In The Name Of God

Project Title : MIPS Pipeline Processor with Java

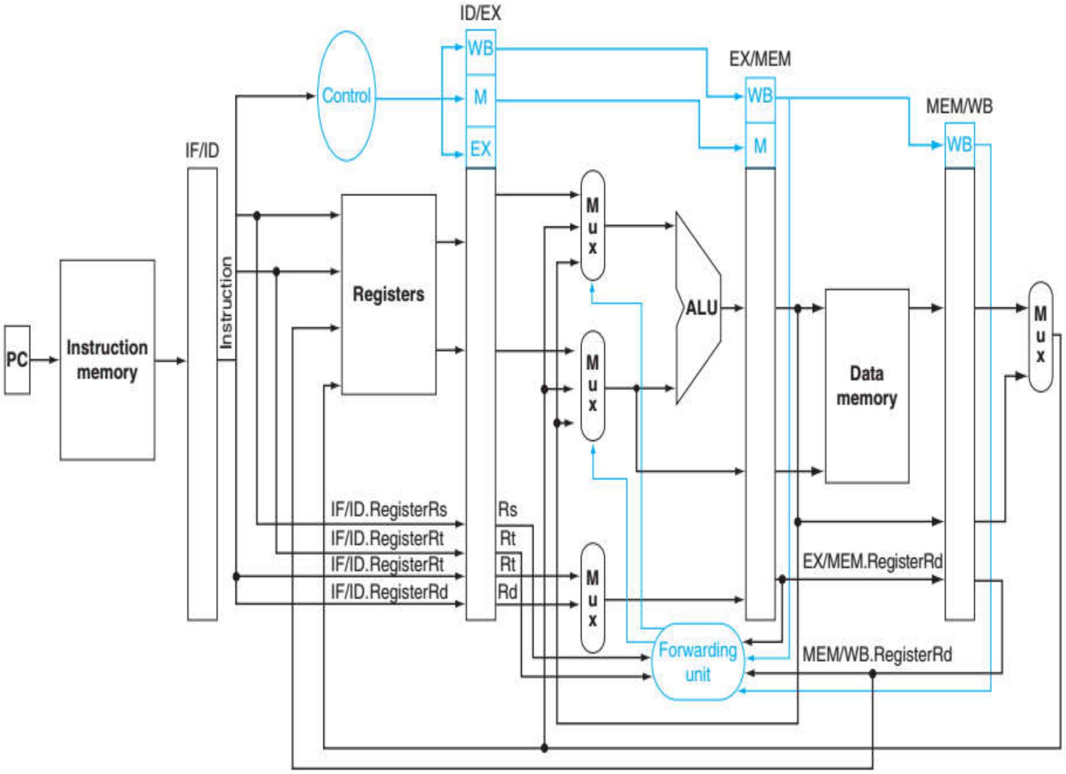
Professor : Dr . Mousavi

July 5th 2018 - 4th Semester - Shiraz University

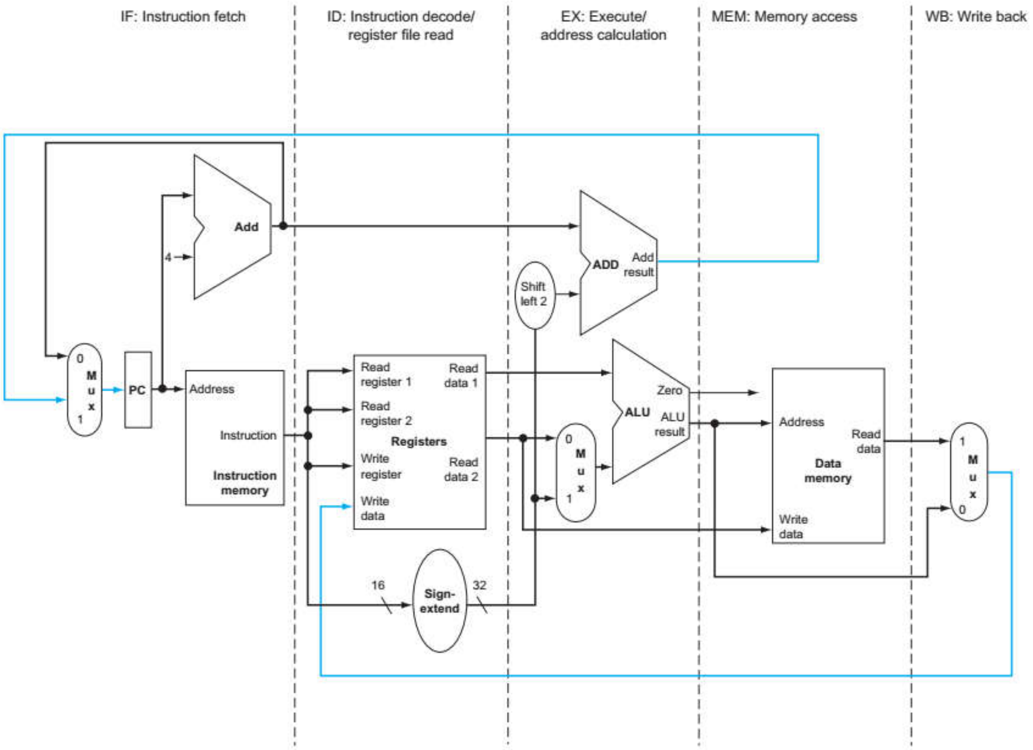
This project was started on July 1st 2018 and ended on July 5th 2018 by 8 hours/Day coding .

