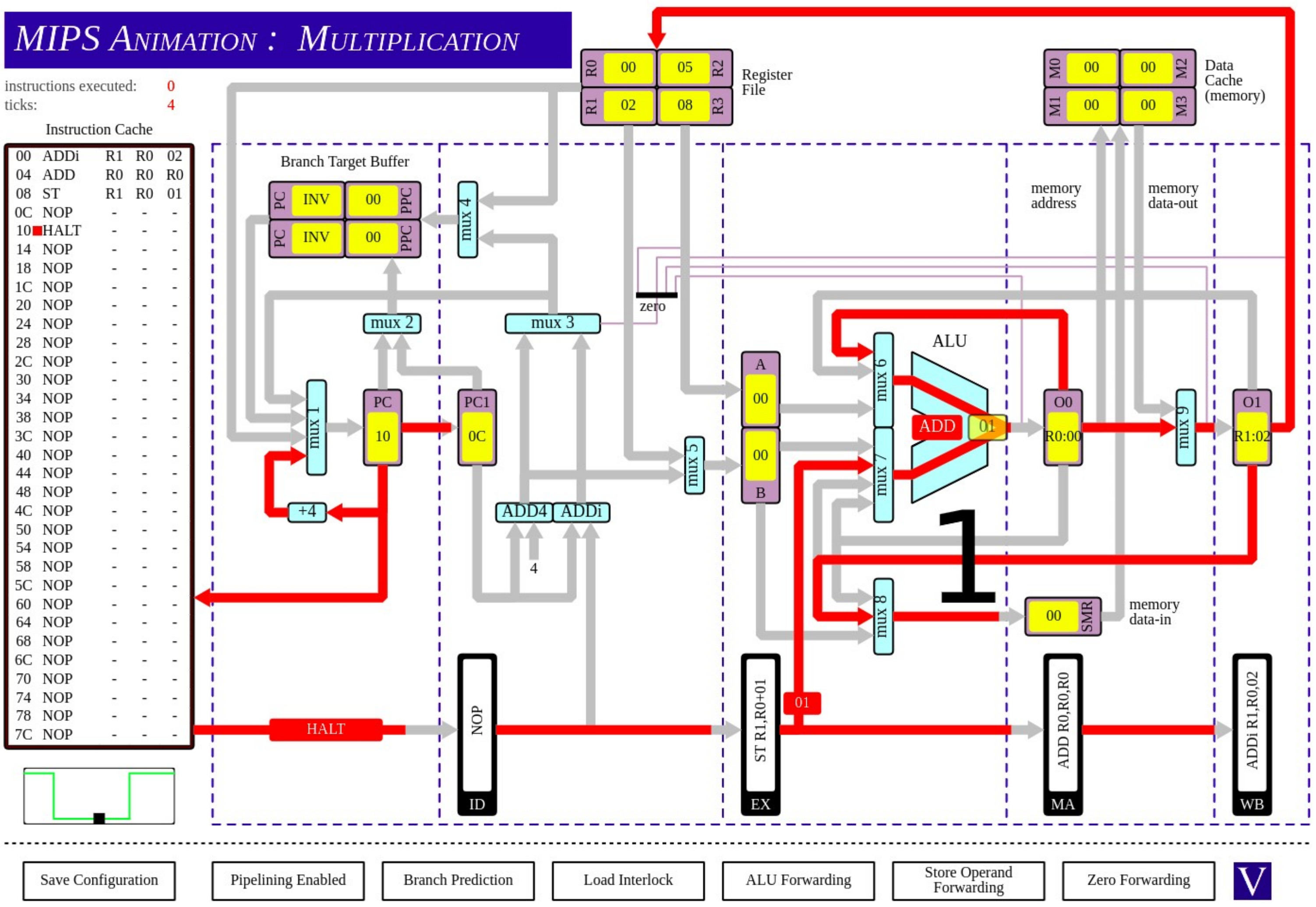
Question 1

1) O1 to Mux8

```
ADDi R1 R0 02
ADD  R0 R0 R0
ST   R1 R0 01
```



MIPS ANIMATION : MULTIPLICATION

instructions executed: 0
ticks: 4

Instruction Cache

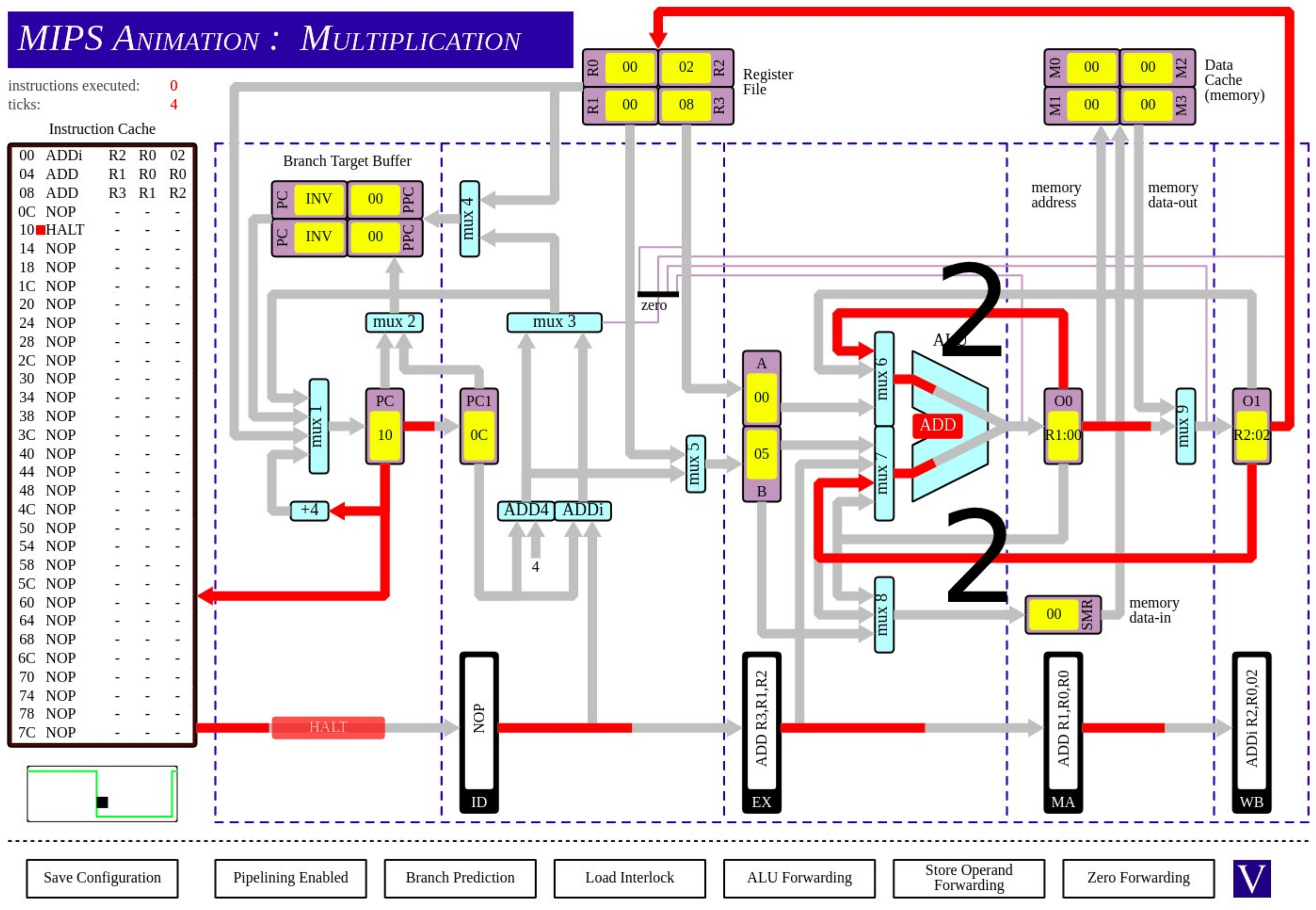
```
00  ADDi   R1  R0  02
04  ADD    R0  R0  R0
08  ST     R1  R0  01
0C  NOP    -   -   -
10 HALT    -   -   -
14  NOP    -   -   -
18  NOP    -   -   -
1C  NOP    -   -   -
20  NOP    -   -   -
24  NOP    -   -   -
28  NOP    -   -   -
2C  NOP    -   -   -
30  NOP    -   -   -
34  NOP    -   -   -
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    -   -   -
```

Register File: R0 00, R2 05, R1 02, R3 08

Data Cache (memory): M0 00, M2 00, M1 00, M3 00

2) O0 to Mux6 and O1 to Mux7 simultaneously

ADDi R2 R0 02
ADD R1 R0 R0
ADD R3 R1 R2



MIPS ANIMATION : MULTIPLICATION

instructions executed: 0
ticks: 4

Instruction Cache

00	ADDi	R2	R0	02
04	ADD	R1	R0	R0
08	ADD	R3	R1	R2
0C	NOP	-	-	-
10	HALT	-	-	-
14	NOP	-	-	-
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	-	-	-

