More on Pipelining

CSE 2312
Computer Organization and Assembly Language Programming
Vassilis Athitsos
University of Texas at Arlington

Fetch-Decode-Execute Cycle in Detail

- The CPU clock ticks to mark start of cycle.
    1. Fetch next instruction from memory
    2. Change program counter to point to next instruction
    3. Determine type of instruction just fetched
    4. If instruction uses a word in memory, locate it
    5. Fetch word, if needed, into a CPU register.
    6. Execute instruction.
    7. The clock cycle is completed. Go to step 1 to begin executing the next instruction.

Toy ISA Instructions

- **add A B C**:
  - Adds contents of registers A and B, stores result in register C.
- **addi N A C**:
  - Adds integer N to contents of register A, stores result in register C.
- **load address A**:
  - Loads data from the specified memory address to register A.
- **store A address**:
  - Stores data from register A to the specified memory address.
- **goto line**:
  - Set the instruction counter to the specified line. That line should be executed next.
- **if A line**:
  - If the contents of register A are NOT 0, set the instruction counter to the specified line. That line should be be executed next.

Defining Pipeline Behavior

- In the following slides, we will explicitly define how each instruction goes through the pipeline.
- This is a toy ISA that we have just made up, so the following conventions are designed to be simple, and easy to apply.
- You may find that, in some cases, we could have followed other conventions that would make execution even more efficient.

Pipeline Steps for: **add A B C**

- **Fetch Step:**
- **Decode Step:**
- **Operand Fetch Step:**
- **Execution Step:**
- **Output Save Step:**

- **NOTES:**

Pipeline Steps for: **add A B C**

- **Fetch Step:** Fetch instruction from memory location specified by PC. Increment PC to point to the next instruction.
- **Decode Step:** Determine that this statement uses the ALU, takes input from registers A and B, and modifies register C.
- **Operand Fetch Step:** Copy contents of registers A and B to ALU input registers.
- **Execution Step:** The ALU unit performs addition.
- **Output Save Step:** The result of the addition is copied to register C.

- **NOTES:** This instruction must wait at the decode step until all previous instructions have finished modifying the contents of registers A and B.

Pipeline Steps for: **addi N A C**

- **Fetch Step:**
- **Decode Step:**
- **Operand Fetch Step:**
- **Execution Step:**
- **Output Save Step:**

- **NOTES:**

Pipeline Steps for: **addi N A C**

- **Fetch Step:** Fetch instruction from memory location specified by PC. Increment PC to point to the next instruction.
- **Decode Step:** Determine that this statement uses the ALU, takes input from register A, and modifies register C.
- **Operand Fetch Step:** Copy content of register A into one ALU input register, copy integer N into the other ALU input register.
- **Execution Step:** The ALU unit performs addition.
- **Output Save Step:** The result of the addition is copied to register C.

- **NOTES:** This instruction must wait at the decode step until all previous instructions have finished modifying the contents of register A.

Pipeline Steps for: **load address A**

- **Fetch Step:**
- **Decode Step:**
- **Operand Fetch Step:**
- **Execution Step:**
- **Output Save Step:**

- **NOTES:**

Pipeline Steps for: **load address A**

- **Fetch Step:** Fetch instruction from memory location specified by PC. Increment PC to point to the next instruction.
- **Decode Step:** Determine that this statement accesses memory, takes input from **address**, and modifies register A.
- **Operand Fetch Step:** Not applicable for this instruction.
- **Execution Step:** The bus brings to the CPU the contents of **address**.
- **Output Save Step:** The data brought by the bus is copied to register C.

- **NOTES:** This instruction must wait at the decode step until all previous instructions have finished modifying the contents of **address**.

Pipeline Steps for: **store A address**

- **Fetch Step:**
- **Decode Step:**
- **Operand Fetch Step:**
- **Execution Step:**
- **Output Save Step:**

- **NOTES:**

Pipeline Steps for: **store A address**

- **Fetch Step:** Fetch instruction from memory location specified by PC. Increment PC to point to the next instruction.
- **Decode Step:** Determine that this statement accesses memory, takes input from register A, and modifies **address**.
- **Operand Fetch Step:** Not applicable for this instruction.
- **Execution Step:** The bus receives the contents of register A from the CPU.
- **Output Save Step:** The bus saves the data at **address**.

- **NOTES:** This instruction must wait at the decode step until all previous instructions have finished modifying the contents of register A.

