Computer Architecture
CSL3020

Deepak Mishra

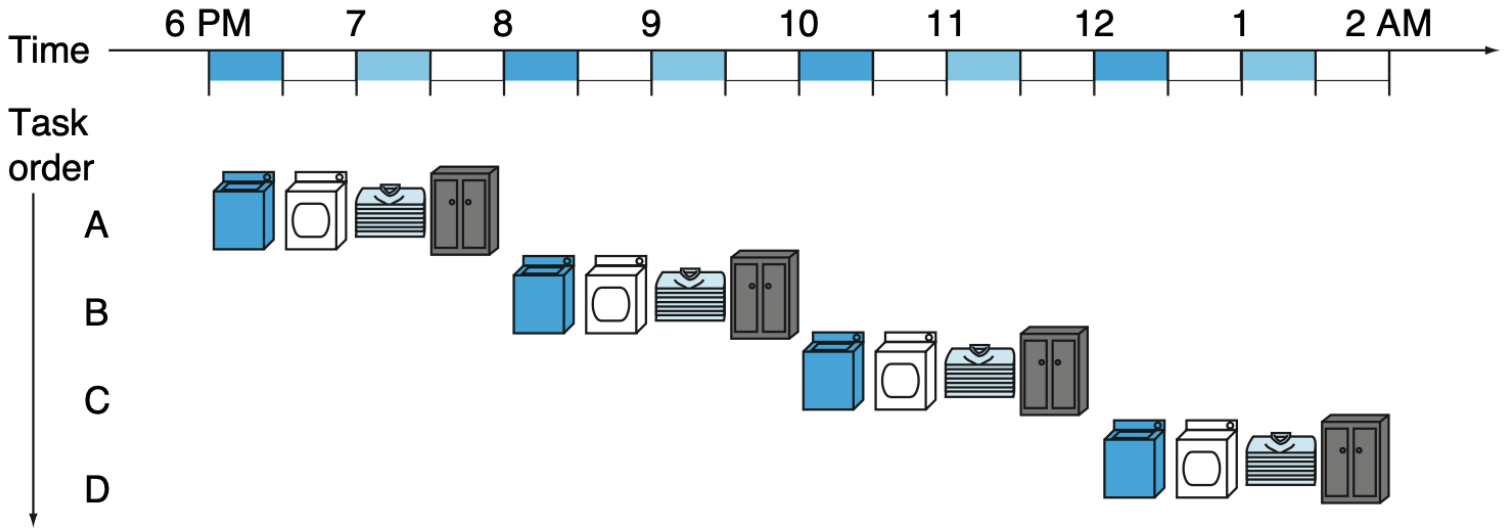
http://home.iitj.ac.in/~dmishra/
Department of Computer Science and Engineering
Indian Institute of Technology Jodhpur

A Simple MIPS Processor

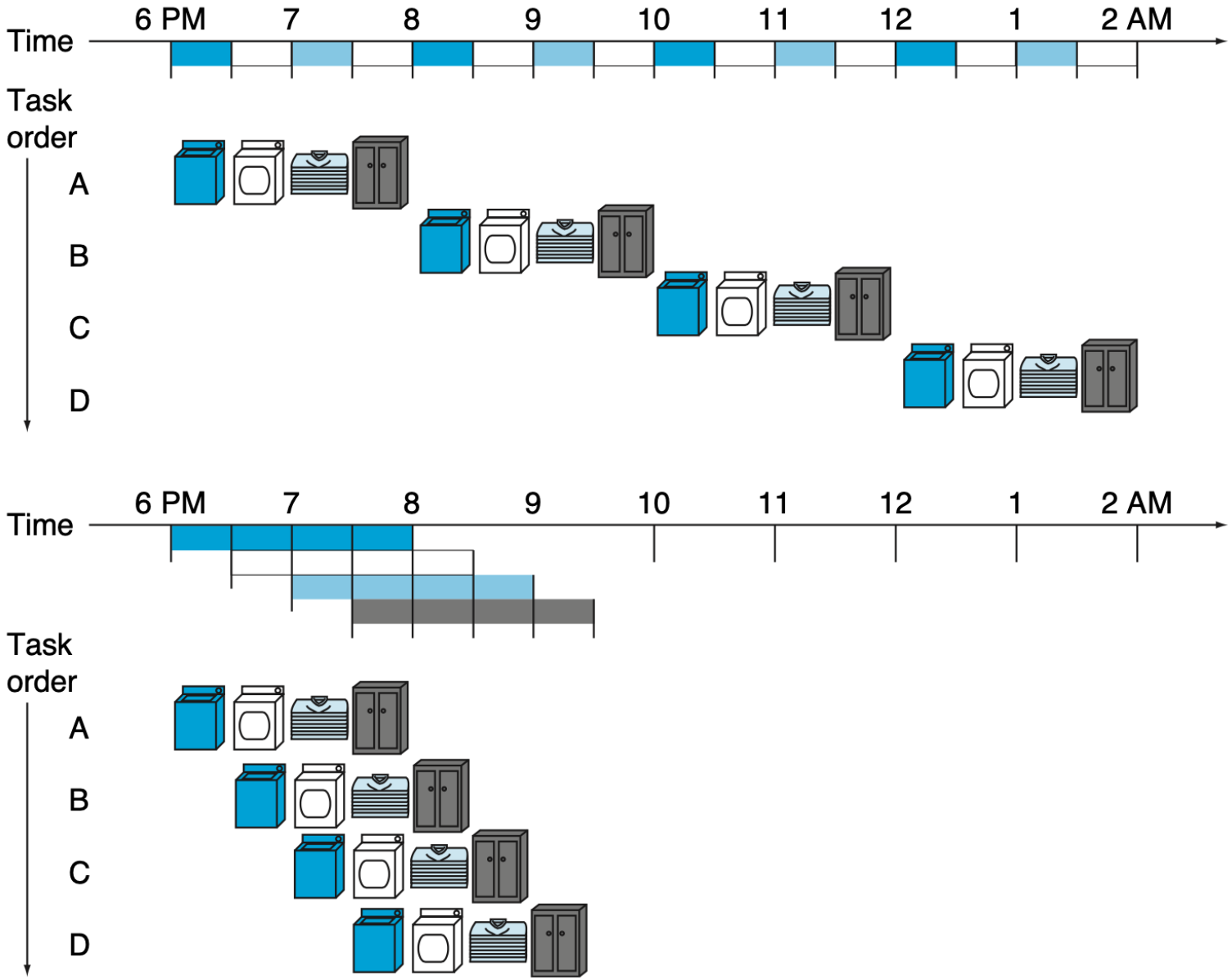
A few comments

- Controls are set using 6-bit opcode field, op ([31:26]), of instruction.
- It is easy to control the operation while using single-cycle implementation.
- However single-cycle design is inefficient as the clock cycle is determined by the longest possible path in the processor.
- Although the CPI is 1, the overall performance of a single-cycle implementation is poor, since the clock cycle is too long.
- An alternative is Pipelining.

Pipelining



Pipelining


Time
6 PM
7
8
9
10
11
12
1
2 AM
Task order
A
B
C
D
Time
6 PM
7
8
9
10
11
12
1
2 AM
Task order
A
B
C
D

Pipelining

- It is an implementation technique in which multiple instructions are overlapped in execution.

Pipelining

- It is an implementation technique in which multiple instructions are overlapped in execution.
- Execution of each instruction is divided into stages.

Pipelining

- It is an implementation technique in which multiple instructions are overlapped in execution.
- Execution of each instruction is divided into stages.
- As long as we have separate resources for each stage, we can pipeline the tasks.

Pipelining

- It is an implementation technique in which multiple instructions are overlapped in execution.
- Execution of each instruction is divided into stages.
- As long as we have separate resources for each stage, we can pipeline the tasks.
- All the pipeline stages take a single clock cycle, so the clock cycle must be long enough to accommodate the slowest operation.

Pipelining

- It is an implementation technique in which multiple instructions are overlapped in execution.
- Execution of each instruction is divided into stages.
- As long as we have separate resources for each stage, we can pipeline the tasks.
- All the pipeline stages take a single clock cycle, so the clock cycle must be long enough to accommodate the slowest operation.
- Under ideal conditions and with a large number of instructions, the speed-up from pipelining is approximately equal to the number of stages.

Pipelining

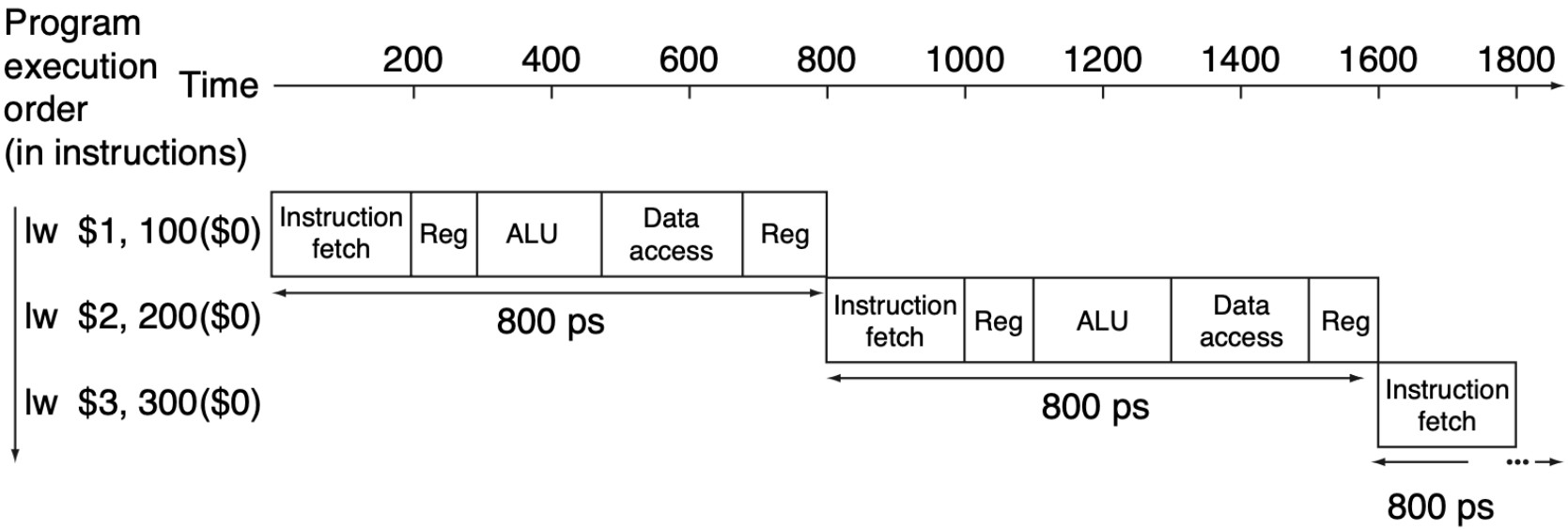
MIPS instructions classically take five steps:

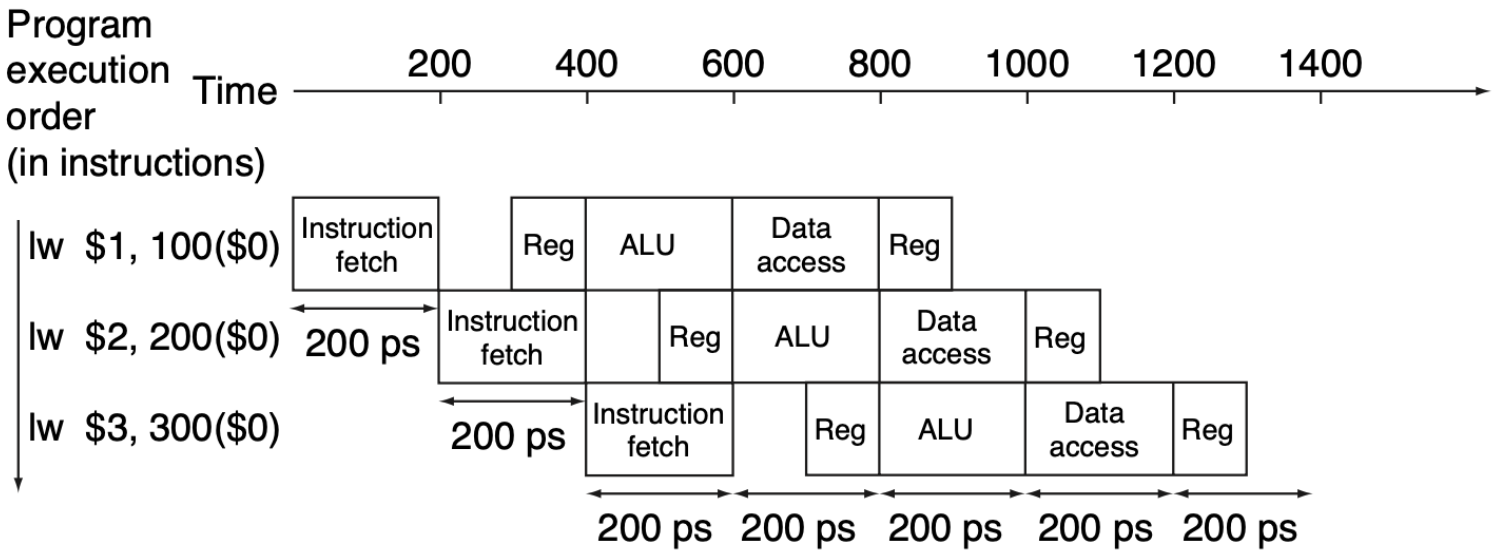
- Fetch instruction from memory.
- Read registers while decoding the instruction.
- Execute the operation or calculate an address.
- Access an operand in data memory.
- Write the result into a register.

Instruction time calculation

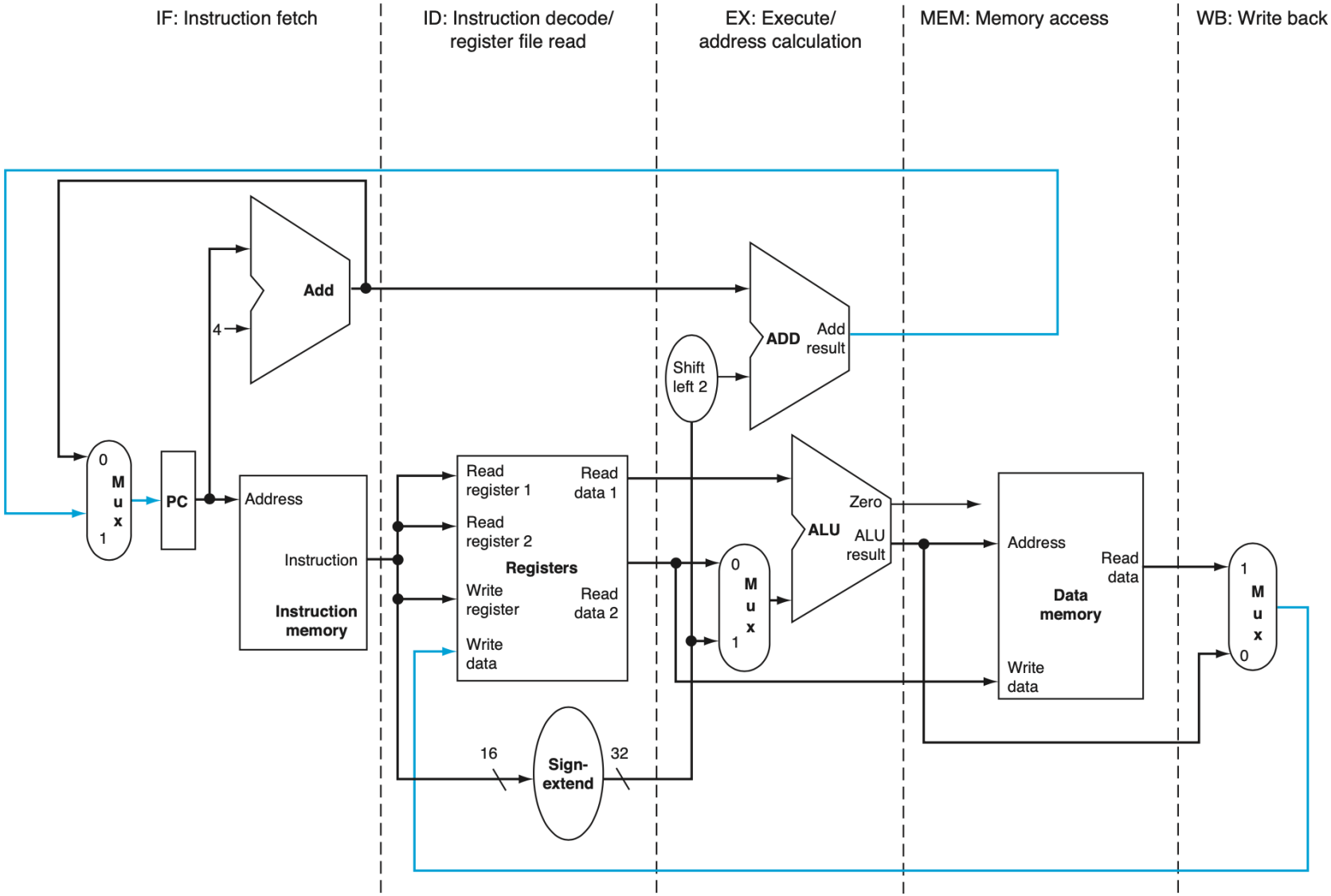
Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (`sw`)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (`add`, `sub`, `AND`, `OR`, `slt`)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (`beq`)	200 ps	100 ps	200 ps			500 ps

Pipelining

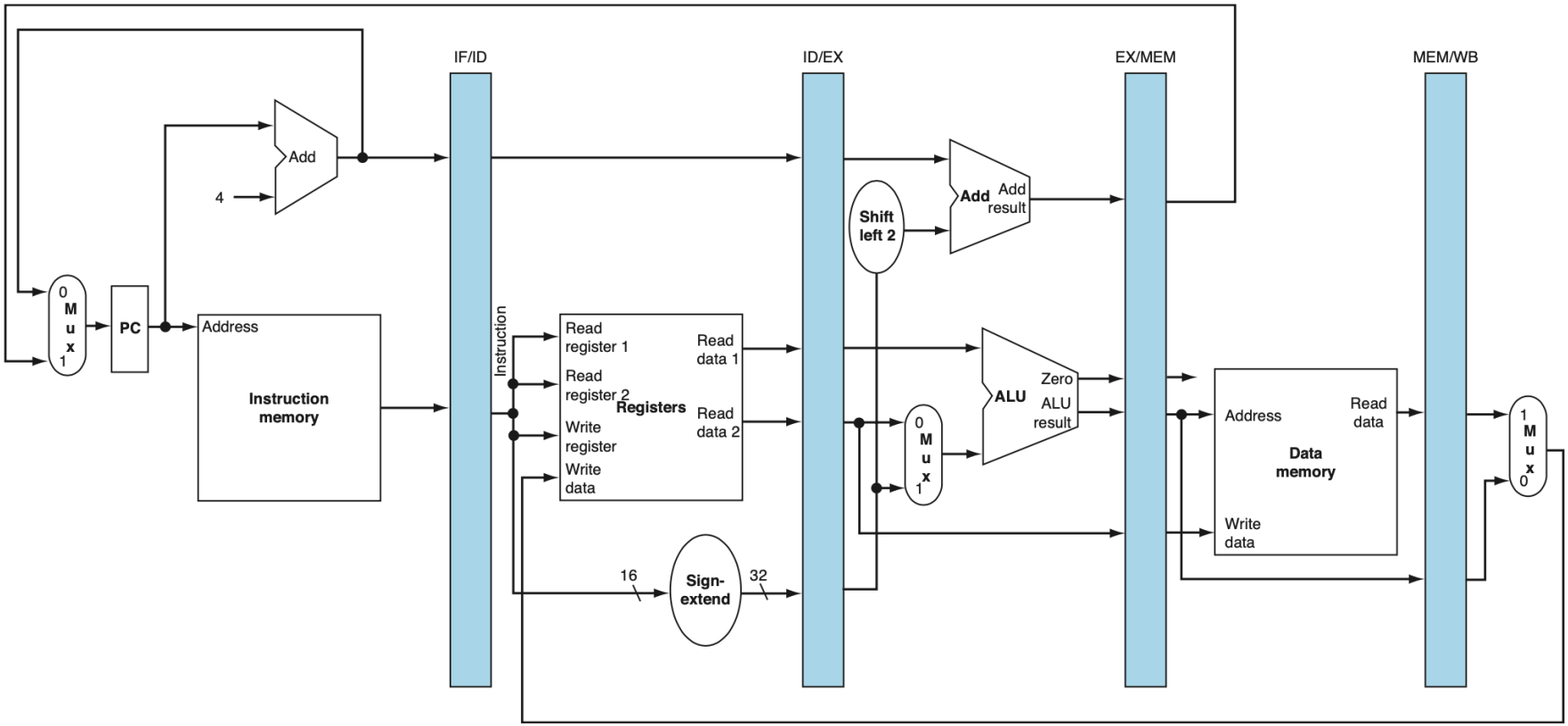

Program execution order (in instructions)
Time
200
400
600
800
1000
1200
1400
1600
1800
lw $1, 100($0)
lw $2, 200($0)
lw $3, 300($0)
Instruction fetch
Reg
ALU
Data access
Reg
800 ps
800 ps
800 ps


