Solutions to exercises on Basic Instruction Level Parallelism

J. Daniel García Sánchez (coordinator)
David Expósito Singh
Javier García Blas

Computer Architecture
ARCOS Group
Computer Science and Engineering Department
University Carlos III of Madrid

1 Exam exercises

Exercise 1

Given the following code fragment:

```
Loop:   LD R4, 0(R2)
        LD R5, 0(R3)
        ADD R6, R4,R5
        SD R6, 0(R3)
        ADD R6, R6, R4
        ADDI R3, R3, #8
        SUB R20, R4, R3
        BNZ R20, Loop
```

1. Enumerate the existing data dependencies in the code. For each one determine the element causing the dependency.
2. Provide a timing for this sequence for the 5 stages RISC pipeline without forwarding or bypassing hardware, but assuming that a data read and a data write to the register file can be performed in the same cycle (assuming writes to the register file are performed in the first half-cycle and reads are performed in the second half-cycle). Assume that branches are handled by flushing the pipeline and that all memory accesses (including instruction fetch) take two cycles. Justify your answer.

Solution 1

Section 1 The following data dependencies can be identified:

- **R4**: **I1** → **I3** (RAW).
- **R5**: **I2** → **I3** (RAW).
- **R6**: **I4** → **I3** (RAW).
- **R6**: **I5** → **I4** (WAR).

- **R6**: **I5** → **I3** (WAW).
- **R4**: **I5** → **I1** (RAW).
- **R3**: **I6** → **I4** (WAR).
- **R3**: **I7** → **I6** (RAW).
- **R20**: **I8** → **I7** (RAW).

If a read and a write of a data in the register file can be done in the same clock cycle, then the Decoding (ID) and write-back (WB) can be done in the same cycle.

Section 2 Table 1 shows the corresponding timing diagram.

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I1: **LD R4**, **0**(**R2**)	IF	IF	ID	EX	M	M	WB								
I2: **LD R5**, **0**(**R3**)			IF	IF	ID	EX	M	M	WB						
I3: **ADD R6**,**R4**,**R5**					IF	IF	ID	–	–	EX	M	WB			
I4: **SD R6**, **0**(**R3**)							IF	IF	–	ID	–	–	EX	M	M
I5: **ADD R6**,**R6**,**R4**										IF	IF	–	ID	EX	–
I6: **ADDI R3**,**R3**, #**8**													IF	IF	ID
I7: **SUB R20**,**R4**,**R3**															IF
I8: **BNZ R20**, **Loop**															
I9															
I9'															

Instr.	16	17	18	19	20	21	22	23	24	25
I1: **LD R4**, **0**(**R2**)										
I2: **LD R5**, **0**(**R3**)										
I3: **ADD R6**,**R4**,**R5**										
I4: **SD R6**, **0**(**R3**)	WB									
I5: **ADD R6**,**R6**,**R4**	M	WB								
I6: **ADDI R3**,**R3**, #**8**	EX	M	WB							
I7: **SUB R20**,**R4**,**R3**	IF	ID	–	EX	M	WB				
I8: **BNZ R20**, **Lopp**		IF	IF	ID	–	–	EX	M	WB	
I9				IF						
I9'							IF	IF	ID	EX

Table 1: Time diagram for exercise 1.

Exercise 2

A certain processor runs the following code segment:

```
i0:    lw $r3, 0($r0)
i1:    lw $r1, 0($r3)
i2:    addi $r1, $r1, 1
i3:    sub $r4, $r3, $r2
i4:    sw $r1, 0($r3)
i5:    bnz $r4, i0
```

Assume that the processor has 5 stages pipelined architecture (fetch, decode, execute, memory and write-back) without forwarding. All stages run in one cycle, except load and store operations which require two additional cycles for memory access latency, and branching instructions which require one additional execution cycle. Assume that both the banch target address and the evaluation of the branch condition are known at the end of the decode stage.

1. Identify RAW data hazards in the code.
2. Show a timing diagram with execution stages for each instruction in one iteration.
3. Determine how many cycles are required for executing one loop iteration when there is no branch prediction.
4. Determine how many cycles are required for executing one loop iteration when a branch predictor (always predicting to taken) is used.

