CSU34021 Tutorial 4

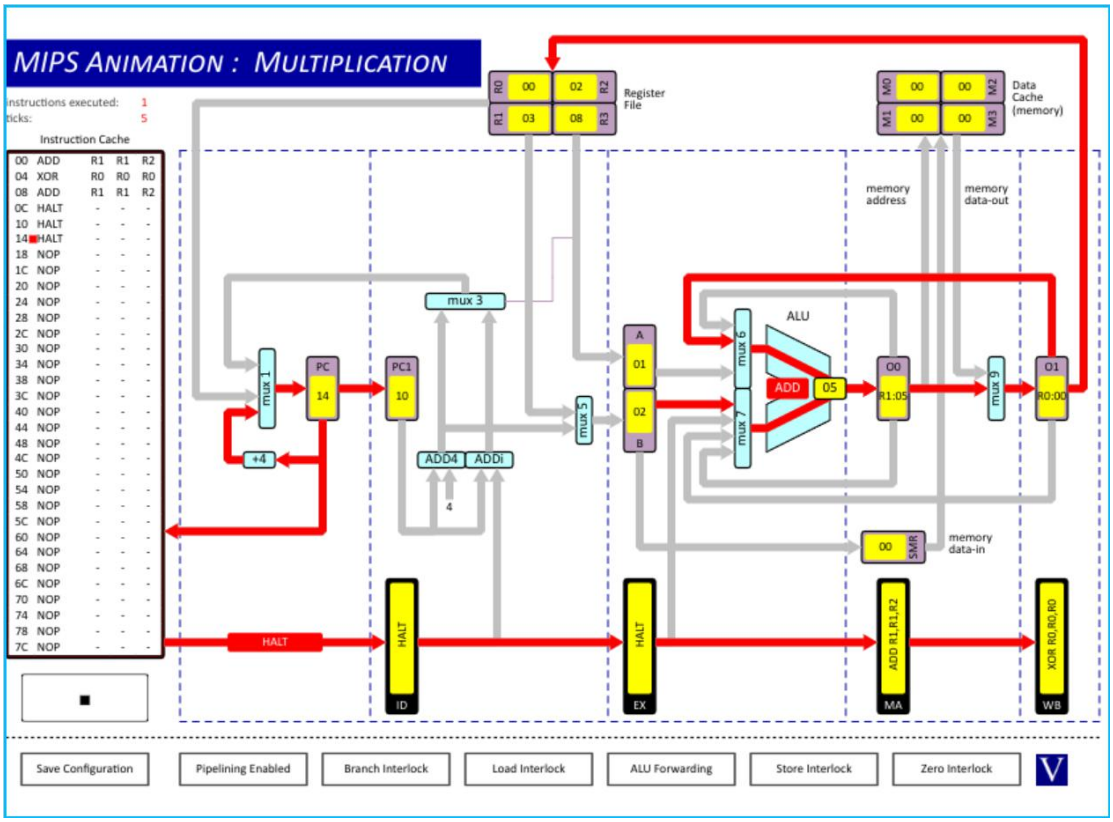
SiKai Lu
14333615

Q1. The figure below shows the internal data paths of the DLX/MIPS processor.

1. O1 to MUX6

Instruction 3 needs the result of the Instruction 3.The value R1 is fast forwarded into MUX6.

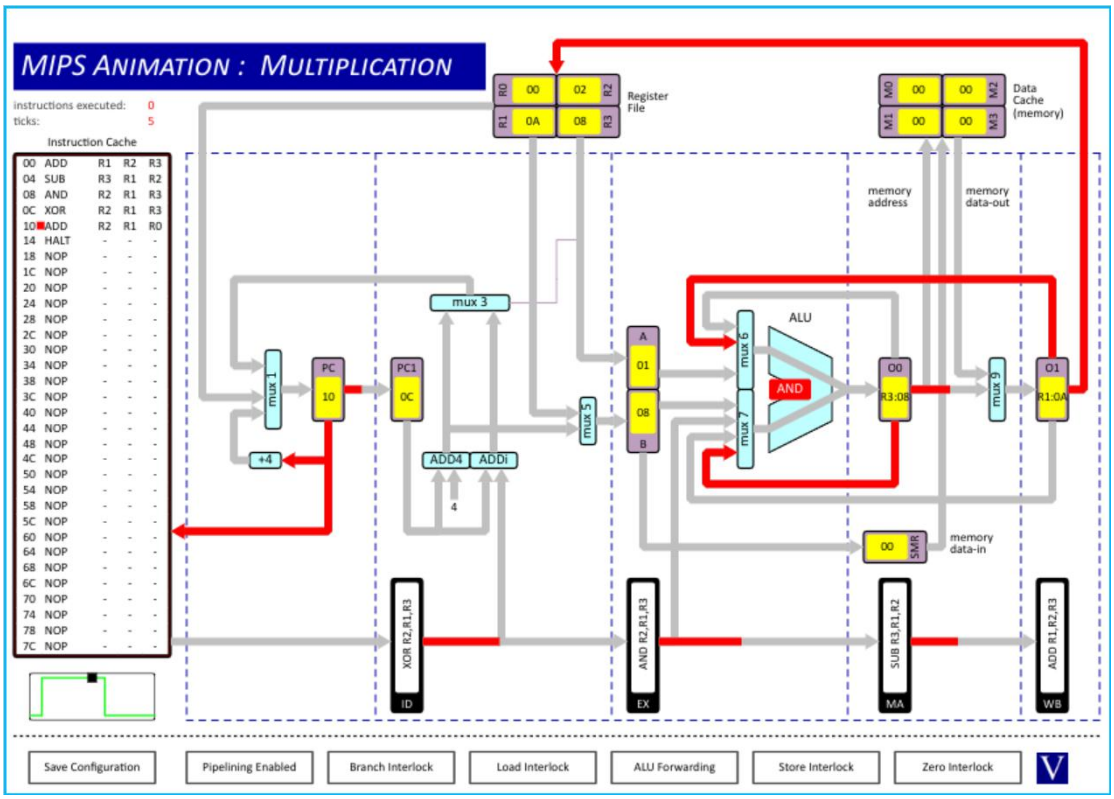
- ADD R1,R1,R2;
- XOR R0,R0,R0;
- AND R1,R1,R2;