Program execution order (in instructions)
Time
200
400
600
800
1000
1200
1400
lw $1, 100($0)
lw $2, 200($0)
lw $3, 300($0)
Instruction fetch
Reg
ALU
Data access
Reg
200 ps
200 ps
200 ps
200 ps
200 ps
200 ps
200 ps

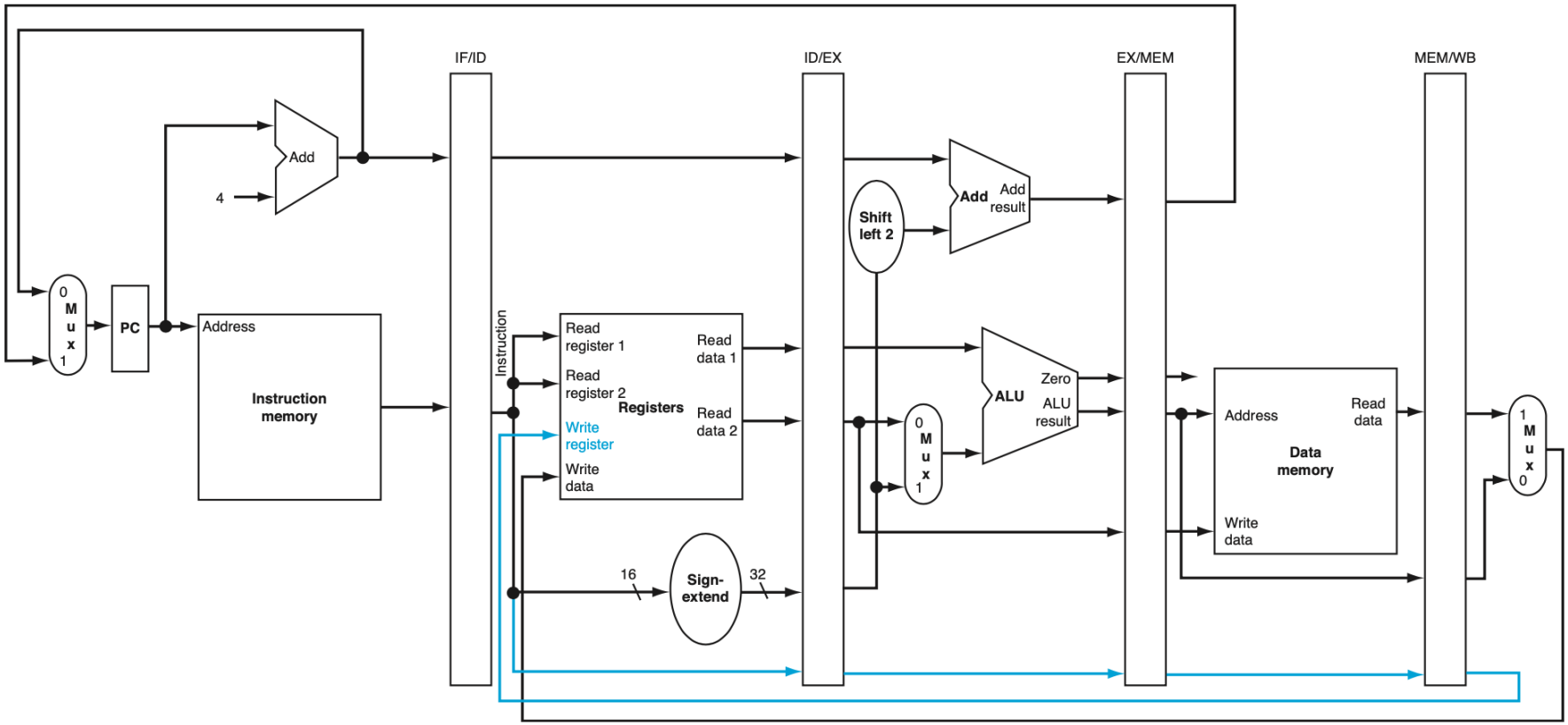
Single-cycle datapath with five stages



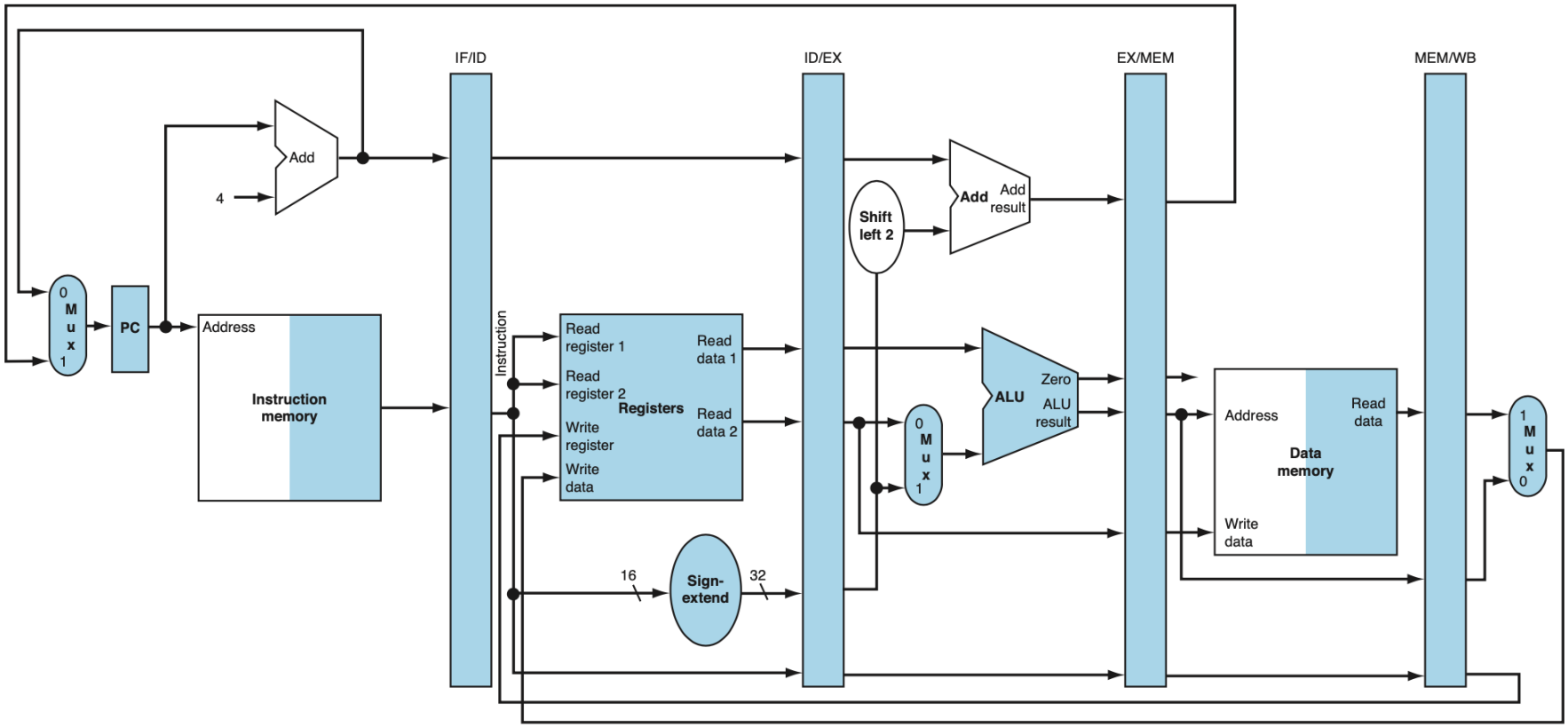
Pipelined version of the datapath with pipeline registers



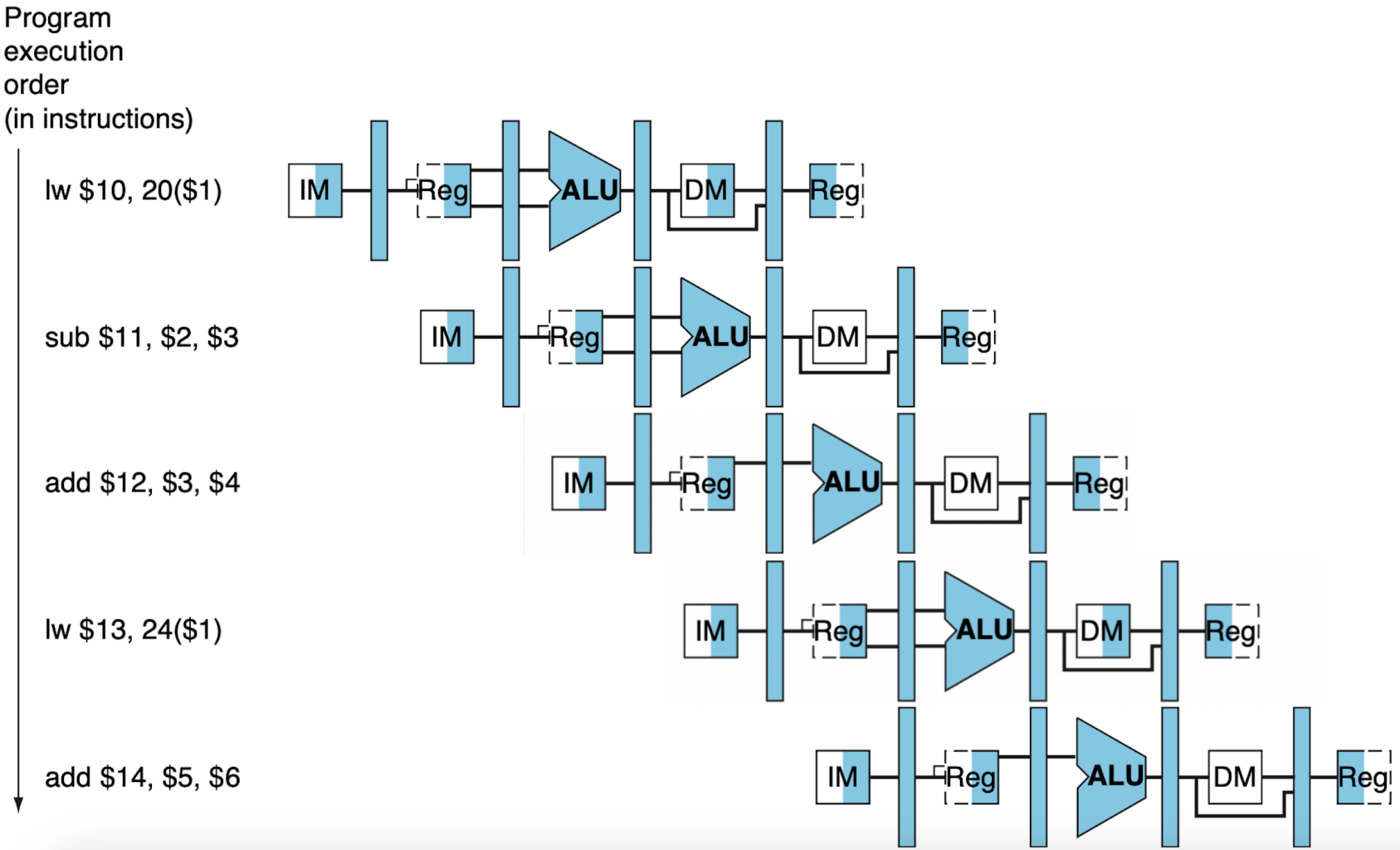
Corrected pipelined datapath to handle load instruction