Solution 2

Section 1

- **$r3**: **i0** → **i1**
- **$r1**: **i1** → **i2**
- **$r1**: **i2** → **i4**
- **$r4**: **I3** → **i5**

Section 2 Table 2 shows the corresponding timing diagram.

- **I0**: Requires three memory cycles
- **I1**: You can not start decoding until WB of **$r3** is done. Requires 3 cycles of memory.
- **I2**: It can not be fetched until the stage is released by **I1**. You can not start to decode until the WB of **$r1**
- **I4**: The decoding can not be started until **I2** does WB of **$r1**
- **I5**: Fetch can not start until **I4** decoding starts. The memory cycle cannot be started until the memory cycle of **I4** does not end.

Section 3 In this case, instruction **I0** from next iteration cannot be fetched until both the branch target address and the branch condition result are known. This happens at the end of decode stage for **I5**. Consequently, **I0** is fetched in cycle 17 and one iteration requires 16 cycles.

Section 4 In this case, the prediction is performed in the decoding step, which is when it is known that it is a branch instruction, so **I0** starts in cycle 17, and one iteration requires 16 cycles.

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I0: **lw $r3**, 0(**$r0**)	IF	ID	EX	M	M	M	WB								
I1: **lw $r1**, 0(**$r3**)		IF	ID	–	–	–	–	EX	M	M	M	WB			
I2: **addi $r1**, **$r1**, 1			IF	–	–	–	–	ID	–	–	–	–	EX	M	WB
I3: **sub $r4**, **$r3**, **$r2**								IF	–	–	–	–	ID	EX	M
I4: **sw $r1**, 0(**$r3**)													IF	ID	–
I5: **bnz $r4**, **I0**														IF	–

Instr.	16	17	18	19	20	21
I0: **lw $r3**, 0(**$r0**)						
I1: **lw $r1**, 0(**$r3**)						
I2: **addi $r1**, **$r1**, 1						
I3: **sub $r4**, **$r3**, **$r2**	WB					
I4: **sw $r1**, 0(**$r3**)	EX	M	M	M	WB	
I5: **bnz $r4**, **I0**	ID	EX	EX	–	M	WB

Table 2: Timing diagram for exercise 2.

Exercise 3

The following code is written in MIPS assembler. Assume that, before starting instructions execution, registers **R3** and **R5** contain, respectively, the memory addresses of the first and last element in an array with 9 entries (initial value for **R1**=**0x010** and **R5**=**0x018**).

```
Loop:  LD    R4  0(R1)
       DIV   R2  R2  R4
       ADD   R1  R1  #1
       SUB   R5  R5  #1
       SD    R4  0(R5)
       SUB   R6  R1  R5
       BNEZ  R6  Loop
```

1. Express all RAW and WAW data hazards in the code.
2. Provide a timing diagram assuming that the processor is pipelined with 5 stages (*fetch*, *decode*, *execution*, *memory* y *write back*). An instruction per cycle is issued and the processor **does not use forwarding**. Assume that there is pipeline freezing for branches and that **there is one additional cycle per memory access in reads (LD)**, which does not happen in writes. Effective branch address and branch outcome are determined during decode stage.
3. Determine the number of cycles needed by the loop (all iterations) to run.

Solution 3

Section 1 When instructions **I1** and **I7** are executed, (being **I1** the first one), the following hazards are obtained:

- **R4**: **I1** → **I2**
- **R4**: **I1** → **I5**
- **R5**: **I4** → **I5**
- **R5**: **I4** → **I6**
- **R1**: **I3** → **I6**
- **R6**: **I6** → **I7**

Section 2 Table 3 shows the corresponding time diagram. The solution assumes that the branch target address is known at the end of the instruction decode stage.

