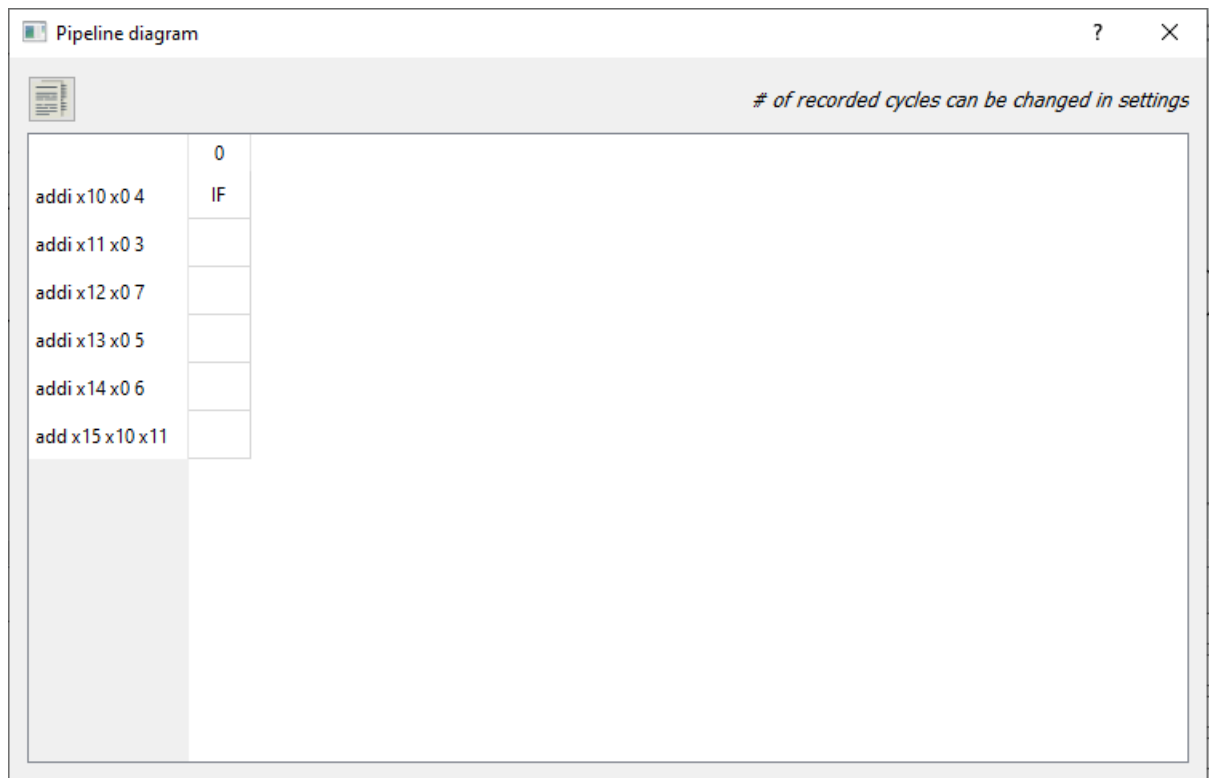


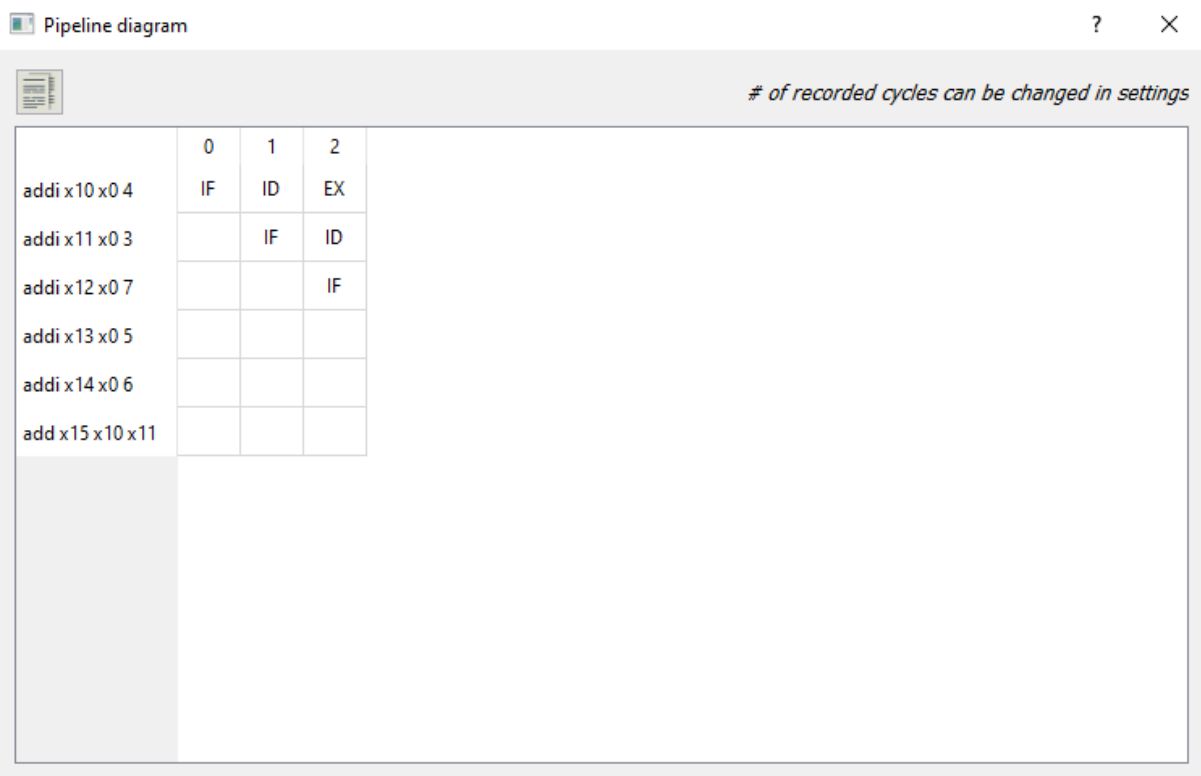
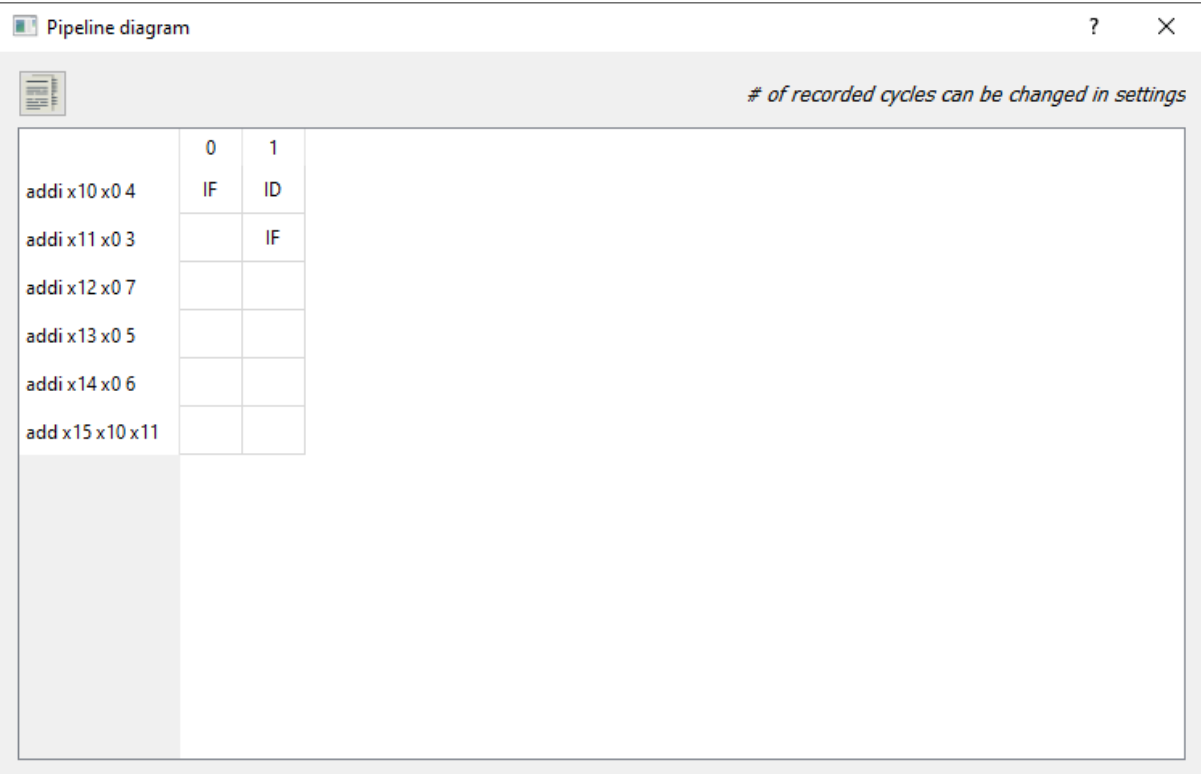
Levi de Lima Pereira Júnior - 121210472

Roteiro 8

Código 1 -

```
addi x10, x0, 4
addi x11, x0, 3
addi x12, x0, 7
addi x13, x0, 5
addi x14, x0, 6
add x15, x10, x11
```





Pipeline diagram

? X



of recorded cycles can be changed in settings

	0	1	2	3
addi x10 x0 4	IF	ID	EX	MEM
addi x11 x0 3		IF	ID	EX
addi x12 x0 7			IF	ID
addi x13 x0 5				IF
addi x14 x0 6				
add x15 x10 x11				