Pipeline Steps for: **goto line**

- **Fetch Step:**
- **Decode Step:**
- **Operand Fetch Step:**
- **Execution Step:**
- **Output Save Step:**

- **NOTES:**

Pipeline Steps for: **goto line**

- **Fetch Step:** Fetch instruction from memory location specified by PC. Increment PC to point to the next instruction.
- **Decode Step:** Determine that this statement is a **goto**. Flush (erase) what is stored at the fetch step in the pipeline.
- **Operand Fetch Step:** Not applicable for this instruction.
- **Execution Step:** Not applicable for this instruction.
- **Output Save Step:** The program counter (PC) is set to the specified **line**.

- **NOTES:** See next slide.

Pipeline Steps for: **goto line**

- **NOTES:** When a **goto** instruction completes the decode step:
  - The pipeline **stops receiving** any new instructions. However, instructions that entered the pipeline **before** the **goto** instruction continue normal execution.
  - The pipeline ignores and does not process any further the instruction that was fetched while the **goto** instruction was decoded.
- Fetching statements resumes as soon as the **goto** instruction has finished executing, i.e., when the **goto** instruction **has completed** the output save step.

Pipeline Steps for: **if A line**

- **Fetch Step:**
- **Decode Step:**
- **Operand Fetch Step:**
- **Execution Step:**
- **Output Save Step:**

- **NOTES:**

Pipeline Steps for: **if A line**

- **Fetch Step:** Fetch instruction from memory location specified by PC. Increment PC to point to the next instruction.
- **Decode Step:** Determine that this statement is an **if** and that it accesses register **A**. Flush (erase) what is stored at the fetch step in the pipeline.
- **Operand Fetch Step:** Copy contents of register A to first ALU input register.
- **Execution Step:** The ALU compares the first input register with 0, and outputs 0 if the input register equals 0, outputs 1 otherwise.
- **Output Save Step:** If the ALU output is 1, the program counter (PC) is set to the specified **line**. Nothing done otherwise.

- **NOTES:** See next slide.

Pipeline Steps for: **if A line**

- **NOTE 1:** an **if** instruction must wait at the decode step until all previous instructions have finished modifying register A.
- When an **if** instruction completes the decode step:
  - The pipeline **stops receiving** any new instructions. However, instructions that entered the pipeline **before** the **if** instruction continue normal execution.
  - The pipeline erases and does not process any further the instruction that was fetched while the **if** instruction was decoded.
- Fetching statements resumes as soon as the **if** instruction has finished executing, i.e., when the **if** instruction **has completed** the output save step.

Pipeline Execution: An Example

- Consider the program on the right.
- The previous specifications define how this program is executed step-by-step through the pipeline.
- To trace the execution, we need to specify the inputs to the program.
- Program inputs:
- Program outputs:

line 1: **load address2 R2**
line 2: **load address1 R1**
line 3: **if R1 6**
line 4: **addi 20 R1 R3**
line 5: **goto 7**
line 6: **addi 10 R1 R3**
line 7: **addi 5 R2 R4**
line 8: **store R4 address10**
line 9: **addi 30 R2 R5**
line 10: **store R5 address11**
line 11: **add R3 R2 R8**
line 12: **store R8 address12**

Pipeline Execution: An Example

- Consider the program on the right.
- The previous specifications define how this program is executed step-by-step through the pipeline.
- To trace the execution, we need to specify the inputs to the program.
- Program inputs:
  - address1, let's assume it contains 0.
  - address2, let's assume it contains 10.
- Program outputs:
  - address10
  - address11
  - address12

line 1: **load address2 R2**
line 2: **load address1 R1**
line 3: **if R1 6**
line 4: **addi 20 R1 R3**
line 5: **goto 7**
line 6: **addi 10 R1 R3**
line 7: **addi 5 R2 R4**
line 8: **store R4 address10**
line 9: **addi 30 R2 R5**
line 10: **store R5 address11**
line 11: **add R3 R2 R8**
line 12: **store R8 address12**

line 1: **load address2 R2**
line 2: **load address1 R1**
line 3: **if R1 6**
line 4: **addi 20 R1 R3**
line 5: **goto 7**
line 6: **addi 10 R1 R3**
line 7: **addi 5 R2 R4**
line 8: **store R4 address10**
line 9: **addi 30 R2 R5**
line 10: **store R5 address11**
line 11: **add R3 R2 R8**
line 12: **store R8 address12**

