Computer Architecture

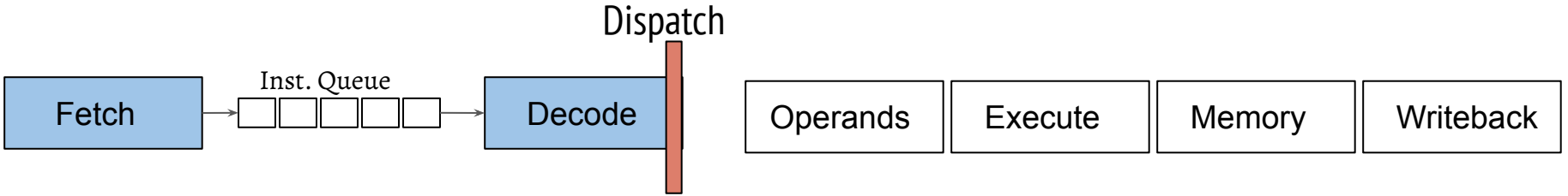
Dynamic Scheduling: Scoreboard

Debadatta Mishra, CSE, IITK

Dynamic Scheduling: Overall scheme

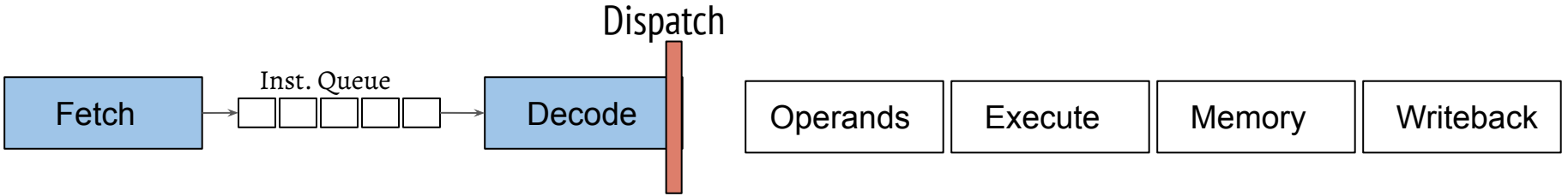
- Considering the five stage pipeline, in which stage, the dynamic scheduling should take place and why?
- What are the changes to the pipeline structure?
- What are the relevant design questions?

Dynamic Scheduling: Overall scheme



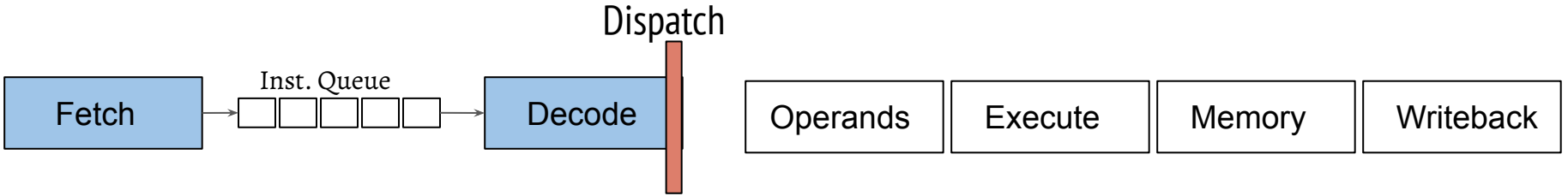
- Considering the five stage pipeline, in which stage, the dynamic scheduling should take place and why? Decode; dependency can be determined only after decoding the instructions
- What are the changes to the pipeline structure? Decode and RF access should be two different stages. Typically, an instruction queue between the fetch and decode.
- What are the relevant design questions?

