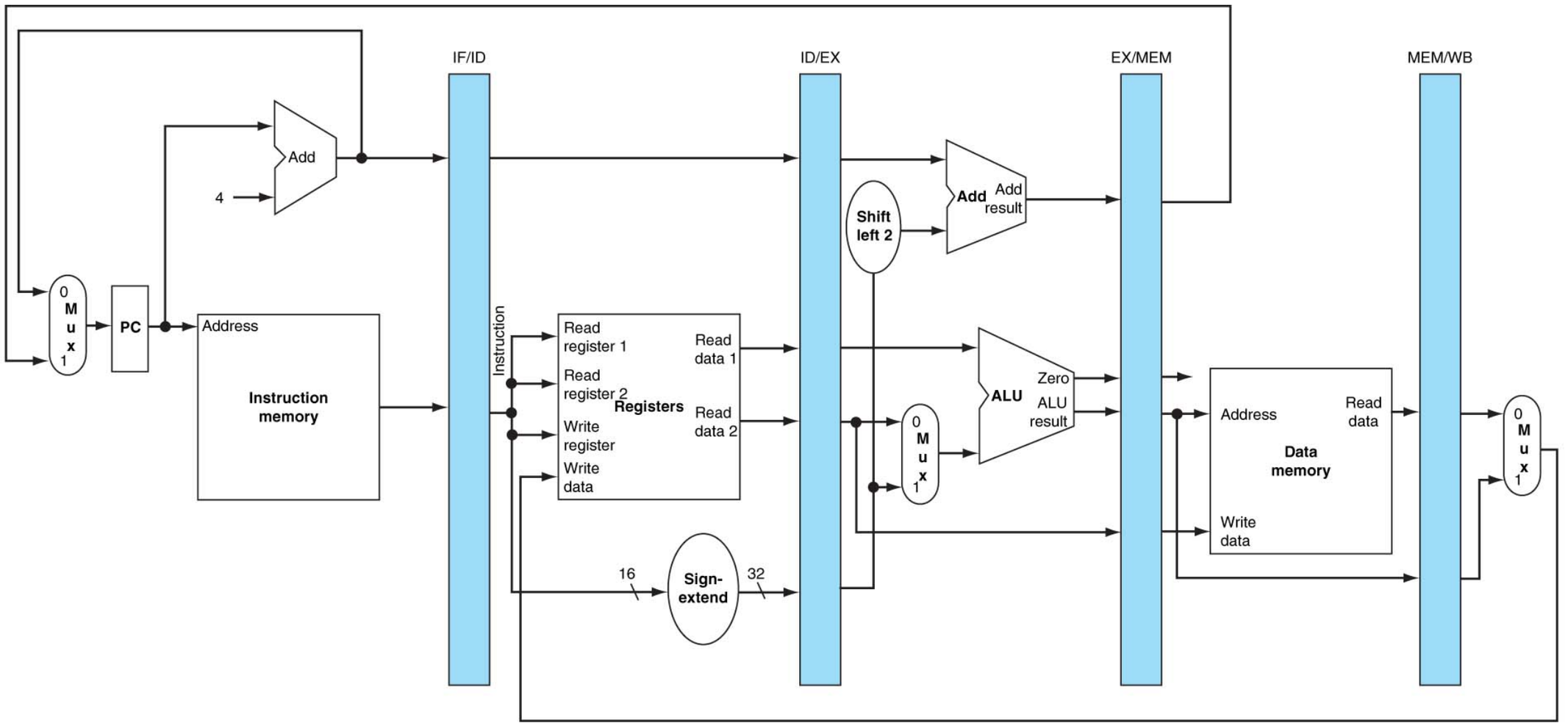
Final Review

Jason Mars

Pipeline In Execution

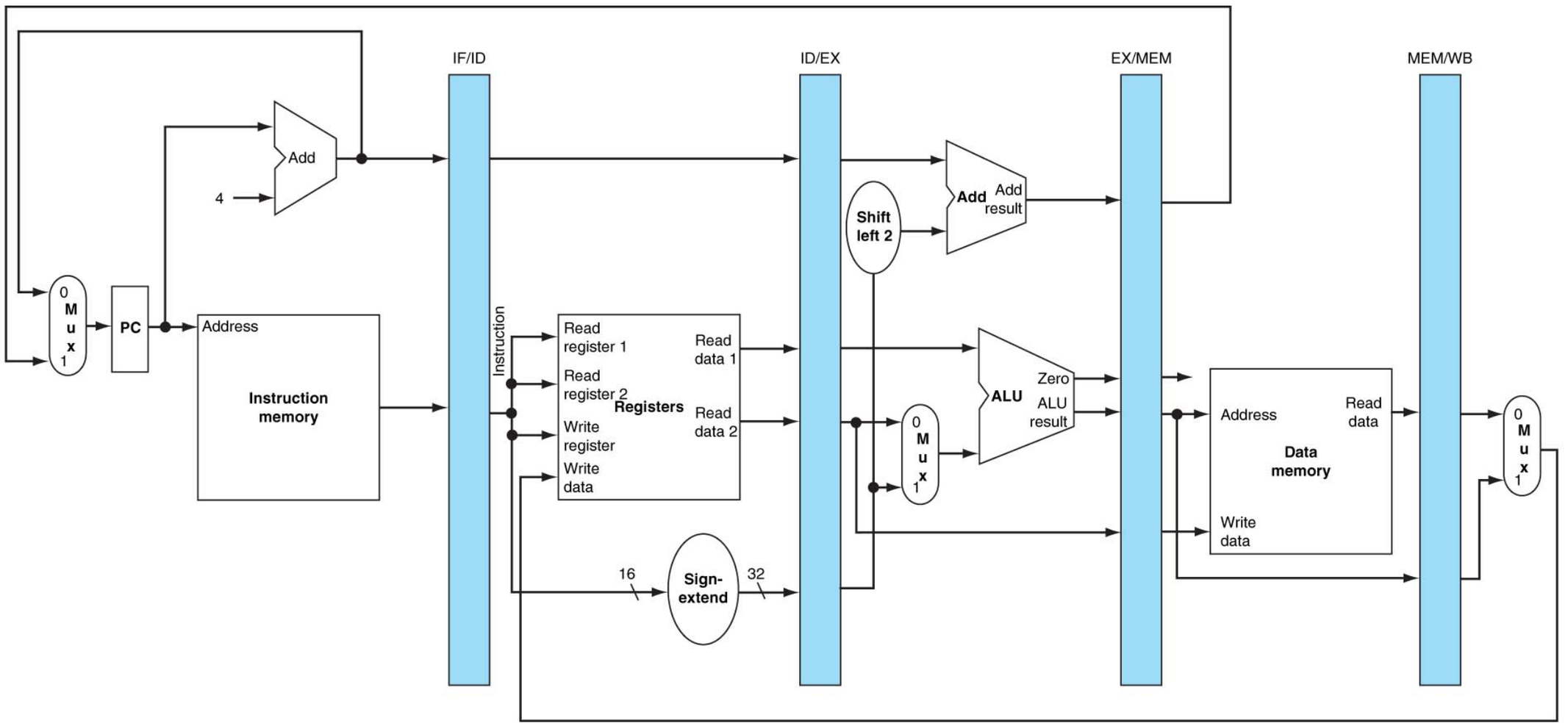
Instruction Fetch	Decode / Reg. Fetch	Execute	Memory	Write-back



Pipeline In Execution

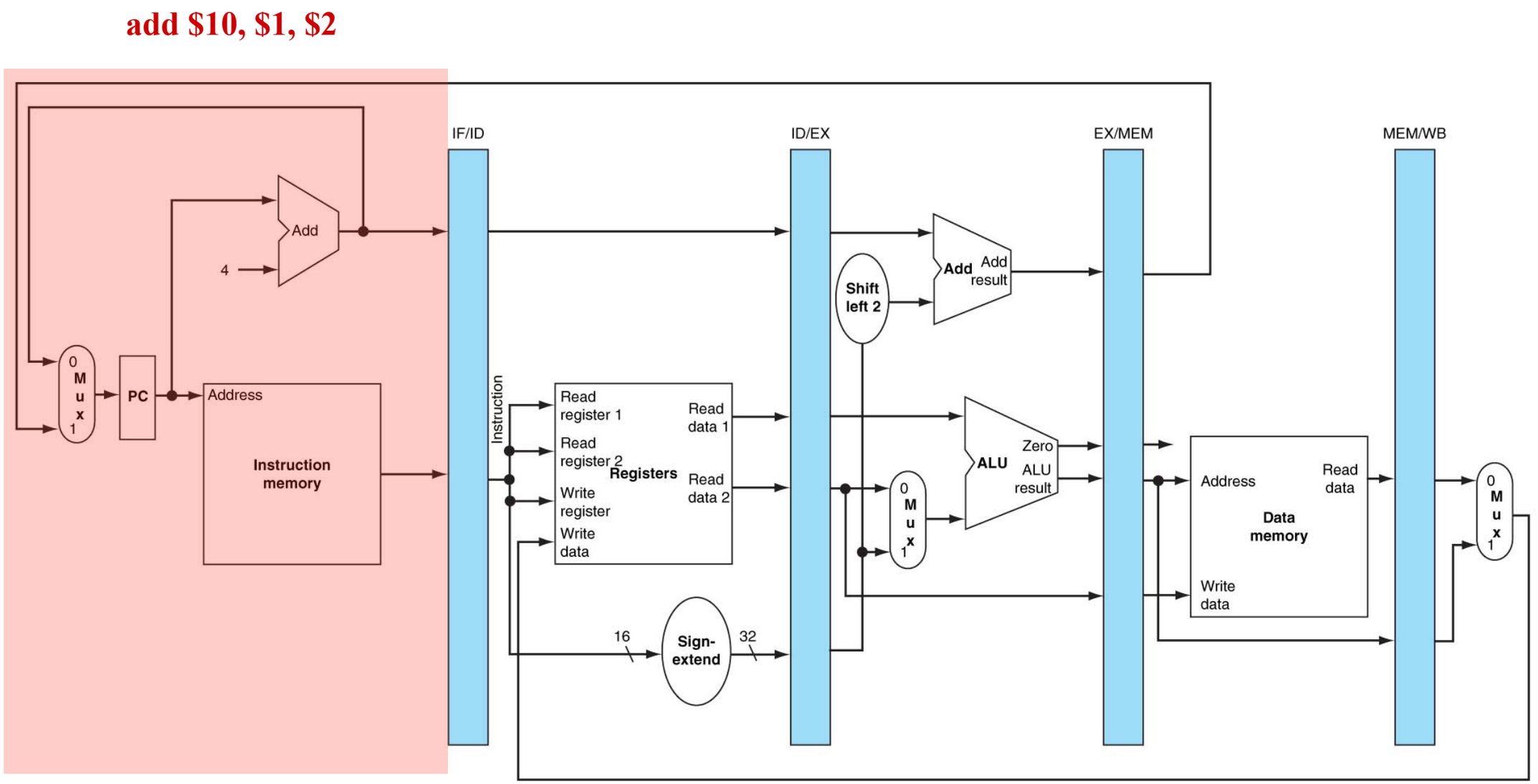
Instruction Fetch	Decode / Reg. Fetch	Execute	Memory	Write-back

add $10, $1, $2



Pipeline In Execution

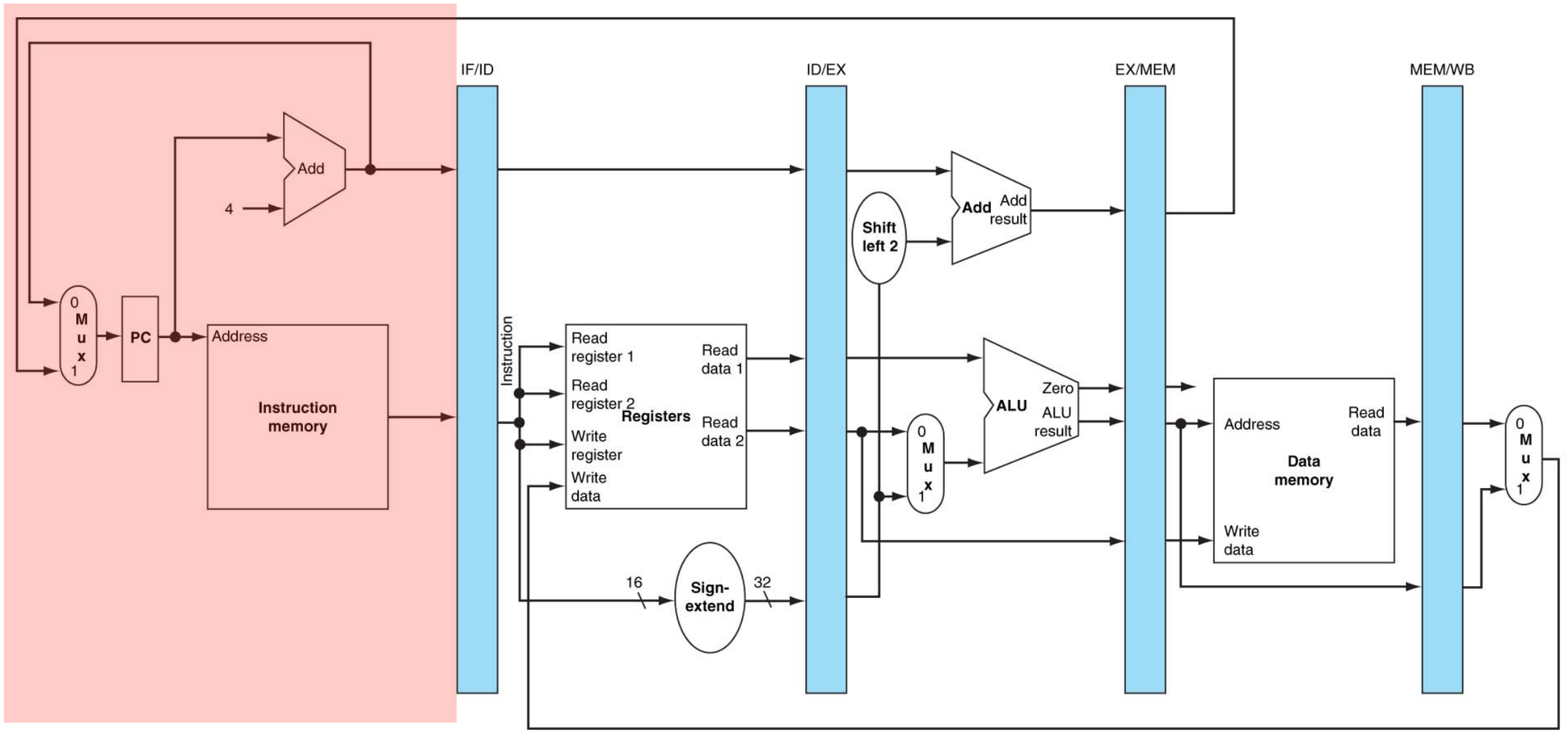
Instruction Fetch | Decode / Reg. Fetch | Execute | Memory | Write-back



Pipeline In Execution

Instruction Fetch	Decode / Reg. Fetch	Execute	Memory	Write-back

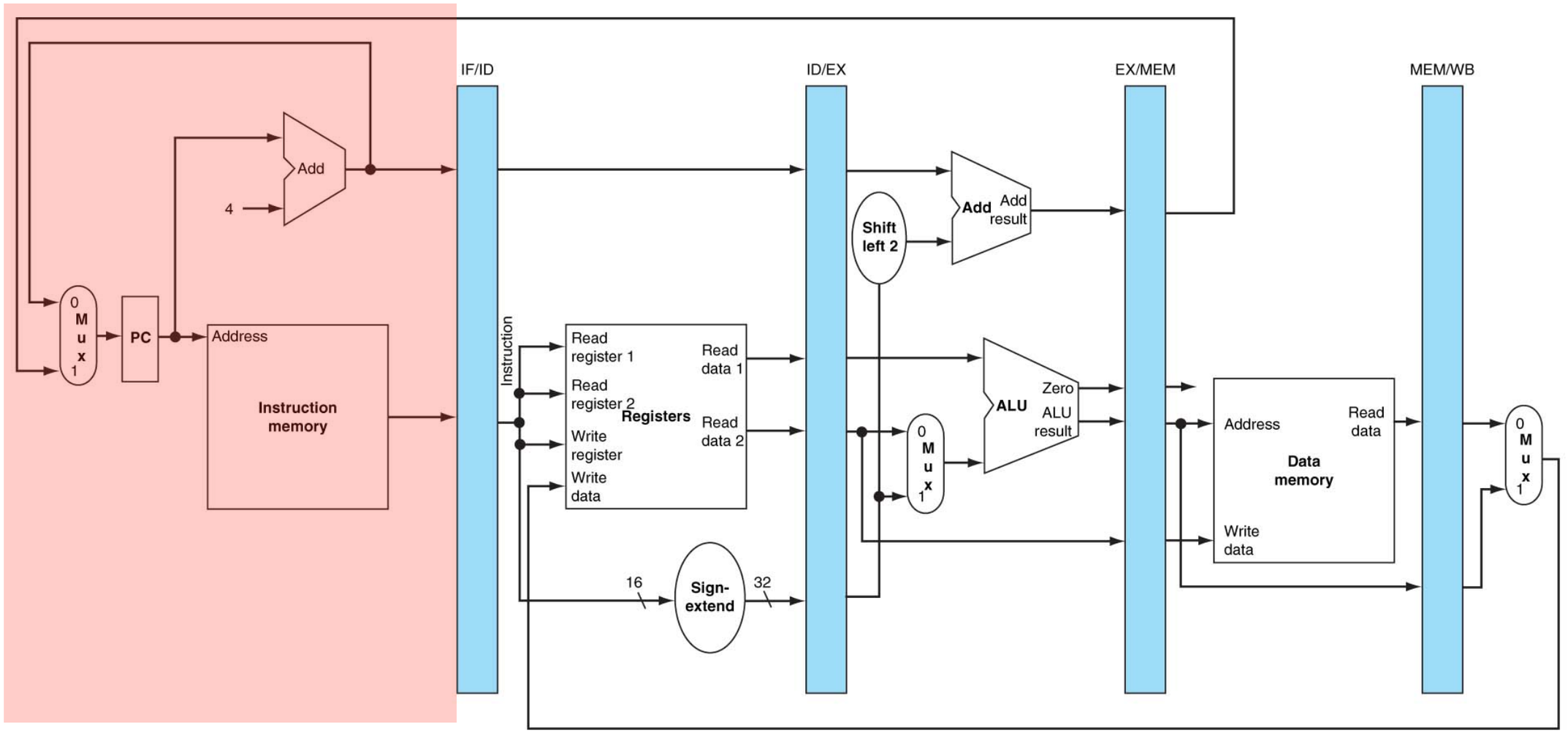
add $10, $1, $2



Pipeline In Execution

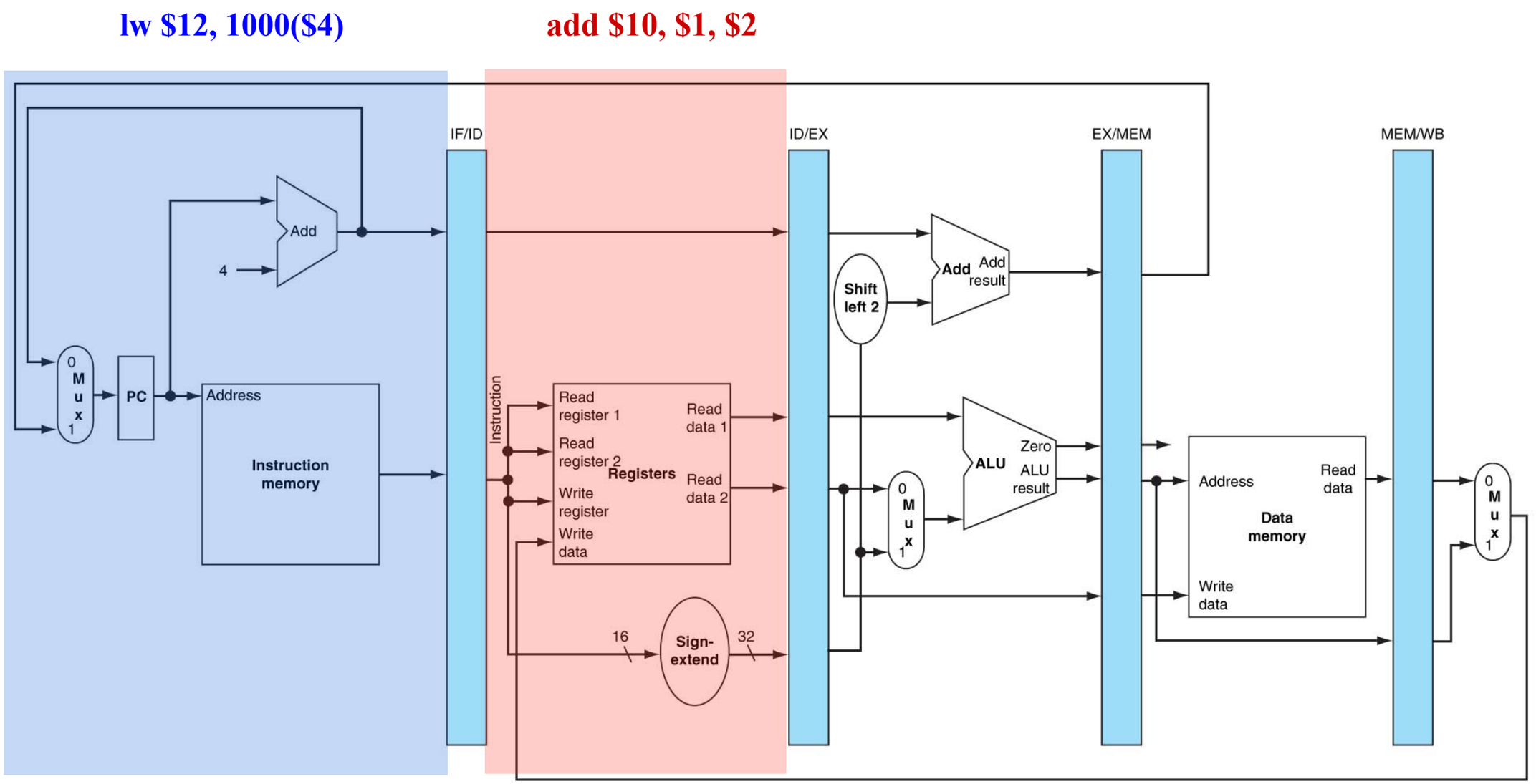
Instruction Fetch	Decode / Reg. Fetch	Execute	Memory	Write-back

