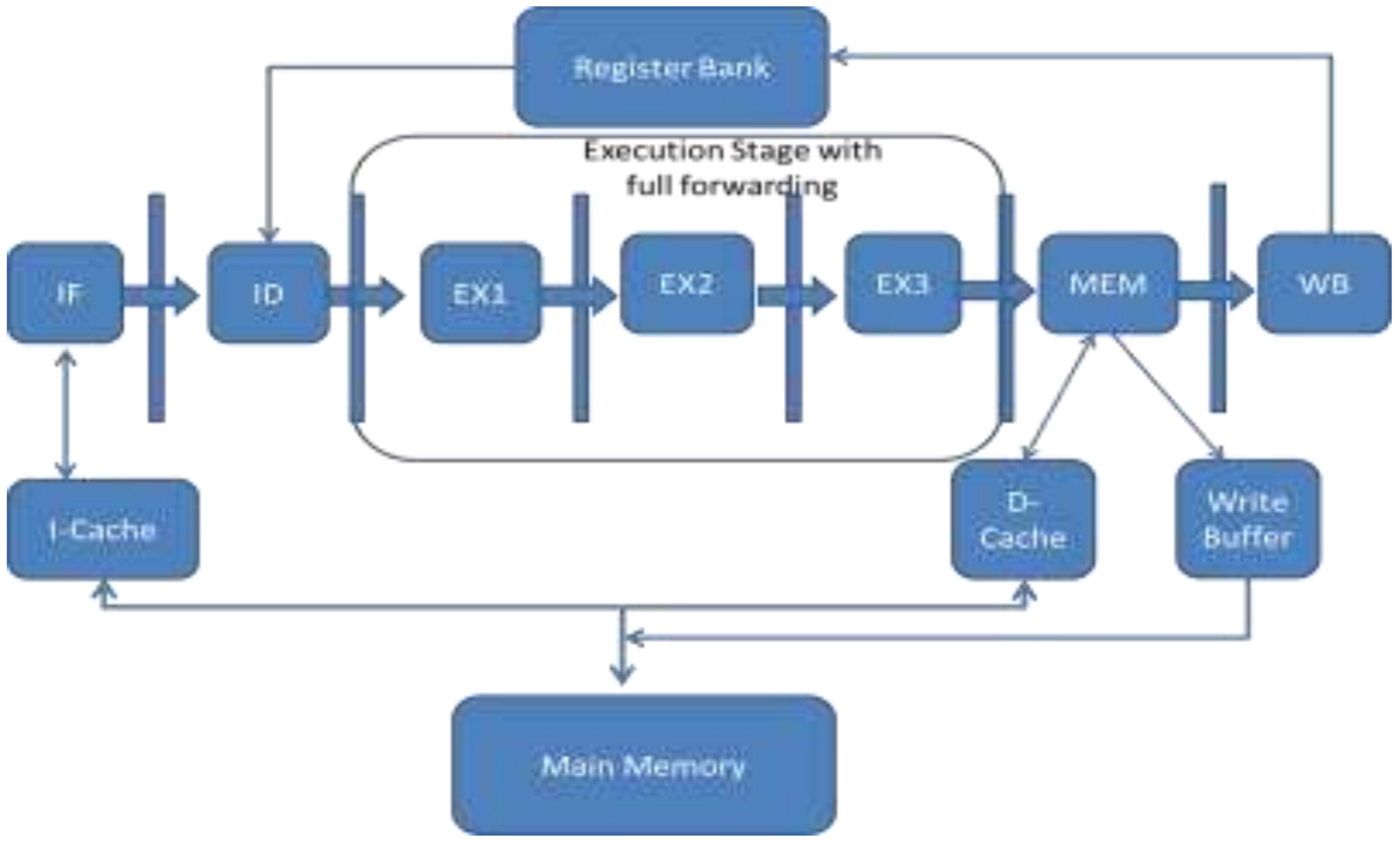
Project Tips

Design


Register Bank
Execution Stage with full forwarding
IF
ID
EX1
EX2
EX3
MEM
WB
I-Cache
D-Cache
Write Buffer
Main Memory

Instruction Fetch (IF) Stage

- First tries to fetch the instruction from instruction cache
- If the instruction is not in instruction cache, the instruction needs to be brought into the instruction cache from memory. After the instruction is brought into the instruction cache, then the instruction can be fetched from the instruction cache

Instruction Cache

- Cache can store 4 blocks; each block is four words
  - Cache can store up to 16 words; each instruction is a word implies instruction cache can fit up to 16 instructions

