 \leftarrow R4 - 25$   
7:  $R3 \leftarrow R7 - 2$   
8:  $R4 \leftarrow R1 + R3$   
9:  $R10 \leftarrow 0$   
10:  $R1 \leftarrow R1 + 30$

# Problem 3

a. Identify the dependencies in the following code:

1:  $R1 \leftarrow 100$

2:  $R5 \leftarrow R1 + R2$

3:  $R7 \leftarrow R5 + 1$

4:  $R1 \leftarrow R2 + R4$

5:  $R5 \leftarrow 0$

6:  $R2 \leftarrow R4 - 25$

7:  $R3 \leftarrow R7 - 2$

8:  $R4 \leftarrow R1 + R3$

9:  $R10 \leftarrow 0$

10:  $R1 \leftarrow R1 + 30$

# Problem 3

a. Identify the dependencies in the following code:

```
1: R1 ← 100
2: R5 ← R1 + R2
3: R7 ← R5 + 1
4: R1 ← R2 + R4
5: R5 ← 0
6: R2 ← R4 - 25
7: R3 ← R7 - 2
8: R4 ← R1 + R3
9: R10 ← 0
10: R1 ← R1 + 30
```



# Problem 3

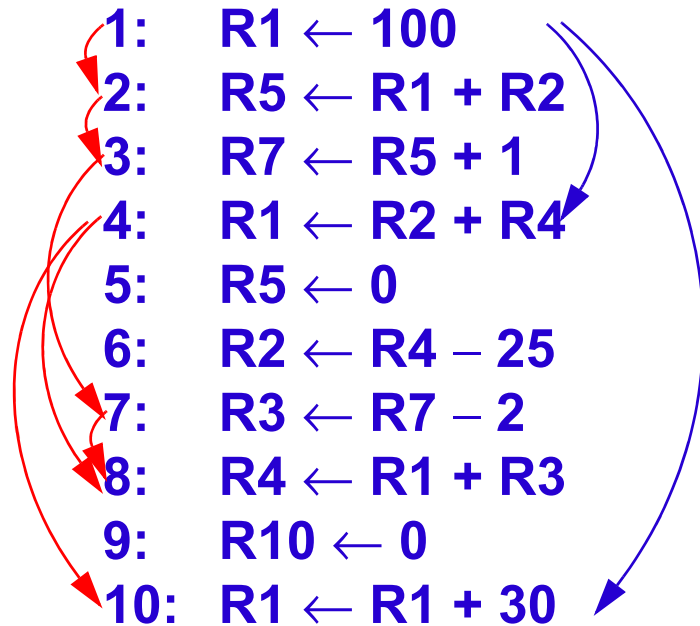
a. Identify the dependencies in the following code:

```
1: R1 ← 100
2: R5 ← R1 + R2
3: R7 ← R5 + 1
4: R1 ← R2 + R4
5: R5 ← 0
6: R2 ← R4 - 25
7: R3 ← R7 - 2
8: R4 ← R1 + R3
9: R10 ← 0
10: R1 ← R1 + 30
```

True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

# Problem 3

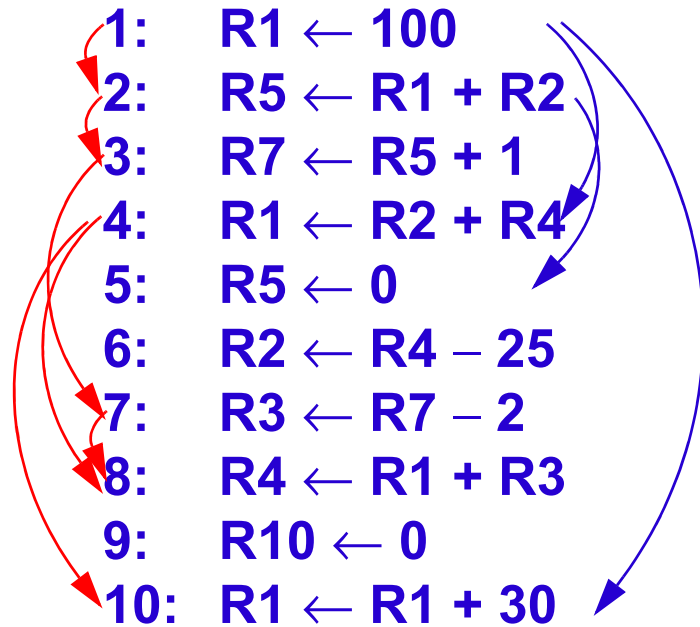
a. Identify the dependencies in the following code:



True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

# Problem 3

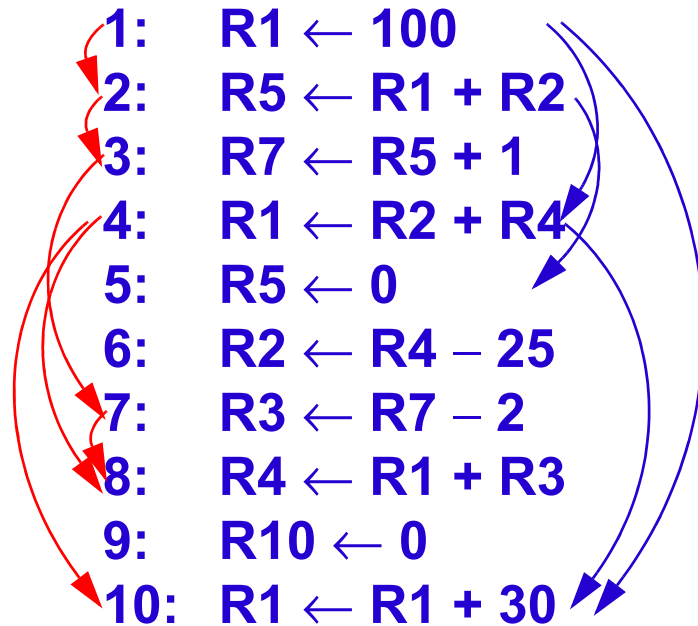
a. Identify the dependencies in the following code:



True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

# Problem 3

a. Identify the dependencies in the following code:

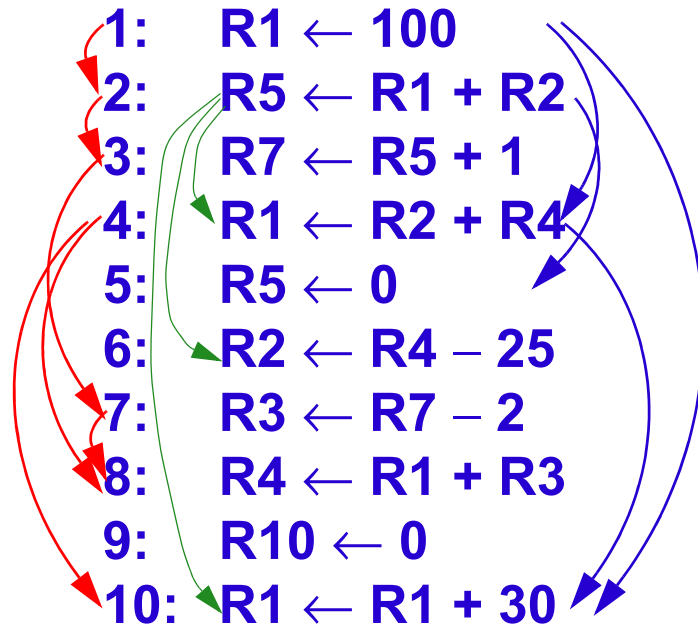


True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

# Problem 3

a. Identify the dependencies in the following code:

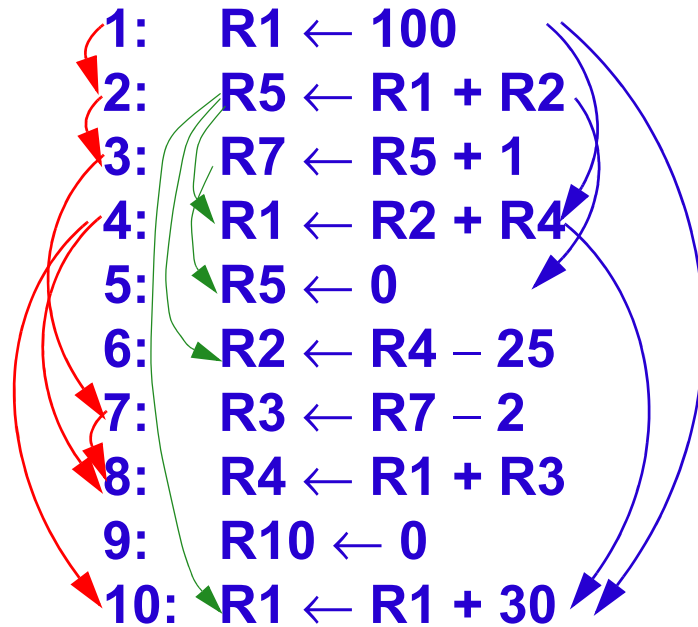


True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

# Problem 3

a. Identify the dependencies in the following code:

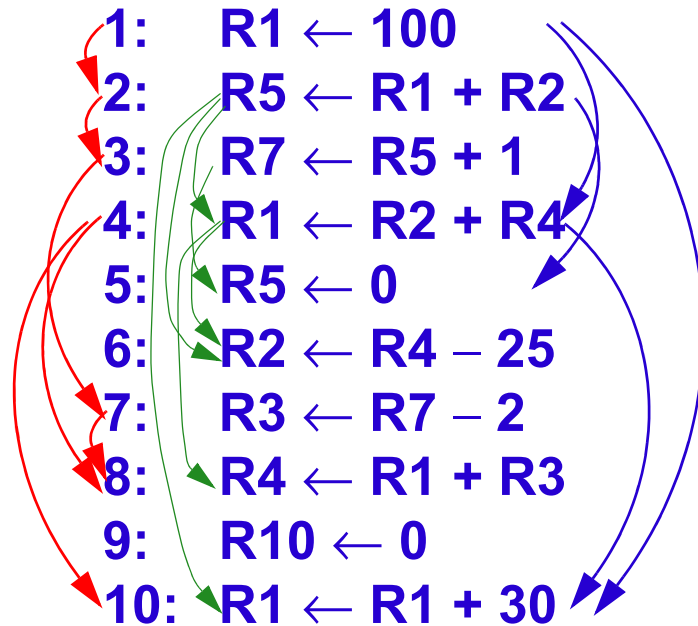


True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

# Problem 3

a. Identify the dependencies in the following code:

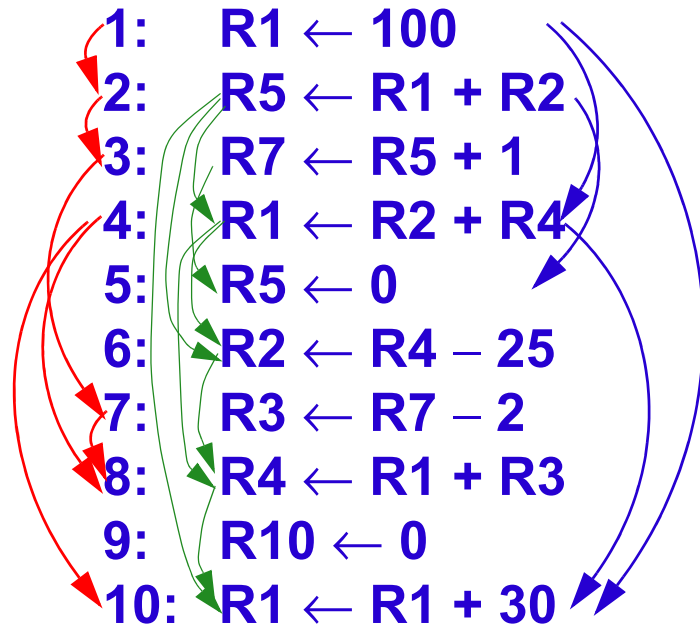


True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

# Problem 3

a. Identify the dependencies in the following code:



True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

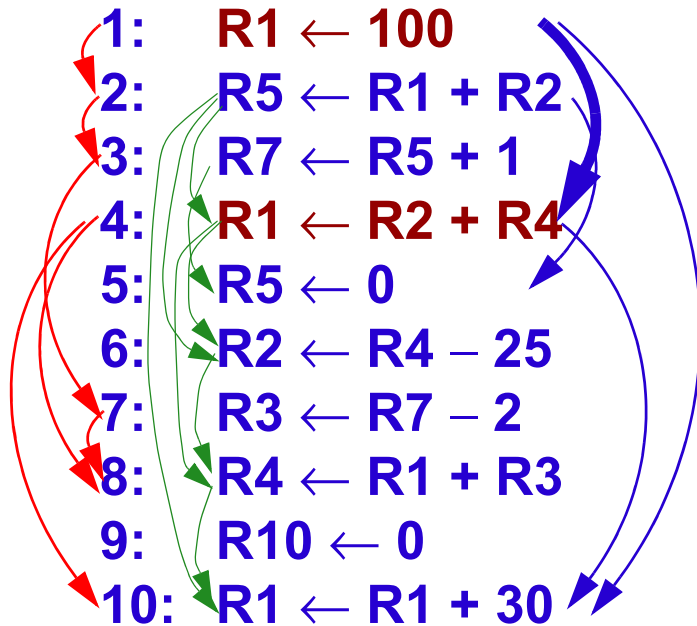
Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

Antidependency (WAR): 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10



# Problem 3

b. Rename registers to prevent output - and antidependencies:



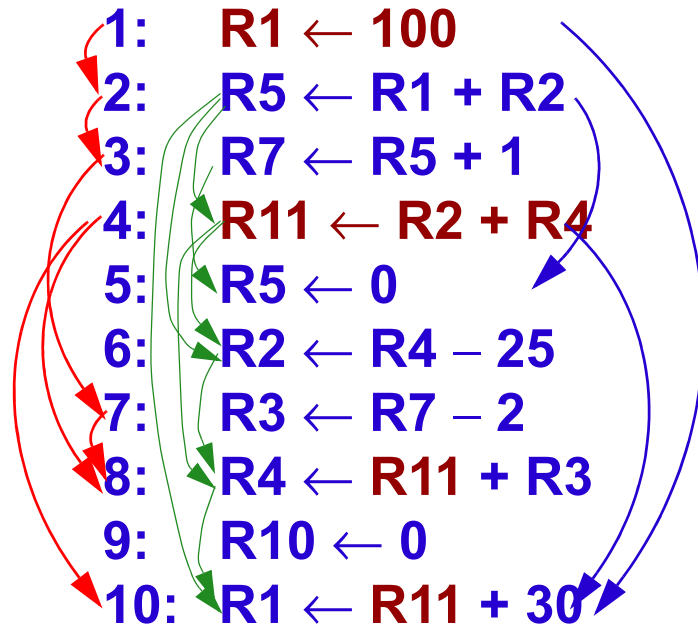
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

Antidependency (WAR): 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10

# Problem 3

b. Rename registers to prevent output - and antidependencies:



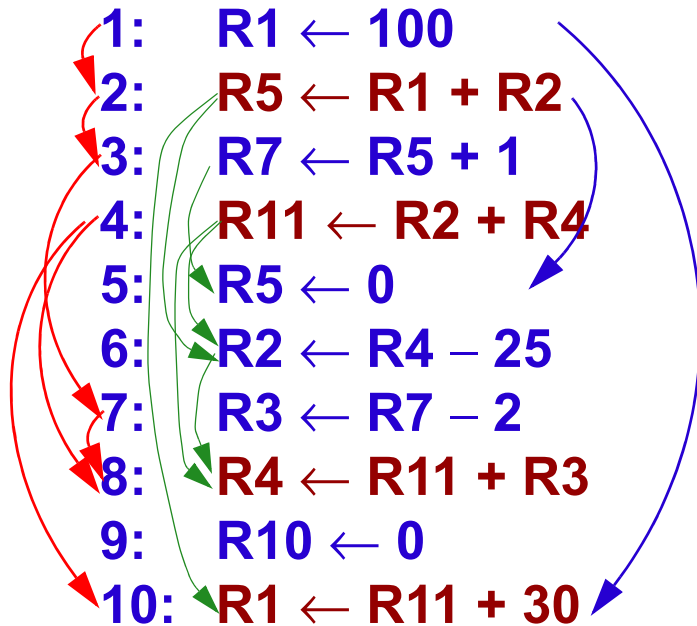
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, 1 - 10, 2 - 5, 4 - 10.

Antidependency (WAR): 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10

# Problem 3

b. Rename registers to prevent output - and antidependencies:



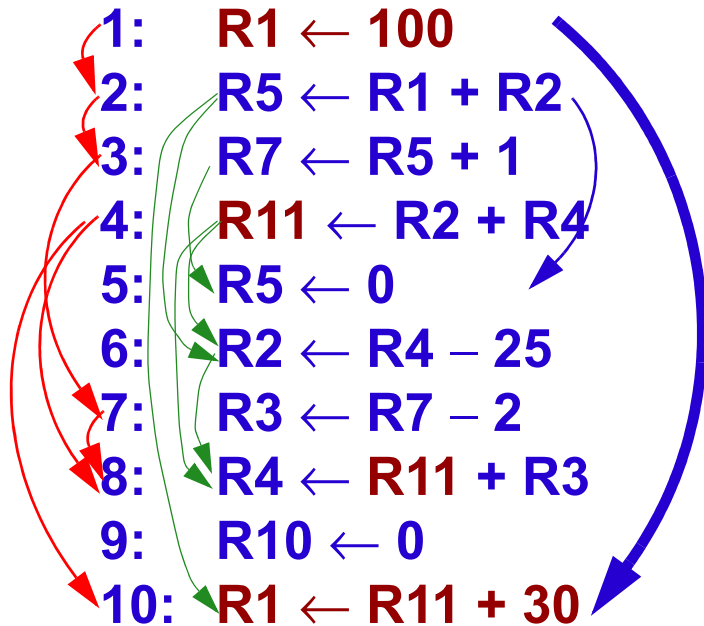
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, 1 - 10, 2 - 5, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, ~~8 - 10~~

# Problem 3

b. Rename registers to prevent output - and antidependencies:



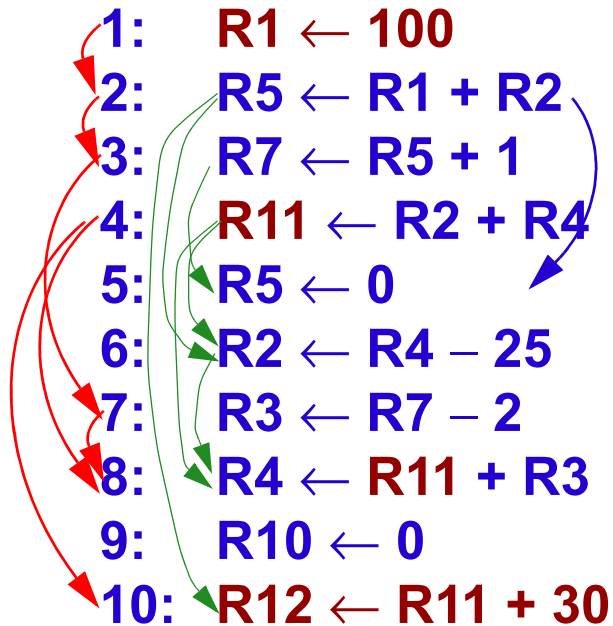
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, 1 - 10, 2 - 5, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, ~~8 - 10~~

# Problem 3

b. Rename registers to prevent output - and antidependencies:



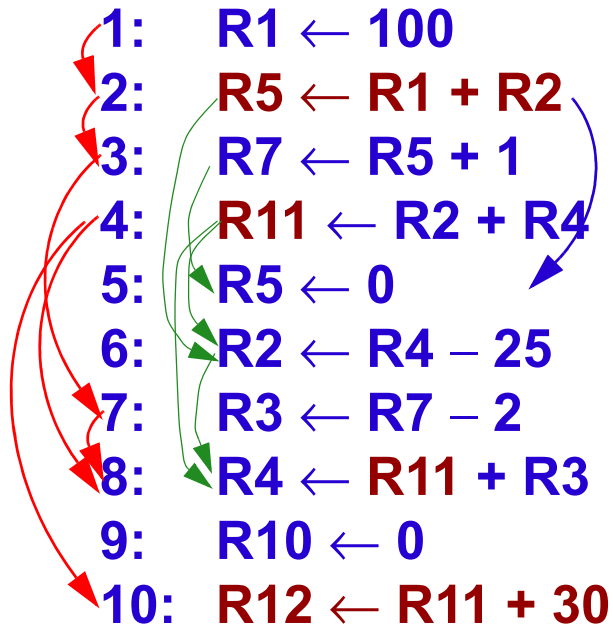
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, 2 - 5, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, ~~8 - 10~~

# Problem 3

b. Rename registers to prevent output - and antidependencies:



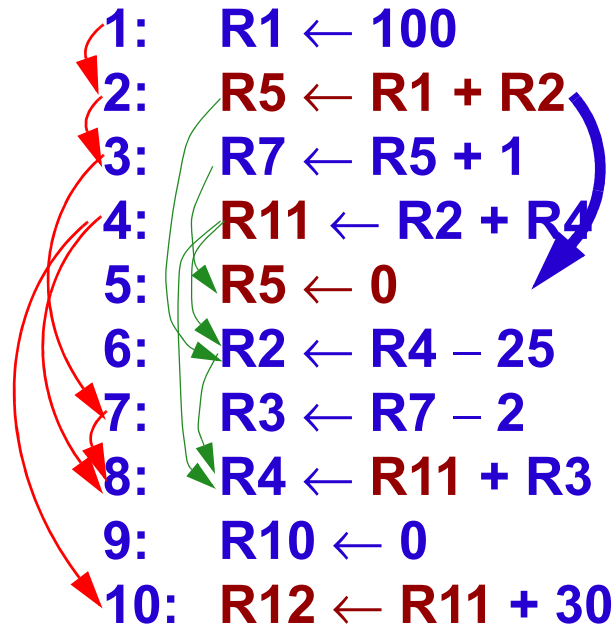
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, 2 - 5, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, ~~8 - 10~~

# Problem 3

b. Rename registers to prevent output - and antidependencies:



True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, 2 - 5, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, ~~8 - 10~~

# Problem 3

b. Rename registers to prevent output - and antidependencies:

1: R1 ← 100  
2: R5 ← R1 + R2  
3: R7 ← R5 + 1  
4: R11 ← R2 + R4  
5: R51 ← 0  
6: R2 ← R4 - 25  
7: R3 ← R7 - 2  
8: R4 ← R11 + R3  
9: R10 ← 0  
10: R12 ← R11 + 30

True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, ~~2 - 5~~, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, ~~8 - 10~~



# Problem 3

b. Rename registers to prevent output - and antidependencies:

1: R1 ← 100  
2: R5 ← R1 + R2  
3: R7 ← R5 + 1  
4: R11 ← R2 + R4  
5: R51 ← 0  
6: R2 ← R4 - 25  
7: R3 ← R7 - 2  
8: R4 ← R11 + R3  
9: R10 ← 0  
10: R12 ← R11 + 30

True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, ~~2 - 5~~, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, 2 - 6, ~~3 - 5~~, 4 - 6, 4 - 8, 6 - 8, ~~8 - 10~~

# Problem 3

b. Rename registers to prevent output - and antidependencies:

1: R1 ← 100  
2: R5 ← R1 + R2  
3: R7 ← R5 + 1  
4: R11 ← R2 + R4  
5: R51 ← 0  
6: R2 ← R4 - 25  
7: R3 ← R7 - 2  
8: R4 ← R11 + R3  
9: R10 ← 0  
10: R12 ← R11 + 30

True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, ~~2 - 5~~, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, ~~2 - 6~~, ~~3 - 5~~, ~~4 - 6~~, ~~4 - 8~~, ~~6 - 8~~, ~~8 - 10~~

# Problem 3

b. Rename registers to prevent output - and antidependencies:

1: R1 ← 100  
2: R5 ← R1 + R2  
3: R7 ← R5 + 1  
4: R11 ← R2 + R4  
5: R51 ← 0  
6: R21 ← R4 - 25  
7: R3 ← R7 - 2  
8: R4 ← R11 + R3  
9: R10 ← 0  
10: R12 ← R11 + 30

True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, ~~2 - 5~~, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, ~~2 - 6~~, ~~3 - 5~~, 4 - 6, 4 - 8, 6 - 8, ~~8 - 10~~

# Problem 3

b. Rename registers to prevent output - and antidependencies:

1: R1 ← 100  
2: R5 ← R1 + R2  
3: R7 ← R5 + 1  
4: R11 ← R2 + R4  
5: R51 ← 0  
6: R21 ← R4 - 25  
7: R3 ← R7 - 2  
8: R4 ← R11 + R3  
9: R10 ← 0  
10: R12 ← R11 + 30

True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, ~~2 - 5~~, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, ~~2 - 6~~, ~~3 - 5~~, ~~4 - 6~~, 4 - 8, 6 - 8, ~~8 - 10~~

# Problem 3

b. Rename registers to prevent output - and antidependencies:

1: R1 ← 100  
2: R5 ← R1 + R2  
3: R7 ← R5 + 1  
4: R11 ← R2 + R4  
5: R51 ← 0  
6: R21 ← R4 - 25  
7: R3 ← R7 - 2  
8: R4 ← R11 + R3  
9: R10 ← 0  
10: R12 ← R11 + 30

True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

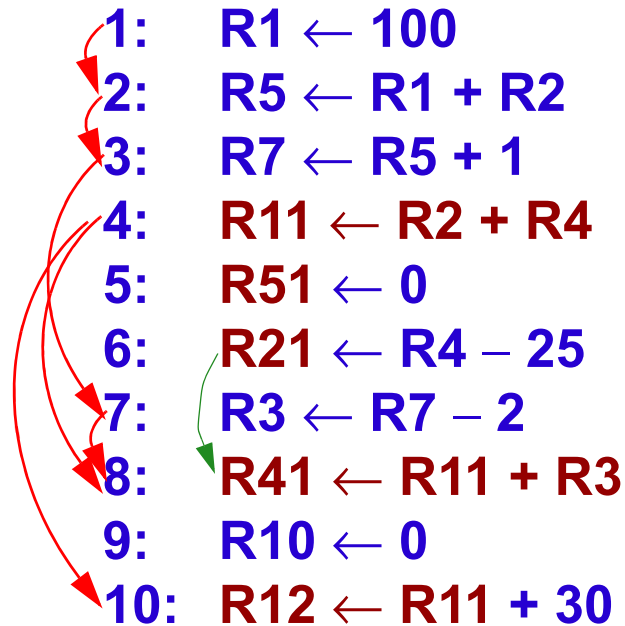
Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, ~~2 - 5~~, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, ~~2 - 6~~, ~~3 - 5~~, ~~4 - 6~~, 4 - 8, 6 - 8, ~~8 - 10~~

# Problem 3

b. Rename registers to prevent output - and antidependencies:

1: R1 ← 100  
2: R5 ← R1 + R2  
3: R7 ← R5 + 1  
4: R11 ← R2 + R4  
5: R51 ← 0  
6: R21 ← R4 - 25  
7: R3 ← R7 - 2  
8: R41 ← R11 + R3  
9: R10 ← 0  
10: R12 ← R11 + 30



True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, ~~2 - 5~~, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, ~~2 - 6~~, ~~3 - 5~~, ~~4 - 6~~, ~~4 - 8~~, 6 - 8, ~~8 - 10~~

# Problem 3

b. Rename registers to prevent output - and antidependencies:

1: R1 ← 100  
2: R5 ← R1 + R2  
3: R7 ← R5 + 1  
4: R11 ← R2 + R4  
5: R51 ← 0  
6: R21 ← R4 - 25  
7: R3 ← R7 - 2  
8: R41 ← R11 + R3  
9: R10 ← 0  
10: R12 ← R11 + 30

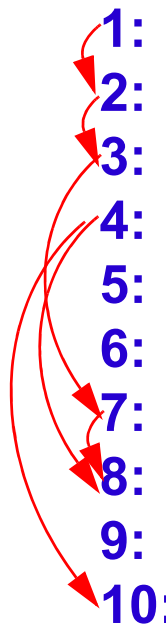
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, ~~2 - 5~~, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, ~~2 - 6~~, ~~3 - 5~~, ~~4 - 6~~, ~~4 - 8~~, ~~6 - 8~~, ~~8 - 10~~

# Problem 3

b. Rename registers to prevent output - and antidependencies:



1: R1  $\leftarrow$  100  
2: R5  $\leftarrow$  R1 + R2  
3: R7  $\leftarrow$  R5 + 1  
4: **R11**  $\leftarrow$  R2 + R4  
5: **R51**  $\leftarrow$  0  
6: **R21**  $\leftarrow$  R4 - 25  
7: R3  $\leftarrow$  R7 - 2  
8: **R41**  $\leftarrow$  **R11** + R3  
9: R10  $\leftarrow$  0  
10: **R12**  $\leftarrow$  **R11** + 30

