06 Performance Optimization

06.04 Dynamic optimization

- Out-of-order execution
- Speculation
- Sub-clock-cycle operations

Dynamic optimization

- Out of order (OOO) execution
  - Dynamic scheduling (Tomasulo's algorithm)
- Speculation
  - Tentative decisions on conditional branches with unknown conditions
- Sub-clock-cycle operations
  - More than one instruction processed per clock cycle

Data dependencies (1)

- (RAW) Read after write
  - The second instruction reads a register written by the first one
- (WAR) Write after read
  - The first instruction reads the content of a register before it is overwritten by the second one
- (WAW) Write after write
  - Both instructions write on the same register

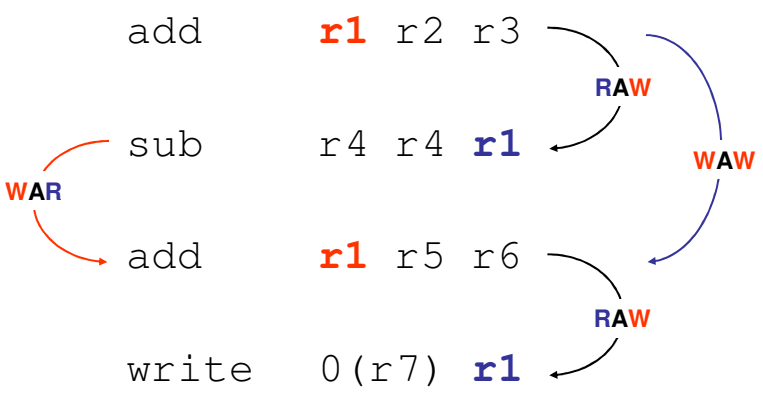
```
add r1 r2 r3
sub r4 r4 r1
```

```
load r1 0(r2)
bnz r1 label
```

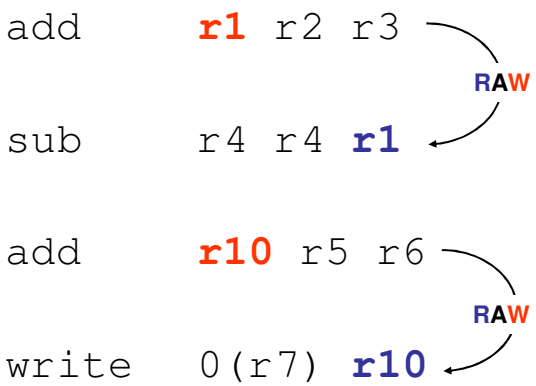
```
add r3 r2 r1
load r1 0(r4)
```

```
add r1 r2 r3
load r1 0(r4)
```

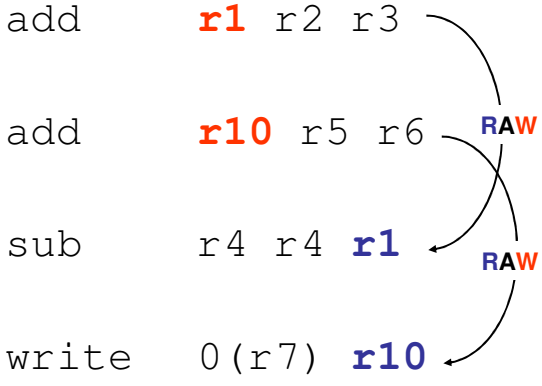

Data dependencies (2)



Data dependencies (3)



Data dependencies (4)



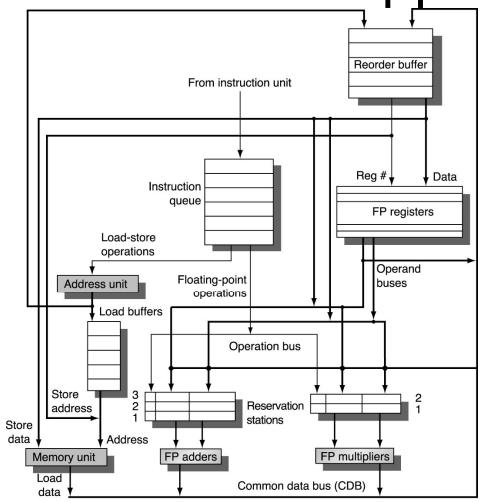
OOO execution opportunities

- Give priority to ready-to-execute instructions
  - Enqueu instructions on resource-specific lanes
- Avoid false data dependences
  - Use temporary registers avoid WAR and WAW
- Forward partial results to all execution units
  - Use a shared internal bus for forwarding
- Preserve sequential coherence when writing on the register file

OOO execution support (1)