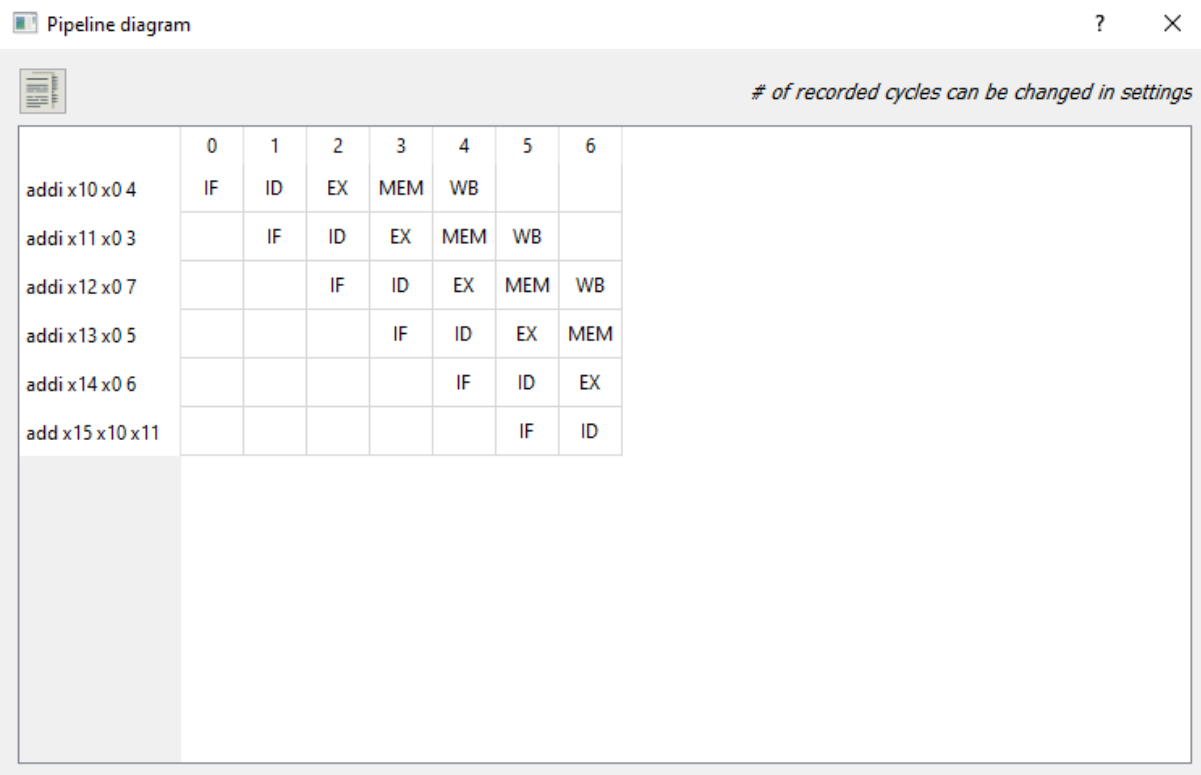
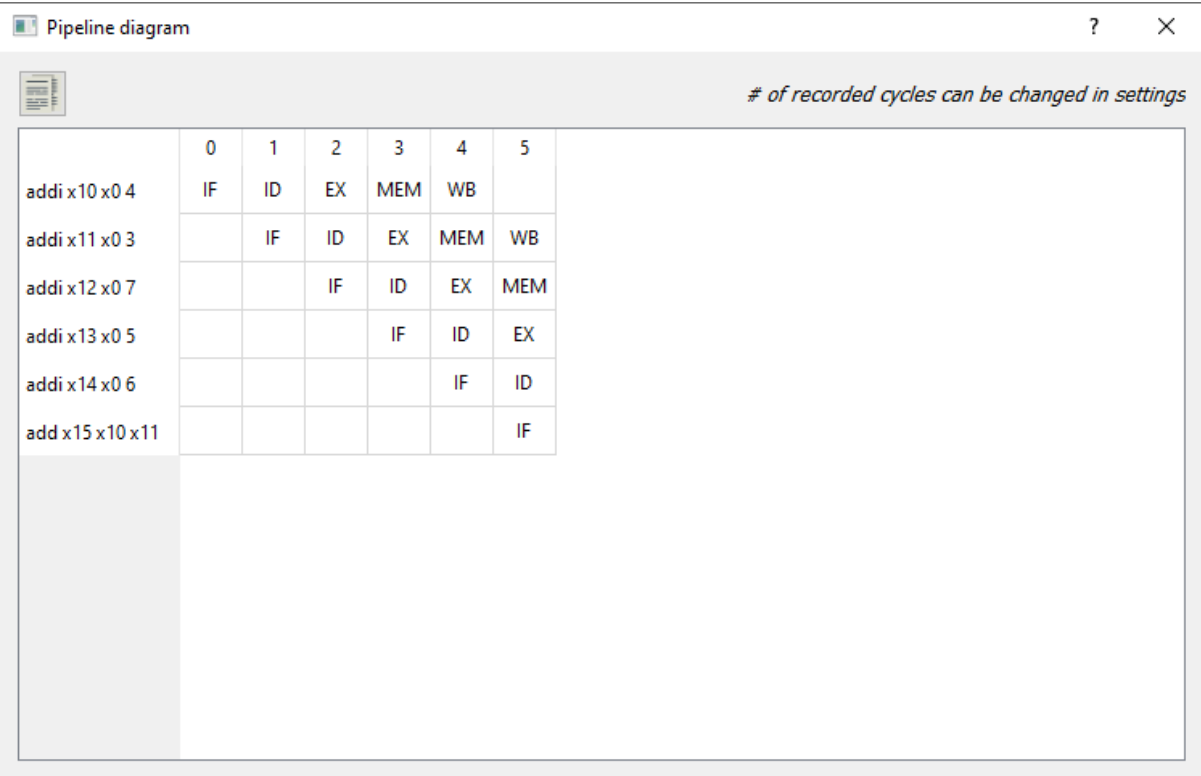
Pipeline diagram