2.O0 to MUX7 and O1 to MUX6 (simultaneously)

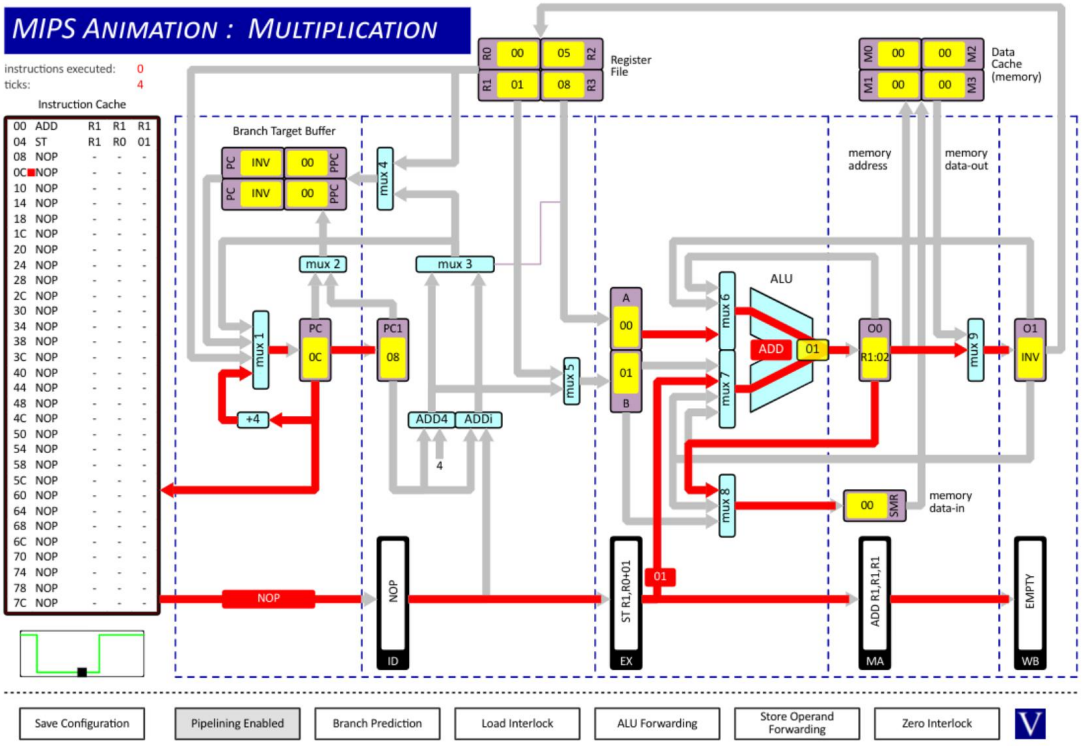
- ADD R1,R2,R3;
- SUB R3,R1,R2;
- AND R2,R1,R3;

Instruction 3 needs the results from two previous instructions,the value of R1 && R3 are fast forwarded to Instruction 3.



3. O0 to MUX8
    - ADD R1,R1,R1
    - ST R1,R0,01

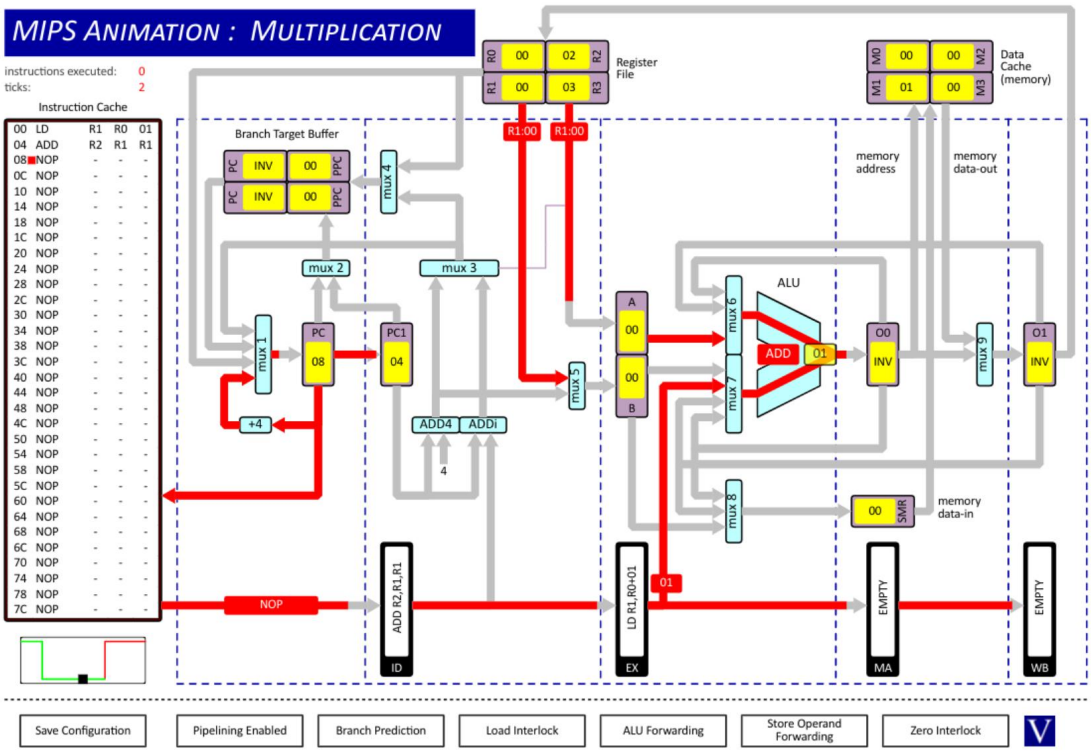
    Instruction 2 needs the result of Instruction I which temporarily stored in the O0 to be passed into SMDR(Stored(?) memory data register).



4.EX to MUX7

- LD R1,R0,#1;
- ADD R2,R1,R1;

Instruction 1 takes an immediate value,#1,as a operand required for the calculation of the effective address.