True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

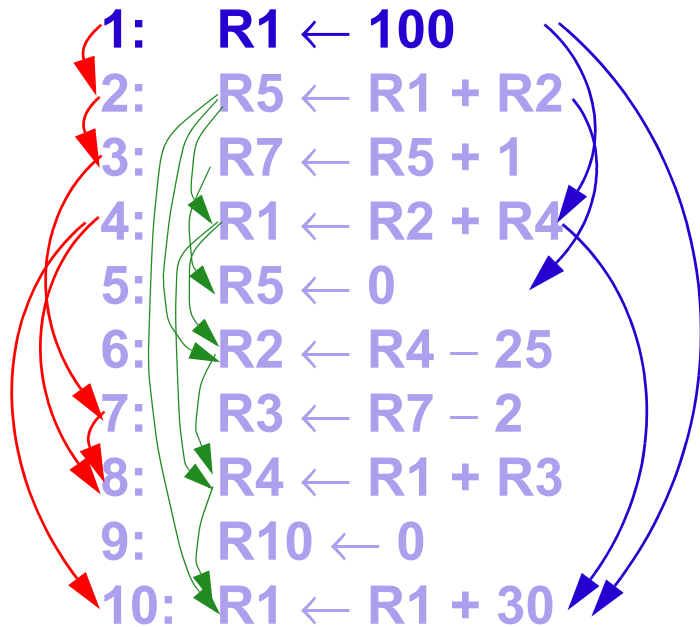
Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, ~~2 - 5~~, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, ~~2 - 6~~, ~~3 - 5~~, ~~4 - 6~~, ~~4 - 8~~, ~~6 - 8~~, ~~8 - 10~~



# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:  
c1) in order execution;



	ALU	ALU
Cycle 1	1	

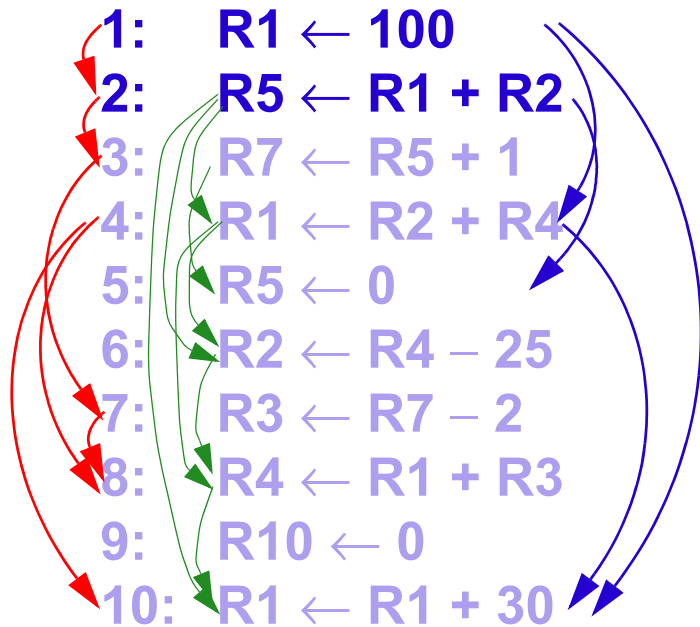
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

Antidependency (WAR): 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10

# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:
- c1) in order execution;



	ALU	ALU
Cycle 1	1	
Cycle 2	2	

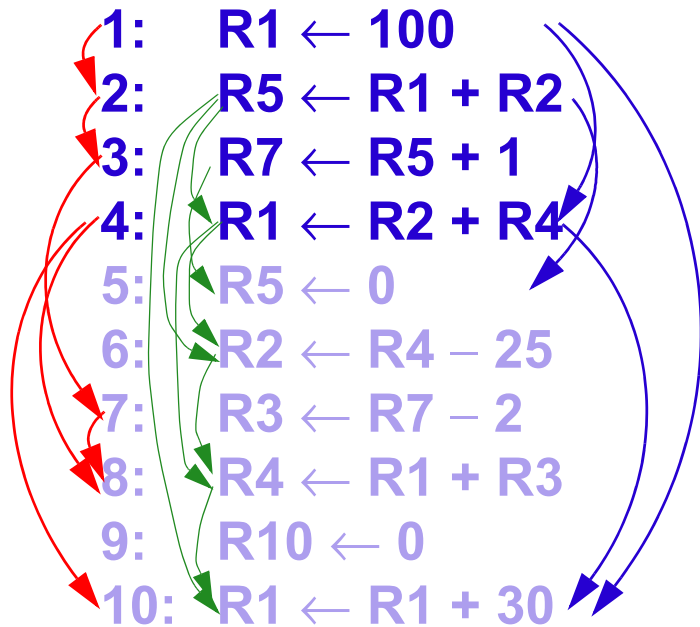
**True data dependency (RAW):** 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

**Output dependency (WAW):** 1 - 4, 1 - 10, 2 - 5, 4 - 10.

**Antidependency (WAR):** 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10

# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:
- c1) in order execution;



	ALU	ALU
Cycle 1	1	
Cycle 2	2	
Cycle 3	3	4

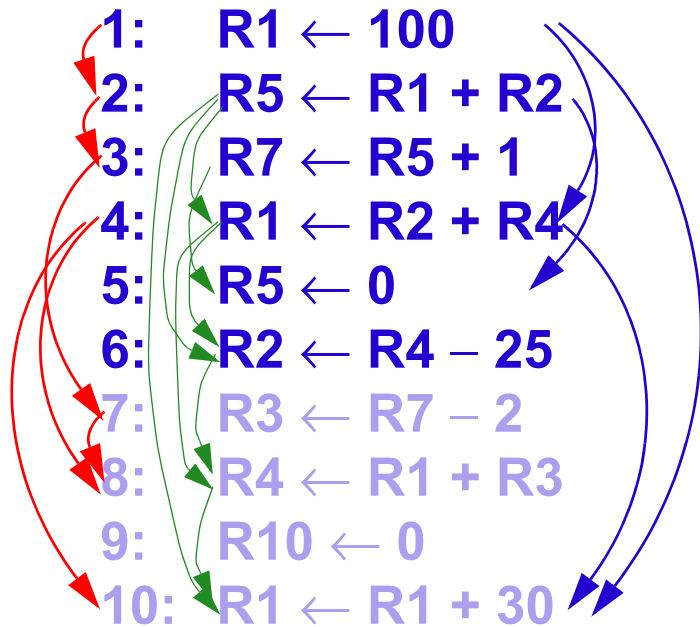
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

Antidependency (WAR): 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10

# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:
- c1) in order execution;



	ALU	ALU
Cycle 1	1	
Cycle 2	2	
Cycle 3	3	4
Cycle 5	5	6

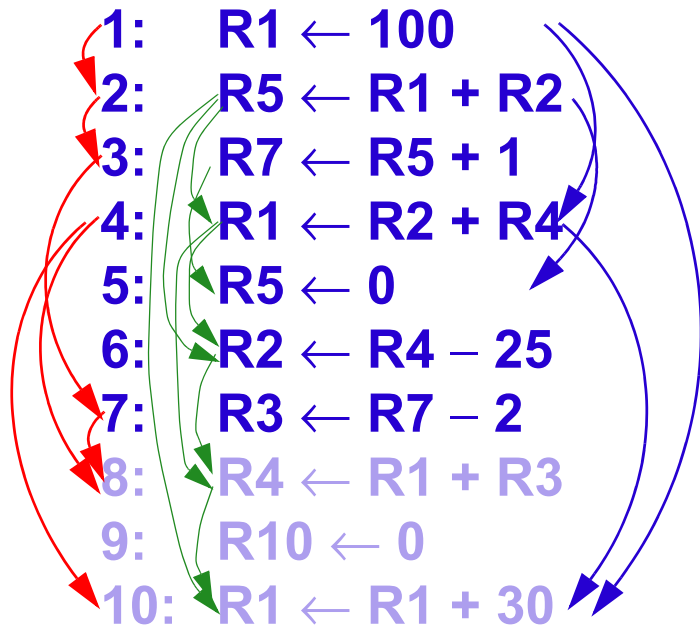
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

Antidependency (WAR): 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10

# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:
- c1) in order execution;



	ALU	ALU
Cycle 1	1	
Cycle 2	2	
Cycle 3	3	4
Cycle 4	5	6
Cycle 5	7	

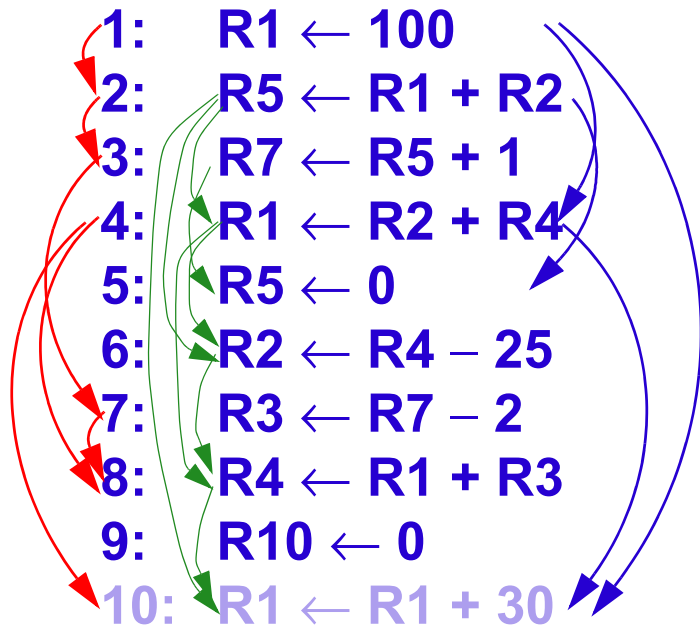
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

Antidependency (WAR): 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10

# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:
- c1) in order execution;



	ALU	ALU
Cycle 1	1	
Cycle 2	2	
Cycle 3	3	4
Cycle 4	5	6
Cycle 5	7	
Cycle 6	8	9

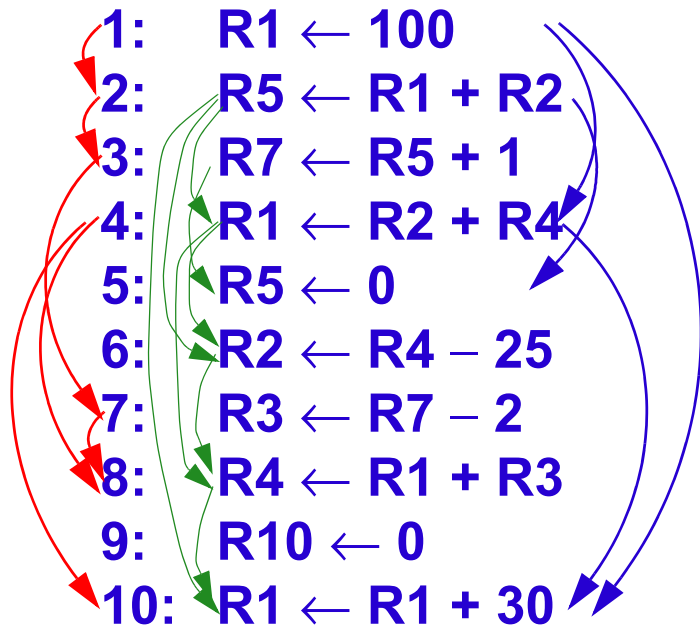
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