Register File: R0 00, R1 00, R2 02, R3 08

Data Cache (memory): M0 00, M1 00, M2 00, M3 00

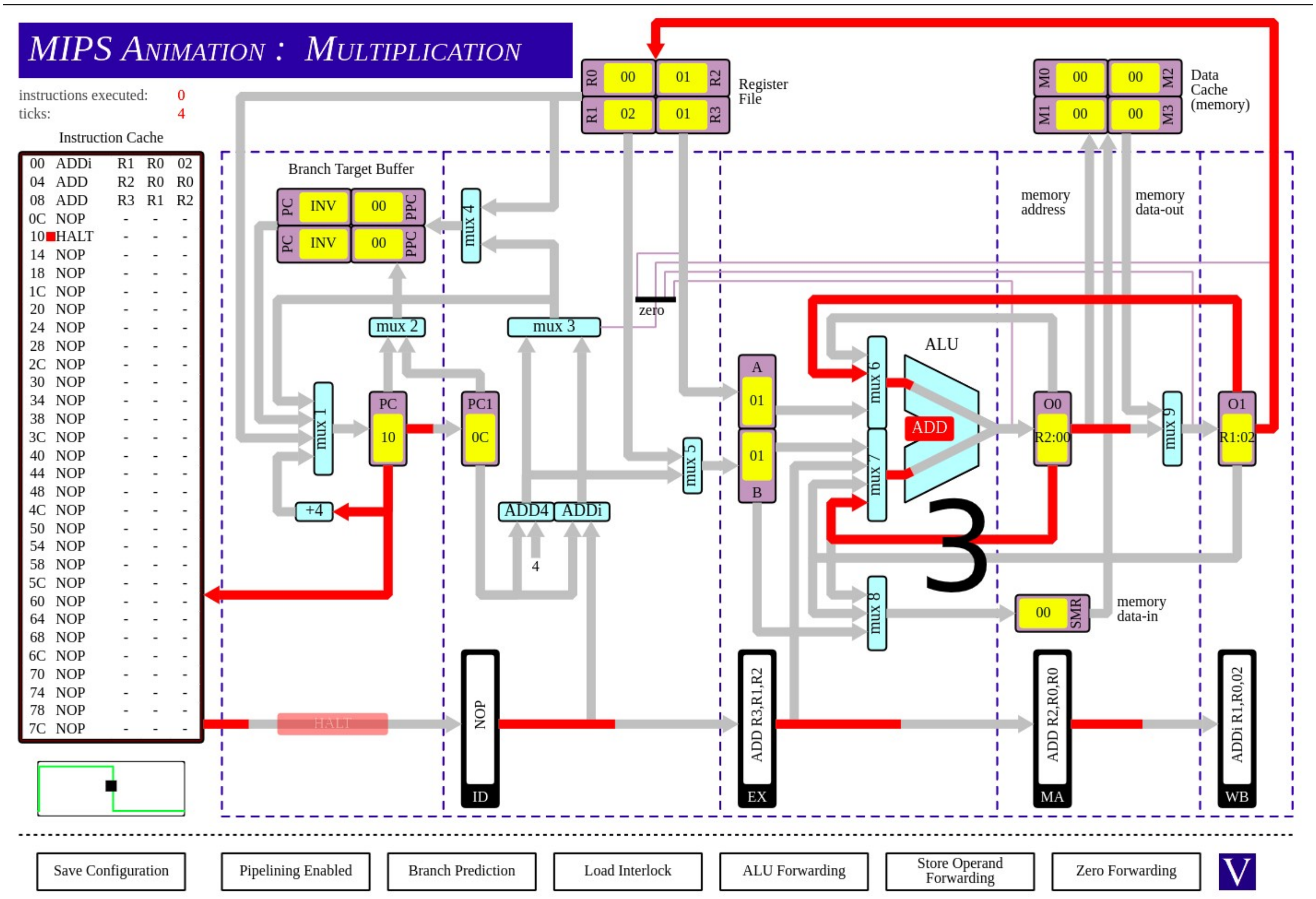
Branch Target Buffer: PC INV 00 PPC; PC INV 00 PPC

PC 10, PC1 0C, A 00, B 05, O0 R1:00, O1 R2:02, SMR 00

ID: NOP; EX: ADD R3,R1,R2; MA: ADD R1,R0,R0; WB: ADDi R2,R0,02

3) O0 to Mux7

ADDi R1 R0 02
ADD R2 R0 R0
ADD R3 R1 R2



MIPS ANIMATION : MULTIPLICATION

instructions executed: 0
ticks: 4

Instruction Cache

00	ADDi	R1	R0	02
04	ADD	R2	R0	R0
08	ADD	R3	R1	R2
0C	NOP	-	-	-
10	HALT	-	-	-
14	NOP	-	-	-
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	-	-	-

Register File

R0	00	01	R2
R1	02	01	R3

Data Cache (memory)