lw $12, 1000($4) **add $10, $1, $2**



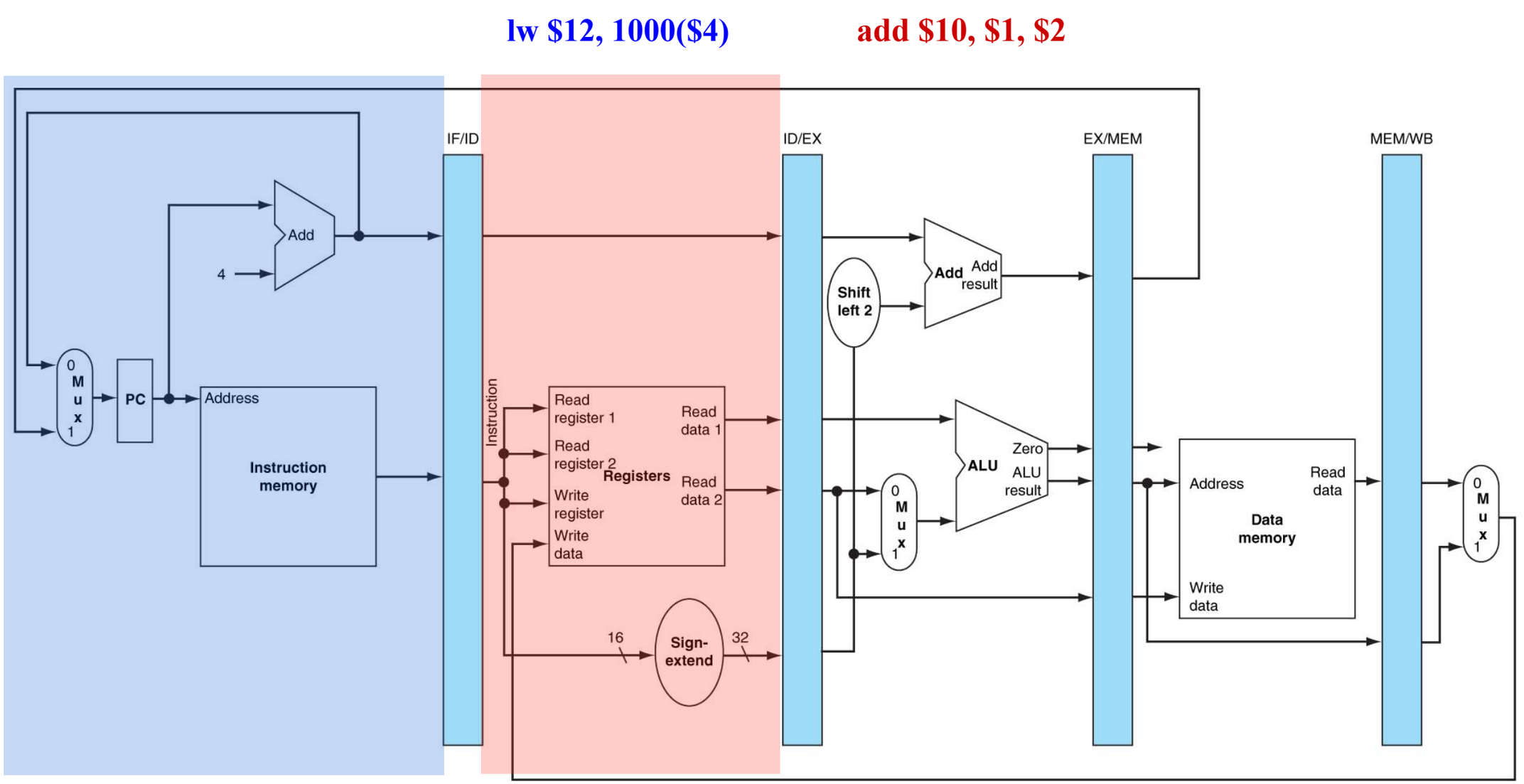
Pipeline In Execution

Instruction Fetch | Decode / Reg. Fetch | Execute | Memory | Write-back



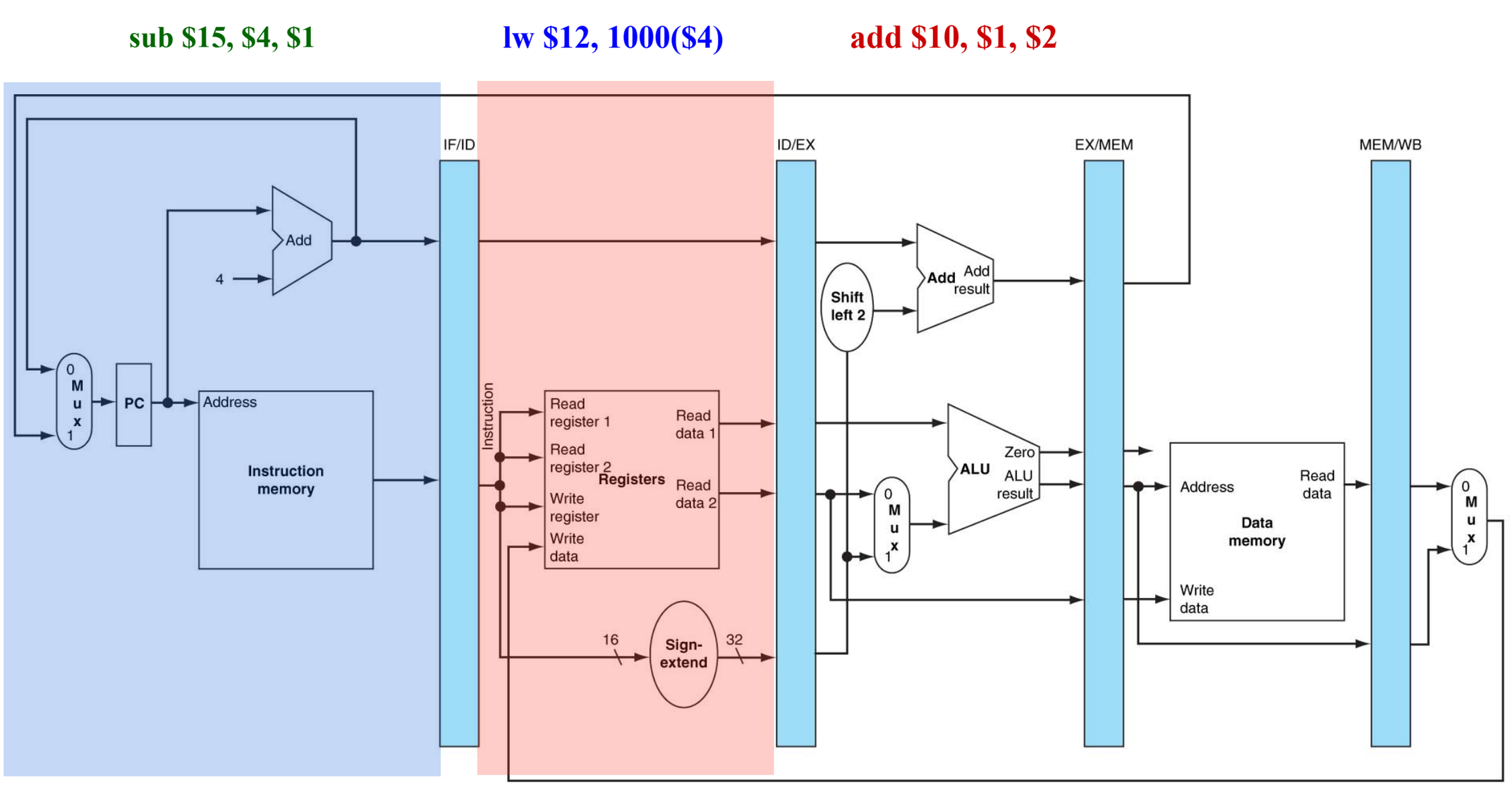
Pipeline In Execution

Instruction Fetch | Decode / Reg. Fetch | Execute | Memory | Write-back



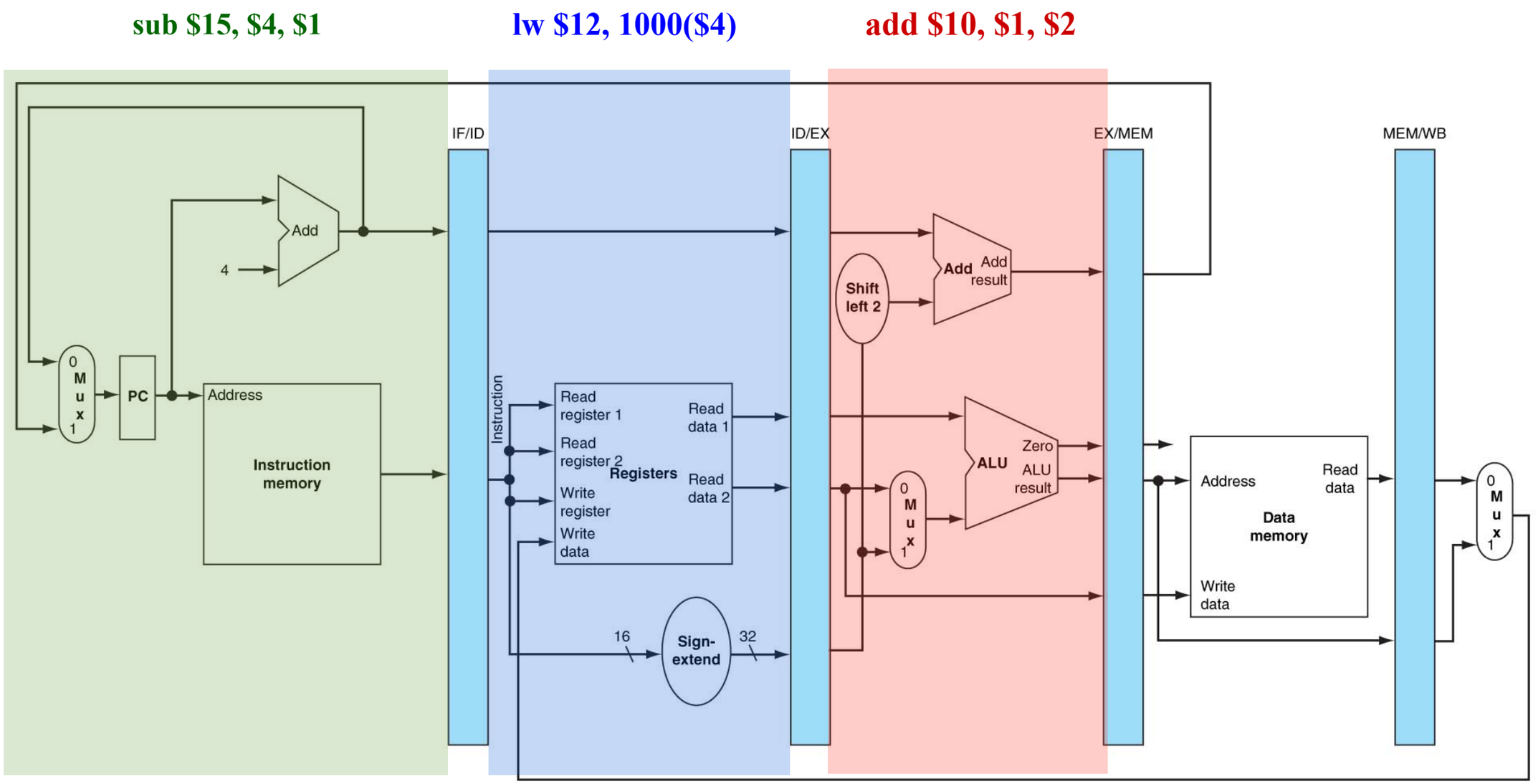
Pipeline In Execution

Instruction Fetch | Decode / Reg. Fetch | Execute | Memory | Write-back



Pipeline In Execution

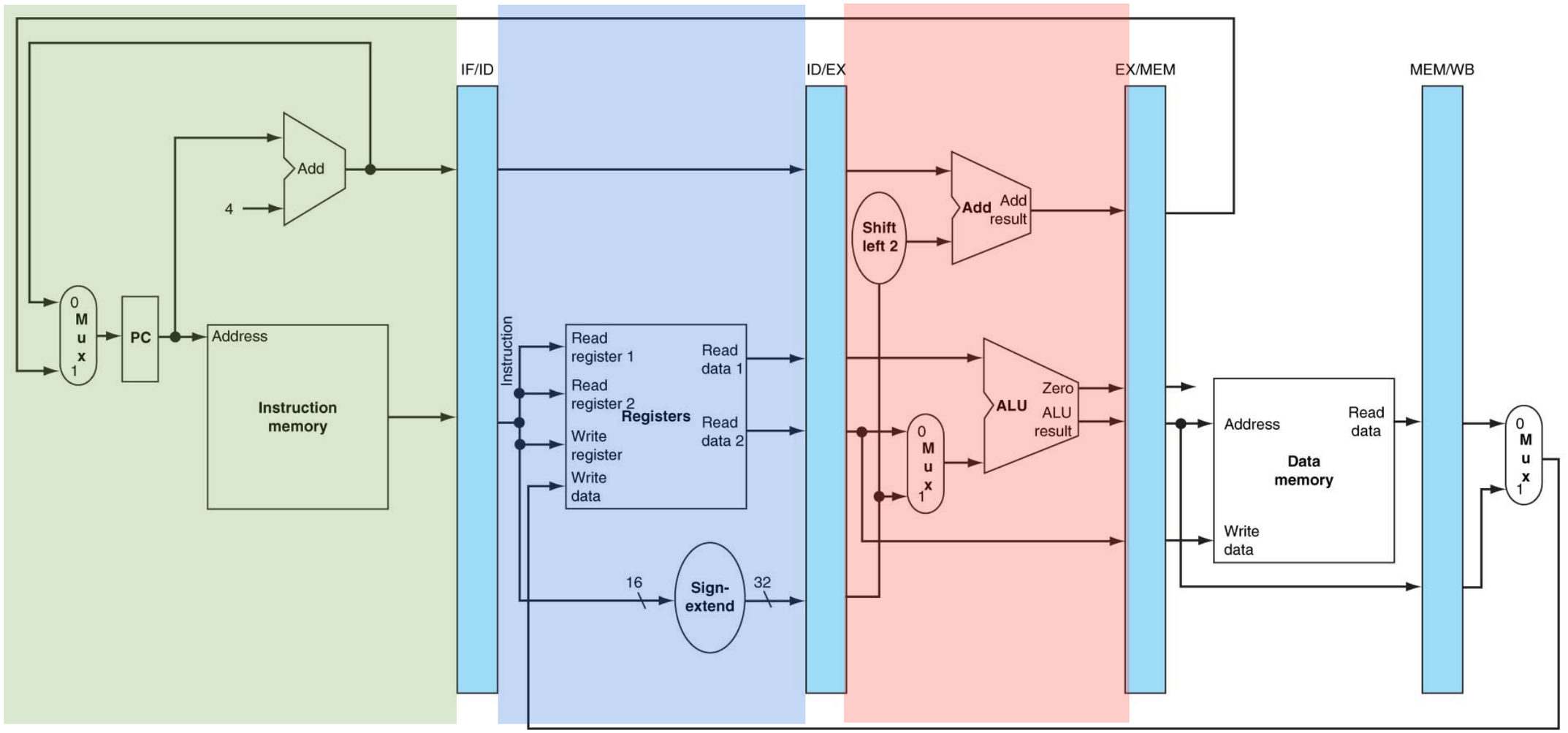
Instruction Fetch | Decode / Reg. Fetch | Execute | Memory | Write-back



Pipeline In Execution

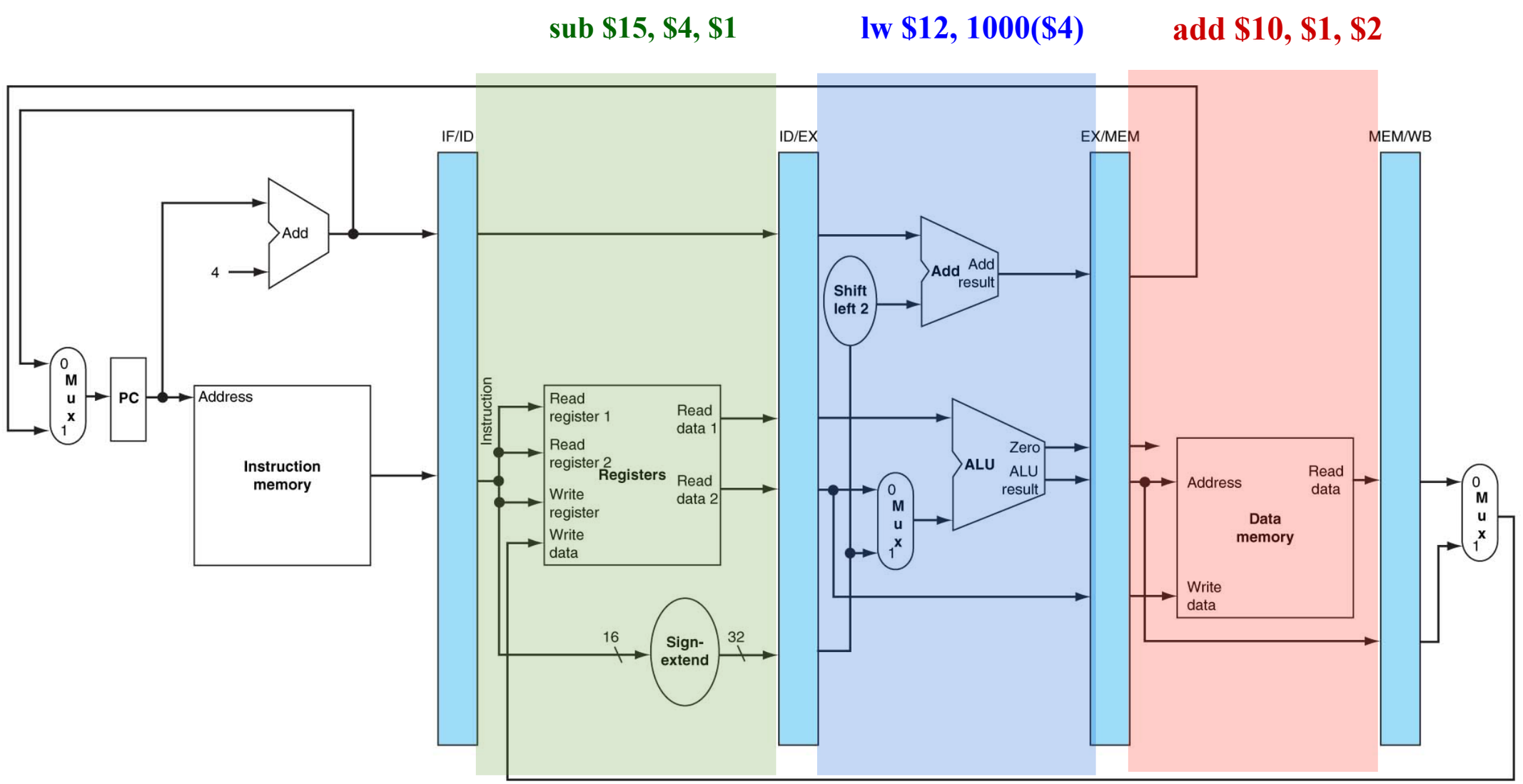
Instruction Fetch | Decode / Reg. Fetch | Execute | Memory | Write-back

sub $15, $4, $1 | **lw $12, 1000($4)** | **add $10, $1, $2**



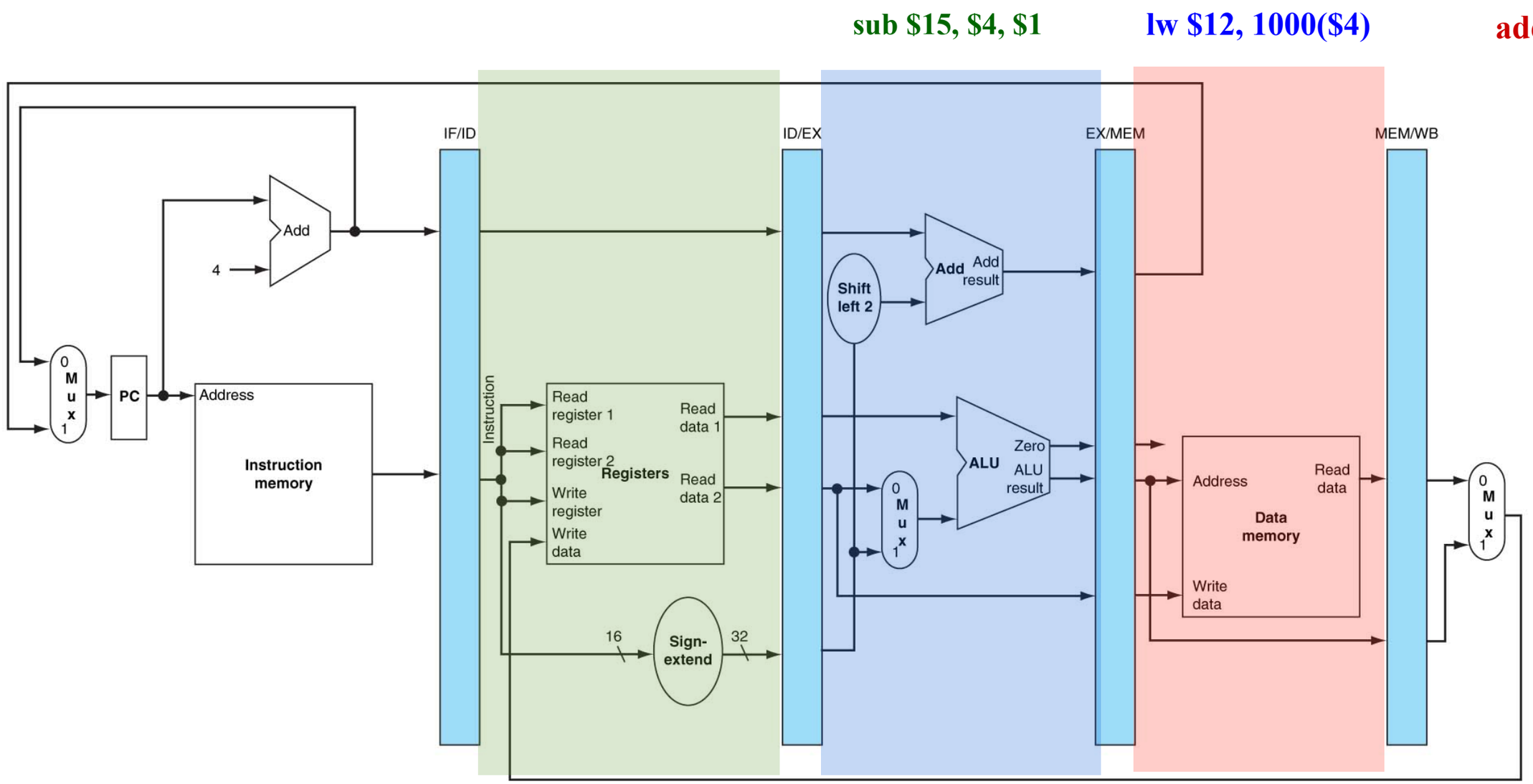
Pipeline In Execution

Instruction Fetch | Decode / Reg. Fetch | Execute | Memory | Write-back



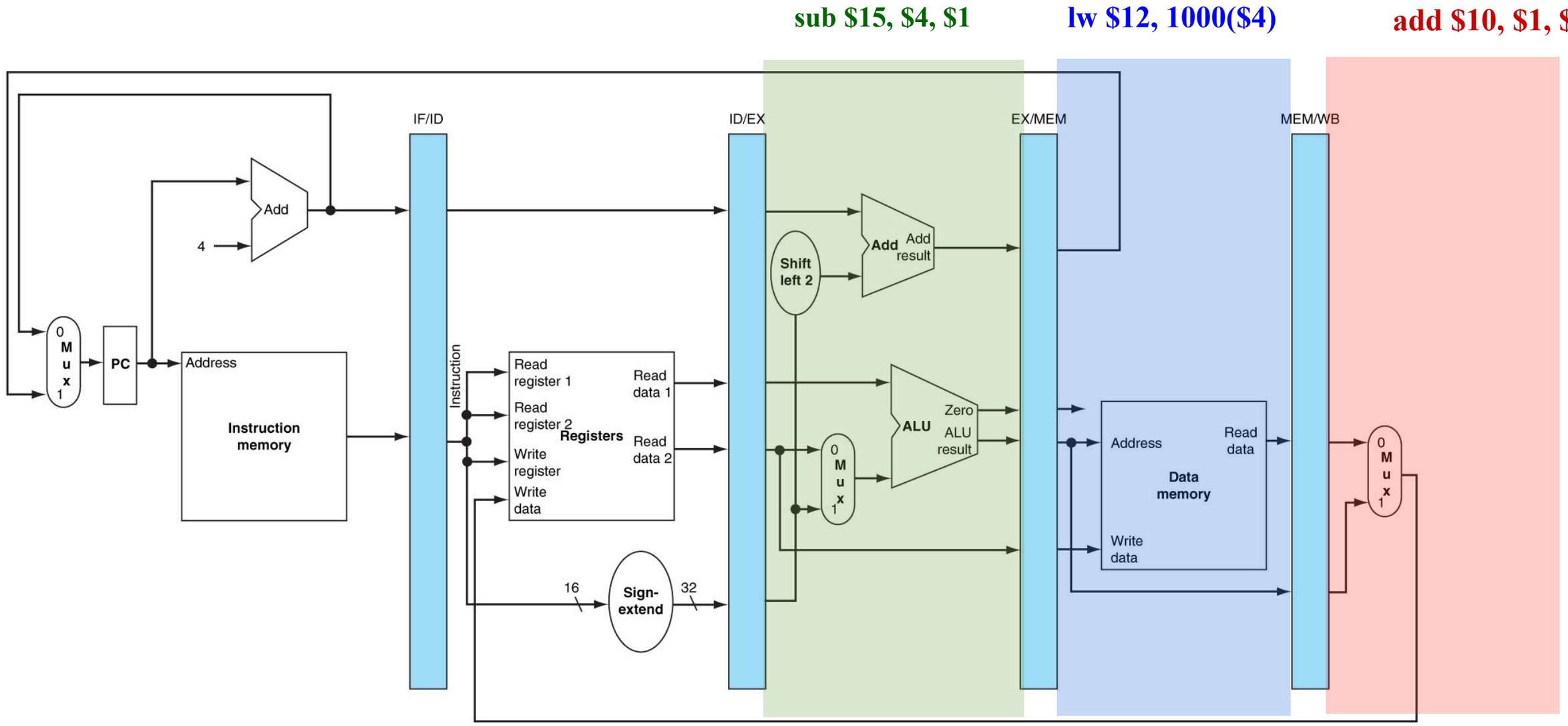
Pipeline In Execution

Instruction Fetch | Decode / Reg. Fetch | Execute | Memory | Write-back



Pipeline In Execution

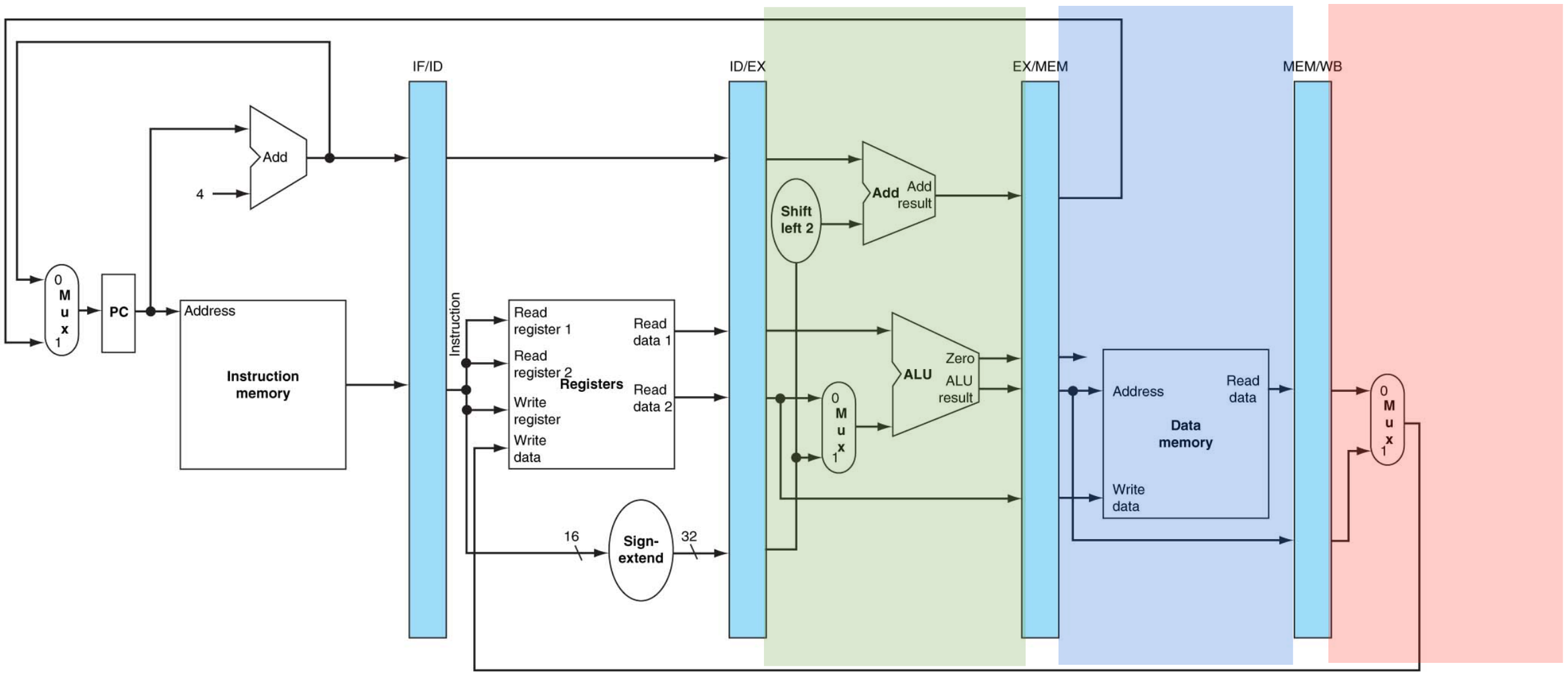
Instruction Fetch | Decode / Reg. Fetch | Execute | Memory | Write-back



Pipeline In Execution

Instruction Fetch	Decode / Reg. Fetch	Execute	Memory	Write-back

sub $15, $4, $1 **lw $12, 1000($4)**

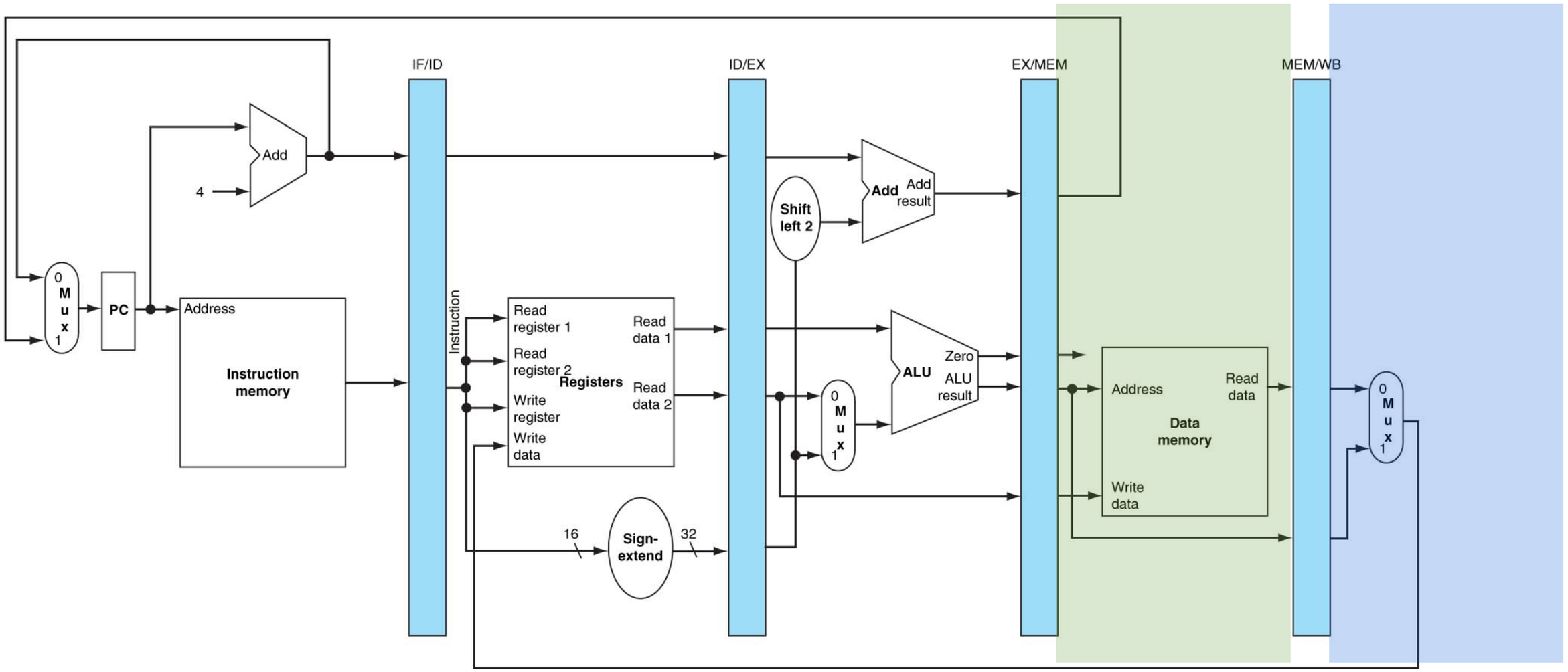


Pipeline In Execution

Instruction Fetch	Decode / Reg. Fetch	Execute	Memory	Write-back

sub $15, $4, $1

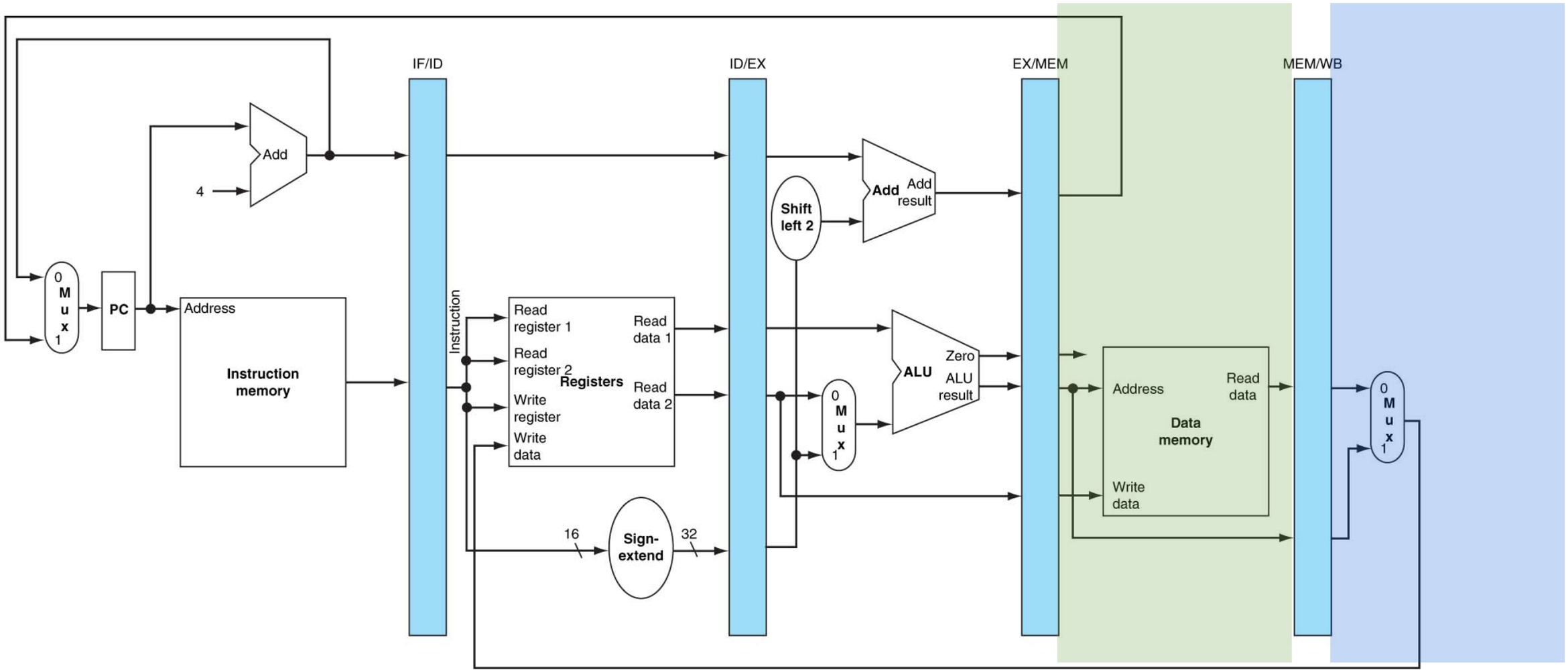
lw $12, 1000($4)



Pipeline In Execution

Instruction Fetch	Decode / Reg. Fetch	Execute	Memory	Write-back

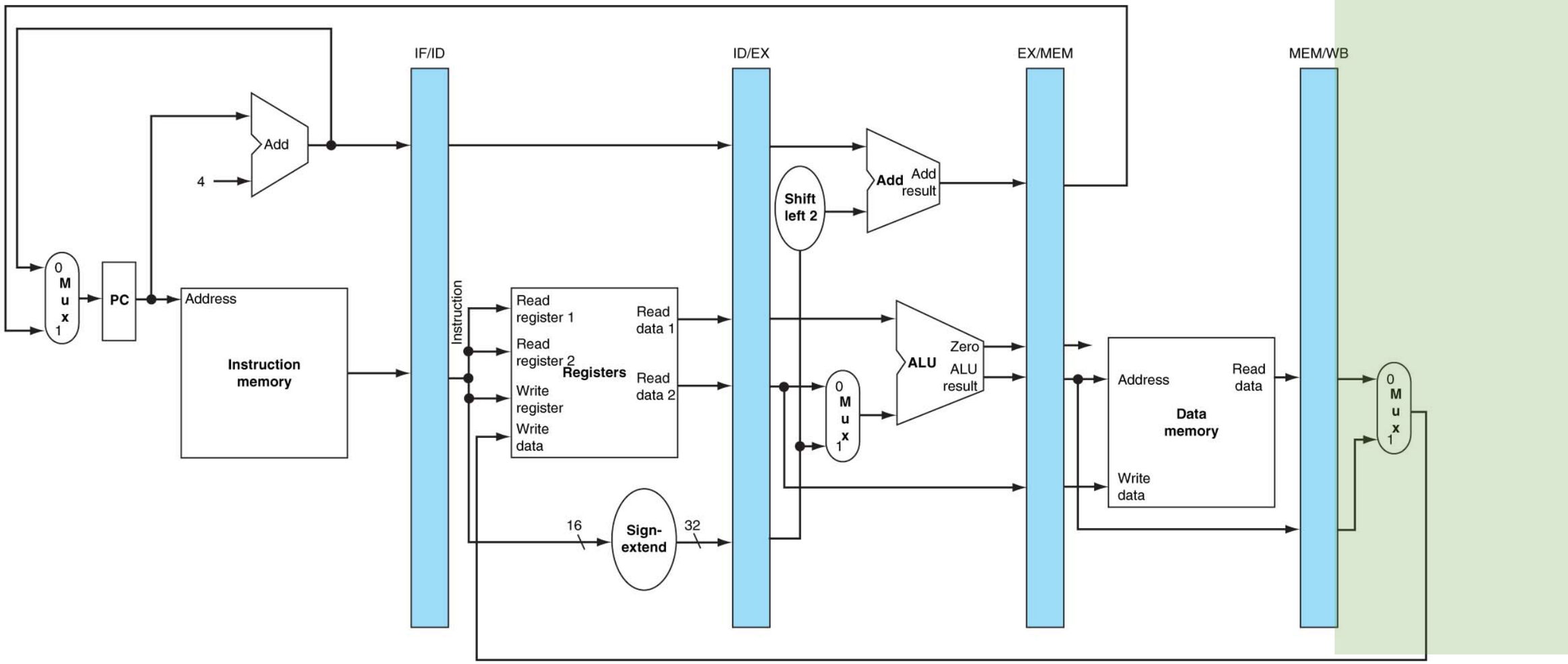
sub $15, $4, $1



Pipeline In Execution

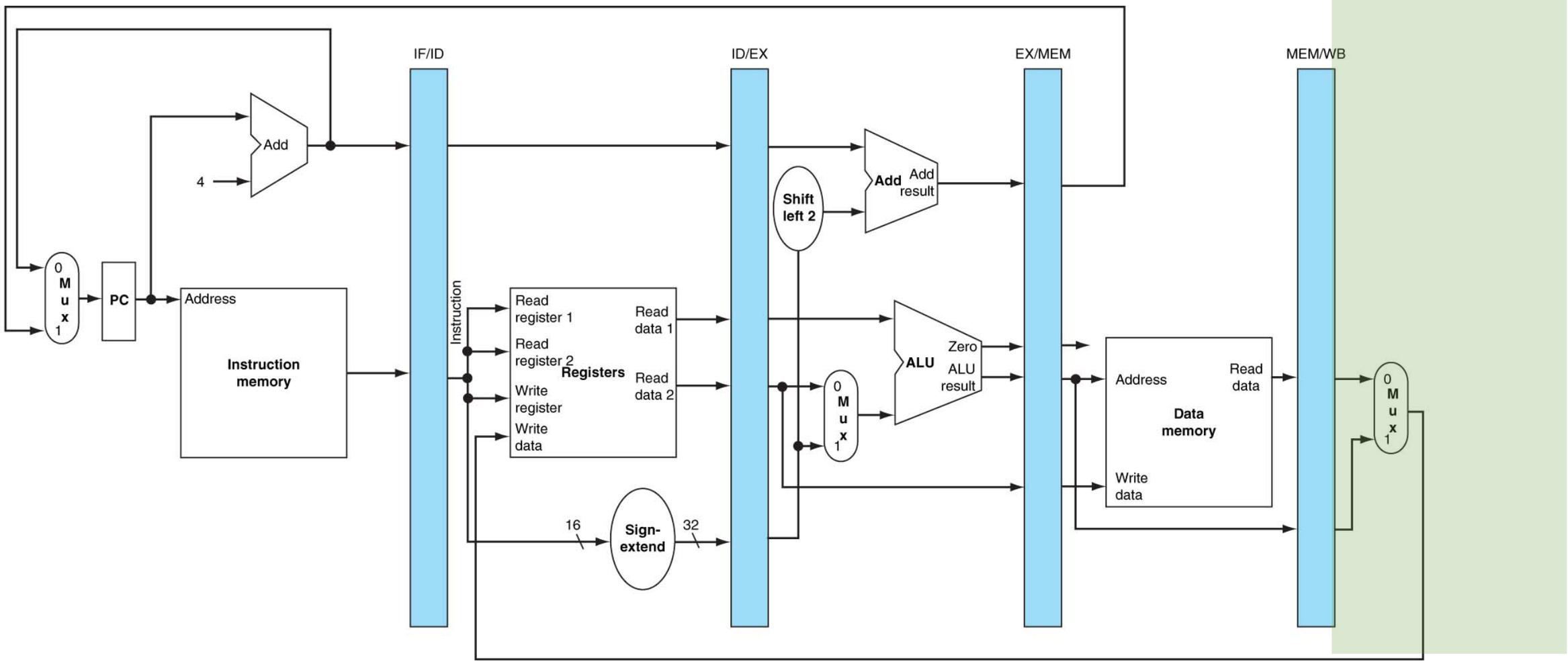
Instruction Fetch	Decode / Reg. Fetch	Execute	Memory	Write-back

