Simple Example — Memory Addressing

Recall that modern processors use (several levels of) caches so that instructions and data can be loaded into the CPU faster than from main memory. In this exercise we will see the benefit of caches when a simple code is executed on a pipelined processor. In particular, we will assume that there are separate, onboard caches for data and instructions, and see that this helps getting rid of some delay slots (vacant pipeline stages during the execution of the code).

Consider the computation

$$(M1 + M2) * (M3 + M4)$$

where $M1, \ldots$ denote direct (memory) addressing.

1. Write an assembly program typical of RISC machines for this computation. First load the data and then process the arithmetic operations. You can assume that there are an unlimited number of registers available and store the result in register $r1$.

2. Show the execution of your program from item 1 on a pipelined processor. There are five pipeline stages: IF, ID, RR, EX, WB. Assume that instructions and data have to be loaded from main memory. Explain what happens during the pipeline stages for the various instructions and whether certain instructions can skip some of the pipeline stages.

3. Repeat the previous item, but this time assume that instructions and data are fetched from onboard instruction and data caches, respectively, thus there is no resource conflict on the bus.

Solution to Simple Example — Memory Addressing

Solution:

1. Here is an assembly code satisfying the requirements:

```
I1: LOAD r1, M1
I2: LOAD r2, M2
I3: LOAD r3, M3
I4: LOAD r4, M4
I5: ADD r1, r2
I6: ADD r3, r4
I7: MUL r1, r3
```

2. All instruction go through IF and ID. Using direct memory addressing does not need reading registers, so I1, I2, I3 and I4 skip RR. We assume that data transfer between main memory and the MBR of the CPU happens during EX. Finally, the data is copied from the MBR into the target register during WB. Arithmetic instructions need RR, EX and WB.

3. Here is a diagram showing the execution of the code. The sign **X** indicates where delay slots occur due to traffic on the bus.

	IF	ID	RR	EX	WB	Comments
1	I1					
2	I2	I1				
3	**X**	I2		I1		I1 skips RR, data bus busy
4	**X**			I2	I1	I2 skips RR, data bus busy
5	I3				I2	
6	I4	I3				
7	**X**	I4		I3		I3 skips RR, data bus busy
8	**X**			I4	I3	I4 skips RR, data bus busy
9	I5				I4	
10	I6	I5				
11	I7	I6	I5			
12		I7	I6	I5		
13		I7		I6	I5	r1,r3 not ready
14		I7			I6	r3 not ready
15			I7			
16				I7		
17					I7	

Observe the parallel processing of I5 and I6, and the delay in processing I7 (due to data dependencies).

4. Here is the execution using caches. In this case, instructions still have to be loaded from the cache and then decoded, thus we need the stages IF and ID. However, loading data from the onboard cache into the registers is much faster than loading from main memory, thus this can be done during WB (hence we do not need EX for load instructions). **Note that, in general, there is no guarantee that the cache contains the required data, here we consider the optimal case!**

	IF	ID	RR	EX	WB	Comments
1	I1					
2	I2	I1				
3	I3	I2			I1	I1 skips RR,EX
4	I4	I3			I2	I2 skips RR,EX
5	I5	I4			I3	I3 skips RR,EX
6	I6	I5			I4	I4 skips RR,EX
7	I7	I6	I5			
8		I7	I6	I5		
9		I7		I6	I5	r1,r3 not ready
10		I7			I6	r3 not ready
11			I7			
12				I7		
13					I7	