M0	00	00	M2
M1	00	00	M3

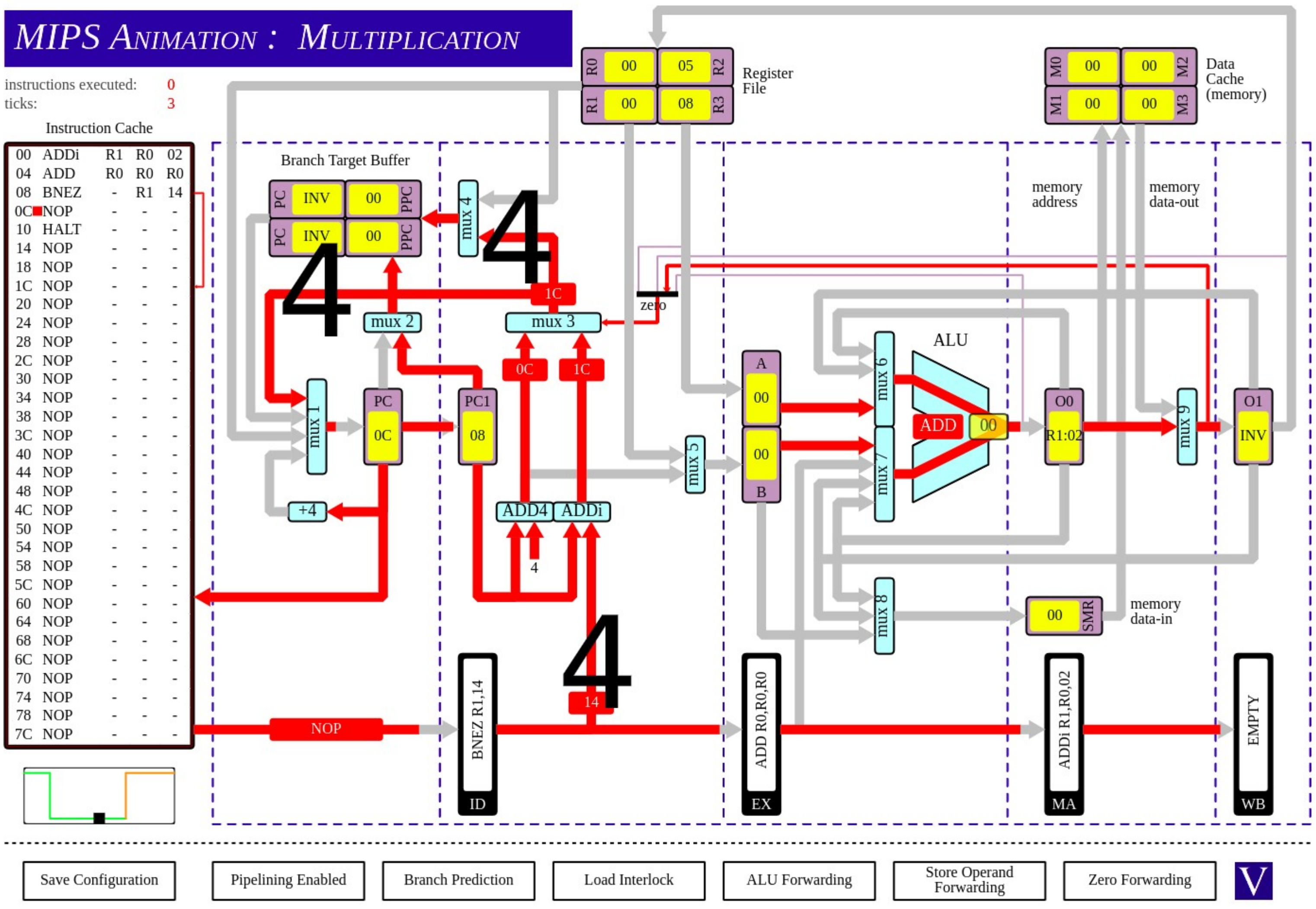
Branch Target Buffer

PC	INV	00	PPC
PC	INV	00	PPC

ID: NOP
EX: ADD R3,R1,R2
MA: ADD R2,R0,R0
WB: ADDi R1,R0,02

4) ID to Mux3 to Mux1/Mux4

ADDi R1 R0 02
ADD R0 R0 R0
BNEZ – R1 14


