CS311:Computer Organization
Substitute Lecture:

Problem Solving for Homework #2

Wonyoung Lee

Embedded Computing Lab.

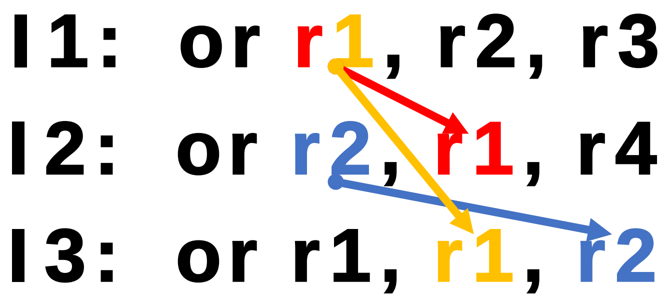
- References
  1. Patterson, David A., and John L. Hennessy. ***Computer Organization and Design MIPS Edition: The Hardware/Software Interface***. Newnes, 2013. (Textbook)

Problem 1

1.a Indicate every dependence that incurs hazards.

```
I1:  or r1, r2, r3
I2:  or r2, r1, r4
I3:  or r1, r1, r2
```

1.a Indicate every dependence that incurs hazards.



```
I1:  or r1, r2, r3
I2:  or r2, r1, r4
I3:  or r1, r1, r2
```

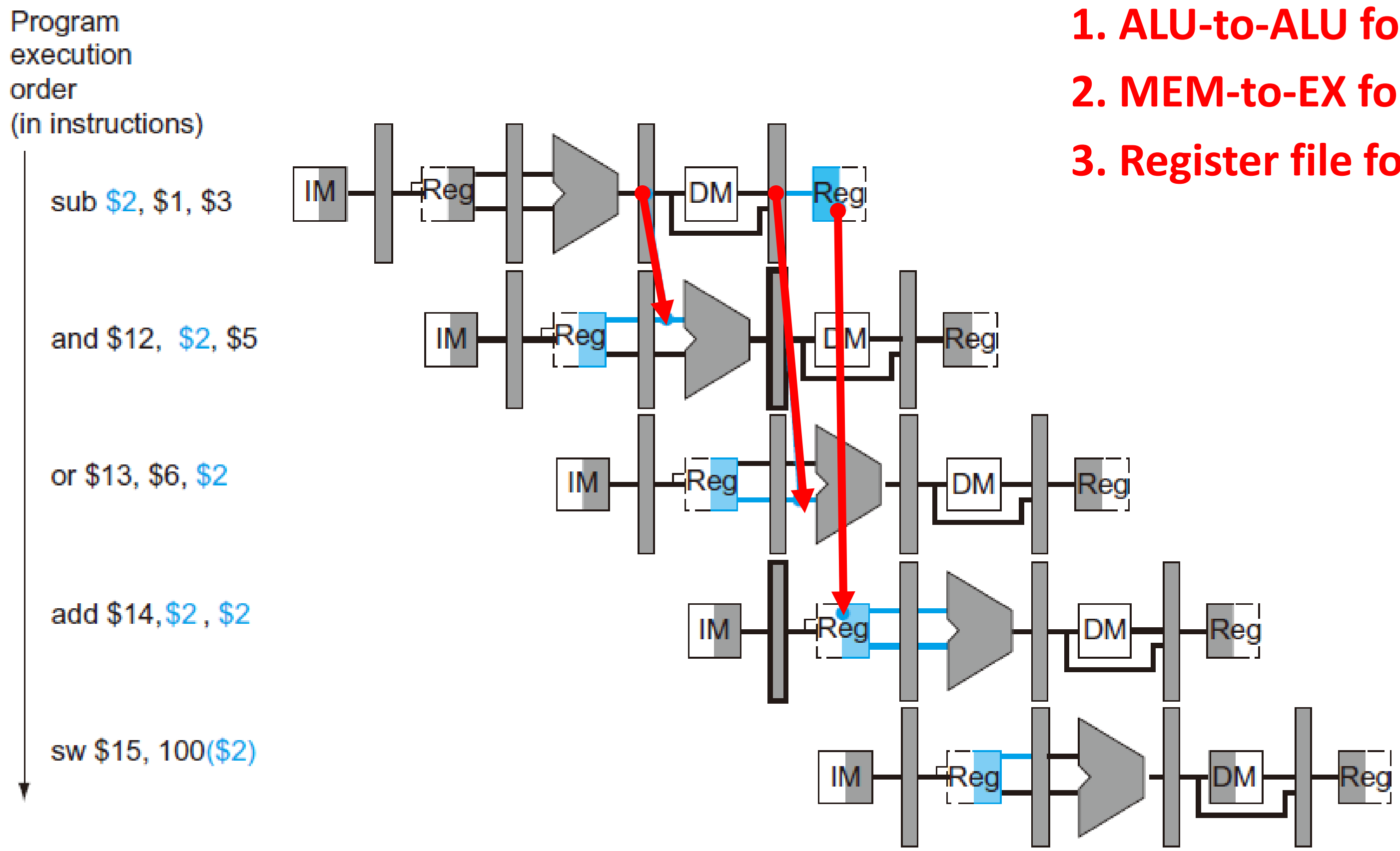
Answer:

- **Dependence on register r1 from I1 to I2**
- **Dependence on register r1 from I1 to I3**
- **Dependence on register r2 from I2 to I3**

(In “from A to B”, the position of A, B can be switched)

1.b Assume there is **no forwarding** in this pipelined processor. Indicate hazards and add the minimum number of **nop** instructions to eliminate them.

```
I1: or r1, r2, r3
I2: or r2, r1, r4
I3: or r1, r1, r2
```



1. ALU-to-ALU forwarding
2. MEM-to-EX forwarding
3. Register file forwarding

FIGURE 4.53 The dependences between the pipeline registers move forward in time, so it is possible to supply the inputs to the ALU needed by the AND instruction and OR instruction by forwarding the results found in the pipeline registers. The values in the pipeline registers show that the desired value is available before it is written into the register file. We assume that the register file forwards values that are read and written during the same clock cycle, so the add does not stall, but the values come from the register file instead of a pipeline register. Register file "forwarding"—that is, the read gets the value of the write in that clock cycle—is why clock cycle 5 shows register $2 having the value 10 at the beginning and −20 at the end of the clock cycle. As in the rest of this section, we handle all forwarding except for the value to be stored by a store instruction.

