Tomasulo's
(requires RAT)

Lesson 7

Instruction Scheduling

Hardware implementation of register renaming and out-of-order execution using Tomasulo's Algorithm.

Improving IPC

ILP should be at least 4 instructions per cycle
Control Dependencies are mitigated with Branch Prediction
WAR and WAW data dependencies are removed with register renaming (RAT)
RAW Data dependencies can be improved through out of order execution
Structural Dependencies can be improved with wider issue processors.

with OoO execution, instead of being completely blocked by a True Dependence we can find another instruction that is ready to execute & execute that one instead.

- Instruction Queue
- Reservation Stations
- Register File
- Execution Unit
- Result Broadcast Bus

Tomasulo's Algorithm → Out of Order Execution

Tomasulo's algorithm determines which instructions have inputs ready for execution, it uses a form of register renaming, and it is very similar to the method used today.
The differences between what is used today and Tomasulo's algorithm are:

so that they can move on to the next stage (execution).

1. All instructions use the algorithm, not just floating point ones.
2. Hundreds of instructions are ~~considering~~ considered when performing out-of-order execution and register renaming, not just a few.
3. There is now support for exception handling.

Instruction to Broadcast Path

Data Manipulation Path:

this is what gives us the out of order capability

Instruction Queue → Reservation stations (values come from the registers)→ Execution Units (Adders and Multiplier) → Broadcast on the Bus

Load/Store Path: (these instruction

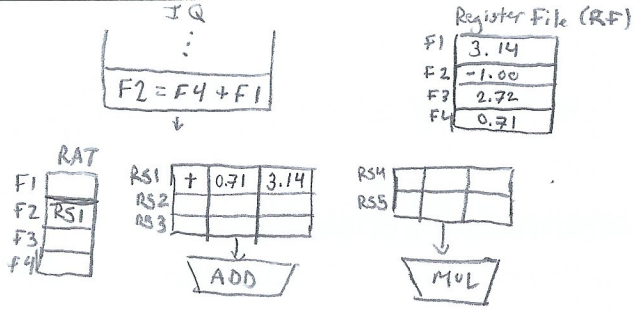
Buffer is like a queue, this keeps the LOAD+STORE instructions IN-ORDER (these need to occur in order)

Instruction Queue → Adder (for PC) → Load or Store Buffer → Memory → Broadcast on the bus

Issue = instructions exit the queue and go to either the RS or the PC-Adder.
Dispatch = Instruction exits RS and goes to either the Adder or the Multiplier for execution.
Write Result (or Broadcast) = the instruction exits the Adder or the Multiplier and is put on the bus.

Issue

1. Next instruction from the IQ - instructions are issued in program order (INORDER)
2. Determine origin of inputs
3. Get free RS of the correct kind - if there are no free RS, the instruction needs to wait for an open RS. (stalling the IQ)
4. Put instruction in the appropriate RS
5. Tag the destination register - this register will hold the result of the instruction and all other instructions will be able to access this result if necessary.



Issue Example

Steps for Issue

1. Take instruction from IQ
2. Look in the RAT
   - if RAT has a value for the register - use the register pointed to
   - if RAT does not have a value - look in the register file
3. Reservations Stations - some are adders and some are for multipliers
   - if an RS is open - use it
   - if an RS is not available - wait for one to be open
4. Store in the RAT the register that holds the result of the instruction. (there, the result is going to be tagged w/ the reservation station #)

Dispatch (aka "capture")

Dispatch needs to latch the results and determine which instructions are ready to execute.

1. Free the RS (who's result is currently being broadcast - ex: if broadcasting result w/ "RS1" tag, then free RS1)
2. Match the broadcast tag with the ~~operations~~ operands in the RS
3. Insert the value in the appropriate RS
4. Once an RS has all of its required inputs - it can execute in the Adder or the Multiplier.
5. When the result is ready from the Adder or the Multiplier, broadcast it on the bus.