Antidependency (WAR): 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10

# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:  
c1) in order execution;



	ALU	ALU
Cycle 1	1	
Cycle 2	2	
Cycle 3	3	4
Cycle 4	5	6
Cycle 5	7	
Cycle 6	8	9
Cycle 7	10	

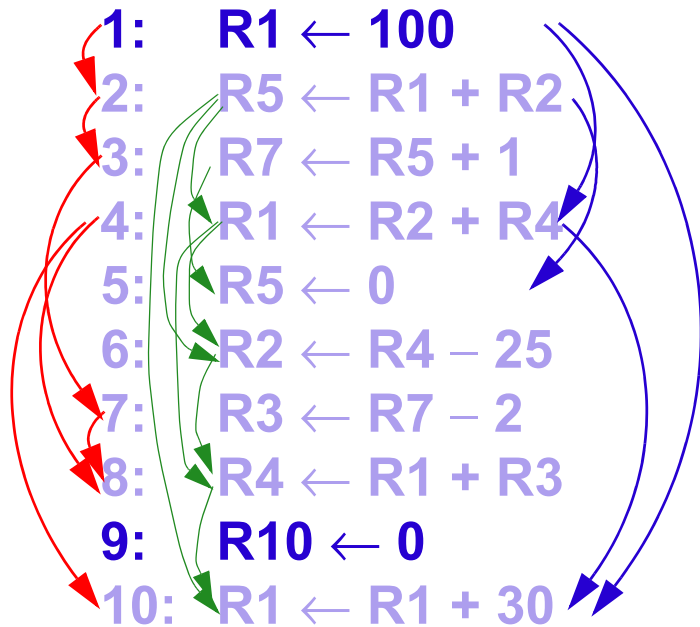
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

Antidependency (WAR): 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10

# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:  
 c2) out-of-order order execution, without renaming;



	ALU	ALU
Cycle 1	1	9

True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

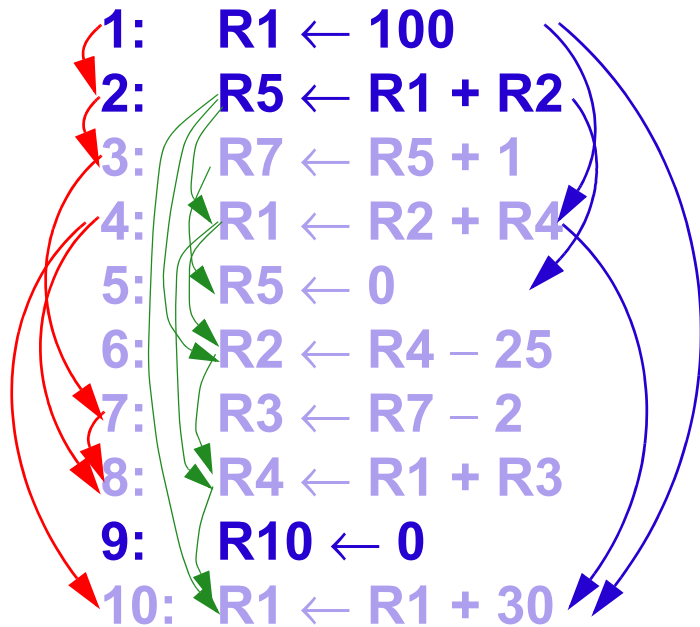
Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

Antidependency (WAR): 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10



# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:  
 c2) out-of-order order execution, without renaming;



	ALU	ALU
Cycle 1	1	9
Cycle 2	2	

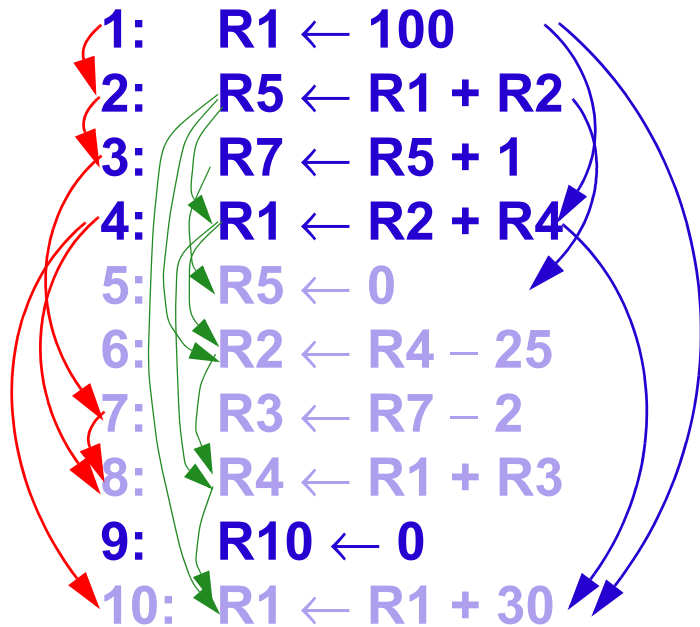
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

Antidependency (WAR): 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10

# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:  
c2) out-of-order order execution, without renaming;



	ALU	ALU
Cycle 1	1	9
Cycle 2	2	
Cycle 3	3	4

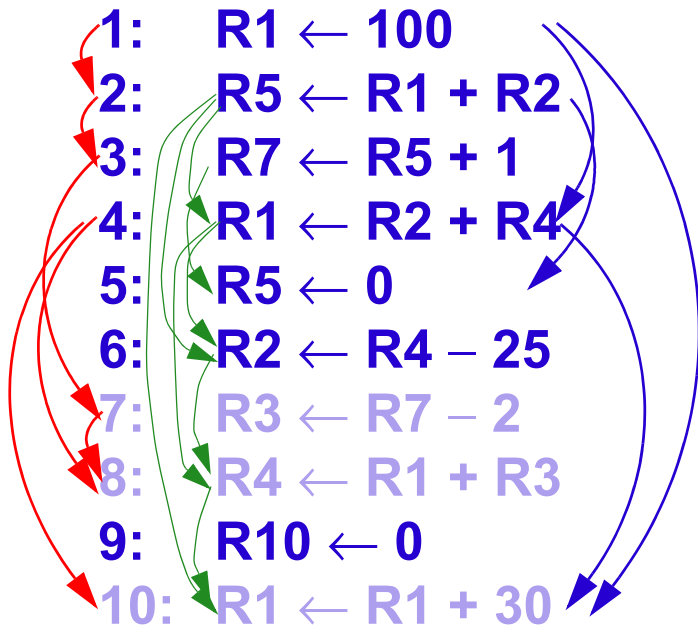
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

Antidependency (WAR): 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10

# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:  
 c2) out-of-order order execution, without renaming;



	ALU	ALU
Cycle 1	1	9
Cycle 2	2	
Cycle 3	3	4
Cycle 4	5	6

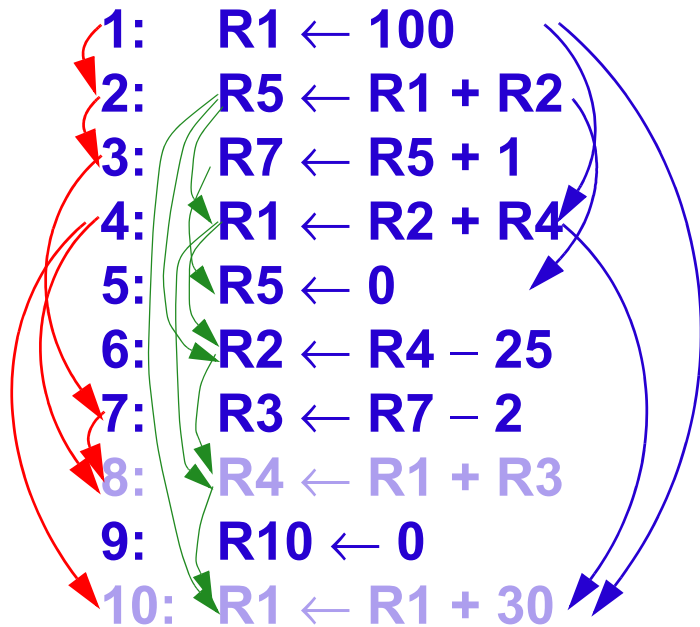
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

Antidependency (WAR): 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10

# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:  
 c2) out-of-order order execution, without renaming;



	ALU	ALU
Cycle 1	1	9
Cycle 2	2	
Cycle 3	3	4
Cycle 4	5	6
Cycle 5	7	

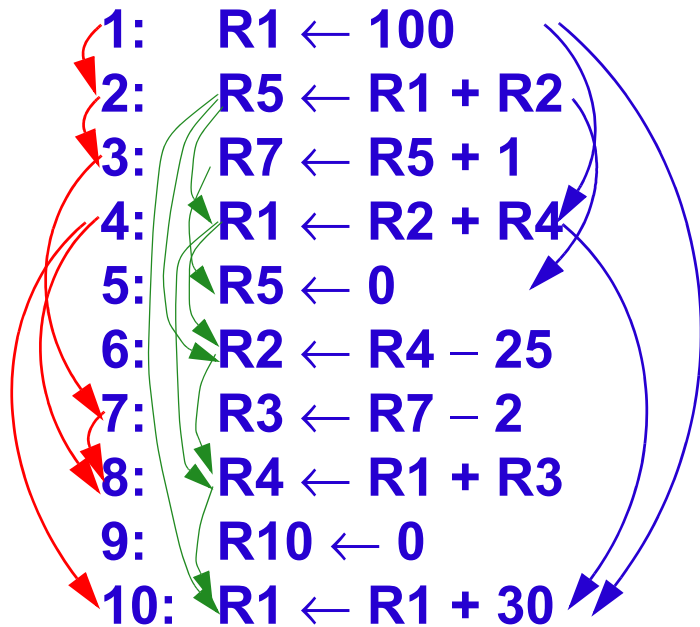
True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

Antidependency (WAR): 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10

# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:  
 c2) out-of-order order execution, without renaming;



	ALU	ALU
Cycle 1	1	9
Cycle 2	2	
Cycle 3	3	4
Cycle 4	5	6
Cycle 5	7	
Cycle 6	8	10!