Register file "forwarding"—that is, the read gets the value of the write in that clock cycle—

1. ALU-to-ALU forwarding
2. MEM-to-EX forwarding
3. Register file forwarding

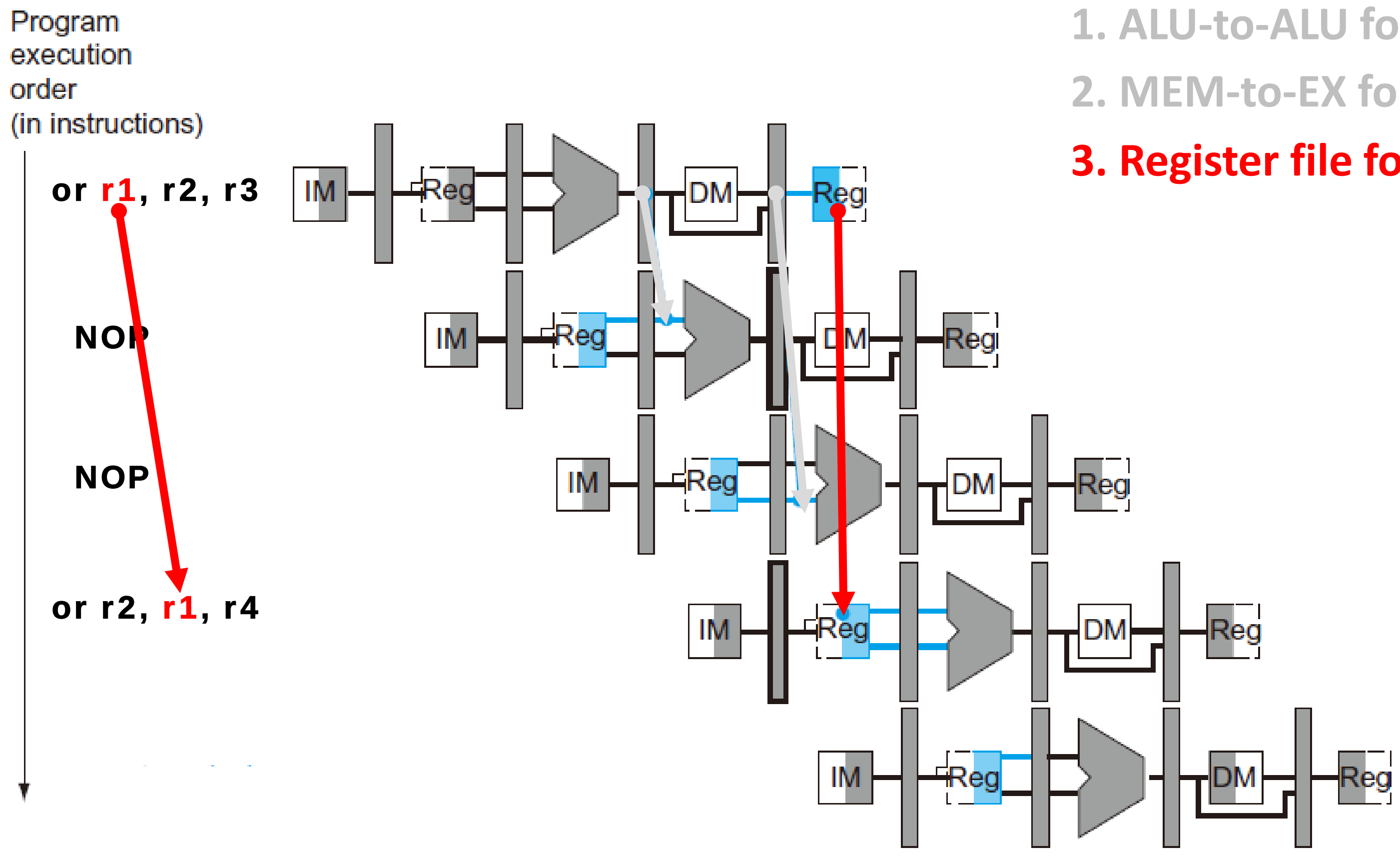
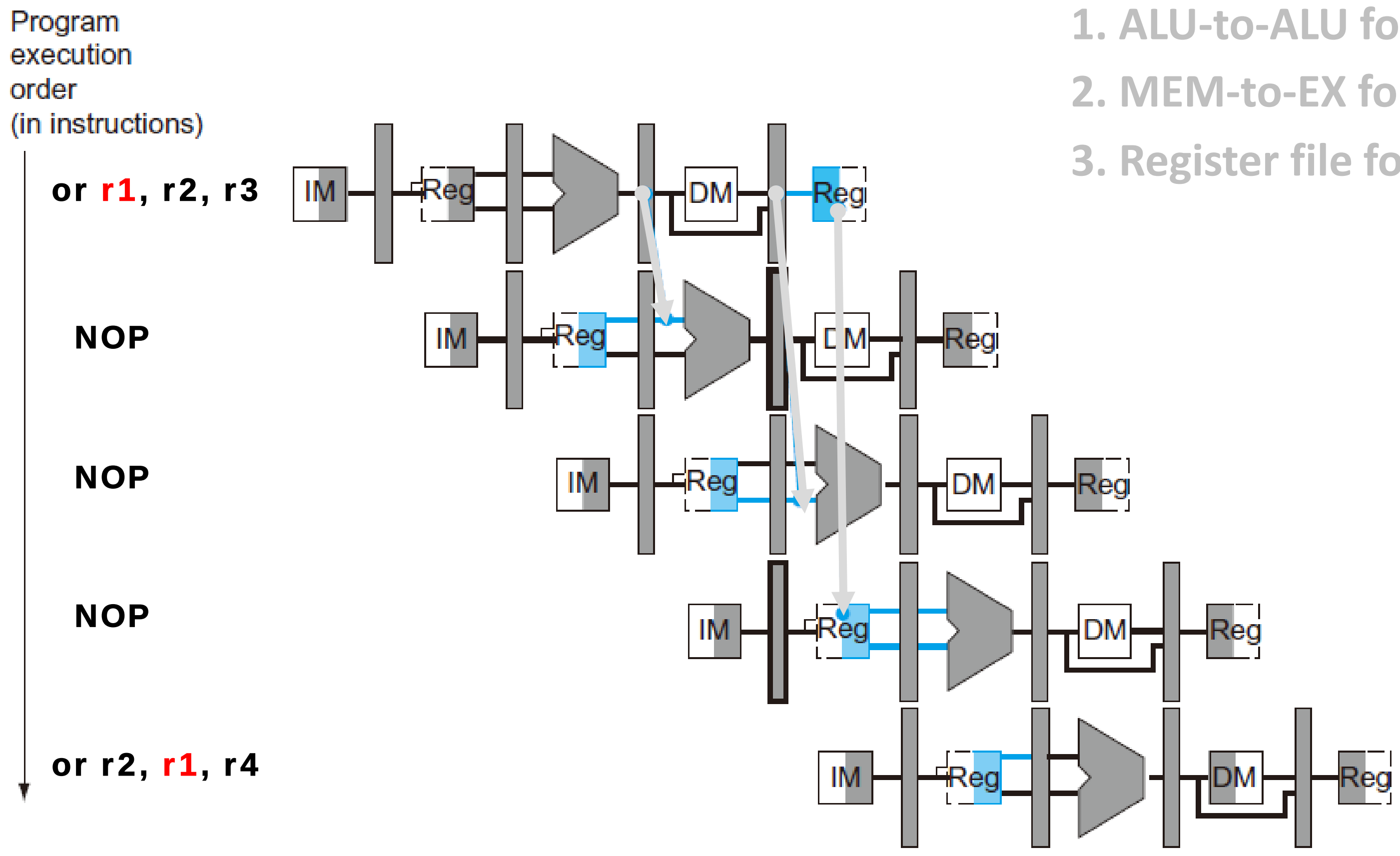


