COL216 Assignment 2

Dhruv Gupta 2021CS50125
Niranjan Sarode 2021CS50612

1 Comparative Analysis of Pipeline Processors

Comparative analysis of the 4 pipelines which we had to construct 5stage, 5stage_bypassing , 79stage , 79stage_bypassing by using 5 test cases and 5 small test cases which are:

- **Small test Case 1(STC1):**

```
lw $10, 20($1)
sub $11, $2, $3
add $12, $3, $4
lw $13, 24($1)
add $14, $5, $6
```

- **Small test Case 2(STC2):**

```
sub $2, $1, $3
add $14, $2, $2
sw $15, 100($2)
```

- **Small test Case 3(STC3):**

```
add $1, $1, $2
add $1, $1, $3
add $1, $1, $4
```

- **Small test Case 4(STC4):**

```
lw $2, 20($1)
add $9, $4, $2
slt $1, $6, $7
```

- **Test Case 1(TC1):**

```
addi $1, $0, 2
addi $2, $0, 1
sw $1 , 1023($2)
add $1,$1,$1
sub $2 , $1, $2
lw $3, 1021($2)
```

- **Test Case 2(TC2):**

```
addi $1, $0, 2
addi $2, $0, 1
beq $1 , $2 , one
sw $1 , 1023($2)
addi $2 , $2 , 1
bne $1 , $2 , one
beq $1 , $2 , two
one: addi $2 , $2 , -1
two: lw $3 , 1022($2)
```

- **Test Case 3(TC3):**

```
addi $1, $0, 2
addi $2, $0, 1
sw $1 , 1002($1)
sw $2 , 1006($1)
add $3, $2, $1
sub $4, $2, $1
sw $3 , 1011($2)
sw $4 , 1015($2)
lw $5 , 1002($1)
lw $6 , 1006($1)
lw $7 , 1010($1)
lw $8 ,  1014($1)
add $7, $7, $8
add $6, $6, $7
add $5, $6, $5
```

- **Test Case 4(TC4):**

```
addi $t1, $0, 200
addi $t8, $0, 42
sw $t8, 8($t1)
addi $t3, $0, 3
addi $t4, $0, 12
addi $t7, $0, 100
sw $t7, 4($t1)
lw $t0, 4($t1)
add $t2, $t3, $t4
mul $t5, $t0, $t2
sw $t5, 4($t1)
lw $t6, 8($t1)
add $s0, $t6, $t5
```

- **Test Case 5(TC5):**

```
main:
  addi  $t0, $0, 10
  add   $t1, $0, $0
  addi  $s1, $zero, 320
loop:
  beq   $t1, $t0, exit
  sw    $s0, 12($s1)
  addi  $s0, $s0, 100
  addi  $t1, $t1, 1
  j   loop
```

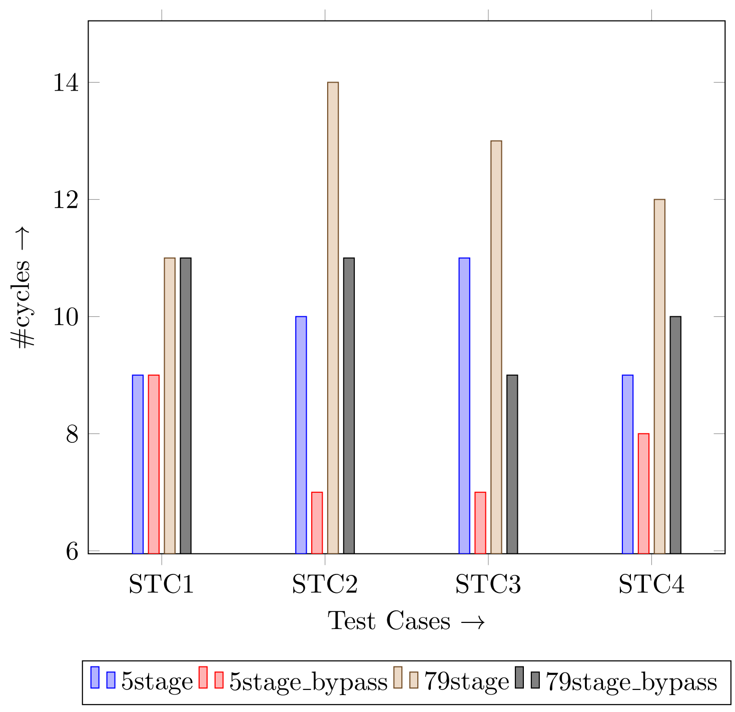
1.1 Barplots

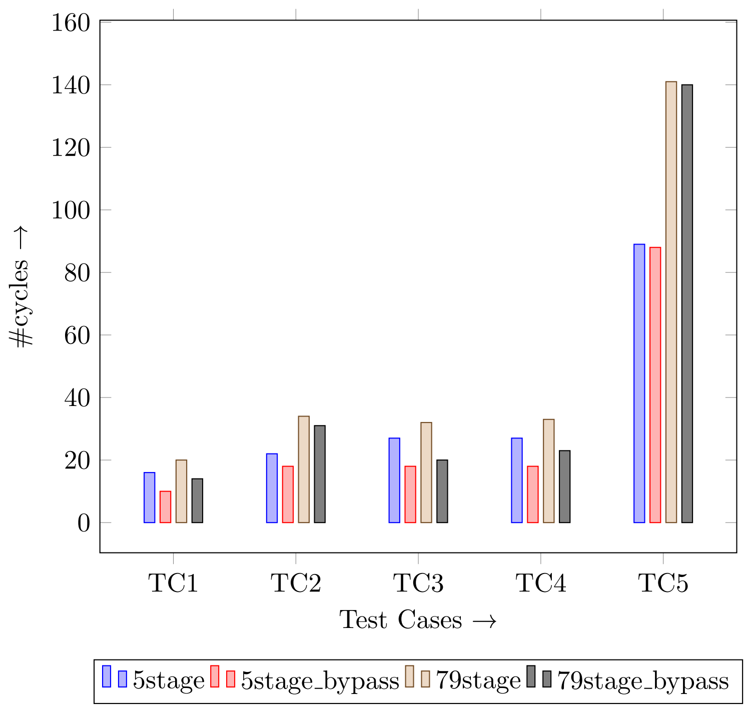
Test Case	Pipeline	5stage	5stage_bypass	79stage	79stage_bypass
STC1		9	9	11	11
STC2		10	7	14	11
STC3		11	7	13	9
STC4		9	8	12	10

