CS4100: Computer System Design

Dynamic Scheduling

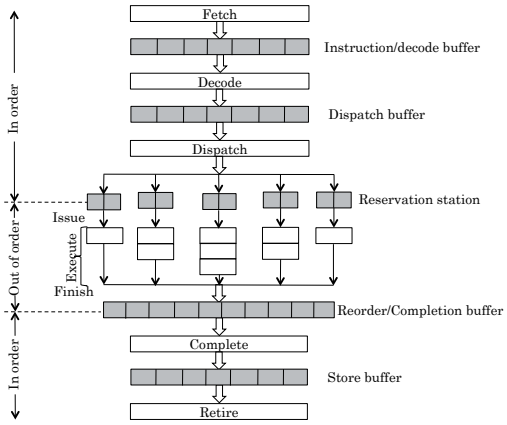
Madhu Mutyam
PACE Laboratory
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Oct 5, 2015

Dynamic Scheduling

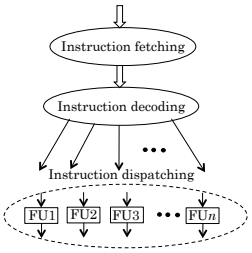
- In-order instruction issue and execution can degrade the performance
  - Stalled instruction in the pipeline prevent subsequent instructions to proceed
- Eliminate false dependences through register renaming
- Rearrange the instruction execution at runtime to reduce the stalls while maintaining *data flow* and *exception behaviour*
- Advantages:
  - Allows code compiled for one microarchitecture to run efficiently on different microarchitecture
  - Handles dependences that may be unknown at compile time
  - Allows the processor to tolerate unpredictable delays

Superscalar Organization



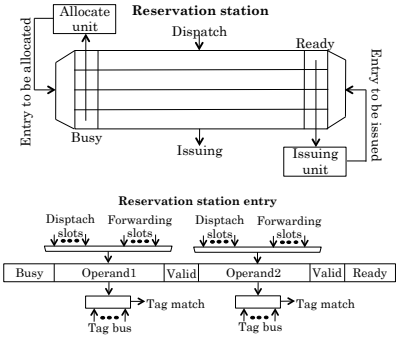
Instruction Dispatch

- Route decoded instructions to appropriate functional units



Reservation Station

- Reservation station decouples instruction decoding and instruction execution
- Main tasks: *Dispatching – Waiting – Issuing*



- Checks structural and RAW hazards

Reservation Station: Centralized Vs Distributed

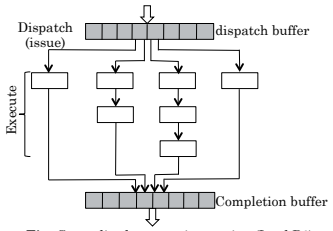


Fig: Centralized reservation station (Intel P6)

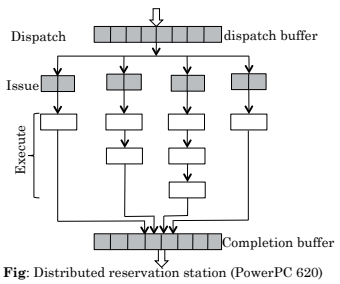
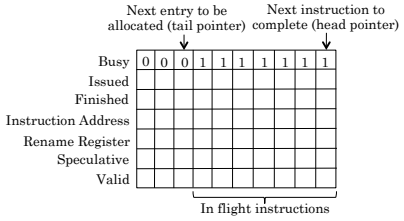


Fig: Distributed reservation station (PowerPC 620)

- Entries can be utilised effectively in centralized reservation station, while distributed reservation stations can be single-ported

Reorder Buffer

- Contains all *in-flight* instructions
  - Include instructions in RS + instructions executing in FUs + instructions which are finished execution but waiting to be completed in program order
- Only finished and non-speculative instructions can be completed



Instruction Completion and Retire

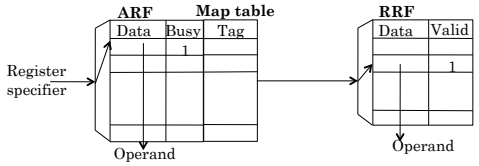
- *Completion* – finish the execution and update the machine state
- *Retire* – update the memory
  - A store may complete by writing to store buffer, but it retires only when the data is written to the memory
- When an interrupt occurs, stop fetching new instructions and finish the execution of all in-flight instructions
- When an exception occurs, the result of the computation may no longer be valid
  - *Precise exception* – checkpoint the state of the machine just prior to the execution of an excepting instruction and resume execution by restoring the checkpointed state
  - Precise exception is handled by the instruction completion stage using ROB