FIGURE 4.53 The dependences between the pipeline registers move forward in time, so it is possible to supply the inputs to the ALU needed by the AND instruction and OR instruction by forwarding the results found in the pipeline registers. The values in the pipeline registers show that the desired value is available before it is written into the register file. We assume that the register file forwards values that are read and written during the same clock cycle, so the add does not stall, but the values come from the register file instead of a pipeline register. Register file "forwarding"—that is, the read gets the value of the write in that clock cycle—is why clock cycle 5 shows register \$2 having the value 10 at the beginning and –20 at the end of the clock cycle. As in the rest of this section, we handle all forwarding except for the value to be stored by a store instruction.



1. ALU-to-ALU forwarding
2. MEM-to-EX forwarding
3. Register file forwarding

FIGURE 4.53 The dependences between the pipeline registers move forward in time, so it is possible to supply the inputs to the ALU needed by the AND instruction and OR instruction by forwarding the results found in the pipeline registers. The values in the pipeline registers show that the desired value is available before it is written into the register file. We assume that the register file forwards values that are read and written during the same clock cycle, so the add does not stall, but the values come from the register file instead of a pipeline register. Register file "forwarding"—that is, the read gets the value of the write in that clock cycle—is why clock cycle 5 shows register \$2 having the value 10 at the beginning and –20 at the end of the clock cycle. As in the rest of this section, we handle all forwarding except for the value to be stored by a store instruction.

1.b Assume there is **no forwarding** in this pipelined processor. Indicate hazards and add the minimum number of **nop** instructions to eliminate them.

```
I1:  or r1, r2, r3
  NOP
  NOP
I2:  or r2, r1, r4
  NOP
  NOP
I3:  or r1, r1, r2
```

With Register file forwarding
: 7 instructions

```
I1:  or r1, r2, r3
  NOP
  NOP
  NOP
I2:  or r2, r1, r4
  NOP
  NOP
  NOP
I3:  or r1, r1, r2
```

Without Register file forwarding
: 9 instructions

1.c Assume there is full forwarding. Indicate hazards and add the minimum number of **nop** instructions to eliminate them.

```
I1:  or r1, r2, r3
I2:  or r2, r1, r4
I3:  or r1, r1, r2
```

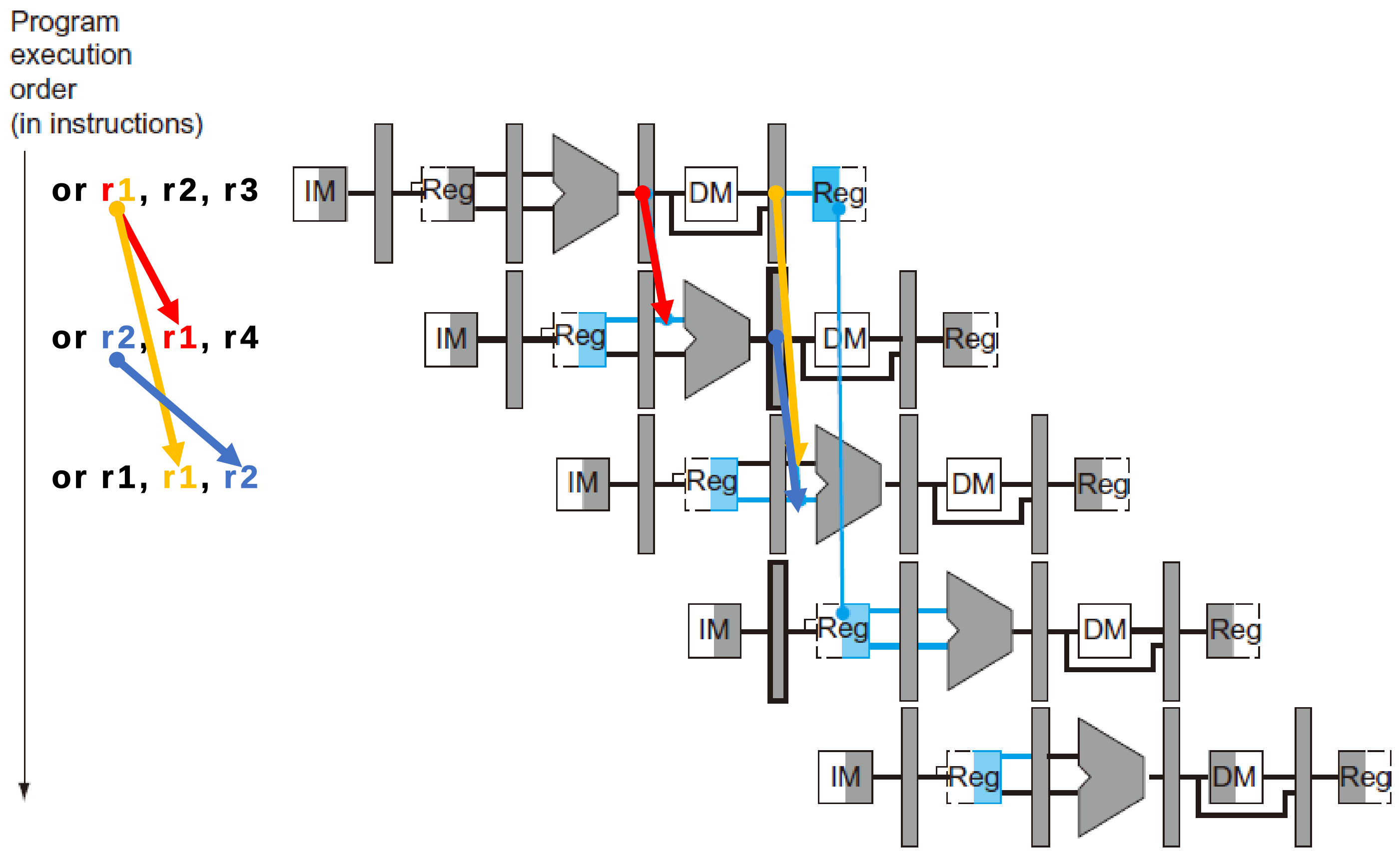


FIGURE 4.53 The dependences between the pipeline registers move forward in time, so it is possible to supply the inputs to the ALU needed by the AND instruction and OR instruction by forwarding the results found in the pipeline registers. The values in the pipeline registers show that the desired value is available before it is written into the register file. We assume that the register file forwards values that are read and written during the same clock cycle, so the add does not stall, but the values come from the register file instead of a pipeline register. Register file "forwarding"—that is, the read gets the value of the write in that clock cycle—is why clock cycle 5 shows register $2 having the value 10 at the beginning and −20 at the end of the clock cycle. As in the rest of this section, we handle all forwarding except for the value to be stored by a store instruction.

