Computer Architecture (Spring 2020)

Pipelining

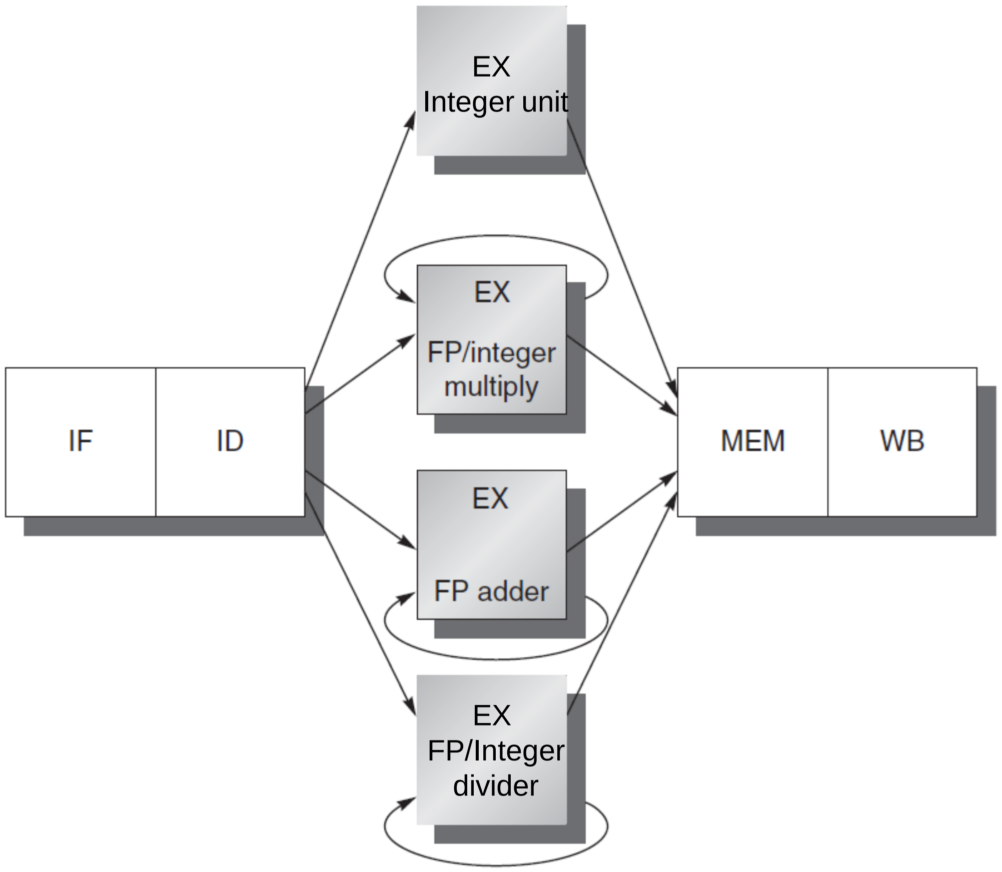
Dr. Duo Liu (刘铎)
Office: Main Building 0626
Email: liuduo@cqu.edu.cn

Pipeline Terminology

- Fetched instruction
  - has been retrieved from the I-Cache
- Decoded instruction
  - has completed the ID stage
- Issued instruction
  - has been decoded and has started its execution
    - in MIPS, it has moved from ID state to EX stage
- Executed instruction
  - has completed its execution in the EX stage
- Committed instruction
  - is guaranteed to complete
    - in MIPS, it has entered the WB stage (or the end of the MEM stage) and the status vector doesn’t contain any exception flag for it

Handling Multi-Cycle Operations in Pipelined Implementations

- Motivation: Impractical to require that all FP operations complete in 1 or 2 clock cycles
  - CCT would become too large
  - it would require huge design costs
- For higher clock rates
  - put fewer logic levels in each pipeline stage
    - more pipe stages for complex operations
    - more pipeline registers necessary
    - tradeoff lower CCT vs. higher CPI

