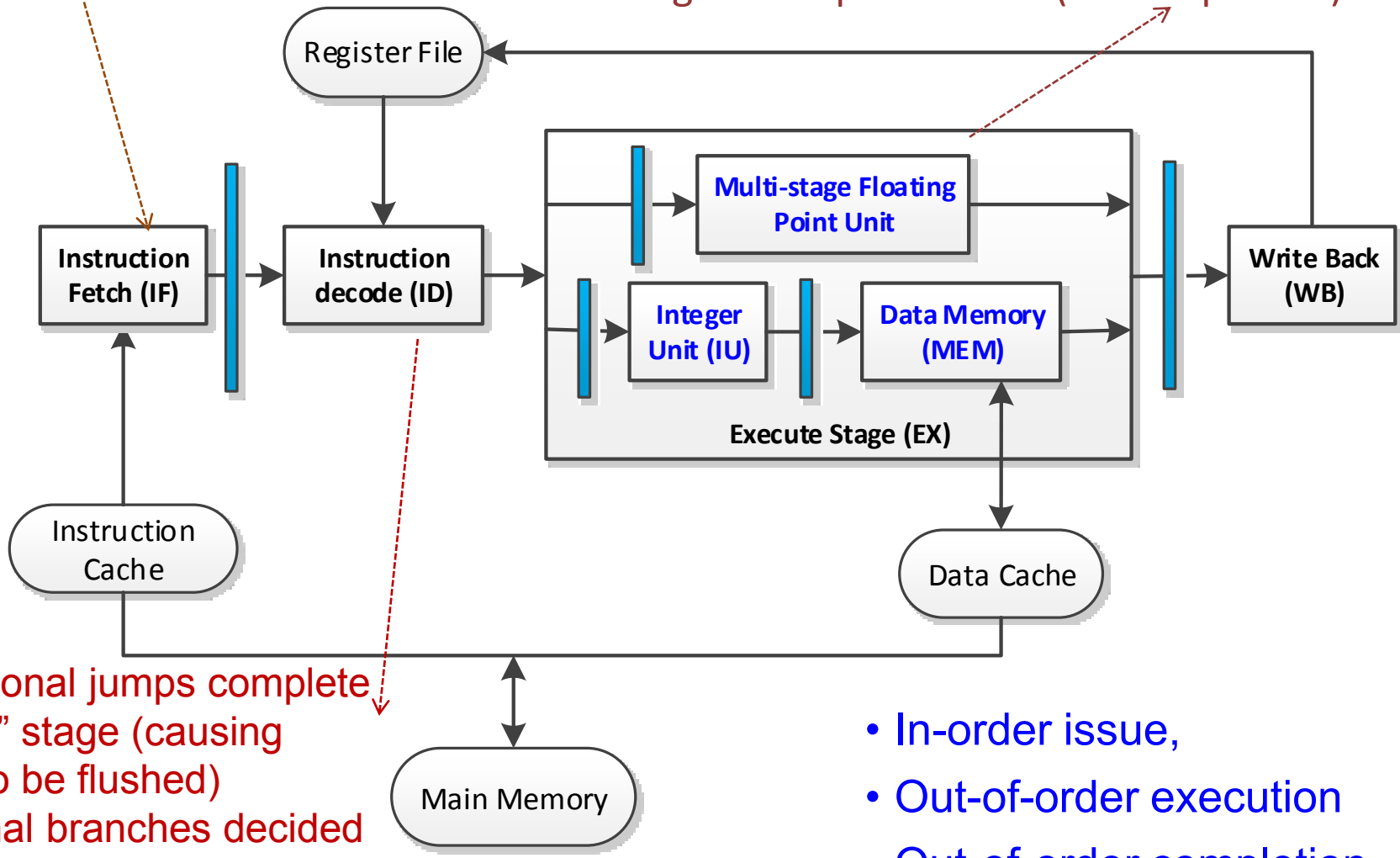
When a fetched instruction cannot be issued, the next instruction cannot be fetched.

