Project Report

Computer Architecture (UE20EC353)

Gautham B(PES1UG20EC044), Dheemanth R Joshi(PES1UG20EC059)

1 Aim

In this report we will analyse the performance of a *pipelined* processor. In particular we will analyse how the compiler optimization technique called *Loop-unrolling* can be used to improve the performance of the pipeline at the cost of using more registers.

2 Software Used

To illustrate the performance improvements, we make use of the *RIPES-RISC V* simulator.

3 Methodology

To showcase the the techinque of loop unrolling, we will take up the program of summing up an array of integers. The performance improvement can be analysed by:

1. Runnning the non-unrolled code.
2. Runnning the unrolled loop code.
3. Compare Clock cycles, CPI, Stalls in each case.

We will use a loop unrolling scheme which unrolls four elements at a time and we will use an array which in 100 elements wide.

4 Code

No unrolling:

```
.data
    array: .word 1,2,3...100
    array_size: .word 100

.text
    .globl main
    .align 2

main:
    # Load array size
    lw t0, array_size
```

```
    # Load array pointer
    la t1, array

    # Initialize sum
    li t2, 0

    # Loop to sum array elements
    loop:
        lw t3, 0(t1)
        add t2, t2, t3
        addi t1, t1, 4
        addi t0, t0, -1
        bnez t0, loop

    # End of program
    nop
```

Unrolled code:

```
.data
    array: .word 1,2,3...100
    array_size: .word 100

.text
    .globl main
    .align 2

main:
    # Load array size
    lw t0, array_size

    # Load array pointer
    la t1, array

    # Initialize sum
    li t2, 0
    li a0,4
    # Loop unrolled to sum array elements
```

```
loop_unrolled:
    lw t3, 0(t1)
    lw t4, 4(t1)
    lw t5, 8(t1)
    lw t6, 12(t1)
    add t2, t2, t3
    add t2, t2, t4
    add t2, t2, t5
    add t2, t2, t6
    addi t1, t1, 16
    addi t0, t0, -4
    ble t0,a0,loop
    j loop_unrolled
# Loop to sum remaining array elements
loop:
    lw t3, 0(t1)
    add t2, t2, t3
    addi t1, t1, 4
    addi t0, t0, -1
    bnez t0, loop

# End of program
nop
```

5 Results and Conclusions

5.1 Program performance

We show the following results for both the codes:

1. # Clock cycles taken
2. Clocks per instruction (CPI)
3. Instructions per clock (IPC)

No unrolling:

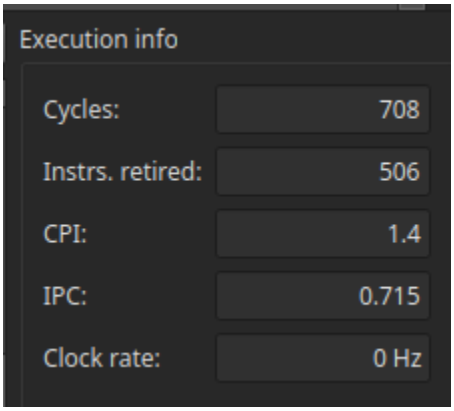


Figure 1: Non-unrolled loop

Unrolled loop:

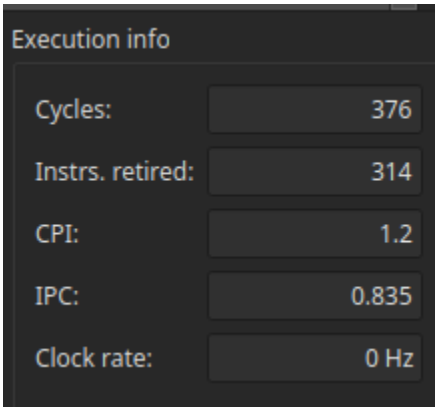


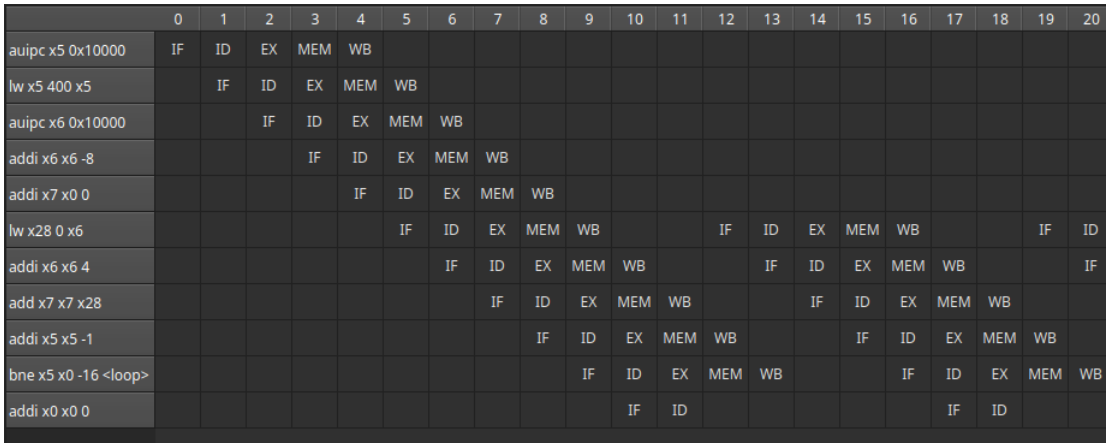
Figure 2: Unrolled loop

A clear performance improvement is observable in the unrolled case which can be explained by observing the pipeline diagrams in Section. 5.2.

5.2 Pipeline performance

Here, we will look at the pipeline performance for both cases.