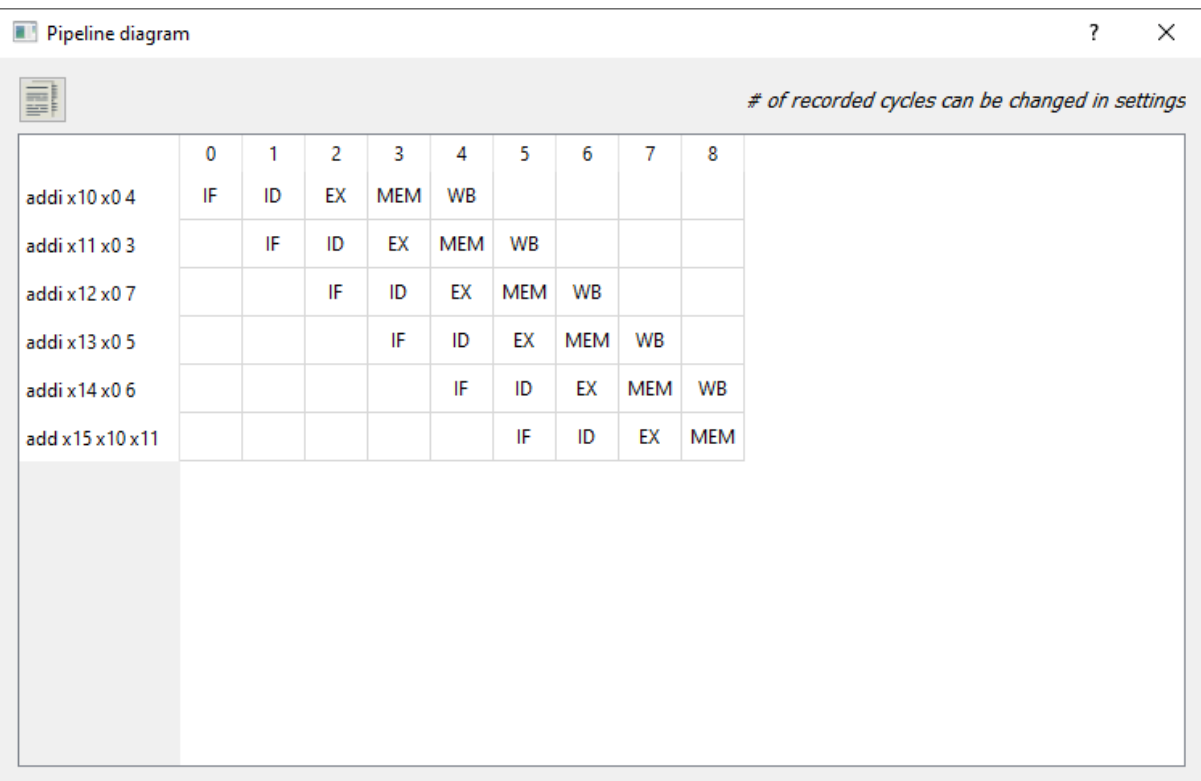
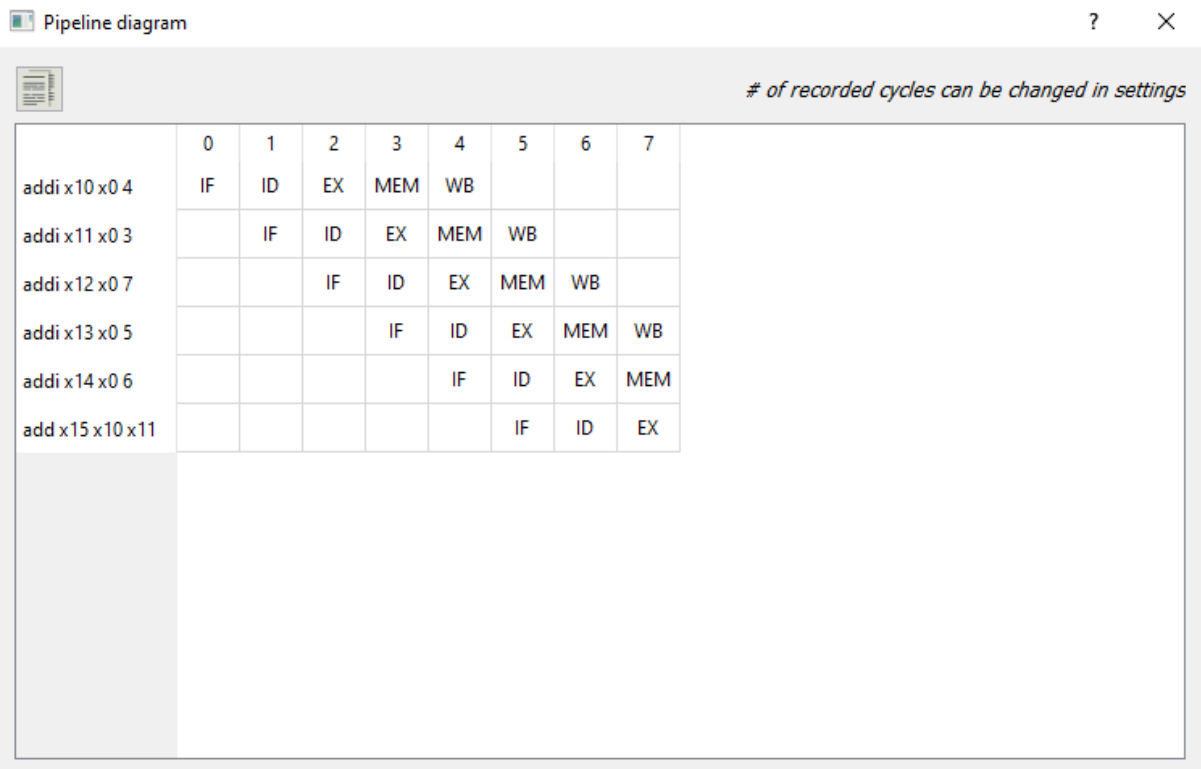
? X