sub $15, $4, $1



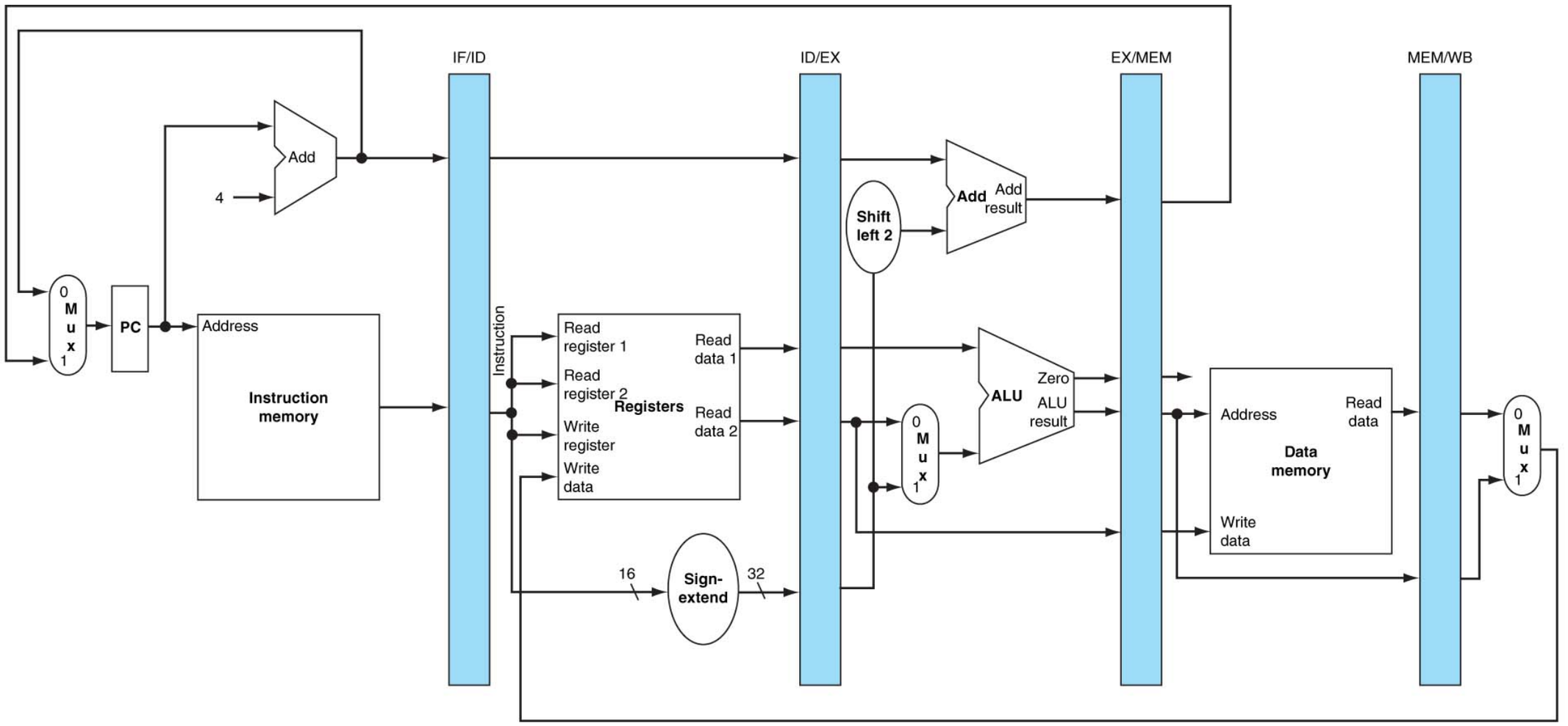
Pipeline In Execution

Instruction Fetch	Decode / Reg. Fetch	Execute	Memory	Write-back

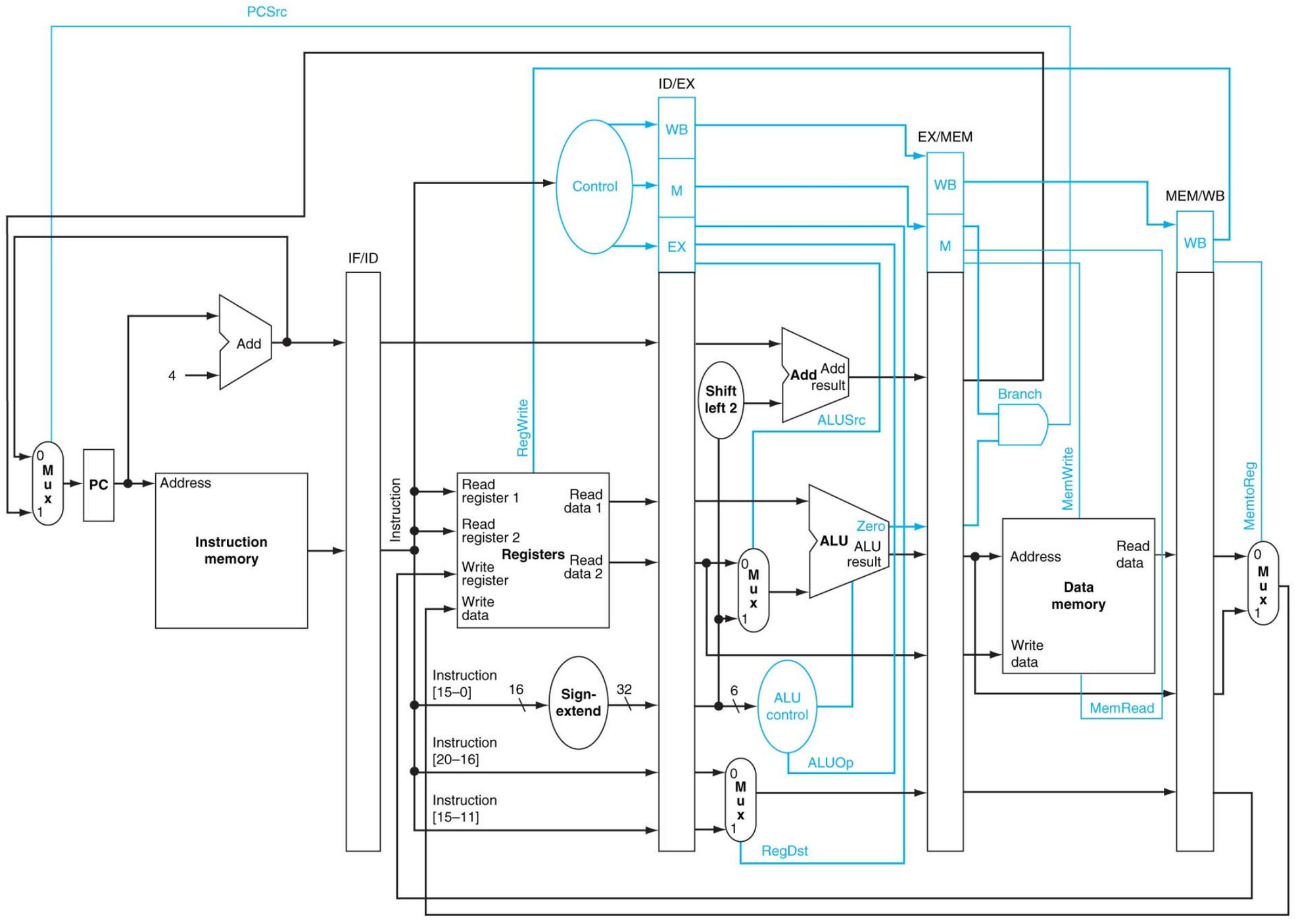


Pipeline In Execution

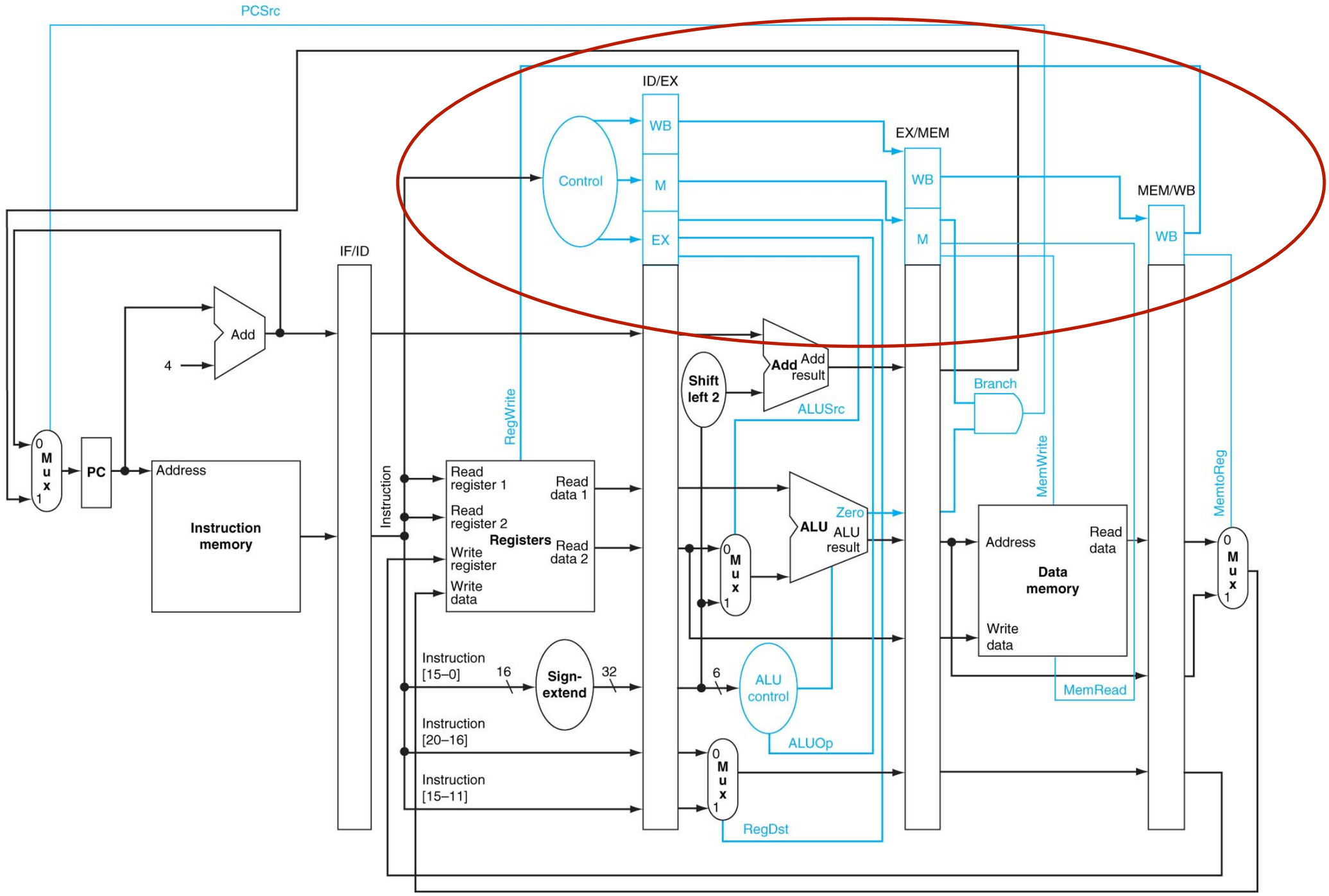
Instruction Fetch	Decode / Reg. Fetch	Execute	Memory	Write-back



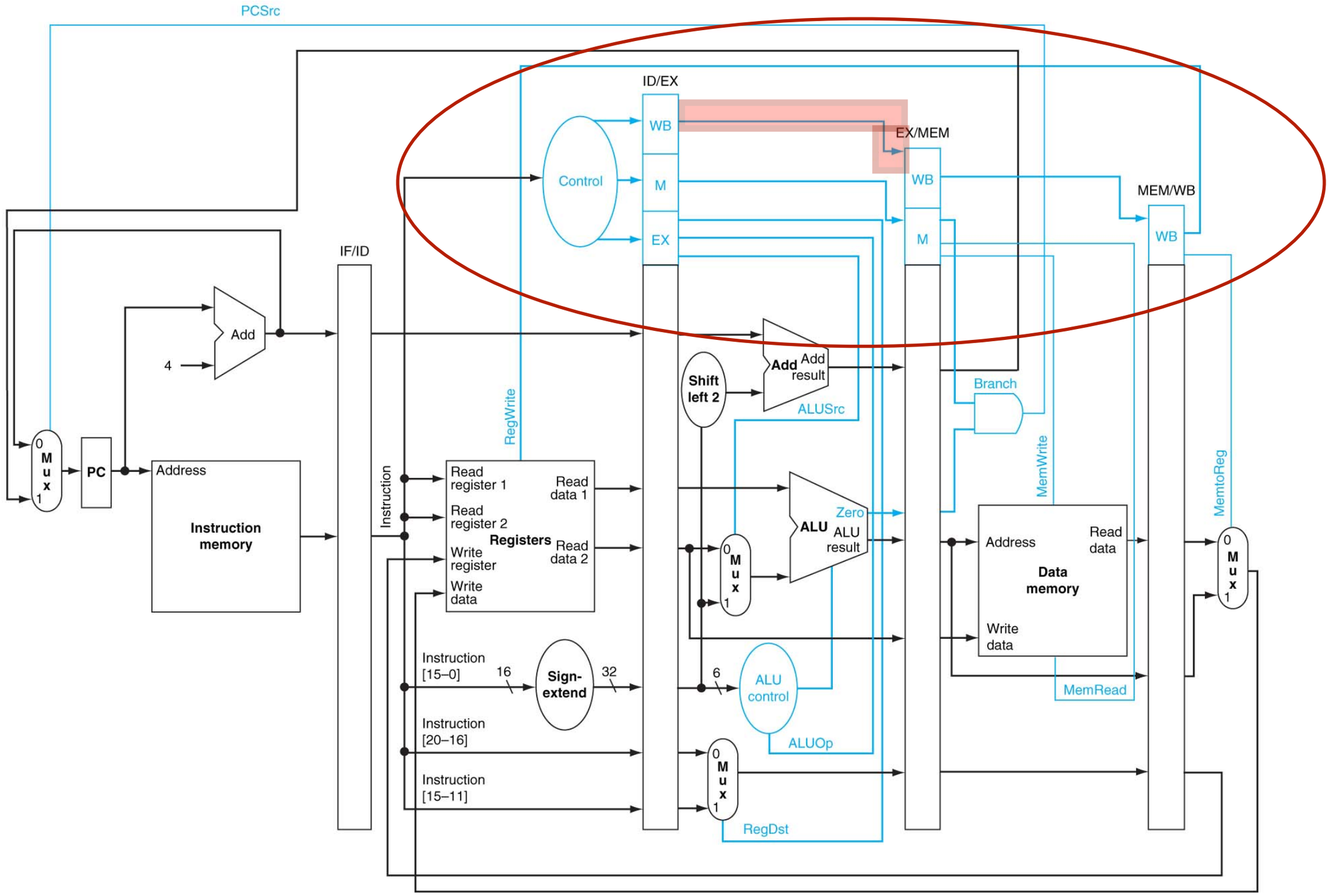
Pipeline with Control Logic



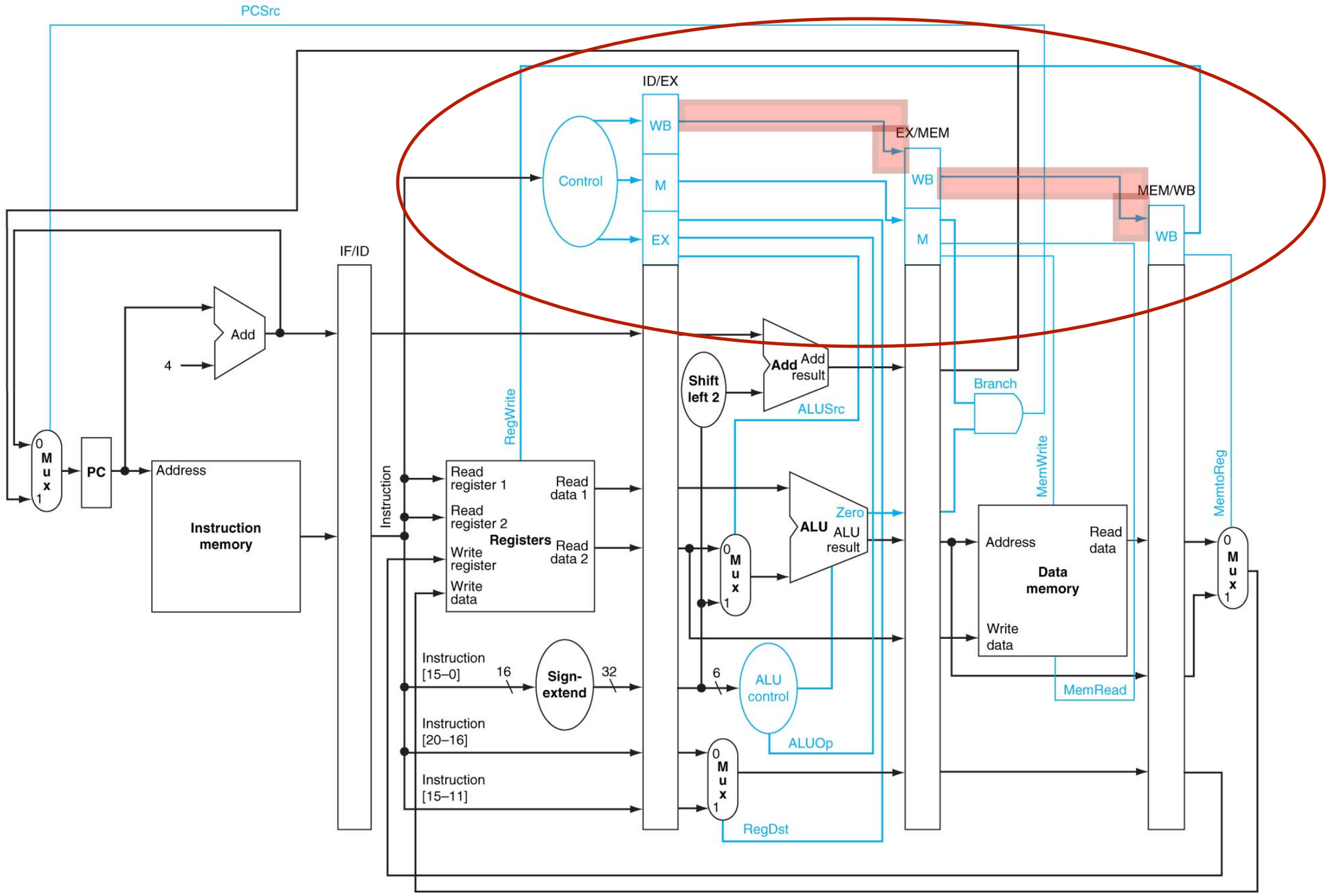
Pipeline with Control Logic



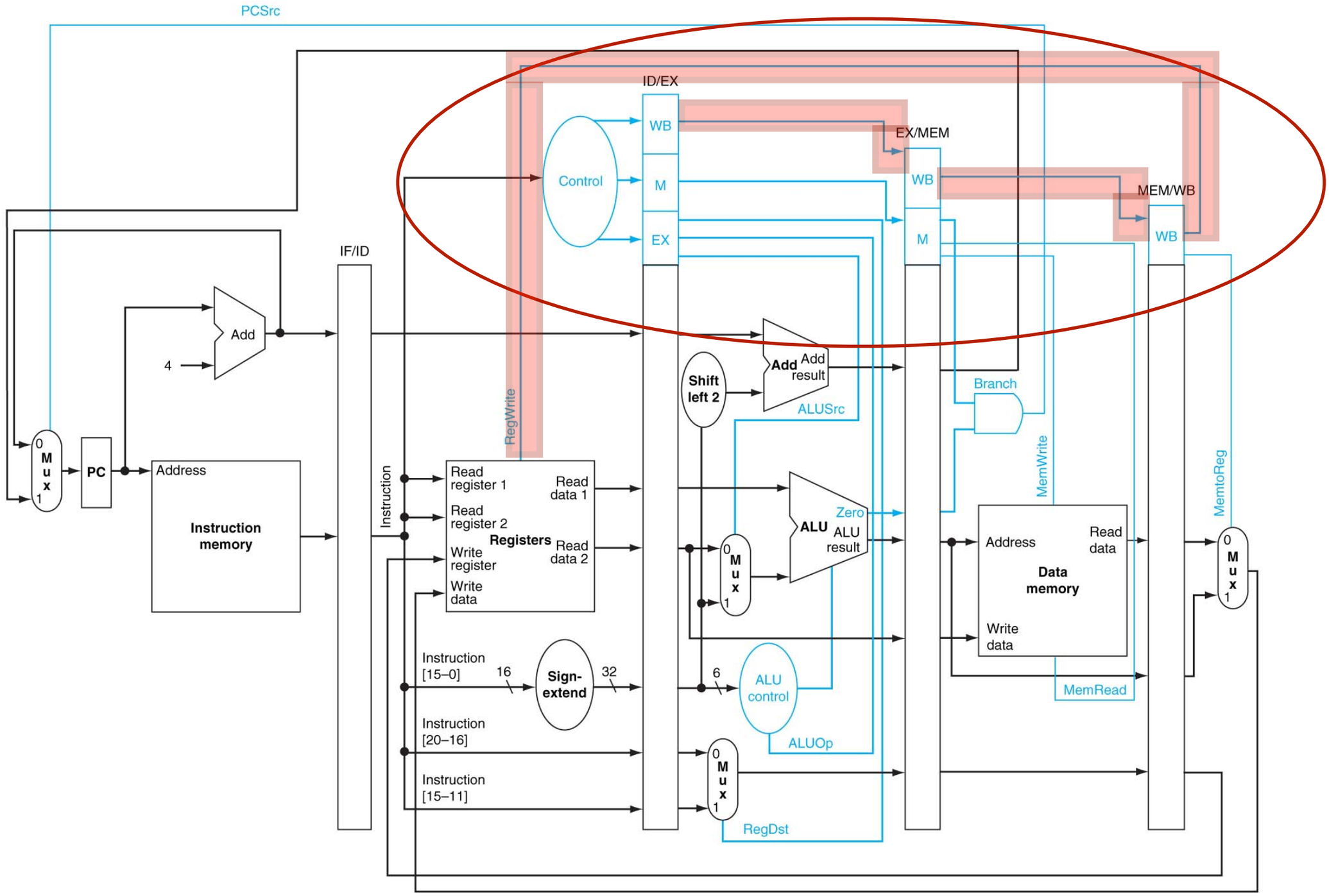
Pipeline with Control Logic



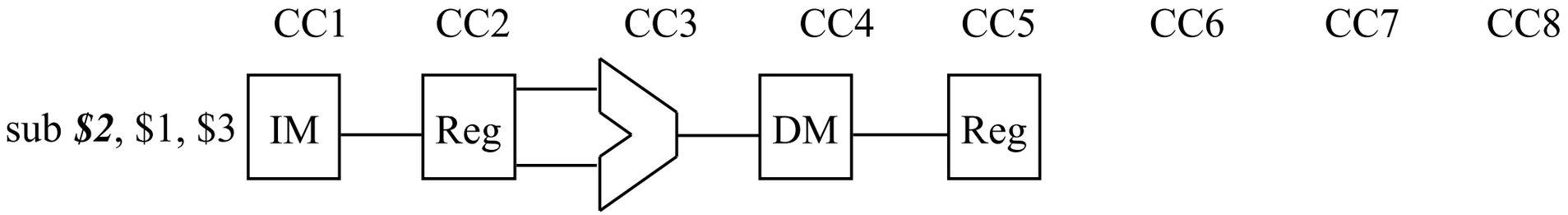
Pipeline with Control Logic



Pipeline with Control Logic



Hardware Stalls



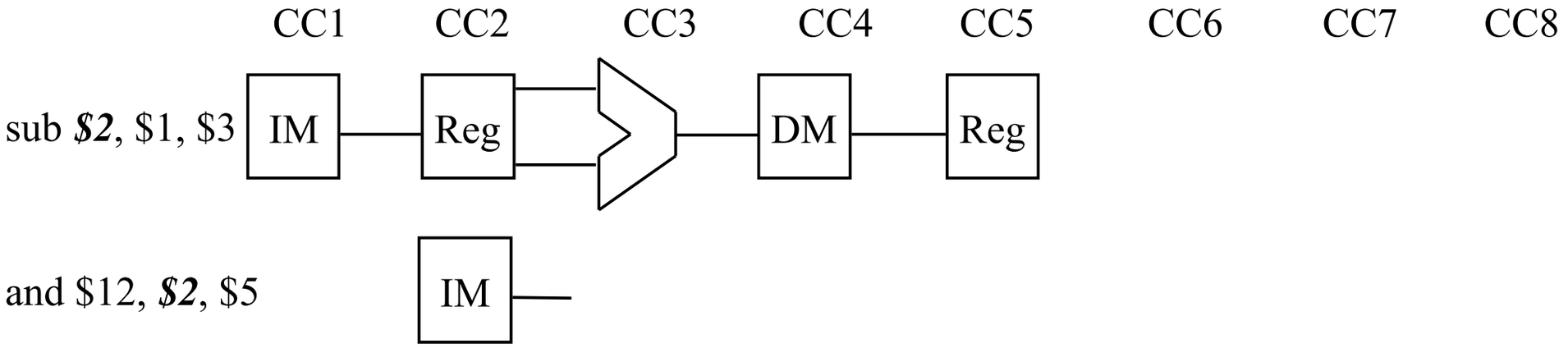
and $12, ***$2***, $5

or $13, $6, ***$2***

add $14, ***$2, $2***

sw $15, 100(***$2***)

Hardware Stalls

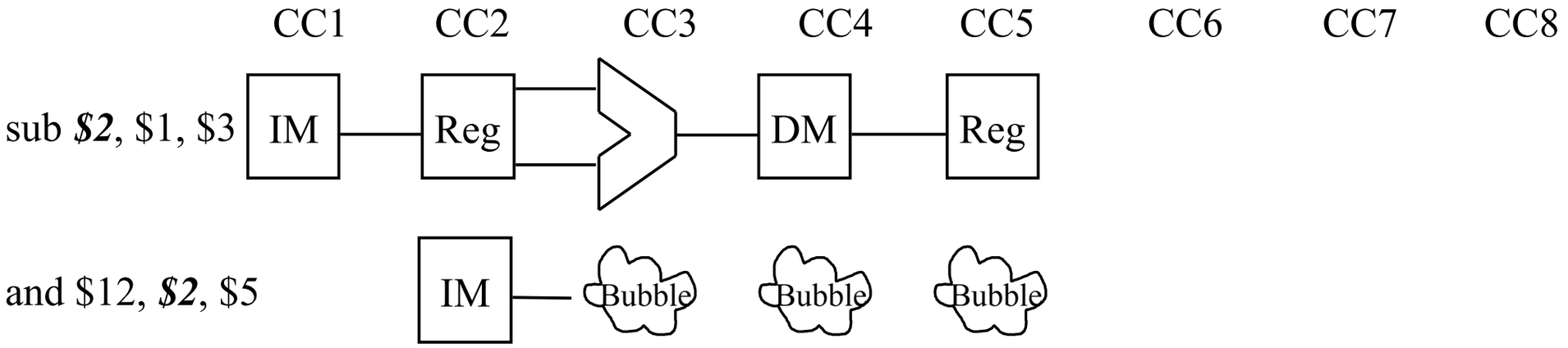


or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)

Hardware Stalls

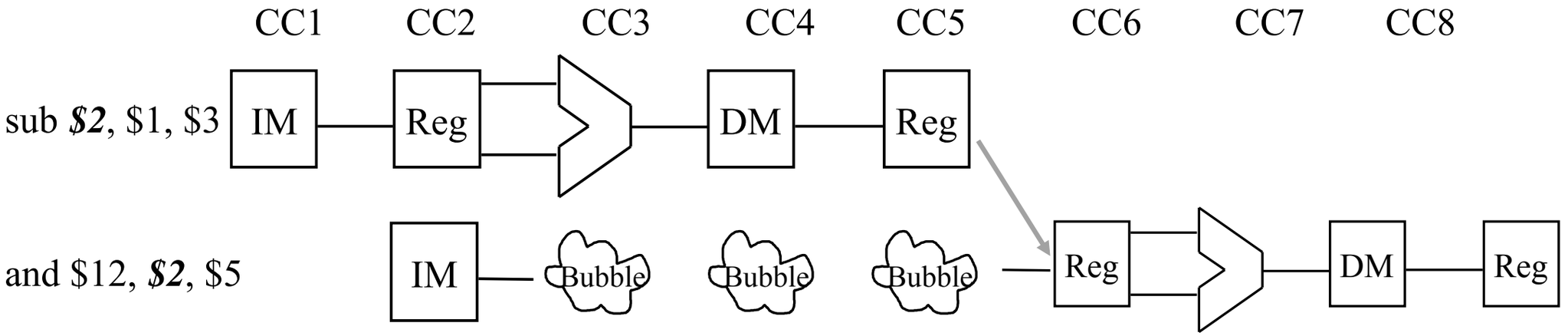


or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)

