CS 3725 Feb. 16/12

Superscaler Machines

Prelude: All the ideas related to removing/alleviating bottlenecks due to D-unit dependency (multiple d-units, virtual d-units)

IO
OO } register renaming
OI } score boarding in conjunction with conditional issuing of instructions

Idea
- In order issue, out of order execution
- Issue one inst at a time

i1 (scalar)
i2 | Iu → MIPS
i3 | Iu
i4 | Iu
| Iu

ie. There is no issuing bottleneck due to dependency

D-unit takes ∅ time (Not in reality)

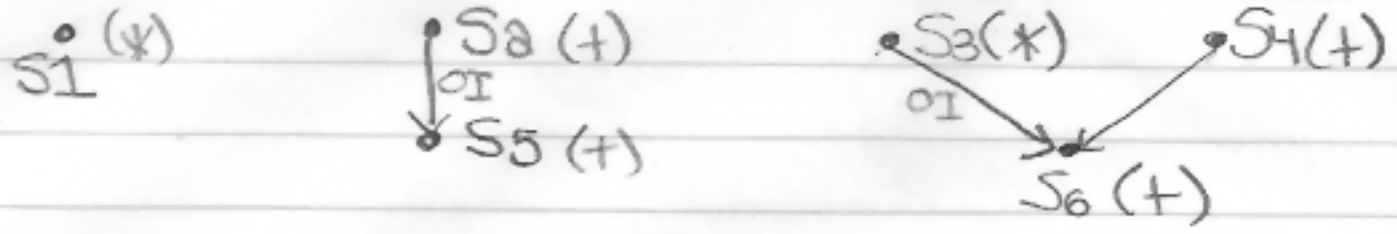
Superscaler

i1 | Iu
i2 | Iu
i3 | Iu

The instructions are issued in parallel. Some may be conditionally.

Ex: Consider the program segment P (done before), after Register Renaming

Precedence Log

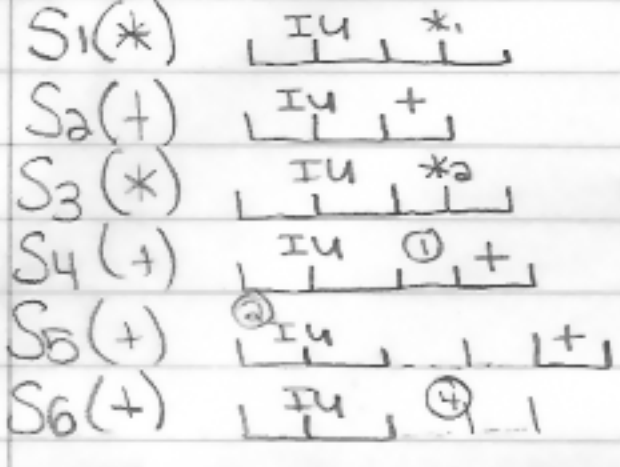


Mi: Virtual +-er with buffer size 2
Two multipliers $*_1$ $*_2$
+ time = 1
* time = 2
Iu time = 2
No score board.

S1 (*)	Iu	Iu	*	*						
Sa (+)	Iu	Iu	+							
S3 (*)	Iu	Iu	*	*						
S4 (+)	Iu	Iu	①	+						
S5 (+)	②	②	③	Iu	Iu	+				
S6 (+)	④	④	④	④	Iu	Iu	+			

① S4 is issued to the available buffer in the +er
② S5 cannot be issued this time because of an OI dependency with Sa
③ Since Sa is complete which makes a buffer in + available
④ Has to wait for ⊕ buffer to be free

M_3: Virtual t-er with buffer size 2
Two multiplyers $*^1$ $*^2$
+ time = 1
* time = 2
Iu time = 2
Same as M_1 except it has a scoreboard.



② S_5 is conditionally issued to the scoreboard
③ Even though the input of S_5 is available, it will only be executed when t-er finishes its current inst S_4
④ S_5 gets all its input values whes it goes through the scoreboard

Q: Where do we find parallelism in our programs?

- How do we package them?
- What kind of architecture would be suitable?

Consider the following code:

```
FOR I = 1 To 100 DO
    A[I] ← B[I] + C[I]
```

Sequential machine vs Array Processor (Vector Processor)

In sequential processor:
- 100 Inst Mem Fetching
- 3 calculations for 3 operand addresses
  → Total 300 address calculations
- 300 data memory fetch (read/write)
- 100 additions
- 100 computations of branch target address
- 100 branch operations
- 100 increments of index (add operation)

Equivalently

```
A[1] = B[1] + C[1]
A[2] = B[2] + C[2]
   ...
A[100] = B[100] + C[100]
```

All independent of each other.

Inherent (logical) parallelism
And a machine which will explat that parallelism.

Vector Languages
- A[1...100] ← B[1...100] + C[1...100]
- A ← B + C