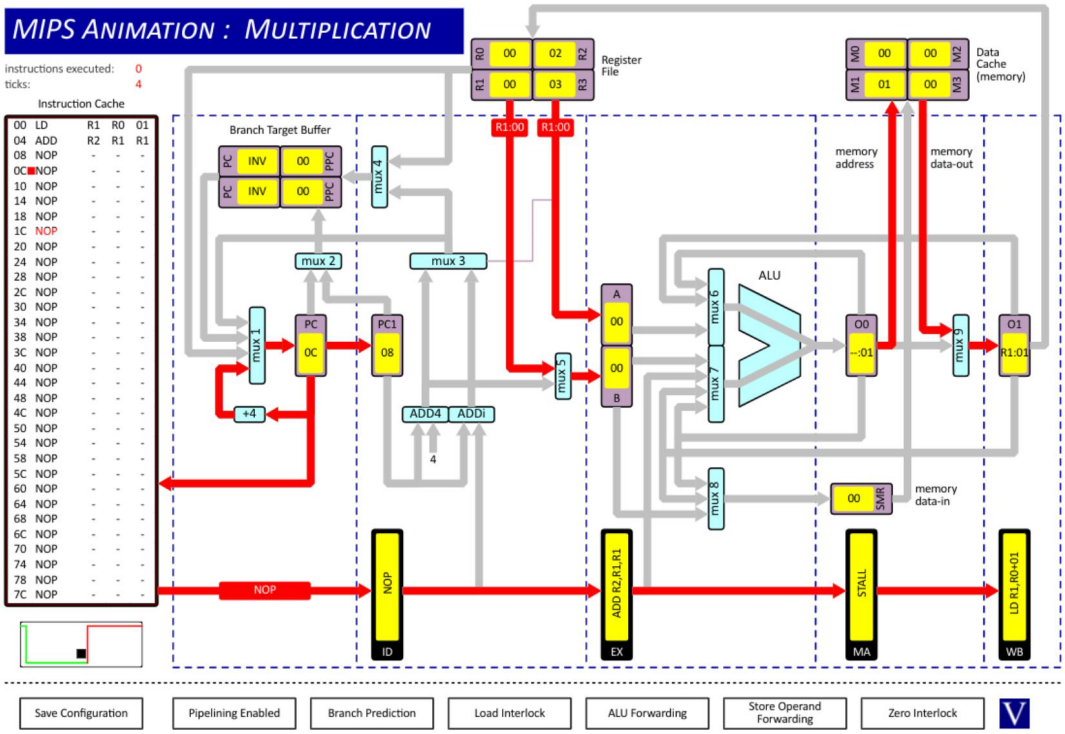


Q3. Click "Instruction Cache" until the program shown in Q1 is displayed. The program calculates r1 = r2*r3 (0x48 = 0x09 x 0x08)

5.Data Cache to MUX9

- LD R1,R0,#1;
- ADD R2,R1,R1;

Instruction 1 use the effective address to read data from Data Cache.Data cache passes the data into MUL9