Dynamic Scheduling: Overall scheme



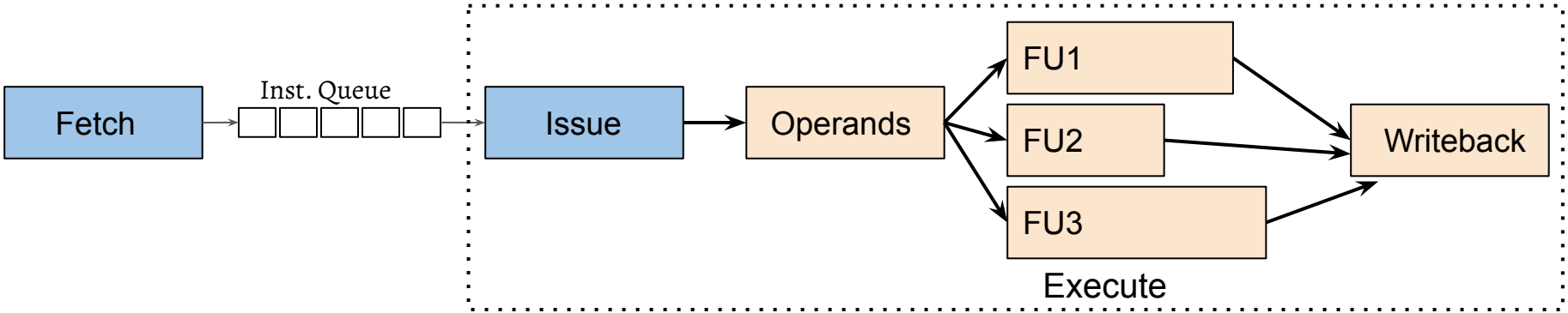
- Considering the five stage pipeline, in which stage, the dynamic scheduling should take place and why? Decode; dependency can be determined only after decoding the instructions
- What are the changes to the pipeline structure? Decode and RF access should be two different stages. Typically, an instruction queue between the fetch and decode.
- What are the relevant design questions?
  - What is the size of instruction queue? Importantly, can instructions across branches be part of the scheduling decision?
  - What to do with instructions with different dependencies? {Stall} or {schedule but handle during later stages}
  - How many instructions to schedule?

Dynamic Scheduling: Overall scheme



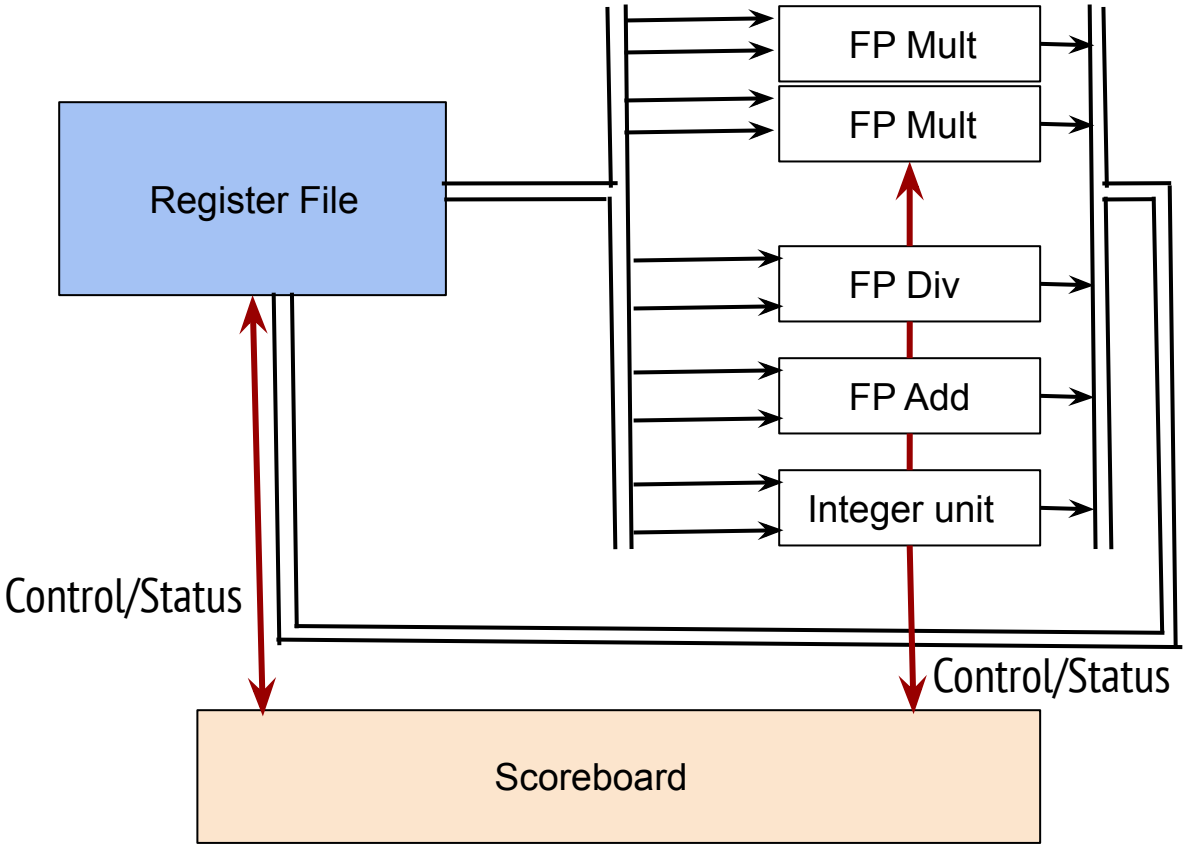
- What is the size of instruction queue? Importantly, can instructions across branches be part of the scheduling decision? For improved performance, a design should allow instructions across branches. Issues to tackle: branch misprediction {undoing the effects of WB and Mem Store}
- What to do with instructions with different dependencies? Stall or schedule but handle during later stages? Advancing the instructions as much seems to be a good idea, but locking resources while waiting for operands (RAW) can be detrimental. Advancing instructions with WAR/WAW hazards should make sure of program correctness.
- How many instructions to schedule? More FUs/access ports to RF/Mem, instruction/FU state maintenance overheads are not the only issues here. Penalty of doing useless work and nullifying their impacts because of control hazards is also an important concern