Hardware Stalls

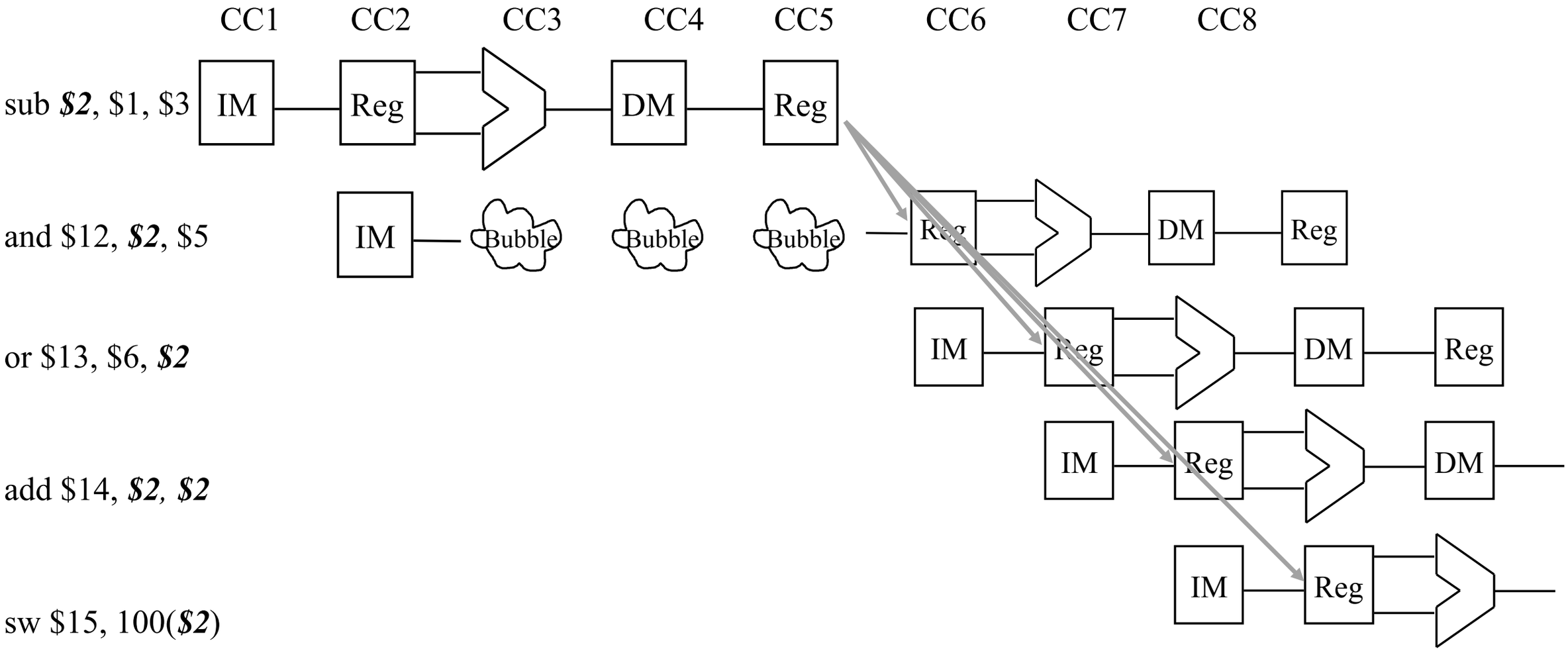


or $13, $6, ***$2***

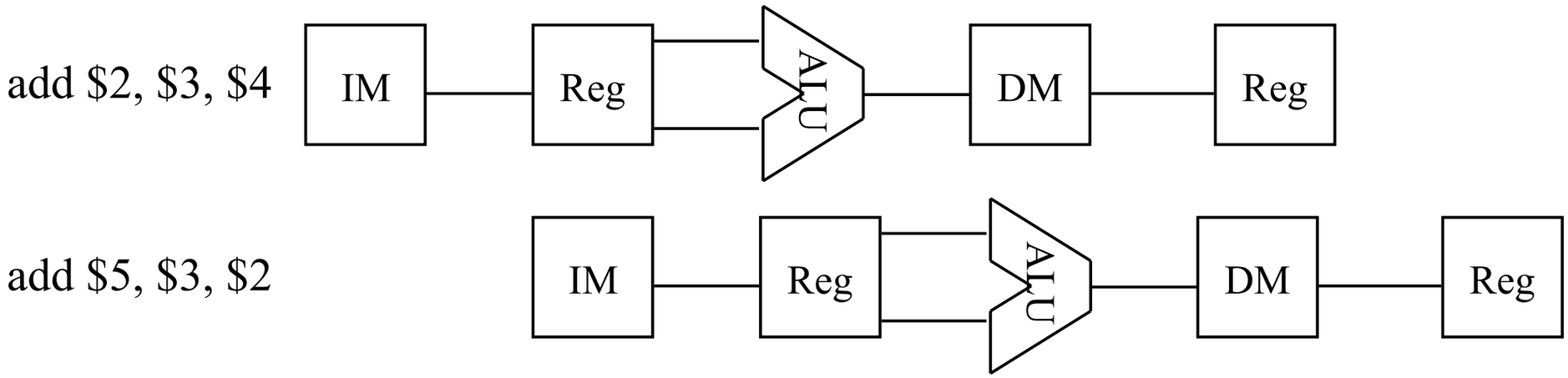
add $14, ***$2, $2***

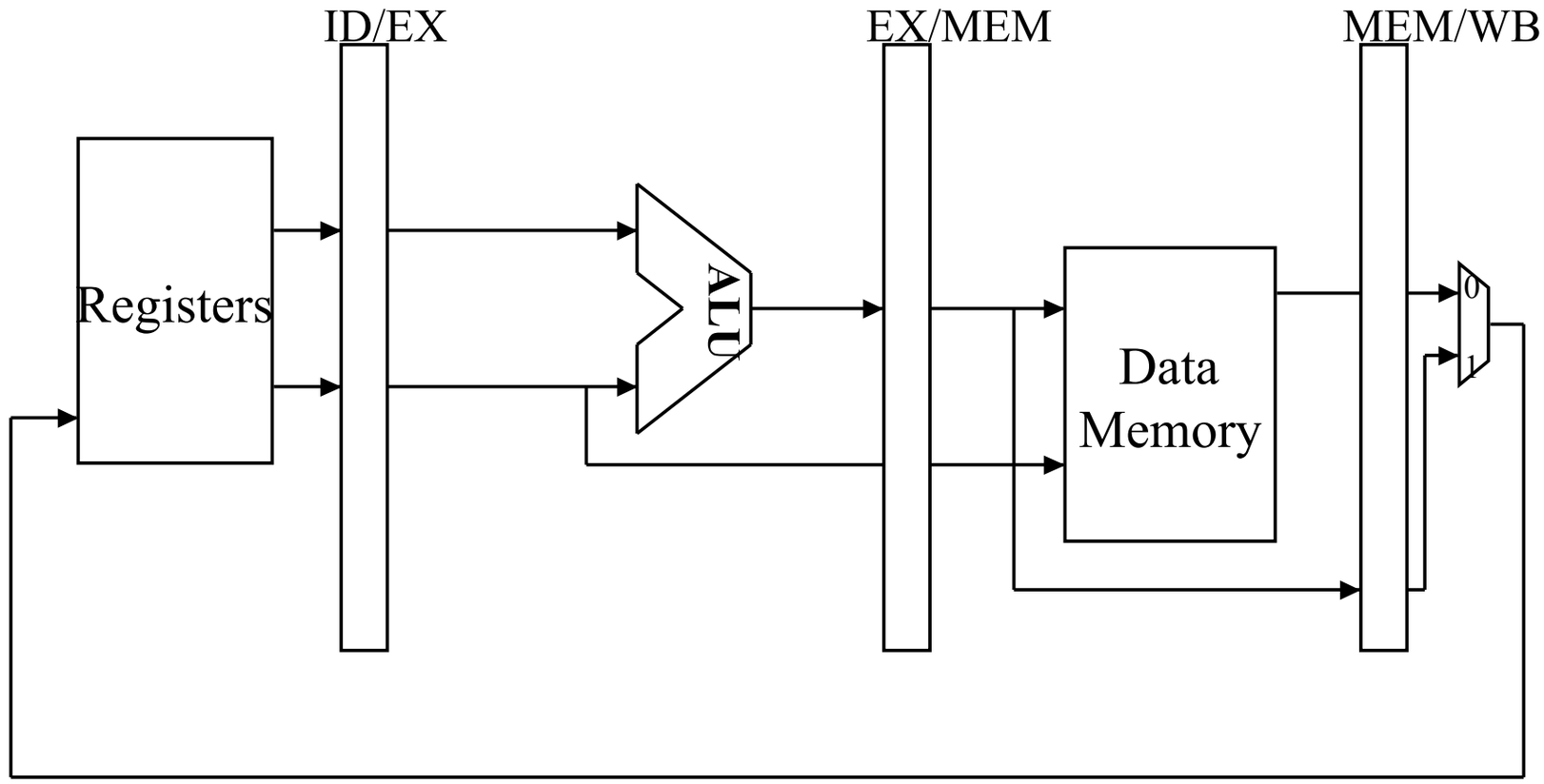
sw $15, 100(***$2***)

Hardware Stalls

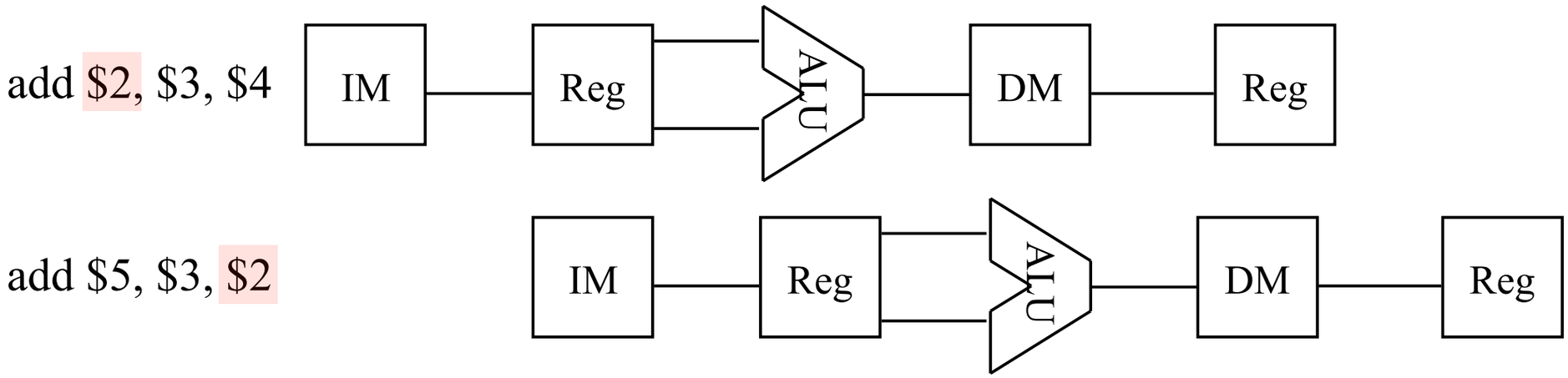


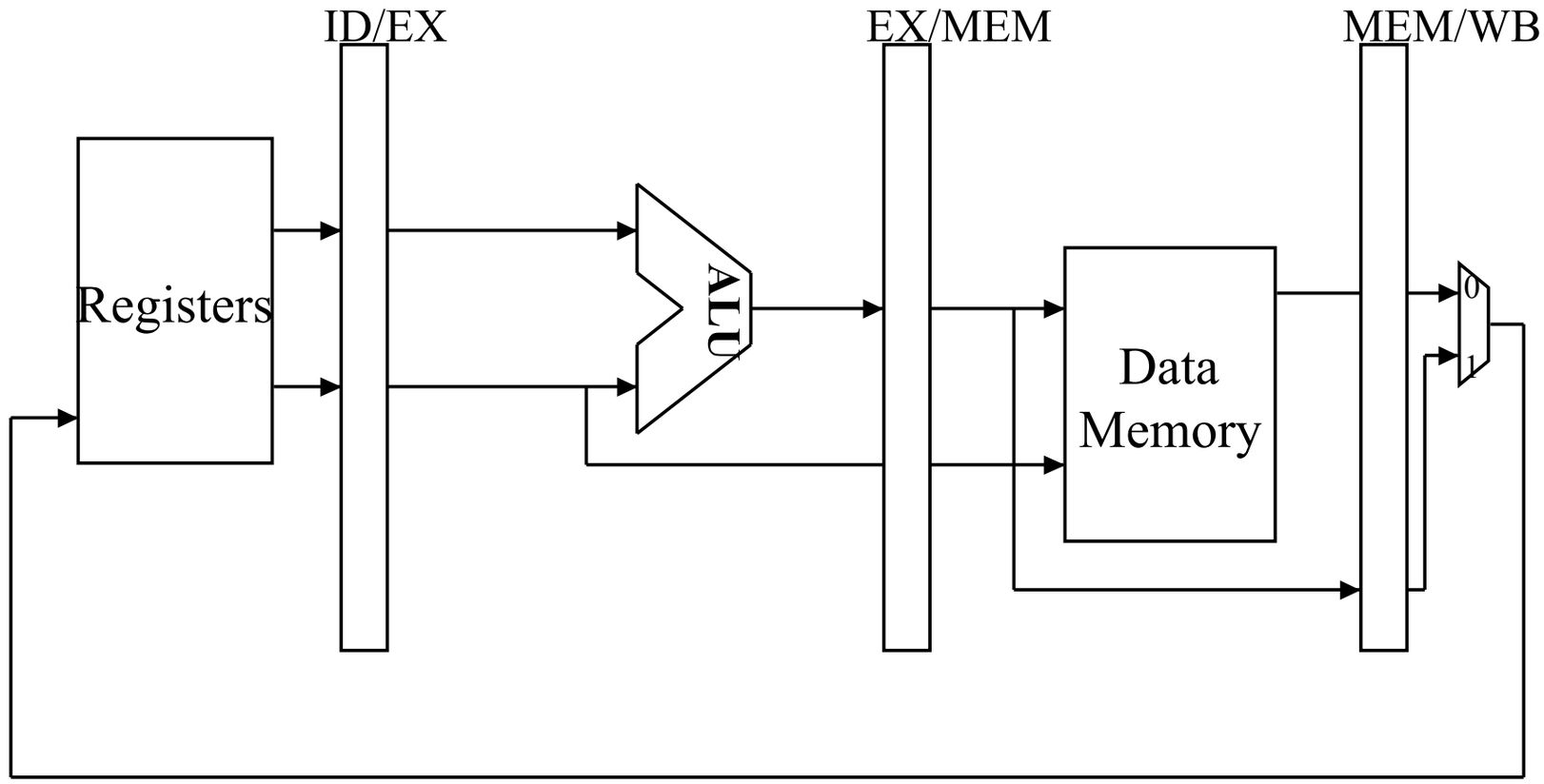
Forwarding


add $2, $3, $4
IM
Reg
ALU
DM
Reg
add $5, $3, $2
IM
Reg
ALU
DM
Reg

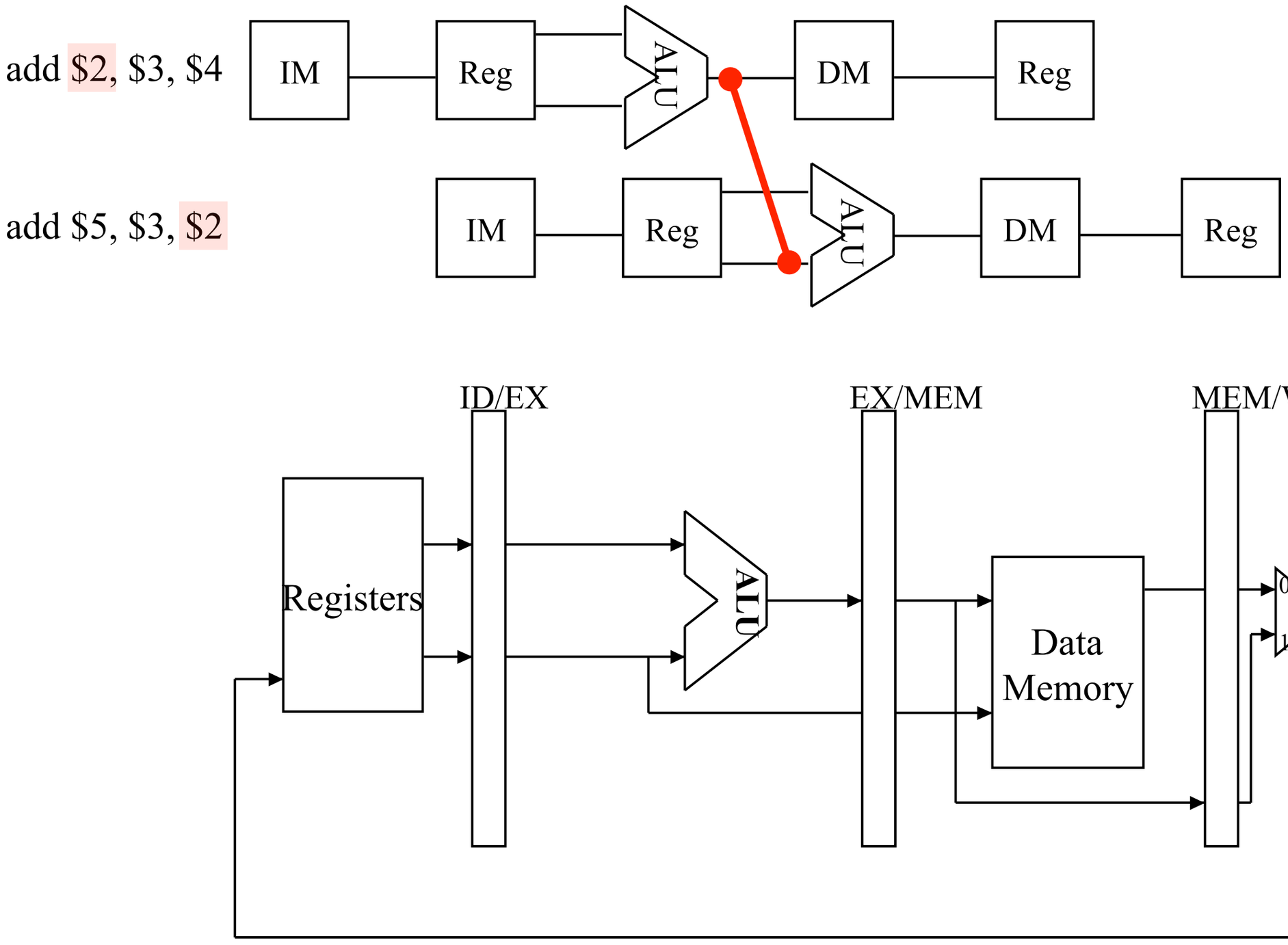

ID/EX
EX/MEM
MEM/WB
Registers
ALU
Data
Memory
0
1

Forwarding

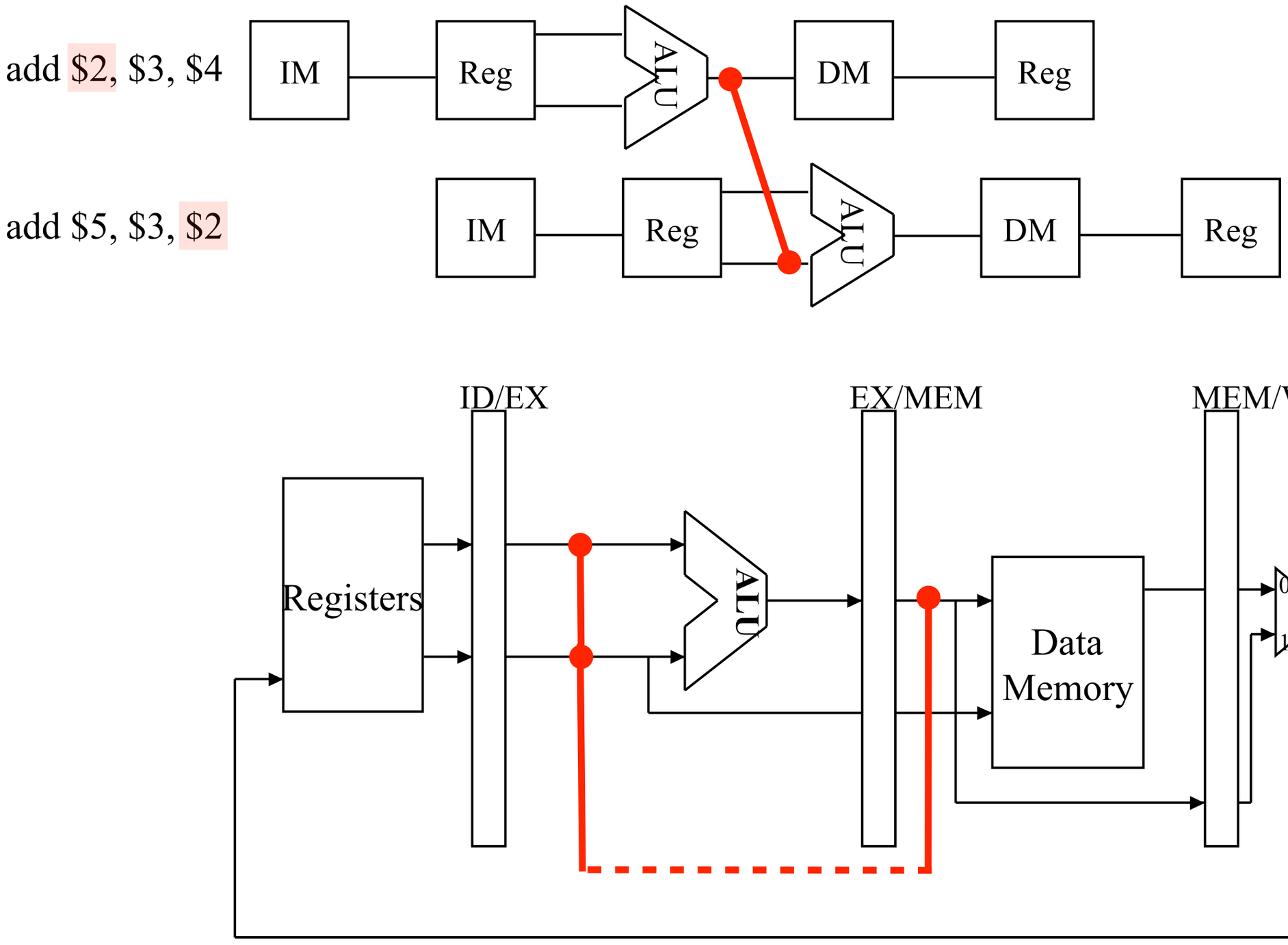

add $2, $3, $4
IM
Reg
ALU
DM
Reg
add $5, $3, $2
IM
Reg
ALU
DM
Reg


ID/EX
EX/MEM
MEM/WB
Registers
ALU
Data
Memory
0
1

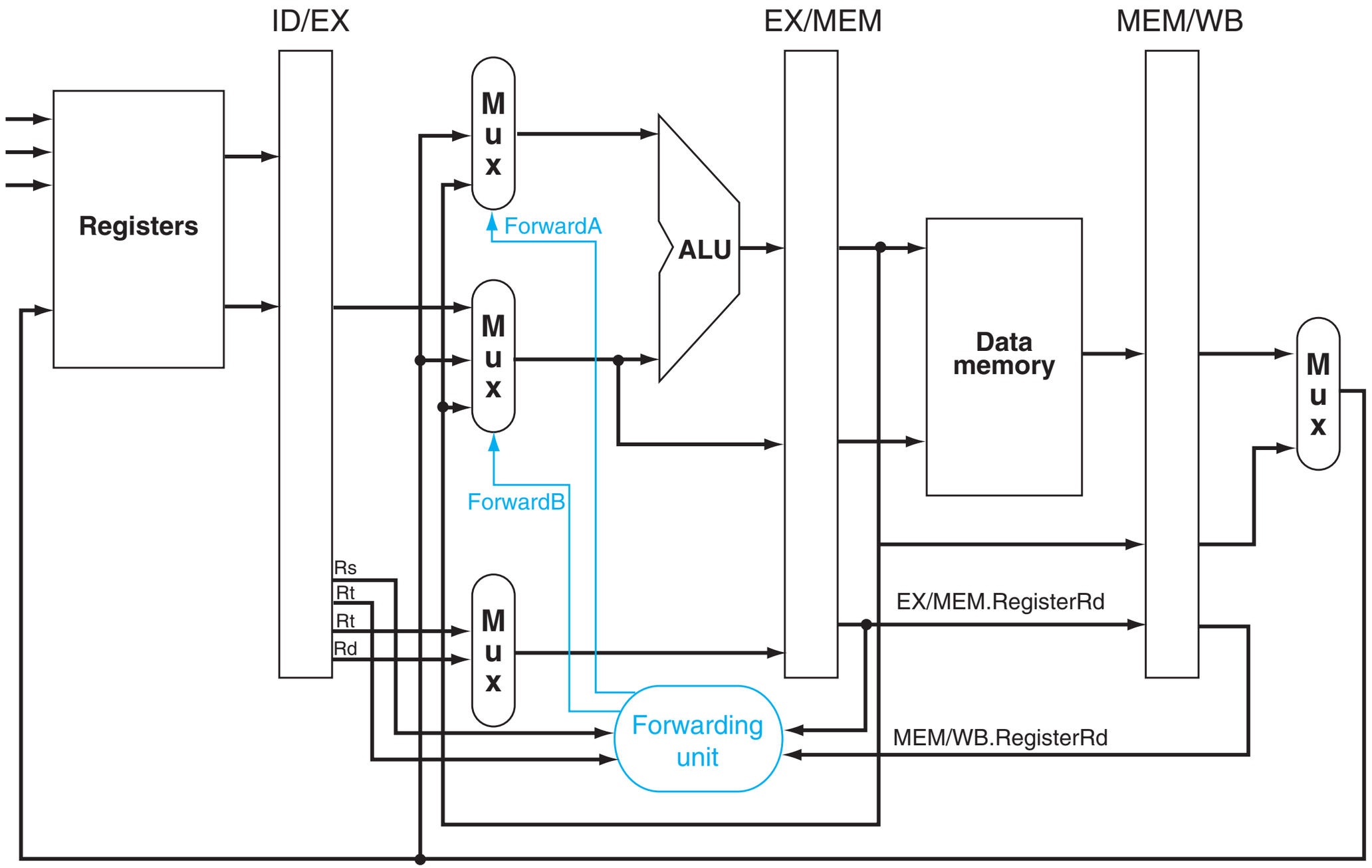
Forwarding


add $2, $3, $4
IM
Reg
ALU
DM
Reg
add $5, $3, $2
IM
Reg
ALU
DM
Reg
ID/EX
EX/MEM
MEM/WB
Registers
ALU
Data Memory
0
1

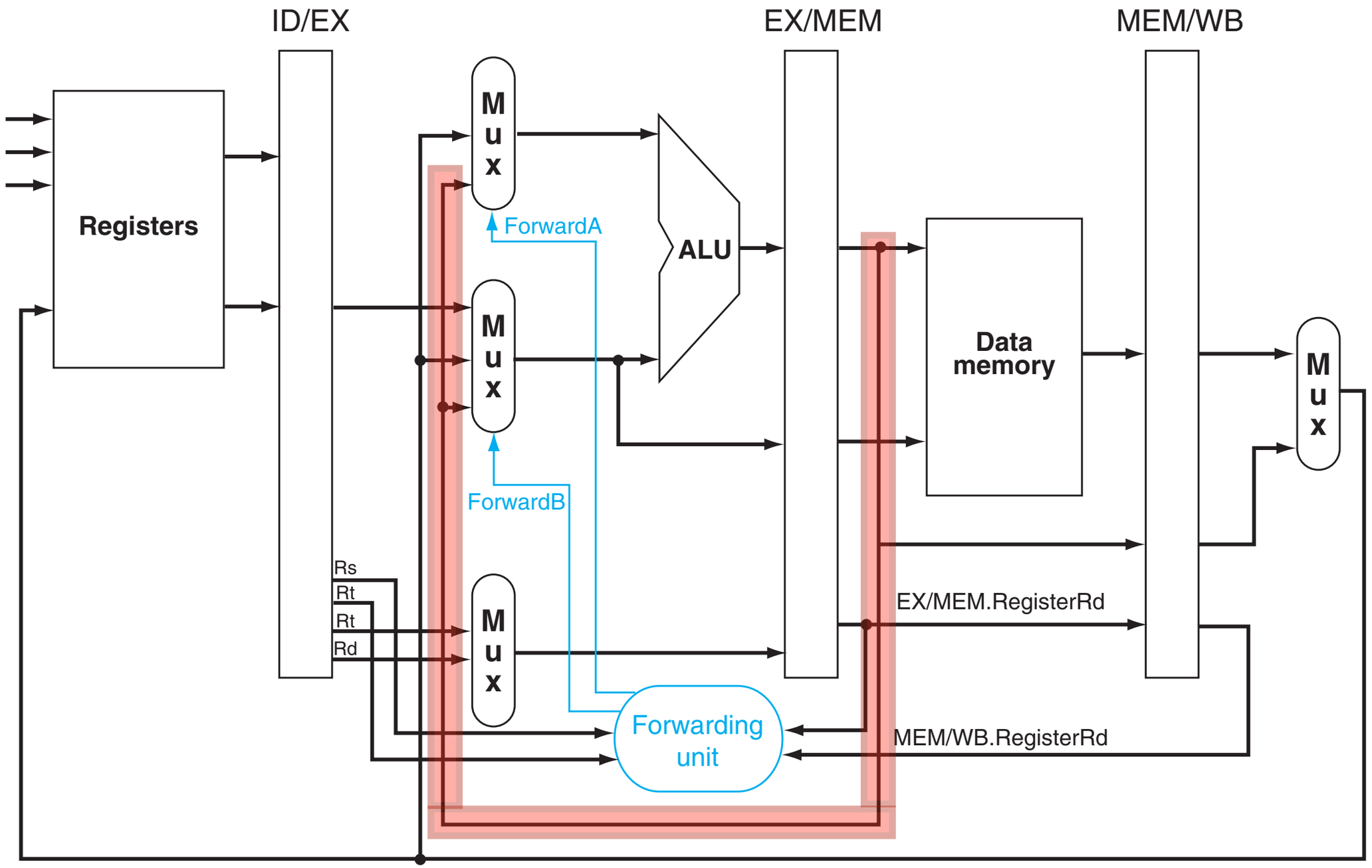
Forwarding


add $2, $3, $4
IM
Reg
ALU
DM
Reg
add $5, $3, $2
IM
Reg
ALU
DM
Reg
ID/EX
EX/MEM
MEM/WB
Registers
ALU
Data Memory
0
1