- Instruction decode and register fetch in separate phases
- Instruction queue
- *Reservation stations*
- *Common data bus* (CDB)
- *Reorder buffer* (ROB)
- The above-mentioned components provide a hardware implementation of Tomasulo's algorithm for dynamic scheduling

OOO execution support (2)



OOO execution phases

1. *Issue*
   - Get an instruction from the instruction queue and put it in the reservation station
2. *Execute*
   - When the resource is available and the operands are available at the reservation station, execute the instruction
3. *Write results*
   - Send the result (through the CDB) to the ROB and to all reservation stations waiting for it
4. *Commit*
   - Move the results from the head of the ROB to either the register file or the memory

OOO execution example

```
L.D     F6,34(R2)
L.D     F2,45(R3)
MUL.D   F0,F2,F4
SUB.D   F8,F6,F2
DIV.D   F10,F0,F6
ADD.D   F6,F8,F2
```

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Load1	no							
Load2	no							
Add1	no							
Add2	no							
Add3	no							
Mult1	no	MUL.D	Mem[45+Regs[R3]]	Regs[F4]			#3	
Mult2	yes	DIV.D		Mem[34+Regs[R2]]	#3		#5	

Reorder buffer

Entry	Busy	Instruction	State	Destination	Value
1	no	L.D F6,34(R2)	Commit	F6	Mem[34+Regs[R2]]
2	no	L.D F2,45(R3)	Commit	F2	Mem[45+Regs[R3]]
3	yes	MUL.D F0,F2,F4	Write result	F0	#2 x Regs[F4]
4	yes	SUB.D F8,F6,F2	Write result	F8	#1 – #2
5	yes	DIV.D F10,F0,F6	Execute	F10	
6	yes	ADD.D F6,F8,F2	Write result	F6	#4 + #2

FP register status

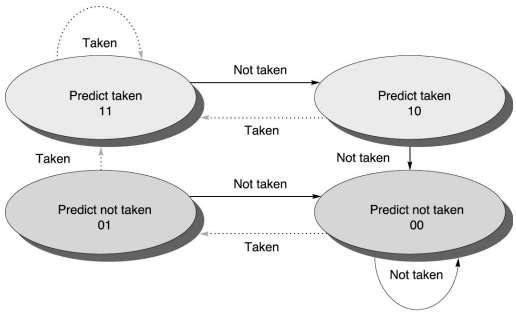
Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
Reorder #	3						6		4	5
Busy	yes	no	no	no	no	no	yes	...	yes	yes

Speculation

- Branch prediction
  - Use a predictor to predict the value of not-yet available branch conditions
  - Delay commitment of speculative instructions
  - Empty the ROB in case of wrong prediction
- Strategies:
  - Branch prediction
  - Target prediction

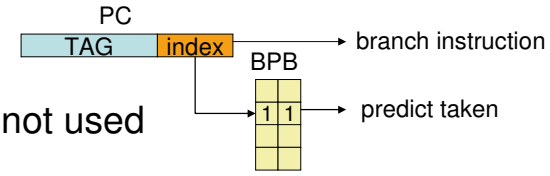
Branch prediction

- 1-bit predictor
  - 2 mispredictions per condition change
- 2-bit predictor
  - 1 misprediction per condition change
- n-bit predictors



Branch prediction buffer

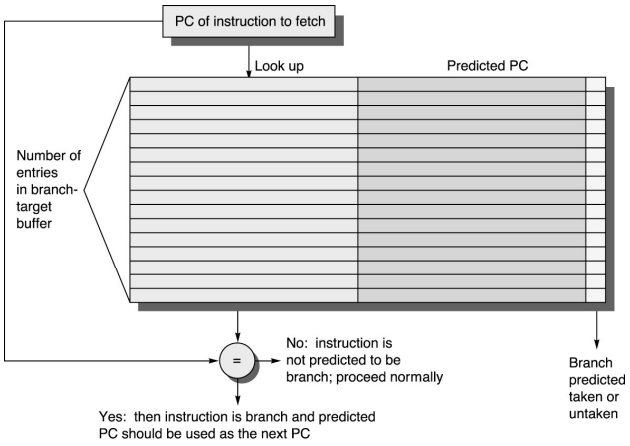
- 1 predictor for all branches
- 1 predictor for each branch
- *Branch prediction buffer*
  - N-entry table
  - Indexed by part of the instruction address (PC)
  - Tag not used



Branch target buffer

- Directly provide the target PC in case of predicted taken branches
- Works like a fully-associative cache
  - A cache hit provide the (predicted) target PC
  - A cache miss means PC=PC+1 (either the instruction is not a branch, or the branch is untaken)
- Predictions must be instruction-specific
- Predict-untaken branches are removed from the buffer
- Any branch-prediction strategy can be associated with the target (e.g., 1-bit, 2-bit, n-bit, …)

Branch target buffer



Branch target buffer