True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): 1 - 4, 1 - 10, 2 - 5, 4 - 10.

Antidependency (WAR): 2 - 4, 2 - 10, 2 - 6, 3 - 5, 4 - 6, 4 - 8, 6 - 8, 8 - 10

# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:  
c3) out-of-order order execution, with renaming;

1: R1 ← 100  
 2: R5 ← R1 + R2  
 3: R7 ← R5 + 1  
 4: **R11** ← R2 + R4  
 5: R51 ← 0  
 6: R21 ← R4 - 25  
 7: R3 ← R7 - 2  
 8: R41 ← R11 + R3  
 9: R10 ← 0  
 10: R12 ← R11 + 30

	ALU	ALU
Cycle 1	1	4

True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, ~~2 - 5~~, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, ~~2 - 6~~, ~~3 - 5~~, ~~4 - 6~~, ~~4 - 8~~, ~~6 - 8~~, ~~8 - 10~~

# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:  
 c3) out-of-order order execution, with renaming;

- 1: R1 ← 100
  - 2: R5 ← R1 + R2
  - 3: R7 ← R5 + 1
  - 4: **R11** ← R2 + R4
  - 5: **R51** ← 0
  - 6: R21 ← R4 - 25
  - 7: R3 ← R7 - 2
  - 8: R41 ← R11 + R3
  - 9: R10 ← 0
  - 10: R12 ← R11 + 30
- 

	ALU	ALU
Cycle 1	1	4
Cycle 2	2	5

True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, ~~2 - 5~~, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, ~~2 - 6~~, ~~3 - 5~~, ~~4 - 6~~, ~~4 - 8~~, ~~6 - 8~~, ~~8 - 10~~

# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:  
 c3) out-of-order order execution, with renaming;

1: R1 ← 100  
 2: R5 ← R1 + R2  
 3: R7 ← R5 + 1  
 4: **R11** ← R2 + R4  
 5: **R51** ← 0  
 6: **R21** ← R4 - 25  
 7: R3 ← R7 - 2  
 8: R41 ← R11 + R3  
 9: R10 ← 0  
 10: R12 ← R11 + 30

	ALU	ALU
Cycle 1	1	4
Cycle 2	2	5
Cycle 3	3	6

True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, ~~2 - 5~~, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, ~~2 - 6~~, ~~3 - 5~~, ~~4 - 6~~, ~~4 - 8~~, ~~6 - 8~~, ~~8 - 10~~



# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:  
c3) out-of-order order execution, with renaming;

1: R1 ← 100  
 2: R5 ← R1 + R2  
 3: R7 ← R5 + 1  
 4: **R11** ← R2 + R4  
 5: **R51** ← 0  
 6: **R21** ← R4 - 25  
 7: R3 ← R7 - 2  
 8: R41 ← R11 + R3  
 9: R10 ← 0  
 10: R12 ← R11 + 30

	ALU	ALU
Cycle 1	1	4
Cycle 2	2	5
Cycle 3	3	6
Cycle 4	7	9

True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, ~~2 - 5~~, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, ~~2 - 6~~, ~~3 - 5~~, ~~4 - 6~~, ~~4 - 8~~, ~~6 - 8~~, ~~8 - 10~~

# Problem 3

- c. Consider a superscalar computer on which the execution of each instruction takes one cycle; the computer has two arithmetic units. Show how instructions are executed in consecutive cycles with:  
 c3) out-of-order order execution, with renaming;

1: R1 ← 100  
 2: R5 ← R1 + R2  
 3: R7 ← R5 + 1  
 4: **R11** ← R2 + R4  
 5: **R51** ← 0  
 6: **R21** ← R4 - 25  
 7: R3 ← R7 - 2  
 8: **R41** ← **R11** + R3  
 9: R10 ← 0  
 10: **R12** ← **R11** + 30

	ALU	ALU
Cycle 1	1	4
Cycle 2	2	5
Cycle 3	3	6
Cycle 4	7	9
Cycle 5	8	10

True data dependency (RAW): 1 - 2, 2 - 3, 3 - 7, 4 - 8, 4 - 10, 7 - 8.

Output dependency (WAW): ~~1 - 4~~, ~~1 - 10~~, ~~2 - 5~~, ~~4 - 10~~.

Antidependency (WAR): ~~2 - 4~~, ~~2 - 10~~, ~~2 - 6~~, ~~3 - 5~~, ~~4 - 6~~, ~~4 - 8~~, ~~6 - 8~~, ~~8 - 10~~

# Problem 4

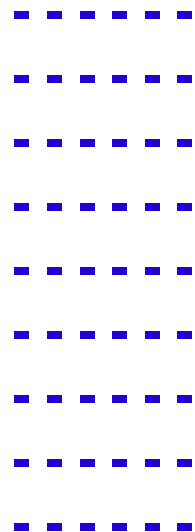
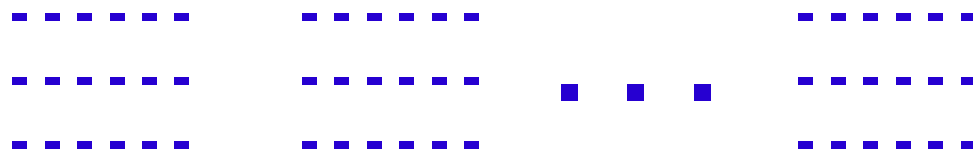
An application program is executed on a nine-processor cluster. The program took time  $T$  on this cluster. Further, it was found that 25% of  $T$  was time in which the application was running simultaneously on all nine processors. The remaining time, the application had to run on a single processor.

- a. Calculate the speedup under the aforementioned conditions (relative to execution on a single processor).
- b. Suppose that we are able to effectively use 17 processors rather than 9 on the parallelized portion of the code. Calculate the speedup (relative to execution on a single processor) that is achieved.

# Problem 4

An application program is executed on a nine-processor cluster. The program took time  $T$  on this cluster. Further, it was found that 25% of  $T$  was time in which the application was running simultaneously on all nine processors. The remaining time, the application had to run on a single processor.

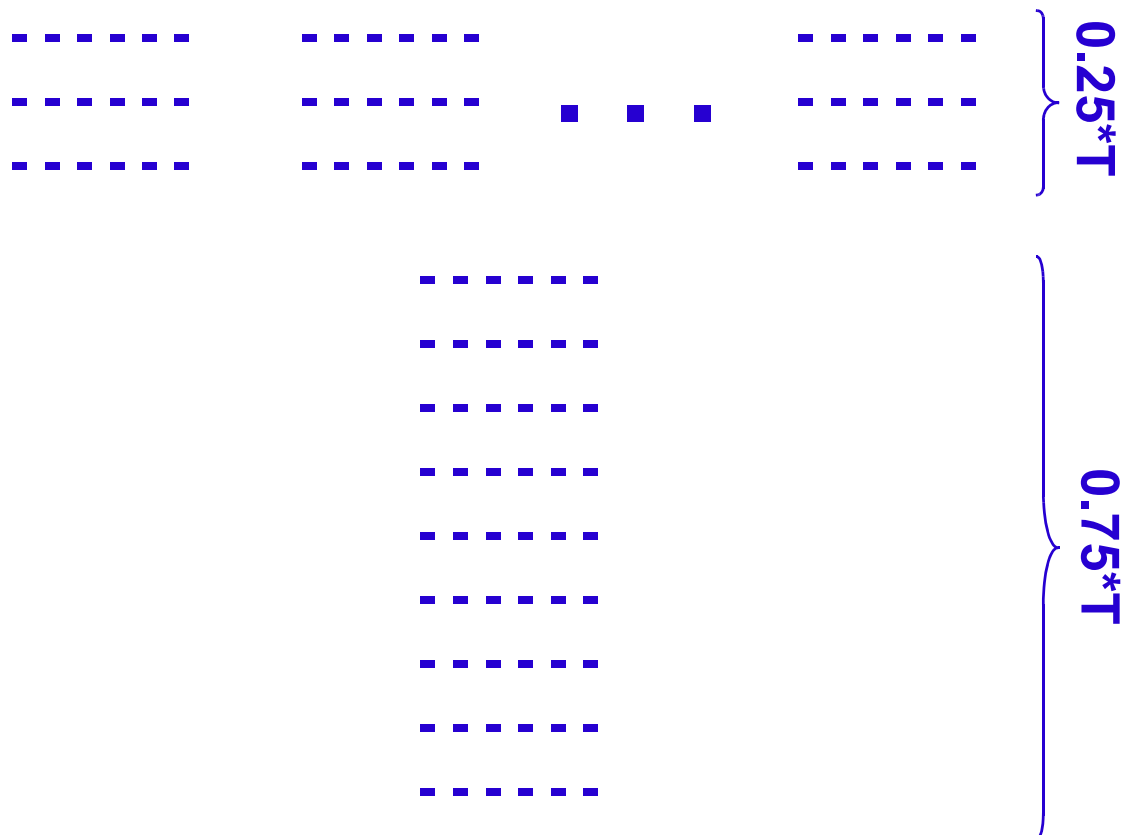
- a. Calculate the speedup under the aforementioned conditions (relative to execution on a single processor).