Dynamically schedule instruction execution with multi-cycle floating point functional units whose configuration are provided as configuration parameters (in an input file)



- Unconditional jumps complete in the "ID" stage (causing fetched to be flushed)
- Conditional branches decided in the "ID" stage.
- "not-taken prediction" will be used in "IF" stage.

- In-order issue,
- Out-of-order execution
- Out-of-order completion
- No data bypassing

Input/Output Files

```
linux2[1]% ./simulator
Usage: simulator inst.txt data.txt reg.txt config.txt result.txt
```

Program:

- set of MIPS instruction
- Two HLT instruction marks end of program
- Labels are used for branching

Initial values for data memory and the integer registers

The "config.txt" file should include:

FP adder: *<cycle count>, <pipelined: <yes/no>*

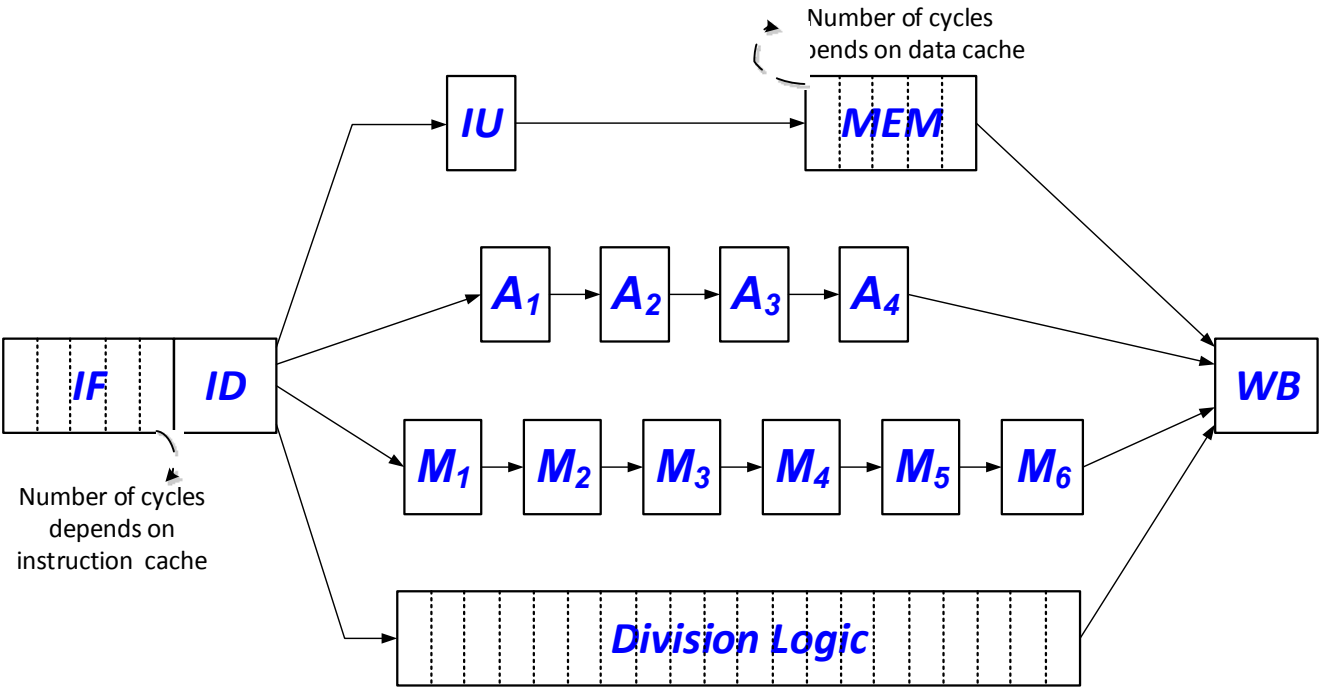
FP Multiplier: *< cycle count >, <pipelined: <yes/no>*