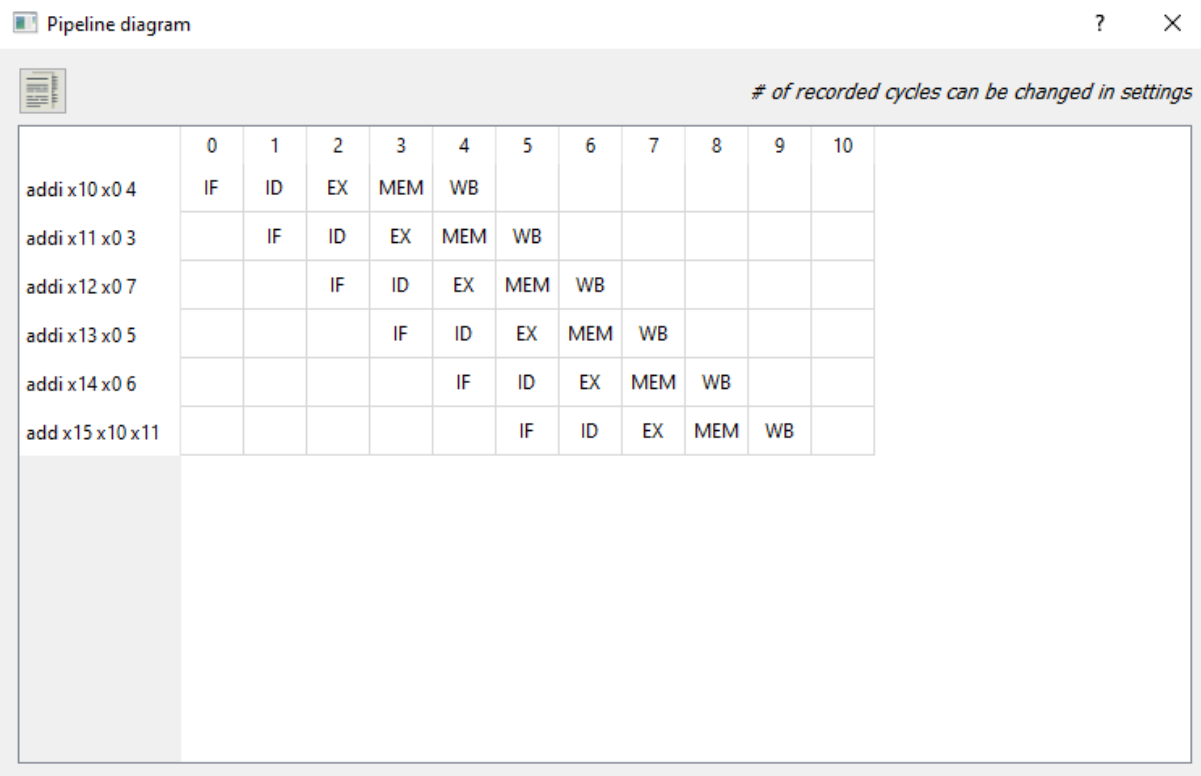
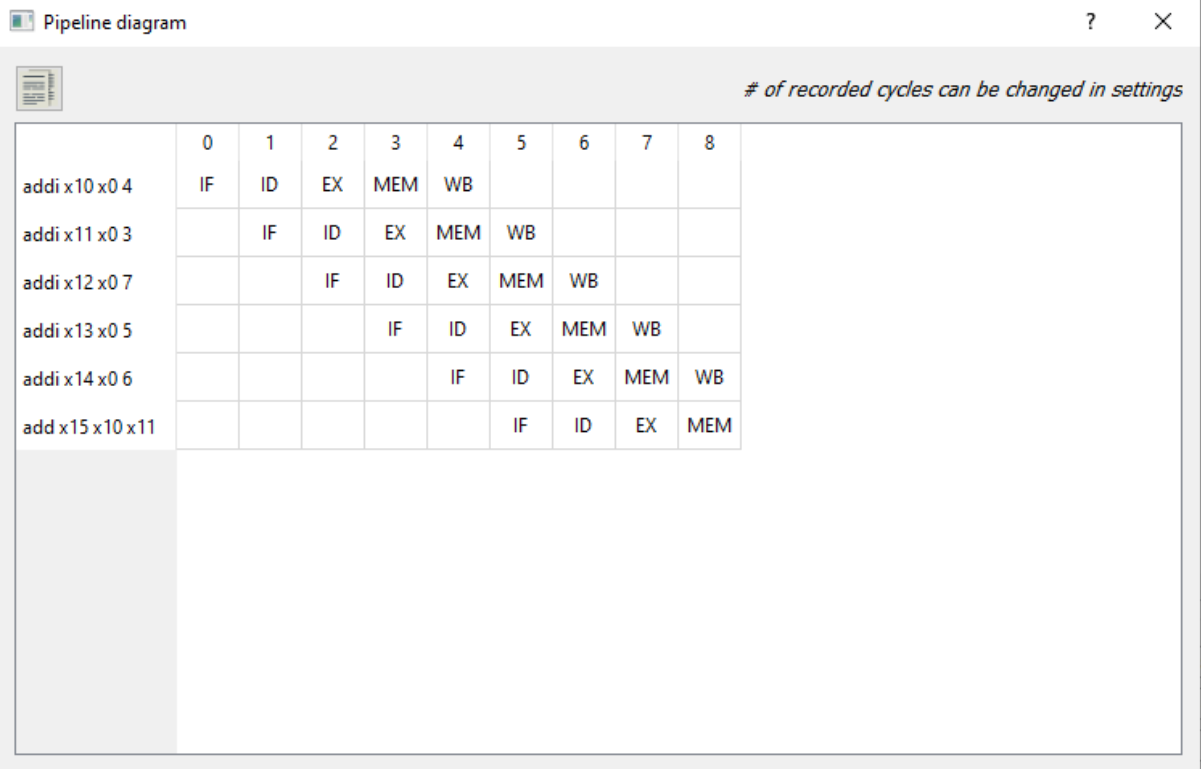