If more than 1 instruction is ready to ~~Broadcast~~ Dispatch

Which instruction should be dispatched first.

options:
- Can choose the oldest instruction first
- The instruction with the most dependencies goes first (how many instructions are waiting for this result)- this is difficult to implement in hardware
- Random selection

Broadcast

1. Put the tag (RS) and the result on the bus. (update the operands waiting to capture that tagged RS result)
2. Write the result to the register file. (find which register the result actually belongs to using the RAT & then update that register w/ the result)
3. Update the RAT. An empty RAT entry means the value is ready in the register file
4. Free the reservation station (change the valid bit) that matches the result's tag

Update RAT by erasing the entry to indicate that the result can be found in the register.

ONLY if there's a RAT entry that has this RS tag. If not, that means the register has been renamed.

What if there is More the 1 Broadcast Ready?

How to decide which result is broadcast first?

options:
1. A separate bus for each arithmetic unit - this increases the hardware needs
2. Give priority to the slower unit - the slower unit instructions will most likely have more dependencies. The multiplication/divide unit takes more cycles to complete - so it is usually given priority.

Broadcasting a Stale Result

A result is stale if it no longer has an entry in the RAT.

A stale result is broadcast and placed in the appropriate RS entry. The RAT is not altered because any future instructions will not use this value.

Remember this on Midterms and Finals

So even if a result is "stale" & its register has been renamed, other instructions could still be depending on its results & sitting in the Reservation stations waiting for them.

Review

For each instruction:

Issue

Capture/Dispatch

Write Result

In each cycle (at the same time)

Issue

Capture/Dispatch - one for each arithmetic unit

Write results for different instructions for the same instruction

Can an issue and dispatch be done in the same cycle? It depends on the processor. Issue time is shorter than the cycle time, so there can be time to do the dispatch. But the cycle should be as short as possible, so it should not be feasible.

Capturing operands and dispatching should not be done in the same cycle. (depends on processor)

Broadcasting and issuing to the same RS can be done in the same cycle but it requires additional logic to work.

In the same cycle, when "issue" & "write result" both need to update the RAT, ALWAYS update the RAT w/ the issue step.

Load and Store Instructions

There can be data dependencies through memory.

RAW occurs when a store word to a memory location occurs after a load word of that same memory location.

WAR occurs when a load word to a memory location occurs after a store word.

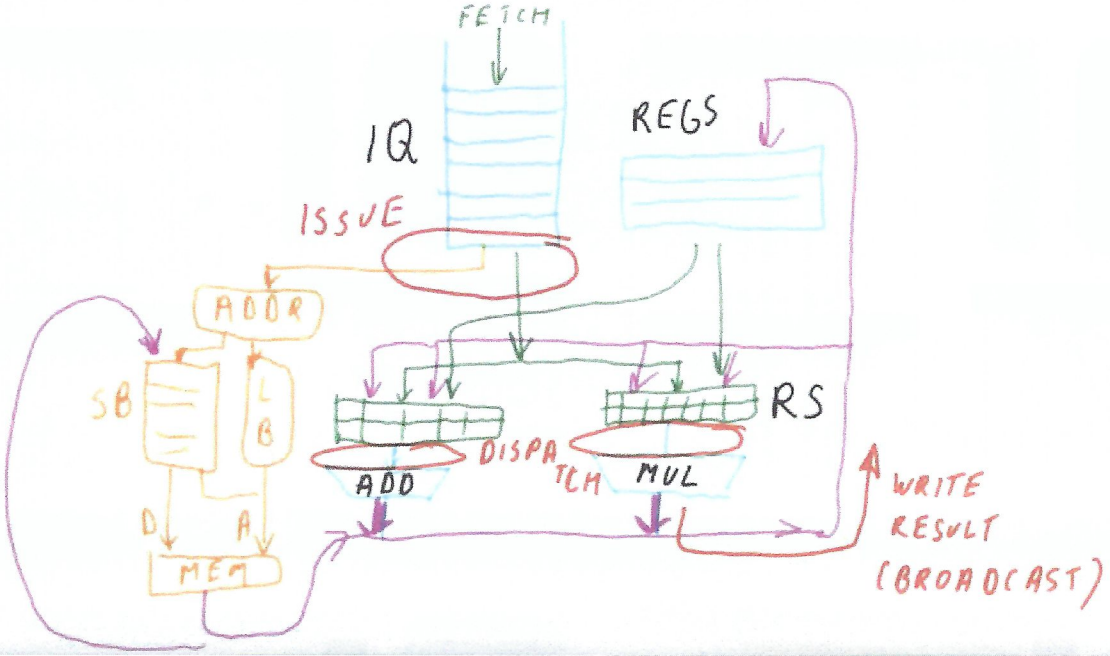
WAW occurs when there are two store words to the same memory location.

