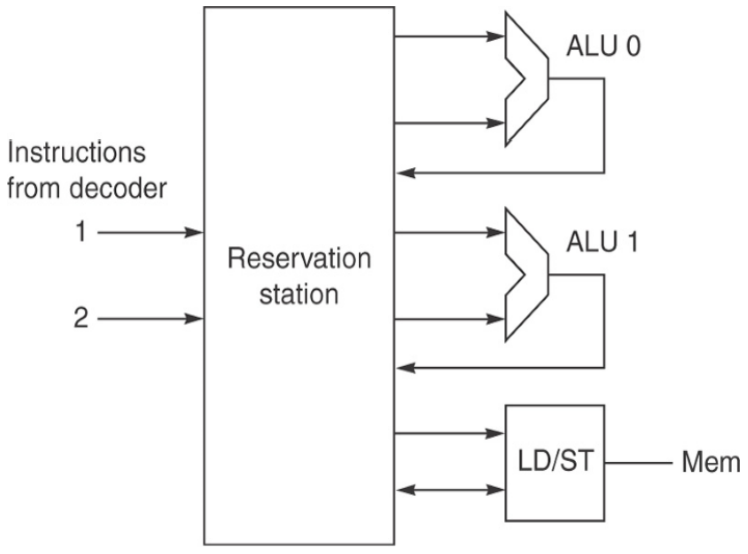
Project's exercises

1) Consider the following code sequence executing on the microarchitecture shown below. Assume that the ALUs can perform all arithmetic and branch operations, and that the centralized Reservation Station (RS) can dispatch at most one operation to each functional unit per cycle (i.e., one instruction to each ALU plus one instruction to LD/ST unit). If more than two ALU instructions can be dispatched, then instructions are dispatched in program order. Assume that dispatching instructions from RS to functional units requires one cycle and once instructions are in the function units they have the latencies shown on right. Also, assume functional unit results can be fully bypassed to subsequent instructions, i.e., when a result is available at cycle *t*, any instructions dependent on the result (and have all their operands available) can be dispatched at cycle *t*+1. All functional units are fully pipelined.

```
Loop:
       L.D   F2,  0(Rx)
       DIV.D F8,  F2, F0
       MUL.D F2,  F6, F2
       L.D   F4,  0(Ry)
       ADD.D F4,  F0, F4
       ADD.D F10, F8, F2
       DADDI Rx,  Rx, #8

       DADDI Ry,  Ry, #8

       S.D   F4,  0(Ry)

       DSUB  R20, R4, Rx

       BNEZ  R20, Loop
```

Latencies	Cycles
DIV.D	10
MUL.D	4
L.D	3
ADD.D	2
DADDI, S.D, BNEZ, DSUB	1



1.1) Suppose all of the instructions from the code sequence above are present in the RS without register renaming at cycle 0. Indicate all the RAW, WAR, and WAW hazards and show how the RS should dispatch these instructions using a timing table similar to the one shown below. The first `L.D` instruction dispatched at cycle 1 is shown.

Cycle	ALU0	ALU1	LD/ST
1			`L.D F2,0(Rx)`
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			

1.2) Now rewrite the code using register renaming. Assume the free list contains rename registers T0, T1, T2, T2, etc. Also, assume these registers can be used to rename both FP and integer registers. Suppose the code with registers renamed is resident in the RS at cycle 0. Show how the RS should dispatch these instructions out-of-order to obtain the optimal performance.

Cycle	ALU0	ALU1	LD/ST
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

1.3) Part (b) assumes that the centralized RS contains all the instructions in the code sequence. But in reality, the entire code sequence of interest is not present in the RS, and thus the RS must choose to dispatch what it has. Suppose the RS is initially empty. In cycle 0, the first two register-renamed instructions of the code sequence appear in the RS. Further assume that the front-end (decoder and register renaming logic) will continually supply two new instructions per clock cycle. Show the cycle-by-cycle order of dispatch of the RS. The contents of the RS for the Cycle 0 and Cycle 1 are shown below.

Cycle 0	Cycle 1	Cycle 2	Cycle 3

`L.D T0,0(Rx)`
`DIV.D T1,T0,F0`
	`DIV.D T1,T0,F0`
	`MUL.D T2,F6,T0`
	`L.D T3,0(Ry)`		

Cycle 4	Cycle 5	Cycle 6	Cycle 7

Cycle 8	…	Cycle 13	Cycle 14

Cycle	ALU0	ALU1	LD/ST
1			`L.D T0,0(Rx)`
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

2) Consider the implementation of the Tomasulo's algorithm with Reorder Buffer (ROB) shown below, which consists of four stages: Issue, Execute, Write-back, and Commit. Simulate the execution of the following piece of code using Tomasulo's algorithm and show the contents of the RS, ROB, and register file entries for each cycle (shown in the following page).

- An ROB entry contains three fields:
  - Committed – Yes (committed) and No (not committed)
  - Dest – destination register identifier ○ Data – value
- In addition to the Busy and Value fields, a register contains ROB # that indicates the ROB entry that will generate the result.
- In addition to Op, Busy, V_j, V_k, Q_j, and Q_k fields, a RS contains Dest field that indicates the ROB entry where the result will be written to. (I left out Busy field because of lack of space).

Assume the following: (1) Dual issue, write-back, and commit, i.e., two instructions can be issued, forwarded to the CDB, and committed per cycle; (2) add latency is 1 cycle and multiply latency is 2 cycles; (3) an instruction can begin execution in the same cycle that it is issued, assuming all dependencies are satisfied. Also, forwarded results are immediately available for use in the next cycle. *Note that this code takes exactly 7 cycles to complete!*

```
ADD.D F4,F0,F8
MUL.D F2,F0,F4
ADD.D F4,F4,F8
MUL.D F8,F4,F2
```