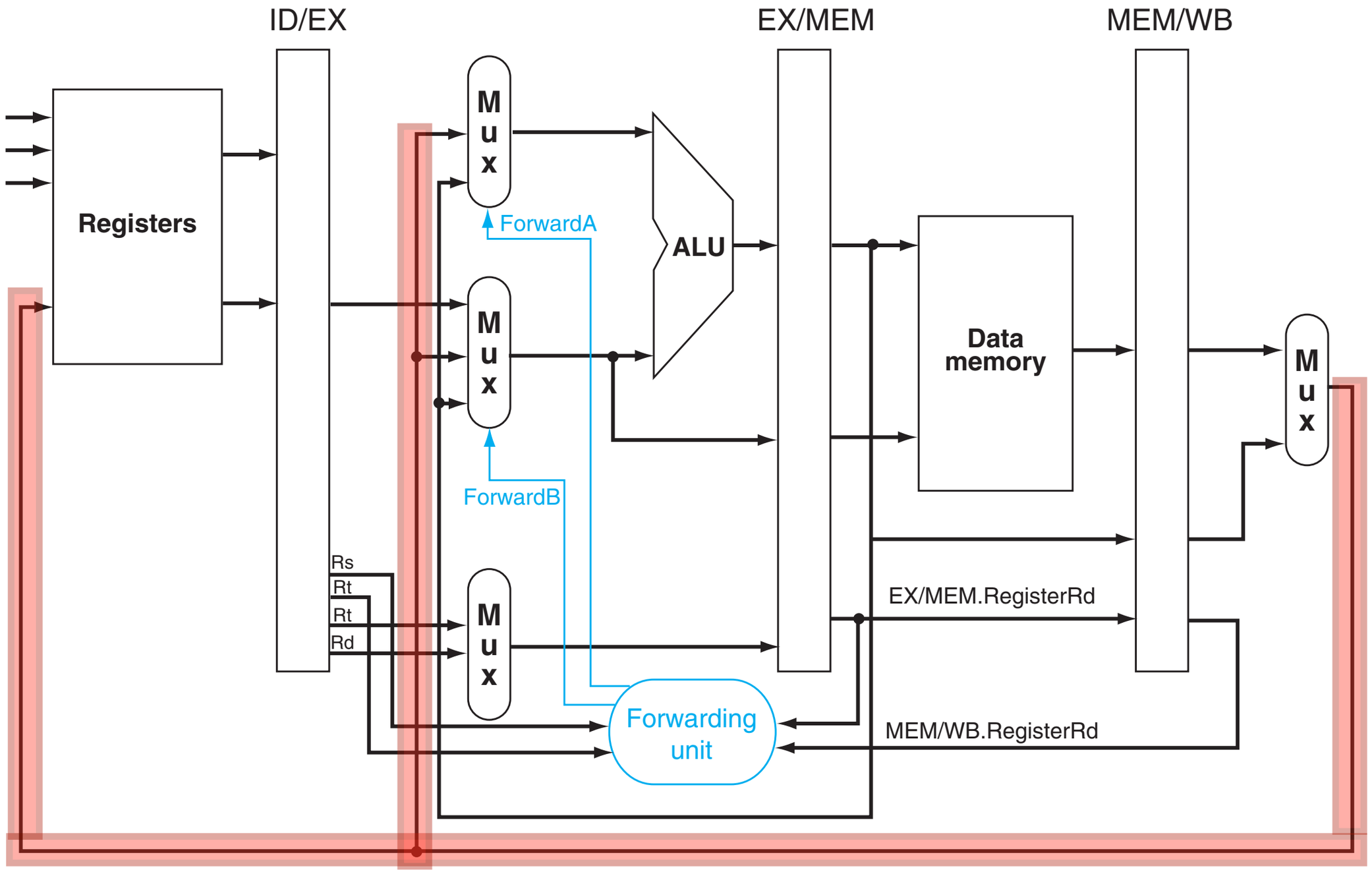
Reducing Hazards via Forwarding



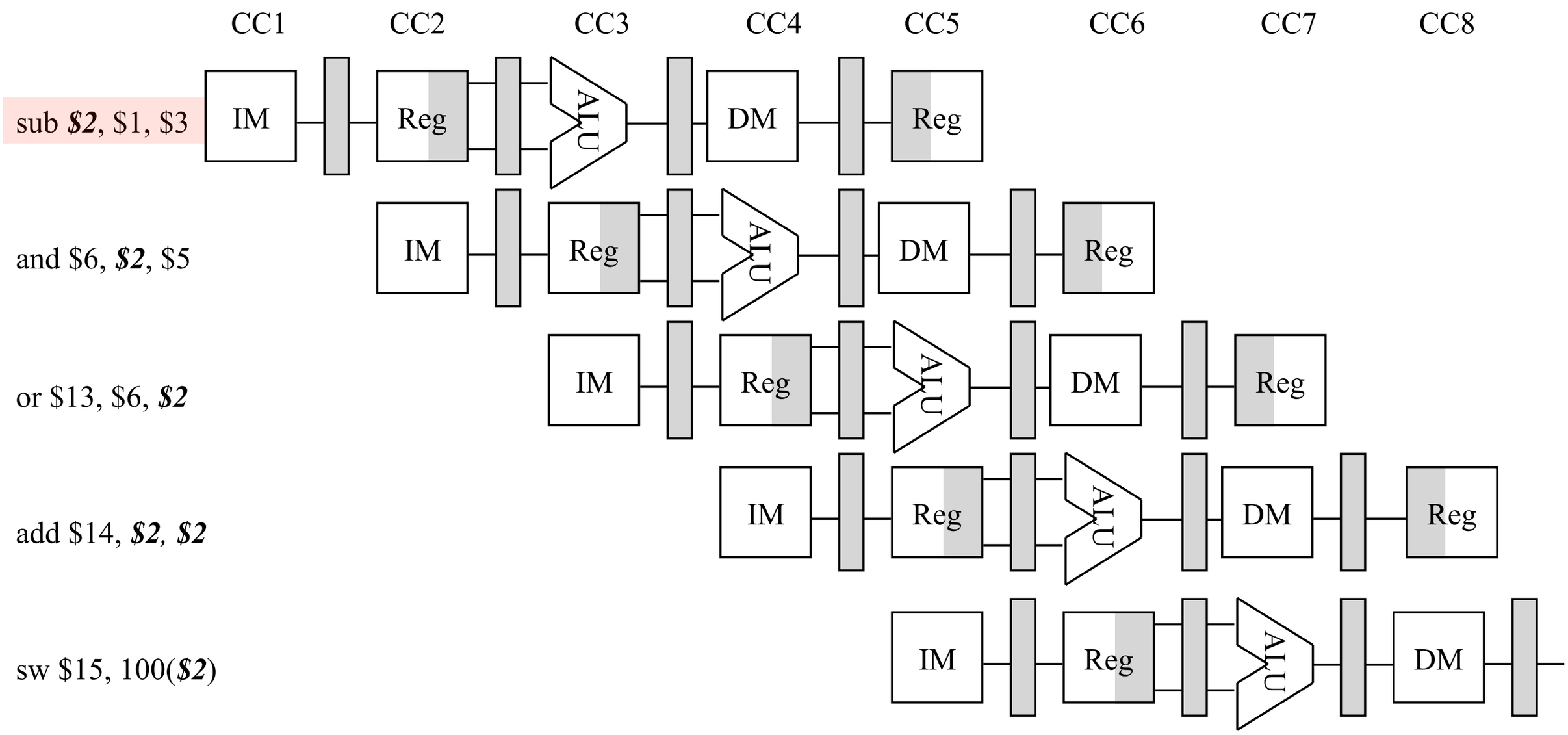
Reducing Hazards via Forwarding



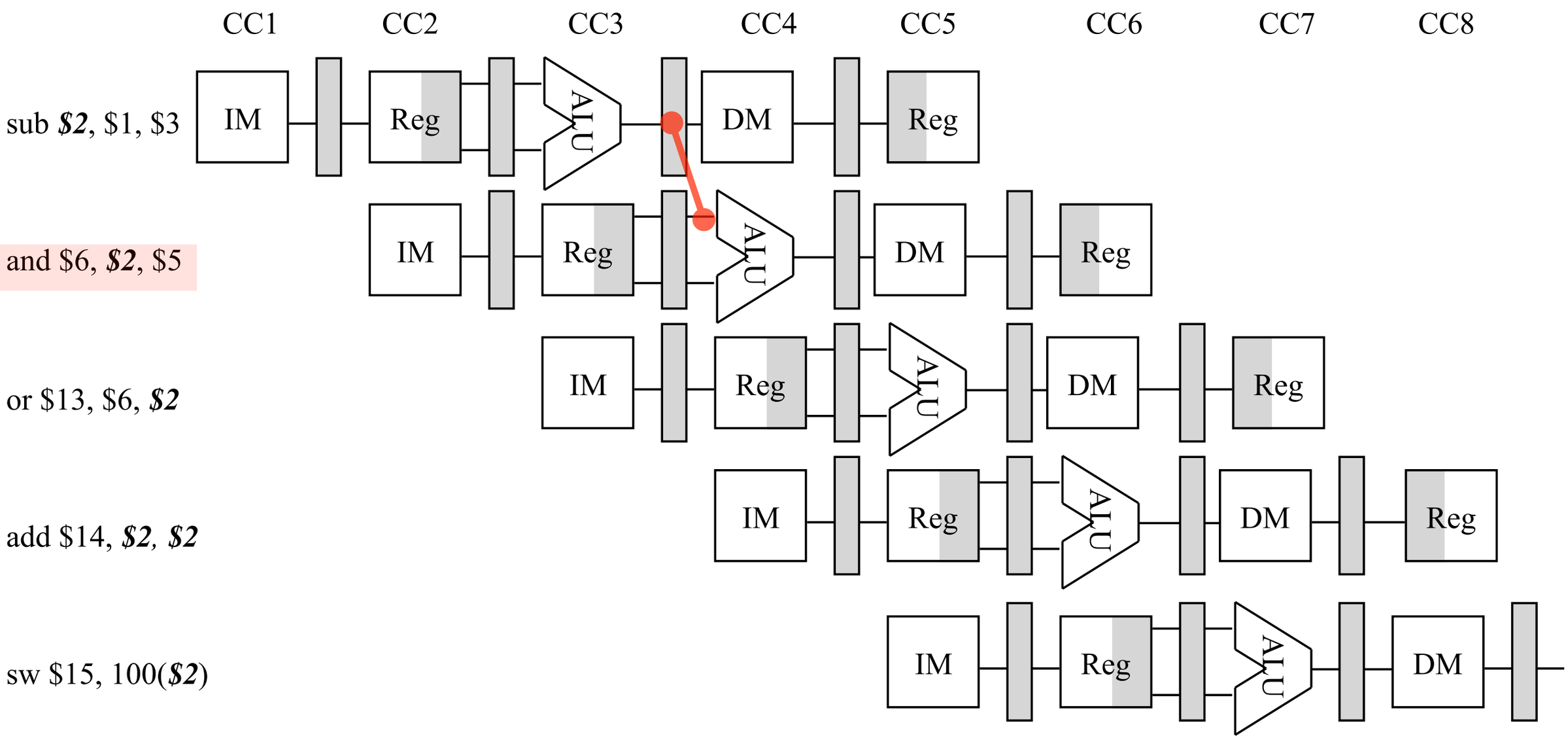
Reducing Hazards via Forwarding



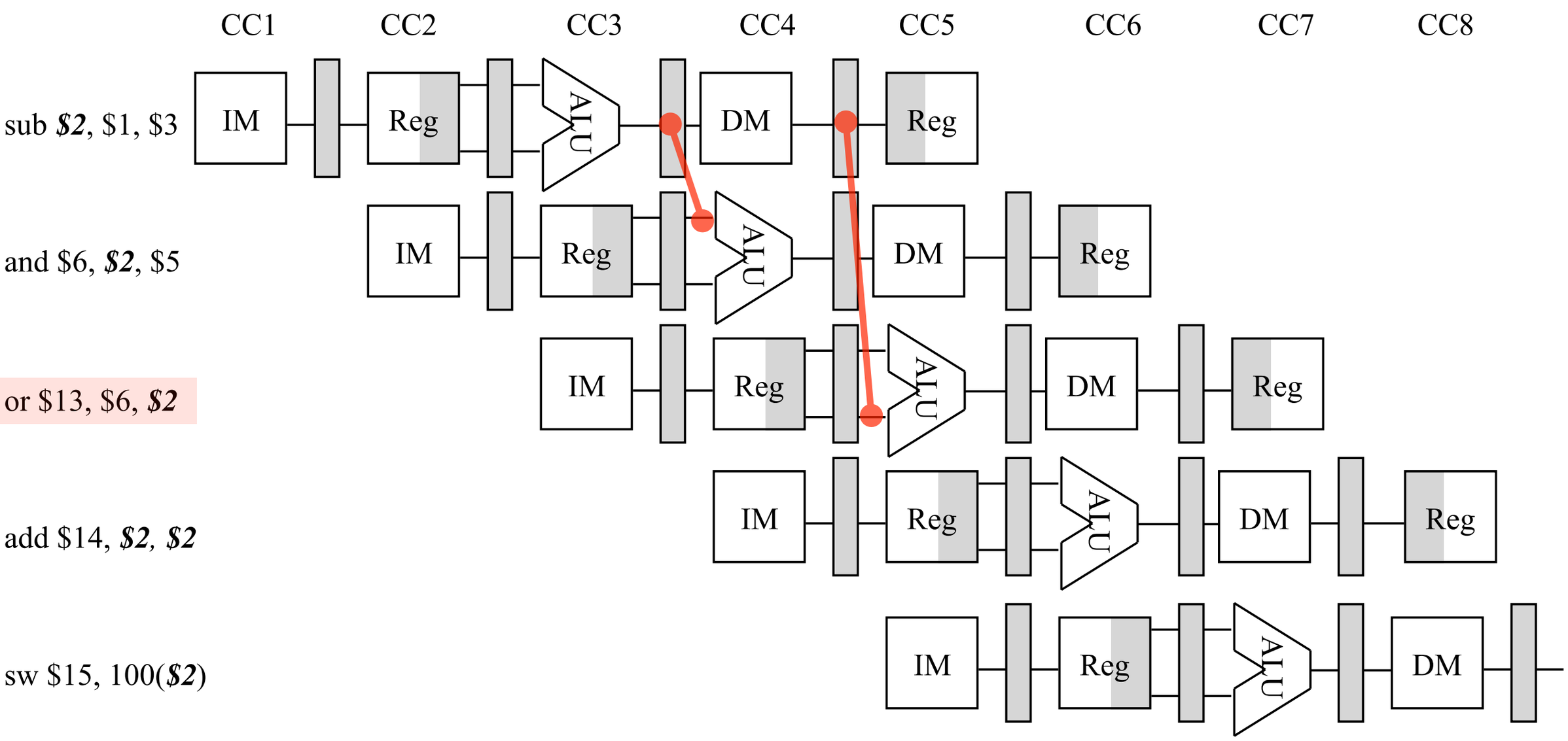
Eliminating Hazards via Forwarding



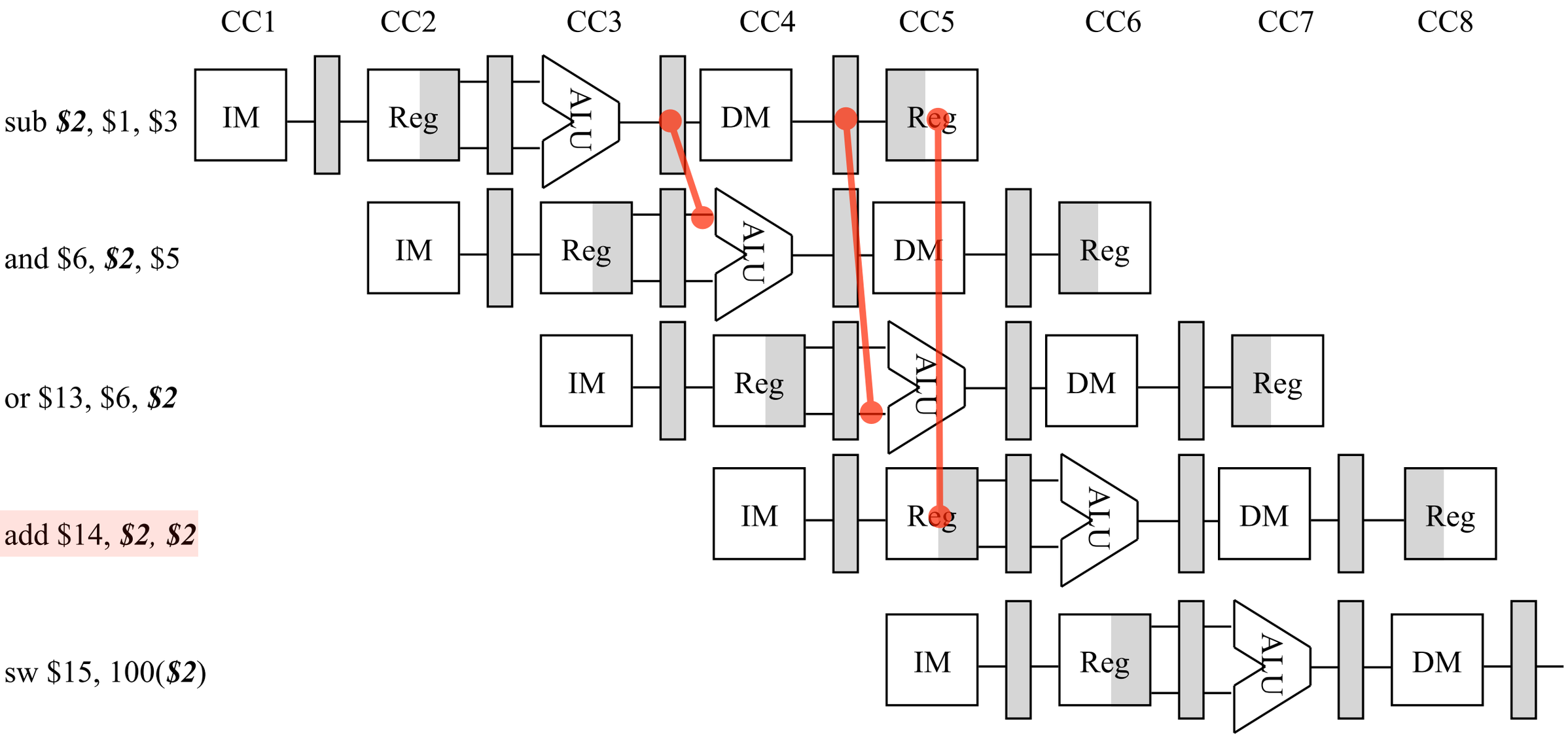
Eliminating Hazards via Forwarding



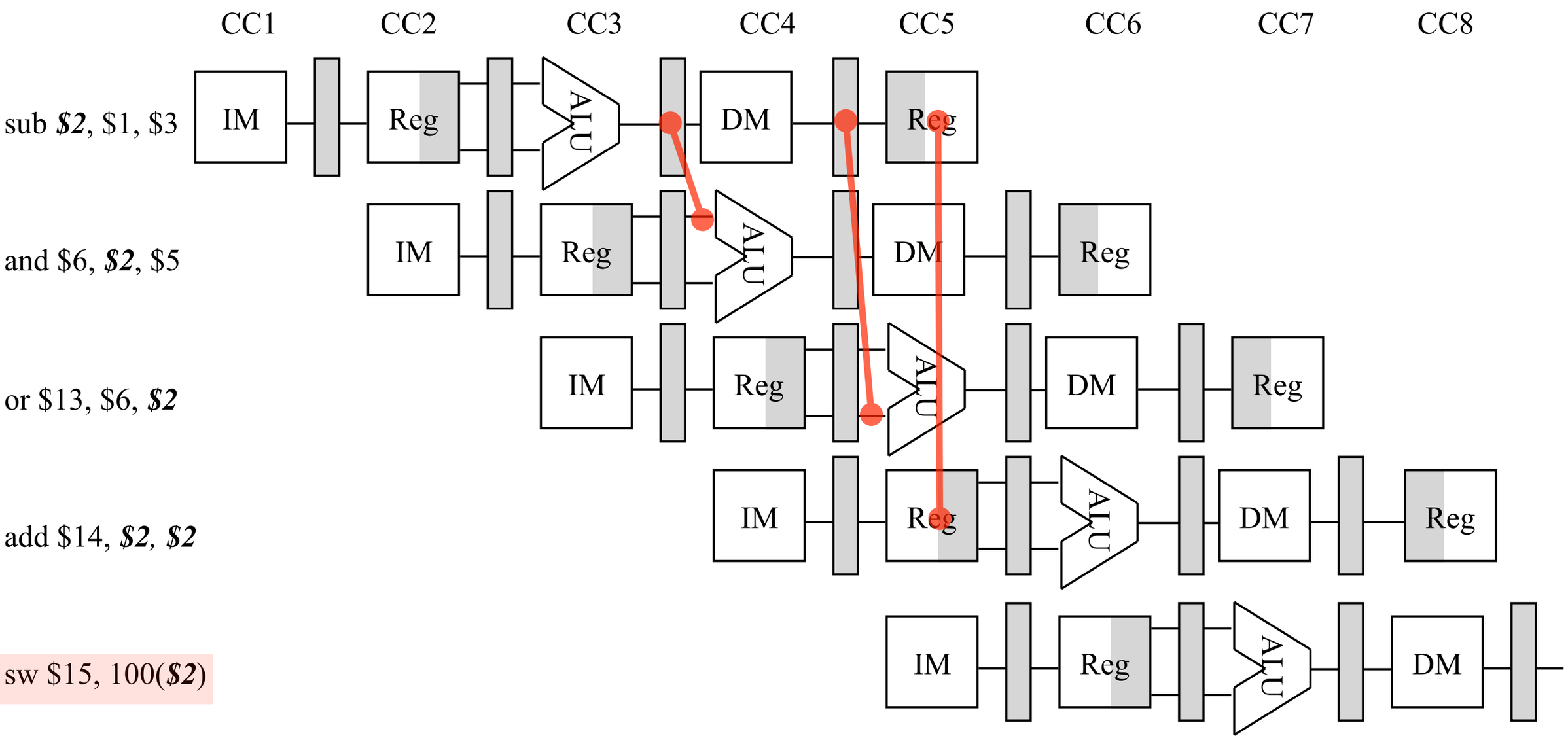
Eliminating Hazards via Forwarding



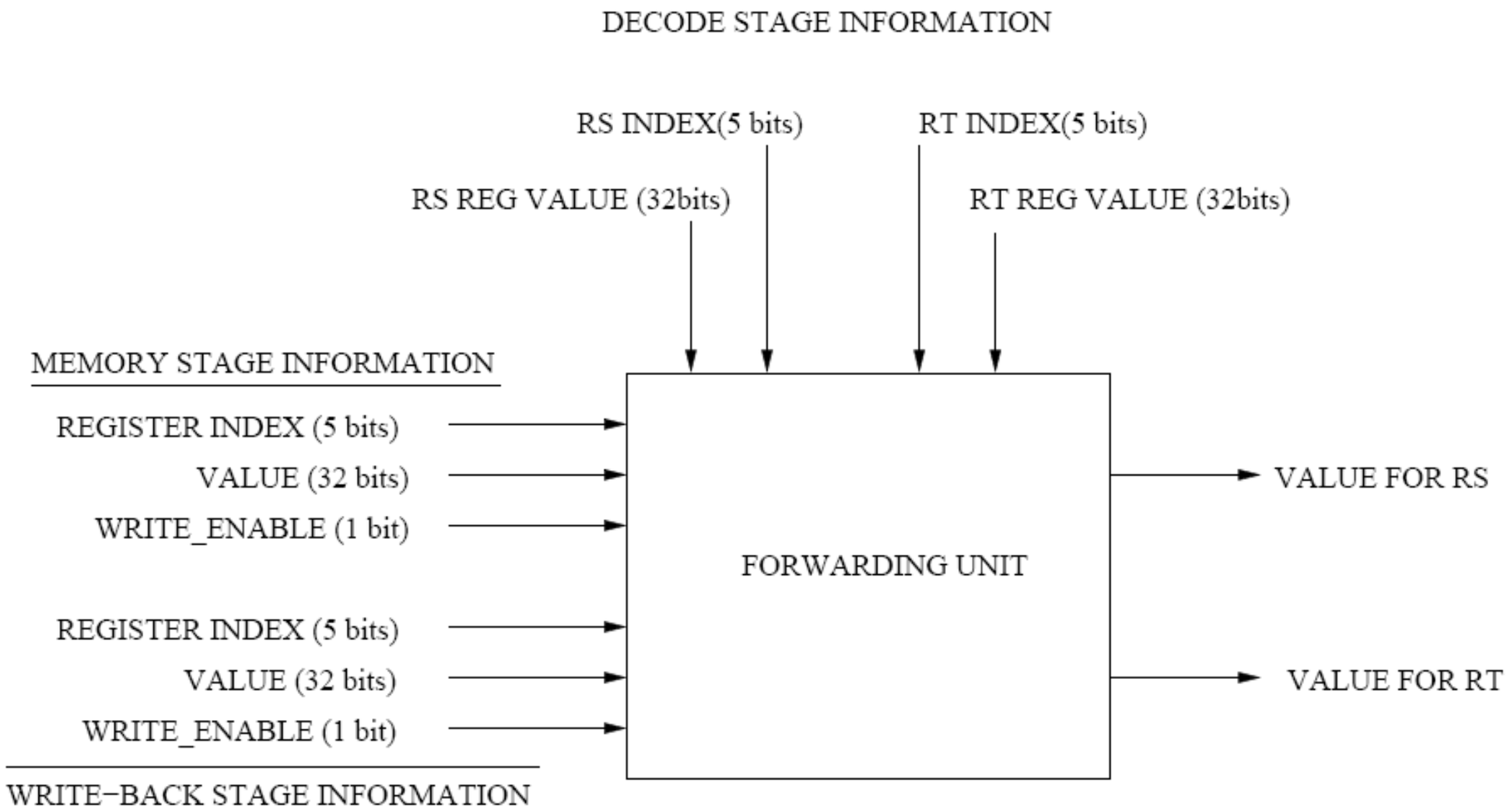
Eliminating Hazards via Forwarding



Eliminating Hazards via Forwarding



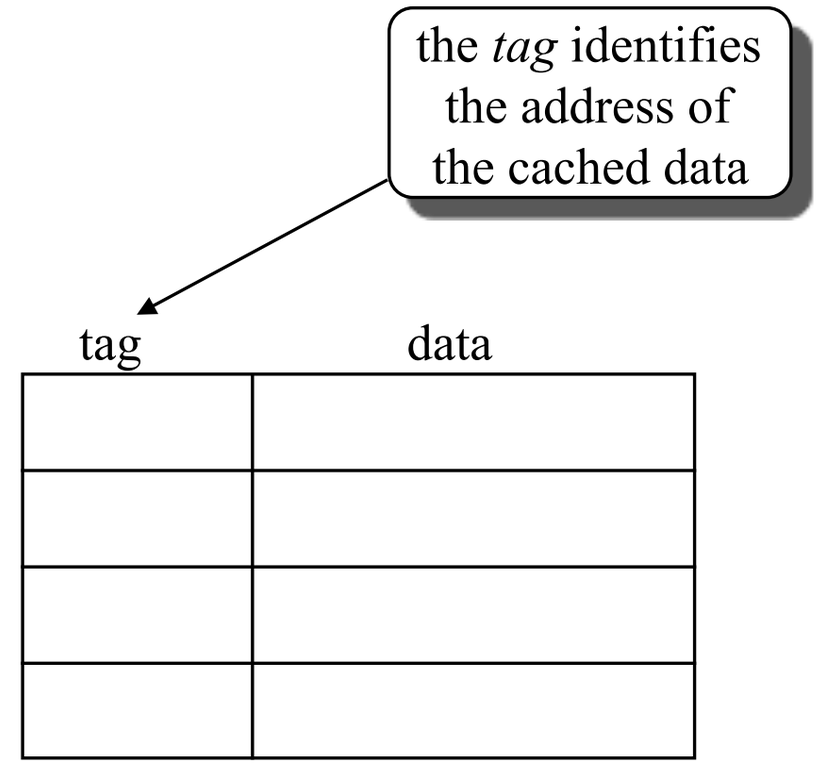
Test Preview



A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



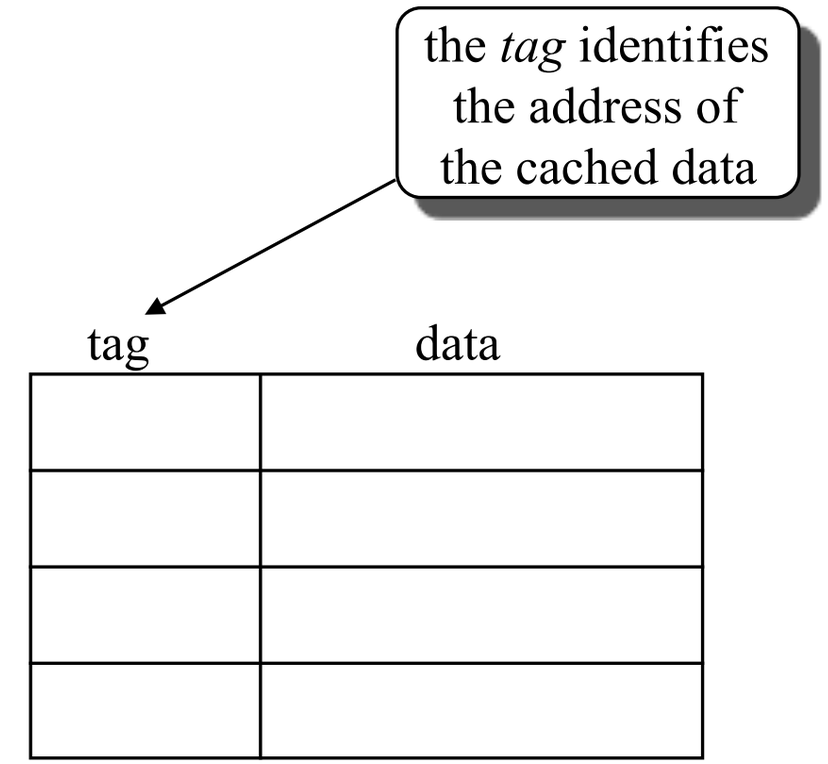
4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called _____
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

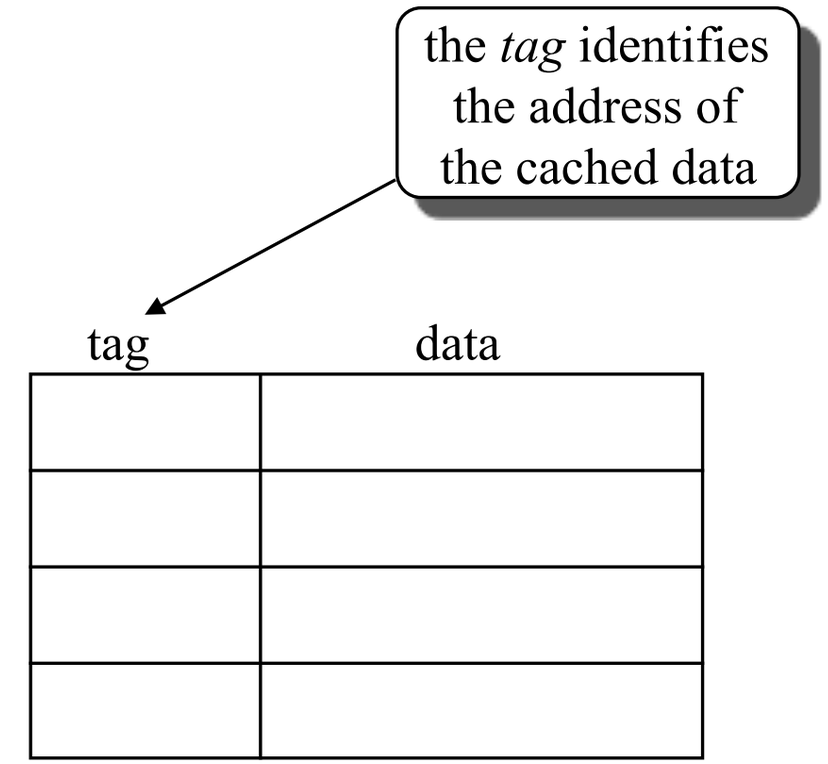


4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



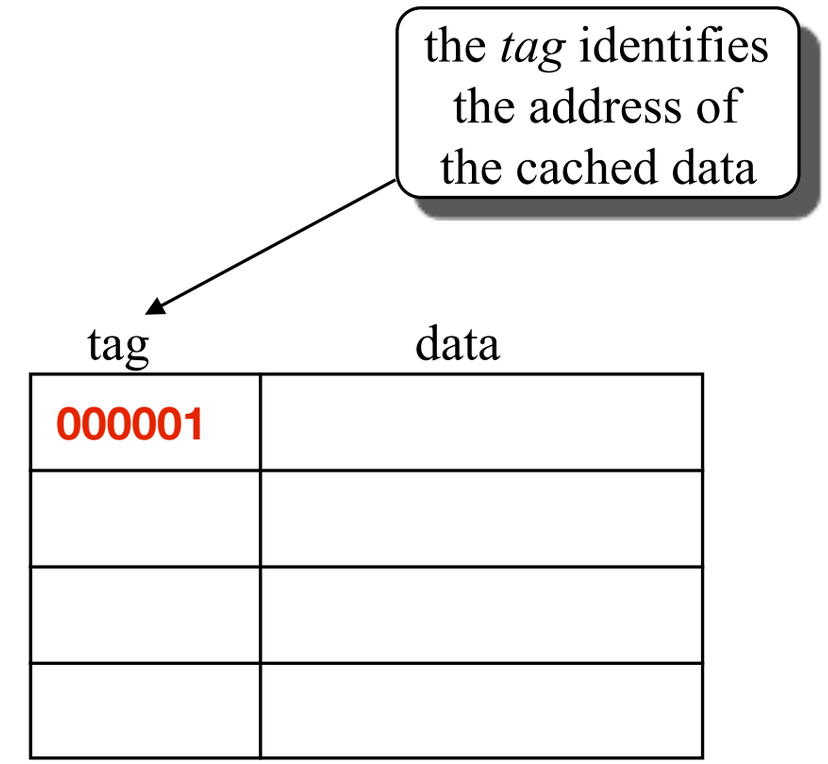
4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



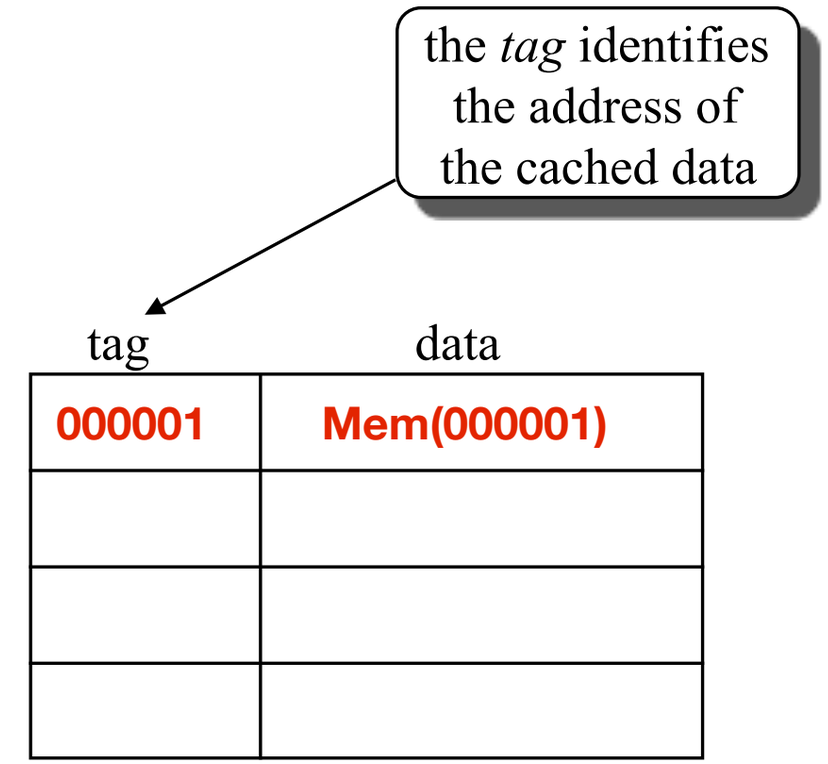
4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



