Lab 7

Goals

1. Performance benefits of data forwarding
2. Understanding the importance of pipeline scheduling
3. A simple but practical introduction to loop unrolling

Instructions

1. These exercises are to be done individually.
2. Create a directory called <rollno>-<labno>. Eg. **140050001-07**. Store all the relevant files in this directory.
    1. In the exercises, you will be asked to provide explanations to various questions and to attach screenshots. Note them down in a **single file "explanations.pdf". Provide the assembly code separately**. In the code you will be writing the modifications as comments and in the pdf file the justification of these changes.
    2. While submitting (on moodle), you have to create a tar.gz or zip of the entire directory in which all your relevant files reside.
3. Before leaving the lab, ensure that you have signed the attendance sheet.

Spillover from Lab 6:

- **Question 0a [1 marks]:** Write a program which causes one or more control stalls. Name the file as "**lab7-0a.s"**
- **Question 0b [1 marks]:** Write a program which causes stall in the ID stage of a branch instruction. Name the file as "**lab7-0b.s"**

You may reuse parts of code from the previous lab. Provide appropriate explanations and screenshots in **"explanations.pdf"**

Benefits of data forwarding in the DLX pipeline

- Convert the following C code into DLX code:

```
int a[] = {1, 2, 3, 4, 5, 6};
int b[] = {2, 3, 4, 5, 6, 7};
int c[] = {0, 0, 0, 0, 0, 0};
int d[] = {0, 0, 0, 0, 0, 0};
int n = 6;
int alpha = 10;

int f(int x){
      return alpha*x;
}

int main(){
      for (int i = 0; i < n; i++) {
            c[i] = a[i] * b[i];
            d[i] += f(c[i]);
      }
}
```

- **Question 1a [2 marks]:** Blindly translate the code. Run the code on WinDLX, with data forwarding enabled. Then subsequently run the same code with data forwarding disabled. What is the number of clock cycles taken for execution in each case. What is the speedup due to data forwarding? Name the code as "**lab7-1a.s**". Attach the screenshots of WinDLX GUI showing the number of cycles taken in both cases in **explanations.pdf**.

- **Question 1b [2 marks]:** If you blindly translate the C code to DLX assembly code, there must be two branch instructions per loop iteration. You can optimize this to have only one branch operation per loop iteration, if you assume n > 0 (which the compiler can actually check statically). Do this optimization. When forwarding is enabled, what is the speedup due to this optimization? Attach the screenshot in **explanations.pdf** showing number of execution cycles. Name the code as "**lab7-1b.s**".

Function Inlining

- **Question 2a [3 marks]:** Now, convert the above code to replace the function call with the function body(inline function). Assume data forwarding enabled. Compare the number of execution cycles with the number of cycles in question 1b. Attach the screenshot in **explanations.pdf** showing number of execution cycles. Name the code as "**lab7-2-inline.s**".

- From now onwards, for further modifications, we will always use the above code (data forwarding, function inlining and single-branch-per-iteration version of the code).

Code scheduling for performance improvement

- **Question 3a [4 marks]:** Schedule the code within the loop, such that the number of stall cycles is minimized. State in comments what changes you have made as compared to the previous code and write the justification behind the optimization in **explanations.pdf**. Also write the number of cycles of the unscheduled as well as the scheduled code. Also attach screenshot of the WinDLX GUI showing the performance improvement due to such scheduling. Name this code as "**lab7-3-sched.s**".

- **Question 3b [2 marks]:** Clearly explain in **explanations.pdf**, as to which are the stalls which you are ***not*** yet able to mask through scheduling.

Loop unrolling to reduce loop overhead

- Loop unrolling is a performance enhancement technique which trades off code size for code speed. That is, it achieves better execution speed at the cost of potentially larger code size. You can read briefly about loop unrolling in your textbook or online.

- You are now going to unroll the ***unscheduled* code** or **lab7-2-inline.s**. **Unroll it twice**. That is, two iterations of the earlier loop is now effectively achieved by one iteration of this loop. In your new loop, the loop variable will be incremented by 2 for each iteration. Name this new code "**lab7-4-unroll.s**".

- **Question 4 [4 marks]:** Run the code with and without loop unrolling. Attach the screenshot of WinDLX GUI showing number of cycles, and explain the performance improvement due to unrolling in **explanations.pdf**.

Code scheduling after unrolling

- A non-trivial benefit of loop unrolling is the large number of opportunities it gives for better code scheduling, to avoid stalls. To do such scheduling, you may have to do some register renaming to remove anti-dependencies.

  Register renaming is a form of pipelining that deals with data dependences between instructions by renaming their register operands.

  An anti-dependency, also known as write-after-read (WAR), occurs when an instruction requires a value that is later updated. In the following example, instruction 2 anti-depends on instruction 3 — the ordering of these instructions cannot be changed, nor can they be executed in parallel (possibly changing the instruction ordering), as this would affect the final value of A.

  1. B = 3
  2. A = B + 1
  3. B = 7

- **Question 5a [4 marks]:** Optimize the above code (**lab7-4-unroll.s)** with code scheduling along with the above stated optimizations and name this file "**lab7-5-unroll-sched.s**". Write in its comments the differences compared to the previous version and justify these modifications with brief explanation in the file **explanations.pdf**. Also attach the screenshot of WinDLX GUI showing the performance improvement in terms of number of cycles due to the scheduling and unrolling.

- **Question 5b [2 marks]:** Clearly show through comments in your code, as to which are the stalls which you are ***not*** yet able to mask through scheduling.

Final List of files to be attached:

1. explanations.pdf
2. lab7-0a.s
3. lab7-0b.s
4. lab7-1a.s
5. lab7-1b.s
6. lab7-2-inline.s
7. lab7-3-sched.s
8. lab7-4-unroll.s
9. lab7-5-unroll-sched.s