1.c Assume there is full forwarding. Indicate hazards and add the minimum number of **nop** instructions to eliminate them.

```
I1:  or r1, r2, r3
I2:  or r2, r1, r4
I3:  or r1, r1, r2
```

With or Without Register file forwarding

: 3 instructions

1.d What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

With Register file forwarding

- **No-forwarding**

  = 7 instructions = 11 cycles
- **Full forwarding**

  = 3 instructions = 7 cycles

Without Register file forwarding

- **No-forwarding**

  = 9 instructions = 13 cycles
- **Full forwarding**

  = 3 instructions = 7 cycles

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1	🟥	🟥	🟥	🟥	▨										
I2	🟥	🟥	🟥	🟥	▨	▨									
I3	🟥	🟥	🟥	🟥	▨	▨	▨								
I4	🟥	🟥	🟥	🟥	▨	▨	▨	▨							
I5	🟥	🟥	🟥	🟥	▨	▨	▨	▨	▨						
I6	🟥	🟥	🟥	🟥		▨	▨	▨	▨	▨					
I7	🟥	🟥	🟥	🟥			▨	▨	▨	▨	▨				
I8	🟥	🟥	🟥	🟥				▨	▨	▨	▨	▨			
I9	🟥	🟥	🟥	🟥					▨	▨	▨	▨	▨		
I10	🟥	🟥	🟥	🟥						▨	▨	▨	▨	▨	

Warm-up Cycles

1.d What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

With Register file forwarding

- **No-forwarding**
  - **11 cycles x 150 ps = 1650 ps**
- **Full forwarding**
  - **7 cycles x 200 ps = 1400 ps**
- **Speedup**
  - **1650 / 1400 = 1.179**

Without Register file forwarding

- **No-forwarding:**
  - **13 cycles x 150 ps = 1950ps**
- **Full forwarding**
  - **7 cycles x 200 ps = 1400 ps**
- **Speedup**
  - **1950 / 1400 = 1.393**

1.e Add the minimum number of **nop** instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to the EX stage).

```
I1: or r1, r2, r3
I2: or r2, r1, r4
I3: or r1, r1, r2
```

1. ALU-to-ALU forwarding
2. MEM-to-EX forwarding
3. Register file forwarding

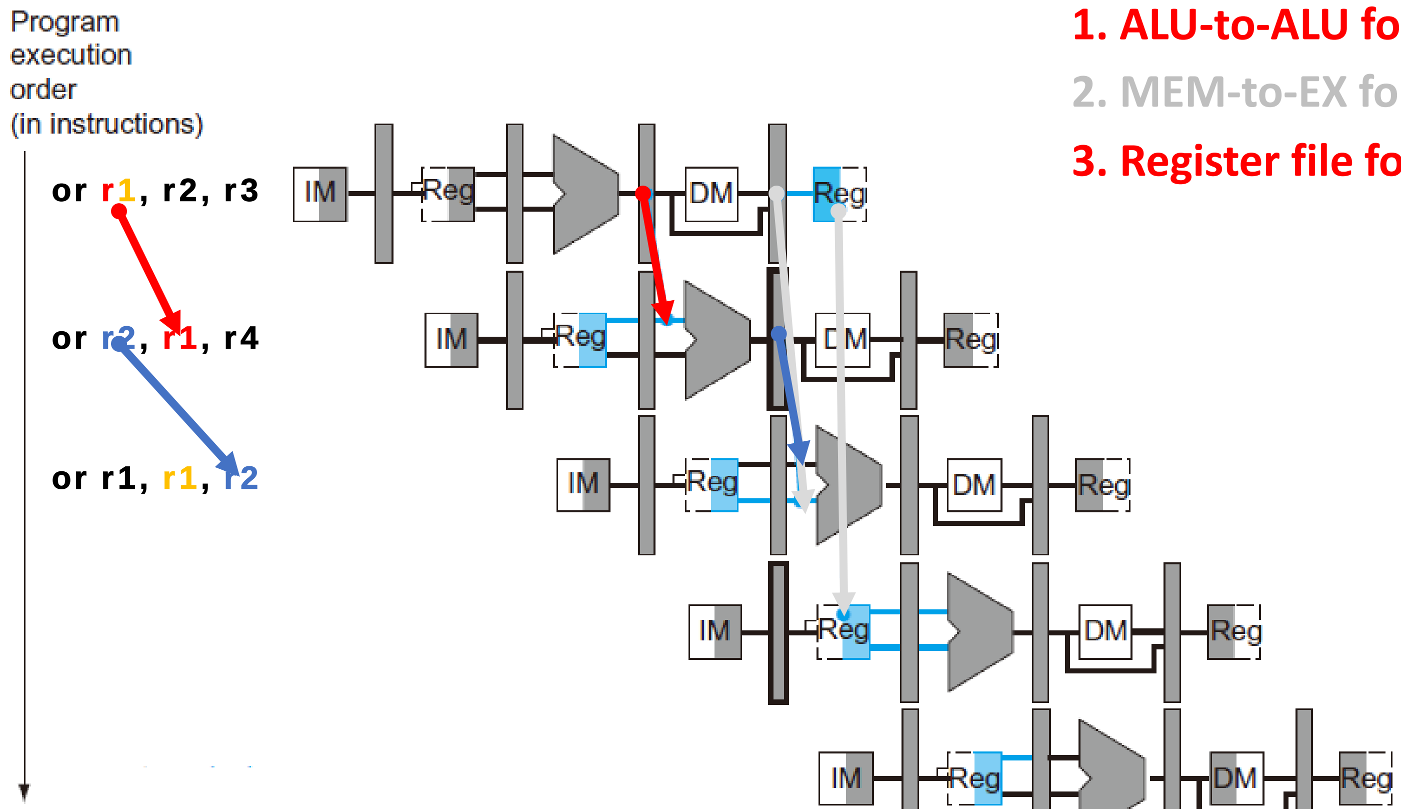


FIGURE 4.53 The dependences between the pipeline registers move forward in time, so it is possible to supply the inputs to the ALU needed by the AND instruction and OR instruction by forwarding the results found in the pipeline registers. The values in the pipeline registers show that the desired value is available before it is written into the register file. We assume that the register file forwards values that are read and written during the same clock cycle, so the add does not stall, but the values come from the register file instead of a pipeline register. Register file "forwarding"—that is, the read gets the value of the write in that clock cycle—is why clock cycle 5 shows register $2 having the value 10 at the beginning and –20 at the end of the clock cycle. As in the rest of this section, we handle all forwarding except for the value to be stored by a store instruction.