- Each memory fetch brings four contiguous words (a block) from memory into instruction cache
  - Each word access from memory is 3 cycles => access 4 words to bring into cache takes 12 cycles

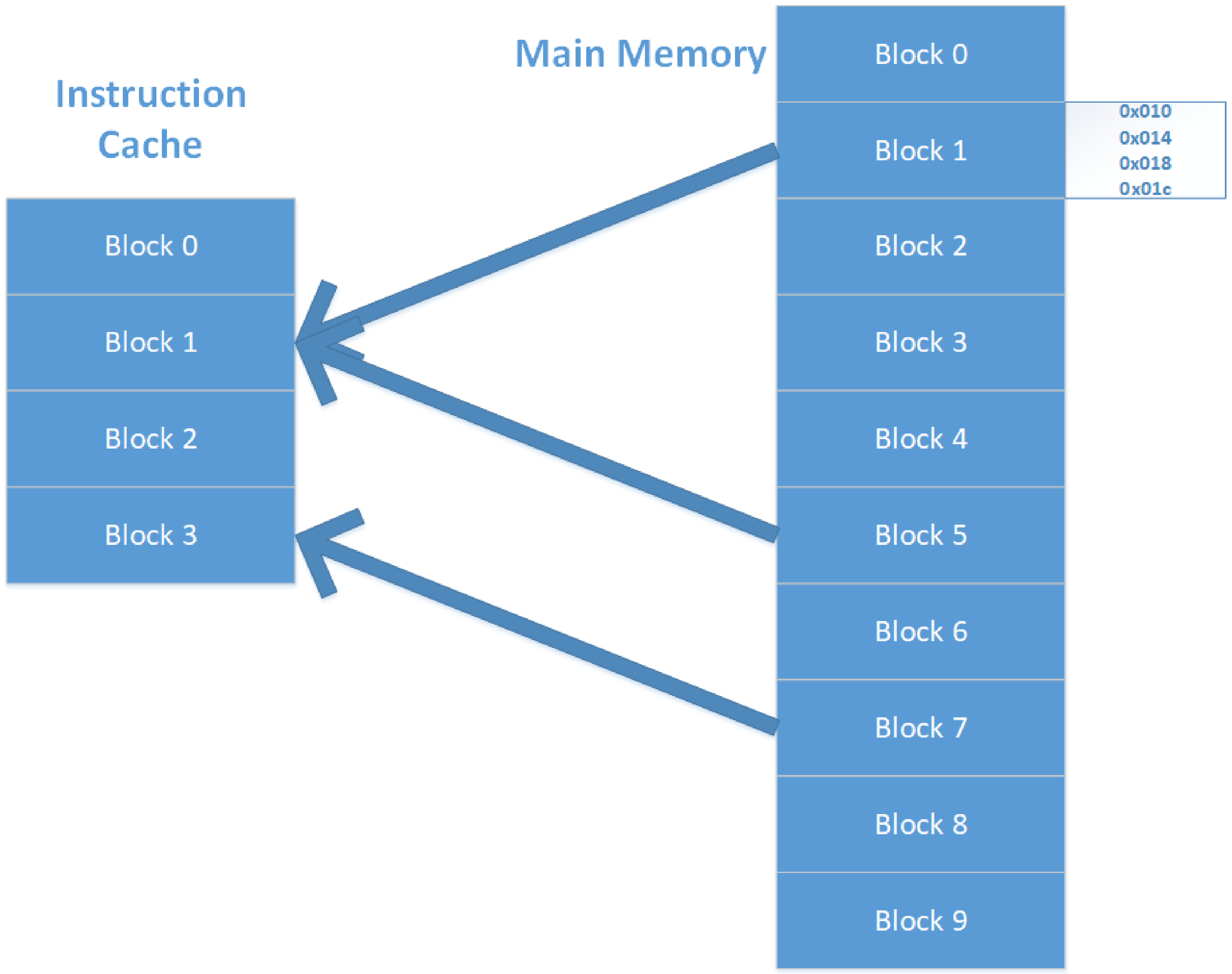
Instruction Cache

- Direct mapped
  - A memory block can only map to a single index in the cache
  - Cache_block_number = main_memory_block_number % 4
  - Main_memory_word_number = address / 4
  - Main_memory_block_number = main_memory_word_number / 4

