Comprehensive Example

Consider the computation $(M1 * M2) + (M3 * M4)$.

1. Write a program in assembly language that performs the above computation. Try to use a minimal number of registers.

2. Draw a diagram showing the execution of your program on a five-stage pipeline using in-order execution.

3. Draw a diagram showing the execution of your program on a five-stage pipeline using out-of-order execution.

4. Identify the dependencies in your code.

5. Remove as many dependencies as possible from your code (by using register renaming) and reorder the code to minimize the number of delay slots when executed on a five-stage pipeline. Show the pipeline activity in a diagram.

You can assume that there is an onboard instruction cache from which the instructions are fetched (so there is no resource conflict between fetching an instruction and executing a data-transfer instruction) and that there is an instruction window, where fetched and decoded instructions are stored.

Solution to Comprehensive Example

1. Here is the code:

```
I1  LOAD r1, M1
I2  LOAD r2, M2
I3  MUL r1, r1, r2
I4  LOAD r2, M3
I5  LOAD r3, M4
I6  MUL r2, r3, r2
I7  ADD r1, r1, r2
```

2. Here is the diagram showing delay slots for in-order execution:

	IF	ID	IW	RR	EX	WB	Comments
1	I1						
2	I2	I1					
3	I3	I2			I1		I1 skips RR
4	I4	I3			I2	I1	I2 skips RR
5	I5	I4	I3			I2	r2 not ready
6	I6	I5	I4	I3			
7	I7	I6	I5,I4		I3		I4,I5 skip RR
8		I7	I6,I5		I4	I3	r2,r3 not ready
9			I7,I6		I5	I4	r2 not ready
10			I7,I6			I5	
11			I7	I6			
12			I7		I6		
13			I7			I6	
14				I7			
15					I7		
16						I7	

3. Here is the diagram showing delay slots for out-of-order execution:

	IF	ID	IW	RR	EX	WB	Comments
1	I1						
2	I2	I1					
3	I3	I2			I1		I1 skips RR
4	I4	I3			I2	I1	I2 skips RR
5	I5	I4	I3			I2	r2 not ready
6	I6	I5		I3	**I4**		I4 skips RR
7	I7	I6	I5		I3	**I4**	I5 skips RR
8		I7	I6		I5	I3	r3 not ready
9			I7,I6			I5	r2 not ready
10			I7	I6			
11			I7		I6		
12			I7			I6	
13				I7			
14					I7		
15						I7	

4. We have the following dependencies:
W-R: I1-13(r1), I2-I3(r2), I3-I7(r1), I4-I6(r2), I5-I6(r3), I6-I7(r2)
W-W: I1-I3(r1), I1-I7(r1), I2-I4(r2), I2-I6(r2), I3-I7(r1), I4-I6(r2)
R-W: I3-I4(r2), I3-I6(r2), I3-I7(r1)

5. Here is the code using register renaming (to remove R-W dependencies)

```
I1   LOAD r1, M1
I2   LOAD r2, M2
I3   MUL r1, r1, r2
I4'  LOAD r12, M3
I5   LOAD r3, M4
I6'  MUL r12, r3, r12
I7'  ADD r1, r1, r12
```

We replaced r2 by r12 in I4 to remove the R-W dependency I3-I4(r2). Note that we have to make the same replacements in I6 and I7 to maintain the semantics of the code (i.e., to maintain the data dependencies between I4 and I6 and I7).

After removing the dependency I3-I4(r2) the code can be reorganized as follows.

```
I1   LOAD r1, M1
I2   LOAD r2, M2
I4'  LOAD r12, M3
I5   LOAD r3, M4
I3   MUL r1, r1, r2
I6'  MUL r12, r3, r12
I7'  ADD r1, r1, r12
```

Here is the diagram showing the execution of the reorganized code.

	IF	ID	IW	RR	EX	WB	Comments
1	I1						
2	I2	I1					
3	I4'	I2			I1		I1 skips RR
4	I5	I4'			I2	I1	I2 skips RR
5	I3	I5			I4'	I2	I4' skips RR
6	I6'	I3			I5	I4'	I5 skips RR
7	I7'	I6'		I3		I5	
8		I7'		I6'	I3		
9			I7'		I6'	I3	r1,r12 not ready
10			I7'			I6'	
11				I7'			
12					I7'		
13						I7'	