Table 1: Number of cycles for the Small Test Cases

Test Case	Pipeline	5stage	5stage_bypass	79stage	79stage_bypass
TC1		16	10	20	14
TC2		22	18	34	31
TC3		27	20	28	22
TC4		27	18	32	20
TC5		89	88	141	140

Table 2: Number of cycles for the Test Cases





1.2 Comparision and observations

- **5stage vs 79stage:**

  As we can see from the bar plots that the 5 stage pipeline has a lower number of cycles than the 79 stage pipeline. Thats because each stage has more cycles and more potential for stalls which signifincantly increases the number of cycles in 79stage but still this can be desirable because **changing from 5 stage to 79 stage pipeline will have more thoughput at the same time more latency**. The tradeoff between thoughtput and latency will decide which is better.

- **Importance of bypassing:**

  We can see that in both cases 5stage and 79stage we can significantly reduce cycle count by bypassing which leads to overall lower CPI and better processor. Bypassing is a important tool to decrease the number of stall cycles in structural and data hazards.

2 Design Decisions

2.1 Design Decisions for $5stage$

- We used the given github for the basic structure of the code including error handling and file reading , parsing address and RegisterMap.
- We created 5 methods in the struct MIPS_Architecture for $ifexecute$, *idexecute* ,*exexecute*, *memexecute* and *wbexecute*.
- We used 2 global variables *add_stall* and *load_stall* that denotes stall cycles needed before EX and MEM step respretively.
- We took care of control hazards by assuming a no operation step by adding 2 stall cycles at end of every Branch operation *beq* and *bne*. After one stall cycle the variables are updates as:

$$add\_stall = max(0, add\_stall - 1)$$

$$laod\_stall = max(0, load\_stall - 1)$$

- For data hazards with *add*, *sub*, *mul* and *slt* we use 2 stall cycles of *add_stall* type if destination of current opeation is needed in previous operation and 1 stall cycles if needed in previous to previous instruction.
- For data hazards with *lw* we use 2 stall cycles of *load_stall* type if destination of current opeation is needed in previous operation and 1 stall cycles if needed in previous to previous instruction. If a subsequent instruction requires the destination register of *lw* we used 2 stall cycles of *load_stall* type and 1 if next to next instruction.
- For data hazards with *sw* we use 2 stall cycles of *load_stall* type if destination or required register of current opeation is needed in previous operation and 1 stall cycles if needed in previous to previous instruction

2.2 Design Decisions for $5stage\_bypass$

We reused code for $5stage$ for most part but made these notable changes

- We eliminated the structral hazards by bypassing the value confliting at the EX stage before further processing.
- We reduced the number of stall cycles by 1 for all the data hazards by bypassing the value at earlier stage.

2.3 Design Decisions for $79stage$

We reused code for $5stage$ for most part but made these notable changes

- We took the 9 stages as $fetch$, $fetch2$, *decode*, $decode2$, *regread*, *ALU* ,*data_memory*, $data\_memory2$ and *regwrite*. With *data_memory*, $data\_memory2$ used only for load and store operations.
- We distributed the IF work in $fetch$ and $fetch2$, ID work in *decode*,$decode2$ and *regread* and EX work in *ALU* and MEM work in *data_memory* and $data\_memory2$ and WB work in *regwrite*.
- We changed the data hazard in general case from 2 in previous step and 1 in previous to previous to 3 in previous 2 in previous previous and 1 in previous previous previous if data dependency found.
- Fow load and load and store and store cases we used similar 3 ,2 and 1 stalls as in previous step except reducing stalls by 1 if offset was different in some cases
- Data depencency in beq/bne related data hazards is handled similarly.
- For control hazards we introduced 5 stalls always after branch instruction.

2.4 Design Decisions for $79stage\_bypass$

We reused most of the code from $79stage$ but made these notable changes

- We eliminated the structral hazards by bypassing the value confliting at the ALU stage before further processing.
- We reduced the data stalls in most cases to 0 or 1 by bypassing the values from ALU to $decode2$.

3 Branch Prediction

S.No.	Branch Prediction Strategy	% Accuracy
1.	Using 2 bit Saturation counter	86.31%
2.	Using BHR with n=2	71.53%
3.	Using mix of these 2 strategies	79.01%

Table 3: Branch Prediction Strategies

3.1 using 2 bit saturation counter

For this we associate a FSM along with a current state for every branch instruction and update the FSM state based on the outcome of the branch. We use a 2 bit counter to represent the state of the FSM.

3.2 using BHR with last 2 bits

For this we use a BHR of size 2 and use the last 2 bits of the address of the branch instruction to index into the BHR. We use the value at the index to predict the outcome of the branch.

3.3 A way of combining these strategies

We use a mix of these 2 strategies. We use the 2 bit saturation counter to predict the outcome of the branch and if the prediction is wrong we use the BHR to predict the outcome of the branch.

4 Work Split

S.No.	Team Member	% Work Split
1.	Dhruv Gupta	50%
2.	Niranjan Sarode	50%

Table 4: Work-Split