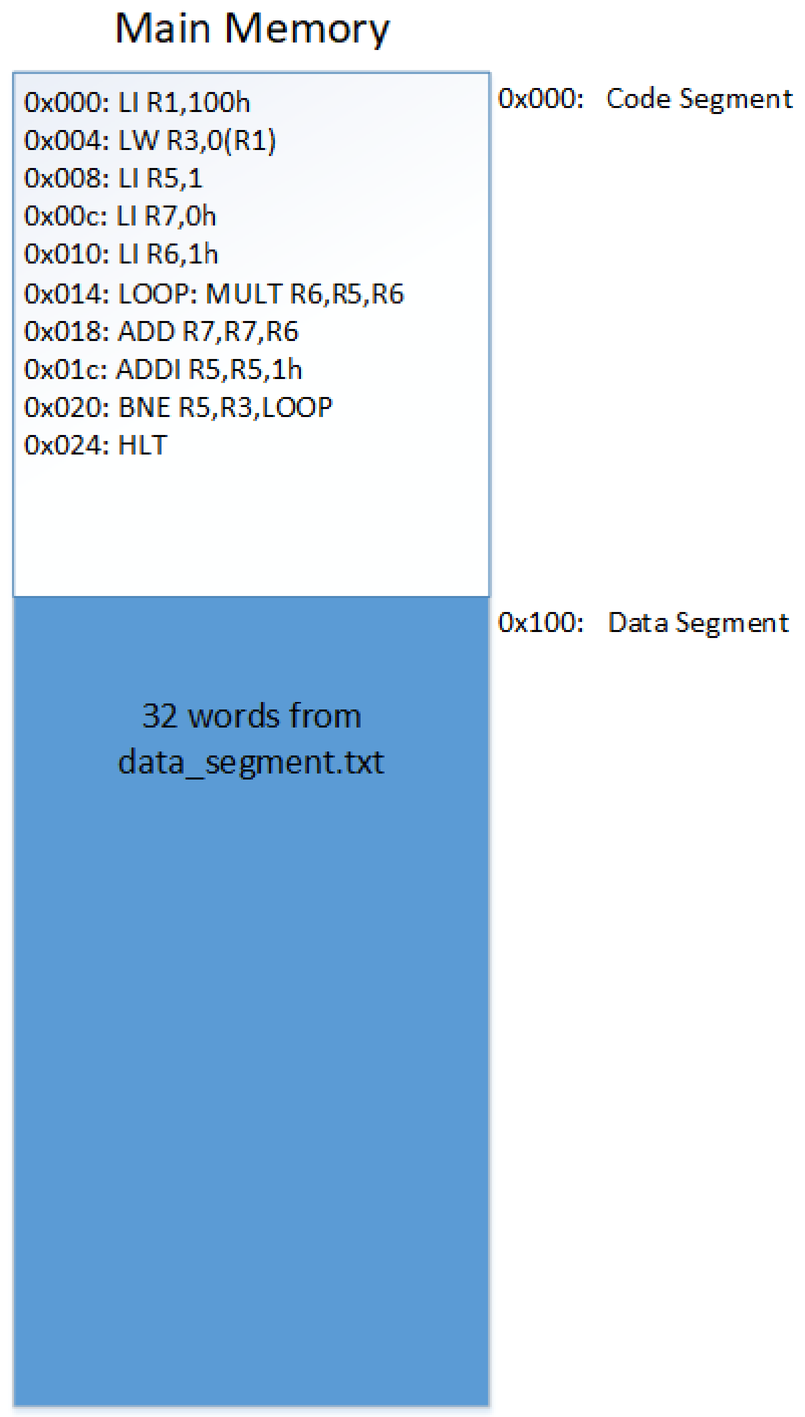
- E.g. instruction at address 0x014 (20 decimal)
  - Main_memory_word_number = 20/4 = 5
  - Main_memory_block_number = 5/4 = 1 (round down)
  - Cache_block_number = 1%4 = 1

  - If instruction at address 0x014 needs to be brought into memory, then it along with the other 3 instructions in its main memory block will be brought into cache block 1

Instruction Cache



Main Memory Layout



```
0x000: LI R1,100h
0x004: LW R3,0(R1)
0x008: LI R5,1
0x00c: LI R7,0h
0x010: LI R6,1h
0x014: LOOP: MULT R6,R5,R6
0x018: ADD R7,R7,R6
0x01c: ADDI R5,R5,1h
0x020: BNE R5,R3,LOOP
0x024: HLT
```

Instruction Decode (ID) Stage

- Determine what operation to perform
- Determine operands
- For Beq and Bne, compute if branch should or should not be taken