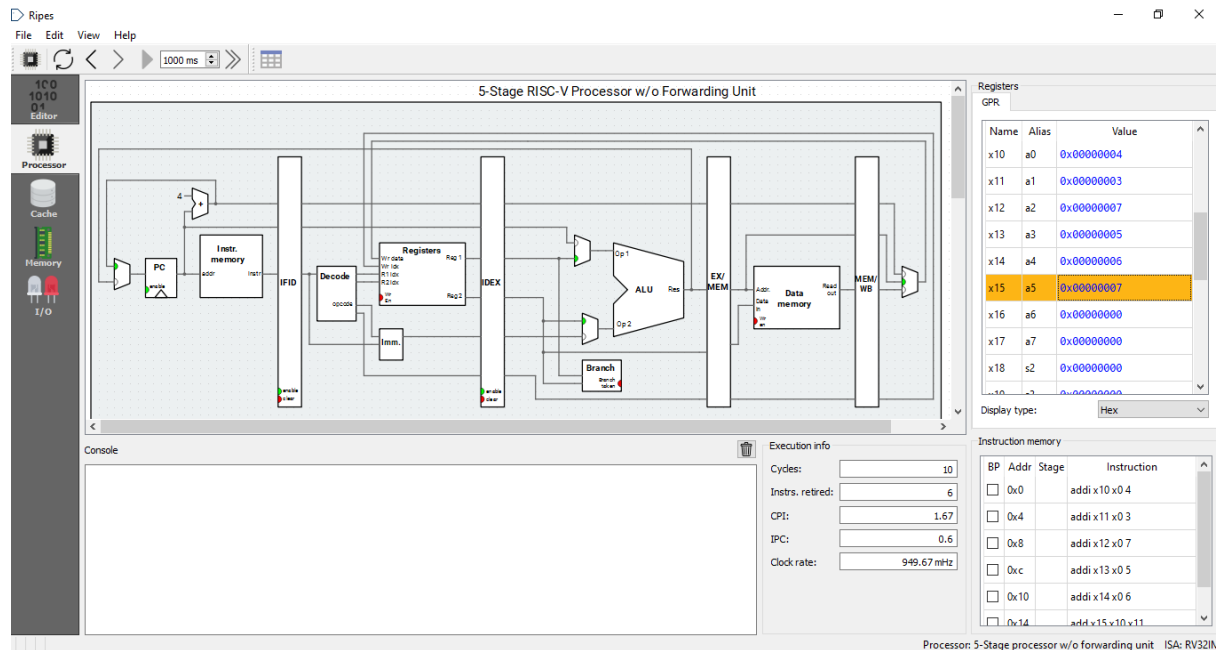
of recorded cycles can be changed in settings

	0	1	2	3	4
addi x10 x0 4	IF	ID	EX	MEM	WB
addi x11 x0 3		IF	ID	EX	MEM
addi x12 x0 7			IF	ID	EX
addi x13 x0 5				IF	ID
addi x14 x0 6					IF
add x15 x10 x11					









Aqui temos um código em assembly do RISC-V que ao implementá-lo com pipeline não teremos dependência de instruções. Foram usados 10 pulsos de clock para o código RISC-V com pipeline.

Como são 5 estágios de pipeline então:

Primeiro, é buscada a instrução na memória e é armazenada no buffer no primeiro pulso de clock.

No segundo pulso a instrução que foi buscada é decodificada e determinada seu tipo e operando enquanto é buscado outra instrução na memória e é armazenada no buffer.