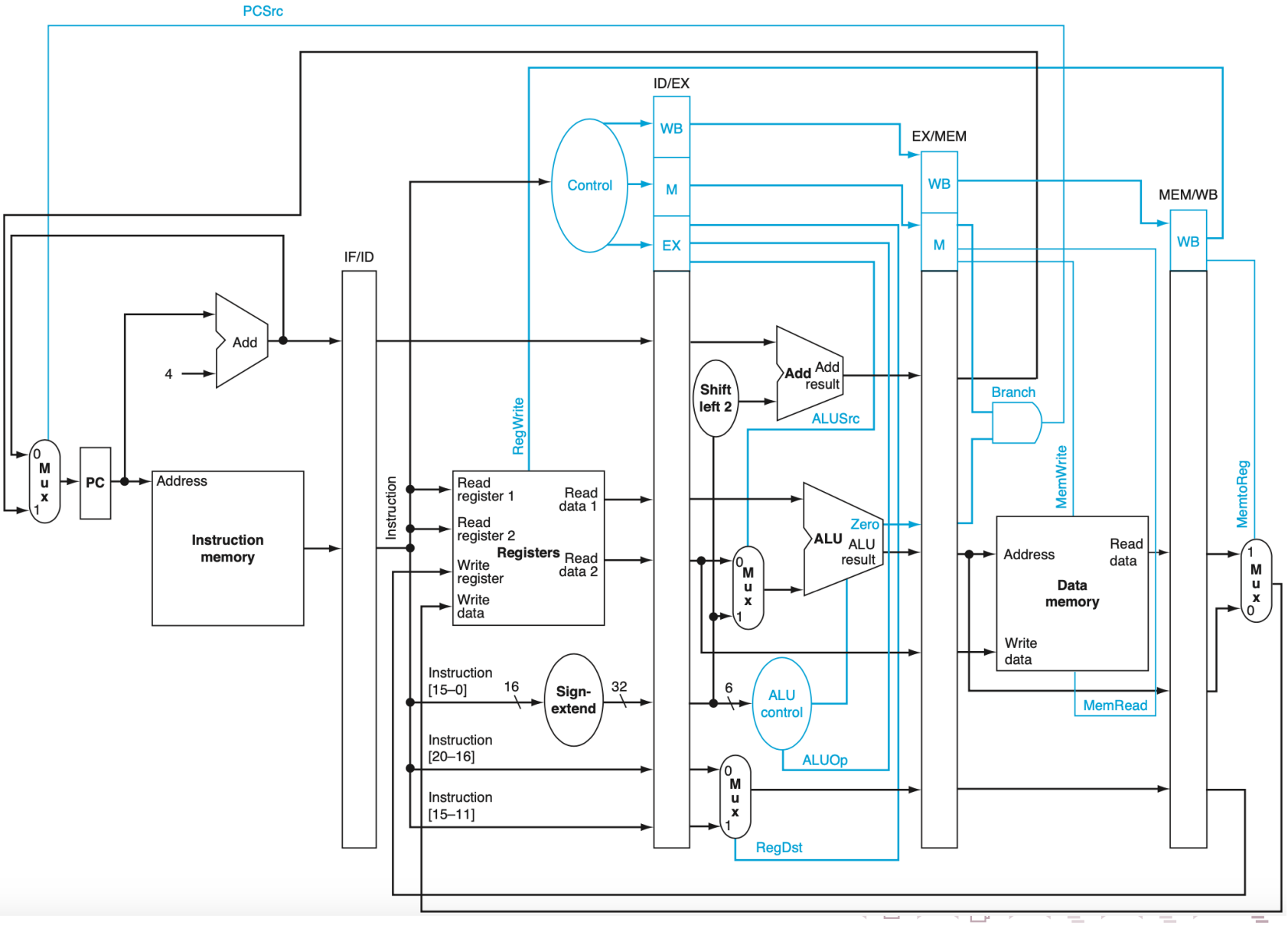
Portion of the datapath that is used in all five stages of a load instruction



Multiple-clock-cycle pipeline diagram of five instructions



Pipelined datapath with control signals



Pipelining Hazards

Hazards are situations in pipelining when the next instruction cannot execute in the following clock cycle.

Pipelining Hazards

Hazards are situations in pipelining when the next instruction cannot execute in the following clock cycle.

1. *Structural Hazards:* Situation in which hardware does not support the combination of instructions.

Pipelining Hazards

Hazards are situations in pipelining when the next instruction cannot execute in the following clock cycle.

1. *Structural Hazards:* Situation in which hardware does not support the combination of instructions.
2. *Data Hazards:* When a planned instruction cannot execute because the data that is needed to execute the instruction is not yet available.

```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```

Pipelining Hazards

Hazards are situations in pipelining when the next instruction cannot execute in the following clock cycle.

1. *Structural Hazards:* Situation in which hardware does not support the combination of instructions.
2. *Data Hazards:* When a planned instruction cannot execute because the data that is needed to execute the instruction is not yet available.

   add $s0, $t0, $t1
   sub $t2, $s0, $t3

3. *Control Hazards:* Also called branch hazard. Arise from the need to make a decision based on the results of one instruction while others are executing.

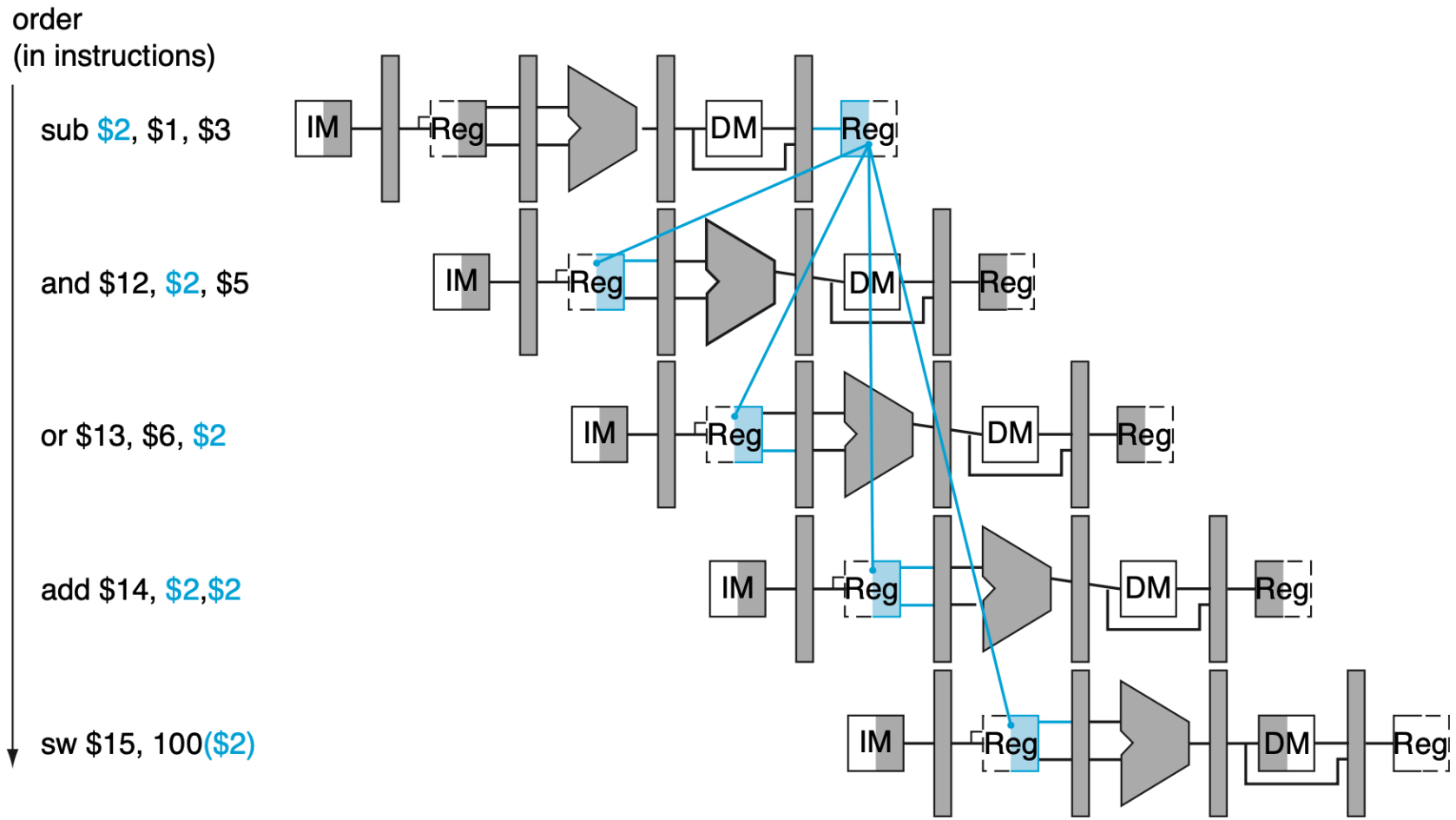
Pipelining Hazards - Data Hazards

Consider the code snippet below