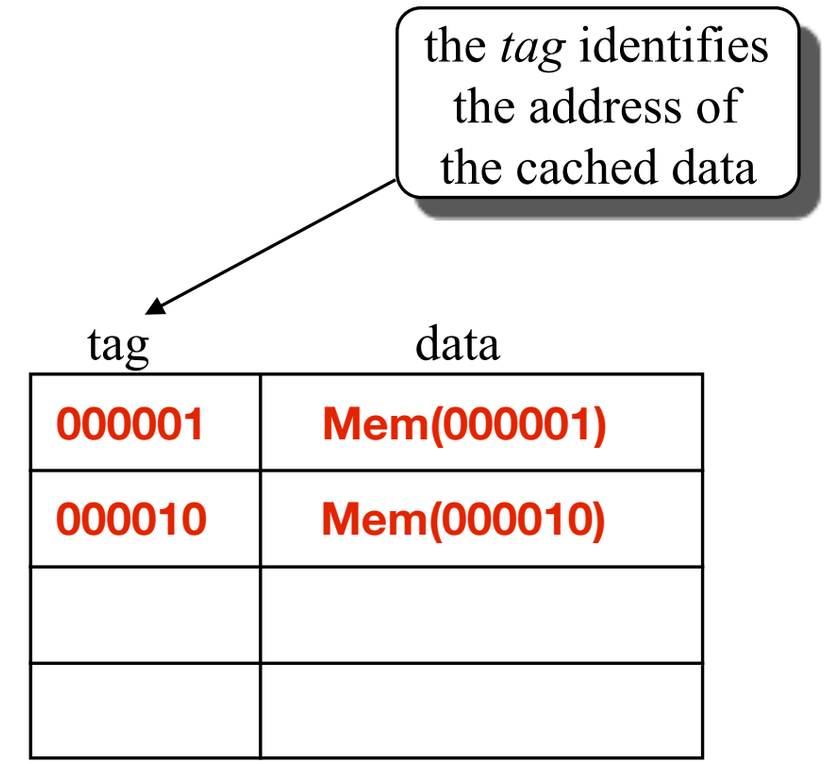
4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



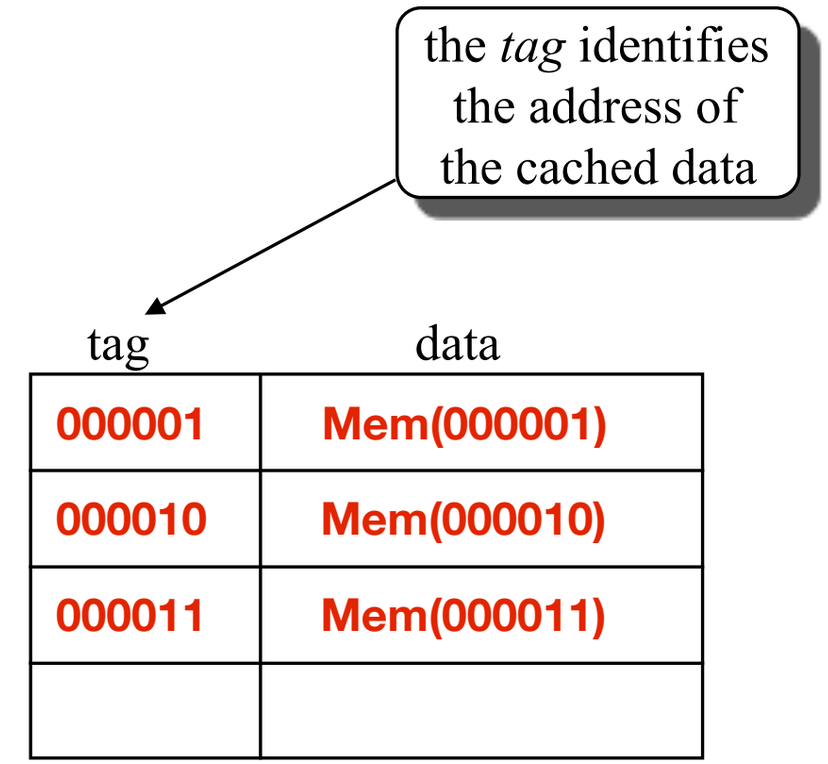
4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



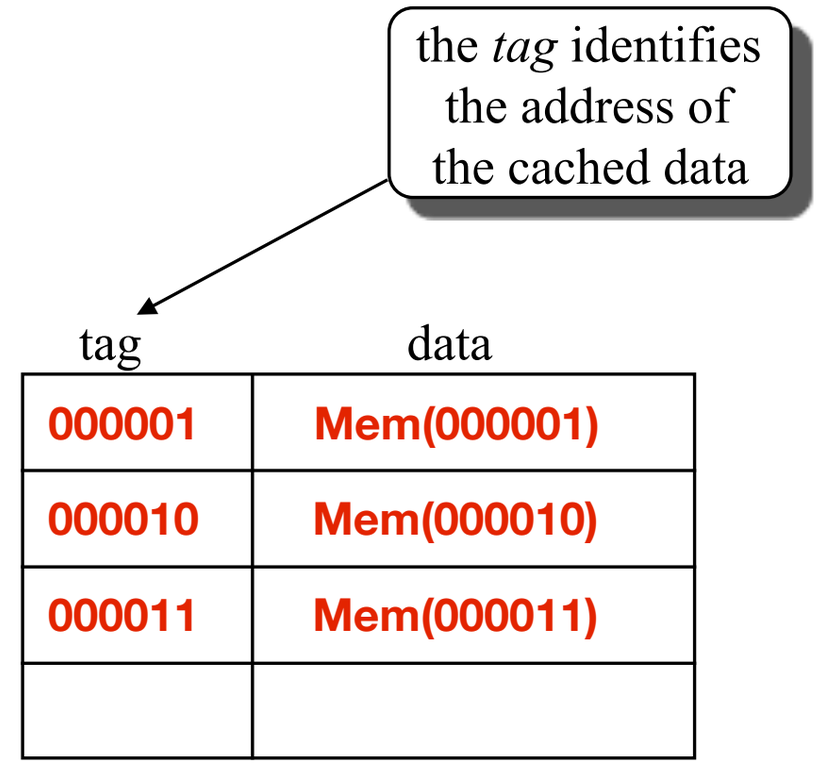
4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



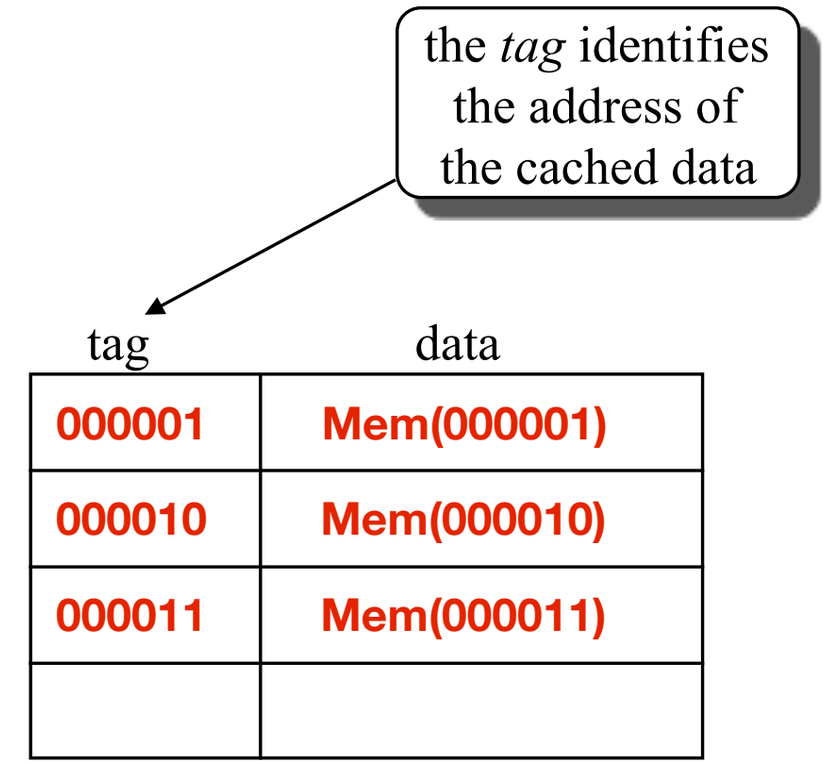
4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



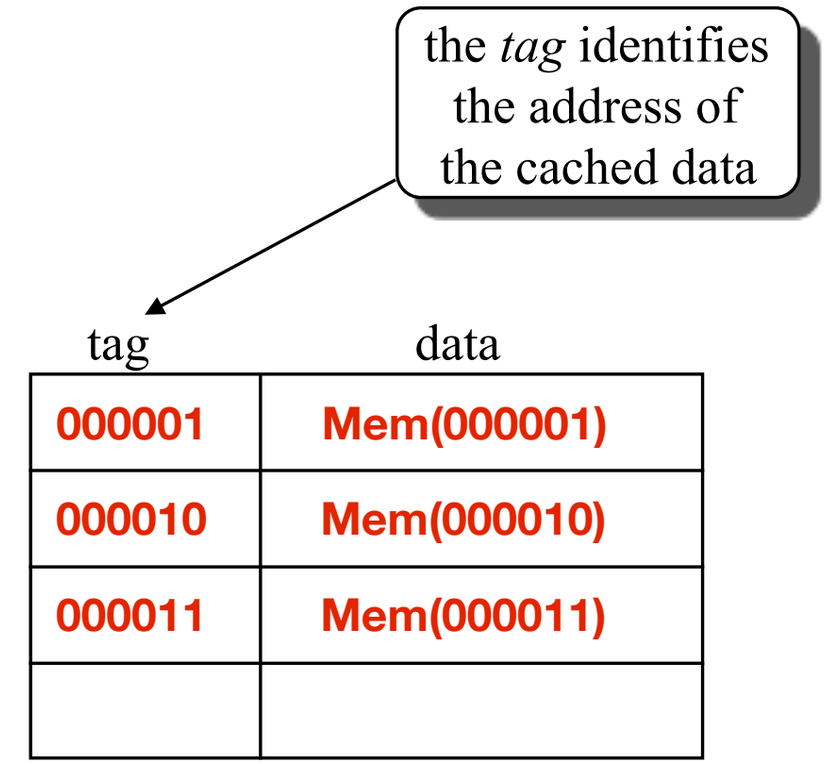
4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



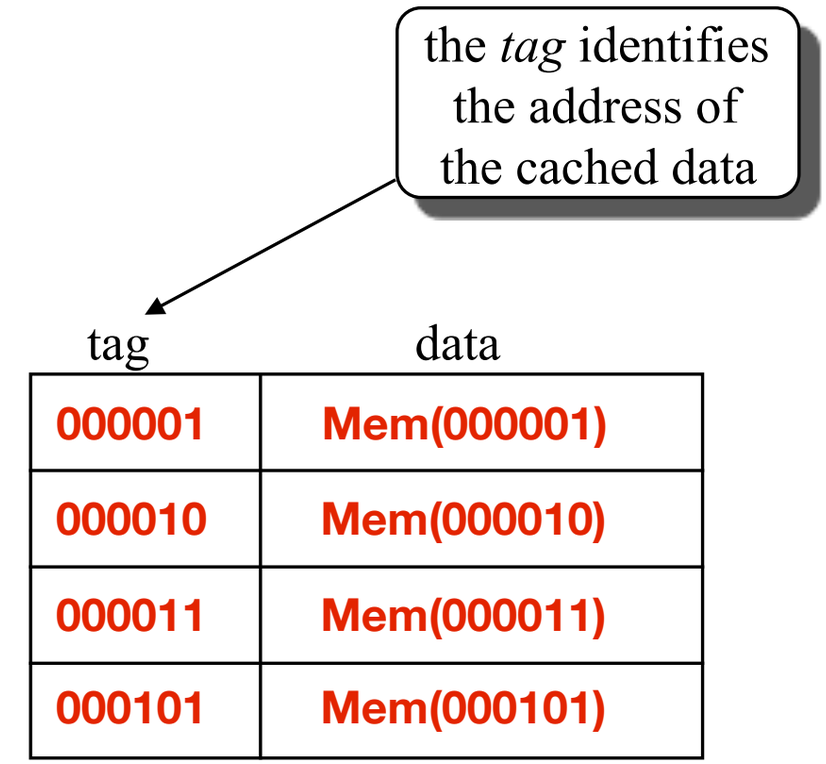
4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



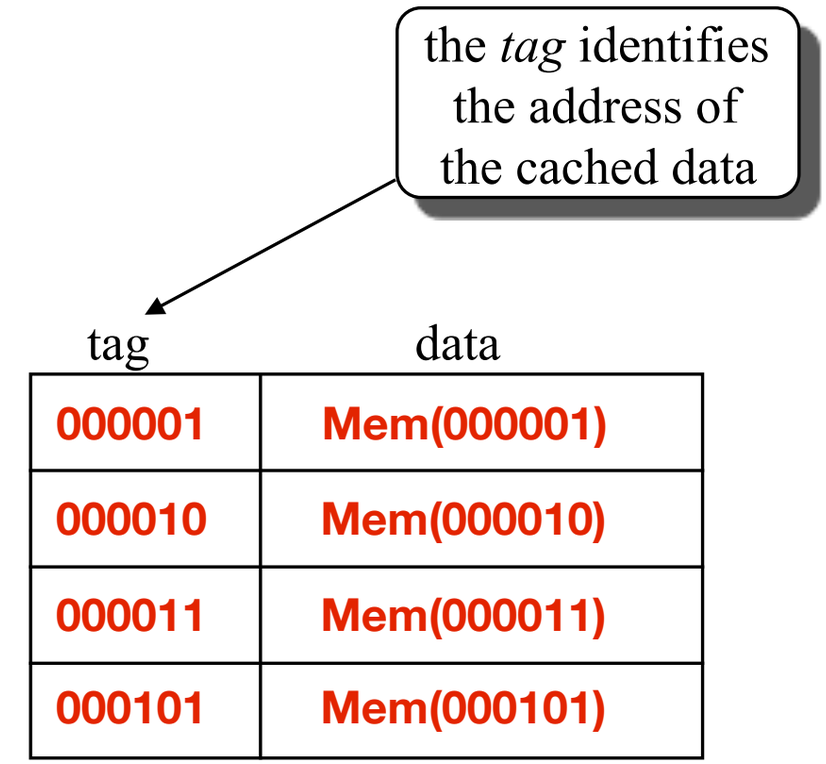
4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

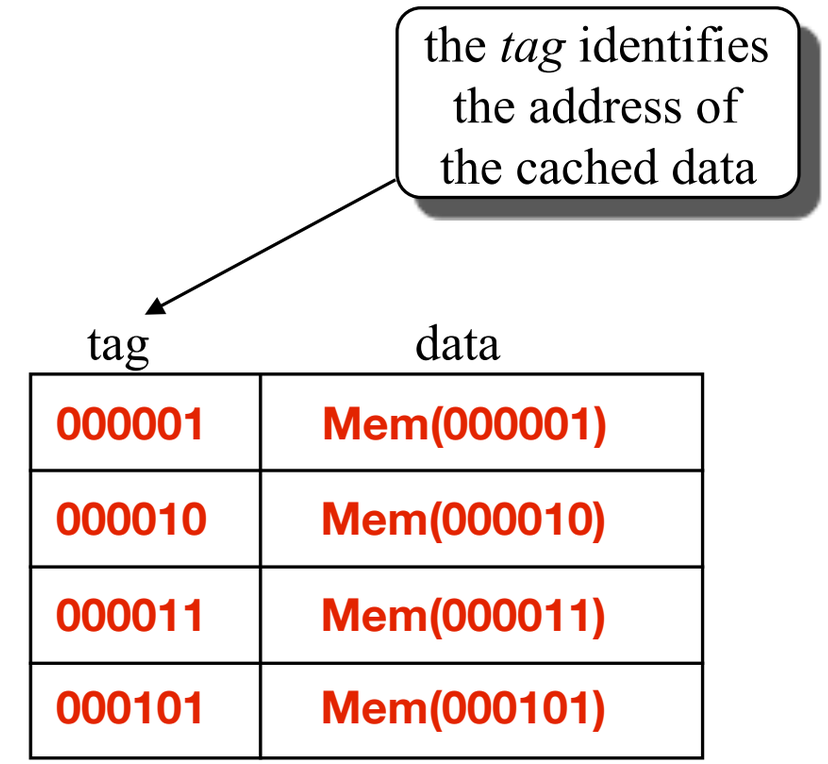


4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

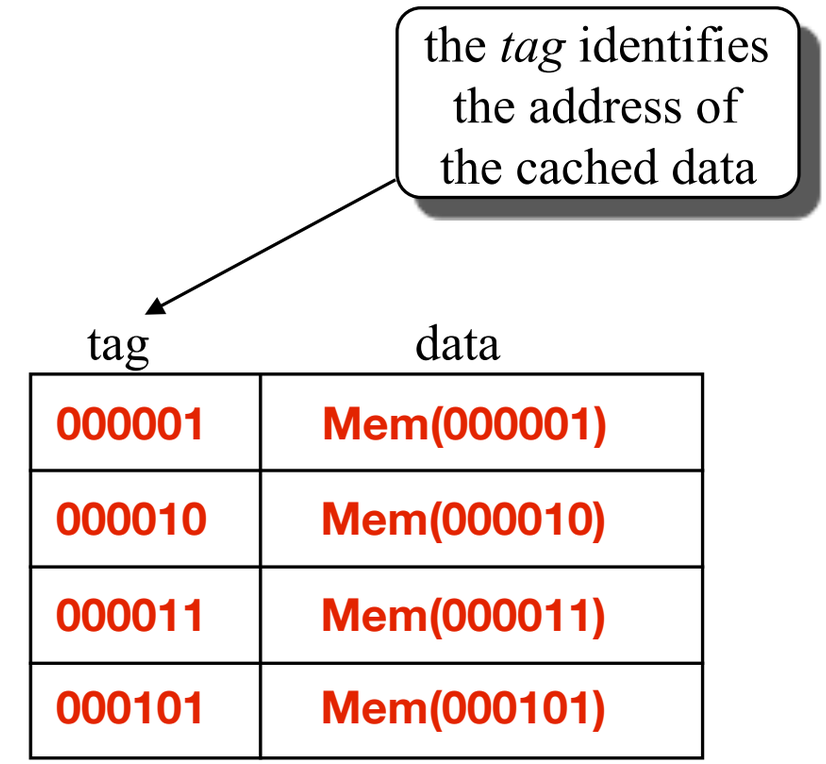


4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU ( ).

A Simple Cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

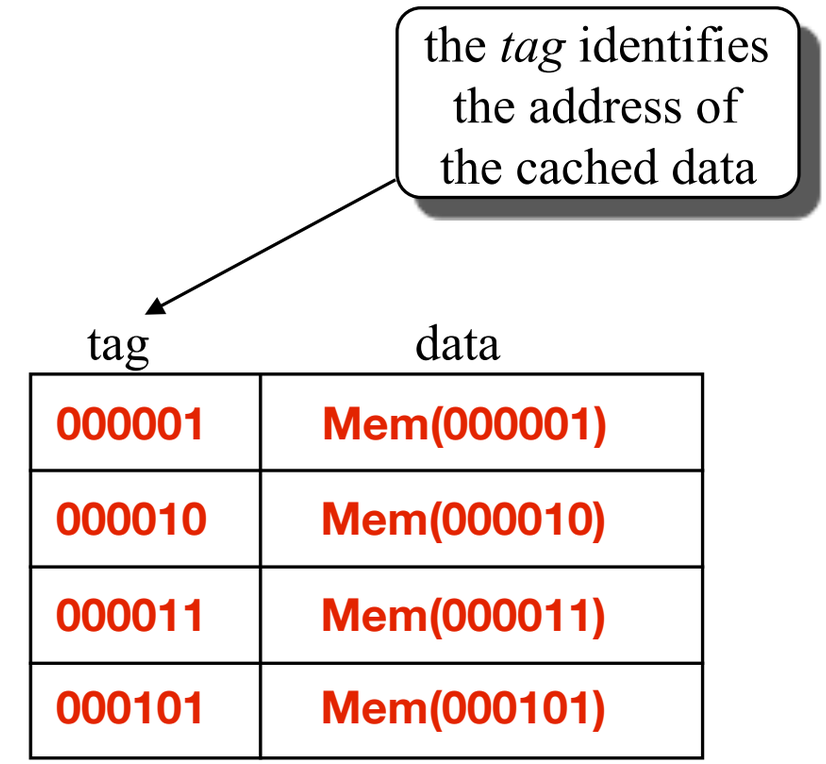


4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU (**Least Recently Used**).

A Simple Cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

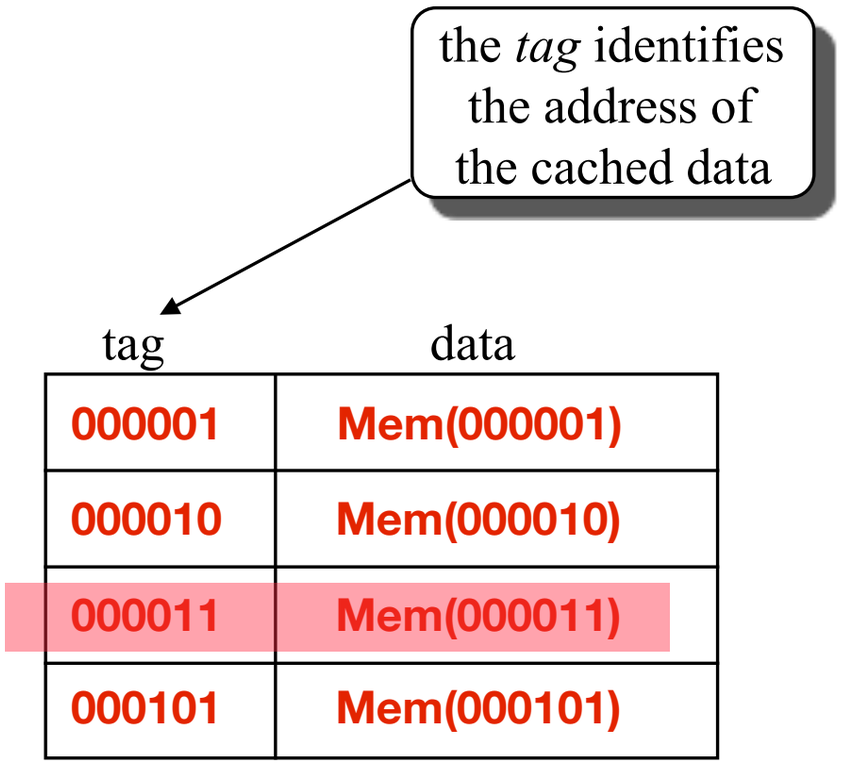


4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU (**Least Recently Used**).

A Simple Cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



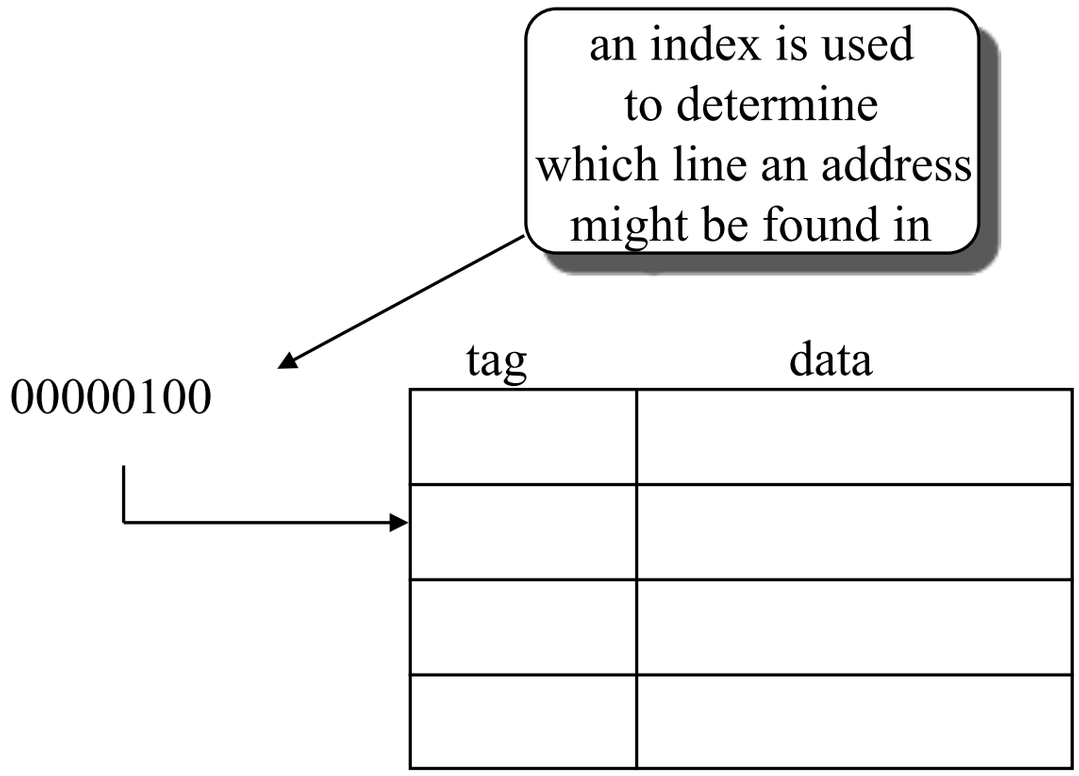
4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is LRU (**Least Recently Used**).

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



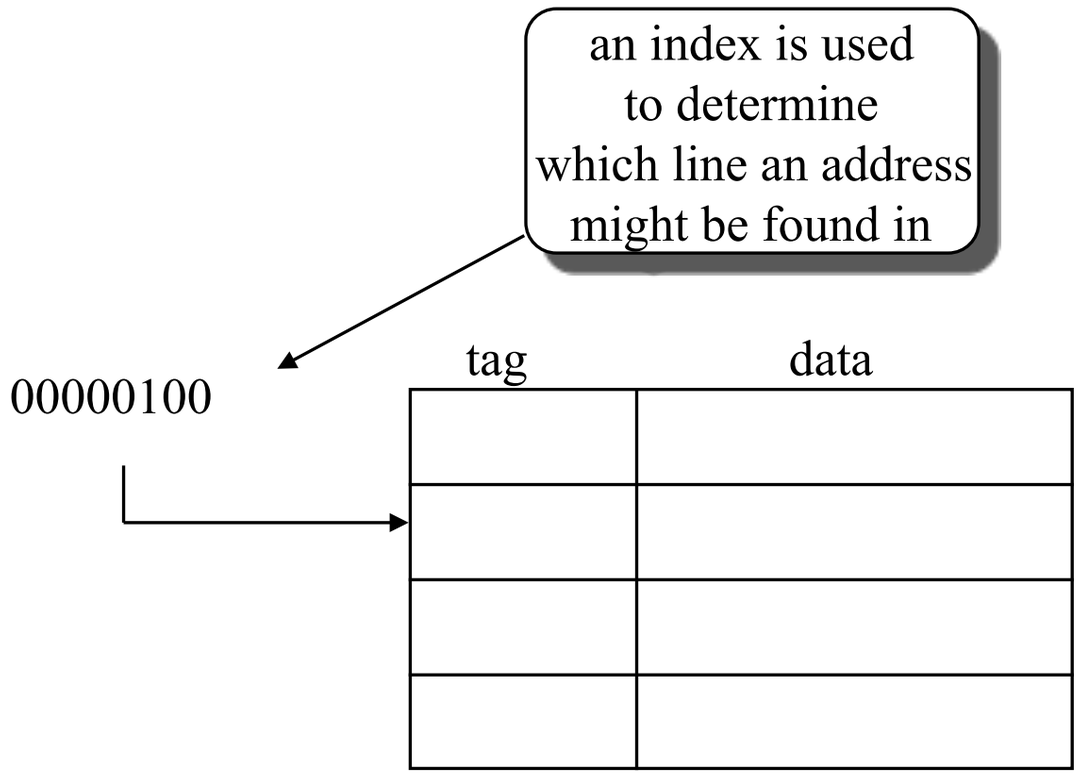
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called _____.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



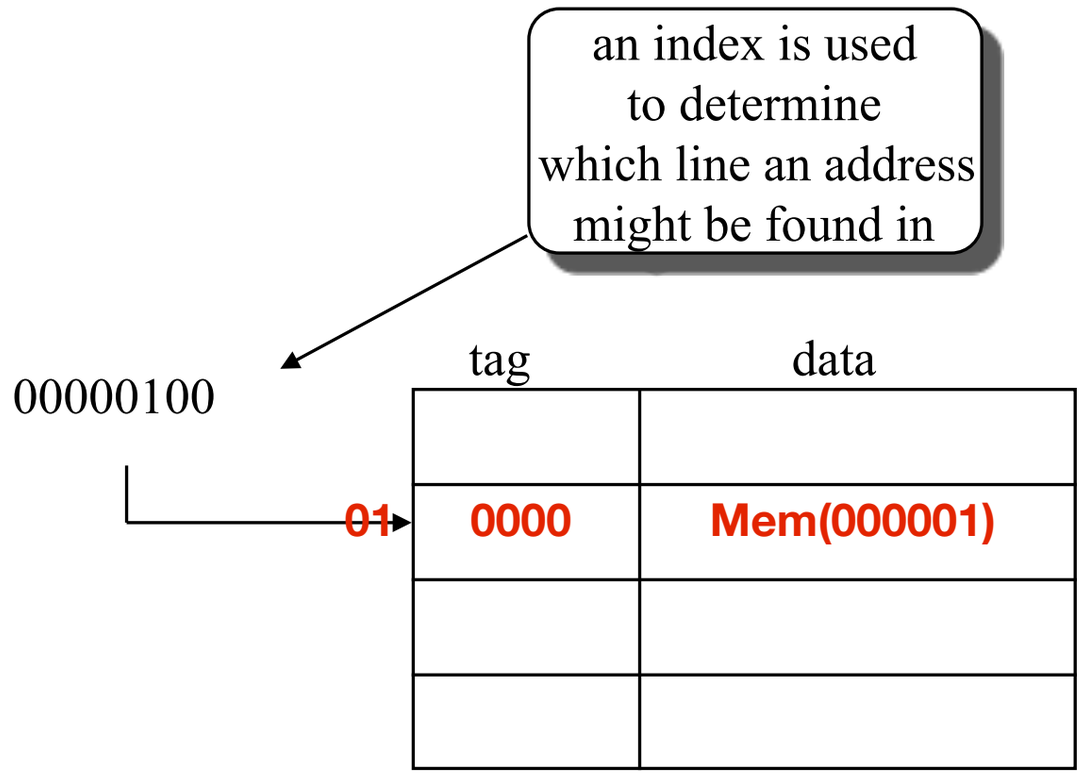
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



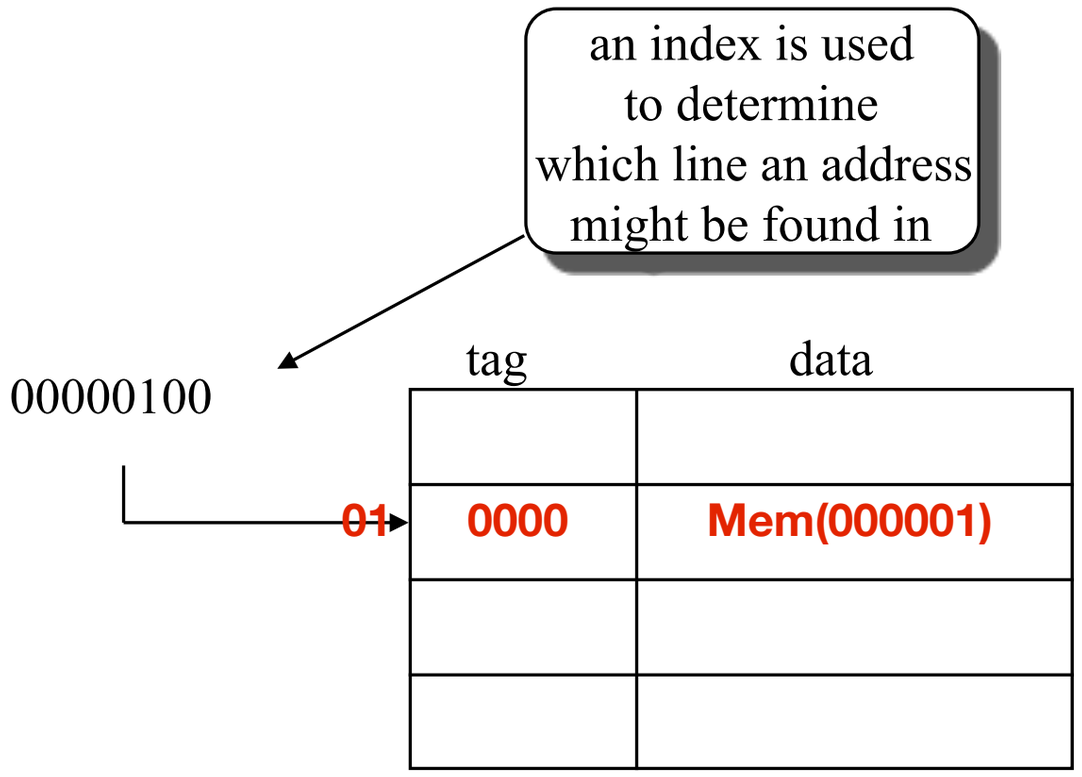
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