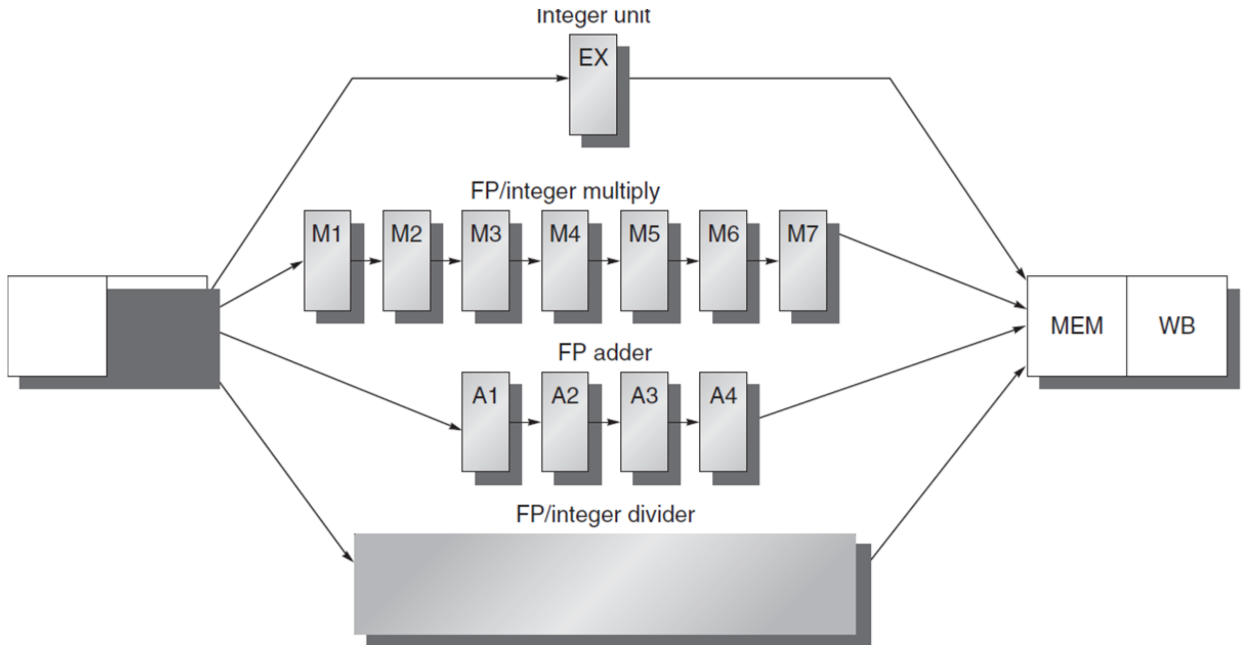


Latency and Initiation Interval

Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data memory (integer and FP loads)	1	1
FP add	3	1
FP multiply (also integer multiply)	6	1
FP divide (also integer divide)	24	25

- Two important terms of pipelined operations
  - Latency: the number of intervening cycles between an instruction that produces a result and an instruction that uses the result.
  - Initiation interval: The number of cycles that must elapse between issuing two operations of a given type.
- Since most operations consume their operands at the beginning of EX, the latency is usually the # of stages after EX that an instruction produces a result
  - Integer ALU operations are all 0;
  - Loads are 1, since their results can be used after one intervening cycle.
  - Stores, which consume the value being stored 1 cycle later, are little different. The latency for the value being stored will be 1 cycle less.
- Pipeline latency is essentially equal to 1 cycle less than the depth of the execution pipeline, which is the number of stages from the EX stage to the stage that produces the result.

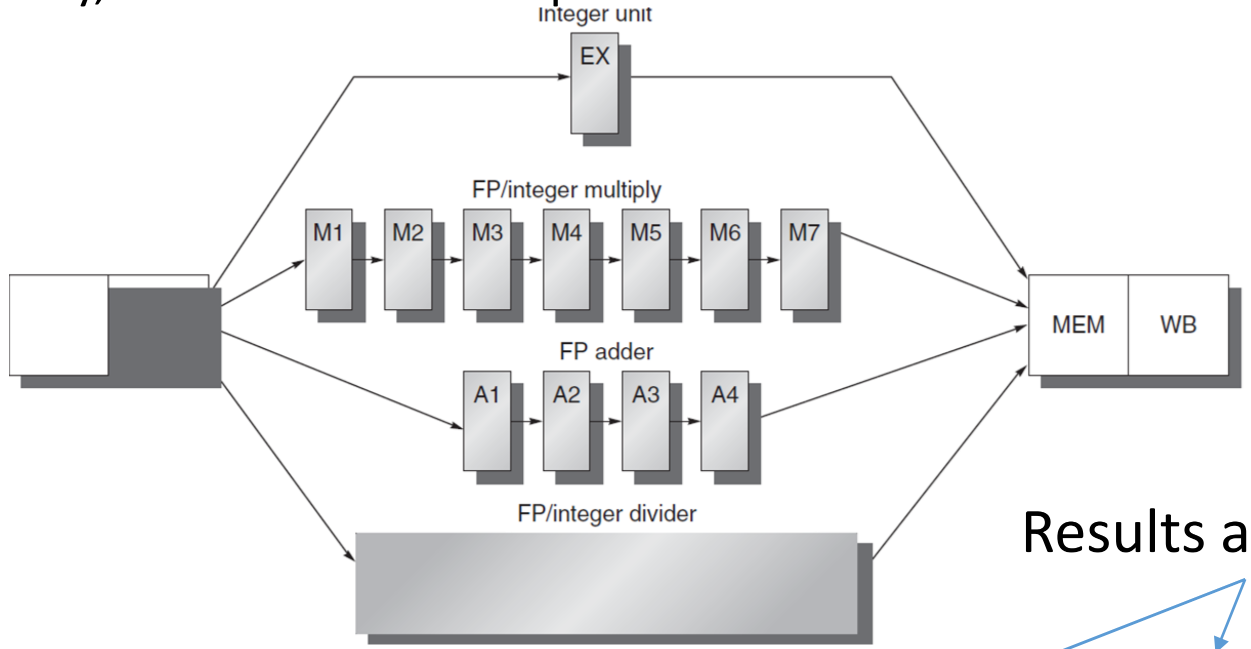
Multi-Stage” vs. “Multi-Cycle” Operations: i.e. Pipelined Stages vs. Sequential Stages



FunctionalUnit	(Additional)Latency	Initiation Interval
IntegerALU	0	1
Data Memory(Int&FP Loads)	1	1
FP Add	3	1
Int&FP Multiply	6	1
Int&FP Divide (SEQUENTIAL!)	24	25

A pipeline supporting multiple outstanding FP operations.