These need to obeyed or eliminated by;

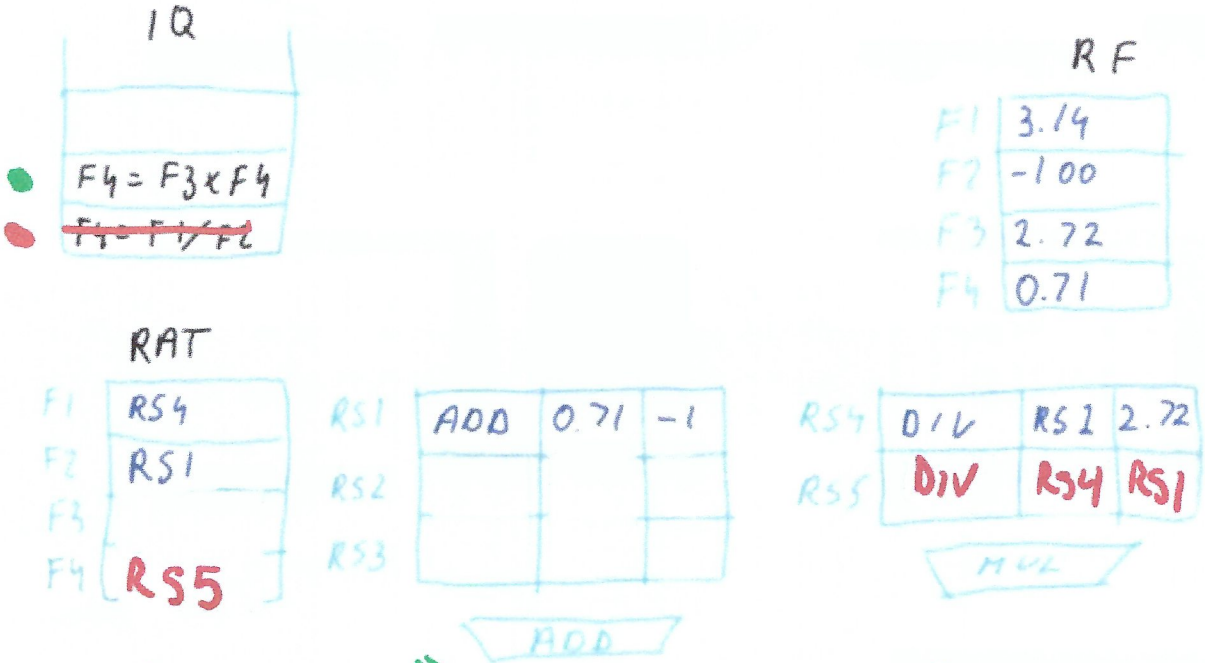
-Doing loads and stores in-order

-Identify dependencies and reorder - this is done in modern processors, but not in Tomasulo's algorithm.

TOMASULO'S ALGORITHM - THE PICTURE



Issue Example:



No space in "MUL" reservation station for the next instruction (F4=F3xF4), so processor would have to wait for the next cycle

mple Cycles 1-2 - Georgia Tech - HPCA: Part 2

TOMASULO - LONG EXAMPLE

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp
LD1	✓	L		134			✓ 4
LD2	✓	L		245			
AD1							
AD2							
AD3							
ML1							
ML2							

	Instruction	Is	Ex	Wr
1.	LD F6, 34(R2)	1	2	
2	LD F2, 45(R3)	2		
3.	MUL D F0, F2, F4			
4	SUB D F8, F2, F6			
5	DIV D F10, F0, F6			
6	ADD D F6, F8, F2			

Register Status:

F0	F2	F4	F6	F8	F10
	LO2		LO1		

Cycle: 2

ple Cycles 3-4 - Georgia Tech - HPCA: Part 2

TOMASULO - LONG EXAMPLE

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp
LD1	✓	L		134			✓ 4
LD2	✓	L		245			
AD1							
AD2							
AD3							
ML1	✓	M		2.5	LO2		
ML2							

This depends on whether our Load/Store unit is pipelined. If so, then we can begin execution in cycle 3. If not we need to wait till cycle 4 (when the prev. load is complete) in order to utilize it. In this example we must wait.