Execute (Ex) Stage

- Execute instruction

Memory (Mem) Stage

- For load instructions, fetch data from cache and update cache block timestamp.
  - If data is not in cache, must first fetch data from memory and bring the main memory block containing the data into data cache.
  - If the “dirty bit” for the cache block containing the data is set to 1, must first flush the write buffer and then bring the main memory block containing the data into data cache

- For store instructions, write data to write buffer
  - If the write address of the data is in the data cache, must also set the “dirty bit” of the data cache block containing that write address to 1
    - Setting the “dirty bit” is how the cache knows that it no longer has up to date data

Data Cache

- 2-way set associative
  - A memory block can map to two indexes in the cache
  - If the main memory block number is even, it can be placed in data cache block 0 or 2
  - If the main memory block number is odd, it can be placed in data cache block 1 or 3

- Least Recently used policy
  - A main memory block has two possible data cache blocks it can be stored in. If both data cache blocks are occupied, whichever data cache block has the oldest timestamp will be replaced

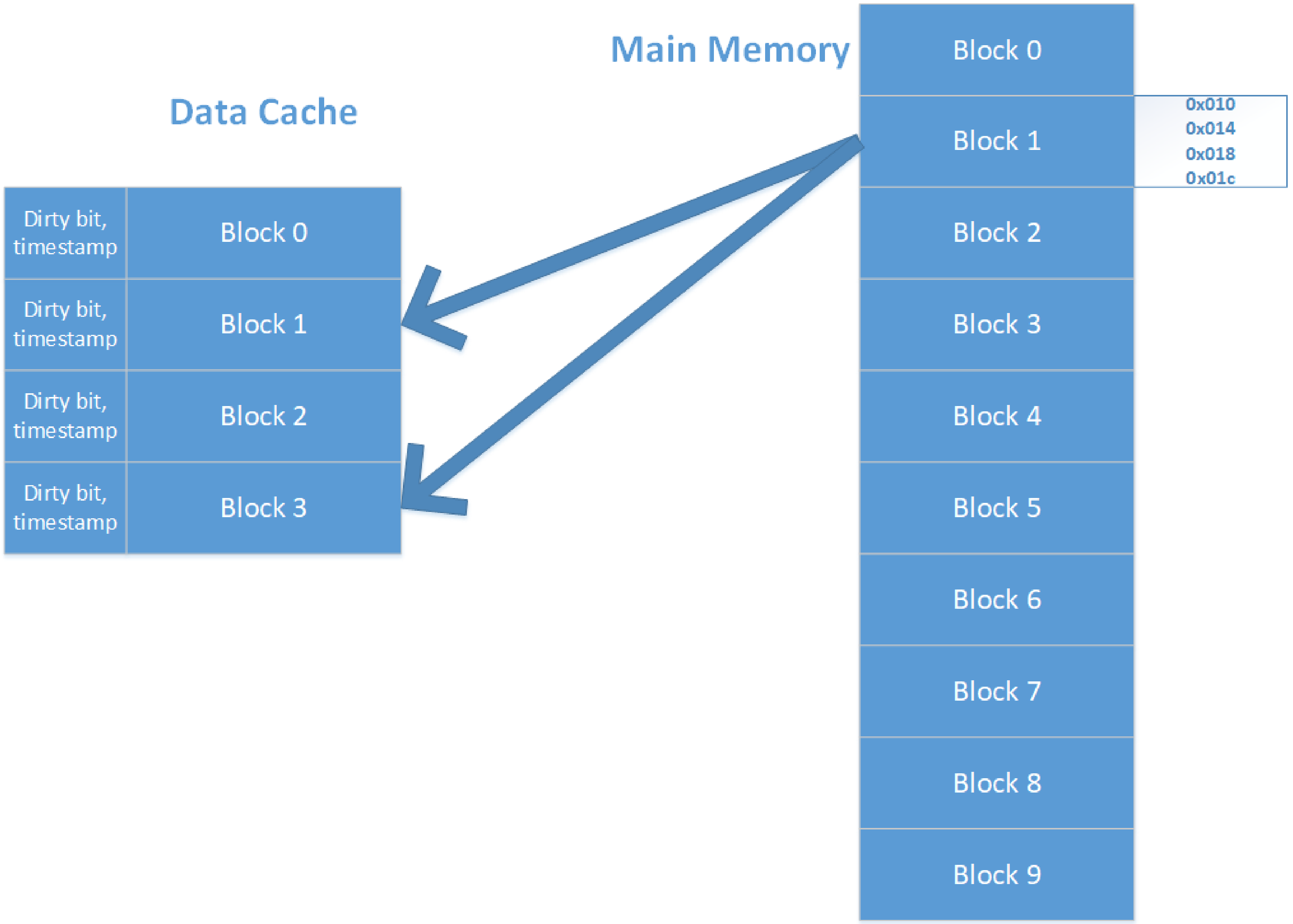
Data Cache

- E.g. Data at address 0x014 (20 decimal)
  - Main_memory_word_number = 20/4 = 5
  - Main_memory_block_number = 5/4 = 1 (round down)

