System Hardware

Tutorial 8 Pipeline Data Flows & Stalls

Pipeline Recall From Lecture-7

- Instruction Pipelining is a naturally efficient way for a datapath to execute machine instructions
- The basic idea is to allow different workstations (boxes) to work on different instructions at the same time.
- Overlapping the execution of different machine instructions, produces significant improvement in the pipeline's execution ***throughput***.
- ***Recall that instruction throughput*** is the number of instructions that can be executed in a unit of time.

The MIPS Pipeline Example

Clock Cycle	1	2	3	4	5	6	7	8	9	0	
Instruction1	f	d	x	m	w						
Instruction2		f	d	x	m	W					
Instruction3			f	d	x	M	w				
Instruction4				f	d	X	m	w			
Instruction5					f	D	x	m	w		
Instruction6						F	d	x	m	w	
Instruction7							f	d	x	m	w

- We have also seen this example that illustrates the overlapping of the tasks of the 5 boxes f, d, x, m, and w.
- Instruction latency (time to execute 1 instruction) is 5 cycles.
- When the pipeline achieves cruising speed (equilibrium), a new instruction completes every cycle.
- The pipeline ***execution bandwidth*** is 1 instruction per cycle.
- The pipeline speedup is the ***pipeline depth*** (number of stages).

The MIPS Pipeline Summary

- Each of the five boxes is a ***combinational circuit***, but the clocked pipeline as a whole is a ***sequential circuit***.
- Each box is followed immediately on its right by a set of ***non-ISA pipeline registers*** , called ***pipeline latches***.
- ***Non-ISA*** means they are distinct from the ***CPU ISA registers***.
- Prior to the end of a clock cycle, all the results from a given stage must be stored in the pipeline latch to its right, in order that these values to be preserved across clock cycles, and serve as inputs for the next stage.
- The boxes are combinational circuits that compute Boolean functions, and the latches provide controls to the boxes.

The MIPS Pipeline Control Signals

- At the beginning of each time cycle (c), each box receives a ***Control Signal*** alongside the associated data. Each box performs its specified action during cycle c.
- The following control signals will be considered: ***no-op***, ***add***, ***multiply***, ***load***, ***store***, and ***write back***. The ***no-op*** signal means "do nothing".
- **Example: m-box in a store** receives a store signal from the pipeline, the data to be stored and the memory address to store the data at.
- However, the **m-box in a load** only receives a load signal and the memory address from where the data will be fetched.
- For instructions with destination register (e.g. add, load, mul), the **w-box** receives from the pipeline: a ***write back signal***, the data value (to be ***written back***) and a register name (where the data will be written).

The MIPS Pipeline Space-Time Diagram

- A space-time diagram (on slide 3) shows the evolution of the pipeline in time, and is useful in capturing aspects of the control and data flow.
- Let's repeat the figure with some actual instructions.

Clock Cycle	1	2	3	4	5	6	7	8	9
add r1, r2, r3	f	d	x	n	w				
add r2, r3, r4		f	d	x	n	w			
add r3, r4, r5			f	d	x	n	w		
add r4, r5, r6				f	d	x	n	w	
add r5, r6, r7					f	d	x	n	w

There is no data flows between instructions (they are independent).

Equilibrium is reached in cycle 5: one instruction is fetched and one result is produced (bandwidth=1 instruction per cycle). Latency=5 cycles. Pipeline concurrency = bandwidth * latency = 5.

The MIPS Pipeline
Data Flows & Control Signals

- Data flows between different boxes working on the same instruction. **In the first instruction** of the previous example:
  - The x-box receives the values of 'r2' and 'r3', which it adds.
  - The w-box receives the register name "r1" and the sum computed by the x-box and does a write-back of the computed value to r1.
  - However, the m-box receives no data at all. Since an add is not a memory reference, the m-box has no work to do. Instead, it performs a no-op.
- In cycle 1, The f-box receives the control signal 'fetch', while ***all*** the other boxes receive the control signal ***no-op***.

The MIPS Pipeline
Examples of Control Signals

- In every cycle, the f- and d-boxes receive either the ***no-op*** control signal or the ***fetch*** and ***decode*** control signals, respectively. From a control standpoint, these are not very interesting case statements. For this reason, we focus on the x-, m-, and w-boxes.
- When the x-box does something, it may be any one of a range of arithmetic or logical operations.
- When the m-box does something, it is either a load or a store.
- When the w-box does something, it is always write back.
- If a box name appears in the column corresponding to cycle 'c', then that box is active in cycle 'c'.