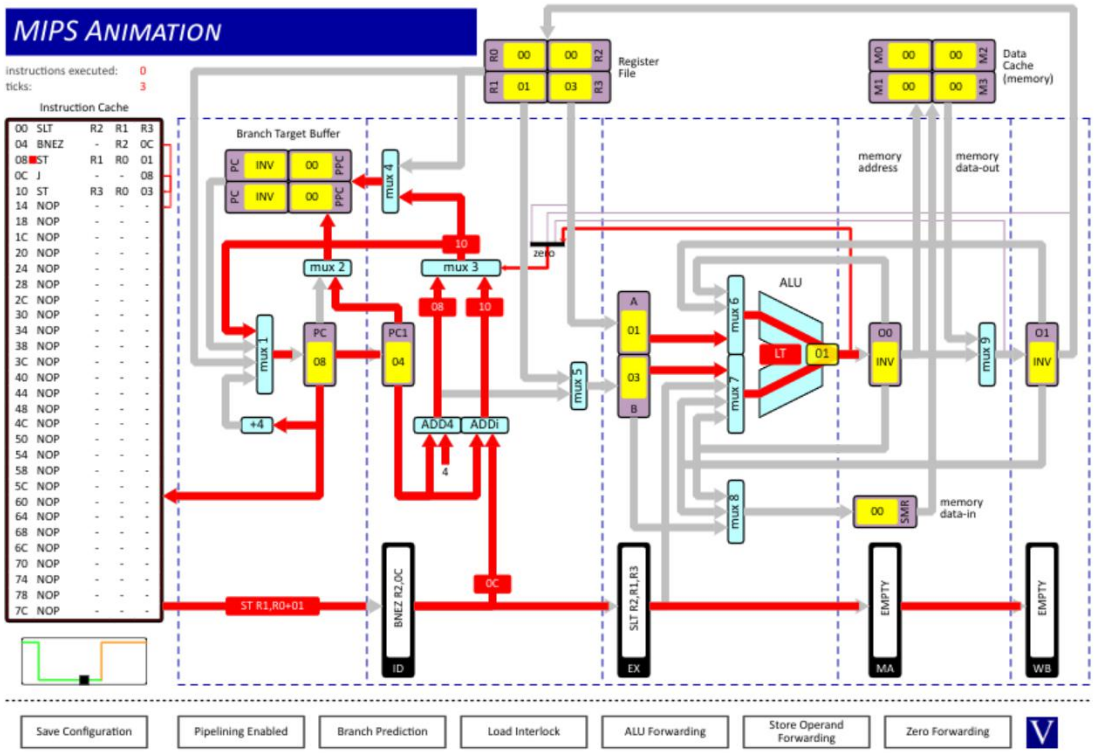


6. O0 to Zero detector

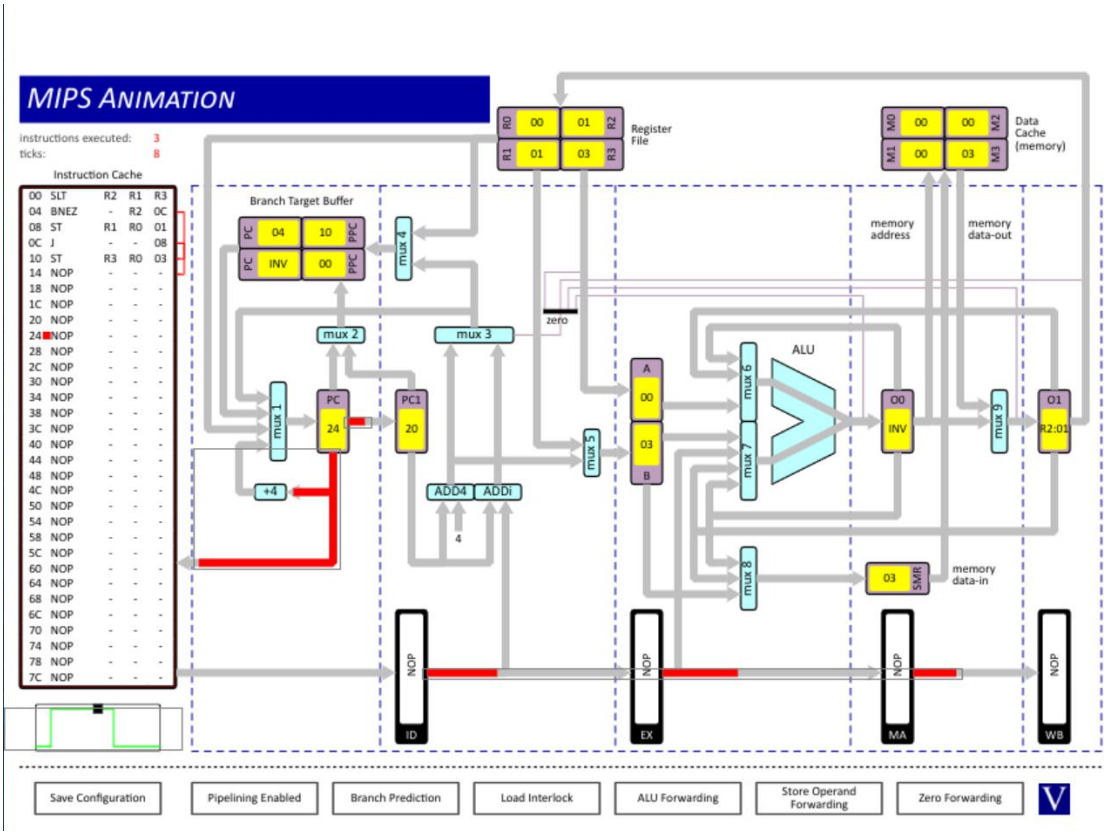
R1=1; R3=3;

- SLT R2,R1,R3;
- BNEZ R2,0C;
- ST R1,R0,#1;
- J #8;
- ST R3,R0,#3;

```
R2 = (R1<R3)?1:0
If(R2=0){
  M[1]=1;
}else{
   M[3]=3;
}
```



Result : Store #3 into M[3];



7. Register File to MUX1

R1=1;R3=3;

- SLT R2,R1,R3;
- BEQZ R2,0C;
- ST R1,R0,#1;
- ADDI R2,R2,#7
- JR R2;
- ST R3,R0,#3;

R2 = (R1<R3)?1:0

If(R2!=0){

M[3]=3;

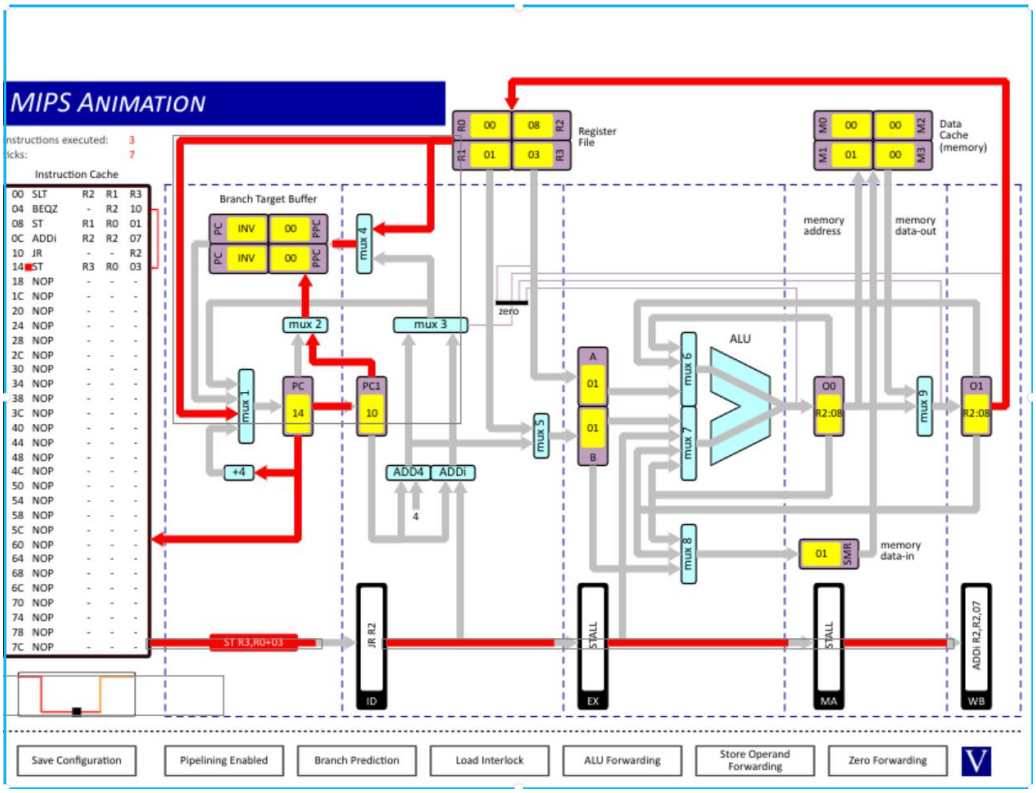
}else{

M[3]=1;

R2=R2+7;

Jump 2 instructions //skip ST R3,R0,#3;

