LECTURE - 13

Dynamic Scheduling

- Better than static scheduling
- Scoreboarding:
  - Used by the CDC 6600
  - Useful only within basic block
  - WAW and WAR stalls
- Tomasulo algorithm:
  - Used in IBM 360/91 for the FP unit
  - Main additional feature: register renaming to avoid WAR and WAW stalls

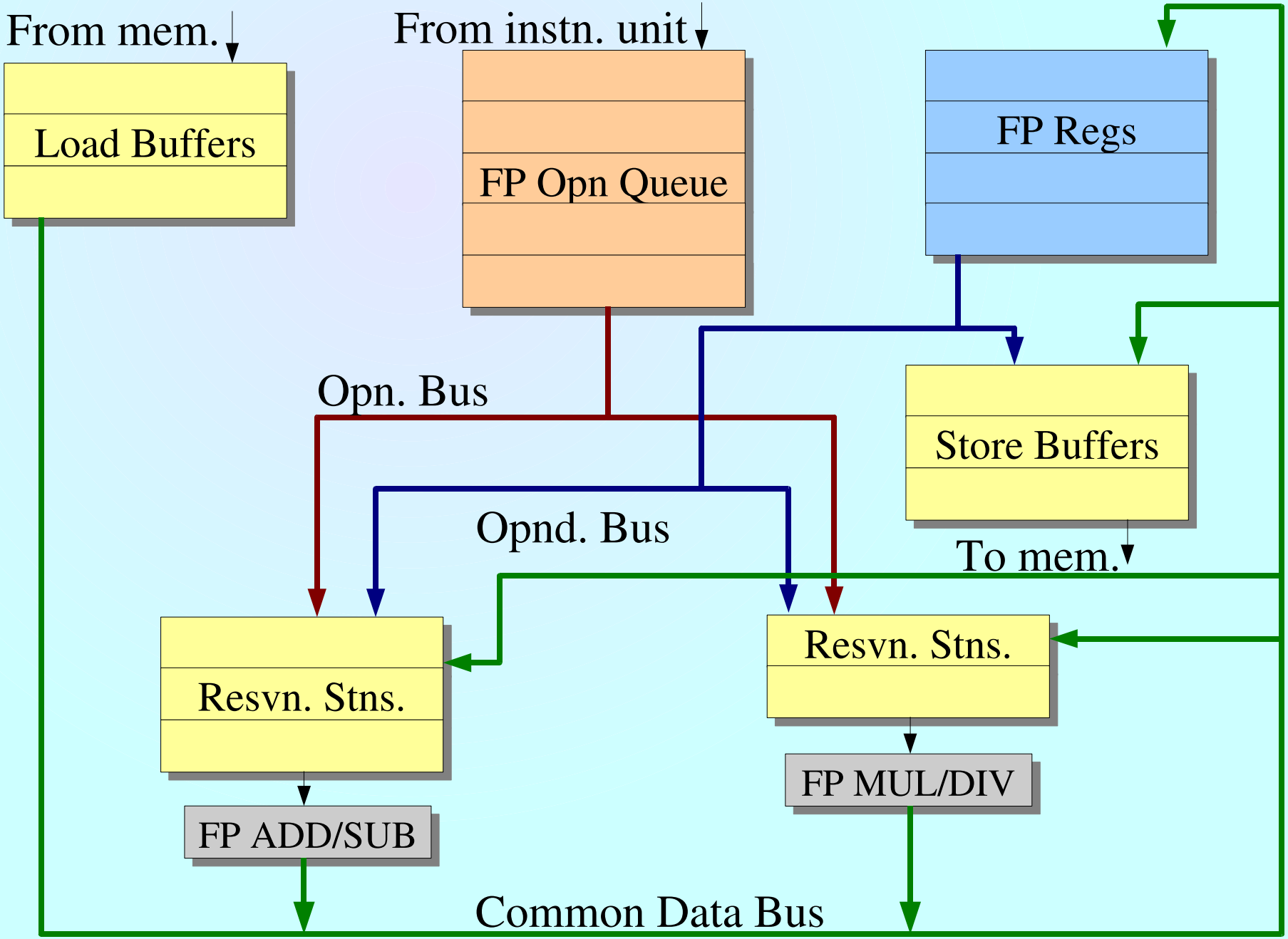
Register Renaming: Basic Idea

- Compiler maps memory --> registers statically
- Register renaming maps registers --> virtual registers in hardware, dynamically
- Should keep track of this mapping
  - Make sure to read the current value
- Num. virtual registers > Num. ISA registers usually
- Virtual registers are known as *reservation stations* in the IBM 360/91

Tomasulo: Main Architectural Features

- Reservation stations: fetch and buffer operand as soon as it is available
- Load/store buffers: have the address (and data for store) to be loaded/stored
- Distributed hazard detection and execution control
- Common Data Bus (CDB): results passed from where generated to where needed
- Note: IBM 360/91 also had reg-mem instns.

The Tomasulo Architecture



Pipeline Stages

- Issue:
  - Wait for free Reservation Station (RS) or load/store buffer, and place instruction there
  - Rename registers in the process (WAR and WAW handled here)
- Execute (EX):
  - Monitor CDB for required operand
  - Checks for RAW hazard in this process
- Write Result (WB):
  - Write to CDB
  - Picked up by any RS, store buffer, or register

Register Renaming

- In RS, operands referred to by a tag (if operand not already in a register)
- The tag refers to the RS (which contains the instruction) which will produce the required operand
- Thus each RS acts as a virtual register

The Data Structure

- Three parts, like in the scoreboard:
  - Instruction status
  - Reservation stations, Load/Store buffers, Register file