```
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
```

The last four instructions are all dependent on the result in register $2 of the first instruction.

Data dependencies leading to data hazards.



Pipelining Hazards - Data Hazards

One solution to the problem is stalls.

- Stalls delay the operation of the dependent instructions.

Pipelining Hazards - Data Hazards

One solution to the problem is stalls.

- Stalls delay the operation of the dependent instructions.
- Stalls can be introduced by hardware interlocks.

Pipelining Hazards - Data Hazards

One solution to the problem is stalls.

- Stalls delay the operation of the dependent instructions.
- Stalls can be introduced by hardware interlocks.
- These can also be realized using a special instruction called **nop** - An instruction that does no operation to change state.

Pipelining Hazards - Data Hazards

One solution to the problem is stalls.

- Stalls delay the operation of the dependent instructions.
- Stalls can be introduced by hardware interlocks.
- These can also be realized using a special instruction called **nop** - An instruction that does no operation to change state.
- nop is realized by changing control fields of pipeline register to 0.

Pipelining Hazards - Data Hazards

One solution to the problem is stalls.

- Stalls delay the operation of the dependent instructions.
- Stalls can be introduced by hardware interlocks.
- These can also be realized using a special instruction called **nop** - An instruction that does no operation to change state.
- nop is realized by changing control fields of pipeline register to 0.
- Such delays degrade the performance.

Pipelining Hazards - Data Hazards

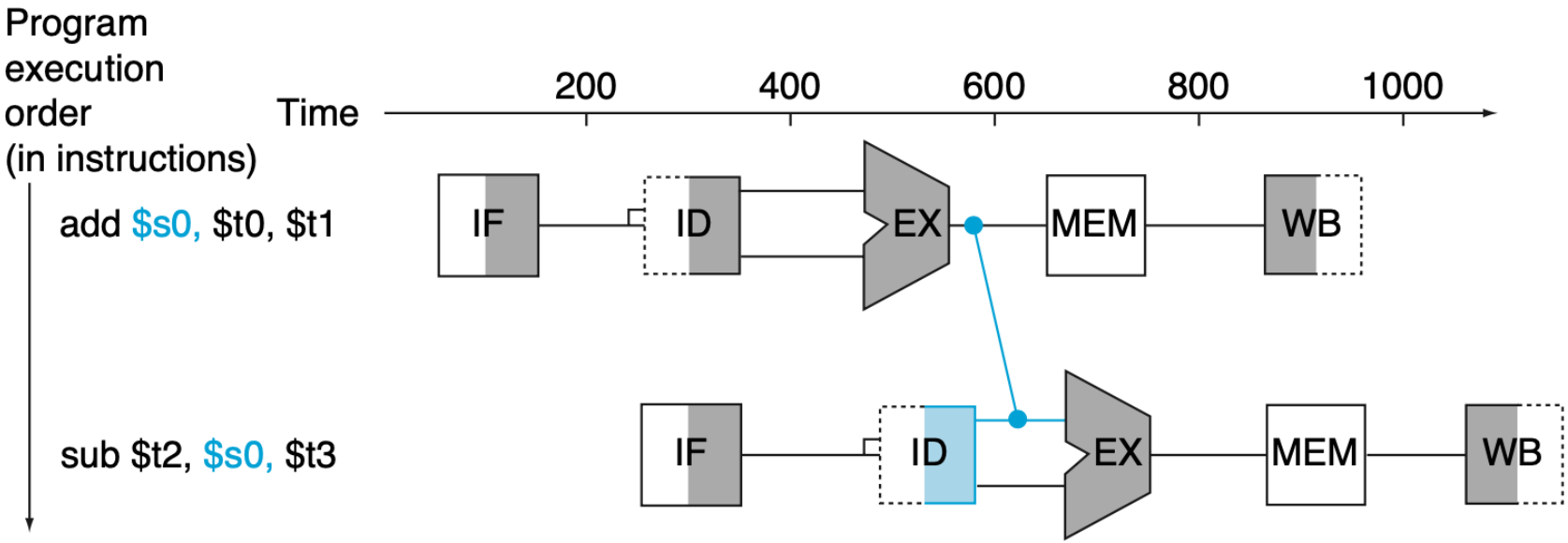
One solution to the problem is stalls.

- Stalls delay the operation of the dependent instructions.
- Stalls can be introduced by hardware interlocks.
- These can also be realized using a special instruction called **nop** - An instruction that does no operation to change state.
- nop is realized by changing control fields of pipeline register to 0.
- Such delays degrade the performance.
- An alternative is to use forwarding.

Pipelining Hazards - Data Hazards

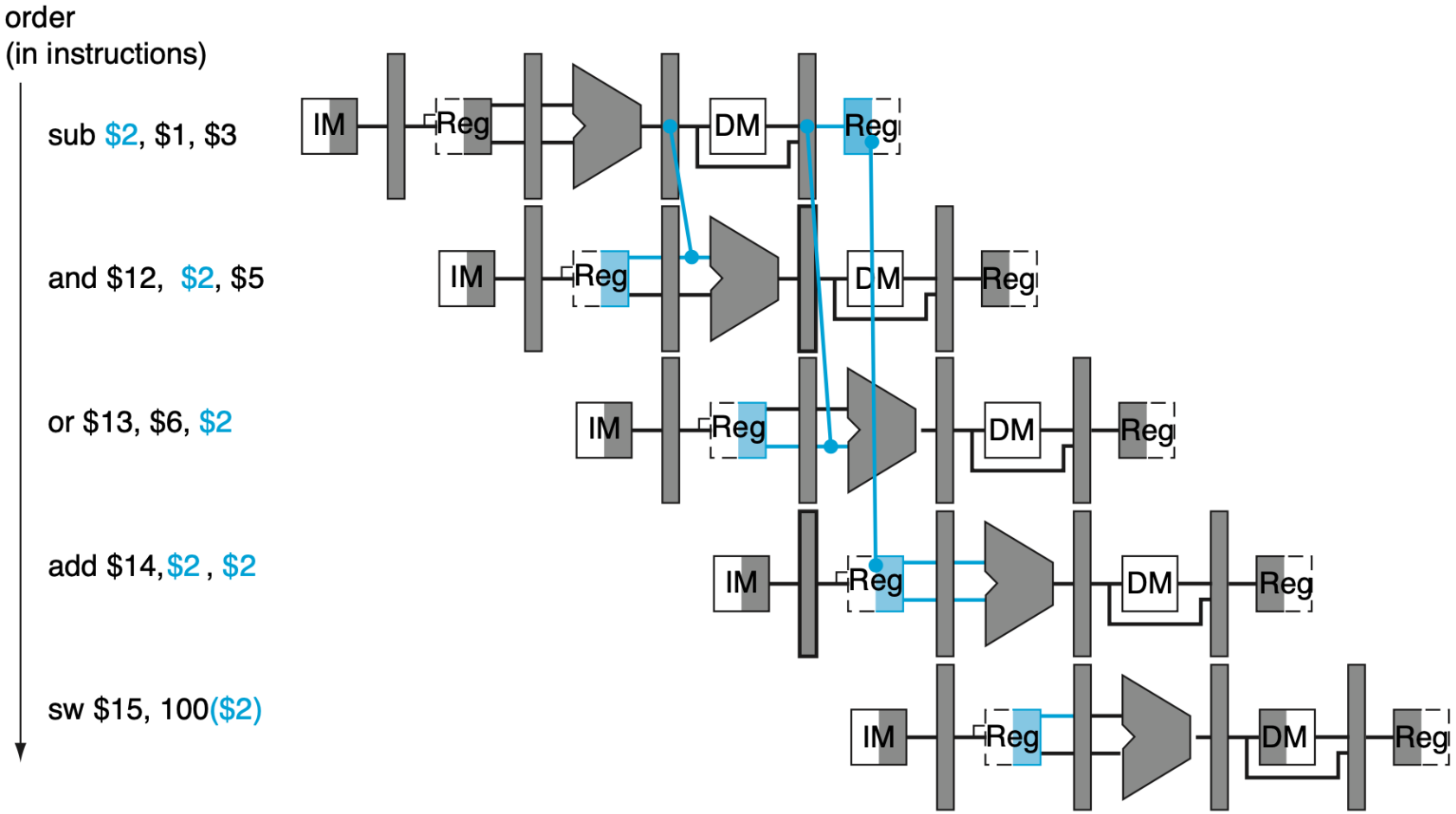
Forwarding, also called bypassing, is retrieval of the missing data element early from the internal resources.

Forwarding, also called bypassing, is retrieval of the missing data element early from the internal resources.



Pipelining Hazards - Data Hazards

Updated data dependencies - no dependency in backward direction



