From simple pipelined to superscalar

In this exercise we will see the benefits of superscalar processors which

- have several execution units (so there may be several pipelines running simultaneously)
- use the register renaming technique (to remove false dependencies)
- use out-of-order execution of the code (to minimize the number of delay slots).

Consider the computation

$$((r1 * M1) + ([r2] * M2)) * (M3 + M4)$$

where $M1, \ldots$ denote direct (memory) addressing, $r1$ denotes (direct) register addressing and $[r2]$ denotes register indirect addressing.

1. Write an assembly program typical of RISC machines for this computation. Use at most three registers and store the result in $r1$.
2. Show the execution of your program from item 1 on a five-stage pipelined processor. Assume that instructions are fetched from an onboard instruction cache. Assume that there is an instruction window where all the fetched and decoded instructions can be stored. Use out-of-order execution.
3. Show the dependencies (both true and false) in your code from item 1.
4. Show the execution of your program from item 1 on a superscalar version of the processor from item 2, where for each of the five pipeline stages there are two functional units. Use out-of-order execution.
5. Rewrite your code from item 1 by using register renaming to remove the false dependencies.
6. Show the execution of your program from item 5 on the superscalar processor from item 4.

Solution:

1. Here is an assembly code satisfying the requirements:

```
 I1: LOAD r3, M1
 I2: MUL r1, r3
 I3: LOAD r2, [r2]
 I4: LOAD r3, M2
 I5: MUL r2, r3
 I6: ADD r1, r2
 I7: LOAD r2, M3
 I8: LOAD r3, M4
 I9: ADD r2, r3
I10: MUL r1, r2
```

2. Here is a diagram showing the out-of-order execution of the code. E.g., processing I7 (EX in cycle 9) can start before processing I5 (RR in cycle 10), but we have to be careful not to mess up the program semantics: I7 can go to WB (in cycle 14) only after I6 has read r2 (in cycle 13).

	IF	ID	IW	RR	EX	WB	Comments
1	I1						
2	I2	I1					
3	I3	I2			I1		
4	I4	I3	I2			I1	r3 not ready
5	I5	I4	I3	I2			I3 needs RR!!!
6	I6	I5	I4	I3	I2		
7	I7	I6	I5,I4		I3	I2	r2,r3 not ready
8	I8	I7	I6,I5		I4	I3	r2 not ready
9	I9	I8	I6,I5		I7	I4	r2 in use
10	I10	I9	I8,I6	I5			r3 in use
11		I10	I9,I8,I6		I5		r2,r3 not ready
12			I10,I9,I6		I8	I5	r1,r2 not ready
13			I10,I9	I6		I8	
14			I10,I9		I6	I7	
15			I10,I9			I6	
16			I10	I9			
17			I10		I9		
18			I10			I9	
19				I10			
20					I10		
21						I10	

3. Write–Read (WR) or Read–After–Write (RAW), a true data dependency: all the pairs of occurrences of a register ri where instruction Ij writes to ri and instruction Ik reads from ri with j<k. For instance, I1-I2 on r3, I2-I6 on r1, I5-I6 on r2, I9-I10 on r2..

   Write–write (WW) or Write–After–Write (WAW), a false dependency: all the pairs of occurrences of a register ri where instruction Ij writes to ri and instruction Ik also writes to ri with j<k. For instance, I1-I4 on r3.

   Read–Write (RW), or Write–After–Read (WAR), a false dependency: all the pairs of occurrences of a register ri where instruction Ij reads from ri and instruction Ik writes to ri with j<k. For instance, I2-I4 on r3.

4. We can avoid some delay slots using the extra functional units and out-of-order execution, but we have to be careful not to mess up the program semantics (e.g., processing I4 can start earlier, but it has to wait with WB until I2 has read r3).

	IF	ID	IW	RR	EX	WB	Comments
1	I1,I2						
2	I3,I4	I1,I2					
3	I5,I6	I3,I4	I2		I1		r3 not ready
4	I7,I8	I5,I6	I2	**I3**	**I4**	I1	
5	I9,I10	I7,I8	I5,I6	I2	I3		r3 in use, r1,r2 not ready
6		I9,I10	I5,I6,I8		I2,**I7**	I3,I4	EX busy
7			I6,I9,I10	I5	**I8**	I2	r1,r2,r3 not ready
8			I6,I9,I10		I5	I8	
9			I6,I9,I10			I5	
10			I9,I10	I6			
11			I9,I10		I6	I7	
12			I10	I9		I6	
13			I10		I9		
14			I10			I9	
15				I10			
16					I10		
17						I10	

5. We can rename r3 in I4,I5 to r3a (removing the WAW I4-I1 and WAR I4-I2 on r3) r2 in I7,I9,I10 to r2a (removing the WAW I7-I5 and WAR I7-I5 on r2) and r3 in I8,I9 to r3b (removing the WAW I8-I4 and WAR I8-I5 on r3).

```
 I1: LOAD r3, M1
 I2: MUL r1, r3
 I3: LOAD r2, [r2]
 I4: LOAD r3a, M2
 I5: MUL r2, r3a
 I6: ADD r1, r2
 I7: LOAD r2a, M3
 I8: LOAD r3b, M4
 I9: ADD r2a, r3b
I10: MUL r1, r2a
```

6. We can get rid of more delay slots due to the removal of false dependencies.

	IF	ID	IW	RR	EX	WB	Comments
1	I1,I2						
2	I3,I4	I1,I2					
3	I5,I6	I3,I4	I2		I1		r3 not ready
4	I7,I8	I5,I6	I2	I3	I4	I1	
5	I9,I10	I7,I8	I5,I6	I2	I3	I4	r2,r3a not ready
6		I9,I10	I5,I6,I8		I2,I7	I3	EX busy
7			I6,I9,I10	I5	I8	I2,I7	r3b,r2a,r1 not ready
8			I6,I9,I10		I5	I8	
9			I6,I10	**I9**		I5	
10			I10	I6	I9		
11			I10		I6	I9	
12			I10			I6	
13				I10			
14					I10		
15						I10	