1. ALU-to-ALU forwarding
2. MEM-to-EX forwarding
3. Register file forwarding

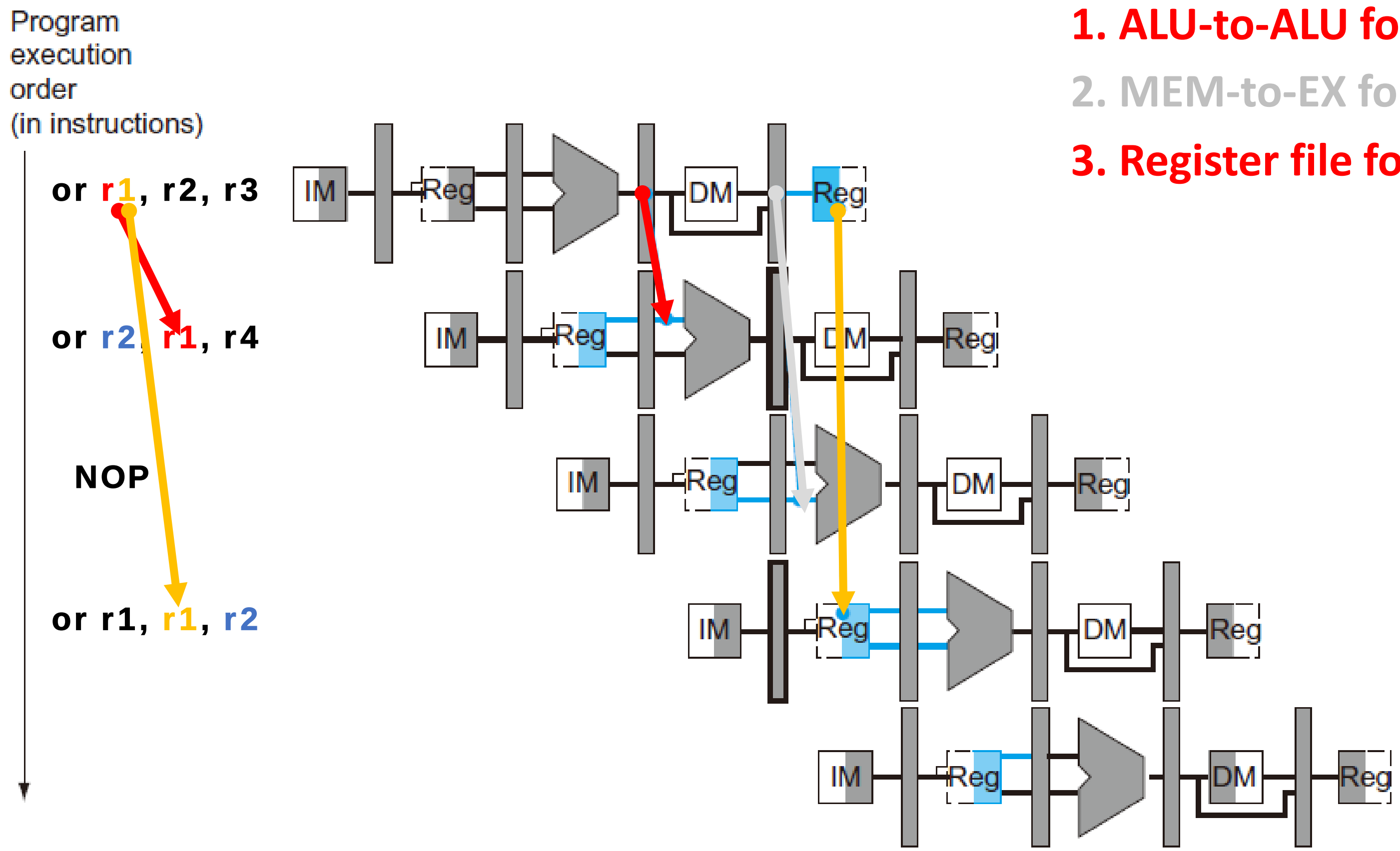


FIGURE 4.53 The dependences between the pipeline registers move forward in time, so it is possible to supply the inputs to the ALU needed by the AND **instruction and** OR **instruction by forwarding the results found in the pipeline registers.** The values in the pipeline registers show that the desired value is available before it is written into the register file. We assume that the register file forwards values that are read and written during the same clock cycle, so the add does not stall, but the values come from the register file instead of a pipeline register. Register file "forwarding"—that is, the read gets the value of the write in that clock cycle—is why clock cycle 5 shows register $2 having the value 10 at the beginning and –20 at the end of the clock cycle. As in the rest of this section, we handle all forwarding except for the value to be stored by a store instruction.