Pipelining Hazards - Data Hazards

Forwarding can help in avoiding following data hazards:

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
1b. EX/MEM.RegisterRd = ID/EX.RegisterRt

Pipelining Hazards - Data Hazards

Forwarding can help in avoiding following data hazards:

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

Pipelining Hazards - Data Hazards

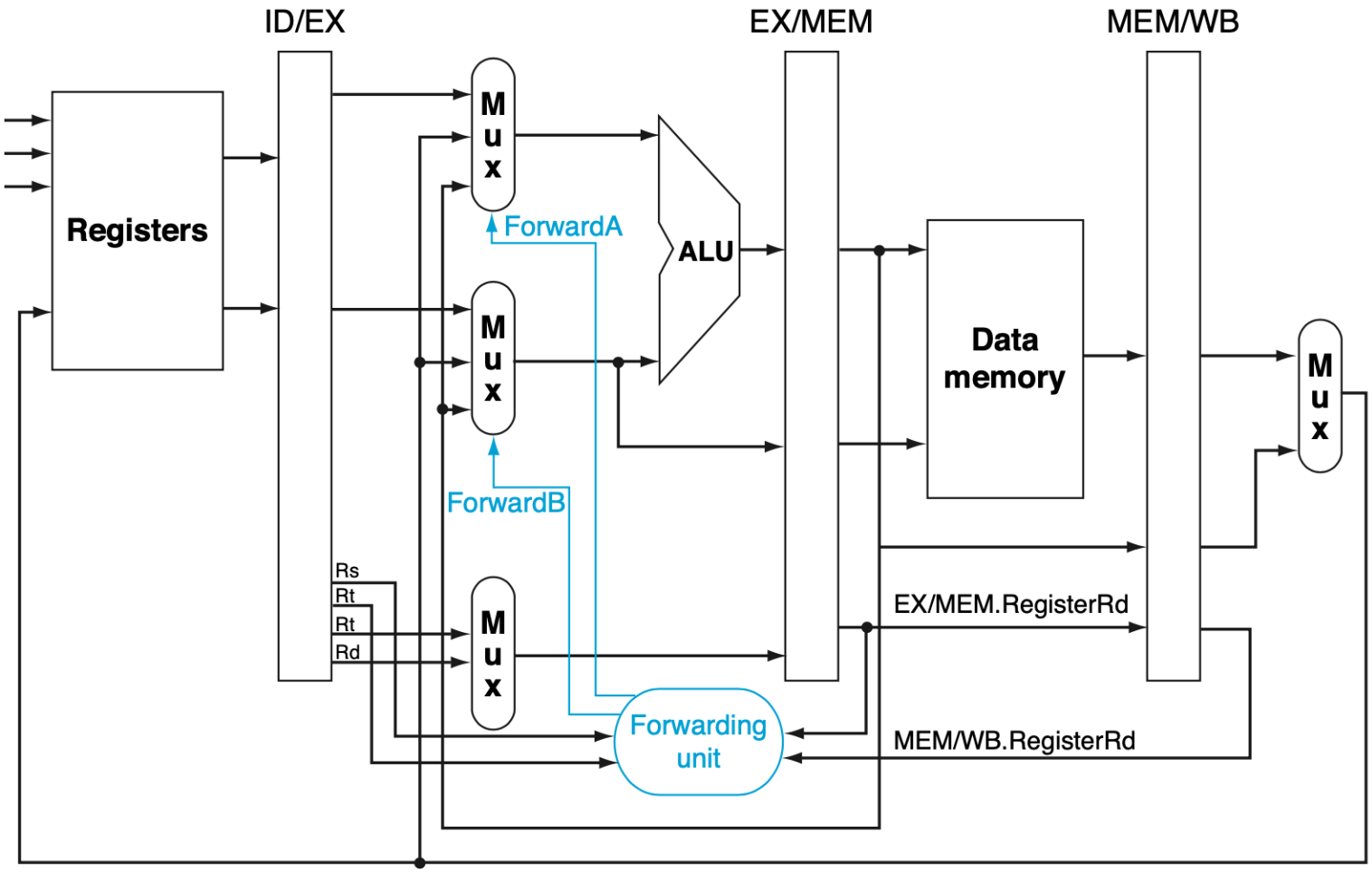
Forwarding can help in avoiding following data hazards:

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

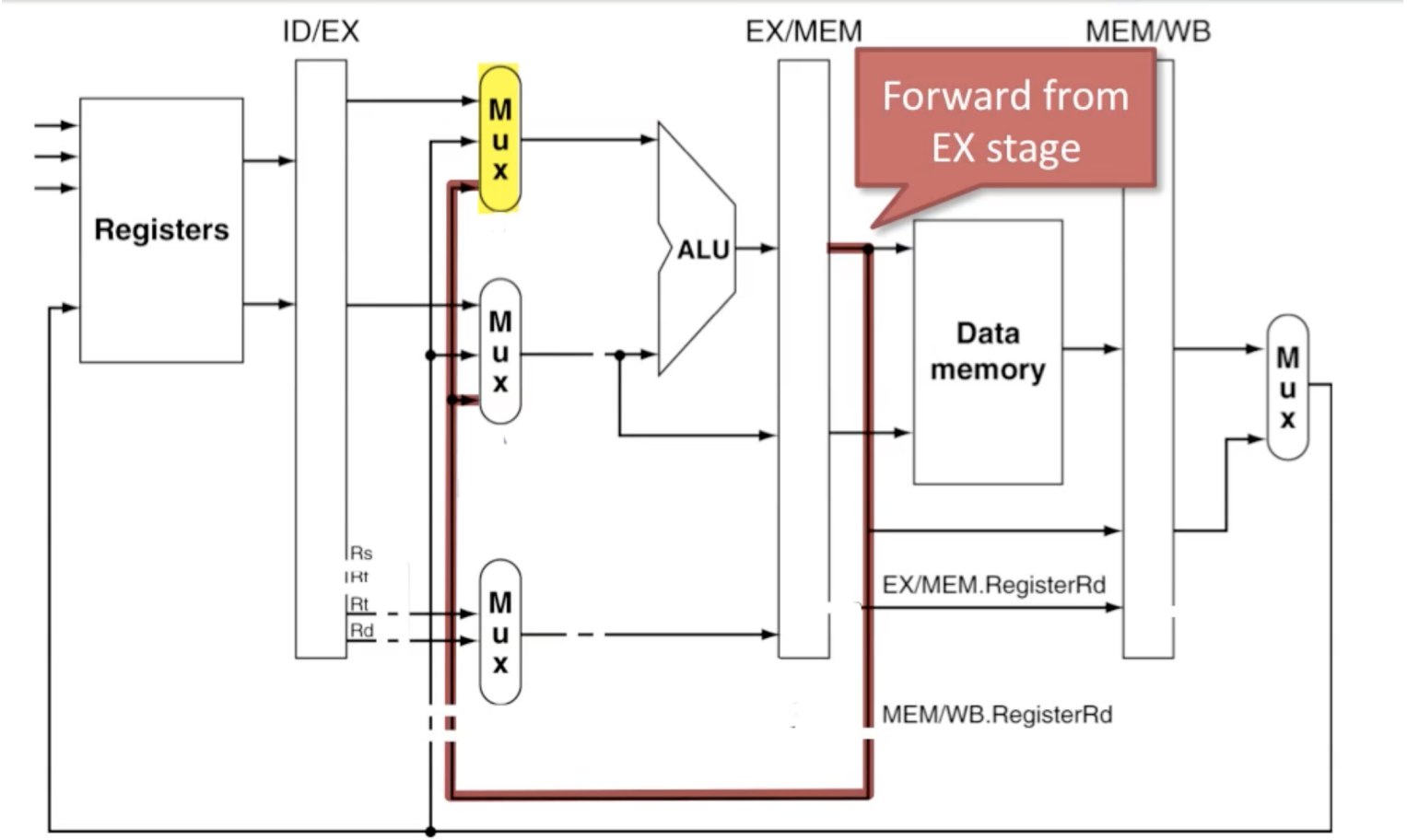
"ID/EX.RegisterRs" refers to the Rs register number found in ID/EX pipeline register.

Pipelining Hazards - Data Hazards

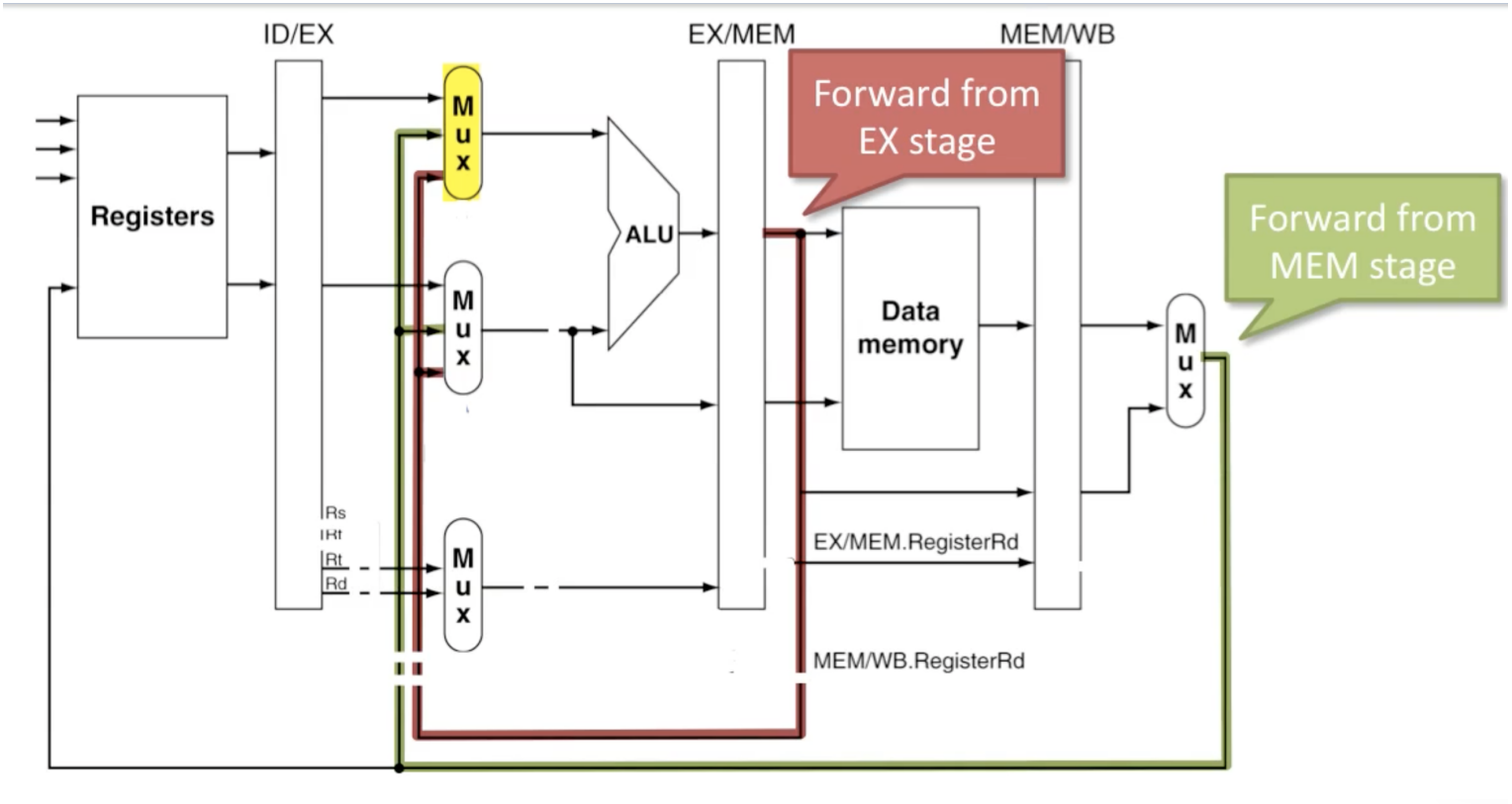
In order to detect the above mentioned data hazards, we need to introduce a "Forwarding unit" in the processor.



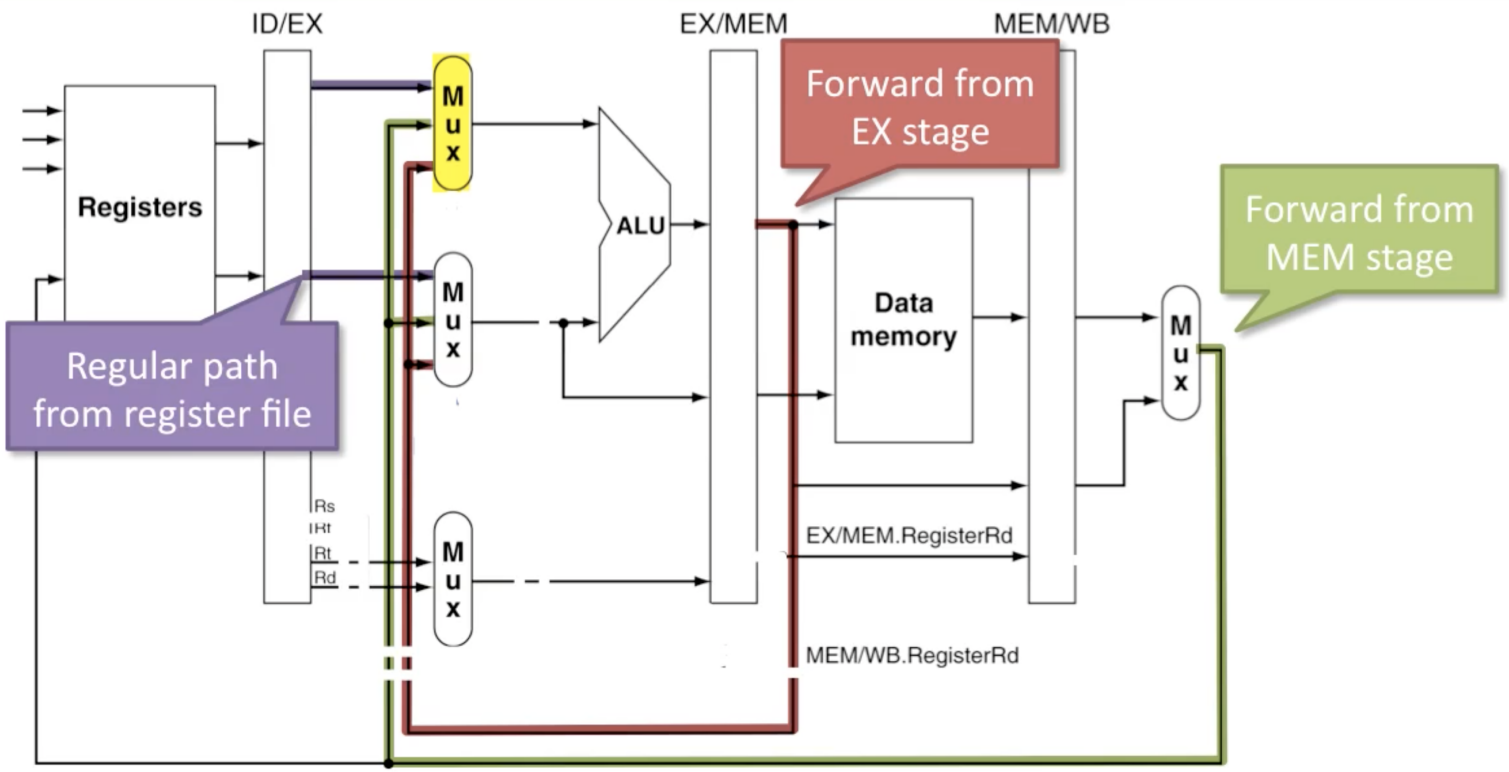
EX/MEM.RegisterRd = ID/EX.RegisterRs



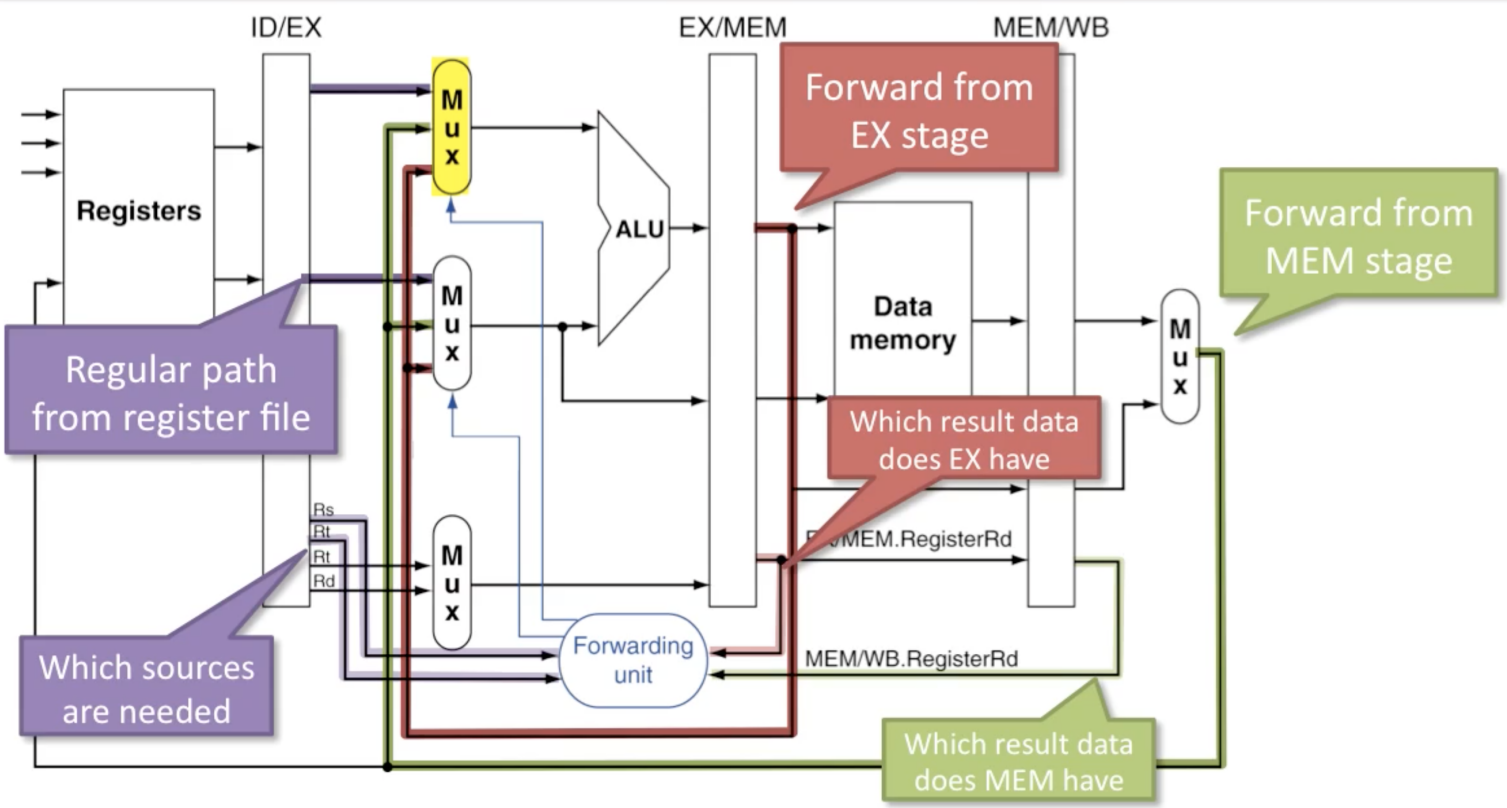
MEM/WB.RegisterRd = ID/EX.RegisterRs



Pipelining Hazards - Data Hazards



Pipelining Hazards - Data Hazards



Pipelining Hazards - Data Hazards

We can now write conditions for detecting hazards and the control signals to resolve them.

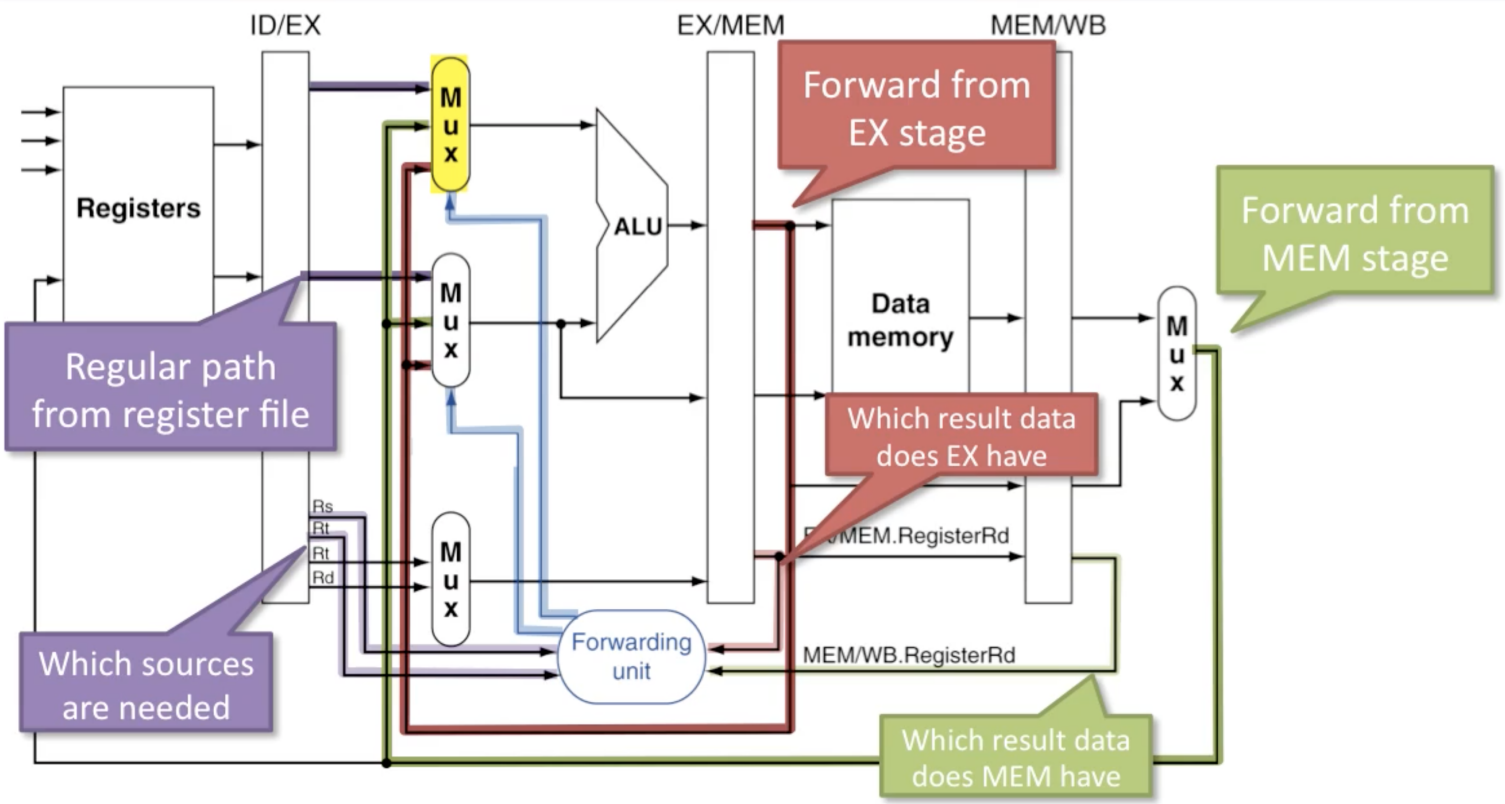
1. *EX hazard:*