	Instruction	Is	Ex	Wr
1.	LD F6, 34(R2)	1	2	
2	LD F2, 45(R3)	2		
3.	MUL D F0, F2, F4	3		
4	SUB D F8, F2, F6			
5	DIV D F10, F0, F6			
6	ADD D F6, F8, F2			

Register Status:

F0	F2	F4	F6	F8	F10
M1	LO2		LO1		

Cycle: 3

ple Cycles 3-4 - Georgia Tech - HPCA: Part 2

TOMASULO - LONG EXAMPLE

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp
LD1							
LD2	✓	L		245			✓
AD1	✓	S		7.1	LO2		
AD2							
AD3							
ML1	✓	M		2.5	LO2		
ML2							

Begins execution

F6 7.1

	Instruction	Is	Ex	Wr
1.	LD F6, 34(R2)	1	2	4
2	LD F2, 45(R3)	2	4	
3.	MUL D F0, F2, F4	3		
4	SUB D F8, F2, F6	4		
5	DIV D F10, F0, F6			
6	ADD D F6, F8, F2			

Register Status:

F0	F2	F4	F6	F8	F10
M1	LO2			A01	

Cycle: 4

ple Cycles 5-6 - Georgia Tech - HPCA: Part 2

TOMASULO - LONG EXAMPLE

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp
LD1							
LD2	✓	L		245			✓
AD1	✓	S		7.1	LO2		
AD2							
AD3							
ML1	✓	M		2.5	LO2		
ML2	✓	D		7.1	M1		

F6 7.1

	Instruction	Is	Ex	Wr
1.	LD F6, 34(R2)	1	2	4
2	LD F2, 45(R3)	2	4	
3.	MUL D F0, F2, F4	3		
4	SUB D F8, F2, F6	4		
5	DIV D F10, F0, F6	5		
6	ADD D F6, F8, F2			

Register Status:

F0	F2	F4	F6	F8	F10
M1	LO2			A01	ML2

Cycle: 5

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp
LD1							
LD2							
AD1	✓	S	-2.5	7.1	LO2		
AD2	✓	A		-2.5	A01	LO2	
AD3							
ML1	✓	M	-2.5	2.5	LO2		
ML2	✓	D		7.1	M1		

F2 -2.5
F6 7.1

	Instruction	Is	Ex	Wr
1	LD F6, 34(R2)	1	2	4
2	LD F2, 45(R3)	2	4	6
3.	MUL D F0, F2, F4	3		
4	SUB D F8, F2, F6	4		
5	DIV D F10, F0, F6	5		
6	ADD D F6, F8, F2	6		

Register Status:

F0	F2	F4	F6	F8	F10
M1	LO2			A02	A01 ML2

Cycle: 6

* can't dispatch in the same cycle that results were captured in.

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp
LD1							
LD2							
AD1	✓	S	-2.5	7.1			✓ 9
AD2	✓	A		-2.5	A01		
AD3							
ML1	✓	M	-2.5	2.5			✓ 17
ML2	✓	D		7.1	M1		

F2 -2.5
F6 7.1

	Instruction	Is	Ex	Wr
1	LD F6, 34(R2)	1	2	4
2	LD F2, 45(R3)	2	4	6
3.	MUL D F0, F2, F4	3	7	
4	SUB D F8, F2, F6	4	7	
5	DIV D F10, F0, F6	5		
6	ADD D F6, F8, F2	6		

Register Status:

F0	F2	F4	F6	F8	F10
M1				A02	A01 ML2

Cycle: 7

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

R2 is 100
R3 is 200
F4 is 2.5

	Busy	Op	Vj	Vk	Qj	Qk	Disp
LD1							
LD2							
AD1							
AD2	✓	A	-9.6	-2.5	A01		
AD3							
ML1	✓	M	-2.5	2.5			✓ 17
ML2	✓	D		7.1	M1		

F2 -2.5
F6 7.1
F8 -9.6

	Instruction	Is	Ex	Wr
1.	LD F6, 34(R2)	1	2	4
2	LD F2, 45(R3)	2	4	6
3.	MUL D F0, F2, F4	3	7	
4	SUB D F8, F2, F6	4	7	9
5	DIV D F10, F0, F6	5		
6	ADD D F6, F8, F2	6		

Register Status:

F0	F2	F4	F6	F8	F10
M1				A02	A01 ML2

Cycle: 9

TOMASULO'S ALGORITHM - TIMING EXAMPLE

Load: 2 cycles
Add: 2 cycles
Mult: 10 cycles
Divide: 40 cycles

	Instruction	Is	Ex	Wr	Comment
1.	LD F6, 34(R2)	1	2	4	
2	LD F2, 45(R3)	2	4	6	
3	MUL D F0, F2, F4	3	7	17	F2 FROM 2. (6)
4	SUB D F8, F2, F6	4	7	9	F2 FROM 2. (6 vs 4)
5.	DIV D F10, F0, F6	5	18	58	F0 FROM 3. (17 vs 4)
6	ADD D F6, F8, F2	6	10	12	F8 FROM 4. (9 vs 6)

LATENCY:
LD : 1 CYCLE
ADD : 1 CYCLE
MUL : 5 CYCLES

OF RES. STATIONS:
LD: 1
ADD: 2
MUL: 2

SAME-CYCLE:
ISSUE → DISPATCH: NO
CAPTURE → DISPATCH: NO
RS FREED → RS ALLOC: NO

	INST	ISSUE	DISP	WR
1	LD F6, 0(R2)	1	2 +1=3	
2	MUL F2, F0, F1	2	3 +5=8	
3	ADD F6, F2, F6	3	9 +1= 10	
4	ADD F6, F2, F6	4	11 +1= 12	
5	ADD F1, F1, F1			
6	ADD F1, F3, F4			

LATENCY:
LD : 1 CYCLE
ADD : 1 CYCLE
MUL : 5 CYCLES

OF RES. STATIONS:
LD: 1
ADD: 2
MUL: 2

SAME-CYCLE:
ISSUE → DISPATCH: NO
CAPTURE → DISPATCH: NO
RS FREED → RS ALLOC: NO

	INST	ISSUE	DISP	WR
1	LD F6, 0(R2)	1	2	3
2	MUL F2, F0, F1	2	3	8
3	ADD F6, F2, F6	3	9	10
4	ADD F6, F2, F6	4	11	12
5	ADD F1, F1, F1	11	12+1=13	
6	ADD F1, F3, F4	13	14+1=15	

From Cycle 4→10, there are no more Add RS spots available. But @ cycle 10, instruction 3 frees 1 Add RS spot, so we can allocate 4 @ the next cycle (11). Then both Add RS spots are full again until cycle 12, & one spot becomes available for allocation @ cycle 13.

Next Question

Assume that the result of an instruction can be written in the last cycle of its execution (so this means that if an instruction takes 3 cycles to execute, then it can write the results on the 3rd cycle as opposed to the next/4th cycle), and that a dependent instruction can (if selected) begin its execution in the cycle after that.

The execution time of all instructions is two cycles, except for multiplication (which takes 4 cycles) and division (which takes 8 cycles).

The processor has one multiply/divide unit and one add/subtract unit. The multiply/divide unit has two reservation stations and the add/subtract unit has four reservation stations.

None of the execution units is pipelined – each can only be executing one instruction at a time.

If a conflict for the use of an execution unit occurs when selecting which instruction should start to execute, the older instruction (the one that appears earlier in program order) has priority. If a conflict for use of the CBD occurs, the result of the add/subtract unit has priority over the result of the multiply/divide unit.