Dynamic Scheduling: Scoreboard



- Assumptions
  - Multiple/pipelined functional units with different execution lengths
  - Schedule (or issue) instructions from the same basic block
  - Multiport register file (why?)
  - No forwarding/bypass network
  - Memory LD/ST is one cycle and requires an adder (for simplifying our discussion, we will remove this soon)

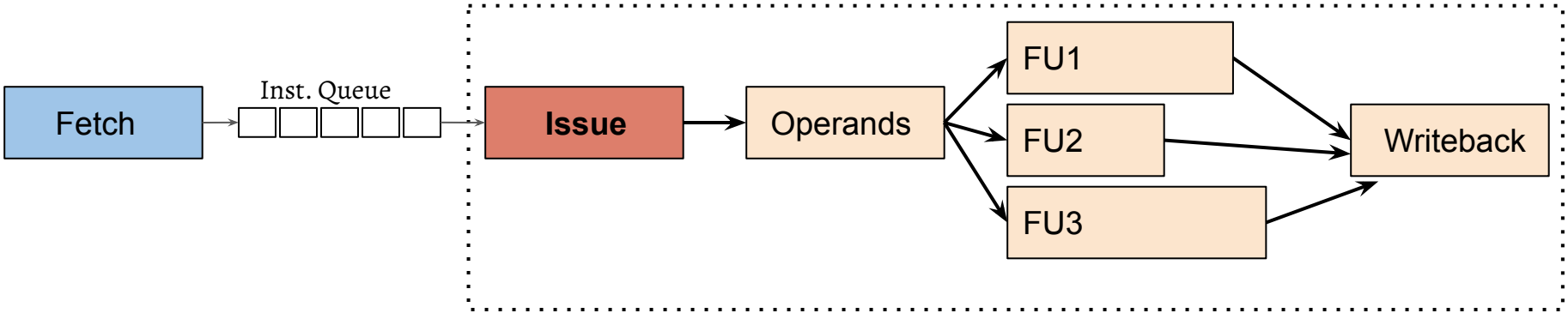
Basic structure of an example scoreboard



Integer unit performs all ALU operations and operations for memory address calculation

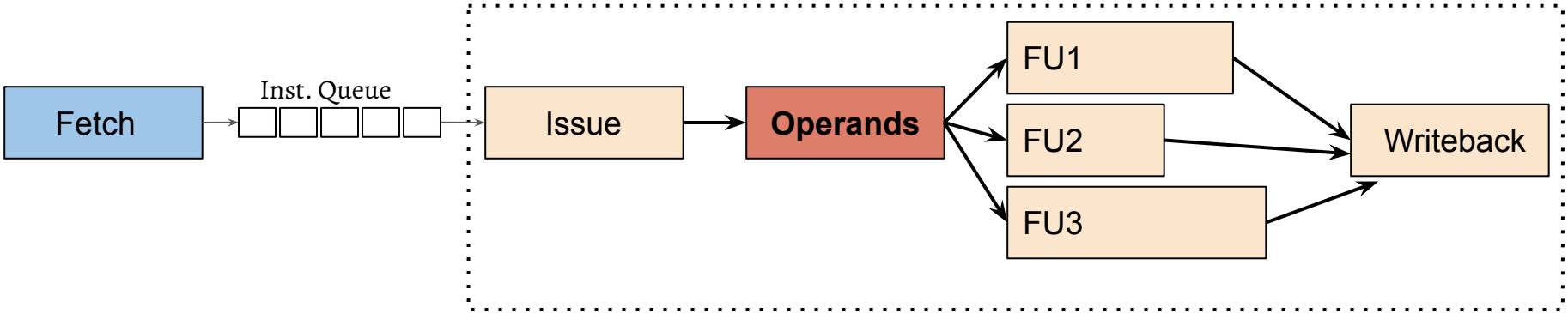
#of concurrent access to register file handled as a structural hazard issue