Time	Fetch	Decode	Operand Fetch	ALU exec.	Output Save	PC	Notes
1	1	X	X	X	X	1	
2	2	1	X	X	X	2	
3	3	2	1	X	X	3	
4	4	3	2	1	X	4	
5	4	3	X	2	1	4	line 3 waits for line 2 to finish.
6	4	3	X	X	2	4	
7	X	X	3	X	X	4	line 3 moves on. **if** detected. Stop fetching, flush line 4 from fetch step.
8	X	X	X	3	X	4	
9	X	X	X	X	3	4	

line 1: **load address2 R2**	line 6: **addi 10 R1 R3**	line 11: **add R3 R2 R8**
line 2: **load address1 R1**	line 7: **addi 5 R2 R4**	line 12: **store R8 address12**
line 3: **if R1 6**	line 8: **store R4 address10**	
line 4: **addi 20 R1 R3**	line 9: **addi 30 R2 R5**	
line 5: **goto 7**	line 10: **store R5 address11**	

Time	Fetch	Decode	Operand Fetch	ALU exec.	Output Save	PC	Notes
9	X	X	X	X	3	4	
10	4	X	X	X	X	4	**if** has finished, PC does **NOT** change.
11	5	4	X	X	X	5	
12	6	5	4	X	X	6	
13	X	X	5	4	X	X	**goto** detected. Stop fetching, flush line 6 from fetch step.
14	X	X	X	5	4	X	
15	X	X	X	X	5	X	
16	7	X	X	X	X	7	**goto** has finished, PC set to 7.
17	8	7	X	X	X	8	

line 1: **load address2 R2**	line 6: **addi 10 R1 R3**	line 11: **add R3 R2 R8**
line 2: **load address1 R1**	line 7: **addi 5 R2 R4**	line 12: **store R8 address12**
line 3: **if R1 6**	line 8: **store R4 address10**	
line 4: **addi 20 R1 R3**	line 9: **addi 30 R2 R5**	
line 5: **goto 7**	line 10: **store R5 address11**	

Time	Fetch	Decode	Operand Fetch	ALU exec.	Output Save	PC	Notes
17	8	7	X	X	X	8	
18	9	8	7	X	X	9	
19	9	8	X	7	X	9	line 8 waits for line 7 to finish.
20	9	8	X	X	7	9	
21	10	9	8	X	X	10	line 8 moves on.
22	11	10	9	8	X	11	
23	11	10	X	9	8	11	line 10 waits for line 9 to finish.
24	11	10	X	X	9	11	
25	12	11	10	X	X	12	line 10 moves on.

line 1: **load address2 R2**	line 6: **addi 10 R1 R3**	line 11: **add R3 R2 R8**
line 2: **load address1 R1**	line 7: **addi 5 R2 R4**	line 12: **store R8 address12**
line 3: **if R1 6**	line 8: **store R4 address10**	
line 4: **addi 20 R1 R3**	line 9: **addi 30 R2 R5**	
line 5: **goto 7**	line 10: **store R5 address11**	

Time	Fetch	Decode	Operand Fetch	ALU exec.	Output Save	PC	Notes
25	12	11	10	X	X	12	line 10 moves on.
26	X	12	11	10	X	X	no more instructions to fetch.
27	X	12	X	11	X	X	line 12 waits for line 11 to finish.
28	X	12	X	X	11	X	
29	X	X	12	X	X	X	line 12 moves on.
30	X	X	X	12	X	X	
31	X	X	X	X	12	X	
32							program execution has finished!

Reordering Instructions

```
line 1: load address2 R2
line 2: load address1 R1
line 3: if R1 6
line 4: addi 20 R1 R3
line 5: goto 7
line 6: addi 10 R1 R3
line 7: addi 5 R2 R4
line 8: store R4 address10
line 9: addi 30 R2 R5
line 10: store R5 address11
line 11: add R3 R2 R8
line 12: store R8 address12
```

- Reordering of instructions can be done by a compiler, as long as the compiler knows how instructions are executed.
- The goal of reordering is to obtain more efficient execution through the pipeline, by reducing dependencies.
- Obviously, reordering is not allowed to change the **meaning** of the program.
- What is the **meaning** of a program?