The uploaded file is consisted of multiple backups which were taken at each stage that the program was developed . each backup directory has it's own RegisterFile , DataFile and TestCase which is suitable for a special aspect added in the following stage of developing.

The total scheme of project is based on the picture below(figure 1.) :



The start of this project was with this scheme(figure 2) :



I started from figure 2 , step by step , and finally got to figure 1 .

The last update has this abilities :

1. It supports all " R_Type " instructions such as : add , sub , and , or , slt , xor , nor
2. It supports all I_Type instructions such as : lw , sw , beq
3. It also supports Jump instruction .
4. It can detect hazards and solve it by forwarding .

We can compile and run pipeline instructions by two ways :

1. Left to right
2. Right to left

If we start the pipeline from left to right , we shall execute the stages as below :

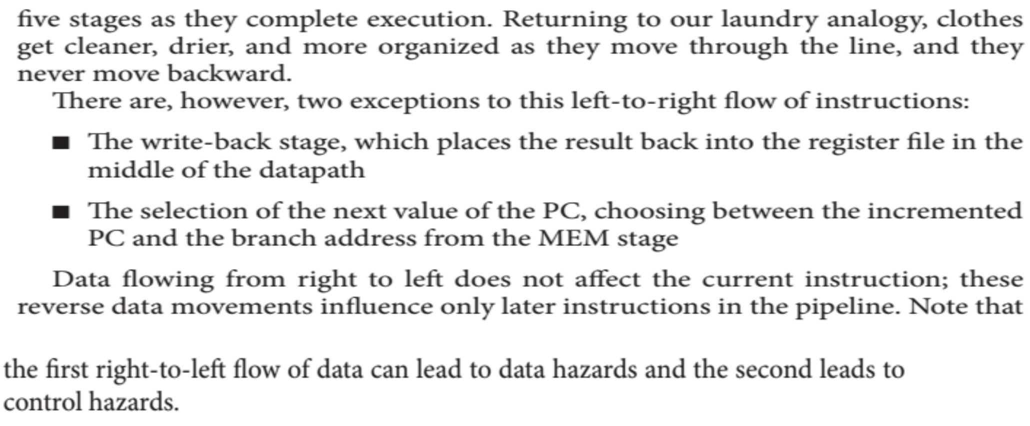
Fetch -> decode -> execute -> memory -> write back

But if we start the pipeline from right to left , we shall execute the stages as below :

Write back -> memory -> execute -> decode -> fetch

The difference is that in the first method we may only face Control Hazards but in the second method we will only face Data Hazards due to the reference :

“



five stages as they complete execution. Returning to our laundry analogy, clothes get cleaner, drier, and more organized as they move through the line, and they never move backward.