1. ALU-to-ALU forwarding
2. MEM-to-EX forwarding
3. Register file forwarding

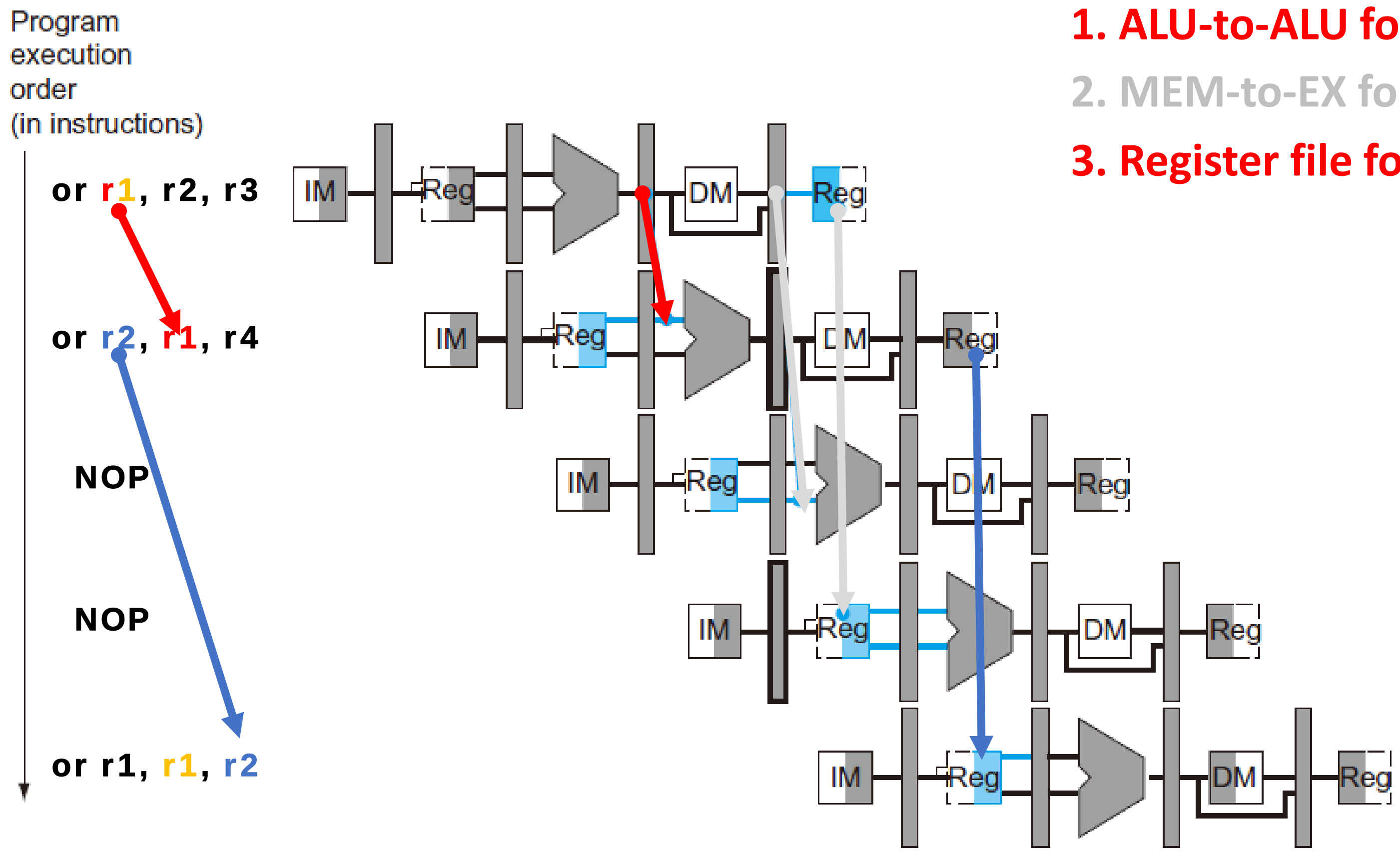


FIGURE 4.53 The dependences between the pipeline registers move forward in time, so it is possible to supply the inputs to the ALU needed by the AND instruction and OR instruction by forwarding the results found in the pipeline registers. The values in the pipeline registers show that the desired value is available before it is written into the register file. We assume that the register file forwards values that are read and written during the same clock cycle, so the add does not stall, but the values come from the register file instead of a pipeline register. Register file "forwarding"—that is, the read gets the value of the write in that clock cycle—is why clock cycle 5 shows register $2 having the value 10 at the beginning and −20 at the end of the clock cycle. As in the rest of this section, we handle all forwarding except for the value to be stored by a store instruction.

1.e Add the minimum number of **nop** instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to the EX stage).

```
I1:  or r1, r2, r3
I2:  or r2, r1, r4
  NOP
  NOP
I3:  or r1, r1, r2
```

With Register file forwarding
: 5 instructions

```
I1:  or r1, r2, r3
I2:  or r2, r1, r4
  NOP
  NOP
  NOP
I3:  or r1, r1, r2
```

Without Register file forwarding
: 6 instructions

1.f What is the total execution time of this instruction sequence with only ALU-ALU forwarding? What is the speedup over a no-forwarding pipeline?

With Register file forwarding

- **No-forwarding**
  - **11 cycles x 150 ps = 1650 ps**
- **ALU-ALU forwarding**
  - **9 cycles x 180 ps = 1620 ps**
- **Speedup**
  - **1650 / 1620 = 1.019**

Without Register file forwarding

- **No-forwarding:**
  - **13 cycles x 150 ps = 1950ps**
- **ALU-ALU forwarding**
  - **10 cycles x 180 ps = 1800 ps**
- **Speedup**
  - **1950 / 1800 = 1.083**

Problem 2

2.a What is the total execution time of this instruction sequence in the 5-stage pipeline that <u>only has one memory</u>?

```
sw r16,12(r6)
lw r16,8(r6)
beq r5,r4,Label       # Assume r5!=r4
add r5,r1,r4
slt r5,r15,r4
```

2.a What is the total execution time of this instruction sequence in the 5-stage pipeline that <u>only has one memory</u>?

```
sw r16,12(r6)
lw r16,8(r6)
beq r5,r4,Label      # Assume r5!=r4
add r5,r1,r4
slt r5,r15,r4
```

- **Answer:** It cannot fetch instruction when **sw** or **lw** accesses the memory. There is two **sw** and **lw**, so it requires two instruction stalls.
  - **11 Cycles**

2.b … MEM and EX stages can be overlapped and the pipeline has only 4 stages. Change this code to accommodate this changed ISA. … what speedup is achieved in this instruction sequence?

```
sw r16,12(r6)

lw r16,8(r6)
beq r5,r4,Label
add r5,r1,r4
slt r5,r15,r4
```

Case 1

```
addi r6,r6,12
sw r16,0(r6)
addi r6,r6,-4
lw r16,0(r6)
beq r5,r4,Label
add r5,r1,r4
slt r5,r15,r4
```

Case 2

2.b … MEM and EX stages can be overlapped and the pipeline has only 4 stages. Change this code to accommodate this changed ISA. … what speedup is achieved in this instruction sequence?

Case 1

- **5-stage pipeline, 5 instructions**
  - **9 cycles**
- **4-stage pipeline, 5 instructions**
  - **8 cycles**
- **Speedup**
  - **9/8 = 1.124**

Case 2

- **5-stage pipeline, 5 instructions**
  - **9 cycles**
- **4-stage pipeline, 7 instructions**
  - **10 cycles**
- **Speedup**
  - **9/10 = 0.9**

2.c Assuming **stall-on-branch** and no delay slots, what speedup is achieved on this code if branch outcomes are determined in the **ID** stage, relative to the execution where branch outcomes are determined in the **EX** stage?

	1	2	3	4	5	6	7	8	9	10	11
sw	▨	▨	▨	▨	▨						
lw		▨	▨	▨	▨	▨					
beq			IF	ID	EX	▨	▨				
add						▨	▨	▨	▨	▨	
slt							▨	▨	▨	▨	▨

	1	2	3	4	5	6	7	8	9	10	11
sw	▨	▨	▨	▨	▨						
lw		▨	▨	▨	▨	▨					
beq			IF	ID	▨	▨	▨				
add					▨	▨	▨	▨	▨		
slt						▨	▨	▨	▨	▨	

- **11 Cycles**
- **Speedup**
  - **11/10 = 1.1**

- **10 Cycles**

2.d …When EX and MEM are done in a single stage, most of their work can be done in parallel. As a result, the resulting <u>EX/MEM stage has a latency that is the original MEM stage latency plus 10 ps</u> needed for the work that could not be done in parallel.