Scoreboard: Issue



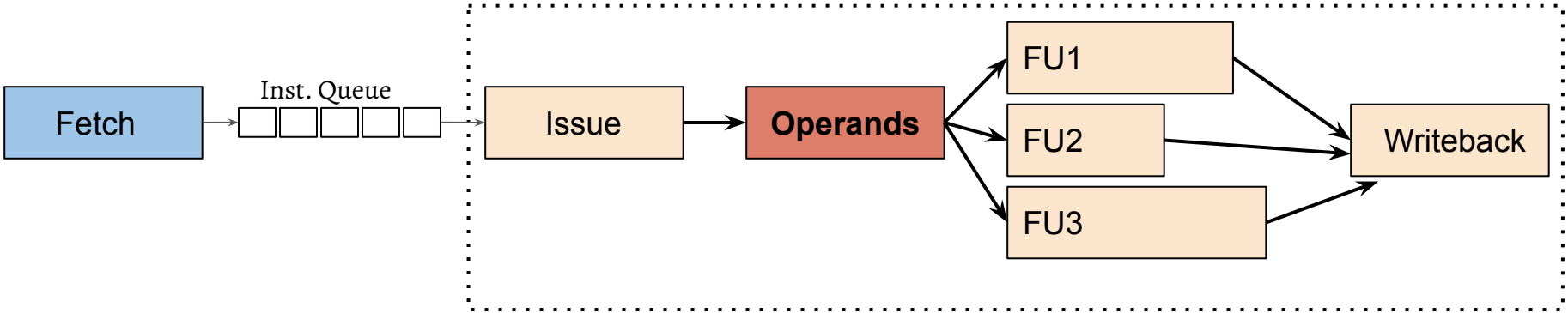
- Decode instruction and check for structural hazards
  - Need information regarding current usage of functional units (part of scoreboard)
  - Stop issue till structural hazard is resolved
- Stall the instruction if its output dependant on any instruction currently in execution
  - WAW is taken care of in the issue stage
- All issued instructions are tracked using the scoreboard

Scoreboard: Read operands (dispatch)



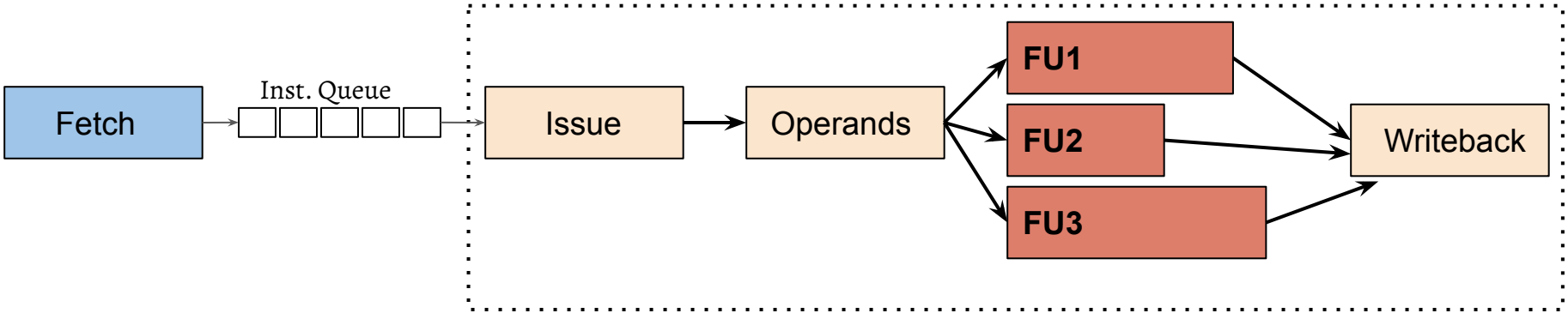
- Instruction with true data dependence stall
  - Information regarding waiting instructions maintained in the scoreboard
  - When source operands are available, the dependent instructions start execution (a.k.a. dispatched OoO, WAR is a possibility!)
- What should be the #of read ports in RF?