Meaning of a Program

```
line 1: load address2 R2
line 2: load address1 R1
line 3: if R1 6
line 4: addi 20 R1 R3
line 5: goto 7
line 6: addi 10 R1 R3
line 7: addi 5 R2 R4
line 8: store R4 address10
line 9: addi 30 R2 R5
line 10: store R5 address11
line 11: add R3 R2 R8
line 12: store R8 address12
```

- What is the **meaning** of a program?
- A program can be modeled mathematically as a function, that takes specific input and produces specific output.
- In this program, what is the input? Where is information stored that the program accesses?
- What is the output? What is information left behind by the program?

Meaning of a Program

line 1: **load address2 R2**

line 2: **load address1 R1**

line 3: **if R1 6**

line 4: **addi 20 R1 R3**

line 5: **goto 7**

line 6: **addi 10 R1 R3**

line 7: **addi 5 R2 R4**

line 8: **store R4 address10**

line 9: **addi 30 R2 R5**

line 10: **store R5 address11**

line 11: **add R3 R2 R8**

line 12: **store R8 address12**

- What is the **meaning** of a program?
- A program can be modeled mathematically as a function, that takes specific input and produces specific output.
- In this program, what is the input? Where is information stored that the program accesses?
  - address1 and address2.
- What is the output? What is information left behind by the program?
  - address10, address11, address12.

Reordering Instructions

line 1: **load address2 R2**
line 2: **load address1 R1**
line 3: **if R1 6**
line 4: **addi 20 R1 R3**
line 5: **goto 7**
line 6: **addi 10 R1 R3**
line 7: **addi 5 R2 R4**
line 8: **store R4 address10**
line 9: **addi 30 R2 R5**
line 10: **store R5 address11**
line 11: **add R3 R2 R8**
line 12: **store R8 address12**

- Reordering is not allowed to change the **meaning** of a program.
- Therefore, when given the same input as the original program, the re-ordered program must produce same output as the original program.
- Therefore, the re-ordered program must ALWAYS leave the same results as the original program on address10, address11, address12, as long as it starts with the same contents as the original program on address1 and address2.

Reordering Instructions

```
line 1: load address2 R2
line 2: load address1 R1
line 3: if R1 6
line 4: addi 20 R1 R3
line 5: goto 7
line 6: addi 10 R1 R3
line 7: addi 5 R2 R4
line 8: store R4 address10
line 9: addi 30 R2 R5
line 10: store R5 address11
line 11: add R3 R2 R8
line 12: store R8 address12
```

- Reordering of instructions can be done by a compiler, as long as the compiler knows how instructions are executed.
- How can we rearrange the order of instructions?
- Heuristic approach: when we find an instruction A that needs to wait on instruction B:
  - See if instruction B can be moved earlier.
  - See if some later instructions can be moved ahead of instruction A.

Reordering Instructions

```
line 1: load address2 R2
line 2: load address1 R1
line 3: if R1 6
line 4: addi 20 R1 R3
line 5: goto 7
line 6: addi 10 R1 R3
line 7: addi 5 R2 R4
line 8: store R4 address10
line 9: addi 30 R2 R5
line 10: store R5 address11
line 11: add R3 R2 R8
line 12: store R8 address12
```

- What is the first instruction that has to wait?
- What can we do for that case?

Reordering Instructions

```
line 1: load address2 R2
line 2: load address1 R1
line 3: if R1 6
line 4: addi 20 R1 R3
line 5: goto 7
line 6: addi 10 R1 R3
line 7: addi 5 R2 R4
line 8: store R4 address10
line 9: addi 30 R2 R5
line 10: store R5 address11
line 11: add R3 R2 R8
line 12: store R8 address12
```

- What is the first instruction that has to wait?
  - line 3 needs to wait on line 2.
- What can we do for that case?
  - Swap line 2 and line 1, so that line 2 happens earlier.

Reordering Instructions

line 1: **load address2 R2**
line 2: **load address1 R1**
line 3: **if R1 6**
line 4: **addi 20 R1 R3**
line 5: **goto 7**
line 6: **addi 10 R1 R3**
line 7: **addi 5 R2 R4**
line 8: **store R4 address10**
line 9: **addi 30 R2 R5**
line 10: **store R5 address11**
line 11: **add R3 R2 R8**
line 12: **store R8 address12**

- What is another instruction that has to wait?
- What can we do for that case?

