CS 3725 Feb 9/12

## Detecting Register Availability at the Hardware Level (Register Conflict Data Dependency)

Idea Associate counter Ci with register Ri.

We manipulate these counters in such a way that we maintain the following invariants

Invariants

① $C_i = 1$ iff Ri is being used (currently) by our instruction as an output register.

② $C_i = \emptyset$ iff Ri is free
(ie. Ri is not being used either as input or as output register by any currently executing register)

③ $C_i < \emptyset$ iff Ri is used as input register by the currently executing instruction.

## Detection of D-Unit Dependency and Register Dependency at the Hardware Level

Assume: $Sm: R_i \leftarrow Dn(R_j, R_k)$

① Test (Before issuing instruction)

- $b_h = \emptyset$? (ie. Dn is free?)
- $C_i = \emptyset$? (ie. $R_i$ is free?)

  In other words, $R_i$ is not being used either as an input or as an output register by any of the currently executing instructions.
- $C_j \leq \emptyset$? } ie. $R_j$ and $R_k$ are either
- $C_k \leq \emptyset$? } free or only being used as a input register by any of the current instructions

Set (Before issuing instruction)

- $b_h \leftarrow 1$ } Reserve $D_h$
- $C_i \leftarrow 1$ } $R_i$ is used as output
- $C_j \leftarrow C_j - 1$ } $R_j$ and $R_k$ used as inputs
- $C_k \leftarrow C_k - 1$ }

Issue

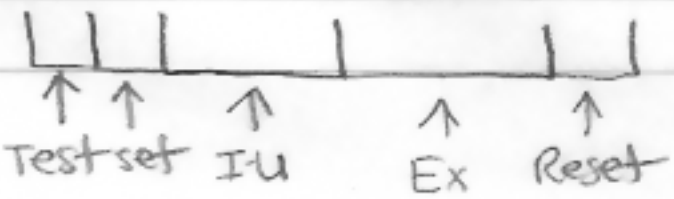
...

Execute

...

Reset (At the end of execution)

- $b_h \leftarrow \emptyset$ } Release $D_h$
- $C_i \leftarrow \emptyset$ } Release the output register
- $C_j \leftarrow C_j + 1$ } one less instruction
- $C_k \leftarrow C_k + 1$ } using $R_j$ and $R_k$ as inputted register.



How to Alleviate Bottlenecks due to:
① D-Unit Dependency
② Register Dependency

How to Alleviate Instruction Issuing Bottleneck due to D-unit dependency.
Solution ① More D-units (expensive!)
② Virtual D-units

Virtual D-Units
Idea ① With each D-Unit we maintain a FIFO buffer (Instruction Buffer)

② When issuing an instruction to $D_h$, we place the instruction in the buffer $B_L$ assuming there is an empty slot

③ $D_h$ will execute instructions in the order they were sent - ie. FIFO.

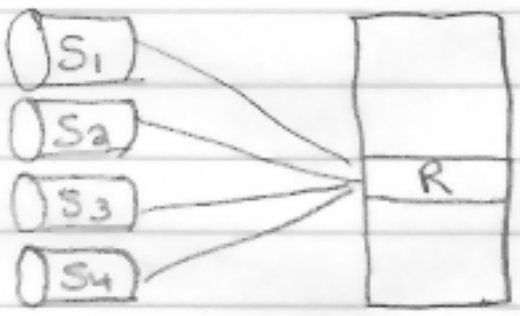
④ We maintain a counter $C_h$ (not as a flag as we did for non-virtual D-unit)
a) Initially $C_h$ = the max size of the buffer
b) Everytime an instruction is issued to $B_h$, $C_h \leftarrow C_h - 1$
c) Instruction is issued to $B_h$ iff $C_h > \emptyset$ (ie there is an empty slot)
d) After execution $C_h \leftarrow C_h + 1$

## How to Alleviate Instruction Issuing Bottleneck Due to Register Conflict

$S_i$ : $I_i \rightarrow O_i$
$S_j$ : $I_j \rightarrow O_j$

Three types of Data Dependencies
① IO
② OI
③ OO

ex: $S_1$ : R ← P+Q
$S_2$ : T ← R+S
$S_3$ : R ← X·Y
$S_4$ : A ← R+B




Register Renaming (It removes OO dependency and OI dependency among instructions)

| Original Code | | After Register Rename |
|---|---|---|
| $S_1$ : | R ← P+Q ] OI | $R_1$ ← P+Q ] OI |
| $S_2$ : | OO [ T ← R+S ] IO | T ← $R_1$ + S |
| $S_3$ : | [ R ← X*Y ] | $R_2$ ← X*Y |
| $S_4$ : | A ← R+Z | A ← $R_2$ + Z |