universidade de aveiro
ua

Universidade de Aveiro

Mestrado Integrado em Engenharia de Computadores e Telemática

Arquitectura de Computadores Avançada

Lesson 2: DLX - Pipelining

Academic year 2015/2016 Nuno Lau/José Luis Azevedo

1. Consider a program that sums 8 integer values placed in memory starting at address **`vec`**.

   1.1. Write the program using DLX assembly. The sum result should be stored in register **`r1`** (a table with the DLX processor instruction set is available in the course web site). The values stored in memory should be set using the **`.word`** directive.

   1.2. Determine, **analytically**, the number of clock cycles the program takes to execute in a processor with a five-stage pipeline, supposing that no stalls will occur during the whole execution.

   1.3. Run your program using the **`WinDLX`** simulator with the *Enable Forwarding* option **disabled** (menu *Configure*). Examine, step by step, the evolution of the processor registers as well as the state of the pipeline. Determine the number of stall clock cycles.

   1.4. Analyse the temporal diagram of the pipeline state during an iteration of the main loop of your program. Justify each stall indicated by **`WinDLX`**, specifying the type of stall and, if appropriate, which processor register originated it.

   1.5. Through the *Statistics* window of **`WinDLX`**, take note of the total number of clock cycles it took to execute the program, as well as the total number of Data, Control and Structural Stalls.

   1.6. Determine, **analytically**, the total number of clock cycles the program takes to execute, without forwarding, and compare the results with those obtained in the simulator.

   1.7. Change the **`WinDLX`** configuration in order to enable the pipeline forwarding and rerun the program. Check which changes occurred in the number and type of stalls. Determine, analytically, the total number of clock cycles and compare with the results obtained in the simulator.

   1.8. Still using the pipeline forwarding configuration enabled, analyze the timing diagram of the pipeline for an iteration of the main loop. Identify all the forwarding situations that occurred, stating the source and destination pipeline stages of the forwarding action as well as the processor register(s) involved.

   1.9. Determine the total number of clock cycles it takes to execute the whole program (still using forwarding). With that:

   - calculate the speedup that is achieved, taking as reference the execution without forwarding;
   - calculate the program execution time, supposing that the clock frequency of the processor is 100MHz.

2. Consider the following program:

```
;Adapted from problem 3.1 in H&P's book to run on WinDLX
      .data
      .align 4
_a:   .word 1, 10, 100, 2, 4, 8, -1, -10

      .text
      .global main
main:
      lhi    r2, (_a >> 16)      ; r2 = (_a >> 16) << 16
      addui  r2, r2,(_a & 0xffff)
      addui  r3, r2, 32
loop:
      lw     r1, 0(r2)
      addi   r1, r1, 1
      sw     0(r2), r1
      addui  r2, r2, 4
      sub    r4, r3, r2
      bnez   r4, loop
      trap   0
```

2.1. Disable the forwarding feature and:

- run the whole program and take note of the number of clock cycles it takes to execute, as well as the number of stall clock cycles;
- justify the occurrence of stalls during an iteration of the main loop.

2.2. Enable the forwarding feature and:

- run the whole program and take note of the number of clock cycles it takes to execute, as well as the number of stall clock cycles;
- justify the stalls that occur during an iteration of the main loop;
- identify the forwarding situations and characterize them;
- determine the speedup obtained by using the forwarding technique.

2.3. Optimize the original program in order to reduce its execution time (using forwarding). Please note that you are only allowed to reorder instructions and change some instruction parameters - changes in the type and/or number of instructions, are not permitted.

- did you manage to eliminate all stalls caused by data dependencies?
- determine the speedup obtained, taking as reference the non-optimized version.

3. Consider the following program:

```
;Adapted from problem 3.2 in H&P's book for running on WinDLX
      .data
      .align 4
_a:   .double 1.0, 10.0, 100.0, 1000.0
_b:   .double 1.0, 2.0, 3.0, 4.0

      .text
      .global main
main:
      lhi    r2, (_a >> 16)
      addui  r2, r2, (_a & 0xffff)
      lhi    r3, (_b >> 16)
      addui  r3, r3, (_b & 0xffff)
      addui  r4, r2, 32
loop:
      ld     f0, 0(r2)
      ld     f4, 0(r3)
      multd  f0, f0, f4
      addd   f2, f0, f2
      addui  r2, r2, 8
      addui  r3, r3, 8
      sub    r5, r4, r2
      bnez   r5, loop
      trap   0
```

3.1. Disable the forwarding. Run the program and count the total number of cycles and stalls. Justify the stalls that occur during an iteration of the main loop.

3.2. Enable the forwarding. Run the program and count the total number of cycles and stalls. Justify the stalls that occur during an iteration of the main loop. Identify the forwarding situations and characterize them. Determine the speedup obtained by enabling the forwarding technique.

3.3. Optimize the program in order to reduce its execution time (using forwarding). Please note that you are only allowed to reorder instructions and change some instruction parameters - changes in the type and/or number of instructions, are not permitted.

- did you manage to eliminate all stalls caused by data dependencies?
- determine the speedup obtained, taking as reference the non-optimized version.

4. Consider the following sequence of instructions:

```
      .text
main: multf  f2,f4,f5
      addf   f2,f3,f4
      multf  f6,f6,f6
      addf   f1,f3,f5
      addf   f2,f3,f4
      trap   0
```

4.1. Execute the sequence using the **WinDLX** simulator. Justify each one of the observed stalls.