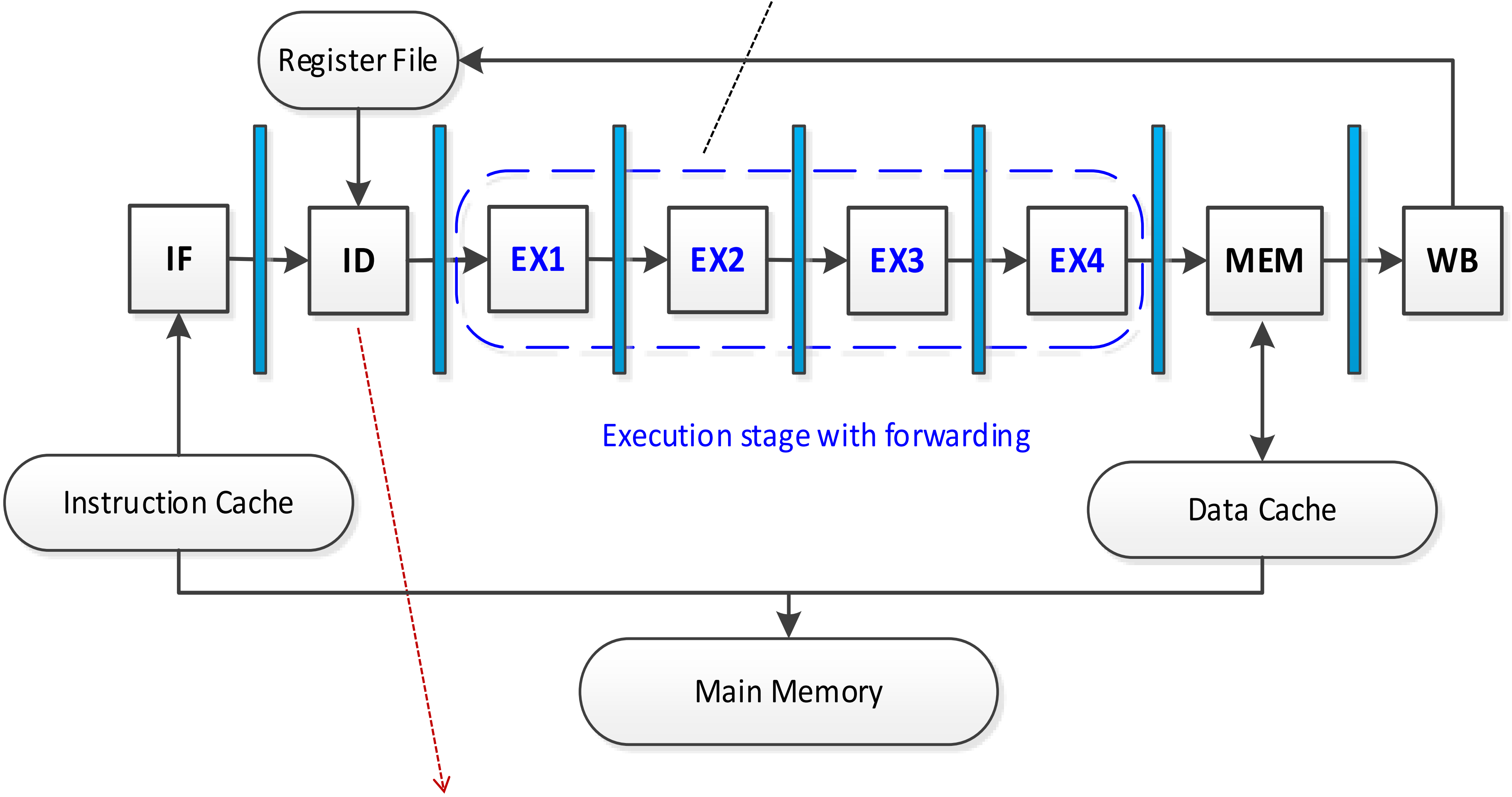
Forwarding

- MEM ➔ the EX 1 & ID stages
- EX 1, 2, 3, 4 ➔ ID stage
- EX 2, 3, 4 ➔ EX 1 stage



- Branches are resolved in the ID stage
- “not-taken prediction” will be used in IF stage

Instruction Class	Instruction Mnemonic
Data Transfers	LW, SW
Arithmetic/ logical	ADD, ADDI, MULT, MULTI, SUB, SUBI, AND, ANDI, OR, ORI, LI, LUI
Control	BEQ, BNE, J
Special purpose	HLT (to halt the simulation)

The table below shows the number of cycles each instruction takes in the EX stage.