IF	ID	EX	MEM	WB
160ps	100ps	120ps	150ps	80ps

IF	ID	EX/MEM	WB
160ps	100ps	150+10ps	80ps

Case 1
- **Speedup**
  - **9/8 = 1.124**

Case 2
- **Speedup**
  - **9/10 = 0.9**

2.e …speedup calculation from 2.c. … Assume that the latency ID stage increases by 100% and the latency of the EX stage decreases by 20 ps when branch outcome resolution is moved from EX to ID.

IF	ID	EX	MEM	WB
160ps	100ps	120ps	150ps	80ps

IF	ID	EX	MEM	WB
160ps	200ps	100ps	150ps	80ps

Branch outcome at EX

- **11 Cycles x 160 ps = 1760 ps**
- **Speedup**
  - **1760 / 2000 = 0.88**

Branch outcome at ID

- **10 cycles x 200 ps = 2000 ps**

2.f Assuming stall-on-branch …if **beq** address computation is moved to the MEM stage? What is the speedup from this change? Assume that the latency of the EX stage is reduced by 20 ps and the latency of the MEM stage is unchanged when branch outcome resolution is moved from EX to MEM.

IF	ID	EX	MEM	WB
160ps	100ps	120ps	150ps	80ps

IF	ID	EX	MEM	WB
160ps	100ps	100ps	150ps	80ps

2.f Assuming stall-on-branch …if **beq** address computation is moved to the MEM stage? What is the speedup from this change? Assume that the latency of the EX stage is reduced by 20 ps and the latency of the MEM stage is unchanged when branch outcome resolution is moved from EX to MEM.

	1	2	3	4	5	6	7	8	9	10	11
sw	▨	▨	▨	▨	▨						
lw		▨	▨	▨	▨	▨					
beq			IF	ID	EX	▨	▨				
add						▨	▨	▨	▨	▨	
slt							▨	▨	▨	▨	▨

	1	2	3	4	5	6	7	8	9	10	11	12
sw	▨	▨	▨	▨	▨							
lw		▨	▨	▨	▨	▨						
beq			IF	ID	EX	MEM	▨					
add							▨	▨	▨	▨	▨	
slt								▨	▨	▨	▨	▨

- **11 Cycles**
- **Speedup**
  - **12/11 = 0.92**

- **12 Cycles**

Problem 3

3.a What is the extra CPI due to mispredicted branches with the Always-Taken predictor? Assume that branch outcomes are determined in the EX stage, …

R-Type	BEQ	JMP	LW	SW
40%	15%	15%	25%	5%

Always-Taken	Always-Not-Taken	2-Bit
35%	65%	90%

- **Answer:** Every mispredicted instruction generates two additional cycles. Therefore,
  - **Extra CPI = $2 \times (1-0.35) \times 0.15 = 0.195$**

R-Type	BEQ	JMP	LW	SW
40%	15%	15%	25%	5%

Always-Taken	Always-Not-Taken	2-Bit
35%	65%	90%

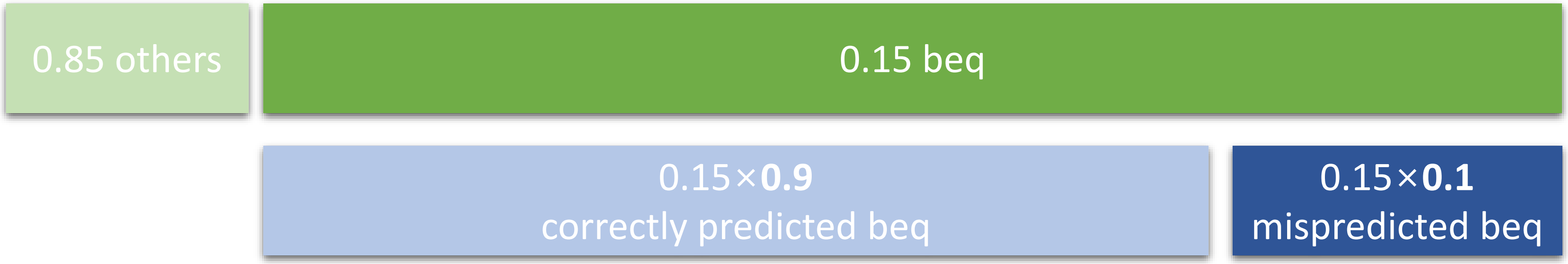
3.b Repeat 3.a for the "Always-Not-Taken" predictor.

- **Extra CPI=2 × (1-0.65) × 0.15=0.105**

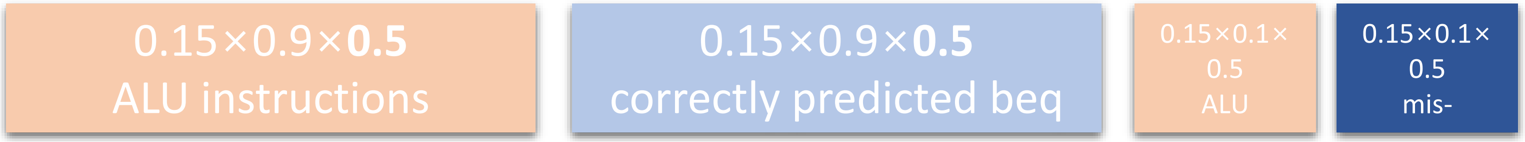
3.c Repeat 3.a for the "2-Bit predictor" predictor.

- **Extra CPI=2 × (1-0.90)×0.15=0.03**

3.d With the 2-Bit predictor, what speedup would be achieved if we could convert half of the branch instructions in a way that replaces a branch instruction with an ALU instruction? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.



