Superscalar Issue Logic
Instruction Issue Logic
(Sohi & Vajapeyam, 1987)

- After instructions are fetched, decoded and renamed, they are placed in instruction buffers where they wait until issue

- An instruction can be issued when its input operands are ready, and there is a functional unit available
Issue Logic Operation

All out-of-order issue methods must handle the same basic steps:

- Identify all instructions that are ready to issue
- Select among ready instructions to issue as many as possible
- Issue the selected instructions, e.g., pass operands and other information to the functional units
- Reclaim instruction window storage used by the now issued instructions
Methods of Organizing Instruction Issue Buffers

- Single shared queue
  - Only for in-order issue
- Multiple queues, one per instruction type
- Multiple reservation stations, one per instruction type
- Single central reservation stations buffer

Tomasulo’s Algorithm

- Based on a technique used in the IBM 360/91 floating point execution unit

- Dispatch:
  - an instruction is read from the instruction queue and sent into an empty reservation station
  - The operands or tags (if the operands are not available) are read from the register file and written into the operand fields of the reservation station
  - Tags identify functional unit that will write a register
Tomasulo’s Algorithm (Cont.)

- Execute:
  - If both operands are available, execute the instruction
  - If one or more of the operands is not available, monitor the result writeback bus until the operand is computed and broadcast

- Result writeback:
  - When a result is computed, broadcast it with its tag on the writeback bus
  - If the tag in its destination register matches, it is written into the register file, and all reservation stations that need it also grab it
Structure of FP unit using Tomasulo’s Algorithm
Central Window Example

- The Register Update Unit (RUU) combines the reservation stations and reorder buffer units
- Allocation and removal is in order
- Instructions may stay in reservation stations longer than necessary
- See Figures 2, 3 and 4 in the Sohi & Vajapeyam paper
Central Window (Cont.)

- Note the continuing importance of the reorder buffer's ability to maintain program order
  - Exception recovery
  - Branch misprediction recovery
  - Memory accesses
  - Central window simplification
Complexity-Effective Superscalar Processors (Palacharla et al., 1997)

- Tradeoff between complexity and speed
  - Pipeline consists of many stages with different functions (Paper Figure 1)
  - Clock speed is determined by slowest, most complex stage
    - Unless stage is split or work is redistributed to other stages (discuss overhead)
    - A good design is balanced: No single stage is the only bottleneck
  - To achieve high IPC, we need larger instruction window which increases complexity
    - Remember execution time equation
Structures with High Complexity

- Structures that get more complex with larger issue width or larger instruction window
  - Register Rename Logic
    - Translates logical registers to physical registers
  - Wakeup Logic
    - Wakes up instructions waiting for their source operands
  - Selection Logic
    - Selects instructions for execution from pool of ready instructions
  - Bypass Logic
    - Bypass operand values from instructions that finished execution but not write back to subsequent instructions
Register Rename Logic

- Translates logical to physical registers by accessing a map table indexed by logical register
- Map table is multi-ported
  - Multiple instructions (each with one or more register operands) need renaming every cycle
- Block diagram (paper figure 2)
- Dependence check logic needed
  - Detects whether logical register being renamed is written by an earlier instruction in the current group being renamed
  - Sets up output MUXes to select appropriate physical registers
Register Rename Logic Implementations

- RAM Scheme (e.g., MIPS R10000)
  - Logical register used to index table, corresponding entry contains current physical register mapping
  - Number of entries = number of logical registers

- CAM Scheme (e.g., DEC 21264)
  - Number of entries = number of physical registers
  - Each entry contains current logical register mapped to this physical register, and a valid bit
  - Renaming done by CAM matching on logical register field

- Dependence check logic done in parallel, has less latency that could be hidden behind map table access

- Rename delay vs. issue width: Paper figure 3
Wakeup Logic

- Responsible for updating source dependences for instructions in issue window waiting for source operands to become available (Paper figure 4)

- **Operation**
  - When a result is produced, result tag is broadcast to all instructions in issue window
  - Each instruction compares tag to its source operands
  - If match, the operand is marked as available by setting appropriate flag (rdyL or rdyR)
  - Once both operands of an instruction are available, the ready flag is set (instruction is ready to execute)

- Issue window is a CAM, buffers drive the result tags, each CAM entry has 2xIW comparators to compare each of the result tags against the two operand tags of the entry

- **Wakeup logic delay**: Paper figure 5, 6
Selection Logic

- Responsible for choosing instructions for execution from the pool of ready instructions
- Inputs: request signals, one per instruction that is set when wakeup logic detects all operands are ready
- Outputs: grant signals, one per request signal that allows the instruction to be issued to the functional unit
- Selection policy decides which requesting instruction to grant (e.g., oldest first)
- Structure: A tree of arbiters that works in two phases (Figure 7)
  - First phase: request signals propagate up the tree, root detects if any instruction is ready, root grants functional unit to one of its children
  - Second phase: grant signal propagates down the tree to the selected instruction
- Selection delay: Figure 8
Data Bypass Logic

- Responsible for forwarding results from completing instructions to dependent instructions, bypassing register file
- Number of bypass paths depends on pipeline depth and issue width
  - Assuming 2-input functional units, IW issue width, S pipe stages after first result-producing stage, we need $2 \times IW^2 \times S$ paths
- Two components of data bypass logic: Datapath and control logic (Paper figure 9)
Data Bypass Logic (Cont.)

- **Datapath:**
  - Result busses used to broadcast bypass values from each functional unit source to all possible destinations
  - Buffers used to drive bypass values on result busses

- **Control logic:**
  - Controls operand MUXes
  - Compares tags of result values with tags of source value required at each functional unit
  - Sets appropriate MUX control signals on match

- Delay is dominated by datapath not control

- Bypass delays: Paper Table 1
- Summary of delay results: Paper Table 2
Reading Assignment

- HW1 and project proposals due on Thursday before class