# Problem 4

An application program is executed on a nine-processor cluster. The program took time  $T$  on this cluster. Further, it was found that 25% of  $T$  was time in which the application was running simultaneously on all nine processors. The remaining time, the application had to run on a single processor.

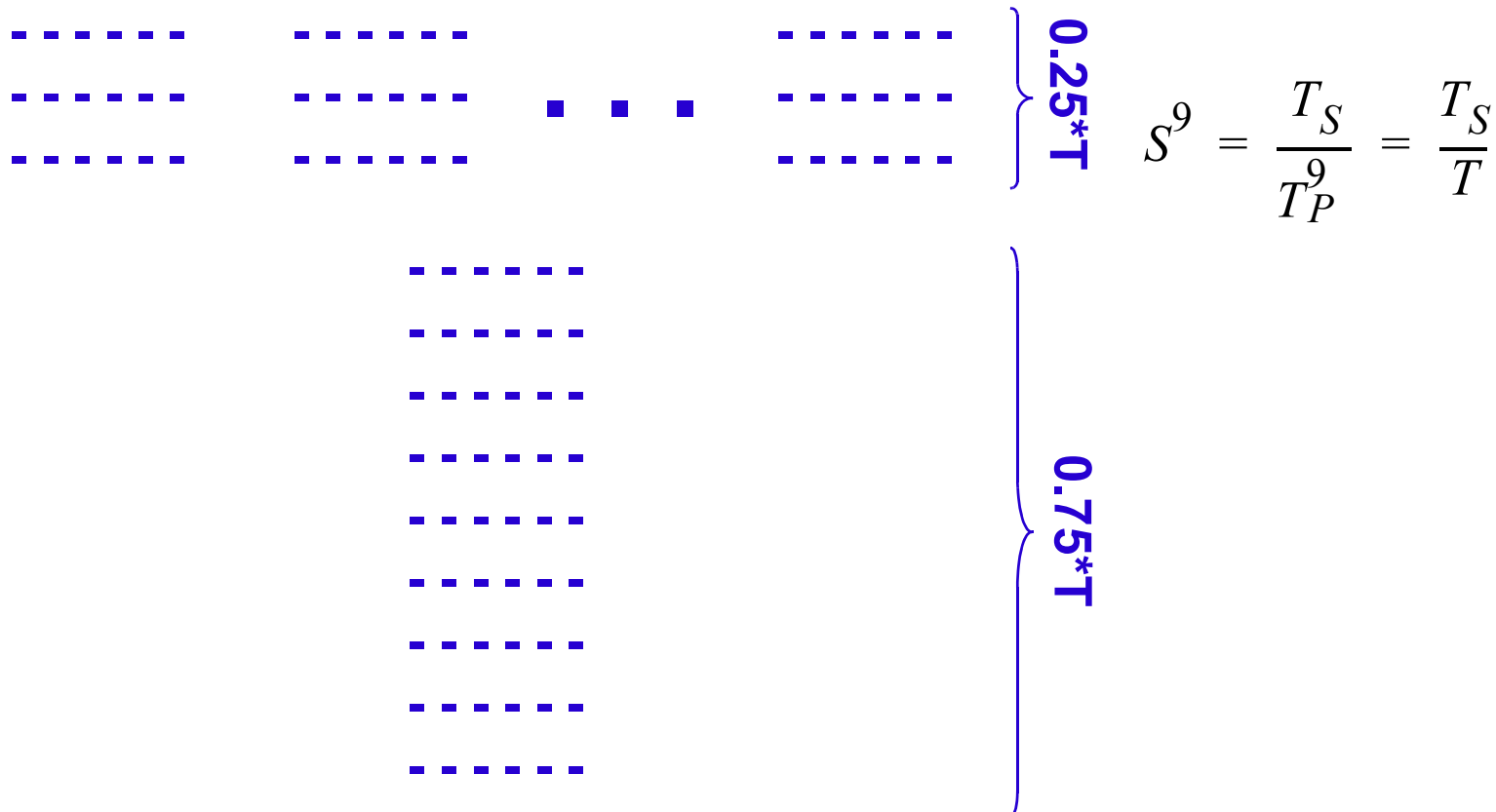
- a. Calculate the speedup under the aforementioned conditions (relative to execution on a single processor).



# Problem 4

An application program is executed on a nine-processor cluster. The program took time  $T$  on this cluster. Further, it was found that 25% of  $T$  was time in which the application was running simultaneously on all nine processors. The remaining time, the application had to run on a single processor.

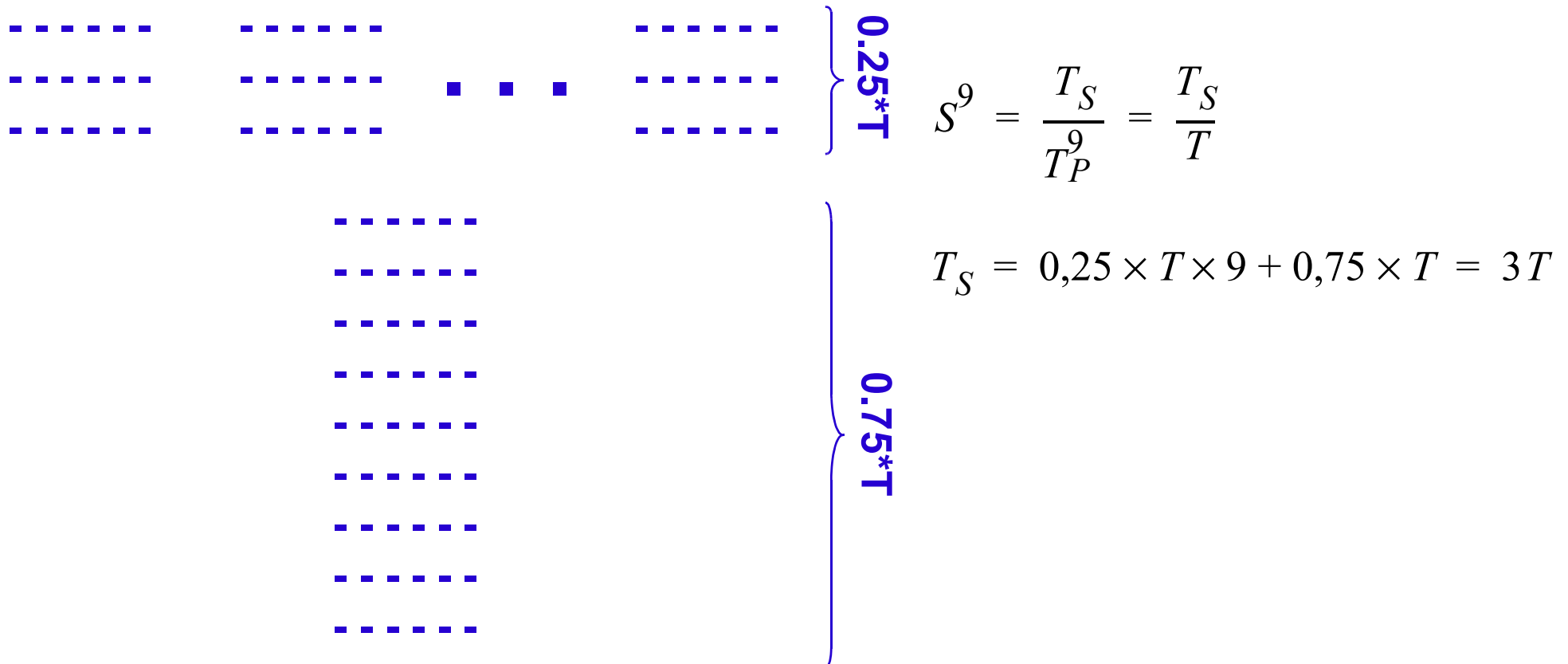
- a. Calculate the speedup under the aforementioned conditions (relative to execution on a single processor).



# Problem 4

An application program is executed on a nine-processor cluster. The program took time  $T$  on this cluster. Further, it was found that 25% of  $T$  was time in which the application was running simultaneously on all nine processors. The remaining time, the application had to run on a single processor.

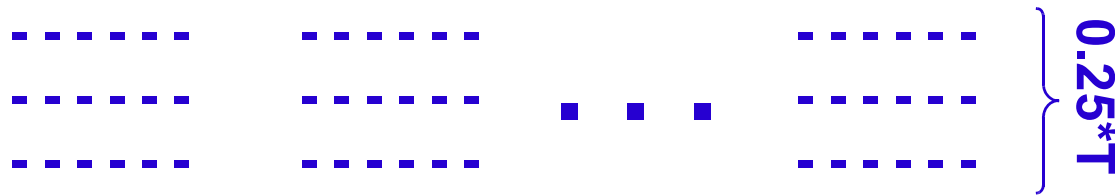
- a. Calculate the speedup under the aforementioned conditions (relative to execution on a single processor).



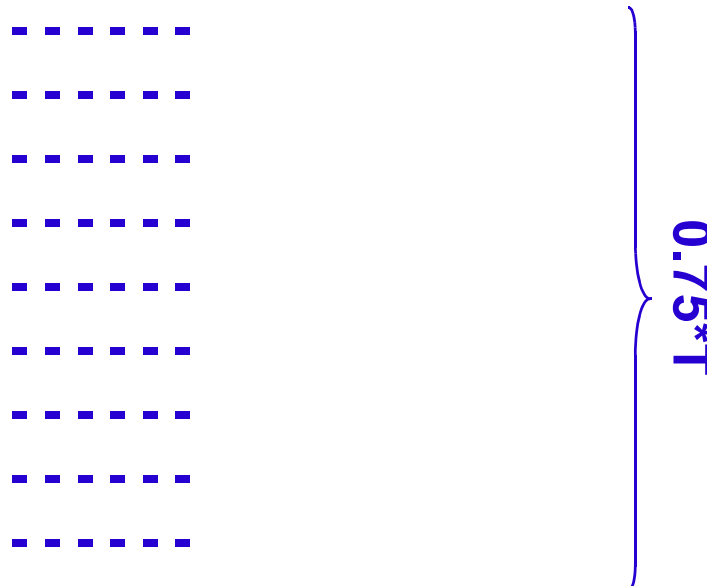
# Problem 4

An application program is executed on a nine-processor cluster. The program took time  $T$  on this cluster. Further, it was found that 25% of  $T$  was time in which the application was running simultaneously on all nine processors. The remaining time, the application had to run on a single processor.