FP divider: *< cycle count >, < pipelined: yes/no>*

Main memory: *<access time (number of cycles)>*

I-Cache: *<access time (number of cycles)>*

D-Cache: *<access time (number of cycles)>*



`result.txt`

Trace the execution by list every instruction and when it passed through the various stages and what hazards it suffered, as well as the instr. and data cache performance

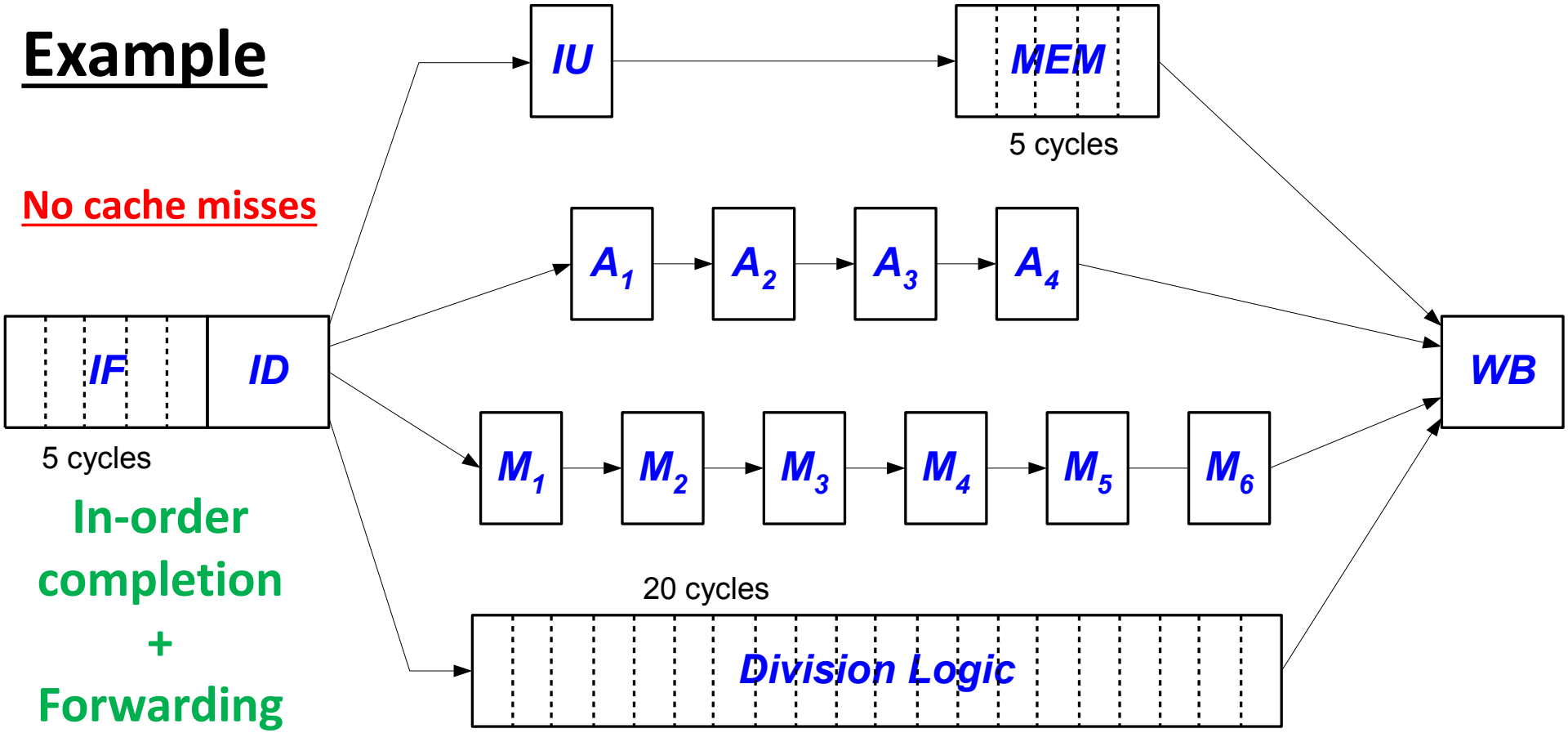
Instruction Class	Instruction Mnemonic
Data Transfers	LW, SW, L.D, S.D
Arithmetic/ logical	DADD, DADDI, DSUB, DSUBI, AND, ANDI, OR, ORI, ADD.D, MUL.D, DIV.D, SUB.D
Control	J, BEQ, BNE
Special purpose	HLT (to stop fetching new instructions)