Section 3 Total execution cycles of a loop iteration: 15

Given that there are 4 iterations (values of **r1** and **r5**: (0x010, 0x018), (0x011, 0x017),(0x012, 0x016), and (0x013, 0x015) respectively), we have:

$$15 \text{ cycles} \cdot 4 \text{ iterations} = 60 \text{ cycles}$$

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1: LD R4 0(R1)	IF	ID	EX	M	M	WB								
I2: DIV R2 R2 R4		IF	ID	–	–	–	EX	M	WB					
I3: ADD R1 R1 #1			IF	–	–	–	ID	EX	M	WB				
I4: SUB R5 R5 #1							IF	ID	EX	M	WB			
I5: SD R4 0(R5)								IF	ID	–	–	EX	M	WB
I6: SUB R6 R1 R5									IF	–	–	ID	EX	M
I7: BNEZ R6 Loop												IF	ID	–
I8													IF	flush

Instr.	15	16	17	18	19	20
I1: LD R4 0(R1)						
I2: DIV R2 R2 R4						
I3: ADD R1 R1 #1						
I4: SUB R5 R5 #1						
I5: SD R4 0(R5)						
I6: SUB R6 R1 R5	WB					
I7: BNEZ R6 Loop	–	EX	M	WB		
I8						
I1 / I8		IF	ID	EX	M	WB

Table 3: Time diagram for exercise 3.

Exercise 4

Consider the following code fragment:

```
Loop:   LD R4,0(R2)
        LD R5,0(R3)
        ADD R6,R4,R5
        SD R6,0(R3)
        BNZ R6,Loop
```

1. Number of needed cycles to run one loop iteration in a non-pipelined processor. Memory access instructions have a 3 cycles latency. Branch instruction has a 1 cycle latency. All these latencies are considered in addition to the instruction fetch/issue cycle.
2. Identify RAW data dependencies in the code.
3. Compute the number of needed cycles to run one iteration of the loop in a 5 stages pipelined processor. The processor uses the forwarding technique and the branch handling strategy is pipeline freezing. Complete a timing diagram.

Solution 4

Section 1 Analysing separately the different types of instructions:
3 memory instruction · (1 issue + 3 latency) = 12 cycles
1 ALU instruction · 1 issue = 1 cycle
1 branch instruction · (1 issue + 1 latency) = 2 cycles
Total = 15 cycles

Section 2 The following dependencies are identified:

- **R4**: **LD** → **ADD**.
- **R5**: **LD** → **ADD**.
- **R6**: **ADD** → **SD**.

Section 3 Table 4 shows the timing diagram:

Instr.	1	2	3	4	5	6	7	8	9	10	11	12
I1: **LD R4,0(R2)**	IF	ID	EX	M	WB							
I2: **LD R5,0(R3)**		IF	ID	EX	M	WB						
I3: **ADD R6,R4,R5**			IF	ID	–	EX	M	WB				
I4: **SD R6,0(R3)**				IF	–	ID	EX	M	WB			
I5: **BNZ R6, Loop**						IF	ID	EX	M	WB		
I1 / **I6**								IF	ID	EX	M	WB

Table 4: Timing diagram for exercise 4.

Exercise 5

Given the following code written in MIPS assembler, where the same memory address is read and written several times and with values for **R1** and **R2** are **R1**=**1** and **R2**=**1000**.

```
        ADDI R3,R3,1
Loop:   LD R4,0(16)
        MULT R5,R4,R4
        ADD R5,R3,R5
        SD R5,0(16)
        DADDI R3,R4,1
        DADDI R1,R1,1
        BNE R1,R2,Loop
```

The code runs in a MIPS-like pipelined processor with the following execution stages: fetch, decode, execute, memory and write-back. The processor issues one instruction per cycle and has forwarding capability. All the stages in the data path run in one cycle except the following cases: **SD** instruction uses and additional cycle to read **register R5 from the register file**, instruction **LD** uses an additional cycle to write **value from memory into register R4 from the register file** and instructions **ADD** and **MULT** use an additional cycle to complete its execution in the ALU. Assume that branch prediction strategy is *not taken*.

1. Express only WAW data hazards in the code showing the instructions causing the hazard and the associated register. In which situation could a WAW hazard cause an incorrect result for the program?
2. Draw a time diagram assuming forwarding. Compute the number of cycles that the program would take to execute.

