

STUDENTS IDENTIFICATION:

| | |
|---------|-----------------|
| Number: | Name: |
| 109474 | Daniel Borges |
| 110181 | Duarte Cruz |
| 110728 | João Tiago Gama |

2.1 Simple execution, without data forwarding techniques

e)

| | |
|--------------|------|
| Clock cycles | 308 |
| Instructions | 170 |
| Average CPI | 1.81 |

| | |
|-------------------------|-----|
| Stalls: - Data | 102 |
| - Structural | 0 |
| Flushes: - Branch Taken | 15 |

- f) The branch prediction policy adopted by the simulator is a static “not taken” prediction. This means that all branch instructions are assumed not to be taken during the fetch stage. This behavior is evidenced by the fact that the program continues fetching instructions sequentially after a branch, and only if the branch is actually taken does the pipeline experience a stall due to the need to flush and redirect the instruction flow.

2.2 Application of data forwarding techniques

c)

| | |
|--------------|------|
| Clock cycles | 222 |
| Instructions | 170 |
| Average CPI | 1.31 |

| | |
|-------------------------|----|
| Stalls: - Data | 16 |
| - Structural | 0 |
| Flushes: - Branch Taken | 15 |

- d) The obtained speedup can be computed as:

$$\text{Speedup} = \frac{T_{\text{old}}}{T_{\text{new}}} = \frac{\left(\frac{1}{f_{\text{old}}}\right) \times \#Cycles_{\text{old}}}{\left(\frac{1}{f_{\text{new}}}\right) \times \#Cycles_{\text{new}}} = \frac{f_{\text{new}}}{f_{\text{old}}} \times \frac{\#Cycles_{\text{old}}}{\#Cycles_{\text{new}}}$$

Given that:

$$\#Cycles_{\text{old}} = 308, \#Cycles_{\text{new}} = 222$$

we obtain:

$$\frac{\#Cycles_{\text{old}}}{\#Cycles_{\text{new}}} = 1.38739$$

Since both processors operate at the same clock frequency ($f_{\text{new}} = f_{\text{old}}$), then:

$$\text{Speedup} = 1.38739$$

2.3 Source code optimization: minimization of data and structural hazards

- a) Attach a copy of the new assembly program.

| | |
|--------------|------|
| Clock cycles | 206 |
| Instructions | 170 |
| Average CPI | 1.21 |

| | | |
|----------|----------------|----|
| Stalls: | - Data | 0 |
| | - Structural | 0 |
| Flushes: | - Branch Taken | 15 |

- d) The obtained speedup can be calculated as follows:

$$\text{Speedup} = \frac{T_{\text{old}}}{T_{\text{new}}} = \frac{\left(\frac{1}{f_{\text{old}}}\right) \times \#Cycles_{\text{old}}}{\left(\frac{1}{f_{\text{new}}}\right) \times \#Cycles_{\text{new}}} = \frac{f_{\text{new}}}{f_{\text{old}}} \times \frac{\#Cycles_{\text{old}}}{\#Cycles_{\text{new}}}$$

Given that:

$$\#Cycles_{\text{old}} = 308, \#Cycles_{\text{new}} = 206$$

we have:

$$\frac{\#Cycles_{\text{old}}}{\#Cycles_{\text{new}}} = 1.49515$$

Since both processors operate at the same clock frequency ($f_{\text{new}} = f_{\text{old}}$), then:

$$\text{Speedup} = 1.49515$$

2.4 Source code optimization: loop unrolling

a) Attach a copy of the new assembly program.

c)

| | |
|--------------|------|
| Clock cycles | 122 |
| Instructions | 110 |
| Average CPI | 1.11 |

| | | |
|----------|----------------|---|
| Stalls: | - Data | 0 |
| | - Structural | 0 |
| Flushes: | - Branch Taken | 3 |

d) The obtained speedup can be expressed as:

$$\text{Speedup} = \frac{T_{\text{old}}}{T_{\text{new}}} = \frac{\left(\frac{1}{f_{\text{old}}}\right) \times \# \text{Cycles}_{\text{old}}}{\left(\frac{1}{f_{\text{new}}}\right) \times \# \text{Cycles}_{\text{new}}} = \frac{f_{\text{new}}}{f_{\text{old}}} \times \frac{\# \text{Cycles}_{\text{old}}}{\# \text{Cycles}_{\text{new}}}$$

Given that:

$$\# \text{Cycles}_{\text{old}} = 308, \# \text{Cycles}_{\text{new}} = 122$$

then:

$$\frac{\# \text{Cycles}_{\text{old}}}{\# \text{Cycles}_{\text{new}}} = 2.52459$$

Since both processors operate at the same clock frequency ($f_{\text{new}} = f_{\text{old}}$), then:

$$\text{Speedup} = 2.52459$$

Attachment 1: Assembly Code 2.3 a)

```
.data
A: .word 1,2,3,4,5,6,7,8,9
    .word 10,11,12,13,14,15,16
B: .word 11,22,33,44,55,66,77
    .word 88,99,111,122,133
    .word 144,155,166
C: .word 0,0,0,0,0,0,0,0,0
    .word 0,0,0,0,0,0,0

.text
main:
    la s3, A          # base of A
    la s4, B          # base of B
    la s5, C          # base of C
    addi s1, zero, 0   # i = 0
    addi s2, zero, 16  # value of N

loop:
    lw t1, 0(s3)
    lw t2, 0(s4)
    addi s1, s1, 1
    addi s3, s3, 4
    mul t3, t2, t1
    addi s4, s4, 4
    add t3, t3, t1
    sw t3, 0(s5)
    addi s5, s5, 4
    bne s1, s2, loop

end:
    addi a7, zero, 10
    ecall           # Exit (syscall)
```

Attachment 2: Assembly Code 2.4a)

```
.data
A: .word 1,2,3,4,5,6,7,8,9
    .word 10,11,12,13,14,15,16
B: .word 11,22,33,44,55,66,77
    .word 88,99,111,122,133
    .word 144,155,166
C: .word 0,0,0,0,0,0,0,0,0
    .word 0,0,0,0,0,0,0

.text
main:
    la s3, A          # base of A
    la s4, B          # base of B
    la s5, C          # base of C
    addi s1, zero, 0   # i = 0
    addi s2, zero, 16  # value of N

loop:
    lw t2, 0(s4)
    lw t1, 0(s3)
    addi t3, t2, 1
    mul t3, t3, t1
    sw t3, 0(s5)

    lw t2, 4(s4)
    lw t1, 4(s3)
    addi t3, t2, 1
    mul t3, t3, t1
    sw t3, 4(s5)

    lw t2, 8(s4)
    lw t1, 8(s3)
    addi t3, t2, 1
    mul t3, t3, t1
    sw t3, 8(s5)

    lw t2, 12(s4)
    lw t1, 12(s3)
    addi t3, t2, 1
    mul t3, t3, t1
    sw t3, 12(s5)

    addi s1, s1, 4
    addi s3, s3, 16
    addi s4, s4, 16
    addi s5, s5, 16
    bne s1, s2, loop

end:
    addi a7, zero, 10
    ecall             # Exit (syscall)
```

Table 1: Pipeline time diagram, without data forwarding techniques.

Table 2: Pipeline time diagram, with data forwarding techniques.

Table 3: Pipeline time diagram, with minimization techniques to reduce the data and structural hazards.

Table 4: Pipeline time diagram: usage of loop unrolling minimization techniques to reduce the control hazards.