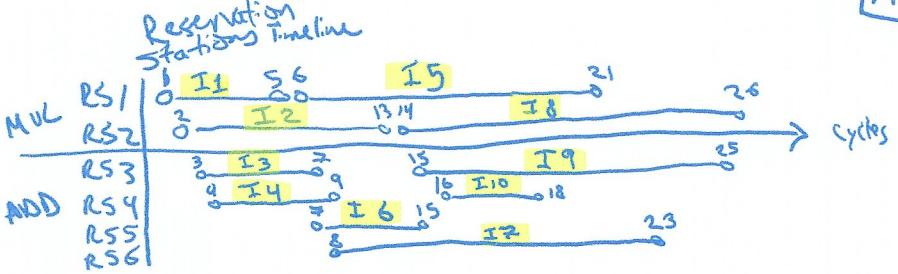
Assume that at start all instructions are already in the instruction queue, but none has yet been issued to any reservation stations. The processor can issue only one instruction per cycle, and there is only one CDB for writing results.

		Issues	Executes	Writes result
I1	MUL F2, F1, F1	1	(1+1) 2 + 3 =	5
I2	DIV F4, F4, F2	2	(5+1) 6 + 7 =	13
I3	ADD F1, F2, F3	3	(5+1) 6 + 1 =	7
I4	ADD F2, F1, F3	4	(7+1) 8 + 1 =	9
I5	DIV F1, F4, F2	(5+1) 6	(13+1) 14 + 7 =	21
I6	SUB F4, F4, F2	7	(13+1) 14 + 1 =	15
I7	ADD F3, F1, F2	8	(21+1) 22 + 1 =	23
I8	MUL F1, F2, F1	(13+1) 14	(21+1) 22 + 3 =	25 26
I9	ADD F3, F3, F4	15	(23+1) 24 + 1 =	25
I10	SUB F4, F5, F6	16	17 + 1 =	18

EXEC Times
MUL (+3 cycles)
DIV (+7 cycles)
everything else (+1 cycle)

RS1
RS2
MUL

RS3
RS4
RS5
RS6
ADD



Next Question:

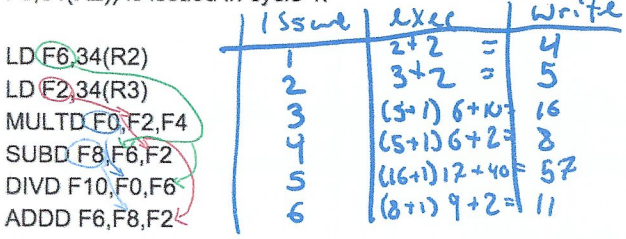
1) There are a lot of reservation stations so we never run out of them
2) Result is broadcast in the cycle after execution completes.
3) If an instruction gets its operand from the broadcast in one cycle, it cannot dispatch in the same cycle.

Then we get:

LD F6, 34(R2) 1 2 4 Execution during cycle 2, 3, and write result on cycle 4
LD F2, 34(R3) 2 3 5 Since there is 3 load units, this instruction can be dispatched on cycle 3
MULTD F0, F2, F4 3 6 16 F2 is broadcast on cycle 5, thus this instruction can begin execution on cycle 6
SUBD F8, F6, F2 4 6 8 F2 is broadcast on cycle 5, thus this instruction can begin execution on cycle 6
DIVD F10, F0, F6 5 17 57 F0 is broadcast on cycle 16, thus this instruction can begin execution on cycle 17
ADDD F6, F8, F2 6 9 11 F8 is broadcast on cycle 8, thus this instruction can begin execution on cycle 9

The cycle in which the instruction writes the result is the cycle it is considered complete.

Schedule the following code using Tomasulo's algorithm assuming the hardware has three Load-units with a two-cycle execution latency, three Add/Sub units with 2 cycles execution latency, and two Mult/Div units where Mult has an execution latency of 10 cycles and Div 40 cycles. Assume the rst instruction (LD F6,34(R2)) is issued in cycle 1.



In which clock cycle (numbered 0,1,2,...) does the second LD instruction complete? 5

In which clock cycle does the MULTD instruction complete? 16

In whic clock cycle does the ADDD instruction complete? 11