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

Instructions	Number of Cycles in "Execute" Stage
HLT, J	0 Cycles (finish in ID stage)
BEQ, BNE	0 Cycle (finish in ID stage)
DADD, DADDI, DSUB, DSUBI, AND, ANDI, OR, ORI	2 Cycles (one for IU + one for MEM stage)
LW, SW, L.D, S.D	1 Cycle + memory access time (D-Cache)
ADD.D, SUB.D	Specified in the "*config.txt*" file
MUL.D	Specified in the "*config.txt*" file
DIV.D	Specified in the "*config.txt*" file

Example

No cache misses



In-order completion + Forwarding

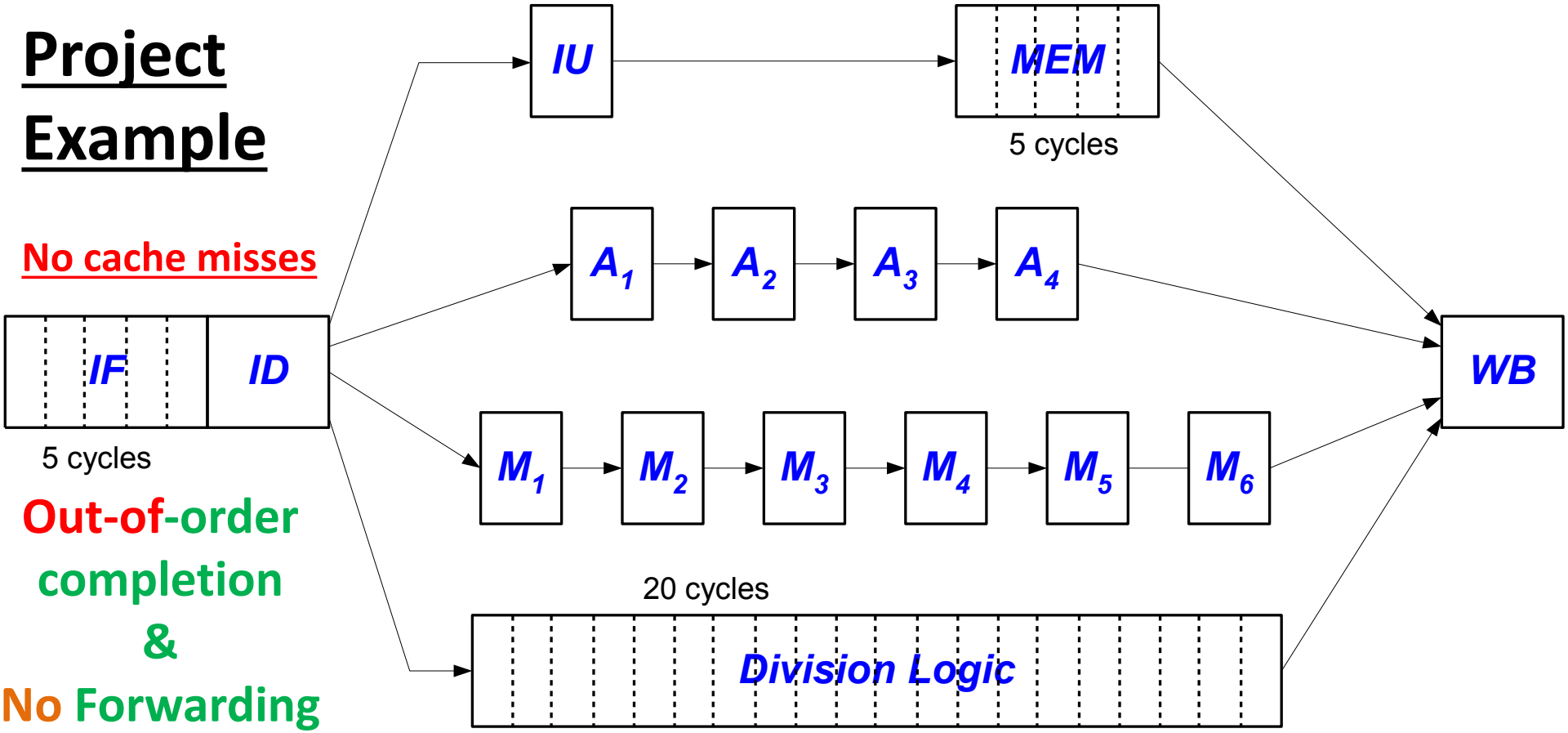
L.D F6, 34(R2)	1	6	7	8	13
L.D F2, 45(R3)	6	11	12	13	18
MUL.D F0, F2, F4	11	16	18		24
SUB.D F8, F6, F2	16	21	22		26
DIV.D F10, F0, F6	21	26	27		47
ADD.D F6, F8, F2	26	31	44		48