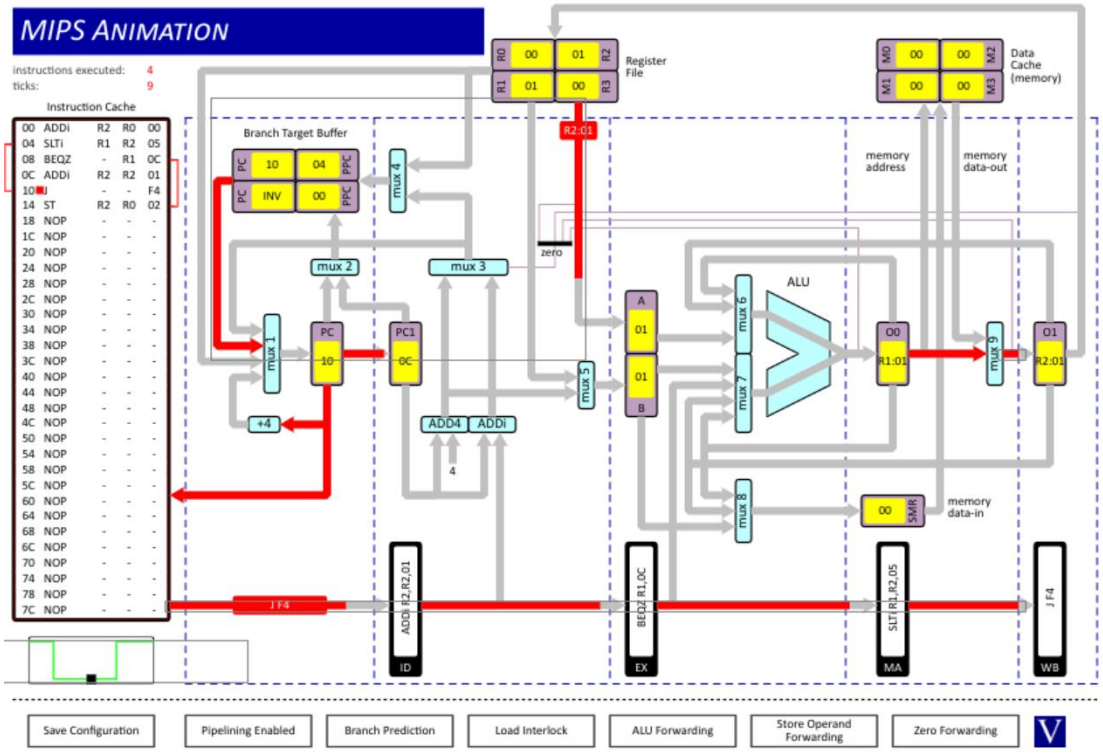
}



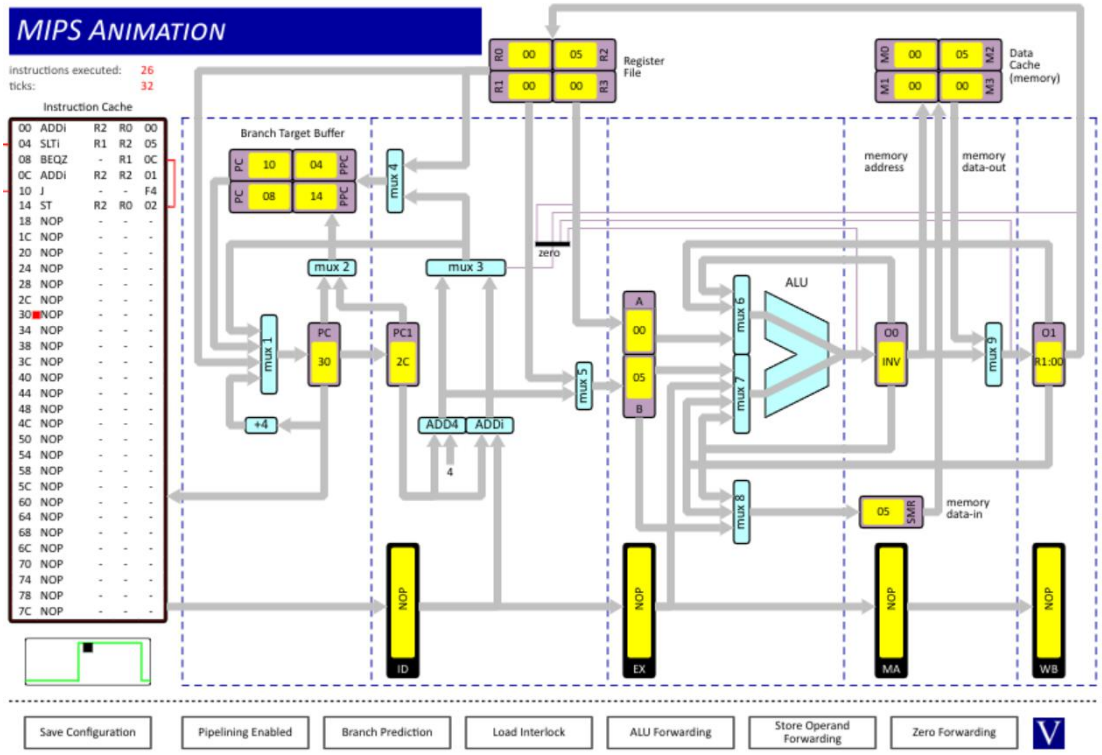
8.Branch Target Buffer to MUX1

- ADDI R2,R0,#0;
- SLTI R1,R2,#5;
- BEQZ R1,0C;
- ADDI R2,R2,#1;
- J F4;
- ST R2,R0,#2;

```
   R2=0;
While(R2<5){
     R2++;
}
M2[R2];
```



Result:


MIPS ANIMATION
instructions executed: 26
ticks: 32
Instruction Cache
00 ADDi R2 R0 00
04 SLTi R1 R2 05
08 BEQZ - R1 0C
0C ADDi R2 R2 01
10 J - - F4
14 ST R2 R0 02
18 NOP
1C NOP
20 NOP
24 NOP
28 NOP
2C NOP
30 NOP
34 NOP
38 NOP
3C NOP
40 NOP
44 NOP
48 NOP
4C NOP
50 NOP
54 NOP
58 NOP
5C NOP
60 NOP
64 NOP
68 NOP
6C NOP
70 NOP
74 NOP
78 NOP
7C NOP
Branch Target Buffer
PC 10 04 PPC
PC 08 14 PPC
Register File
R0 00 05 R2
R1 00 00 R3
Data Cache (memory)
M0 00 05 M2
M1 00 00 M3
memory address
memory data-out
memory data-in
mux 1
mux 2
mux 3
mux 4
mux 5
mux 6
mux 7
mux 8
mux 9
PC 30
PC1 2C
+4
ADD4
ADDi
4
zero
A 00
05 B
ALU
00 INV
05 SMR
O1 R1:00
NOP ID
NOP EX
NOP MA
NOP WB
Save Configuration
Pipelining Enabled
Branch Prediction
Load Interlock
ALU Forwarding
Store Operand Forwarding
Zero Forwarding

Q2. In case of ALU Forwarding is enabled

R1 = 15;

Number of clock cycles : 10;

In case of ALU forwarding is disabled with CPU data dependency interlocks enabled:

R1 = 15;

Number of clock cycles : 18;

In case of ALU forwarding with CPU data dependency interlocks disabled:

R1 = 6;

Number of clock cycles : 10;

ALU forwarding is the most efficient and correct one.It allows that an instruction uses the value of a register immediately after updating the register.In this case, this mechanism prevents 8 pipeline stalls which cause due to the fact that the next instruction compel to wait until the previous instruction loads the result into the register file.

ALU interlock mechanism force the next instruction to wait until the previous instruction loads the result into the register file.Even with pipeline stalls,it provides the ability of giving the correct answer.

If ALU interlock mechanism is disabled,the next instruction can be executed with obsolete registers' value which cause the incorrectness of the result.

Q3. Click "Instruction Cache" until the program shown in Q1 is displayed. The program calculates r1 = r2*r3 (0x48 = 0x09 x 0x08)

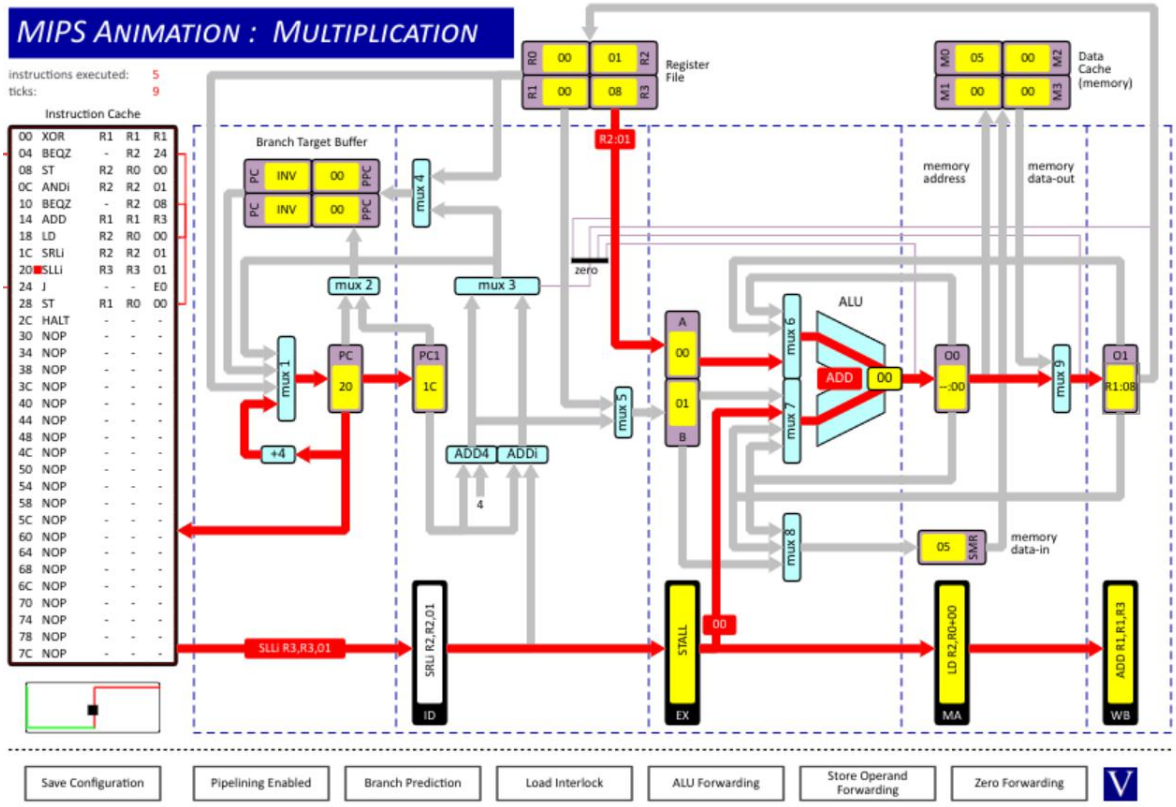
(1)

