DYNAMIC SCHEDULING

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Announcement

- Homework 4
  - Is due tonight 😊
  - Please read the clarifications in Canvas

- Homework 5
  - Will be uploaded tonight 11:59PM
  - Due date: Oct. 4th, 11:59PM
**Goal:** exploiting more ILP by avoiding stall cycles

- Branch prediction can avoid the stall cycles in the frontend
**Goal:** exploiting more ILP by avoiding stall cycles

- Branch prediction can avoid the stall cycles in the frontend
- More instructions are sent to the pipeline
Goal: exploiting more ILP by avoiding stall cycles

- Branch prediction can avoid the stall cycles in the frontend
  - More instructions are sent to the pipeline

- Instruction scheduling can remove unnecessary stall cycles in the execution/memory stage
  - Static scheduling
    - Complex software (compiler)
    - Unable to resolve all data hazards (no access to runtime details)
  - Dynamic scheduling
    - Completely done in hardware
Dynamic Scheduling

- **Key idea:** creating an instruction schedule based on runtime information
  - Hardware managed instruction reordering

Assembly code:

- `DIV F1, F2, F3` - Long latency operation
- `ADD F4, F1, F5` - Dependent instruction
- `SUB F6, F5, F7` - Independent instruction

Out-of-order execution?
Dynamic Scheduling

- **Key idea:** creating an instruction schedule based on runtime information
  
  - Hardware managed instruction reordering
  
  - Instructions are executed in data flow order

Program code:

```
ADDI  R1, R0, #1
ADDI  R2, R0, #4
ADD   R3, R3, R2
ADD   R2, R2, #-1
BNEQ R2, R1, next
ADD   R4, R4, R3
BNEQ R2, R0, loop
```

Data flow:

```
ADDI  R1, R0, #1
ADDI  R2, R0, #4
ADD   R3, R3, R2
ADD   R2, R2, #-1
BNEQ R2, R1, next
ADD   R4, R4, R3
ADD   R3, R3, R2
ADD   R2, R2, #-1
ADD   R3, R3, R2
ADD   R2, R2, #-1
ADD   R3, R3, R2
ADD   R2, R2, #-1
ADD   R3, R3, R2
ADD   R2, R2, #-1
ADD   R4, R4, R3
ADD   R4, R4, R3
```

How to form data flow graph on the fly?
Register Renaming

- Eliminating WAR and WAW hazards
  - Change the mapping between architectural registers and physical storage locations

WAR and WAW hazards can be removed using more registers
Register Renaming

- Eliminating WAR and WAW hazards
  - 1. allocate a free physical location for the new register
  - 2. find the most recently allocated location for the register

DIV F1, F2, F3
ADD F4, F1, F5
SUB F5, F6, F7
ADD F4, F5, F8

DIV P12, P11, P13

Architectural Registers
- F1
- F2
- F3
- F4
- F5
- F6
- F7
- F8

Physical Locations
- P10
- P11
- P12
- P13
- P14
- P15
- P16
- P17
- P18
- P19
Register Renaming

- Eliminating WAR and WAW hazards
  1. allocate a free physical location for the new register
  2. find the most recently allocated location for the register

### Instruction Set

- DIV F1, F2, F3
- ADD F4, F1, F5
- SUB F5, F6, F7
- ADD F4, F5, F8

- DIV P12, P11, P13
- ADD P14, P12, P15
Register Renaming

- Eliminating WAR and WAW hazards
  - 1. allocate a free physical location for the new register
  - 2. find the most recently allocated location for the register

DIV    F1, F2, F3  
ADD    F4, F1, F5  
SUB    F5, F6, F7  
ADD    F4, F5, F8

DIV    P12, P11, P13  
ADD    P14, P12, P15  
SUB    P19, P17, P13

Architectural Registers
- F1  
- F2  
- F3  
- F4  
- F5  
- F6  
- F7  
- F8

Physical Locations
- P10  
- P11  
- P12  
- P13  
- P14  
- P15  
- P16  
- P17  
- P18  
- P19
Register Renaming

- Eliminating WAR and WAW hazards
  - 1. allocate a free physical location for the new register
  - 2. find the most recently allocated location for the register

DIV F1, F2, F3
ADD F4, F1, F5
SUB F5, F6, F7
ADD F4, F5, F8

DIV P12, P11, P13
ADD P14, P12, P15
SUB P19, P17, P13
ADD P18, P19, P16

Architectural Registers
- F1
- F2
- F3
- F4
- F5
- F6
- F7
- F8

Physical Locations
- P10
- P11
- P12
- P13
- P14
- P15
- P16
- P17
- P18
- P19