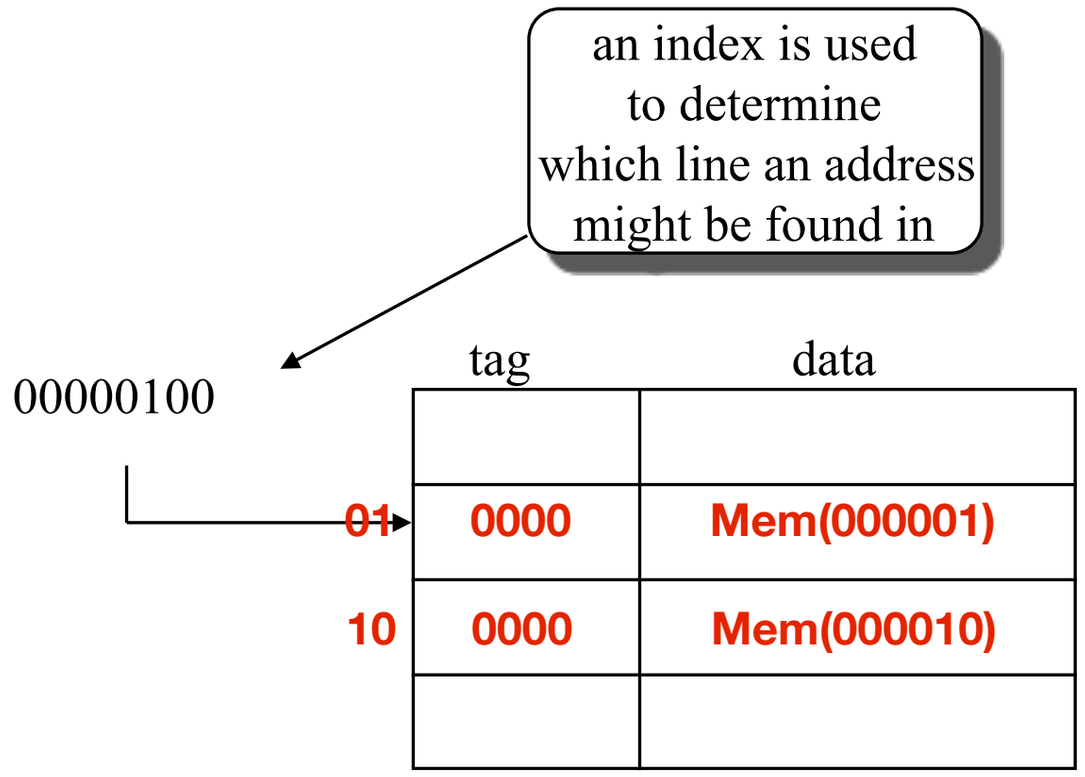
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



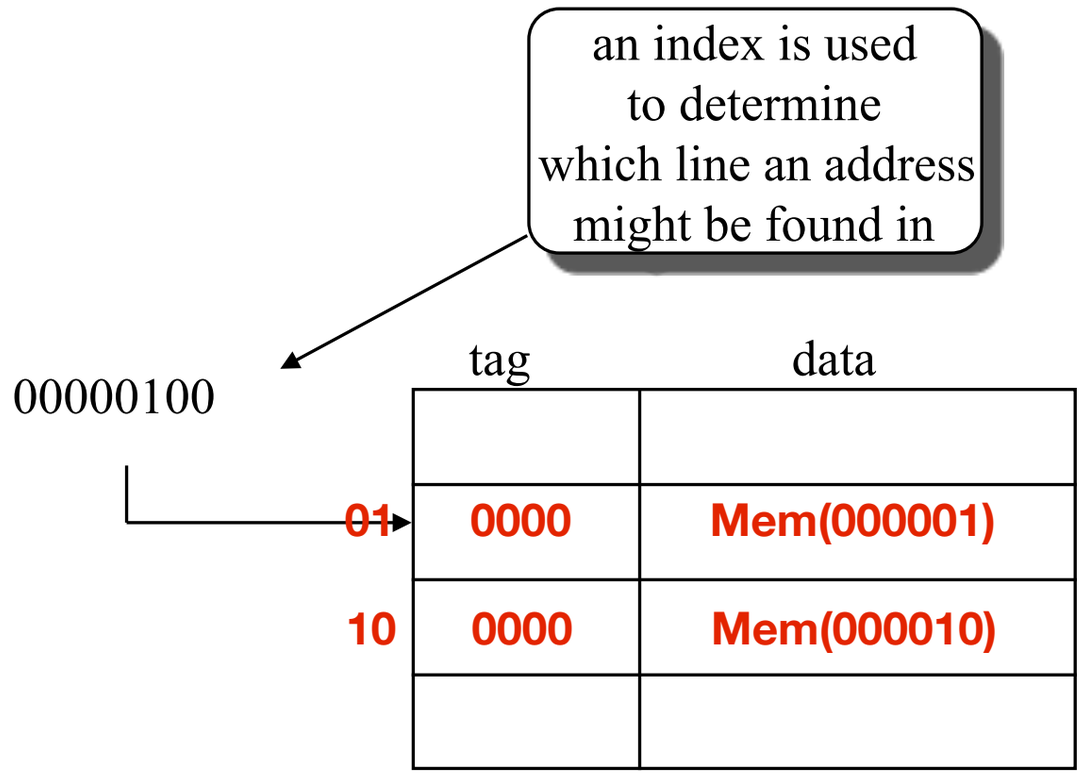
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



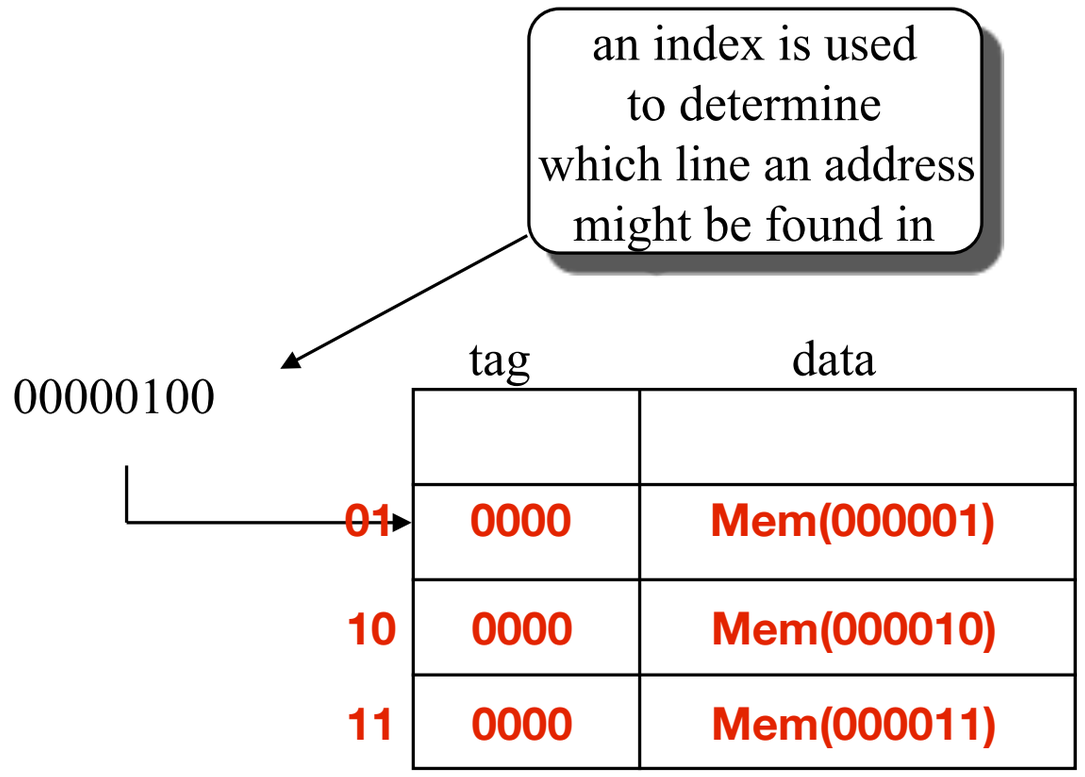
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



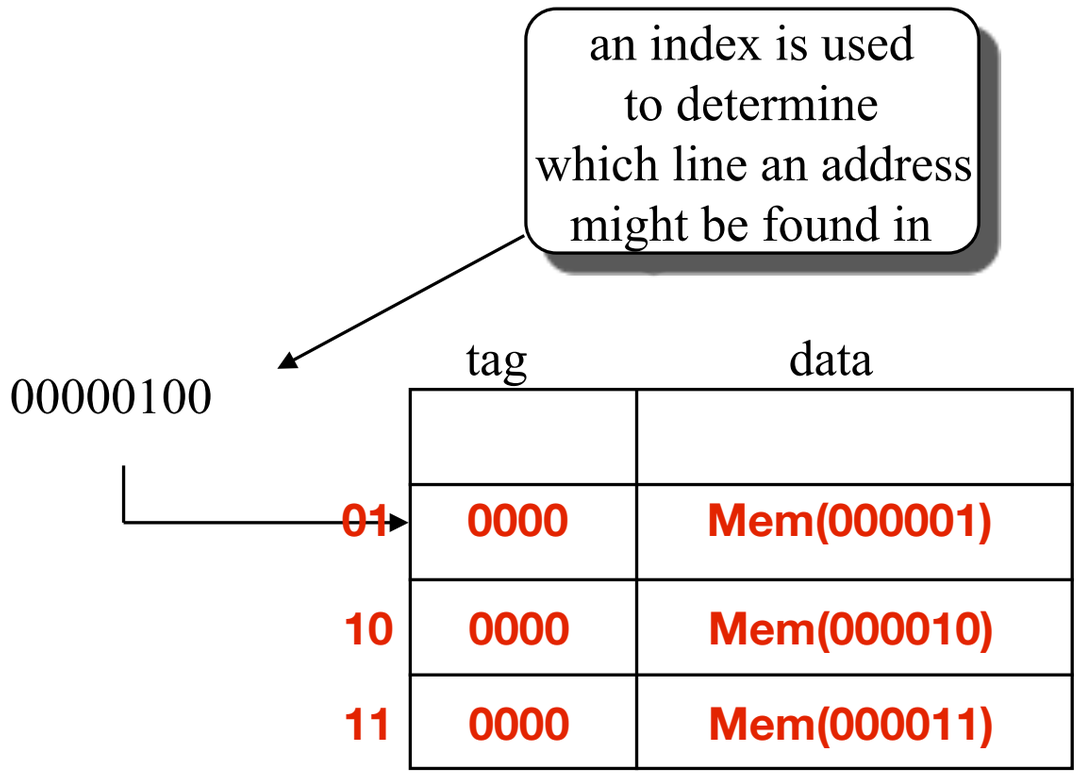
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



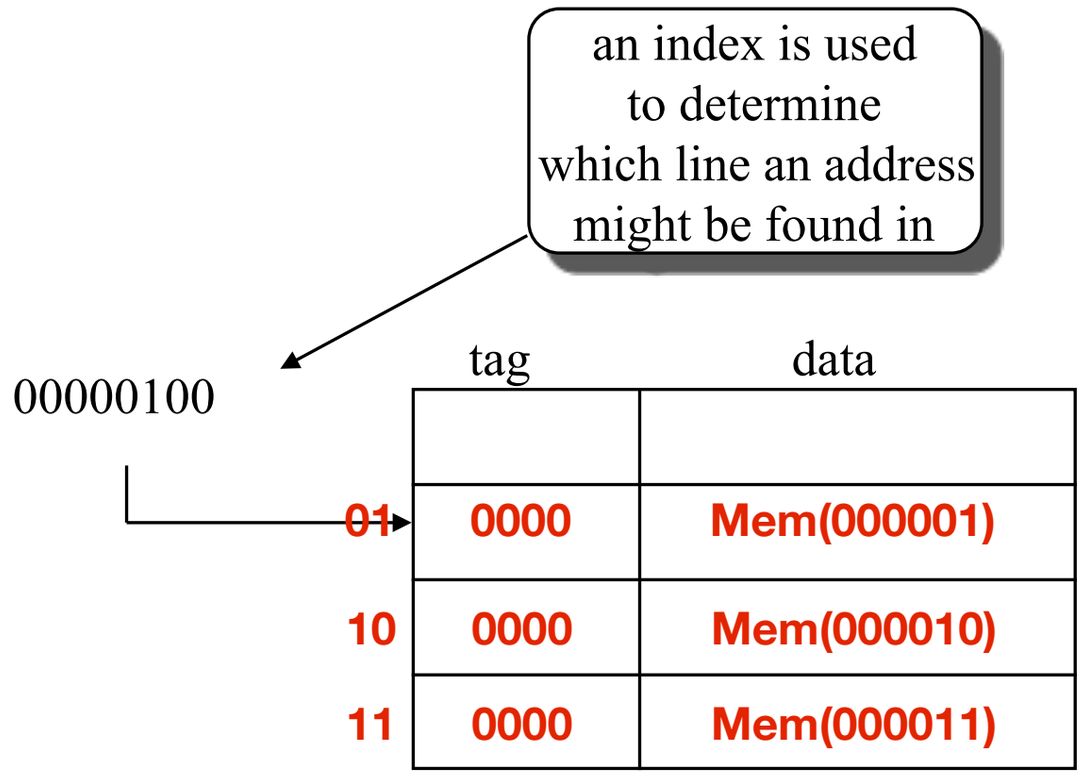
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



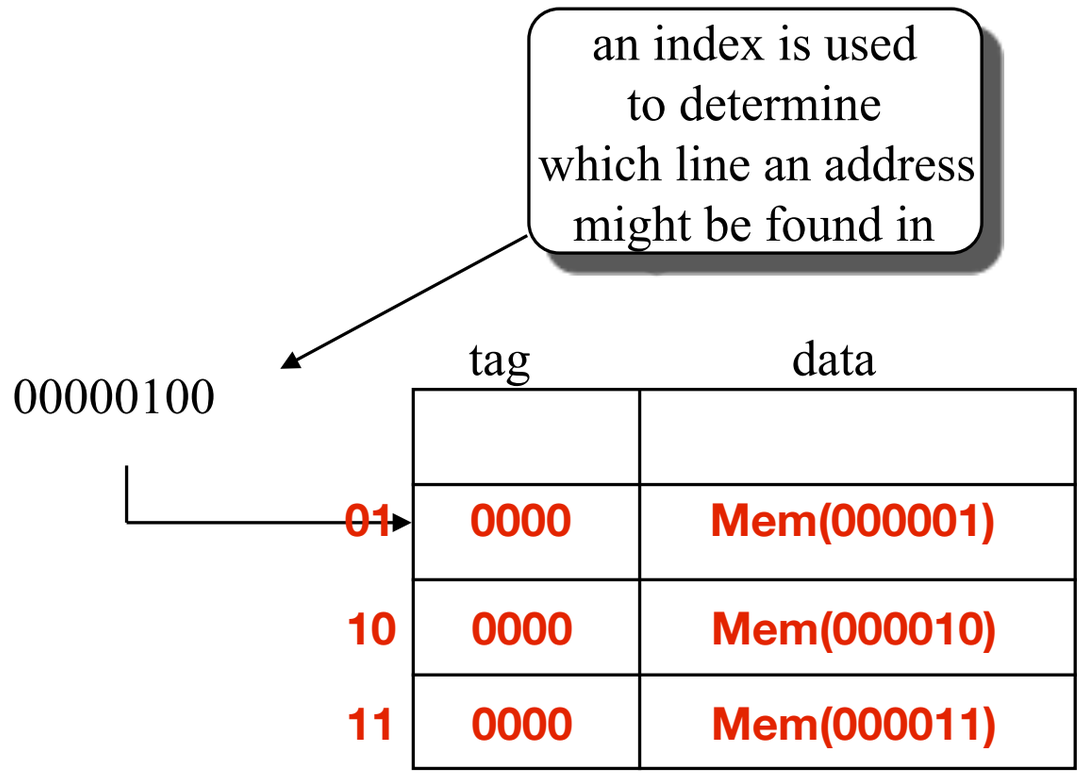
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



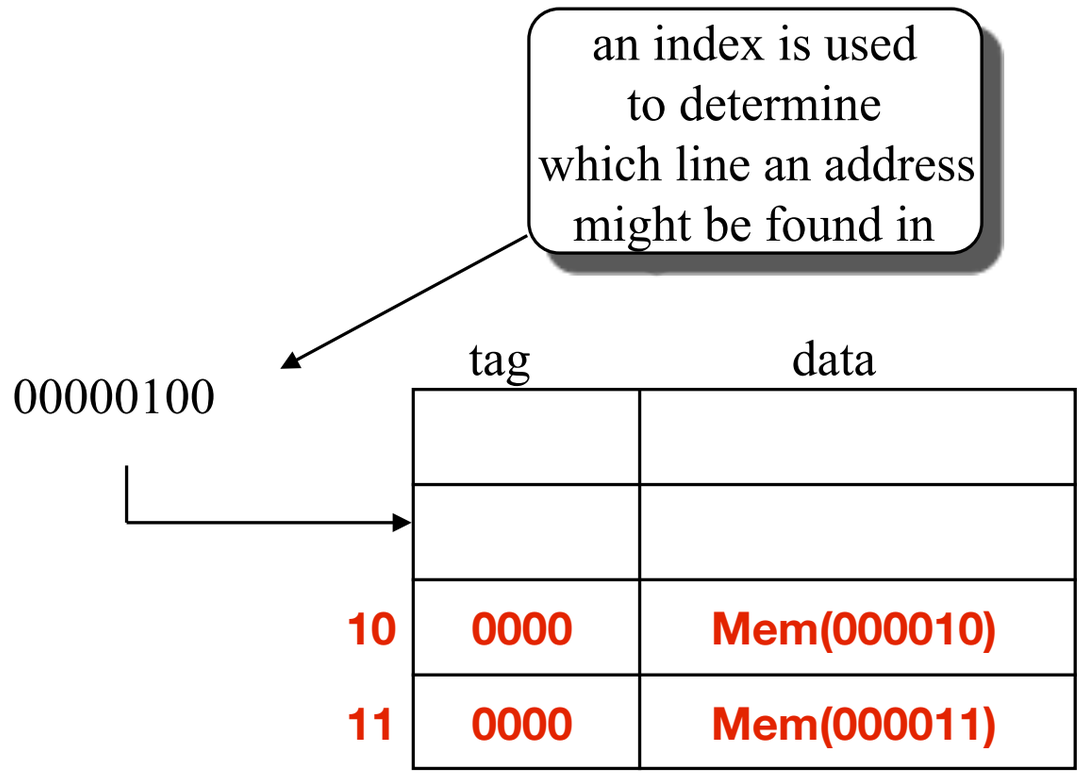
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



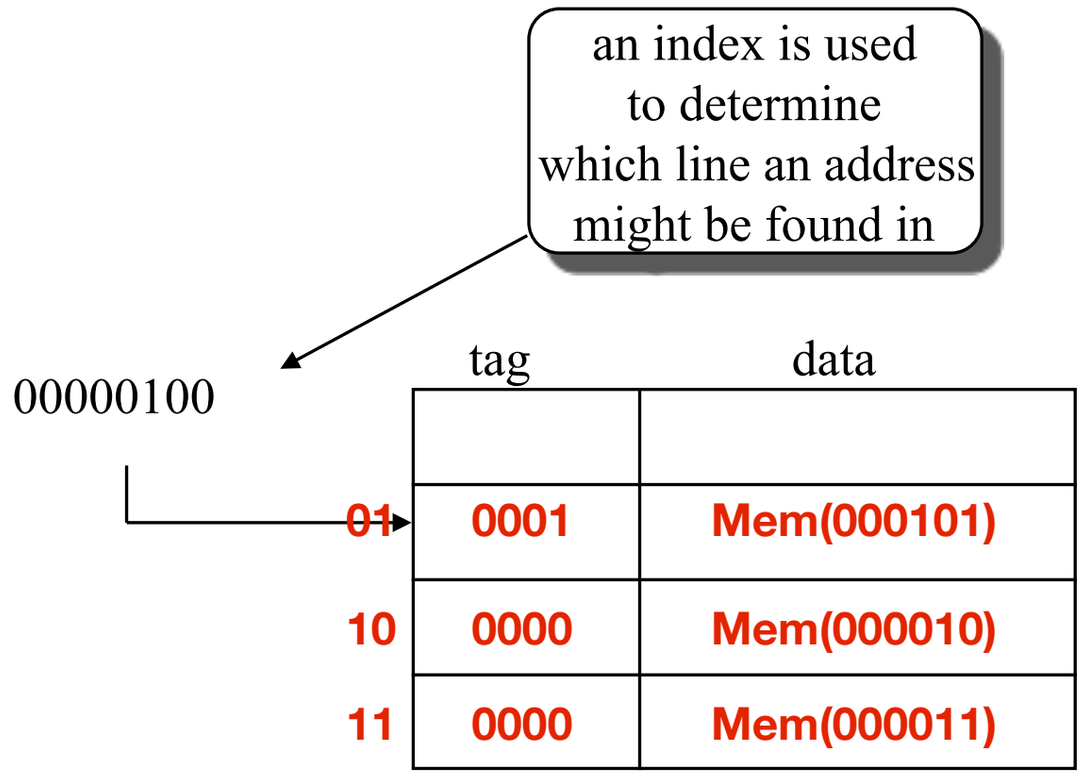
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

An Even Simpler Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



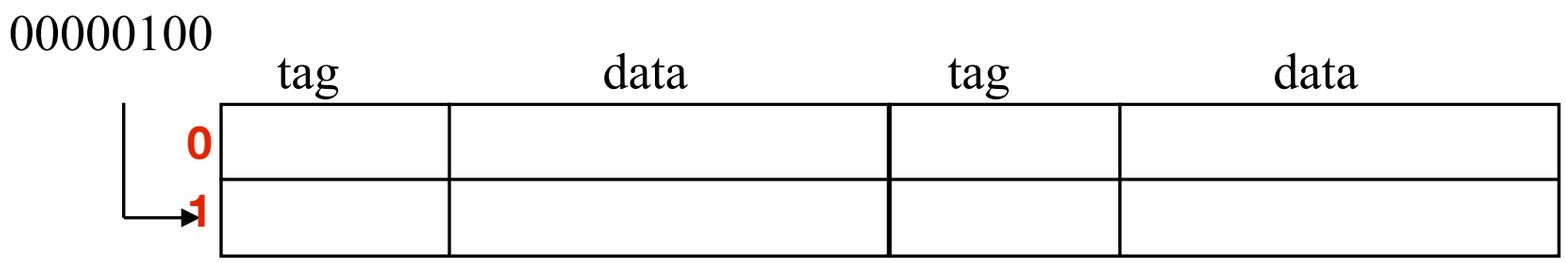
4 entries, each block holds one word, each word in memory maps to exactly one cache location.

- A cache that can put a line of data in exactly one place is called **Direct Mapped**.
- Advantages/disadvantages vs. fully-associative?

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



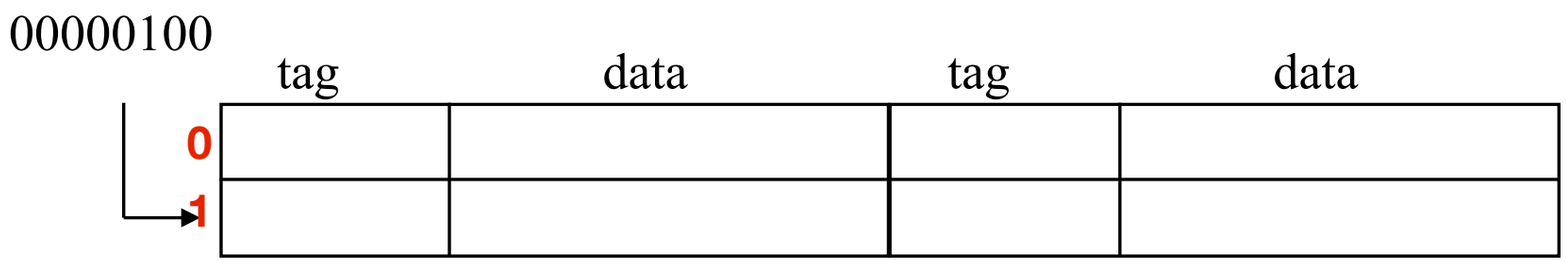
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called _____.
- The cache lines/blocks that share the same index are a cache _____.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



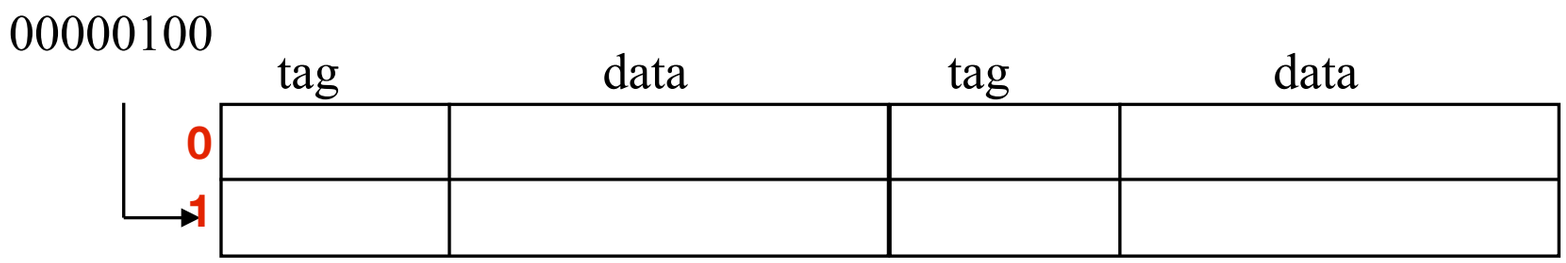
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache _____.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

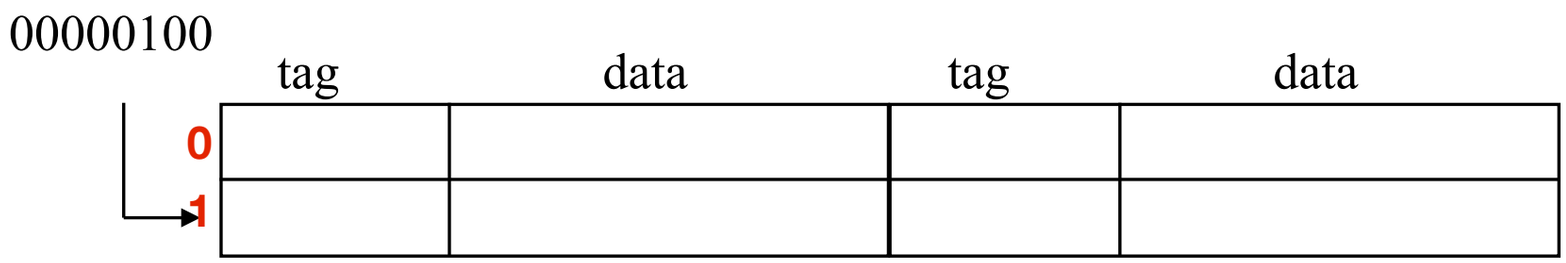


4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



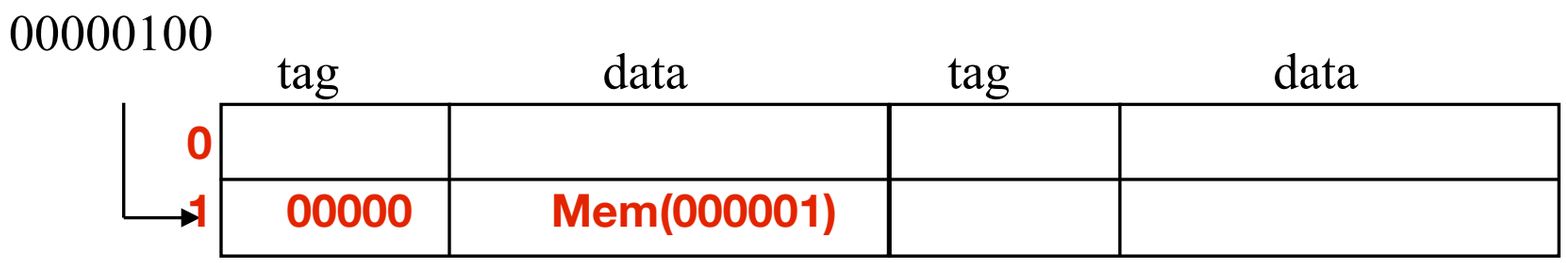
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



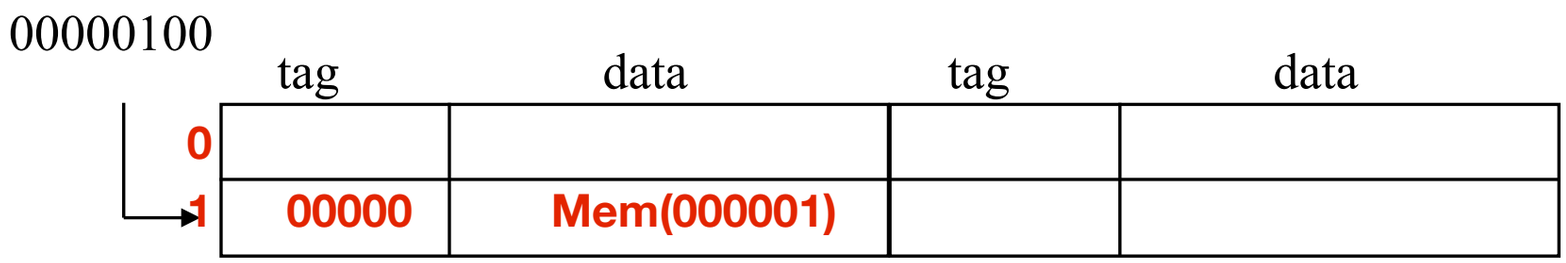
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



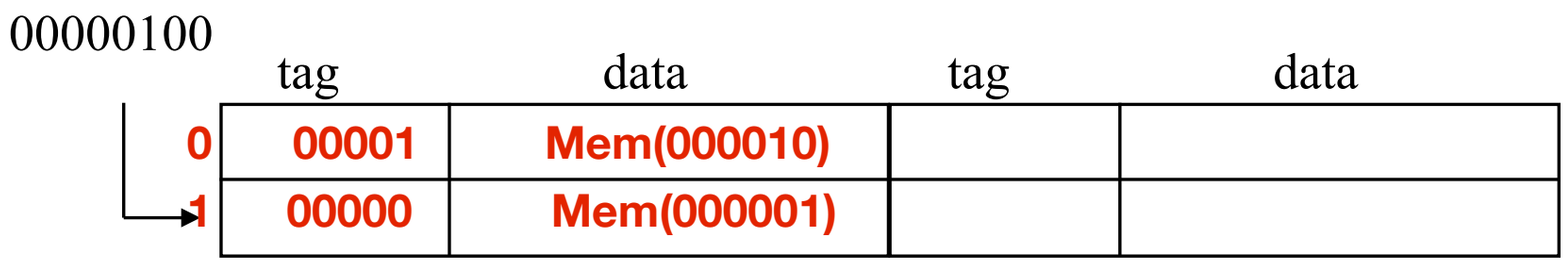
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