Register Reuse

- Static reuse occurs due to compiler optimizations
  - Code generator assumes the availability of unlimited number of symbolic registers
  - Register allocation tool maps the unlimited number of symbolic registers to the limited number of architectural registers provided by the ISA
- Dynamic reuse occurs when a loop of instructions is executed
  - Multiple iterations of a loop can be simultaneously in-flight
- Register reuse can induce name dependencies

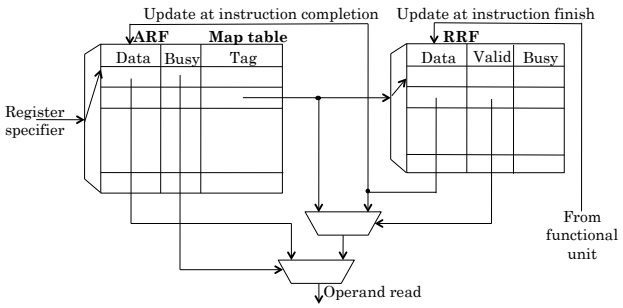
Enforcing Name Dependencies

- Option #1: Stall all dependent instructions until the leading instruction has finished accessing the dependent register
- Option #2: Register renaming
  - Dynamically assign different names to the multiple writings of an architectural register
- Use a separate rename register file (RRF) in addition to the architectural register file (ARF)

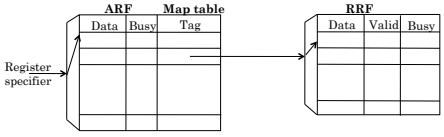


Register Renaming Tasks

- Destination allocate
- Register update
- Source read

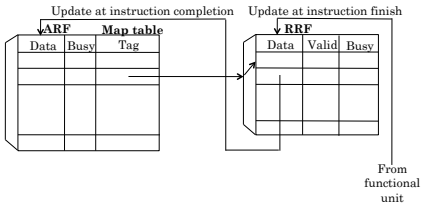


Register Renaming Task #1: Destination Allocate



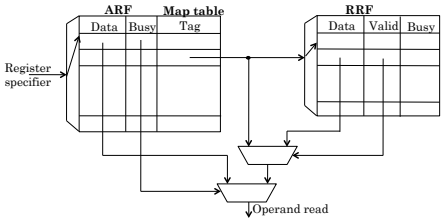
- Find a register in RRF with RRF.Busy = 0
- Set RRF.Busy = 1 and RRF.Valid = 0
- Update the map table with the rename register tag
- Set ARF.Busy = 1

Register Renaming Task #2: Register Update



- When an instruction finishes its execution
  - Write the result into the entry of RRF indicated by the tag
  - Set RRF.Valid = 1
- When the instruction is completed
  - Update the architectural register with the data
  - ARF.Busy = RRF.Busy = 0

Register Renaming Task #3: Source Read



- If ARF.Busy = 0 $\Rightarrow$ instruction execution is completed
  - Fetch the operand from the ARF
- If ARF.Busy = 1 $\Rightarrow$ the content is stale
  - Access the map table to retrieve the rename register tag
  - If RRF.Valid = 1 $\Rightarrow$ instruction execution is finished
    - Fetch the operand from the RRF
  - If RRF.Valid = 0 $\Rightarrow$ instruction is not executed
    - Forward the rename register tag from the map table to the reservation station

Register Renaming Example

Original instructions	
ADD	R1, R2, R3
SUB	R3, R2, R1
MUL	R1, R2, R3
DIV	R2, R1, R3

- Architectural registers: R1, R2, R3
- True and false dependences between instructions
- Rename registers: R4, R5, R6, R7

Free rename registers	Renamed instructions	
R4, R5, R6, R7		
R5, R6, R7	ADD	R4, R2, R3
R6, R7	SUB	R5, R2, R4
R7	MUL	R6, R2, R5
	DIV	R7, R6, R5

Register renaming resolves all fasle dependencies

Thank You