Solution 5

Section 1

- **WAW**: **I4** con **I3** en **R5**.
- **WAW**: **I1** con **I6** en **R3**.

An incorrect result would be obtained if **I4** is executed before **I3** on pipelined processors where instructions can be reordered.

Section 2 Table 5 shows the corresponding timing diagram.
The cycles required to execute the loop can be divided:

- Number of cycles before the loop: 1.
- Number of cycles per iteration of the loop: 12.
- Number of cycles for the last loop iteration: 10.

The loop executes a total of 999 times. Therefore the number of cycles will be:

$$\text{cycles} = 1 + (12 \cdot 998) + 10 = 11987$$

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13
I1: ADDI R3 ,R3, 1	IF	ID	EX	M	WB								
I2: LD R4 , 0(16)		IF	ID	EX	M	WB	WB						
I3: MULT R5 ,R4 ,R4			IF	ID	–	EX	EX	M	WB				
I4: ADD R5 ,R3 ,R5				IF	–	ID	–	EX	EX	M	WB		
I5: SD R5 , 0(16)						IF	–	ID	ID	EX	M	WB	
I6: DADDI R3 ,R4, 1								IF	–	ID	EX	M	WB
I7: DADDI R1 ,R1, 1										IF	ID	EX	M
I8: BNE R1 ,R2 , Loop											IF	ID	–
I9: (sig)												IF	–
I1: ADDI R3 ,R3, 1													

Instr.	14	15	16
I1: ADDI R3 ,R3, 1			
I2: LD R4 , 0(16)			
I3: MULT R5 ,R4 ,R4			
I4: ADD R5 ,R3 ,R5			
I5: SD R5 , 0(16)			
I6: DADDI R3 ,R4, 1			
I7: DADDI R1 ,R1, 1	WB		
I8: BNE R1 ,R2 , Loop	EX	M	WB
I9: (sig)	ID	EX	M
I1: ADDI R3 ,R3, 1	IF	ID	EX

Table 5: Timing Diagram of Exercise 5.

Exercise 6

Let's consider the following code fragment:

```
loop:  lw $f0, 0($r1)
       lw $f2, 0($r2)
       mul.f $f4, $f0, $f2
       add.d $f6, $f6, $f4
       addi $r1, $r1, 4
       addi $r2, $r2, 4
       sub $r3, $r3, 1
       bnez $r3, loop
```

1. Make a list with all possible data dependencies, without considering a specific structure of the pipielined architecture. For each dependency you must provide, register, source instruction, target instruction and type of dependency.

2. Create a time diagram for a MIPS architecture with a 5 stages pipeline and the following considerations:

   - There is no forwarding hardware
   - The architecture allows an instruction to write in a register and other instruction to read that same register in the same cycle without any conflict.
   - The branches are processed by flushing the pipeline.
   - Memory references require a clock cycle.
   - The effective branch address is computed during the execution stage.

3. Determine how many cycles are needed to execute N loop iterations.

4. Create a time diagram for a MIPS architecture with a 5 stages pipeline and the following considerations:

   - There is fordwarding hardware
   - Assume that branches are handled by predicting all branches as taken.

5. Determine how many cycles are needed to run N loop iterations with the conditions of paragraph 4.

Solution 6

Data dependencies If the instructions are numbered from **I1** (first instruction) to **I8** (last instruction) we have the following dependencies:

- **$f0**: **I1** → **I3** (RAW)
- **$f2**: **I2** → **I3** (RAW)
- **$f4**: **I3** → **I4** (RAW)
- **$r3**: **I7** → **I8** (RAW)

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13
I1: lw $f0, 0($r1)	IF	ID	EX	M	WB								
I2: lw $f2, 0($r2)		IF	ID	EX	M	WB							
I3: mul.f $f4, $f0, $f2			IF	ID	–	–	EX	M	WB				
I4: add.d $f6, $f6, $f4				IF	–	–	ID	–	–	EX	M	WB	
I5: addi $r1, $r1, 4							IF	–	–	ID	EX	M	WB
I6: addi $r2, $r2, 4										IF	ID	EX	M
I7: sub $r3, $r3, 1											IF	ID	EX
I8: bnez $r3, loop												IF	ID
I9:													IF
I1: lw $f0, 0($r1)													

