Takehome Assignment Parallel System Architectures

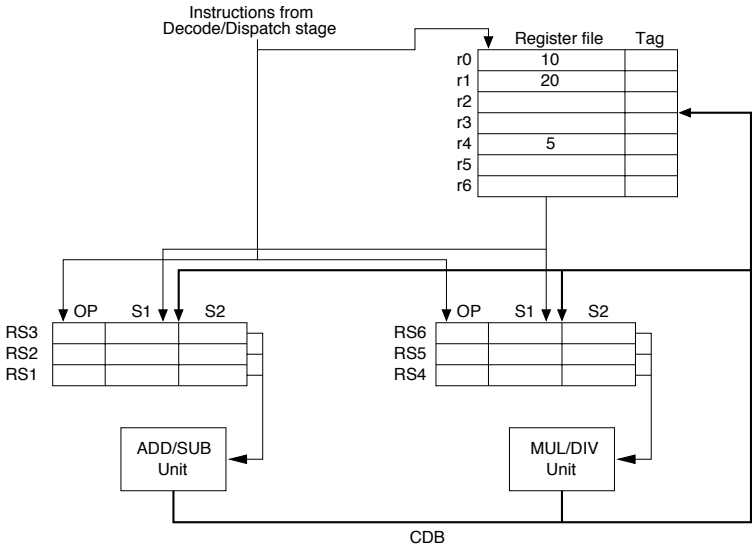
Deadline: 18 January, 23.59h

1a (30%) Suppose we execute the following code:

R3 = R3 op R5
R4 = R3 + 1
R3 = R5 + 1
R7 = R3 op R4

Indicate what the problems are when you execute this code on an out-of-order processor. Also explain how register renaming hardware could resolve these issues.

1b (70%) Assume the below architecture based on Tomasulo scheduling (with register status):



Further assume that the architecture can dispatch 2 instructions per cycle: one to the reservation station of the ADD/SUB unit and one to the reservation station of the MUL/DIV unit. ADD/SUB instructions take 1 cycle to execute while MUL/DIV instructions take 5 cycles to execute. Show for the next 4 instructions how they are scheduled/executed:

```
instructie i:    ADD R2, R0, R1   // R2 = R0+R1
instructie i+1:  MUL R4, R2, R1   // R4 = R2*R1
instructie i+2:  DIV R3, R4, R2   // R3 = R4/R2
instructie i+3:  SUB R4, R2, R0   // R4 = R2-R0
```

Draw the status of the reservation stations and the register file during the important cycles.

TakeHome - Parallel System Architectures

1a) The 1st (R3 := R3 op R5) and the 3rd (R3 := R5 +1) instructions are not dependent on each other. So, if the 1st one is issued the 3rd can also be issued. The 3rd uses R5, and R5 is not being produced anywhere near, so it can be executed pretty soon after the 1st one is executed. However, if we suppose that the 1st instruction is a long latency operation (takes some time to execute) then it could mean that the 3rd instruction is going to finish earlier and write its results to R3. In that case, the 4th instruction will read the result of R3 of the 3rd instruction instead of the 1st. This is called an output dependency and the issue takes place because we are writing to the same register out-of-order and this has consequences for the instructions that read from that register.

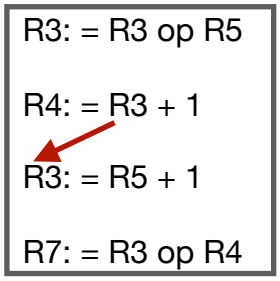
R3: = R3 op R5

R4: = R3 + 1

R3: = R5 + 1

R7: = R3 op R4

Also, the 3rd instruction cannot be completed until the 2nd instruction has read its operands. Otherwise the 3rd instruction may overwrite the operand of the 2nd instruction. The result of the 3rd instruction has an anti-dependency on the 1st operand of the 2nd instruction.



Register renaming: How register renaming resolves those issues?

Those issues take place simply due to the fact that we are reusing the registers to store results. It is clearly a resource (registers are the resources) dependency - or a storage conflict. There is no one-to-one relationship between values and registers because we reuse them to store multiple values. If for every value we use a unique register we would not experience this problem. That's why we need register renaming.

Processors internally have a much bigger register file than the one that show to the programmers. The registers that are visible to the programmers are called architectural registers and are defined by the instruction set of the architecture while those underlying (hidden) registers are called physical registers. There is a mechanism that maps architectural registers to the underlying physical registers. When an instruction is executed, it will write the results using physical registers. These results are not visible to

the outside until the very end, where they are communicated to the architectural registers and published as an output. In that way all the resource dependencies can be eliminated.

In our example using register renaming will work as follows: R3 -> R3b -> R3c

R3b: = R3 op R5

R4: = R3b + 1

R3c: = R5 + 1

R7: = R3c op R4

The initial R3 (1st instruction) is coloured with red, the next time that we want to use R3 (1st instruction left part) we map it to a different physical register (R3b), which means that we get a unique new R3. In the next use of R3 (2nd instruction) we use that remapped location. If we want to use R3 again (3rd instruction) we remap it again to a new physical register (R3c) and again if we want to reuse R3 (4th instruction) we use the latest mapping of it (R3c). Now for every R3 we have a different location and we do not have anymore those false dependencies. Now we are sure that the 2nd instruction will get the result of the 1st and the 4th will get the result of the 3rd.