There are, however, two exceptions to this left-to-right flow of instructions:

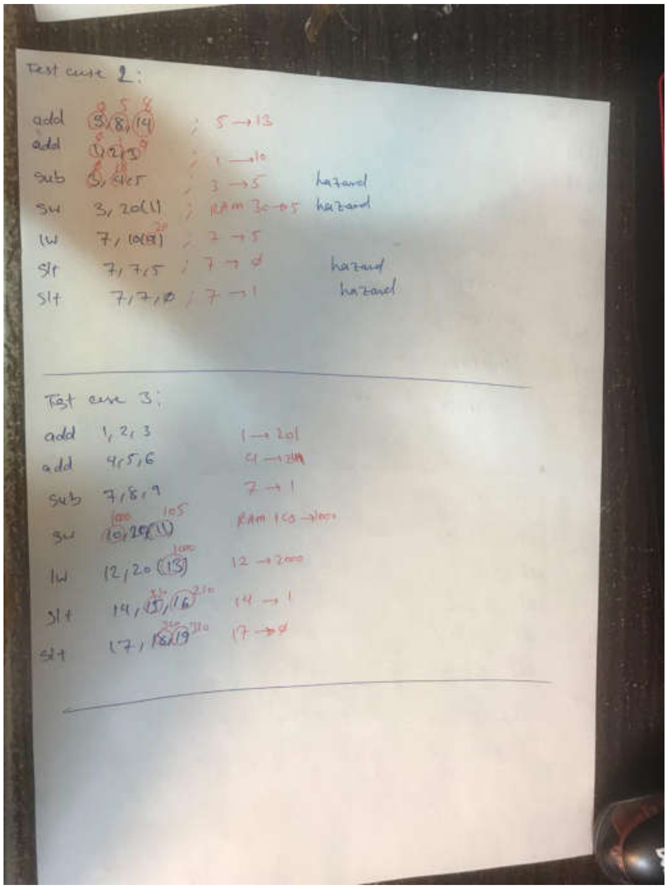
- The write-back stage, which places the result back into the register file in the middle of the datapath
- The selection of the next value of the PC, choosing between the incremented PC and the branch address from the MEM stage

Data flowing from right to left does not affect the current instruction; these reverse data movements influence only later instructions in the pipeline. Note that

the first right-to-left flow of data can lead to data hazards and the second leads to control hazards.

The method used in this project is right to left method which fixes the need of stalling.

My Notes While Coding :

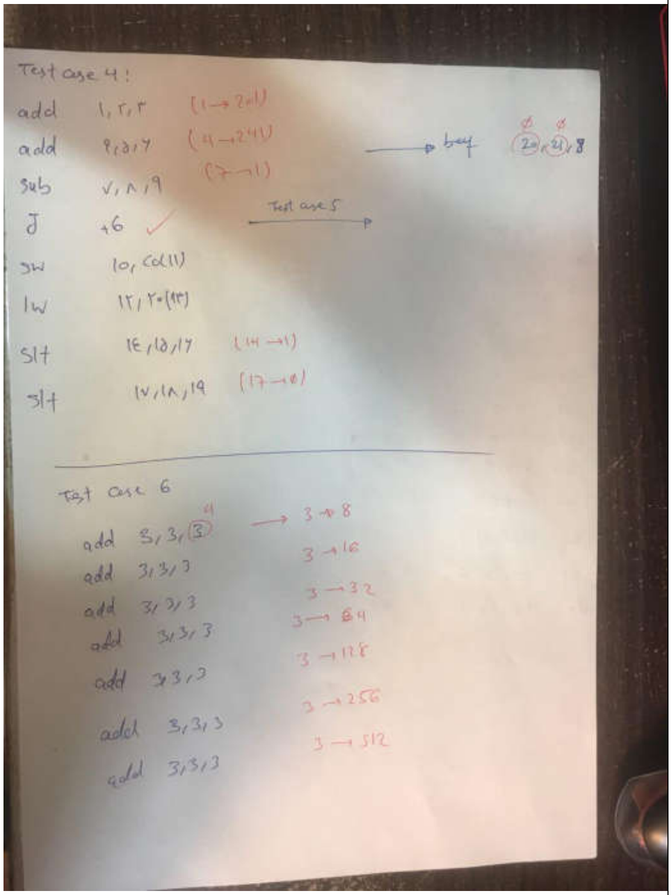


Test case 2:

add 5, 8, 14 ; 5 → 13
add 1, 2, 3 ; 1 → 10
sub 3, 5, 5 ; 3 → 5 hazard
sw 3, 20(1) ; RAM 30 → 5 hazard
lw 7, 10(8) ; 7 → 5
slt 7, 7, 5 ; 7 → 0 hazard
slt 7, 7, 0 ; 7 → 1 hazard

Test case 3:

add 1, 2, 3 1 → 201
add 4, 5, 6 4 → 244
sub 7, 8, 9 7 → 1
sw 10, 20(11) RAM 105 → 100
lw 12, 20(13) 12 → 2000
slt 14, 15, 16 14 → 1
slt 17, 18, 19 17 → 0