Instr.	14	15	16	17	18	19	20	21
I1: lw $f0, 0($r1)								
I2: lw $f2, 0($r2)								
I3: mul.f $f4, $f0, $f2								
I4: add.d $f6, $f6, $f4								
I5: addi $r1, $r1, 4								
I6: addi $r2, $r2, 4	WB							
I7: sub $r3, $r3, 1	M	WB						
I8: bnez $r3, loop	–	–	EX	M	WB			
I9:	flush							
I1: lw $f0, 0($r1)				IF	ID	EX	M	WB

Table 6: First timing diagram for exercise 6.

First timming diagram Given that there is no forwarding, when there is a RAW dependency we have to wait for the WB of the source instruction before starting the execution of the destination instruction. Table 6 shows the corresponding timing diagram.

- Instruction I3 has a stall until I2 has written the value read in **$f2**. We can perform the reading of register file in the same cycle than the one in which I2 writes in the register file (cycle 6).
- Instruction I4 can not start until the fetch unit is released (cycle 6).
- Instruction I4 has a stall until I3 has written the value calculated in **$f4**. We can perform the reading of register file in the same cycle than the one in which I3 writes in the register file (cycle 9).
- Instruction I5 can not start until the Fetch unit is available (cycle 9).
- Instruction I8 has a stall until I7 has written the value **$r3**. We can perform the reading of register file in the same cycle in which I7 writes in the register file (cycle 15).
- Instruction I9 (the next to bnez) can not start fetching until the fetch unit is available (cycle 15).
- Although the branch address is known at the end of the decoding of I8, the branch result (taking or not taking) is not known until the end of the execution. stage Therefore the instruction fetch is repeated in the cycle 17.

First cycle estimation To determine the number of cycles, we need to determine how many cycles are required for any iteration and how many for the last iteration.

The cost of a different iteration of the last is obtained by determining the number of cycles from the start of execution of I1 until it is executed again. This is 16 cycles.

The cost of the last iteration is obtained by determining the number of cycles until the first instruction outside the loop is fetched. Those are 16 cycles. Note that the pipeline is always flushed after a branch instruction as there is no prediction.

$$Cost = 16 \cdot n$$

.

Second cycle estimation Forwarding is now allowed whenever possible and there is no need to wait for the WB stage.

Table 7 shows the timing diagram:

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13
I1: **lw $f0**, 0(**$r1**)	IF	ID	EX	M	WB								
I2: **lw $f2**, 0(**$r2**)		IF	ID	EX	M	WB							
I3: **mul.f $f4**, **$f0**, **$f2**			IF	ID	–	EX	M	WB					
I4: **add.d $f6**, **$f6**, **$f4**				IF	–	ID	EX	M	WB				
I5: **addi $r1**, **$r1**, **4**						IF	ID	EX	M	WB			
I6: **addi $r2**, **$r2**, **4**							IF	ID	EX	M	WB		
I7: **sub $r3**, **$r3**, **1**								IF	ID	EX	M	WB	
I8: **bnez $r3**, **loop**									IF	ID	–	EX	M
I9:										IF			
I1: **lw $f0**, **0**(**$r1**)													IF

Instr.	14	15	16	17	18	19	20	21
I1: **lw $f0**, 0(**$r1**)								
I2: **lw $f2**, 0(**$r2**)								
I3: **mul.f $f4**, **$f0**, **$f2**								
I4: **add.d $f6**, **$f6**, **$f4**								
I5: **addi $r1**, **$r1**, **4**								
I6: **addi $r2**, **$r2**, **4**								
I7: **sub $r3**, **$r3**, **1**								
I8: **bnez $r3**, **loop**	WB							
I9:								
I1: **lw $f0**, **0**(**$r1**)	ID	EX	M	WB				

Table 7: Second time diagram of exercise 6.

- Instruction I3 can start execution after the memory stage of I2 (cycle 6) because of the forwarding.
- Instruction I4 can not start decoding until the decoding unit has been released (cycle 6).
- Instruction I4 can start execution after the execution stage of I3 (cycle 7) because of the forwarding.