RAW + forward

In-order completion

Project Example

No cache misses



Out-of-order completion & No Forwarding

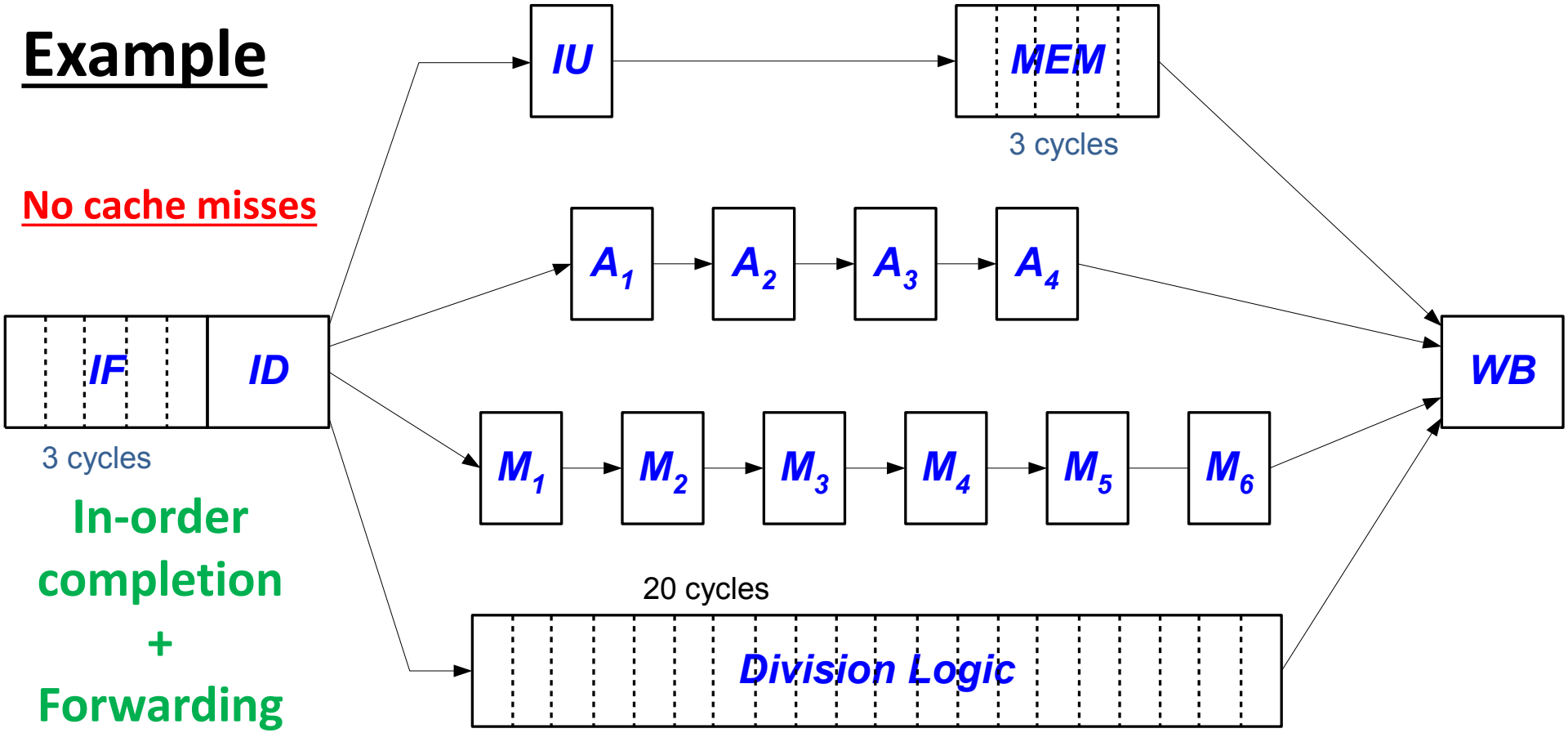
L.D F6, 34(R2)	1	6	7	8	13
L.D F2, 45(R3)	6	11	12	13	18
MUL.D F0, F2, F4	11	16	19		25
SUB.D F8, F6, F2	16	21	22		26
DIV.D F10, F0, F6	21	26	27		47
ADD.D F6, F8, F2	26	31	32		36