1b) According to the assignment description ADD/SUB instructions take 1 cycle to execute while MUL/DIV take 5. In order to solve the assignment I assumed that by "execute" is meant the calculation of the result and the next cycle is responsible to write-back the results. For example, if an ADD instruction is issued on the 1st cycle, on the 2nd cycle the result is ready but on the 3rd the result is written back. In the phase of write-back: the results are broadcasted to the Common Data Bus, the reservation station entries are cleared, the results are written to the register file in the corresponding tag (if it exists) and the tag from the register file gets cleared.

Except from drawing the situation in the register file and the reservation stations for the important cycles, I have also maintained a small table (named "Overview Table") that shows in which cycle an instruction was issued, in which it was executed (results-ready) and in which the results were written-back to the registers file and broadcasted to the CDB. Also, in the tables below that show the situation in the registers file and the reservation stations, I have coloured with red the "busy" entries. A busy entry in the register file is one that is waiting for a value to be written back from a certain instruction. A busy entry in the reservation station is one that is waiting for the result of another instruction.

Overview Table			
	Issued	**Executed (finished)**	**Written-Back**

Overview Table			
ADD R2, R0, R1	1	2	3
MUL R4, R2, R1	1	8	9
DIV R3, R4, R2	2	14	15
SUB R4, R2, R0	2	4	5

Cycle 1: ADD and MUL are issued at the same time (2 instructions per cycle can be dispatched one to ADD/SUB and one to MUL/DIV). MUL is waiting for the result of RS1 (ADD) to start the execution so it “subscribes” for that value. In the register file - register r2 is waiting for the result of RS1 (ADD) and r4 for the result of RS4 (MUL).

	Register File	Tag
r0	10	
r1	20	
r2		RS1 (ADD)
r3		
r4	5	RS4 (MUL)
r5		
r6		

	OP	S1	S2
RS3			
RS2			
RS1	ADD	10	20

	OP	S1	S2
RS6			
RS5			
RS4	MUL	RS1	20

(Continues to the next page…)

Cycle 2: DIV and SUB are issued at the same time (2 instructions per cycle can be dispatched one to ADD/SUB and one to MUL/DIV). SUB and DIV both subscribe for the

result of RS1 (ADD) but DIV also subscribes for the result of RS4 (MUL). In the register file r3 is waiting for the result of RS5 (DIV) and now for r4 the tag of RS4 (MUL), is replaced with the tag RS2 (SUB). That's because in the register file the purpose of the tag is to specify which command will produce the latest value for that register according to the program-order. Also, in this cycle ADD has finished its execution.

	Register File	Tag
r0	10	
r1	20	
r2		RS1 (ADD)
r3		RS5 (DIV)
r4	5	**RS2 (SUB)**
r5		
r6		

	OP	S1	S2
RS3			
RS2	SUB	RS1	10
RS1	ADD	10	20

	OP	S1	S2
RS6			
RS5	DIV	RS4	RS1
RS4	MUL	RS1	20

(Continues to the next page…)

Cycle 3: ADD (was RS1) writes-back the result. It broadcasts it to the CDB and the subscribers RS2 (SUB) and RS4 (MUL) grab it, also the result is written to the register file

in r2. The tag is cleared from the register file as well as the reservation station entry. RS2 (SUB) and RS4 (MUL) start their execution.

	Register File	Tag
r0	10	
r1	20	
r2	30	
r3		RS5 (DIV)
r4	5	RS2 (SUB)
r5		
r6		

	OP	S1	S2
RS3			
RS2	SUB	30	10
RS1			

	OP	S1	S2
RS6			
RS5	DIV	RS4	30
RS4	MUL	30	20

Cycle 5: RS2 (SUB) writes-back the result to the register file. No instruction in the reservation stations has subscribed to it. The tag in the register file is cleared as well as the reservation station entry.

	Register File	Tag
r0	10	
r1	20	
r2	30	
r3		RS5 (DIV)
r4	20	
r5		
r6		

	OP	S1	S2
RS3			
RS2			
RS1			

esult and broa
he reservatio
4 (MUL) . RS

	OP	S1	S2
RS6			
RS5	DIV	RS4	30
RS4	MUL	30	20

	Register File	Tag
r0	10	
r1	20	
r2	30	
r3		RS5 (DIV)
r4	20	
r5		
r6		

	OP	S1	S2
RS3			
RS2			
RS1			

	OP	S1	S2
RS6			
RS5	DIV	600	30
RS4			

result and bi
cleared from the reservation station. The result is also written to the register file in the register r3 and the tag is cleared.

	Register File	Tag
r0	10	
r1	20	
r2	30	
r3	20	
r4	20	
r5		
r6		

	OP	S1	S2
RS3			
RS2			
RS1			

	OP	S1	S2
RS6			
RS5			
RS4			