- Instruction I5 can not start the fetch until the fetch unit is released (cycle 6).
- Instruction I8 must stall in decode until the value of **$r3** (cycle 10) has been computed and passed through forwarding (cycle 11).
- Instruction I9 is fetched in cycle 10. When instruction I8 is decoded in cycle 10, a branch instruction is detected and, as branches are predicted to taken, instruction I1 is fetched again in cycle 13, when the effective branch address is known (it is computed in cycle 12).

Second cycle estimation The cost of an iteration different from the last iteration is 12 cycles. The last iteration requires 12 cycles.

$$Cost = 12 \cdot n$$

Exercise 7

Consider the following code fragment:

```
loop:  lw $f0, 0($r1)
       lw $f2, 0($r2)
       sub.f $f4, $f0, $f2
       mul.d $f4, $f4, $f4
       add.d $f6, $f6, $f4
       addi $r1, $r1, 4
       addi $r2, $r2, 4
       sub $r3, $r3, 1
       bnez $r3, loop
```

1. Make a list with all possible data dependencies, without considering any specific structure of the pipelined architecture. For each dependency specify register, source instruction, target instruction and dependency type.
2. Create a time diagram for a MIPS architecture with a 5 stages pipeline and the following considerations:
   - There is no forwarding hardware.
   - The architecture allows instructions to write to a register and another instruction to read that same register in the same clock cycle without any conflict.
   - Branches are treated by flushing the pipeline.
   - Memory references require one clock cycle.
   - The effective branch address is computed at the execution stage.
3. Determine how many cycles are needed to execute N loop iterations.
4. Create a time diagram for a MIPS architecture with a 5 stages pipeline and the following considerations:
   - There is full forwarding hardware.
   - Assume that branches are handled by predicting all branches as taken.
5. Determine how many cycles are needed to run N iterations with the conditions from paragraph 4.

Solution 7

Data dependencies If the instructions are numbered from **I1** (first statement) to **I9** (last instruction) have the following dependencies:

- **$f0**: **I1** → **I3** (RAW)
- **$f2**: **I2** → **I3** (RAW)
- **$f4**: **I3** → **I4** (RAW, WAW)
- **$f4**: **I4** → **I5** (RAW)
- **$r3**: **I8** → **I9** (RAW)

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1: lw $f0, 0($r1)	IF	ID	EX	M	WB									
I2: lw $f2, 0($r2)		IF	ID	EX	M	WB								
I3: sub.f $f4, $f0, $f2			IF	ID	–	–	EX	M	WB					
I4: mul.d $f4, $f4, $f4				IF	–	–	ID	–	–	EX	M	WB		
I5: add.d $f6, $f6, $f4							IF	–	–	ID	–	–	EX	M
I6: addi $r1, $r1, 4										IF	–	–	ID	EX
I7: addi $r2, $r2, 4													IF	ID
I8: sub $r3, $r3, 4														IF
I9: bnez $r3, loop														
I10:														
I1: lw $f0, 0($r1)														

Instr.	15	16	17	18	19	20	21
I1: lw $f0, 0($r1)							
I2: lw $f2, 0($r2)							
I3: sub.f $f4, $f0, $f2							
I4: mul.d $f4, $f4, $f4							
I5: add.d $f6, $f6, $f4	WB						
I6: addi $r1, $r1, 4	M	WB					
I7: addi $r2, $r2, 4	EX	M	WB				
I8: sub $r3, $r3, 4	ID	EX	M	WB			
I9: bnez $r3, loop	IF	ID	–	–	EX	M	WB
I10:		IF					
I1: lw $f0, 0($r1)						IF	ID

Table 8: First time diagram for exercise 7.

First timing diagram When there is no forwarding, when there is a RAW dependency, it is necessary to wait for the cycle WB from the source instruction before starting the target instruction cycle. Table 8 shows the corresponding timing diagram.

