COSC4201
Instruction Level Parallelism
Dynamic Scheduling

Prof. Mokhtar Aboelaze

Parts of these slides are taken from Notes by
Prof. David Patterson (UCB)

Outline

- Data dependence and hazards
- Exposing parallelism (loop unrolling and scheduling)
- Reducing branch costs (prediction)
- Dynamic scheduling
- Speculation
- Multiple issue and static scheduling
- Advanced techniques
- Example

Introduction

- In dynamic scheduling, the hardware rearranges the instruction execution to reduce stalls.
- Can handle cases where dependence is not known during compile time (memory reference).
- Simplifies the compiler
- Can tolerate unpredictable cases (cache miss).
- Speculation is based on dynamic scheduling.
- Can allow code compiled and for another pipeline to run efficiently on any pipeline

Dynamic Scheduling – The idea

```
DIVD   F0,F2,F4
ADDD   F10,F0,F8
SUBD   F12,F8,F14
```

- SUB can not be issued because of the dependence of ADD on DIV, although there is no data dependence to prevent issuing it.
- In a classical pipeline, structural and data hazard are checked in the ID stage.
- Here we must separate between checking for structural hazards, and waiting for the absence of data hazard.
- In-order issue, but instructions starts executions soon as its data operands are available.

Dynamic Scheduling – The Idea

- **Will distinguish when an instruction *begins execution* and when it *completes execution*; between 2 times, the instruction is *in execution***
- **May create WAR and WAW hazards**

```
DIV.D  F0,F2,F4
ADD.D  F6,F0,F8
SUB.D  F8,F10,F16
MUL.D  F6,F10,F8
```

- **Exceptions?**

Dynamic Scheduling – The Idea

- **The ID stage is split into two stages**
  - Issue **– Decode and check for structural hazards**
  - Read operands **– Wait until there is no dat hazards, then read operand**
- **Before the ID stage, there is an instruction fetch stage that can fetch in a queue or a register.**
- **All instructions pass through the** issue **stage in order**

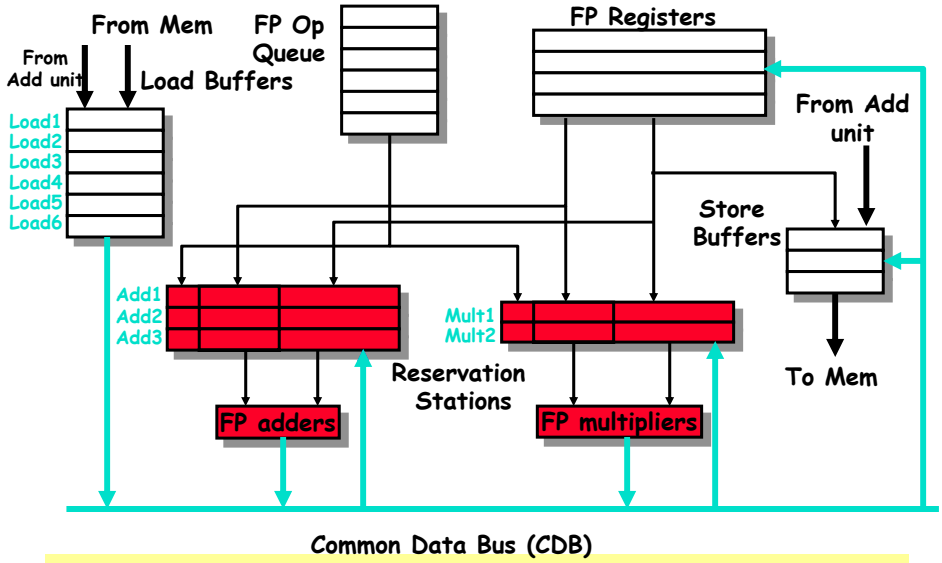
Tomasulo's Algorithm

- Control & buffers distributed with Function Units (FU)
  - FU buffers called "reservation stations"; have pending operands
- Registers in instructions replaced by values or pointers to reservation stations(RS); called register renaming ;
  - Renaming avoids WAR, WAW hazards
  - More reservation stations than registers, so can do optimizations compilers can't
- Results to FU from RS, not through registers, over Common Data Bus that broadcasts results to all FUs
  - Avoids RAW hazards by executing an instruction only when its operands are available
- Load and Stores treated as FUs with RSs as well
- Integer instructions can go past branches (predict taken), allowing FP ops beyond basic block in FP queue

Tomasulo's Algorithm

- RS fetches operands as soon as they are available (no need to read them froma register).
- Pending instructions eliminates.
- Pending instructions designate the RS's that will provide their inputs.
- When successive writes to the same register, only the last one is actually written to the register.
- Common data bus CDB is used to bypass registers in operand fetching.

Tomasulo's algorithm in MIPS



Reservation Station

Op: Operation to perform in the unit (e.g., + or –)

Vj, Vk: Value of Source operands

- Store buffers has V field, result to be stored

Qj, Qk: Reservation stations producing source registers (value to be written)

- Note: Qj,Qk=0 => ready
- Store buffers only have Qi for RS producing result

Busy: Indicates reservation station or FU is busy

Register result status—Indicates which functional unit will write each register, if one exists. Blank when no pending instructions that will write that register.

Tomasulo's Algorithm

° The load and store buffers each have a field A which holds the results of the effective address (initially, the immediate field is stored).

Tomasulo's Algorithm

Stages

1. **Issue—get instruction from FP Op Queue**

   If reservation station free (no structural hazard), control issues instr & sends operands (renames registers).

2. **Execute—operate on operands (EX)**

   When both operands ready then execute; if not ready, watch Common Data Bus for result

3. **Write result—finish execution (WB)**

   Write on Common Data Bus to all awaiting units; mark reservation station available

° Normal data bus: data + destination ("go to" bus)

° Common data bus: data + source ("come from" bus)

- 64 bits of data + 4 bits of Functional Unit source address
- Write if matches expected Functional Unit (produces result)
- Does the broadcast

Tomasulo's Algorithm

- Loads and stores require two-step execution (calculating effective address and load/store).
- Loads and stores are kept in program order
- To preserve exception behavior, no instruction is allowed to initiate execution until all branches that precede it in program order have completed.