Scoreboard: Read operands (dispatch)



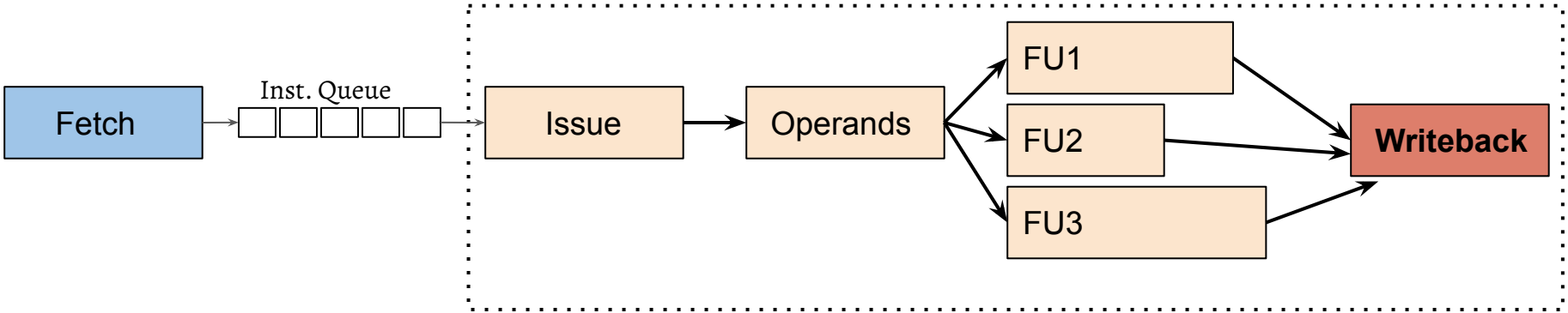
- Instruction with true data dependence stall
  - Information regarding waiting instructions maintained in the scoreboard
  - When source operands are available, the dependent instructions start execution (a.k.a. dispatched OoO)
- What should be the #of read ports in RF? 2 * Number of functional units (assuming functional units are marked free after execute)

Scoreboard: Execute



- FU starts executing the instruction (can consume variable #of cycles)
- Notifies the scoreboard on completion so that the scoreboard can update the state of the FU and the instruction

Scoreboard: Write result



- Scoreboard checks for WAR hazard for the completed instruction and stalls if required
- All instructions clear of WAR can perform write back (more than one possible)
- Subtle issue: When should the FU is considered free, end of execution or after writing results?

Structure of a Scoreboard

Instruction status

Issue	ReadOp	Execute	WriteRes

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk

Register result status

	F0	F1	F2	…	Fn
FU					

Busy: True/False Op: Operation Fi: Destination register

FjFk: Source register numbers

QjQk: Functional units producing FjFk

RjRk: FjFk are ready but not yet read

- Instruction status: Status of every instruction admitted to the scoreboard
- Functional unit status: Indicates the state of the functional unit
- Register result status: Functional units used by active instructions which will write to a given register

A snapshot representation of the scoreboard

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	1	2	3	
I2	2	3		

		I3	I2	I1

Instruction Queue

Clock Cycle

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	1	Add	f7	r2 _	_ _	No No
FA	1	Sub	f3	f2 f4	_ _	Yes Yes

Register result status

	F0	F1	F2	F3	F7
FU				FA	INT

- Snapshot at the beginning of 4th clock cycle
- Clock cycle when the instruction entered that stage is shown (for our understanding)
- `I3` is considered next to be issued

Example basic block

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

- Execution time in cycles: Integer Op = 1, FP add = 2, FP multiply = 10 and FP divide = 40
- Possible hazards
  - Structural: `{I1,I2}{I4,I6}`
  - RAW: `{I1,I4},{I1,I5},{I2,I3},{I2,I4},{I2,I6}{I3,I5}{I4,I6}`
  - WAR: `{I4,I6},{I5,I6}`

Example: Cycle 0

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	0					
FM2	0					
FA	0					
FD	0					