Number of Cycles	Instructions
0 Cycle	J, BEQ, BNE, LI, LUI
1 Cycle	AND, ANDI, OR, ORI, LW, SW
2 Cycles	ADD, ADDI, SUB, SUBI
4 Cycles	MULT, MULTI

Additional Features & Assumptions

- Instructions and data are stored in memory starting at address 0x0 and 0x100, respectively.
- Load and store instructions use word-aligned addresses when accessing data.
- Both conditional and unconditional jump instructions can be forward and backward. You can assume that a program will not create a closed loop.
- An instruction stalled for data hazard in the ID stage can get the values in the same cycle WB takes place.
- The HLT instruction will mark the end of the program, i.e., fetching will seize as soon as the HLT instruction is decoded. In your implementation you can assume that the program will have two HLT instructions at the end in order to stop accessing the cache once the first HLT reaches the ID stage, i.e., the second HLT instruction will be terminated at that time.

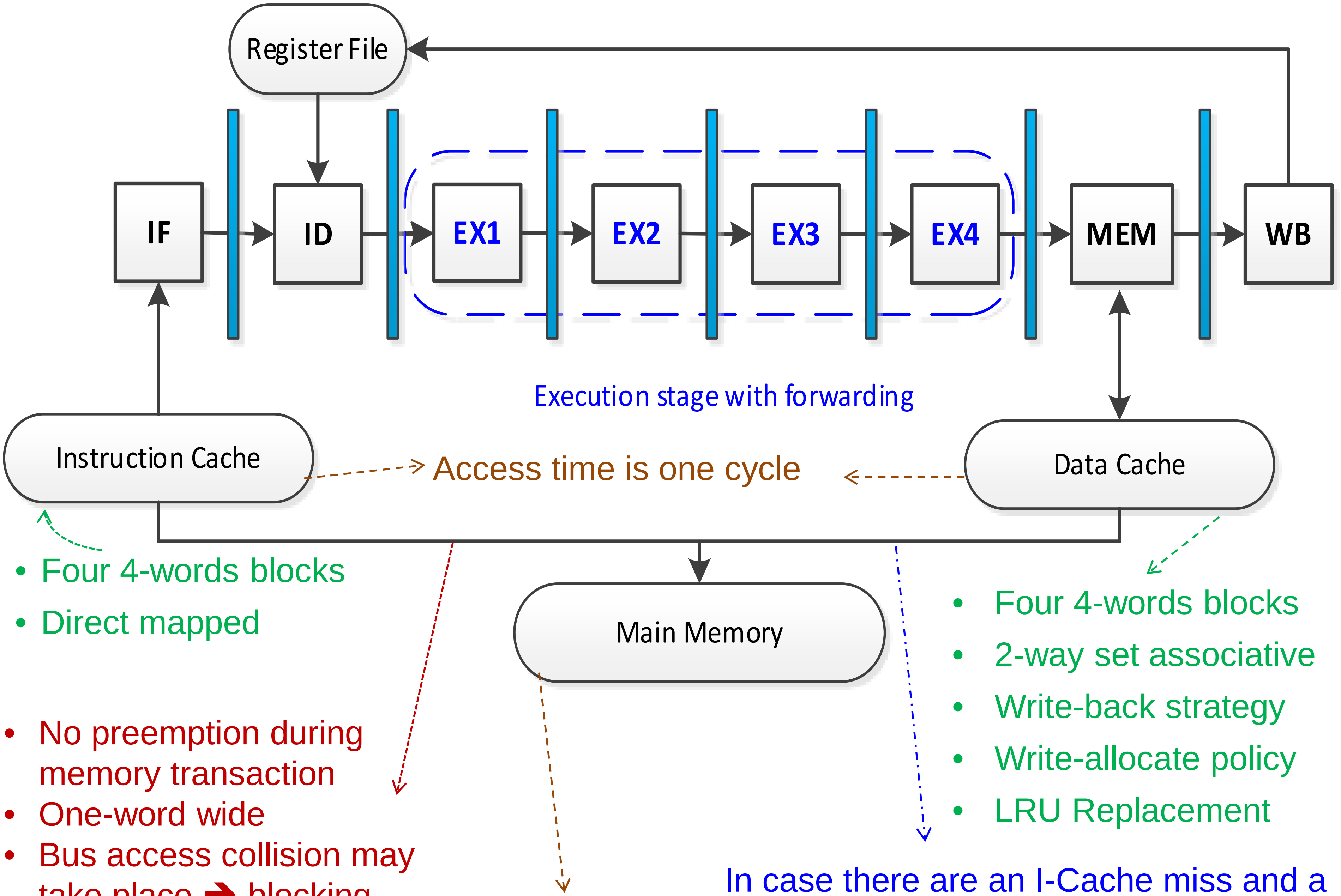
Example

```
        LI      R1, 100h            # addr = 0x100;
        LW      R3, 0(R1)           # boundary = *addr;
        LI      R5, 1               # i = 1;
        LI      R7, 0h              # sum = 0;
        LI      R6, 1h              # factorial = 0x01;
LOOP:   MULT    R6, R5, R6          # factorial *= I;
        ADD     R7, R7, R6          # sum += factorial;
        ADDI    R5, R5, 1h          # i++;
        BNE     R5, R3, LOOP
        HLT
        HLT
```