- Instruction I3 has a stall until I2 has written the value read from **$f2**. We can perform the register file reading in the same cycle than the one in which I2 writes in the register bank (cycle 6).
- Instruction I4 can not start the fetch until the fetch unit is released (cycle 6).
- Instruction I4 has a stall until I3 has written the value calculated in **$ f4**. We can perform the reading of the register file in the same cycle as I3 writes in the register file (cycle 9).
- Instruction I5 can not start the fetch until the fetch unit is released (cycle 9).
- Instruction I5 has a stall until I4 has written the value calculated from **$f4**. We can perform the reading of the register file in the same cycle as I4 writes in the register file (cycle 12).
- Instruction I6 can not start the fetch until the fetch unit is released (cycle 12).
- Instruction I9 has a stall until I8 has written the value **$r3**. We can perform the reading of the register file in the same cycle in which I7 writes in the register file (cycle 18).

- Instruction I10 (the next to bnez) can not start the fetch until the fetch unit is released (cycle 18).
- Although the branch address is known at the end of the Decoding of I9, the branch result (taken or not takend) is not known until the end of the execution stage. Therefore the fetch up is repeated in the cycle twenty.

First cycle estimation To determine the number of cycles, we need to determine how many cycles are require for a generic iteration and how many cycles for the last iteration.

The cost of a different iteration of the last is obtained by determining the number of cycles from the start of the I1 execution until I1 is started again. This is 19 cycles.

The cost of the last iteration is obtained by determining the number of cycles until the first instruction outside the loop is fetched. The pipeline is always flushed after the branch instruction and an instruction is always fetched once the effective branch target address is known. Consequently, last iteration also requires 19 cycles.

$$Cost = 19 \cdot n$$

Second cycle estimation Forwarding is now allowed when is possible to use it and there is no waiting for the WB stage. Table 9 shows the corresponding timing diagram.

- Instruction I3 can start execution after the memory stage of I2 (cycle 6) because of forwarding.
- Instruction I4 can not start fetch until that unit is released (cycle 5).
- Instruction I4 can start execution after the execution stage of I3 (cycle 7) because of forwarding.
- Instruction I5 can not start the fetch until the fetch unit is available (cycle 6).
- Instruction I5 can start execution after the execution stage of I4 (cycle 8) because of forwarding.
- Instruction I9 is decoded in cycle 11, and gets a wrong value for **$r3**. A stall is generated untile the correct value, which is needed for the branch, is available.
- Instruction I10 (next after **bnez**) is fetched in cycle 11. When I9 is decoded, a branch instruction is detected and I10 is retired from the pipeline.
- As branches are predicted to be taken, instruction I1 is fetched in cycle 14, which is when the effective target branch address is known.

Second cycle estimation The cost of an iteration different from the last iteration is 13 cycles. The last iteration requires 13 cycles.

$$Cost = 13 \cdot n$$

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13
I1: lw $f0, 0($r1)	IF	ID	EX	M	WB								
I2: lw $f2, 0($r2)		IF	ID	EX	M	WB							
I3: sub.f $f4, $f0, $f2			IF	ID	–	EX	M	WB					
I4: mul.d $f4, $f4, $f4				IF	–	ID	EX	M	WB				
I5: add.d $f6, $f6, $f4						IF	ID	EX	M	WB			
I6: addi $r1, $r1, 4							IF	ID	EX	M	WB		
I7: addi $r2, $r2, 4								IF	ID	EX	M	WB	
I8: sub $r3, $r3, 4									IF	ID	EX	M	WB
I9: bnez $r3, loop										IF	ID	–	EX
10:											IF		
I1: lw $f0, 0($r1)													

Instr.	14	15	16	17	18	19	20	21
I1: lw $f0, 0($r1)								
I2: lw $f2, 0($r2)								
I3: sub.f $f4, $f0, $f2								
I4: mul.d $f4, $f4, $f4								
I5: add.d $f6, $f6, $f4								
I6: addi $r1, $r1, 4								
I7: addi $r2, $r2, 4								
I8: sub $r3, $r3, 4								
I9: bnez $r3, loop	M	WB						
10:								
I1: lw $f0, 0($r1)	IF	ID	EX	M	WB			

Table 9: Second time diagram for exercise 7.