				I1

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Register result status

	F0	F1	F2	F3	F4	F5	F6	F7	F8
FU									

Example: Cycle 1

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	1			

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	1	add_l	f6	r2 _	_ _	Yes _
FM1	0					
FM2	0					
FA	0					
FD	0					

			I2	I1

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Register result status

	f0	f1	f2	f3	f4	f5	f6	f7	f8
FU							INT		

Example: Cycle 2

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	1	2		

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	1	add_l	f6	r2 _	_ _	Yes _
FM1	0					
FM2	0					
FA	0					
FD	0					

		I3	I2	I1

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Register result status

	f0	f1	f2	f3	f4	f5	f6	f7	f8
FU							INT		

Structural hazard, can't issue I2

Example: Cycle 3

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	1	2	3	

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	1	add_l	f6	r2 _	_ _	No _
FM1	0					
FM2	0					
FA	0					
FD	0					

	I4	I3	I2	I1

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Register result status

	f0	f1	f2	f3	f4	f5	f6	f7	f8
FU							INT		

Structural hazard, can't issue I2

Example: Cycle 4

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	*4*

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	0					
FM2	0					
FA	0					
FD	0					

I5	I4	I3	I2	I1

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Register result status

	f0	f1	f2	f3	f4	f5	f6	f7	f8
FU							INT		

INT can be used in the next CC, f6 available in next CC

Example: Cycle 5

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	*4*
I2	5			

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	1	add_l	f2	r3 _	_ _	Yes _
FM1	0					
FM2	0					
FA	0					
FD	0					

I6	I5	I4	I3	I2

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Register result status

	f0	f1	f2	f3	f4	f5	f6	f7	f8
FU			INT						

Single Issue, load issued

Example: Cycle 6

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	*4*
I2	5	6		
I3	6			

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	1	add_l	f2	r3 _	_ _	Yes _
FM1	1	Mul	f0	f2 f4	INT _	No Yes
FM2	0					
FA	0					
FD	0					

I6	I5	I4	I3	I2

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

Register result status

	f0	f1	f2	f3	f4	f5	f6	f7	f8
FU	FM1		INT						

Example: Cycle 7

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	*4*
I2	5	6	7	
I3	6			
I4	7			

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	1	add_l	f2	r3 _	_ _	No _
FM1	1	Mul	f0	f2 f4	INT _	No Yes
FM2	0					
FA	1	Sub	f8	f6 f2	_ INT	Yes No
FD	0					

I6	I5	I4	I3	I2

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Register result status

	f0	f1	f2	f3	f4	f5	f6	f7	f8
FU	FM1		INT						FA

Example: Cycle 8 (at end)

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	*4*
I2	5	6	7	8
I3	6			
I4	7			
I5	8			

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	1	Mul	f0	f2 f4	INT _	Yes Yes
FM2	0					
FA	1	Sub	f8	f6 f2	_ INT	Yes Yes
FD	1	Div	f10	f0 f6	FM1 _	No Yes

I6	I5	I4	I3	I2

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU	FM1	FD							FA

Example: Cycle 9

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	*4*
I2	5	6	7	8
I3	6	9		
I4	7	9		
I5	8			

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	1	Mul	f0	f2 f4	_ _	Yes Yes
FM2	0					
FA	1	Sub	f8	f6 f2	_ _	Yes Yes
FD	1	Div	f10	f0 f6	FM1 _	No Yes

	I6	I5	I4	I3

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU	FM1	FD							FA

Example: Cycle 10

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	*4*
I2	5	6	7	8
I3	6	9	10(r9)	
I4	7	9	10(r1)	
I5	8			

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	1	Mul	f0	f2 f4	_ _	No No
FM2	0					
FA	1	Sub	f8	f6 f2	_ _	No No
FD	1	Div	f10	f0 f6	FM1 _	No Yes

	I6	I5	I4	I3

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU	FM1	FD							FA

Example: Cycle 11

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	***4***
I2	5	6	**7**	**8**
I3	**6**	**9**	**11(r8)**	
I4	**7**	**9**	**11**	
I5	8			

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	1	Mul	f0	f2 f4	_ _	No No
FM2	0					
FA	1	Sub	f8	f6 f2	_ _	No No
FD	1	Div	f10	f0 f6	FM1 _	No Yes

	I6	I5	I4	I3

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU	FM1	FD							FA

I4 finished execution, I6 can not start yet

Example: Cycle 12 (at end)