No terceiro pulso de clock é feita a busca dos operandos na memória ou nos registradores da instrução que foi decodificada anteriormente, enquanto é decodificada a instrução que foi buscada anteriormente, é lido outra instrução na memória e é armazenada no buffer.

No quarto pulso de clock é executado a instrução que teve os operando, enquanto a instrução que foi codificada no pulso de clock anterior fará a busca dos operando na memória ou registradores, enquanto a instrução que foi buscado no estágio anterior fará a decodificação e ao mesmo tempo será lido outra instrução da da memória ou registradores.

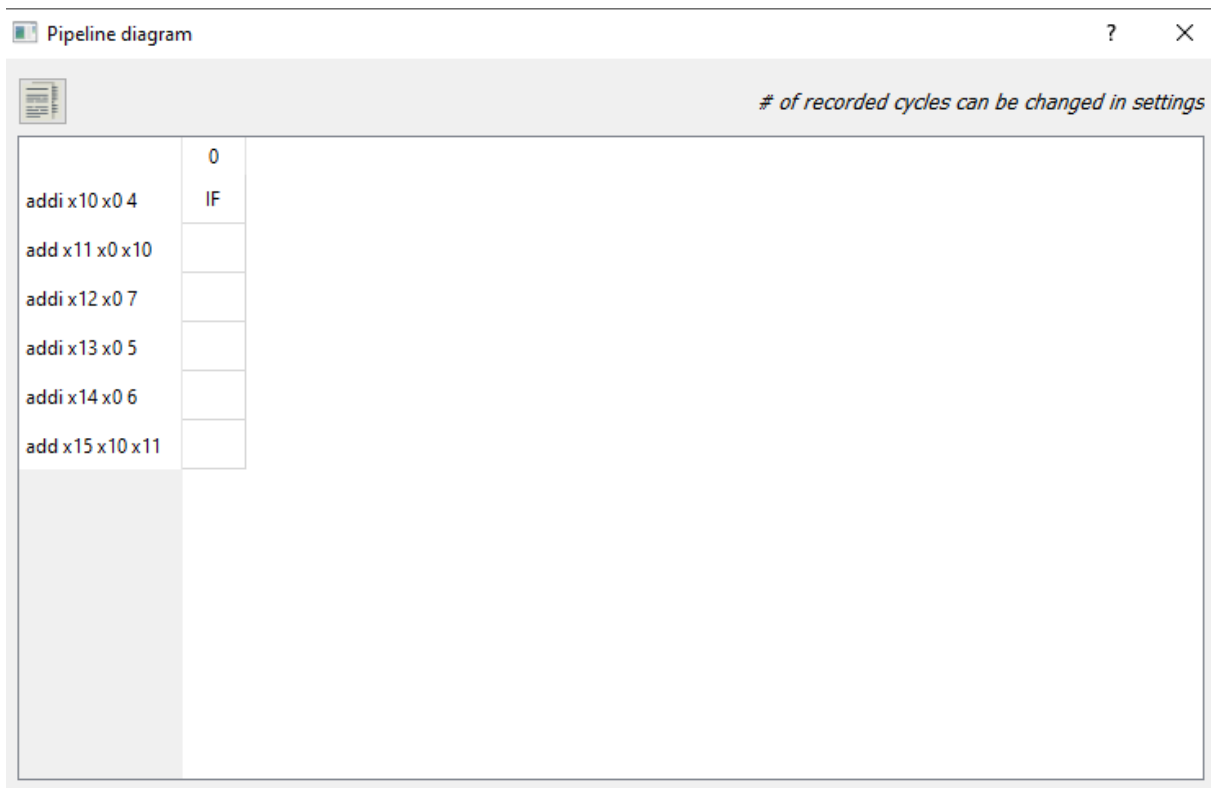
No quinto estágio a instrução que foi executada no pulso de clock anterior será armazenado no registrador, enquanto a instrução que possui os operando fará a execução da instrução, enquanto fará a busca dos operandos da instrução que foi decodificado, enquanto a instrução que foi buscado anteriormente será decodificado e será lido outra instrução da memória.

E assim em diante.

O Pipeline funciona como uma linha de produção onde não precisa esperar a instrução ser terminada para poder executar outra.