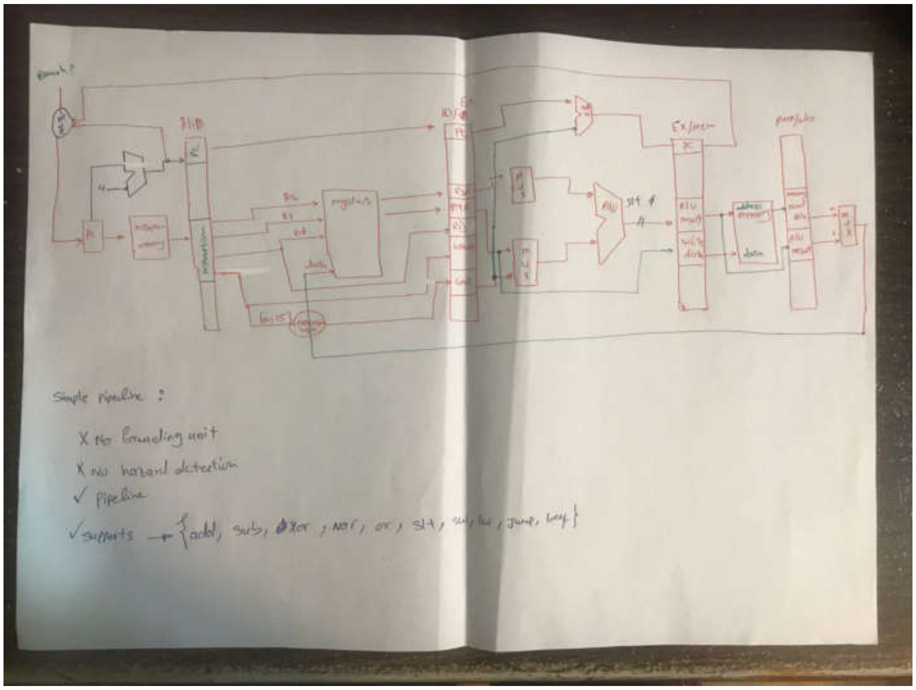


Test case 4 :

add ۱, ۲, ۳ (1 → 201)
add ۴, ۵, ۶ (4 → 241) → beq 20, 21, 8
sub ۷, ۸, ۹ (7 → 1)
J +6 ✓ Test case 5 →
sw 10, ۲۰(11)
lw ۱۲, ۲۰(۱۳)
slt ۱۴, ۱۵, ۱۶ (14 → 1)
slt ۱۷, ۱۸, ۱۹ (17 → 0)