MIPS ANIMATION : MULTIPLICATION

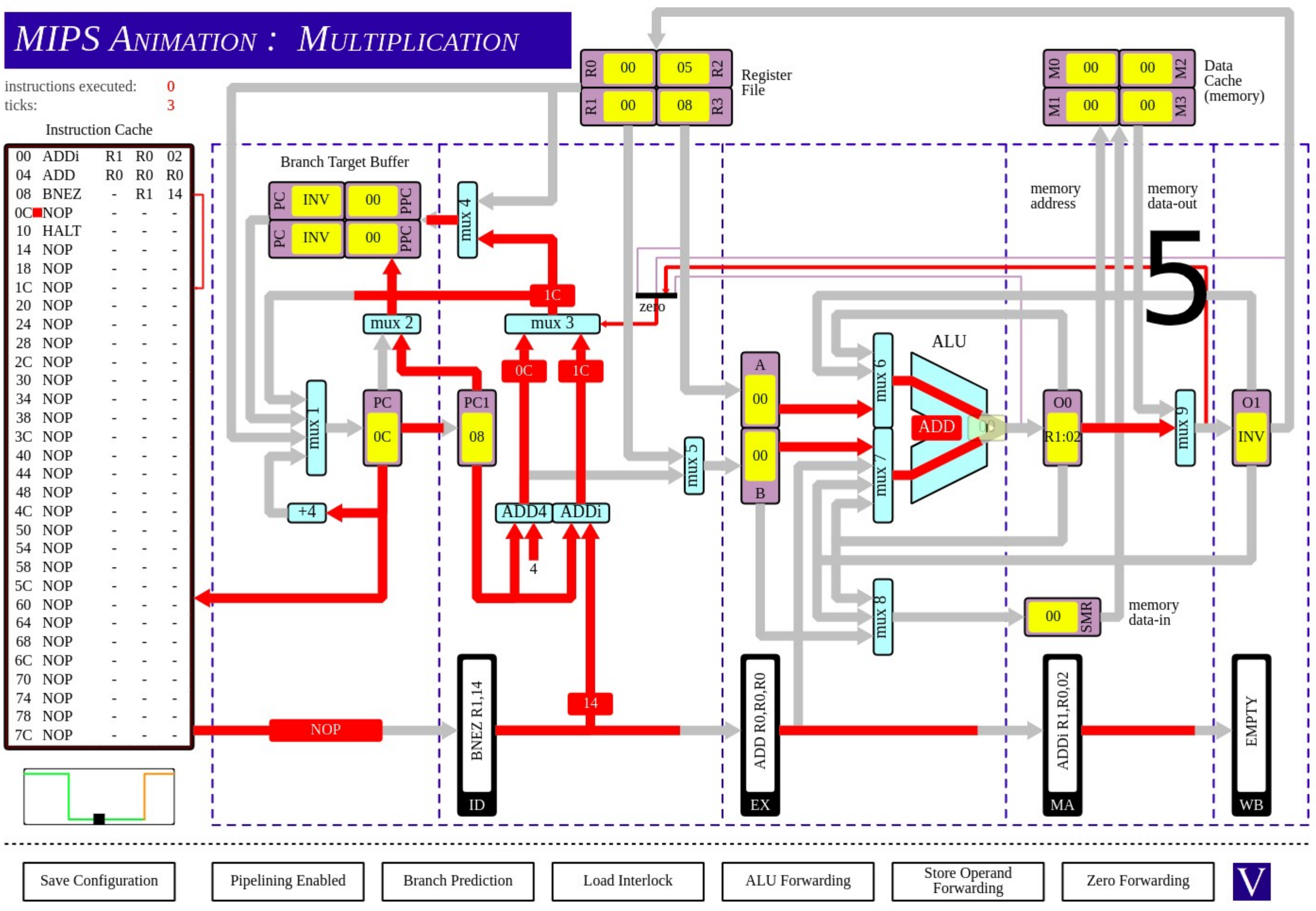
instructions executed: 0
ticks: 3

Instruction Cache

00	ADDi	R1	R0	02
04	ADD	R0	R0	R0
08	BNEZ	-	R1	14
0C	NOP	-	-	-
10	HALT	-	-	-
14	NOP	-	-	-
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	-	-	-