Reordering Instructions

line 1: **load address2 R2**
line 2: **load address1 R1**
line 3: **if R1 6**
line 4: **addi 20 R1 R3**
line 5: **goto 7**
line 6: **addi 10 R1 R3**
line 7: **addi 5 R2 R4**
line 8: **store R4 address10**
line 9: **addi 30 R2 R5**
line 10: **store R5 address11**
line 11: **add R3 R2 R8**
line 12: **store R8 address12**

- What is another instruction that has to wait?
  - line 8 needs to wait on line 7.
- What can we do for that case?
  - We can move line 9 and line 11 ahead of line 8.

Result of Reordering

line 1: **load address2 R2**
line 2: **load address1 R1**
line 3: **if R1 6**
line 4: **addi 20 R1 R3**
line 5: **goto 7**
line 6: **addi 10 R1 R3**
line 7: **addi 5 R2 R4**
line 8: **store R4 address10**
line 9: **addi 30 R2 R5**
line 10: **store R5 address11**
line 11: **add R3 R2 R8**
line 12: **store R8 address12**

line 1 (old 2): **load address1 R1**
line 2 (old 1): **load address2 R2**
line 3 (old 3): **if R1 6**
line 4 (old 4): **addi 20 R1 R3**
line 5 (old 5): **goto 7**
line 6 (old 6): **addi 10 R1 R3**
line 7 (old 7): **addi 5 R2 R4**
line 8 (old 9): **addi 30 R2 R5**
line 9 (old 11): **add R3 R2 R8**
line 10 (old 8): **store R4 address10**
line 11 (old 10): **store R5 address11**
line 12 (old 12): **store R8 address12**

line 1: **load address1 R1**	line 6: **addi 10 R1 R3**	line 11: **store R5 address11**
line 2: **load address2 R2**	line 7: **addi 5 R2 R4**	line 12: **store R8 address12**
line 3: **if R1 6**	line 8: **addi 30 R2 R5**	
line 4: **addi 20 R1 R3**	line 9: **add R3 R2 R8**	
line 5: **goto 7**	line 10: **store R4 address10**	

Time	Fetch	Decode	Operand Fetch	ALU exec.	Output Save	PC	Notes
1	1	X	X	X	X	1	
2	2	1	X	X	X	2	
3	3	2	1	X	X	3	
4	4	3	2	1	X	4	
5	4	3	X	2	1	4	line 3 waits for line 1 to finish.
6	X	X	3	X	2	4	line 3 moves on. **if** detected. Stop fetching, flush line 4 from fetch step.
7	X	X	X	3	X	4	
8	X	X	X	X	3	4	
9	4	X	X	X	X	4	**if** has finished, PC does **NOT** change.

line 1: **load address1 R1**	line 6: **addi 10 R1 R3**	line 11: **store R5 address11**
line 2: **load address2 R2**	line 7: **addi 5 R2 R4**	line 12: **store R8 address12**
line 3: **if R1 6**	line 8: **addi 30 R2 R5**	
line 4: **addi 20 R1 R3**	line 9: **add R3 R2 R8**	
line 5: **goto 7**	line 10: **store R4 address10**	

Time	Fetch	Decode	Operand Fetch	ALU exec.	Output Save	PC	Notes
9	4	X	X	X	X	4	**if** has finished, PC does **NOT** change.
10	5	4	X	X	X	5	
11	6	5	4	X	X	6	
12	X	X	5	4	X	X	**goto** detected. Stop fetching, flush line 6 from fetch step.
13	X	X	X	5	X	X	
14	X	X	X	X	5	X	
15	7	X	X	X	X	7	**goto** has finished, PC set to 7.
16	8	7	X	X	X	8	
17	9	8	7	X	X	9	

line 1: **load address1 R1**	line 6: **addi 10 R1 R3**	line 11: **store R5 address11**
line 2: **load address2 R2**	line 7: **addi 5 R2 R4**	line 12: **store R8 address12**
line 3: **if R1 6**	line 8: **addi 30 R2 R5**	
line 4: **addi 20 R1 R3**	line 9: **add R3 R2 R8**	
line 5: **goto 7**	line 10: **store R4 address10**	

Time	Fetch	Decode	Operand Fetch	ALU exec.	Output Save	PC	Notes
17	9	8	7	X	X	9	
18	10	9	8	7	X	10	
19	11	10	9	8	7	11	
20	12	11	10	9	8	12	
21	X	12	11	10	9	X	
22	X	X	12	11	10	X	
23	X	X	X	12	11	X	
24	X	X	X	X	12	X	
25							program execution has finished!

Execution took 24 clock ticks.

Compare to 31 ticks for the original program.