30 instructions are executed

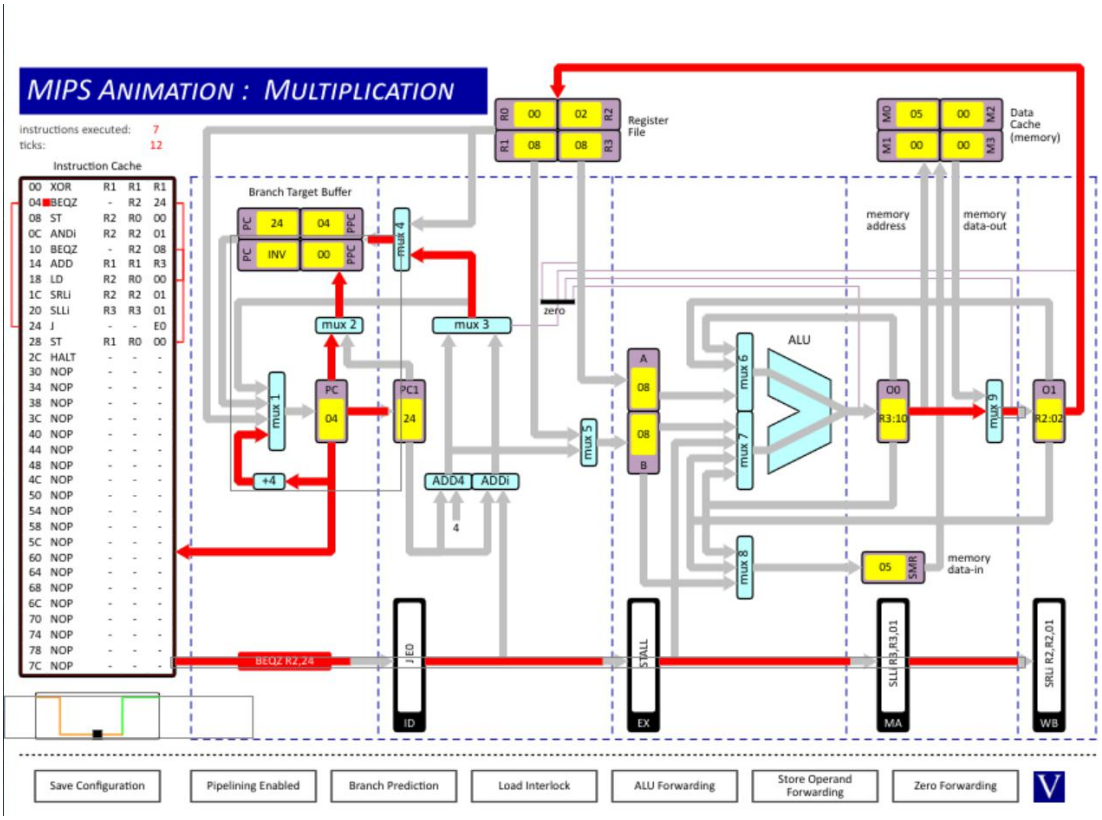
41 clock cycles

When the fifth last instruction finishes, one instruction just finish the first stage of 5 stages.30 + 7 stalls + 4 cycles = 41.

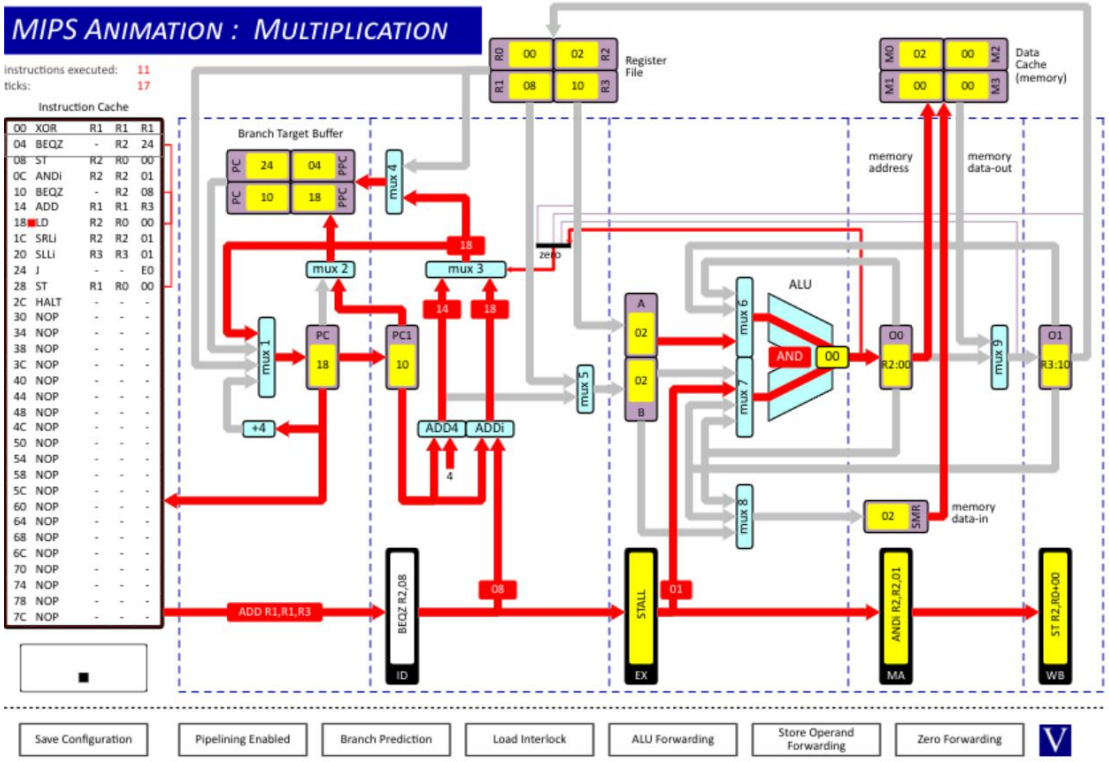
If two consecutive instructions need to access the same data,the second one compel to stall until the first one finish its functionality.