5) Mux9 out to Zero detector

ADDi R1 R0 02
ADD R0 R0 R0
BNEZ – R1 14



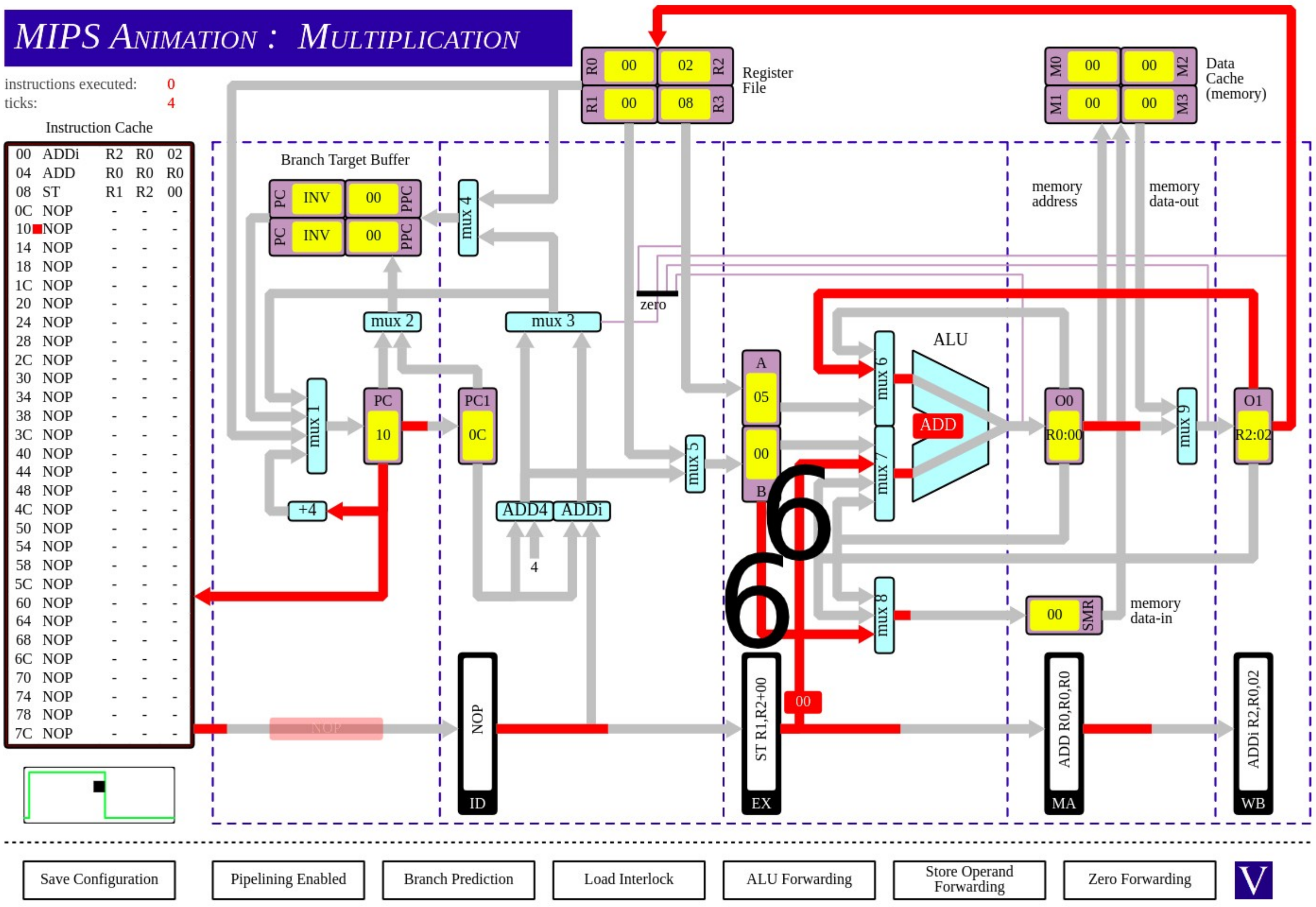
instructions executed: 0
ticks: 3

Instruction Cache

```
00  ADDi   R1  R0  02
04  ADD    R0  R0  R0
08  BNEZ   -   R1  14
0C  NOP    -   -   -
10  HALT   -   -   -
14  NOP    -   -   -
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    -   -   -
```

6) EX to Mux7 and B to Mux8 simultaneously

ADDi R2 R0 R2
ADD R0 R0 R0
ST R1 R2 00



00	ADDi	R2	R0	02
04	ADD	R0	R0	R0
08	ST	R1	R2	00
0C	NOP	-	-	-
10	NOP	-	-	-
14	NOP	-	-	-
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	-	-	-

Question 2

This table shows the computed result and number of ticks needed to execute the code.

	R1 result	**R2 result**	**ticks**
ALU forwarding	0x01	0x00	10
ALU interlock	0x01	0x00	18
No ALU interlock	0x02	0x01	10

Using ALU forwarding and using ALU interlock generated the exact same results (0x01 and 0x01). This is the result assuming conventional instruction operation.

	R1 0x01	**R2 0x02**
r1 = r1 + r2	R1 = 0x03	
r2 = r1 - r2		R2 = 0x01
r1 = r1 ^ r2	R1 = 0x01	
r2 = r1 | r2		R2 = 0x00
r1 = r1 + r2	R1 = 0x01	

When we are not using ALU interlock, the register values from the register file that are obtained during the ID phase are used. This could potentionally be before the results of a previous instruction has been computed and propagated back to the register file. Our results in r1 and r2 differ from before, see below.

	RF r1	**result**	**RF r2**
r1 = r1 + r2	0x01	0x03	0x02
r2 = r1 - r2	0x01	0xFF (-1)	0x02
r1 = r1 ^ r2	0x03	0x00	0x02
r2 = r1 | r2	0x03	0x01	0xFF (-1)
r1 = r1 + r2	0x00	0x02 (r1)	0x01

The program took 10 ticks to execute when ALU forwarding was enabled and also when no ALU interlock was used. This consisted of four ticks to fill the pipeline and six ticks to execute the six instructions (including HALT).

Using ALU interlock adds 8 ticks, as after the first instruction, the four ADD instructions stall for two ticks each as they have to wait for the results of the previous instructions to be computed and propagate to the register file.

	Data stalls	**Reason**
r1 = r1 + r2	0	Wait for r1
r2 = r1 - r2	2	Wait for r2 (and r1)
r1 = r1 ^ r2	2	Wait for r1 (and r2)
r2 = r1 | r2	2	Wait for r2 (and r1)
r1 = r1 + r2	2	Wait for r1 (and r2)

Question 3

(i)