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	***4***
I2	5	6	**7**	**8**
I3	**6**	**9**	**12(r7)**	
I4	**7**	**9**	**11**	**12**
I5	8			

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	1	Mul	f0	f2 f4	_ _	No No
FM2	0					
FA	0					
FD	1	Div	f10	f0 f6	FM1 _	No Yes

	I6	I5	I4	I3

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU	FM1	FD							

I4 finished execution, I6 can not start yet

Example: Cycle 13

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	*4*
I2	5	6	7	8
I3	6	9	13(r6)	
I4	7	9	11	12
I5	8			
I6	13			

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	1	Mul	f0	f2 f4	_ _	No No
FM2	0					
FA	1	Add	f6	f8 f2	_ _	Yes Yes
FD	1	Div	f10	f0 f6	FM1 _	No Yes

	I6	I5		I3

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU	FM1	FD					FA		

Example: Cycle 14

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	***4***
I2	5	6	**7**	**8**
I3	**6**	**9**	**14(r5)**	
I4	**7**	**9**	**11**	**12**
I5	8			
I6	**13**	**14**		

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	1	Mul	f0	f2 f4	_ _	No No
FM2	0					
FA	1	Add	f6	f8 f2	_ _	Yes Yes
FD	1	Div	f10	f0 f6	FM1 _	No Yes

	I6	I5		I3

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU	FM1	FD					FA		

Example: Cycle 15

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	*4*
I2	5	6	7	8
I3	6	9	15(r4)	
I4	7	9	11	12
I5	8			
I6	13	14	15(r1)	

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	1	Mul	f0	f2 f4	_ _	No No
FM2	0					
FA	1	Add	f6	f8 f2	_ _	No No
FD	1	Div	f10	f0 f6	FM1 _	No Yes

	I6	I5		I3

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU	FM1	FD					FA		

Example: Cycle 16

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	***4***
I2	5	6	**7**	**8**
I3	**6**	**9**	**16(r3)**	
I4	**7**	**9**	**11**	**12**
I5	8			
I6	**13**	**14**	**16**	

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	1	Mul	f0	f2 f4	_ _	No No
FM2	0					
FA	1	Add	f6	f8 f2	_ _	No No
FD	1	Div	f10	f0 f6	FM1 _	No Yes

	I6	I5		I3

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU	FM1	FD					FA		

I6 finished execution, can not write the result back

Example: Cycle 19

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	***4***
I2	5	6	**7**	**8**
I3	**6**	**9**	**19**	
I4	**7**	**9**	**11**	**12**
I5	8			
I6	**13**	**14**	**16**	

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	1	Mul	f0	f2 f4	_ _	No No
FM2	0					
FA	1	Add	f6	f8 f2	_ _	No No
FD	1	Div	f10	f0 f6	FM1 _	No Yes

	I6	I5		I3

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU	FM1	FD					FA		

I6 finished execution, can not write the result back

Example: Cycle 20

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	***4***
I2	5	6	**7**	**8**
I3	**6**	**9**	**19**	**20**
I4	**7**	**9**	**11**	**12**
I5	8			
I6	**13**	**14**	**16**	

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	0					
FM2	0					
FA	1	Add	f6	f8 f2	_ _	No No
FD	1	Div	f10	f0 f6	FM1 _	Yes Yes

	I6	I5		I3

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU	FM1	FD					FA		

I6 finished execution, can not write the result back

Example: Cycle 21

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	***4***
I2	5	6	**7**	**8**
I3	**6**	**9**	**19**	**20**
I4	**7**	**9**	**11**	**12**
I5	8	**21**		
I6	**13**	**14**	**16**	

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	0					
FM2	0					
FA	1	Add	f6	f8 f2	_ _	No No
FD	1	Div	f10	f0 f6	FM1 _	Yes Yes

	I6	I5		

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU		FD					FA		

Now I5 can read operands (I3→I5 RAW)

Example: Cycle 22

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	*1*	*2*	*3*	***4***
I2	5	6	**7**	**8**
I3	**6**	**9**	**19**	**20**
I4	**7**	**9**	**11**	**12**
I5	8	**21**	**22(39)**	
I6	**13**	**14**	**16**	**22**

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	0					
FM2	0					
FA	1	Add	f6	f8 f2	_ _	No No
FD	1	Div	f10	f0 f6	FM1 _	No No

	I6	I5		

Instruction Queue

I1: lD f6,34(r2)

I2: lD f2,45(r3)