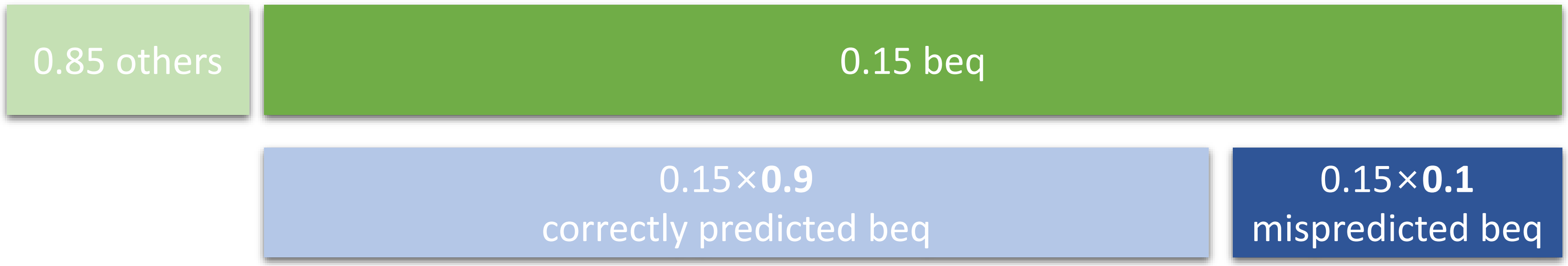
$$CPI_{No\ conversion} = 1 \times (0.85 + 0.15 \times 0.9) + 3 \times (0.15 \times 0.1) = 1.03$$


0.15×0.9×0.5 ALU instructions
0.15×0.9×0.5 correctly predicted beq
0.15×0.1× 0.5 ALU
0.15×0.1× 0.5 mis-

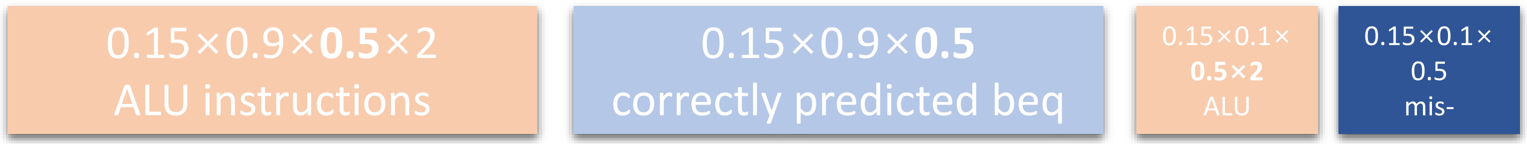
$$CPI_{Conversion} = 1 \times (0.85 + 0.15 \times 0.9 \times 0.5 + 0.15 \times 0.9 \times 0.5 + 0.15 \times 0.1 \times 0.5) + 3 \times (0.15 \times 0.1 \times 0.5) = 1.015$$

$$Speedup = \frac{CPI_{No\ conversion}}{CPI_{Conversion}} = \frac{1.03}{1.015} = 1.015$$

3.e … in a way that replaced each branch instruction with two ALU instructions? …


0.85 others
0.15 beq
0.15×**0.9** correctly predicted beq
0.15×**0.1** mispredicted beq

$$\boldsymbol{CPI_{No\ conversion} = 1 \times (0.85 + 0.15 \times 0.9) + 3 \times (0.15 \times 0.1) = 1.03}$$

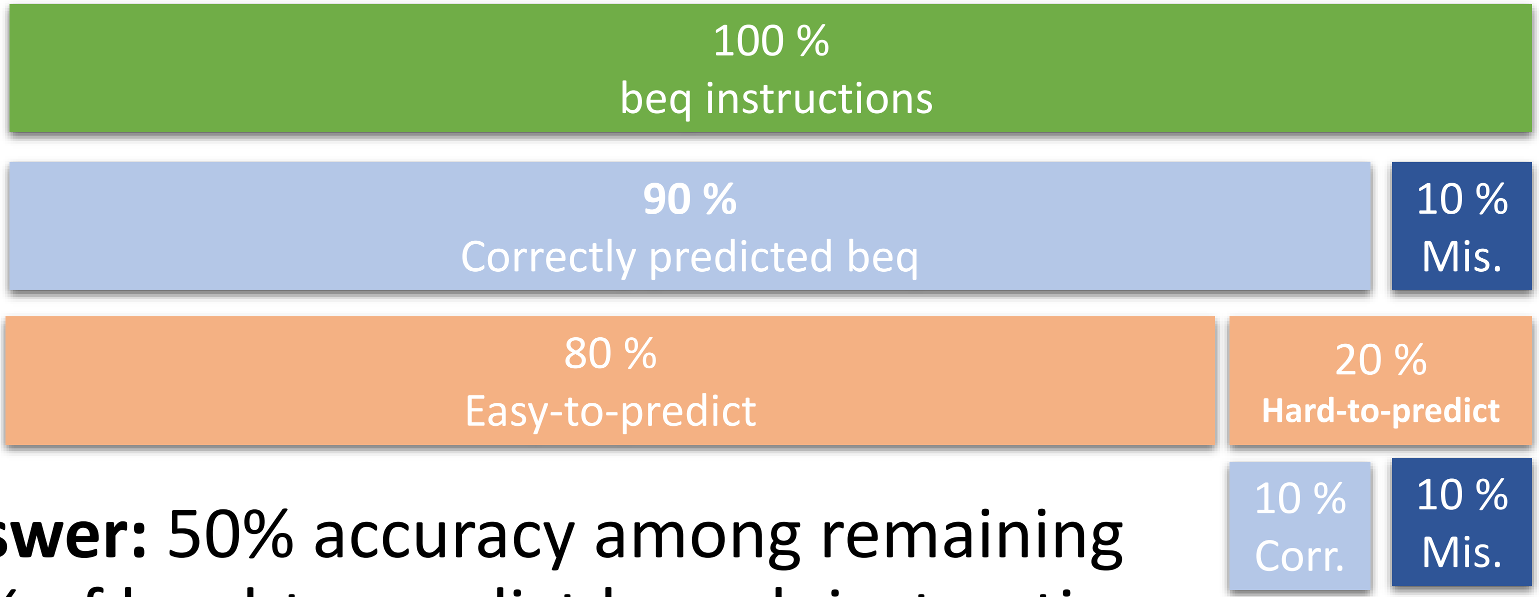

0.15×0.9×**0.5**×2 ALU instructions
0.15×0.9×**0.5** correctly predicted beq
0.15×0.1× **0.5×2** ALU
0.15×0.1× 0.5 mis-

$$\boldsymbol{CPI_{Conversion} = 1 \times (0.85 + 0.15 \times 0.9 \times 0.5 \times 2 + 0.15 \times 0.9 \times 0.5 + 0.15 \times 0.1 \times 0.5 \times 2) + 3 \times (0.15 \times 0.1 \times 0.5) = 1.09}$$

$$\boldsymbol{Speedup_{execution\ time} = \frac{CPI_{No\ conversion}}{CPI_{Conversion}} = \frac{1.03}{1.09} = 0.945}$$

$$\boldsymbol{Speedup_{CPI} = Speedup_{execution\ time} \times \frac{\#\ of\ instruction_{Conversion}}{\#\ of\ instruction_{No\ conversion}} = 0.945 \times 1.075 = 1.015}$$

3.f Some branch instructions are much more predictable than others. If we know that 80% of all executed branch instructions are easy-to-predict loop-back branches that are always predicted correctly, what is the accuracy of the 2-Bit predictor on the remaining 20% of the branch instructions?



- **Answer:** 50% accuracy among remaining 20% of hard-to-predict branch instruction