```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10
```

2. *MEM hazard:*

```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01

if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01
```

Pipelining Hazards - Data Hazards



Forwarding unit control outputs for multiplexers.

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

Pipelining Hazards - Data Hazards

One case where forwarding cannot save the day is “load-use” hazard.

A situation in which an instruction tries to read a register following a load instruction that writes the same register.

lw \$2, 0(\$4)

and \$12, \$2, \$11

Pipelining Hazards - Data Hazards

One case where forwarding cannot save the day is "load-use" hazard.

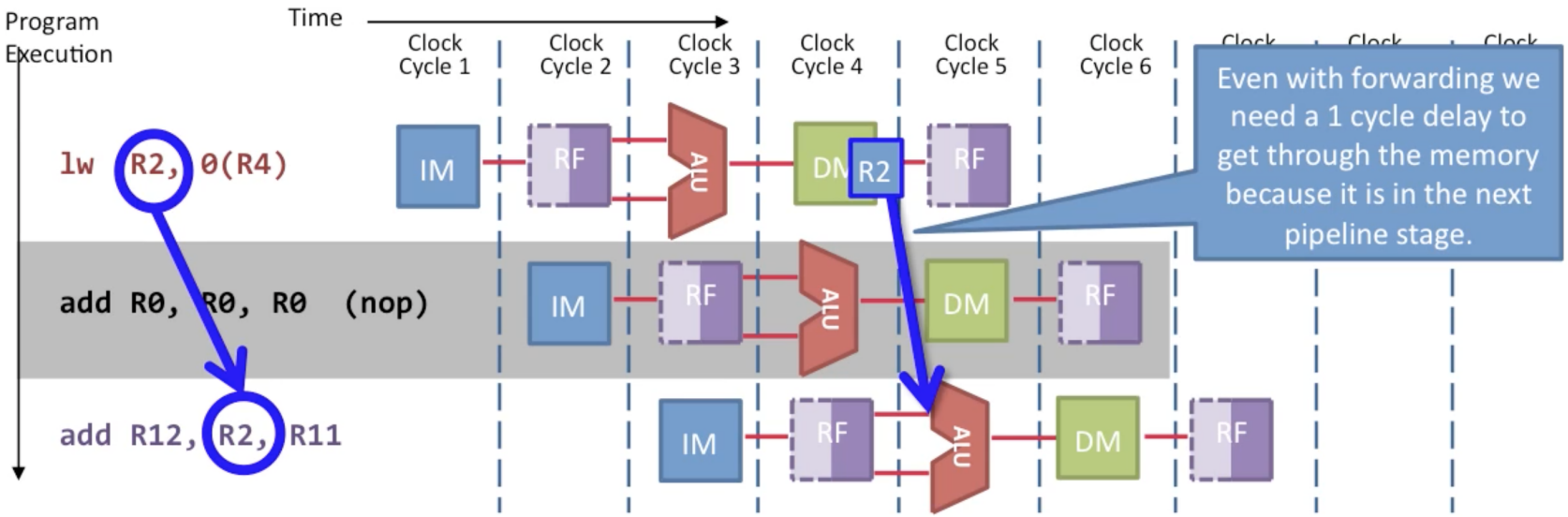
A situation in which an instruction tries to read a register following a load instruction that writes the same register.

lw \$2, 0(\$4)

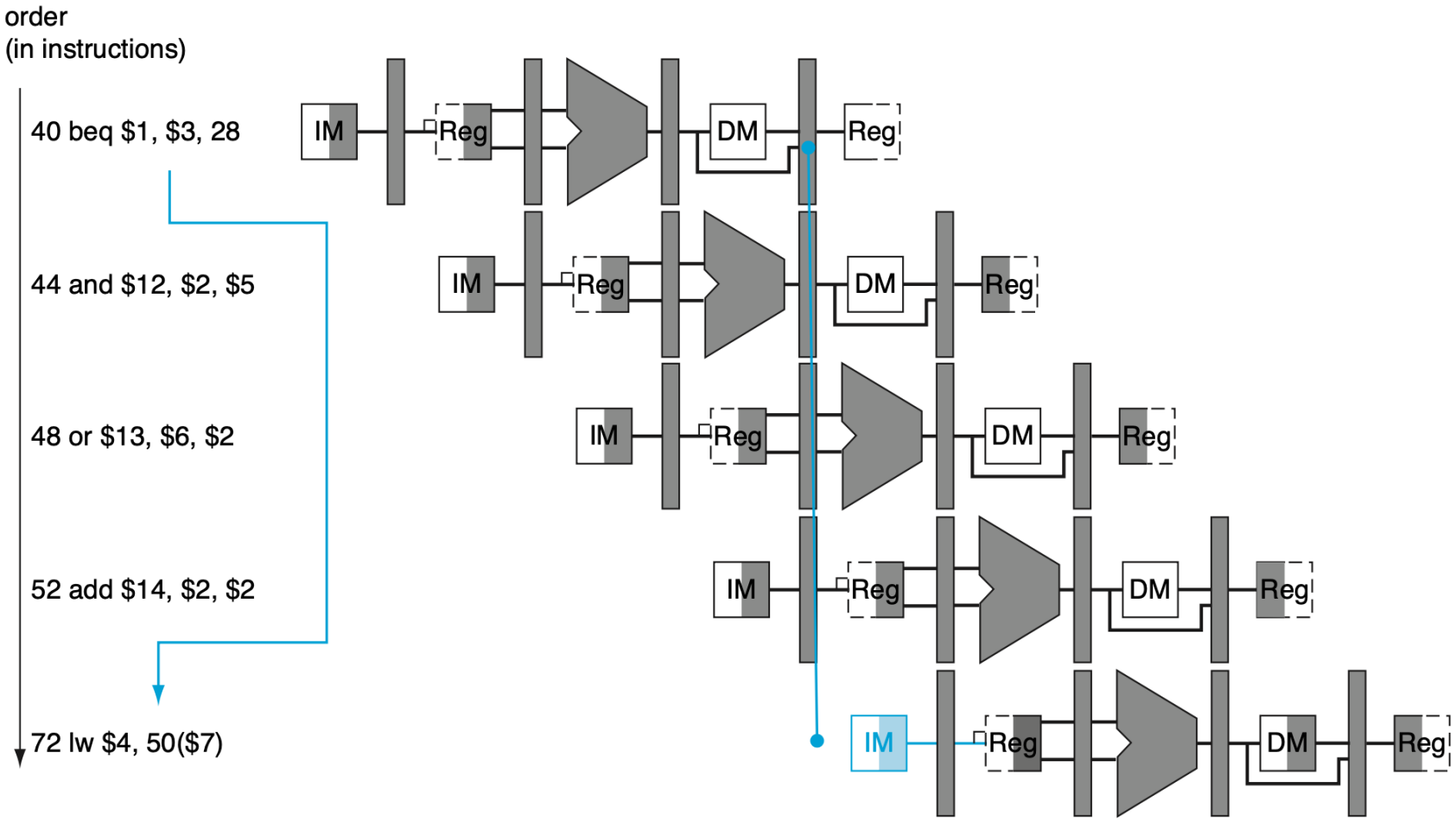
and \$12, \$2, \$11

The solution here is 1-cycle stall.

Pipelining Hazards - Data Hazards



Pipeline hazards involving branches



Pipelining Hazards - Control Hazards

Handling control hazards

- Ignore the branch

Pipelining Hazards - Control Hazards

Handling control hazards

- Ignore the branch
- Branch Speed-up

Pipelining Hazards - Control Hazards

Handling control hazards

- Ignore the branch
- Branch Speed-up
- Branch Prediction

Pipelining Hazards - Control Hazards

Handling control hazards

- Ignore the branch
- Branch Speed-up
- Branch Prediction
- Reorder instruction

Pipelining Hazards - Control Hazards

Handling control hazards

- Ignore the branch
- Branch Speed-up
- Branch Prediction
- Reorder instruction

Pipelining Hazards - Control Hazards

Ignore the branch

- Stalling until the branch is complete is too slow

Pipelining Hazards - Control Hazards

Ignore the branch

- Stalling until the branch is complete is too slow
- A common improvement is to assume that the branch will not be taken

Pipelining Hazards - Control Hazards

Ignore the branch

- Stalling until the branch is complete is too slow
- A common improvement is to assume that the branch will not be taken
- Continue execution

Pipelining Hazards - Control Hazards

Ignore the branch

- Stalling until the branch is complete is too slow
- A common improvement is to assume that the branch will not be taken
- Continue execution
- If the branch is taken, flush instructions in the IF, ID, and EX stages

Branch Speed-up

- Move branch execution earlier in the pipeline

Pipelining Hazards - Control Hazards

Branch Speed-up

- Move branch execution earlier in the pipeline
- MIPS architecture supports fast single-cycle branches

Branch Speed-up

- Move branch execution earlier in the pipeline
- MIPS architecture supports fast single-cycle branches
- Many branches rely only on simple tests can be done with a few gates, without using ALU

Pipelining Hazards - Control Hazards

Branch Speed-up

- Move branch execution earlier in the pipeline
- MIPS architecture supports fast single-cycle branches
- Many branches rely only on simple tests can be done with a few gates, without using ALU
- Branch target address computation and branch decision evaluation, both are moved to ID stage

Branch Speed-up

- Move branch execution earlier in the pipeline
- MIPS architecture supports fast single-cycle branches
- Many branches rely only on simple tests can be done with a few gates, without using ALU
- Branch target address computation and branch decision evaluation, both are moved to ID stage
- This reduces the penalty of a branch to only one instruction

Branch Prediction

- *Dynamic Branch Prediction:* If the branch was taken last time, adjust PC to branch address (1-bit prediction scheme)

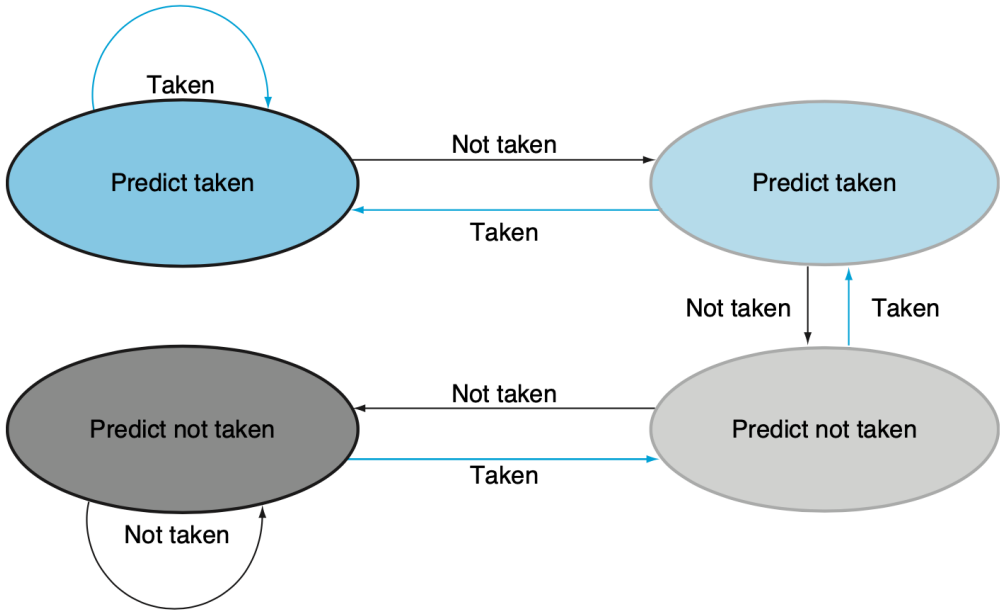
Pipelining Hazards - Control Hazards

Branch Prediction

- *Dynamic Branch Prediction:* If the branch was taken last time, adjust PC to branch address (1-bit prediction scheme)
- Another scheme is 2-bit prediction scheme

Branch Prediction

- *Dynamic Branch Prediction:* If the branch was taken last time, adjust PC to branch address (1-bit prediction scheme)
- Another scheme is 2-bit prediction scheme
- In a 2-bit scheme, a prediction must be wrong twice before it is changed



Reorder instruction

add \$s1, \$s2, \$s3
if \$s2 = 0 then

Reorder instruction

add \$s1, \$s2, \$s3
if \$s2 = 0 then

This can be reordered as
if \$s2 = 0 then
add \$s1, \$s2, \$s3