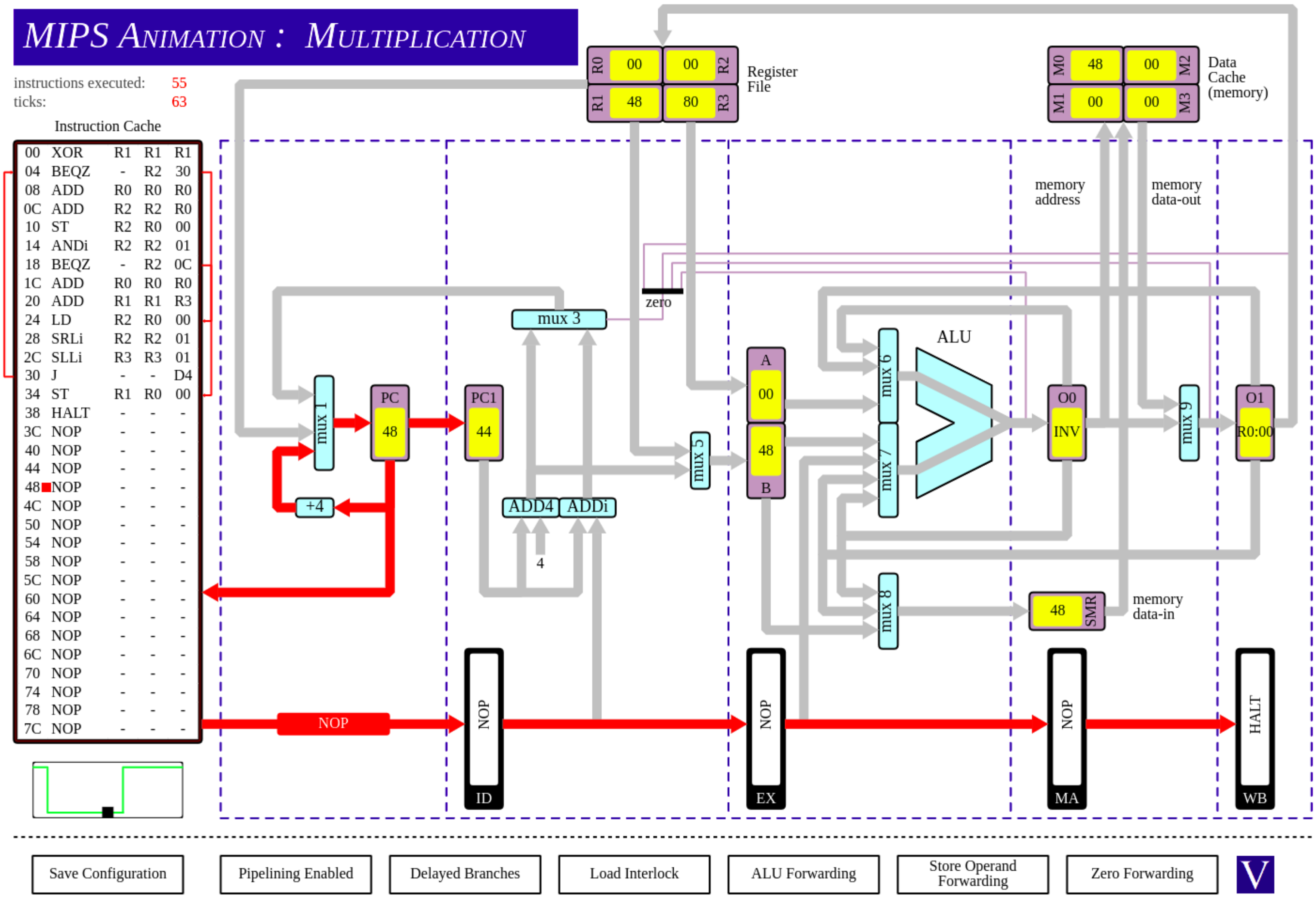
Using branch prediction, it took 38 instructions and 50 to execute the code. The instruction execution order is shown in the table below.

XOR	1				
BEQZ	2	11	19	27	36 **+1**
ST	3	12	20	28	
ANDi	4	13	21	29	
BEQZ	5	14 **+1**	22	30 **+1**	
ADD	6			31	
LD	7	15	23	32	
SRLi	8 *+1*	16 *+1*	24 *+1*	33 *+1*	
SLLi	9	17	25	34	
J	10 **+1**	18	26	35	
ST					37
HALT					38

The 50 ticks consist 4 ticks to fill the pipeline, 38 ticks for the 38 instructions and 8 ticks for the stalls (a stall is marked by a +1 in the table) . I represented a data stall with a red +1 (load hazard in this case) and I represented a control stall with a blue +1. For example, the second BEQZ instruction is correctly predicted tupon its first execution due to the fact that it doesn't branch (this is the default prediction as the branch is not in the branch target buffer), it is incorrectly predicted the second time due to the fact that it branches now, correctly the third time as it branches again and incorrectly the fourth time as it doesn't branch.

(ii)

By enabling delayed branches, this messes up the program and it no longer generates the right result for a multiplication program. This is due to the fact that with branch prediction disabled and delayed branches enabled, the instruction that follows a branch instruction will be executed no matter what. To combat this, I added an ADD R0 R0 R0 instruction after each branch instruction to delay the code so that the delayed branch would have no effect. It worked a treat as you can see below, generating the correct result again (0x48).