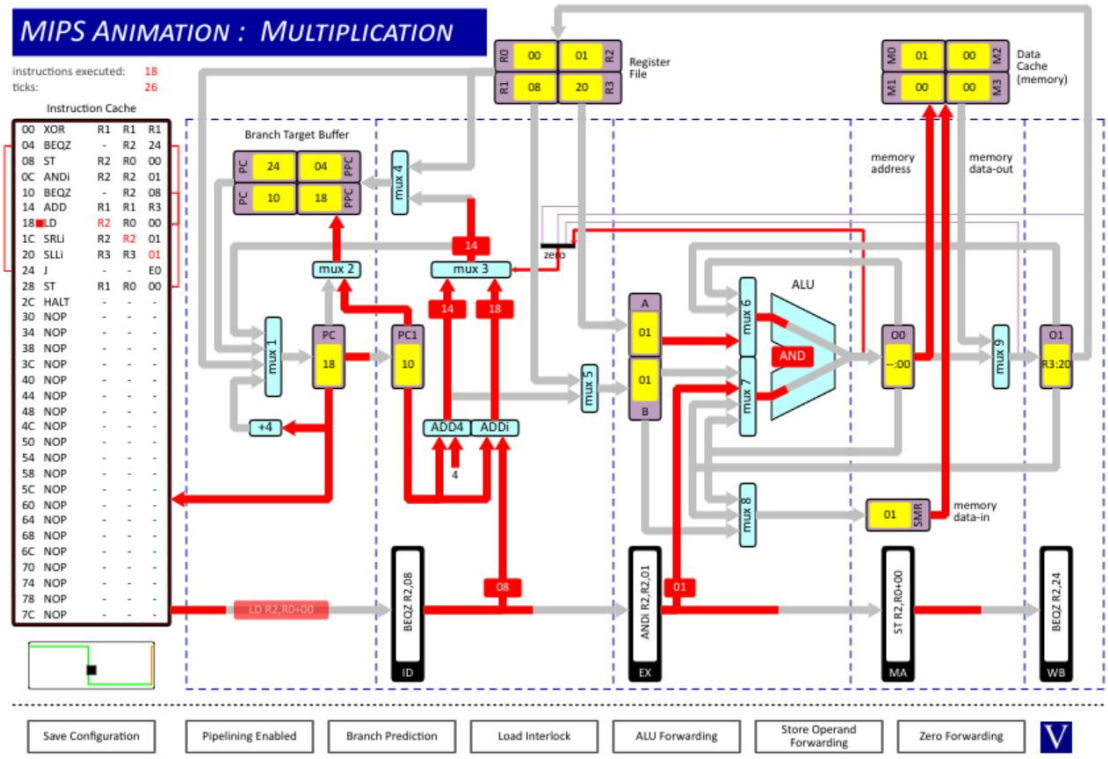
The first ,forth,sixth stall happens when the SRLI R2,R2,1 instruction is waiting for the load instruction reads the value of R2 into the register file.



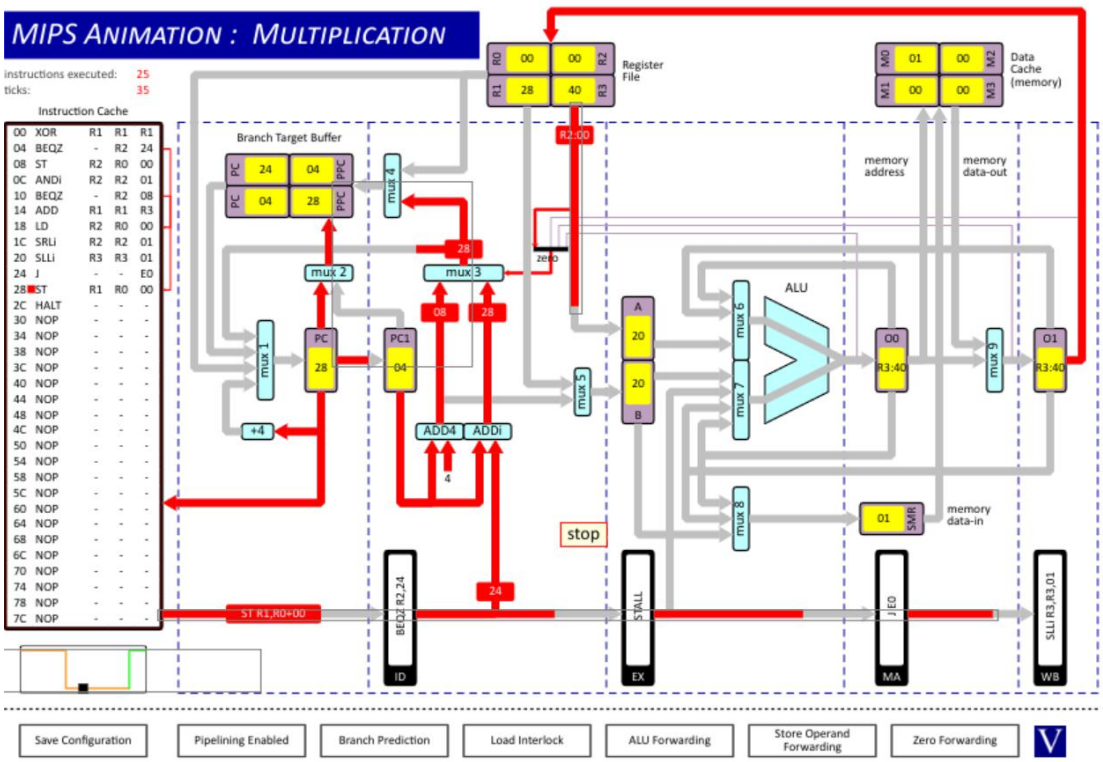
The second stall happens when the J E0 instruction is calculating the destination address and switch off the wrong instruction and reads the right one.



The third stall happens when R2 is equal to 0 after executing ANDI R2,R2,i in which the criterion of BEQZ is matched.The BEQZ calculates the destination address and switch off the wrong instruction and reads the right one.



The fifth stall happens when decoding BEQZ R2 8.Since the address of BEQZ is in the BTB, MIPS takes it for granted that the next instruction to be executed is BEQZ+8.Unfortunately, it's not the case.BEQZ has to discard the BEQZ+8 and read BEQZ+4.



The seventh stall happens when decoding BEQZ R2,24 and the value of R2 is 0.The address of the program counter increase by 24 instead of 4.The BEQZ calculates the destination address and switch off the wrong instruction and reads the right one.

(2)

30 instructions are executed
42 clock cycles
42-30-4=8 stalls.

Part one uses the branch prediction mechanism in which sometimes it predicts the address of the next instruction correctly.On the other hand,it commit stalls when predicts mistakenly.

Part two uses the branch interlock which manually calculate the address of the next instruction when encountered by branches and jumps.

8 stalls = 3 times jump stalls + 3 times load stalls + one time of BEQZ R2 24 + one time of BEQZ R2 08 in which the value of R2 is equal to 0;

(3)

30 instructions are executed
38 clock cycles

It reduces 3 clock cycles by swapping two shift instructions.

The SRLI R2,R2,R1 instruction cannot perform further action until the LD instruction finishes.By swapping SRLI and SLLI,SLLI, unlike SRLI, is capable of performing its functionality with the LD instruction concurrently.