I3: mulD f0,f2,f4

I4: subD f8,f6,f2

I5: divD f10,f0,f6

I6: addD f6,f8,f2

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU		FD					FA		

Now I5 can read operands (I3→I5 RAW)

Example: Cycle 23

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	1	2	3	4
I2	5	6	7	8
I3	6	9	19	20
I4	7	9	11	12
I5	8	21	23(38)	
I6	13	14	16	22

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	0					
FM2	0					
FA	0					
FD	1	Div	f10	f0 f6	FM1 _	No No

		I5		

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU		FD							

Example: Cycle 61

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	1	2	3	4
I2	5	6	7	8
I3	6	9	19	20
I4	7	9	11	12
I5	8	21	61	
I6	13	14	16	22

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	0					
FM2	0					
FA	0					
FD	1	Div	f10	f0 f6	FM1 _	No No

		I5		

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU		FD							

Example: Cycle 62

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	1	2	3	4
I2	5	6	7	8
I3	6	9	19	20
I4	7	9	11	12
I5	8	21	61	62
I6	13	14	16	22

FU status

FUID	Busy	Op	Fi	Fj Fk	Qj Qk	Rj Rk
INT	0					
FM1	0					
FM2	0					
FA	0					
FD	0					

Instruction Queue

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

Register result status

	f0	f10	f2	f3	f4	f5	f6	f7	f8
FU									

Example: Observations

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	1	2	3	4
I2	5	6	7	8
I3	6	9	19	20
I4	7	9	11	12
I5	8	21	61	62
I6	13	14	16	22

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

- Issue is in-order, execution and commit (write back) are out-of-order
- Allows instruction to be issued even if there are RAW dependence. Why this is an issue?

Example: Observations

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	1	2	3	4
I2	5	6	7	8
I3	6	9	19	20
I4	7	9	11	12
I5	8	21	61	62
I6	13	14	16	22

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

- Issue is in-order, execution and commit (write back) are out-of-order
- Allows instruction to be issued even if there are RAW dependence. Why this is an issue? The FU is locked till RAW is resolved
- FU remain occupied even after execution completes, why?

Example: Observations

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	1	2	3	4
I2	5	6	7	8
I3	6	9	19	20
I4	7	9	11	12
I5	8	21	61	62
I6	13	14	16	22

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

- Issue is in-order, execution and commit (write back) are out-of-order
- Allows instruction to be issued even if there are RAW dependence. Why this is an issue? The FU is locked till RAW is resolved
- FU remain occupied even after execution completes, why? To avoid WAR hazards, it can have cumulative effect of RAW + WAR
- What happens when an exception occur?

Example: Observations

Instruction status

Inst	Issue	ReadOp	Execute	WriteRes
I1	1	2	3	4
I2	5	6	7	8
I3	6	9	19	20
I4	7	9	11	12
I5	8	21	61	62
I6	13	14	16	22

```
I1: lD f6,34(r2)
I2: lD f2,45(r3)
I3: mulD f0,f2,f4
I4: subD f8,f6,f2
I5: divD f10,f0,f6
I6: addD f6,f8,f2
```

- Issue is in-order, execution and commit (write back) are out-of-order
- Allows instruction to be issued even if there are RAW dependence. Why this is an issue? The FU is locked till RAW is resolved
- FU remain occupied even after execution completes, why? To avoid WAR hazards, it can have cumulative effect of RAW + WAR
- What happens when an exception occur? Imprecise exception, need additional OS/hw support/mechanisms

Scoreboard: Limitations

- Amount of parallelism available in a basic block is limited
- Instruction window is dependent on # of scoreboard entries
- Execution and result write are tightly coupled ⇒ More structural hazards (impacts ILP)
- Stalls to avoid WAW and WAR ⇒ More structural hazards (impacts ILP)
- No forwarding (1 extra cycle to get the operand)
- What about memory?

Scoreboard: Limitations

- Amount of parallelism available in a basic block is limited
- Instruction window is dependent on # of scoreboard entries
- Execution and result write are tightly coupled ⇒ More structural hazards (impacts ILP)
- Stalls to avoid WAW and WAR ⇒ More structural hazards (impacts ILP)
- No forwarding (1 extra cycle to get the operand)
- What about memory?
    - Allowing multiple memory OPs to enter the scoreboard results in added complexity in ensuring correct dependency. A simple solution is to stall LD/ST if another LD/ST is in execution.