MIPS ANIMATION : MULTIPLICATION

instructions executed: 55
ticks: 63

Instruction Cache

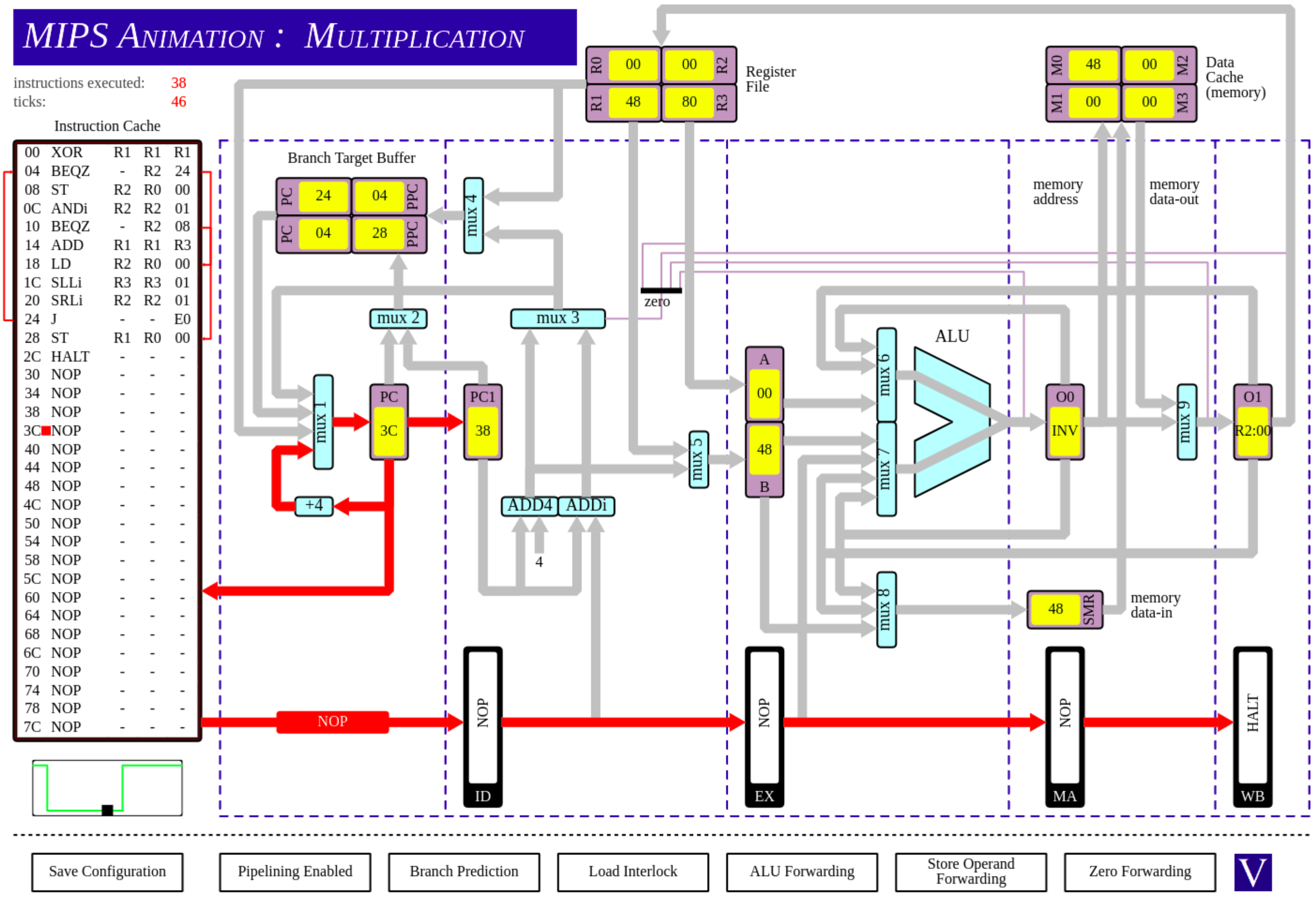
```
00  XOR    R1  R1  R1
04  BEQZ   -   R2  30
08  ADD    R0  R0  R0
0C  ADD    R2  R2  R0
10  ST     R2  R0  00
14  ANDi   R2  R2  01
18  BEQZ   -   R2  0C
1C  ADD    R0  R0  R0
20  ADD    R1  R1  R3
24  LD     R2  R0  00
28  SRLi   R2  R2  01
2C  SLLi   R3  R3  01
30  J      -   -   D4
34  ST     R1  R0  00
38  HALT   -   -   -
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    -   -   -
```

Register File: R0 00, R2 00, R1 48, R3 80
Data Cache (memory): M0 48, M2 00, M1 00, M3 00

The program now takes 63 ticks to execute, and 55 instructions. This is due to the fact that I added the ADD instruction after the branch instructions to stall the pipeline so that the branch instructions would work correctly.

(iii)

The data dependency that I identified was that the program was stalling after the LD instruction as it needed to wait for the load instruction to execute before it could perform the next instruction, as the SRLi instruction depended on using the R2 value that was loaded. To remove this data dependency, I simply swapped the SRLi and SLLi instructions as they did not depend on each other and the SLLi had no correlation with the LD instruction so there was no stall in this segment of the code anymore. Removing this stall from the code reduced the amount of ticks from 50 to 46.



MIPS ANIMATION : MULTIPLICATION

instructions executed: 38
ticks: 46

Instruction Cache

```
00  XOR    R1  R1  R1
04  BEQZ   -   R2  24
08  ST     R2  R0  00
0C  ANDi   R2  R2  01
10  BEQZ   -   R2  08
14  ADD    R1  R1  R3
18  LD     R2  R0  00
1C  SLLi   R3  R3  01
20  SRLi   R2  R2  01
24  J      -   -   E0
28  ST     R1  R0  00
2C  HALT   -   -   -
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    -   -   -
```

Register File: R0 00, R2 00, R1 48, R3 80

Data Cache (memory): M0 48, M2 00, M1 00, M3 00