Example: Without Memory Hierarchy

			1	2	3	4	5	6	7	8	9	10	11	12	13	14
	LI	R1, 100h	IF	ID	EX1	EX2	EX3	EX4	ME	WB						
	LW	R3, 0(R1)		IF	ID	EX1	EX2	EX3	EX4	ME	WB					
	LI	R5, 1			IF	ID	EX1	EX2	EX3	EX4	ME	WB				
	LI	R7, 0h				IF	ID	EX1	EX2	EX3	EX4	ME	WB			
	LI	R6, 1h					IF	ID	EX1	EX2	EX3	EX4	ME	WB		
Loop:	MULT	R6, R5, R6						IF	ID	EX1	EX2	EX3	EX4	ME	WB	
	ADD	R7, R7, R6							IF	ID	stall	stall	stall	EX1	EX2	EX3
	ADDI	R5, R5, 1h														
	BNE	R5, R3, Loop														
	HLT															

			7	8	9	10	11	12	13	14	15	16	17	18	19	20
	LI	R1, 100h	ME	WB												
	LW	R3, 0(R1)	EX4	ME	WB											
	LI	R5, 1	EX3	EX4	ME	WB										
	LI	R7, 0h	EX2	EX3	EX4	ME	WB									
	LI	R6, 1h	EX1	EX2	EX3	EX4	ME	WB								
Loop:	MULT	R6, R5, R6	ID	EX1	EX2	EX3	EX4	ME	WB							
	ADD	R7, R7, R6	IF	ID	stall	stall	stall	EX1	EX2	EX3	EX4	ME	WB			
	ADDI	R5, R5, 1h		IF	stall	stall	stall	ID	EX1	EX2	EX3	EX4	ME	WB		
	BNE	R5, R3, Loop			stall	stall	stall	IF	ID	stall	ID					
	HLT								IF	stall	stall	ID				
	HLT											IF				



Access time is one cycle

- Four 4-words blocks
- Direct mapped

- Four 4-words blocks
- 2-way set associative
- Write-back strategy
- Write-allocate policy
- LRU Replacement

- No preemption during memory transaction
- One-word wide
- Bus access collision may take place ➔ blocking

Access time is 3 cycles

In case there are an I-Cache miss and a D-Cache miss happens in the same cycle, priority will be given to I-Cache

Example: With Memory Hierarchy

			1	2	3	4	5	6	7	8	9	10	11	12	13	14
	LI	R1, 100h	stall	stall	stall	stall	stall	stall	stall	stall	stall	stall	stall	IF	ID	EX1
	LW	R3, 0(R1)													IF	ID
	LI	R5, 1														
	LI	R7, 0h														
	LI	R6, 1h														
Loop:	MULT	R6, R5, R6														
	ADD	R7, R7, R6														
	ADDI	R5, R5, 1h														
	BNE	R5, R3, Loop														
	HLT															