- a. Calculate the speedup under the aforementioned conditions (relative to execution on a single processor).



$$S^9 = \frac{T_S}{T_P^9} = \frac{T_S}{T}$$



$$T_S = 0,25 \times T \times 9 + 0,75 \times T = 3T$$



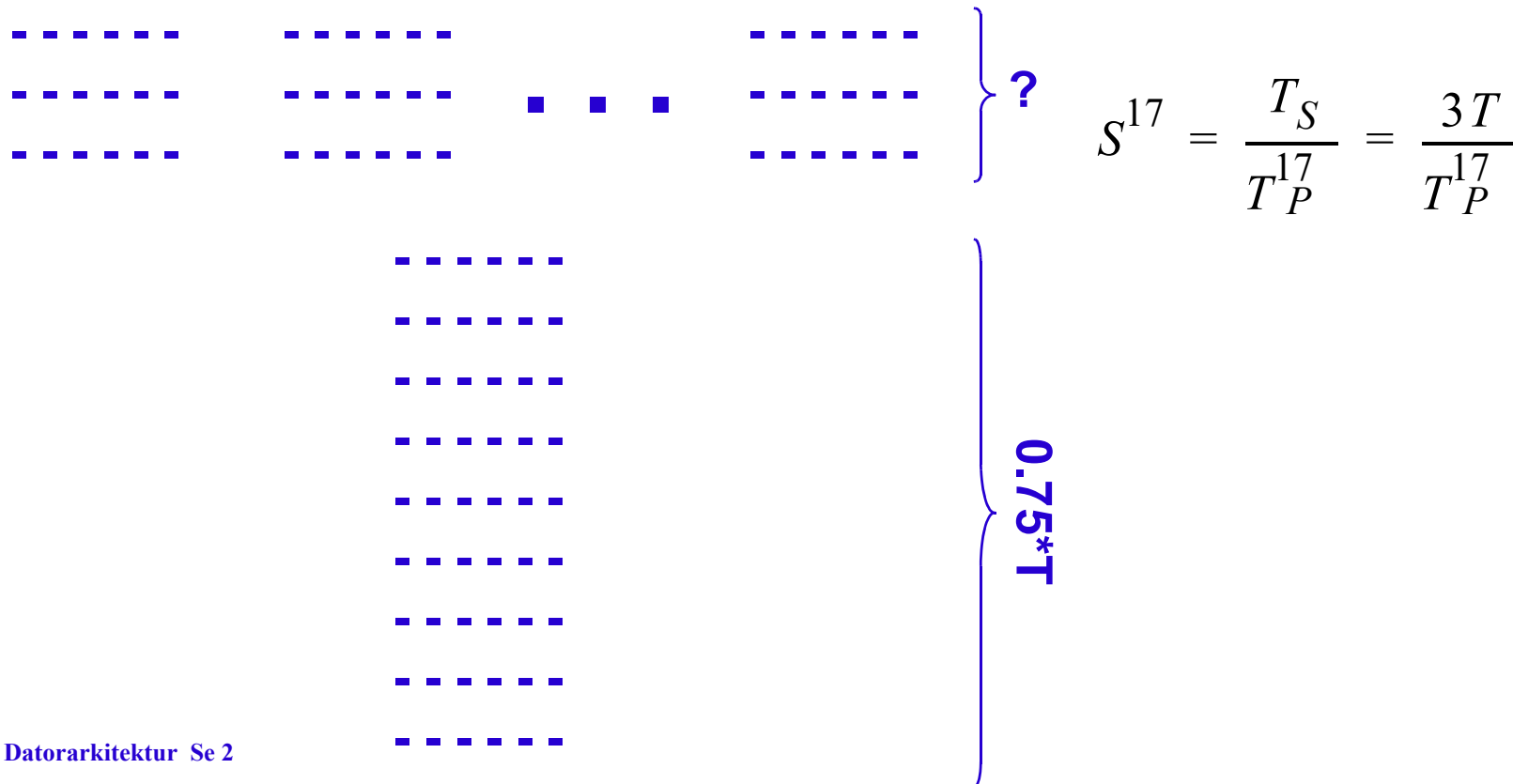
$$S^9 = 3$$



# Problem 4

An application program is executed on a nine-processor cluster. The program took time  $T$  on this cluster. Further, it was found that 25% of  $T$  was time in which the application was running simultaneously on all nine processors. The remaining time, the application had to run on a single processor.

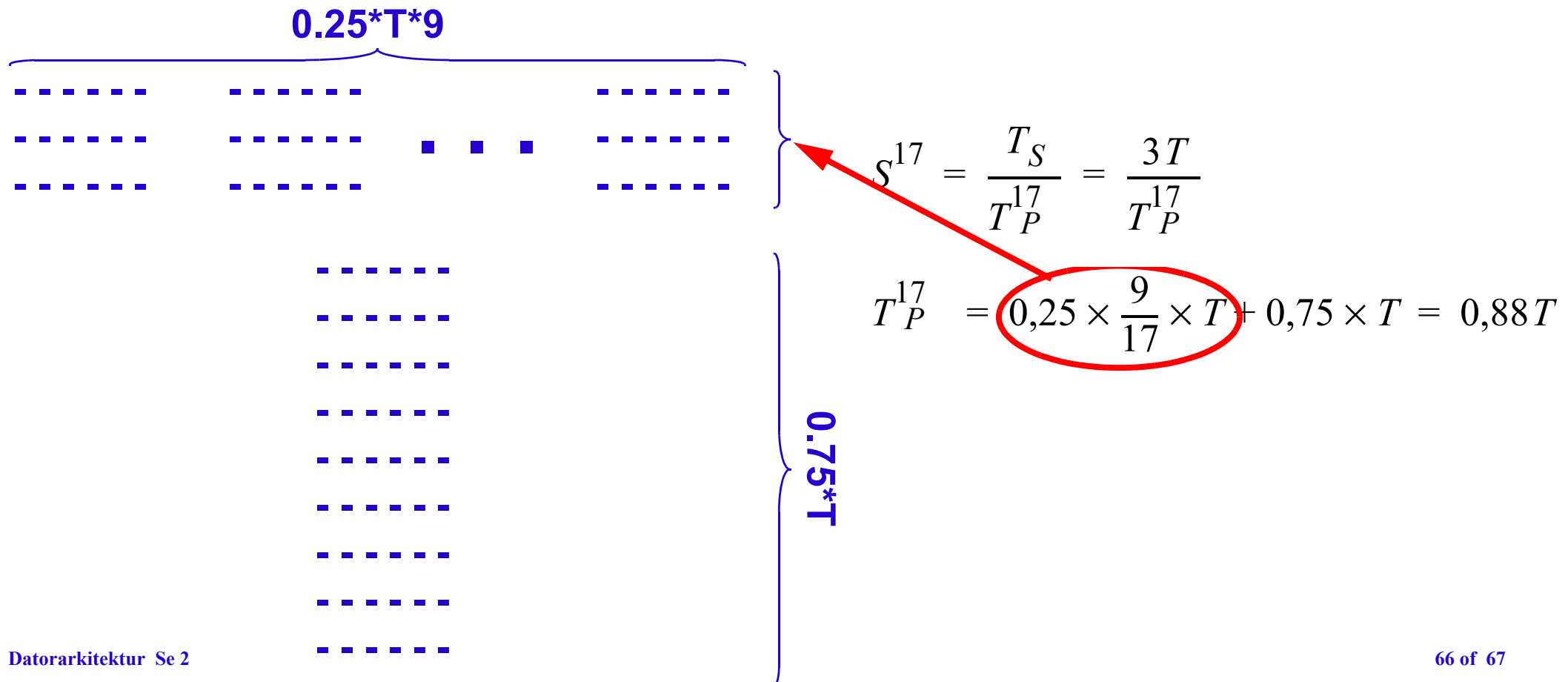
- b. Suppose that we are able to effectively use 17 processors rather than 9 on the parallelized portion of the code. Calculate the speedup (relative to execution on a single processor) that is achieved.



# Problem 4

An application program is executed on a nine-processor cluster. The program took time  $T$  on this cluster. Further, it was found that 25% of  $T$  was time in which the application was running simultaneously on all nine processors. The remaining time, the application had to run on a single processor.

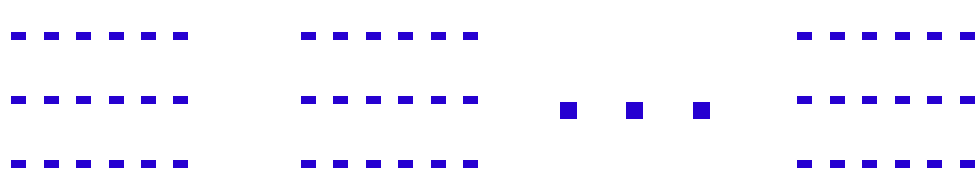
b. Suppose that we are able to effectively use 17 processors rather than 9 on the parallelized portion of the code. Calculate the speedup (relative to execution on a single processor) that is achieved.



# Problem 4

An application program is executed on a nine-processor cluster. The program took time  $T$  on this cluster. Further, it was found that 25% of  $T$  was time in which the application was running simultaneously on all nine processors. The remaining time, the application had to run on a single processor.

b. Suppose that we are able to effectively use 17 processors rather than 9 on the parallelized portion of the code. Calculate the speedup (relative to execution on a single processor) that is achieved.



$$S^{17} = \frac{T_S}{T_P^{17}} = \frac{3T}{T_P^{17}}$$

$$T_P^{17} = 0,25 \times \frac{9}{17} \times T + 0,75 \times T = 0,88T$$

0.75\*T

$$S^{17} = \frac{3T}{0,88T} = 3,4$$