- Main memory block number 1 is odd so it can be mapped to data cache block 1 or 3

- E.g. Data cache block 1 is currently occupied and has a timestamp of 2 pm for last access and data cache block 3 is currently occupied and has a timestamp of 1 pm for last access.
  - Data cache block 3 is least recently used so it will be replace by main memory block 1

Data Cache


Main Memory
Data Cache
Dirty bit, timestamp
Block 0
Dirty bit, timestamp
Block 1
Dirty bit, timestamp
Block 2
Dirty bit, timestamp
Block 3
Block 0
Block 1
0x010
0x014
0x018
0x01c
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
Block 8
Block 9

Write Back (WB) Stage

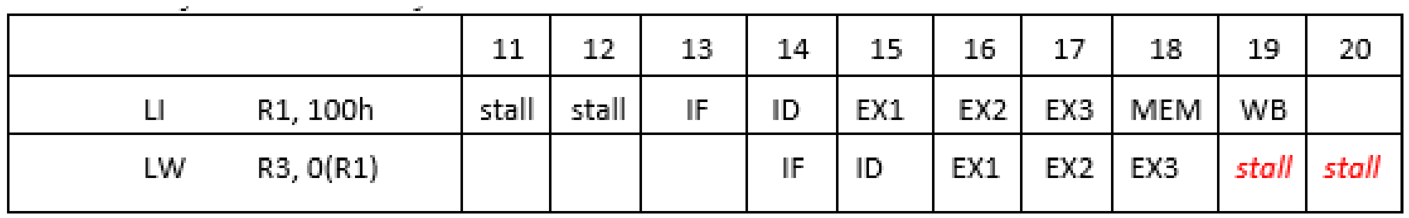
- Write the result of the operation back to the register

Reasons for stalls

- Data hazards
  - An operand of an operation needs the result of a previous operation
  - E.g. from project sample input

```
LOOP:   MULT        R6, R5, R6
        ADD         R7, R7, R6
```

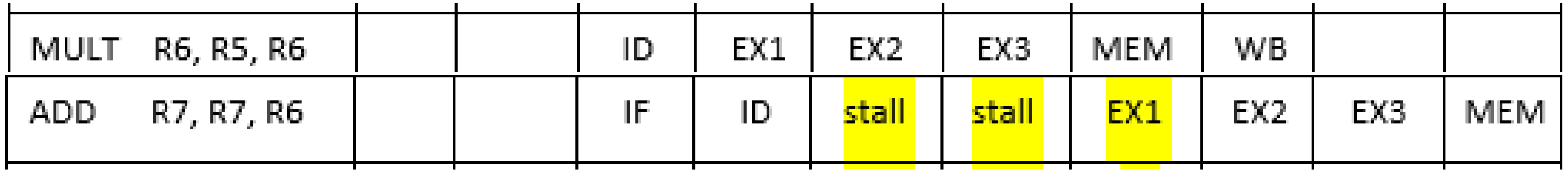
- Structural hazards
  - An operation needs a resource that is in use by a previous operation
  - E.g. from project document: lw needs to wait until li finishes it’s IF stage before it can start it’s IF stage



		11	12	13	14	15	16	17	18	19	20
LI	R1, 100h	stall	stall	IF	ID	EX1	EX2	EX3	MEM	WB	
LW	R3, 0(R1)				IF	ID	EX1	EX2	EX3	*stall*	*stall*

Data forwarding

- To alleviate data hazards, data may be forwarded from a stage earlier than write back to a stage in the next instruction that needs the data
  - Without data forwarding, to use the updated value of R6 in the execute stage of the add instruction we would have to wait until the multiply instruction finished it’s write back stage
- For this design, Data forwarding is possible from the MEM stage to the EX 1 & ID stages, from the EX 1, 2, 3 stages to ID stage, from EX 2, 3 to EX 1
  - Multiply finishes after execute stage 3, and can forward to execute stage 1 of add



MULT R6, R5, R6			ID	EX1	EX2	EX3	MEM	WB		
ADD R7, R7, R6			IF	ID	stall	stall	EX1	EX2	EX3	MEM