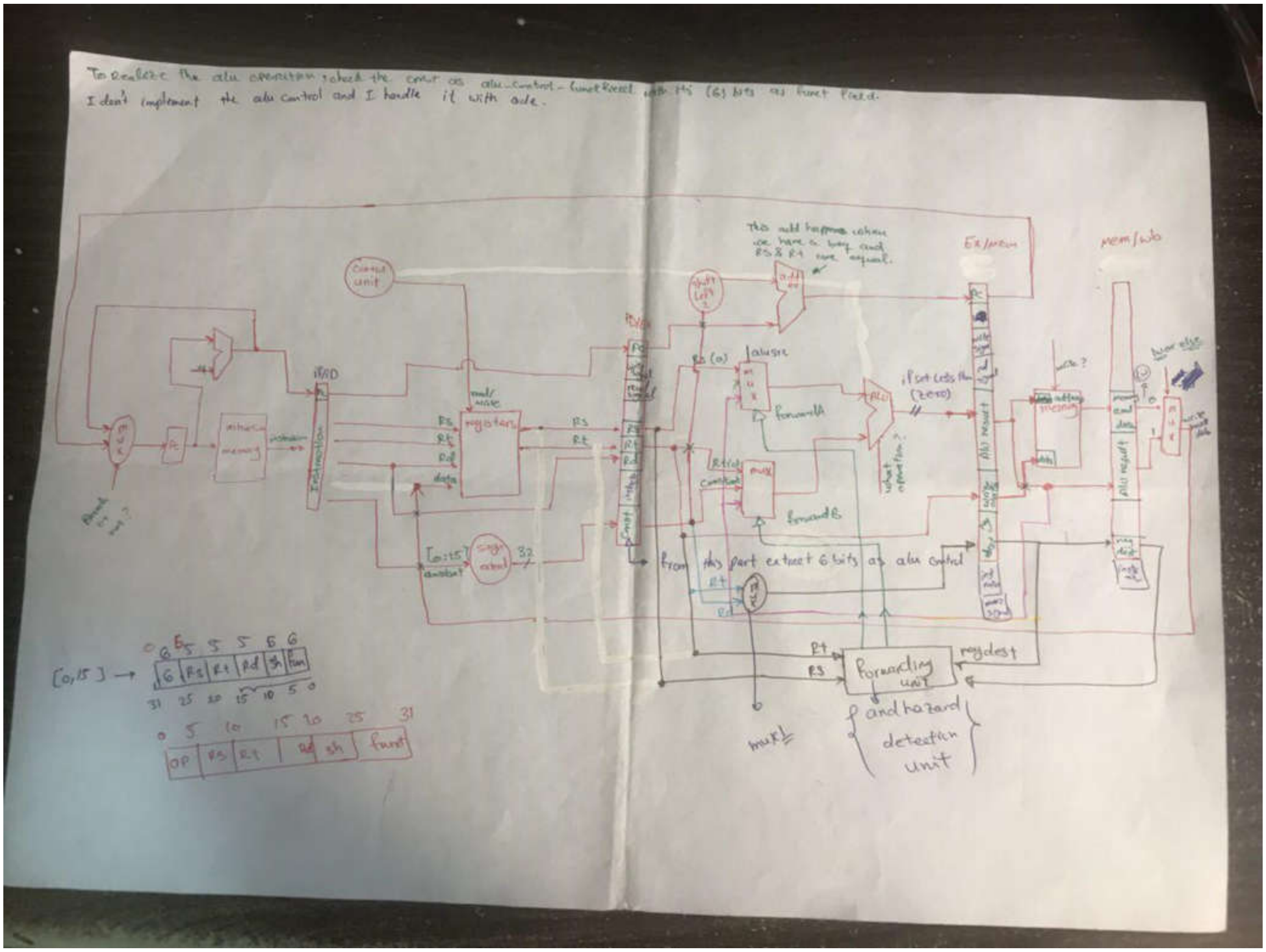
Test case 6

add 3, 3, 3 → 3 → 8
add 3, 3, 3 3 → 16
add 3, 3, 3 3 → 32
add 3, 3, 3 3 → 64
add 3, 3, 3 3 → 128
add 3, 3, 3 3 → 256
add 3, 3, 3 3 → 512

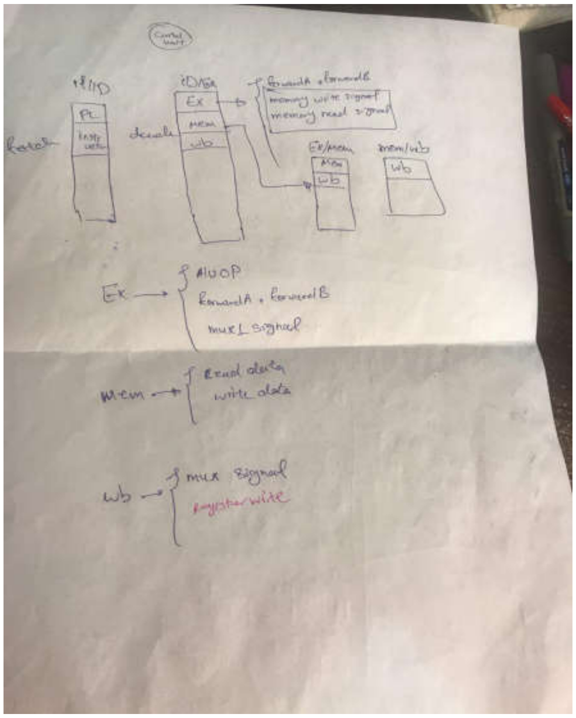


Simple Pipeline :

X No forwarding unit
X No hazard detection
✓ Pipeline
✓ supports → {add, sub, Xor, Nor, or, slt, sw, lw, Jump, beq}



To Realize the alu operation, check the cmr as alu_control-functField with its (6) bits as funct field.
I don't implement the alu control and I handle it with code.



Control unit
IF/ID
PC
instruction
fetch
decode
ID/EX
Ex
mem
wb
forwardA, forwardB
memory write signal
memory read signal
Ex/mem
mem
wb
mem/wb
wb
Ex →
AluOP
forwardA, forwardB
muxL signal
mem →
Read data
write data
wb →
mux signal
Register write