LAB. COMPUTER ARCHITECTURE: STUDENT GUIDE

Unrolling loops

The unrolling loops technique consists in reducing the number of iterations of a loop with the purpose of “saving instructions”, specifically those dedicated to the control of the loop itself, such as updating indexes and the jump itself. To do this, the computation that is performed in an iteration is repeated one or more times within the loop, multiplying the number of necessary iterations. For example, if an element of a series of vectors is processed in each iteration, two or more elements are processed per iteration, decreasing the number of iterations needed to process the entire list.

In this way it is possible to reduce the number of cycles consumed during execution. It is common that with unrolling it is easier to reorder instructions to minimize stalls.

Modify the hardware in DLXV3.1 to assign a total latency of 2 cycles to the sum and multiplication and check that there is only one functional unit of each type.

Let's see this through an example. You can adapt it or/and start from the following code: study it, write it down and write down the stalls etc. in a comment. Then simulate it step by step. Check the cycles.

```
   .data 100
cte:  .double  3
n: .word 12
a: .double  1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12
b: .double  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

   .text 0x1000
ini:
   lw r8,n(r0)
   ld f0,cte(r0)
   xor   r1,r1,r1
loop:
   ld f2,a(r1)
   addd  f4,f0,f2
   sd b(r1),f4
   addi  r1,r1,#8
   subi  r8,r8,#1
   bnez  r8,loop
   nop
trap #6
```

There are 3 stalls (indicate where next to them), and the delay slot has not been used. And 124 cycles are used.

Reordering and taking advantage of the delay slot 76 cycles are obtained, achieving an acceleration of A = 124/76 = 1.63

```
loop1:
   ld f2,a(r1)
   subi  r8,r8,#1
   addd  f4,f0,f2
   addi  r1,r1,#8
   bnez  r8,loop 1
   sd    b-8(r1),f4
trap #6
```

This loop is executed 12 times processing one component per lap.

We can process several components per turn and decrease the number of turns accordingly. A **factor 2** unroll processes two components at each turn, knowing that we have to use **a single index adjustment instruction** to save cycles (otherwise this optimization technique is not justified).

<u>One recommendation</u>: it is easier to make a non-optimized version first by grouping loading instructions, process instructions and index and counters, and then identifying stalls and reordering. Both steps are shown below:

```
ini2:
     ld f0,cte(r0)
     xor    r1,r1,r1
     lw r8,n(r0)
loop2:
      ld     f2,a(r1)
      ld     f6,a+8(r1)
      addd   f4,f0,f2
      addd   f8,f0,f6
      sd     b(r1),f4
      sd     b+8(r1),f8
      addi   r1,r1,#16
      subi   r8,r8,#2
      bnez   r8,loop2
      nop
trap  #6
```

It is possible to execute without stalls in 58 cycles by reordering, with an Acceleration A = 2.14 and with respect to the optimized one without unrolling 1.3

```
loop3:
      ld     f2,a(r1)
      ld     f6,a+8(r1)
      addd   f4,f0,f2
      subi   r8,r8,#2
      addd   f8,f0,f6
      sd     b(r1),f4
      sd     b+8(r1),f8
      bnez   r8,loop3
      addi   r1,r1,#16
trap  #6
```

It is very important to realize that the savings are in that less jumps are executed and less index adjustment instructions: this is organized by the compiler, computer engineers do these things.

In a similar way, we can unroll with **factor 4**

```
ini4:
      ld f0,cte(r0)
      xor    r1,r1,r1
      lw r8,n(r0)
loop4:
      ld     f2,a(r1)
      ld     f6,a+8(r1)
      ld     f10,a+16(r1)
      ld     f14,a+24(r1)
      addd   f4,f0,f2
      addd   f8,f0,f6
      addd   f12,f0,f10
      addd   f16,f0,f14
      sd     b(r1),f4
      sd     b+8(r1),f8
      sd     b+16(r1),f12
      sd     b+24(r1),f16
      addi   r1,r1,#32
      subi   r8,r8,#4
      bnez   r8,loop4
      nop
      trap   #6
```

It is possible to execute without stopping in 49 cycles by rearranging it, with an Acceleration A = _____ and with respect to the optimized one without unrolling A = ______

GOOD LUCK!

Exercises

All with operand overtaking and delayed jump options.

1. For the program of the Scalar Product exercise of two vectors in floating point and for N=32 elements, make a version of unrolling of factor 2 and optimize it to minimize stalls. Next, unroll out with factor 4 and optimize it. Calculate the Acceleration (Gain).

2. Use the program C(i) = a i)·cte+b(i) designed in the previous session and do the same.

3. Change multiplication latency to 4 cycles and re-order trying to eliminate stalls completely.