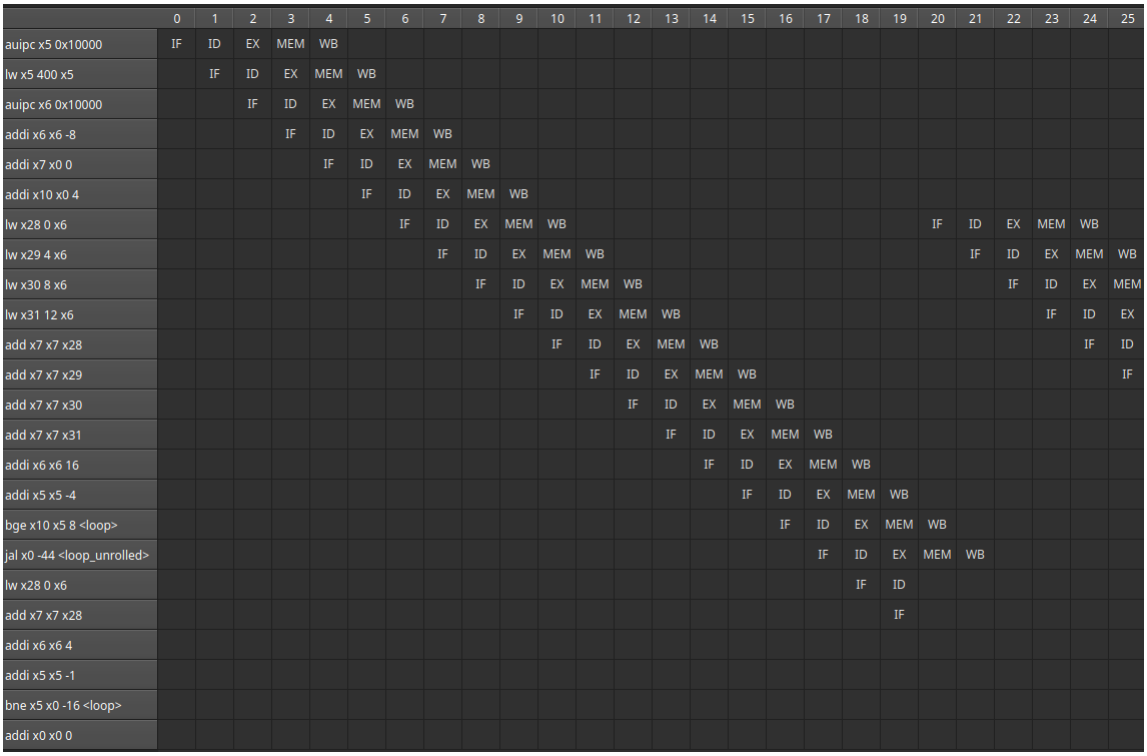
No unrolling:



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
auipc x5 0x10000	IF	ID	EX	MEM	WB																
lw x5 400 x5		IF	ID	EX	MEM	WB															
auipc x6 0x10000			IF	ID	EX	MEM	WB														
addi x6 x6 -8				IF	ID	EX	MEM	WB													
addi x7 x0 0					IF	ID	EX	MEM	WB												
lw x28 0 x6						IF	ID	EX	MEM	WB			IF	ID	EX	MEM	WB			IF	ID
addi x6 x6 4							IF	ID	EX	MEM	WB			IF	ID	EX	MEM	WB			IF
add x7 x7 x28								IF	ID	EX	MEM	WB			IF	ID	EX	MEM	WB		
addi x5 x5 -1									IF	ID	EX	MEM	WB			IF	ID	EX	MEM	WB	
bne x5 x0 -16 <loop>										IF	ID	EX	MEM	WB			IF	ID	EX	MEM	WB
addi x0 x0 0											IF	ID						IF	ID		

Figure 3: Non unrolled pipeline

Unrolled loop:



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
auipc x5 0x10000	IF	ID	EX	MEM	WB																					
lw x5 400 x5		IF	ID	EX	MEM	WB																				
auipc x6 0x10000			IF	ID	EX	MEM	WB																			
addi x6 x6 -8				IF	ID	EX	MEM	WB																		
addi x7 x0 0					IF	ID	EX	MEM	WB																	
addi x10 x0 4						IF	ID	EX	MEM	WB																
lw x28 0 x6							IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB	
lw x29 4 x6								IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB
lw x30 8 x6									IF	ID	EX	MEM	WB										IF	ID	EX	MEM
lw x31 12 x6										IF	ID	EX	MEM	WB										IF	ID	EX
add x7 x7 x28											IF	ID	EX	MEM	WB										IF	ID
add x7 x7 x29												IF	ID	EX	MEM	WB										IF
add x7 x7 x30													IF	ID	EX	MEM	WB									
add x7 x7 x31														IF	ID	EX	MEM	WB								
addi x6 x6 16															IF	ID	EX	MEM	WB							
addi x5 x5 -4																IF	ID	EX	MEM	WB						
bge x10 x5 8 <loop>																	IF	ID	EX	MEM	WB					
jal x0 -44 <loop_unrolled>																		IF	ID	EX	MEM	WB				
lw x28 0 x6																			IF	ID						
add x7 x7 x28																				IF						
addi x6 x6 4																										
addi x5 x5 -1																										
bne x5 x0 -16 <loop>																										
addi x0 x0 0																										

Figure 4: Unrolled pipeline

The unrolling in Fig. 4 is done *four-fold*, i.e we perform the computation of four elements in each iteration. Due do this we essentially performed operations for four elements before encoutering a branch overhead whereas in the case of Fig. 3 we perform computation of only one element before encountering a branch overhead. However the unrolling requires more registers which limits the number of times a given loop can be unrolled.