  - Register status: which unit is going to produce the register value
    - This is the register --> virtual register mapping

Components of RS, Reg. File, Load/Store Buffers

- Each RS has:
  - Op: the operation (+, -, x, /)
  - Vj, Vk: the operands (if available)
  - Qj, Qk: the RS tag producing Vj/Vk (0 if Vj/Vk known)
  - Busy: is RS busy?
- Each reg. in reg. file and store buffer has:
  - Qi: tag of RS whose result should go to the reg. or the mem. locn. (blank ==> no such active RS)
- Load and store buffers have:
  - Busy field, store buffer has value V to be stored

Maintaining the Data Structure

- Issue:
  - Wait until: RS or buffer empty
  - Updates: Qj, Qk, Vj, Vk, Busy of RS/buffer; Maintain register mapping (register status)
- Execute:
  - Wait until: Qj=0 and Qk=0 (operands available)
- Write result:
  - CDB result picked up by RS (update Qj, Qk, Vj, Vk), store buffers (update Qi, V), register file (update register status)
  - Update Busy of the RS which finished

Some Examples

- Randy Katz's CS252 slides... (Lecture 11, Spring 1996)
- Dynamic loop unrolling example from text

Dynamic Loop Unrolling

```
Loop: LD    F0, 0(R1)  // F0 is array element
      ADDD  F4, F0, F2// F2 has the scalar 'C'
      SD    0(R1), F4  // Stored result
      SUBI  R1, R1, 8  // For next iteration
      BNEZ  R1, Loop  // More iterations?
```

- Assume branch predicted to be taken
- Denote: load buffers as L1, L2..., ADDD RSs as A1, A2...
- First loop: F0 --> L1, F4 --> A1
- Second loop: F0 --> L2, F4 --> A2

Summary Remarks

- Memory disambiguation required
- Drawbacks of Tomasulo:
  - Large amount of hardware
  - Complex control logic
  - CDB is performance bottleneck
- But:
  - Required if designing for an old ISA
  - Multiple issue ==> register renaming and dynamic scheduling required
- Next class: branch prediction