RAW with NO forward

Out-of-order completion

Example

No cache misses



In-order completion + Forwarding

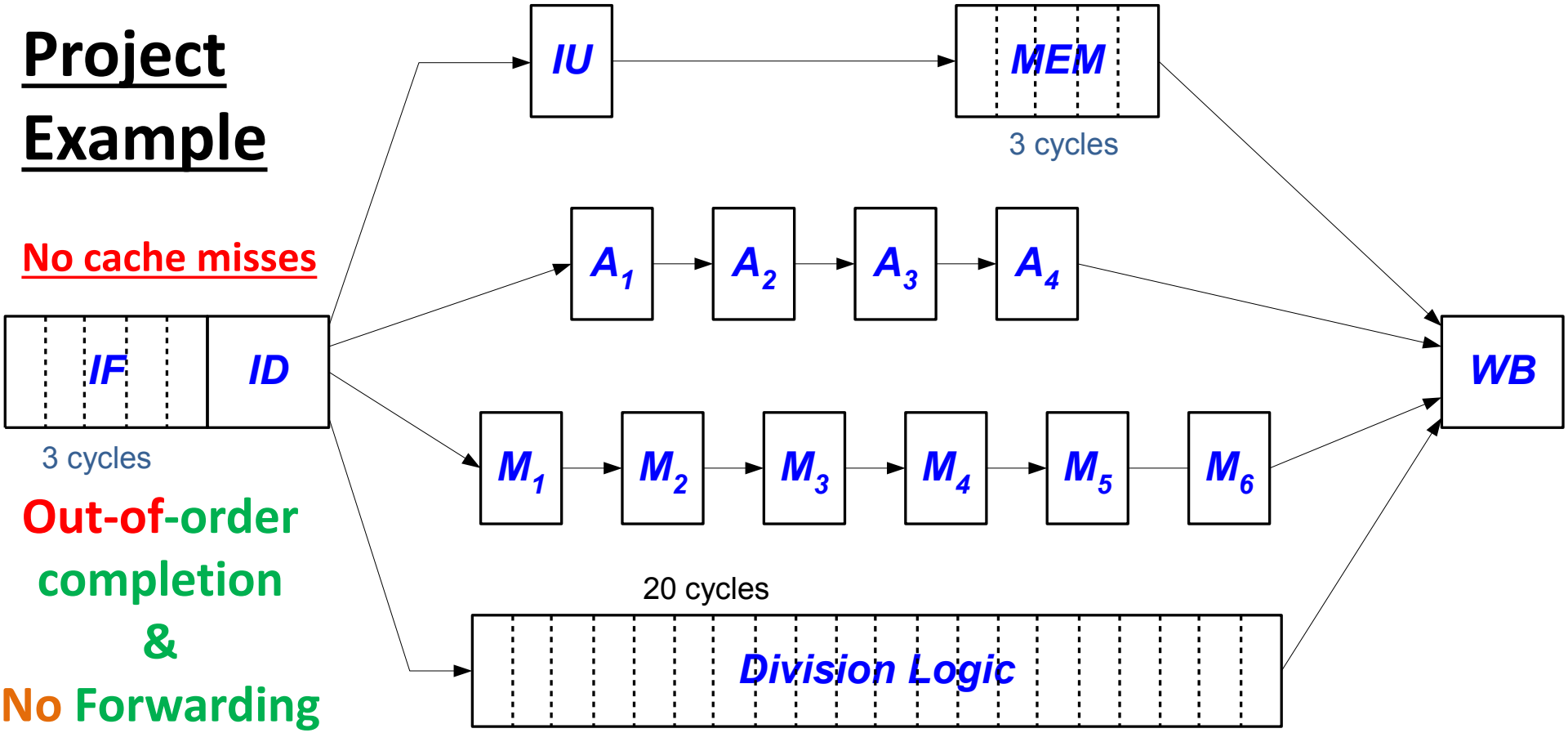
L.D F6, 34(R2)	1	4	5	6	9
L.D F2, 45(R3)	4	7	8	9	12
MUL.D F0, F2, F4	7	10	12		18
SUB.D F8, F6, F2	10	13	15		19
DIV.D F10, F0, F6	13	16	18		38
ADD.D F6, F8, F2	16	19	35		39

RAW + forward

In-order completion

Project Example

No cache misses



Out-of-order completion & No Forwarding

L.D F6, 34(R2)	1	4	5	6	9
L.D F2, 45(R3)	4	7	8	9	12
MUL.D F0, F2, F4	7	10	13		19
SUB.D F8, F6, F2	10	13	14		18
DIV.D F10, F0, F6	13	16	20		40
ADD.D F6, F8, F2	16	19	20		24

RAW with NO forward

Out-of-order completion

Example

inst.txt

```
GG:   L.D   F1,  4(R4)
      L.D   F2,  8(R5)
      ADD.D  F4,   F6,  F2
      SUB.D  F5,   F7,  F1
      MUL.D  F6,   F1, F5
      ADD.D  F7,  F2,  F6
      ADD.D  F6,  F1,  F7
      DADDI  R4,   R4,  2
      DADDI  R5,   R5,  2
      DSUB   R1,   R1,  R2
      BNE    R1,   R3,  GG
      HLT
      HLT
```

config.txt

FP adder: 4, yes

FP Multiplier: 6, yes

FP divider: 20, no

Main memory: 2

I-Cache: 1

D-Cache: 1

Example: Without Memory Hierarchy

(1st iteration)