I-Cache miss

Example: With Memory Hierarchy

			12	13	14	15	16	17	18	19	20	21	22	23	24	25
	LI	R1, 100h	IF	ID	EX1	EX2	EX3	EX4	ME	WB						
	LW	R3, 0(R1)		IF	ID	EX1	EX2	EX3	EX4	*stall*	*stall*	*stall*	*stall*	*stall*	*stall*	*stall*
	LI	R5, 1			IF	ID	EX1	EX2	EX3	EX4						
	LI	R7, 0h				IF	ID	EX1	EX2	EX3						
	LI	R6, 1h					stall	stall	stall	stall	stall	stall	stall	stall	stall	stall
Loop:	MULT	R6, R5, R6														
	ADD	R7, R7, R6														
	ADDI	R5, R5, 1h														
	BNE	R5, R3, Loop														
	HLT															

I-Cache miss

D-Cache miss that is not dealt with until the handling of the I-Cache miss finishes since bus is busy

D-Cache miss BUT memory bus is busy ➔ contention

			24	25	26	27	28	29	30	31	32	33	34	35	36	37
	LI	R1, 100h														
	LW	R3, 0(R1)	*stall*	*stall*	*stall*	*stall*	stall	stall	stall	stall	stall	stall	stall	stall	stall	stall
	LI	R5, 1														
	LI	R7, 0h														
	LI	R6, 1h	stall	stall	stall	IF	ID	EX1	EX2							
Loop:	MULT	R6, R5, R6					IF	ID	EX1							
	ADD	R7, R7, R6						IF	ID							
	ADDI	R5, R5, 1h														
	BNE	R5, R3, Loop														
	HLT															

D-Cache miss

I-Cache miss

Value of R6 is forwarded

The value of R5 is forwarded from EX4 (where LI is stranded)

Cannot progress since the previous 2 instructions are still stranded in the three EX stages

			35	36	37	38	39	40	41	42	43	44	45	46	47	48
	LI	R1, 100h														
	LW	R3, 0(R1)	stall	stall	stall	stall	ME	WB								
	LI	R5, 1					EX4	ME	WB							
	LI	R7, 0h					EX3	EX4	ME	WB						
	LI	R6, 1h					EX2	EX3	EX4	ME	WB					
Loop:	MULT	R6, R5, R6					EX1	EX2	EX3	EX4	ME	WB				
	ADD	R7, R7, R6					ID	stall	stall	stall	EX1	EX2	EX3	EX4	ME	WB
	ADDI	R5, R5, 1h					IF	stall	stall	stall	ID	EX1	EX2	EX3	EX4	ME
	BNE	R5, R3, Loop						stall	stall	stall	stall	stall	stall	stall	stall	stall
	HLT															

D-Cache miss

Data hazard

I-Cache miss

Data hazard is resolved due to the cache miss

			47	48	49	50	51	52	53	54	55	56				
	LI	R1, 100h														
	LW	R3, 0(R1)														
	LI	R5, 1														
	LI	R7, 0h														
	LI	R6, 1h														
Loop:	MULT	R6, R5, R6														
	ADD	R7, R7, R6	ME	WB												
	ADDI	R5, R5, 1h	EX4	ME	WB											
	BNE	R5, R3, Loop	stall	stall	stall	stall	stall	stall	stall	IF	ID					
	HLT										IF	ID				
	HLT											IF				

I-Cache miss

Data hazard is resolved
due to the cache miss

Output

clock cycle that instruction leaves each stage

	LI	R1, 100h	12	13	17	18	19
	LW	R3, 0(R1)	13	14	18	39	40
	LI	R5, 1	14	15	39	40	41
	LI	R7, 0h	15	16	40	41	42
	LI	R6, 1h	27	28	41	42	43
LOOP:	MULT	R6, R5, R6	28	29	42	43	44
	ADD	R7, R7, R6	29	42	46	47	48
	ADDI	R5, R5, 1h	42	43	47	48	49
	BNE	R5, R3, LOOP	54	55			
	HLT		55	56			
	HLT		56				

Cycle number of last stage (EX4 for most instructions)

Branching instructions terminate in the ID stage and does not have entries in EX, MEM and WB stages.

Total number of access requests for instruction cache: 11

Number of instruction cache hits: 8

Total number of access requests for data cache: 1

Number of data cache hits: 0