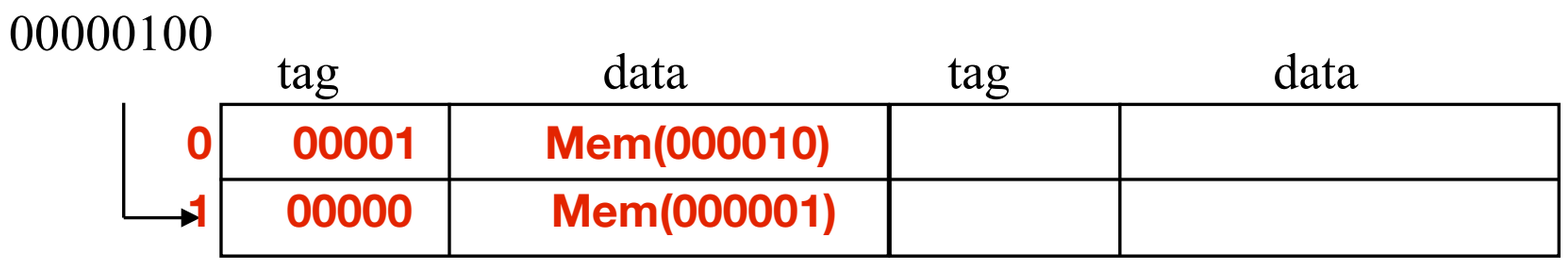
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



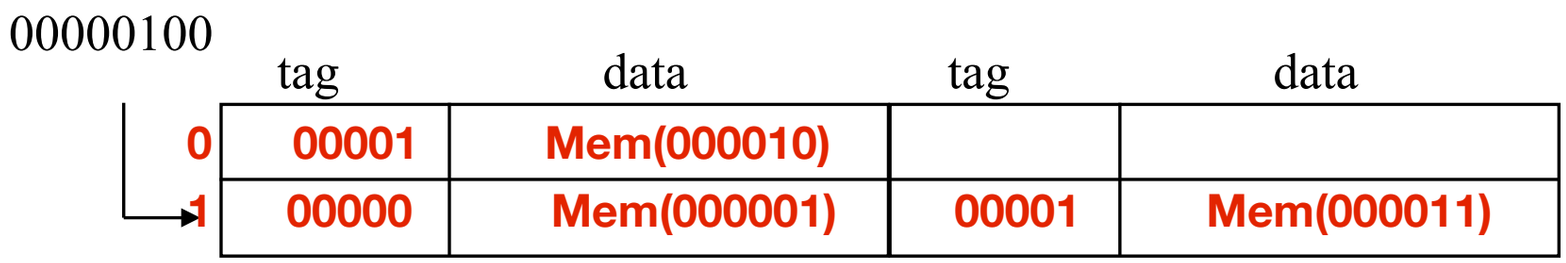
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



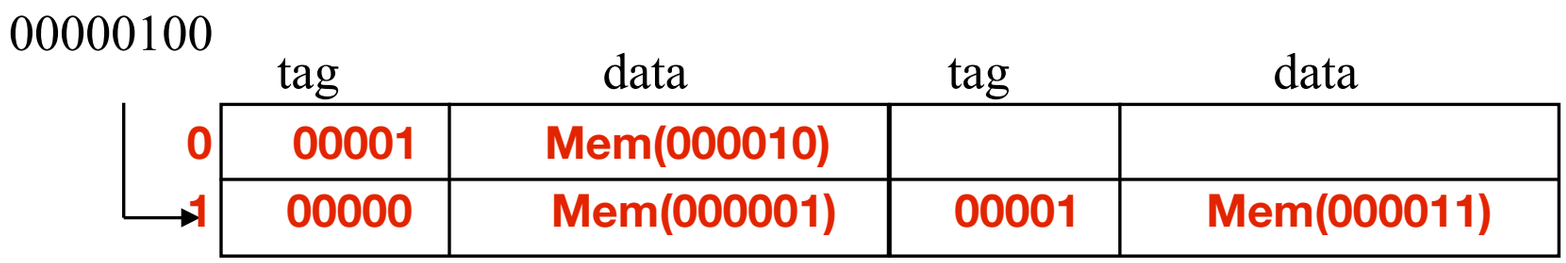
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



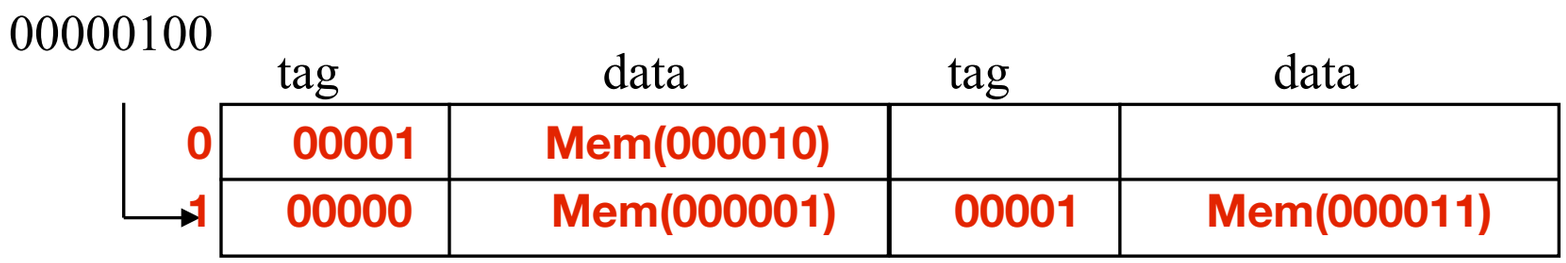
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



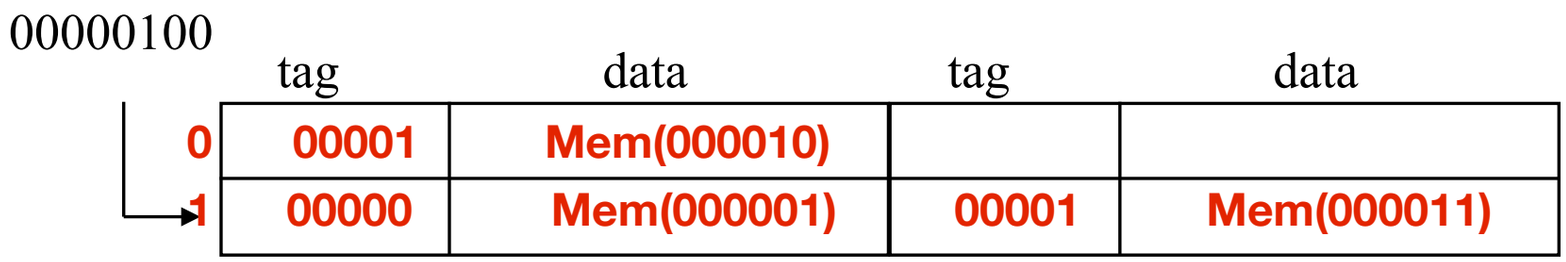
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



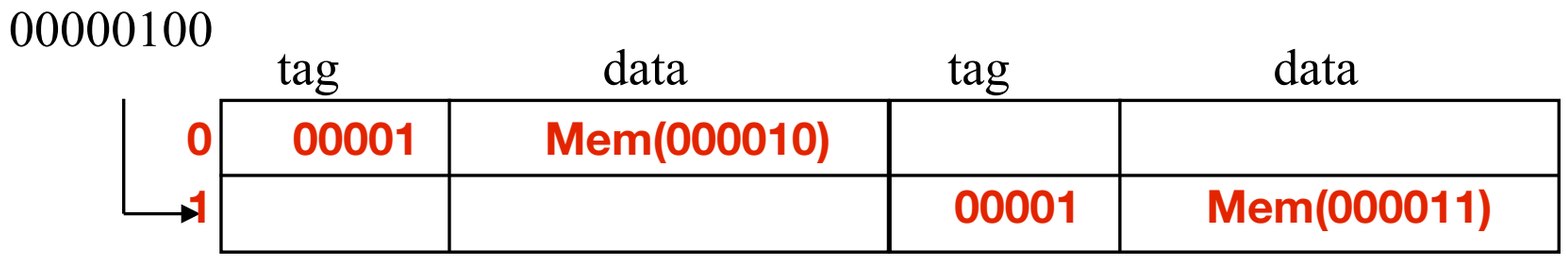
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



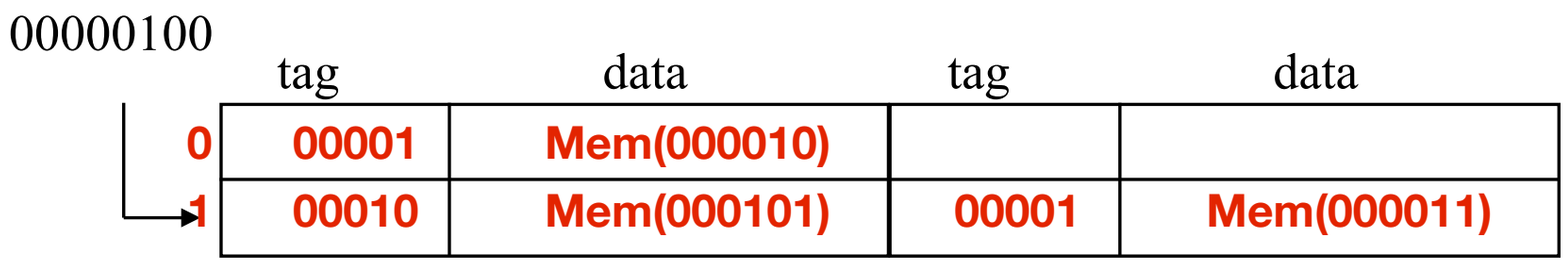
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

A Set Associative Cache

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



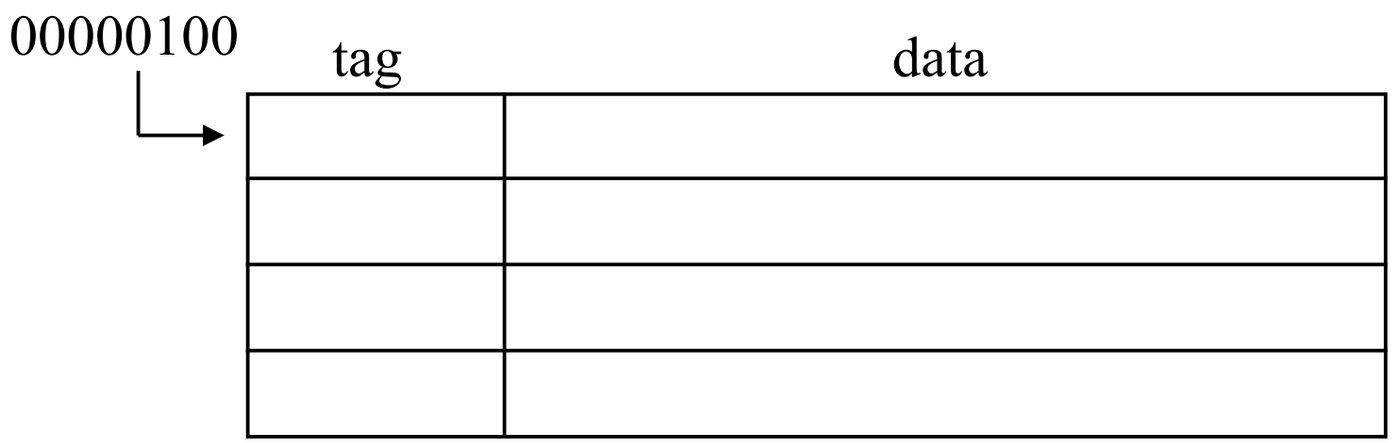
4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly n places is called **n-way set-associative**.
- The cache lines/blocks that share the same index are a cache **set**.

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



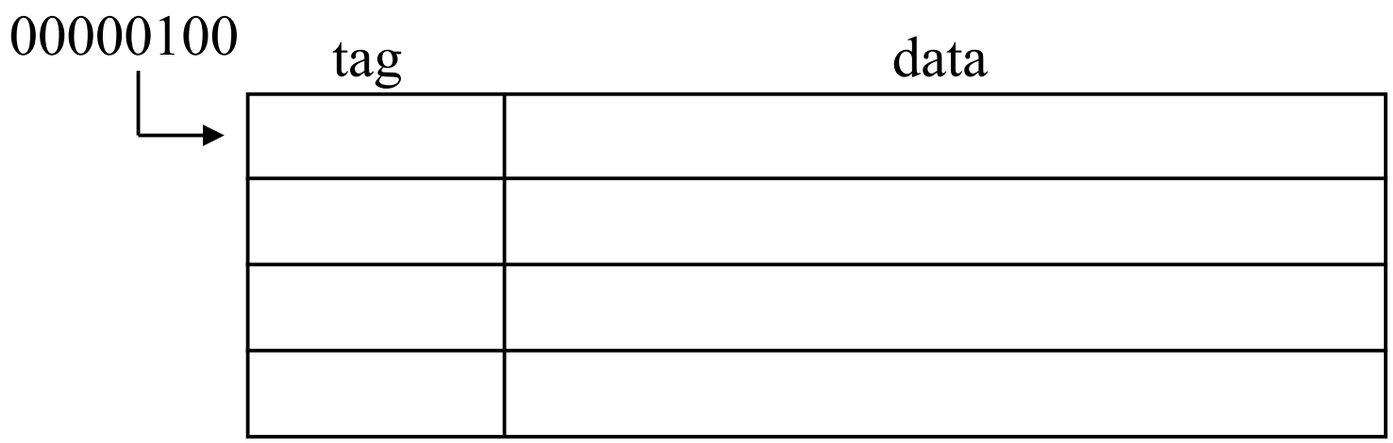
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of _____.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



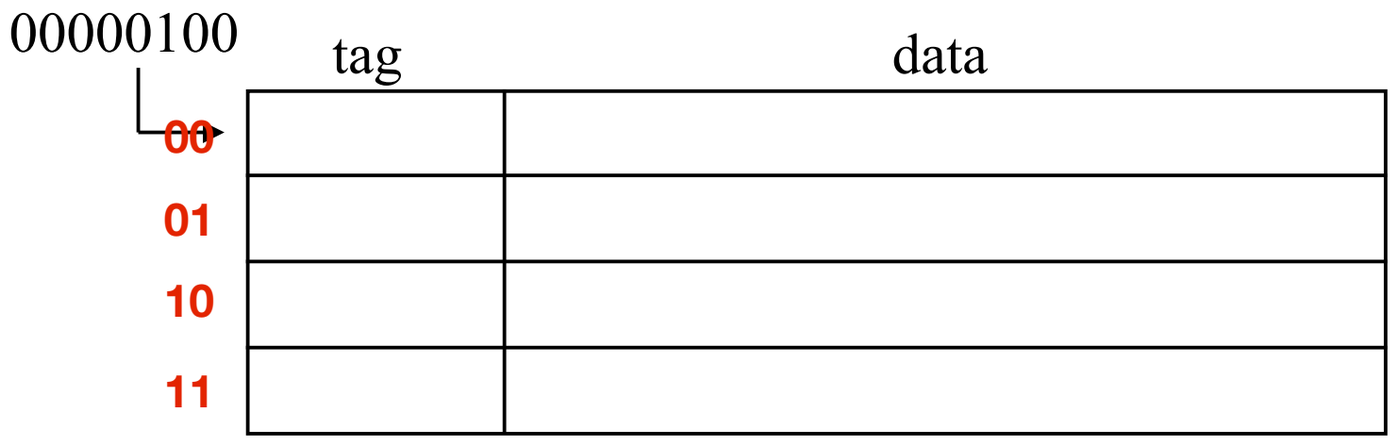
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of **Spacial Locality**.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



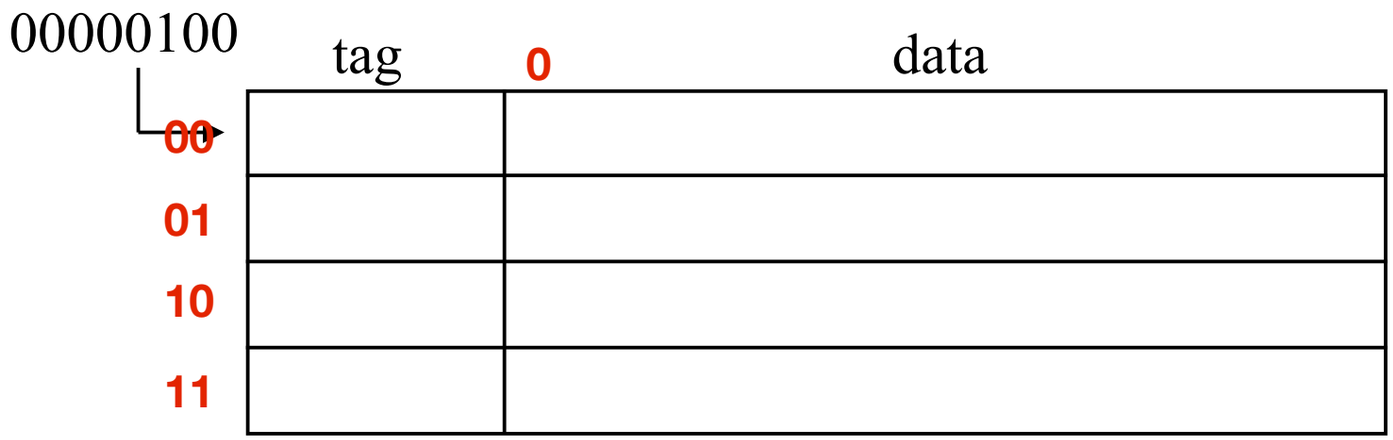
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of **Spacial Locality**.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



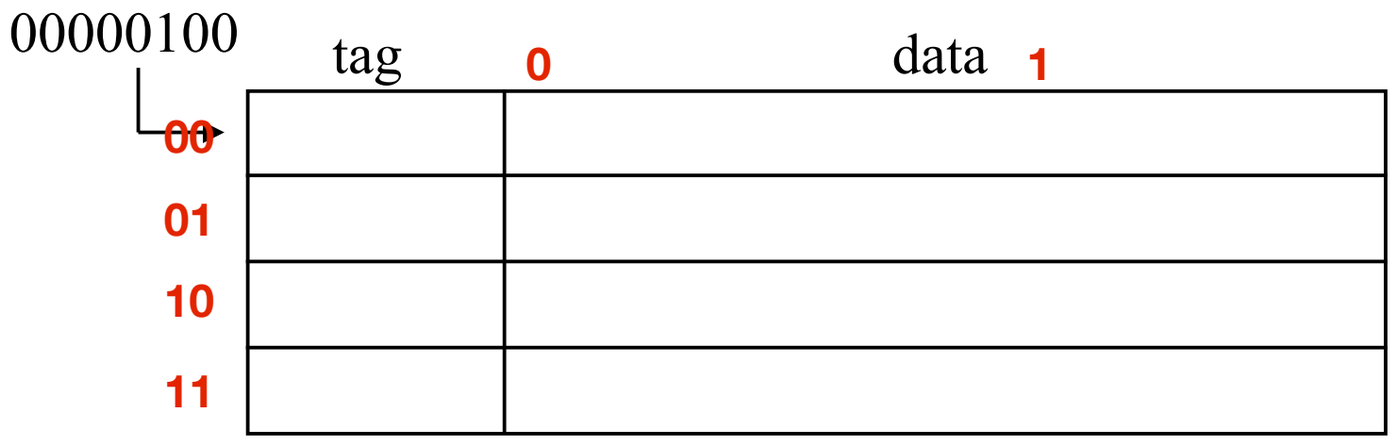
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of **Spacial Locality**.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



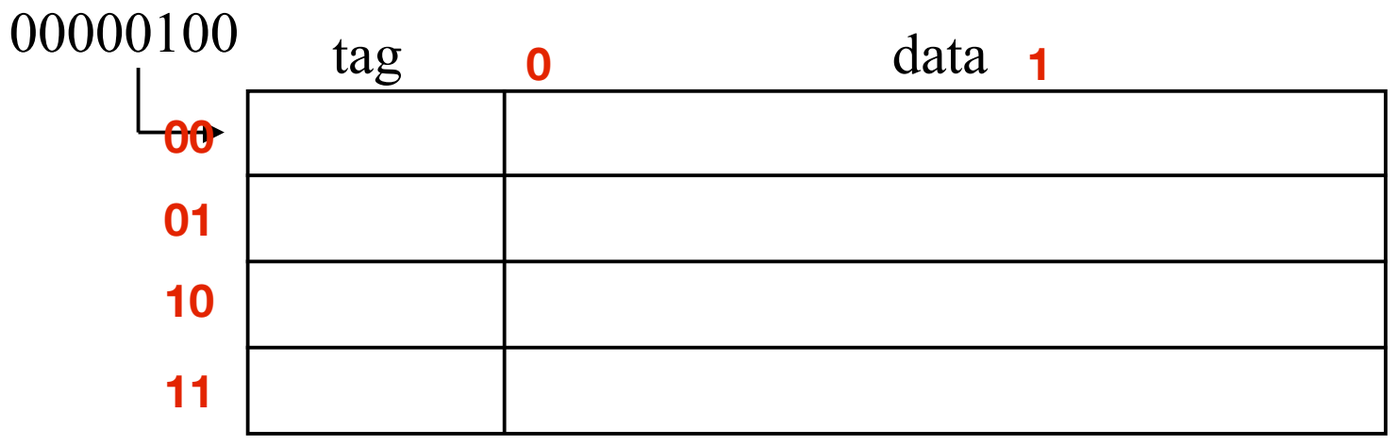
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of **Spacial Locality**.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



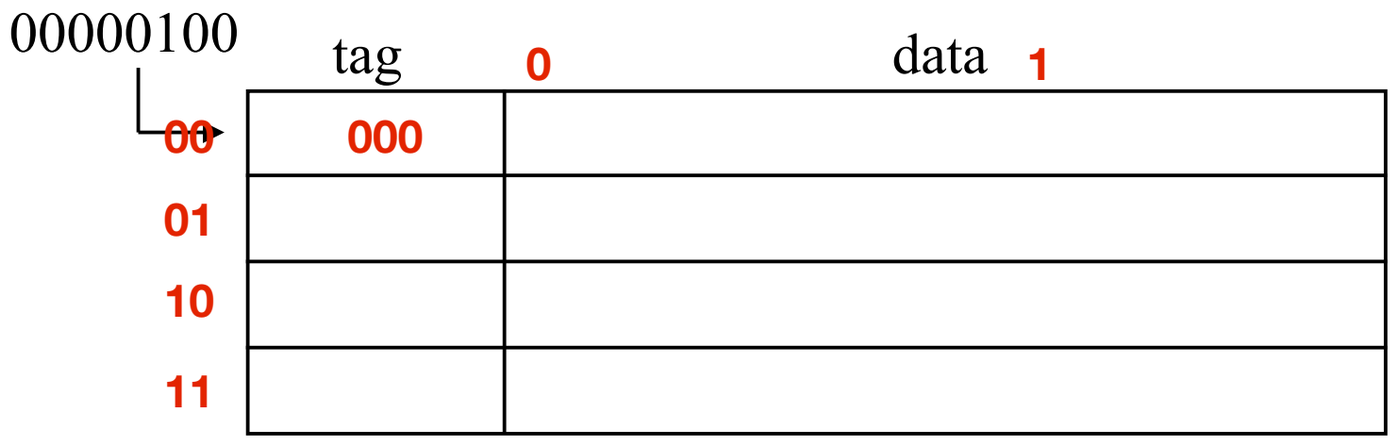
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of **Spacial Locality**.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



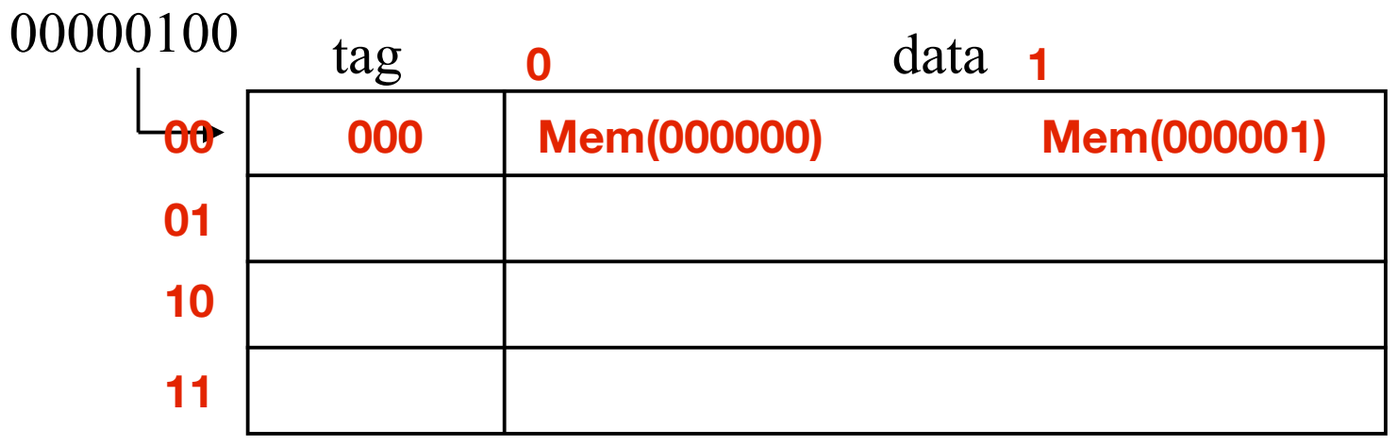
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of **Spacial Locality**.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



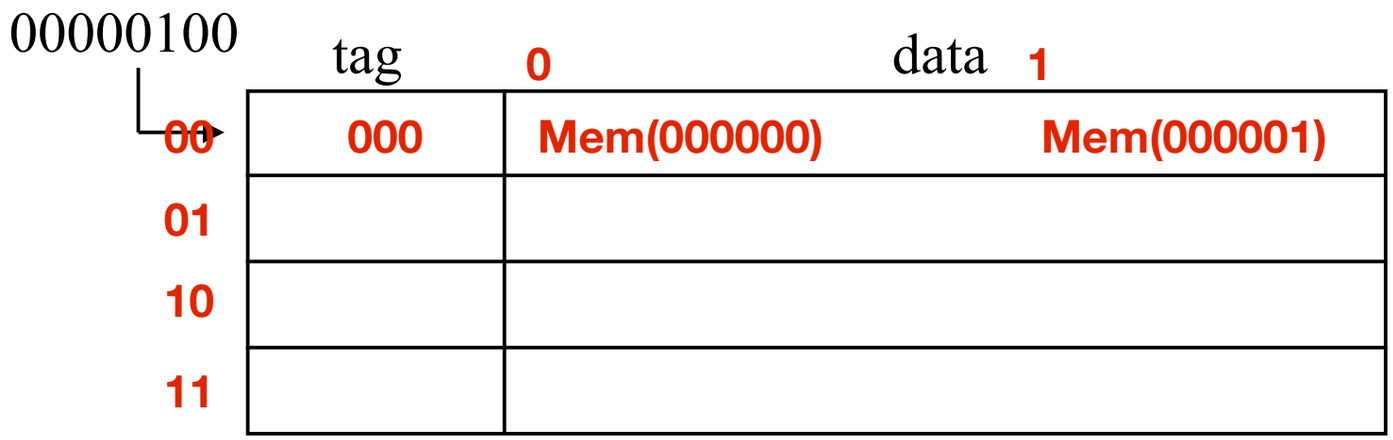
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of **Spacial Locality**.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



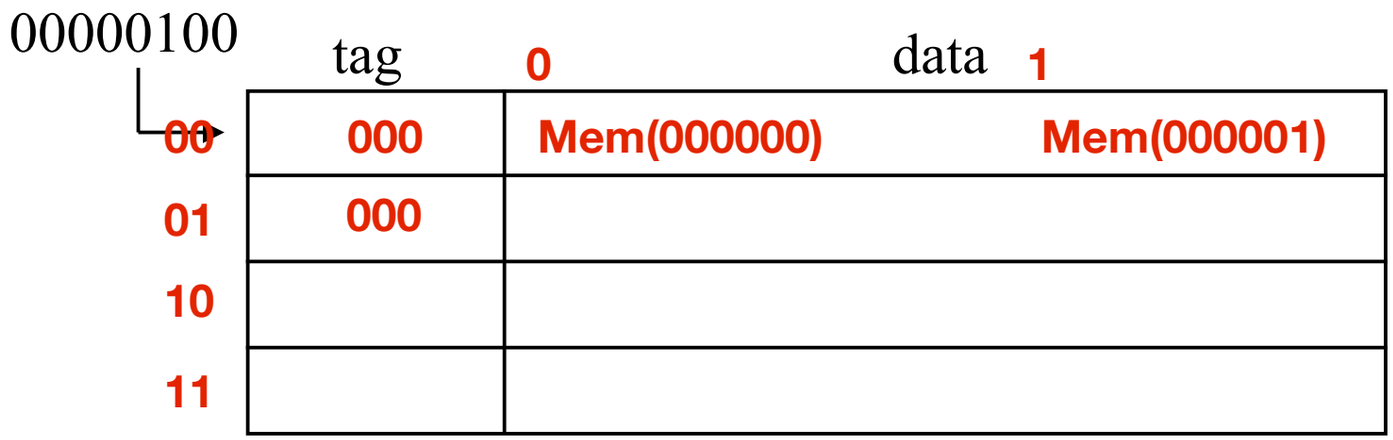
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of **Spacial Locality**.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



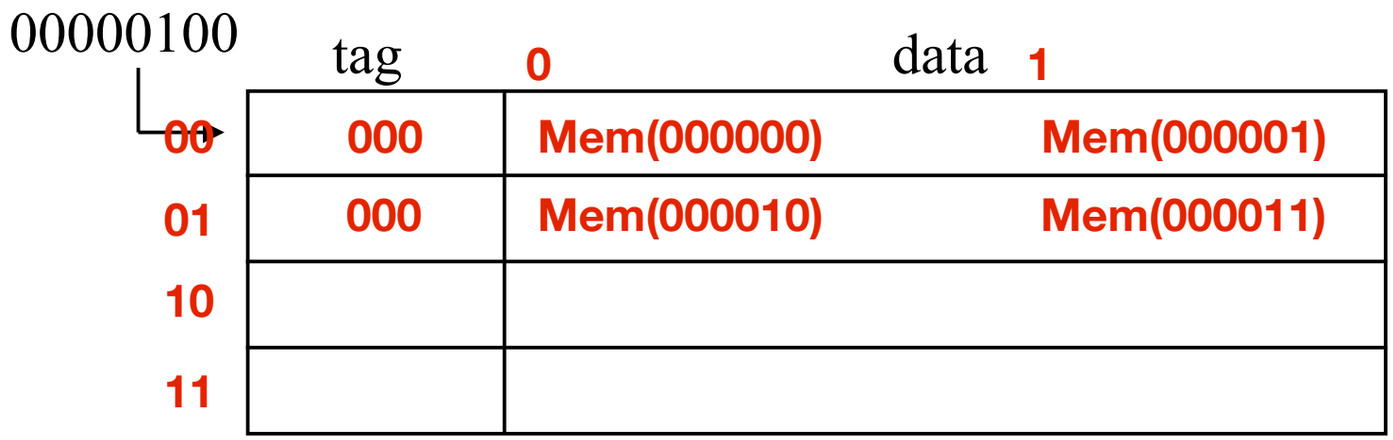
4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of **Spacial Locality**.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer/Larger Cache Blocks

address string:

4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of **Spacial Locality**.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Cache Parameters

Cache size = Number of sets * block size * associativity

-128 blocks, 32-byte block size, direct mapped, size =

-128 KB cache, 64-byte blocks, 512 sets, associativity = ?

Cache Parameters

Cache size = Number of sets * block size * associativity

-128 blocks, 32-byte block size, direct mapped, size =

128 x 32 = 4096 bytes = 4mb

-128 KB cache, 64-byte blocks, 512 sets, associativity = ?

Cache Parameters

Cache size = Number of sets * block size * associativity

-128 blocks, 32-byte block size, direct mapped, size =

128 x 32 = 4096 bytes = 4mb

-128 KB cache, 64-byte blocks, 512 sets, associativity = ?

131072 bytes / 512 = 256 bytes/set

Cache Parameters

Cache size = Number of sets * block size * associativity

-128 blocks, 32-byte block size, direct mapped, size =

128 x 32 = 4096 bytes = 4mb

-128 KB cache, 64-byte blocks, 512 sets, associativity = ?

131072 bytes / 512 = 256 bytes/set

256 bytes / 64 byte = 4 blocks/set = 4-way

Good Luck!