Instruction	IF	ID	EX	WB	RAW	WAR	WAW	Struct
GG: L.D F1, 4(R4)	1	2	5	6	N	N	N	N
L.D F2, 8(R5)	2	3	7	8	N	N	N	Y
ADD.D F4, F6, F2	3	8	12	13	Y	N	N	N
SUB.D F5, F7, F1	8	9	13	14	N	N	N	N
MUL.D F6, F1, F5	9	14	20	21	Y	N	N	N
ADD.D F7, F2, F6	14	21	25	26	Y	N	N	N
ADD.D F6, F1, F7	21	26	30	31	Y	N	N	N
DADDI R4, R4, 4	26	27	29	30	N	N	N	N
DADDI R5, R5, 4	27	28	31	32	N	N	N	Y
DSUB R1, R1, R2	28	29	32	33	N	N	N	Y
BNE R1, R3, GG	29	33			Y	N	N	N
HLT	33				N	N	N	N

Example: Without Memory Hierarchy

(2nd iteration)

Instruction	IF	ID	EX	WB	RAW	WAR	WAW	Struct
GG: L.D F1, 4(R4)	34	35	38	39	N	N	N	N
L.D F2, 8(R5)	35	36	40	41	N	N	N	Y
ADD.D F4, F6, F2	36	41	45	46	Y	N	N	N
SUB.D F5, F7, F1	41	42	46	47	N	N	N	N
MUL.D F6, F1, F5	42	47	53	54	Y	N	N	N
ADD.D F7, F2, F6	47	54	58	59	Y	N	N	N
ADD.D F6, F1, F7	54	59	63	64	Y	N	N	N
DADDI R4, R4, 4	59	60	62	63	N	N	N	N
DADDI R5, R5, 4	60	61	64	65	N	N	N	Y
DSUB R1, R1, R2	61	62	65	66	N	N	N	Y
BNE R1, R3, GG	62	66			Y	N	N	N
HLT	66	67			N	N	N	N
HLT	67							

Important Notes

- The pipeline processor does not have forwarding hardware.
- In addition to the 32 word-size registers (for integers), there are 32 FP registers; each has 64 bits.
- Floating point calculations will have no impact on the required output of your simulator. In fact, only the contents of the integer registers need to be read from the input file, and you do not even need to allocate storage in your simulator for floating point registers.
- The number of cycles required by the ALU depends on the latency of the involved functional unit and whether it is pipelined or not.
- Instructions and data are stored in memory starting at address 0x0 and 0x100 respectively. Load and store instructions use word 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.
- 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 decode stage. You can ignore the second HLT instruction.

Data Hazards

- Integer and floating point operations use the same write port and hence structural hazards can occur.
  - Structural hazards are detected before entering the WB stages.
  - The functional unit that has the instruction will be stalled if the instruction cannot proceed to WB stage.
  - In case multiple instructions are ready at the same time to the WB stage, the priority will be given to the functional unit that in not pipelined and takes the most execution cycle (based on the parameters in “config.txt”). If there is a tie, the instruction that was issued the earliest will have the priority.
- An instruction stalled for RAW hazard in the ID stage can get the values in the same cycle WB takes place.
- WAW hazards are detected at the ID stages and resolved by stalling the pipeline.
- An instruction may suffer multiple hazards (all needs to be reported)
  - WAW and Structural hazards are checked first, then RAW hazard
  - Report ONLY RAW, WAR and WAW data hazards that cause STALLS.

Output

clock cycle that instruction leaves each stage

Hazards that caused stalls

Instruction	FT	ID	EX	WB	RAW	WAR	WAW	Struct
GG: LD F1, 4(R4)	6	7	14	15	N	N	N	N
LD F2, 8(R5)	7	8	16	17	N	N	N	Y
:								
:								
DSUB R1, R1, R2	80	81	84	85	N	N	N	Y
BNE R1, R3, GG					Y	N	N	N
HLT	85	86			N	N	N	N
HLT	91							

Total number of access requests for instruction cache: 24

Number of instruction cache hits: 21

Total number of access requests for data cache: 8

Number of data cache hits: 4

Cycle number of last stage (WB for ALU instructions)

A branching instruction terminates in "ID" stage and does not have entries in the EX and WB stages.