- The FP multiplier and adder are pipelined and have a depth of 7 and 4 stages.
- The FP divider is not pipelined, but requires 24 cycles to complete.
- For example, the 4th instruction after an FP add can use its result without being stalled. Similarly, 7th after an FP multiplier can use its result without being stalled.



Results are ready

MUL.D	IF	ID	*M1*	M2	M3	M4	M5	M6	**M7**	MEM	WB
ADD.D		IF	ID	*A1*	A2	A3	**A4**	**MEM**	WB		
L.D			IF	ID	*EX*	**MEM**	WB				
S.D				IF	ID	*EX*	*MEM*	WB			

Data are needed

Hazards due to Multiple Outstanding Operations

- Structural Hazards
  - sequential (non-pipelined) function units
    - e.g., two “close” DIV operations require stalling
  - number of register writes per cycle may be larger than 1
- More Frequent RAW Data Hazards
    - “instruction y (which follows instruction x in the program order) tries to read a source register before x writes it”
  - longer latency of operations produce more stalls
- Two New Classes of Data Hazards
  - WAW
    - “instruction y tries to write a destination register before x writes it”
  - WAR
    - “instruction y tries to write a destination register before x reads it”
- Out-of-order completion
  - makes exception handling more complex

Hazards due to Multiple Outstanding Operations – Example of WAW Hazard

- WAW
    - “instruction y tries to write a destination register before x writes it”
- One may wonder how can WAW hazards occur since it may seem that a compiler would never generate two writes to the same register without an intervening read
- However such sequences do occur
    - e.g., a program may take a path that the compiler cannot predict, e.g. due to an intervening branch
- Also, in the case of multiple-issue pipelines, an intervening read may be present (and stalled) while the second write could progress

```
DIV.D F1, F2, F4
BEQZ R2, target
LD F9, 0(R10)
ADD.D F1, F5, F6
…; other non-branch instructions
target: SUB.D F8, F9, F1
```

- In this example if the branch is not taken a WAW if possible when the DIV completes its execution after the BEQZ, LD, and ADD complete theirs

Hazards due to Multiple Outstanding Operations – Example of Structural Hazard

Instruction	1	2	3	4	5	6	7	8	9	10	11
MUL.D F0,F4,F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
ADD R1, R3, R4		IF	ID	EX	MEM	WB					
ADD R2, R4, R5			IF	ID	EX	MEM	WB				
ADD.D F2,F4,F6				IF	ID	A1	A2	A3	A4	MEM	WB
ADD R6, R9, R7					IF	ID	EX	MEM	WB		
ADD R8, R3, R9						IF	ID	EX	MEM	WB	
L.D F2, 0(R2)							IF	ID	EX	MEM	WB

- With only 1 write port in register file, the processor must serialize the 3 WB instructions
  - Option 1: Tracking port access during ID and (pre)-stalling corresponding instruction (same logic as for interlock detection)
  - Option 2: Stall the conflicting instructions with smaller latency ("least-waited") just before MEM or WB
    - Notice that there is no structural hazard at MEM because, among the 3 instructions, only L.D really needs to access memory. However, this assumes that there is separate HW to make the other two instructions progress into WB as they need

Hazards due to Multiple Outstanding Operations – Example of RAW Hazard

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
L.D F4, 0(R2)	IF	ID	EX	MEM	WB												
MUL.D F0,F4,F6		IF	ID	●	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
ADD.D F2,F0,F8			IF	●	ID	●	●	●	●	●	●	A1	A2	A3	A4	MEM	WB
S.D F2, 0(R2)				●	IF	●	●	●	●	●	●	ID	EX	●	●	🔴	MEM

- Deeper pipelines increase the stall frequencies
  - each instruction in the example depends on the previous one
  - each instruction proceeds as soon as the awaited data become available (optimal forwarding)
  - to avoid a RAW hazard the S.D must wait for the ADD.D to provide the new value of F2 (via bypassing) before entering MEM
    - NB: Fig. C.37 is not consistent with C.38 since it implies that there is a structural hazard on MEM because ADD.D does not really need the MEM stage (but only to progress execution)

Hazards due to Multiple Outstanding Operations – Control Implementation

- Assuming that all hazard detections are done in ID, before issuing an instruction x we must
  - check for structural hazards
    - the required function unit must be available (e.g., FP Divisor)
    - no future conflicts on the register write port
  - check for RAW data hazards
    - no conflict between x source register and previous instruction destination registers
      - e.g. if x is a FP operation on F2, check that F2 is not a destination in ID/A1, A1/A2, A2/A3
  - check for WAW data hazards
    - no instruction in A1,…,A4, D, M1,…M7 can have the same register destination as x
- Logic implementation is similar as for the MIPS integer pipeline