The MIPS Pipeline
Data Dependency

- In the space-time diagram example we saw in slide 6, all instructions were independent. In other words, there was no data dependency between instructions.
- Lets consider the following program segment:

```
l.d     f4,0(r2)
mul.d   f6,f4,f2
```

- There is a ***data dependence*** (more precisely ***flow dependence***) between the first instruction and the second instruction.
- This simply means that the first instruction ***produces*** a value that is ***consumed*** by the second instruction.
- Clearly, when executing this program, we cannot multiply 'f4‘ by 'f2' until we know what 'f4' is.

The MIPS Pipeline Control Dependency

- Here's another example:

```
        bne   r1,r2,skip
        mul.d f6,f4,f2
skip:
```

- The multiplication instruction is ***control dependent*** on the conditional-branch instruction. This simply means that the branch controls whether the multiplication is executed or not.
- Clearly, when running this program, we should delay execution of 'mul.d f6,f4,f2' until we get a decision from the branch.
- ***To Summarize***: When a program containing data or control dependencies runs on a pipeline with a particular pipeline organization, pipeline hazards may result.

The MIPS Pipeline
Data Dependency Example

- Consider the following instruction sequence and imagine that instructions could communicate through the register file only.

Clock Cycle	1	2	3	4	5	6	7	8
add r1,r2,r3	f	d	x	n	w			
add r4,r1,r6		f	d	s	s	x	n	w

Naïve Pipeline

1	2	3	4	5	6
f	d	x	n	w	
	f	d	x	n	w

Sensible Pipeline

- In the naïve pipeline, the first '**add**' writes its result to r1 in cycle 5. The result can be retrieved and the second '**add**' could begin work in cycle 6. The pipeline will have to ***stall*** and delay the second 'add' by two cycles (Stalls aka ***bubbles***).
- A sensible pipeline avoids stalls: The value of 'r1' is latched into the x/m latch‘ before the end of cycle 3, and can be retrieved at the start of cycle 4.
- Communication through pipeline latches is called ***forwarding*** and will be assumed in all future pipelines.

The MIPS Pipeline Forwarding and Latches

- Communicating through the register file is bad only when it causes a delay.
- Recall that the pipeline receives a new instruction to execute in every cycle. For this reason, a pipeline behaves like a river. Values in latches move one latch to the right in each cycle.
- Since there are only four latches, a value kicked out of the 'm/w latch' goes to the register file.

The MIPS Pipeline Branch Stall

- Consider the following instruction sequence running on a f,d,x,m,w pipeline with forwarding.

Clock Cycle	1	2	3	4	5	6	7	8
`bne r1,r2,loop`	f	d						
`l.d f6,-24(r3)`		f	f	d	x	m	w	

- At the end of cycle 1, PC has been incremented by 4. In cycle 2, **bne** may or may not overwrite the PC. ***Why***?
  - If it does, then the fetch in cycle 2 is a mistake, and must be redone. The fact that we did the fetch without waiting for the branch's decision means that we are following a ***predict not taken*** strategy. This is effectively a ***branch stall***. Other branch strategies: stall pipeline, predict taken.
  - If the branch is not taken, there is no stall.

The MIPS Pipeline Other Stalls

Clock Cycle	1	2	3	4	5
dadd r1,r1,#-8	f	d	x	n	w
bne r1,r2,loop		f	s	d	

In the above diagram, the branch instruction has to wait for the result of the add (r1) to determine if the branch will be taken. This a ***pre-branch stall***.

One More Example:

Clock Cycle	1	2	3	4	5	6	7	8
l.d f6,-24(r3)	f	d	x	m	w			
mul.d f8,f6,f4		f	d	s	x	n	w	

f6 is retrieved from memory at the end of cycle 4 and placed in the m/w latch. Therefore, the multiplication can only begin in cycle 5. This is a ***load or data stall***.

Conclusion

- We have taken a deeper view into the fdxmw pipeline: boxes, latches, data flows, and control signals.
- Examples of Data flows and control signals between boxes.
- A space-time diagram shows the evolution of the pipeline in time (control & data flows).
- Different Types of dependency exist between instructions: data dependency or control dependency.
- Communication in the pipeline can take place through the register file only or also through the latches. The latter is called forwarding and reduces pipeline stalls.
- Different types of pipeline stalls: data stall, branch stall, load stall, pre-branch stall.