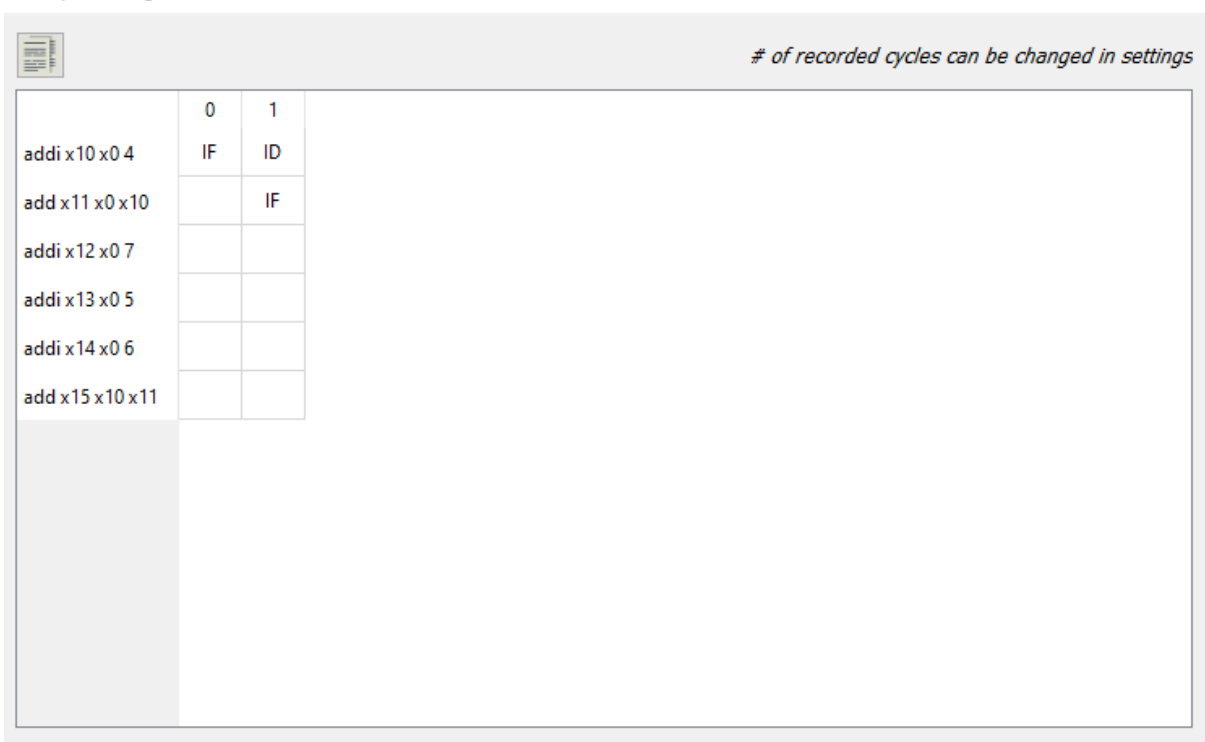
Código 2:

```
addi x10, x0, 4
add x11, x0, x10
addi x12, x0, 7
addi x13, x0, 5
addi x14, x0, 6
add x15, x10, x11
```



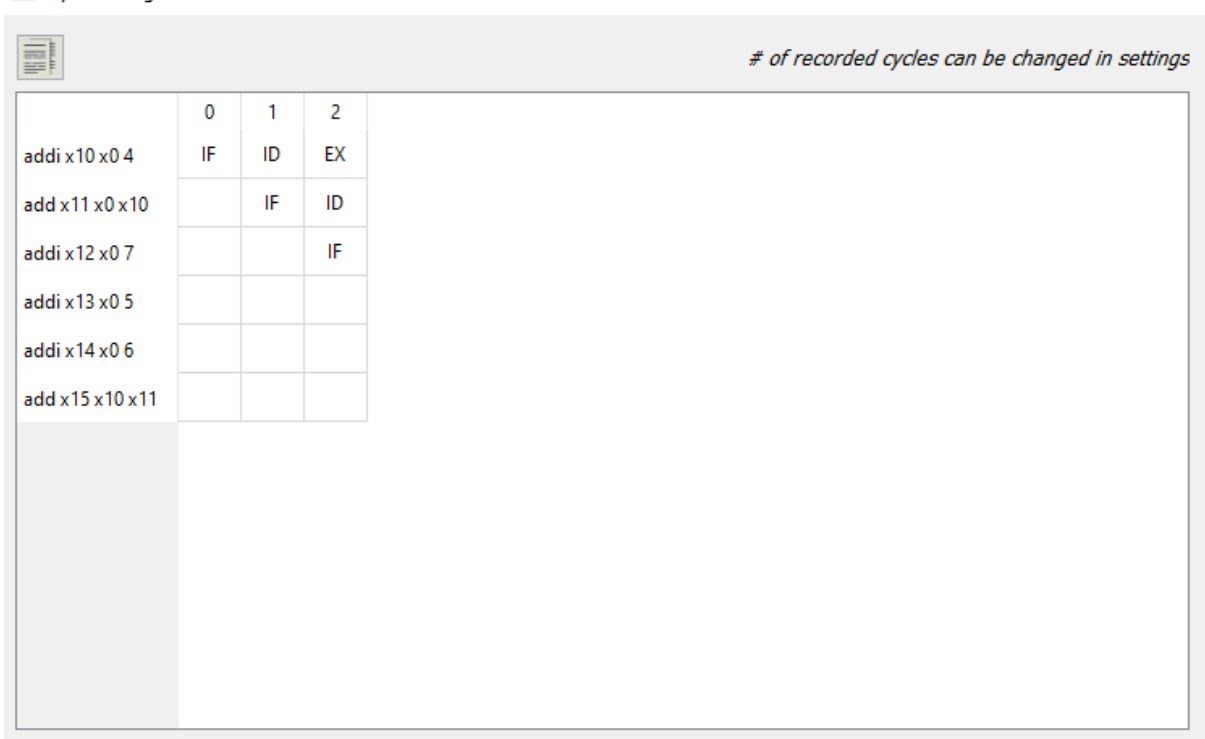
Pipeline diagram

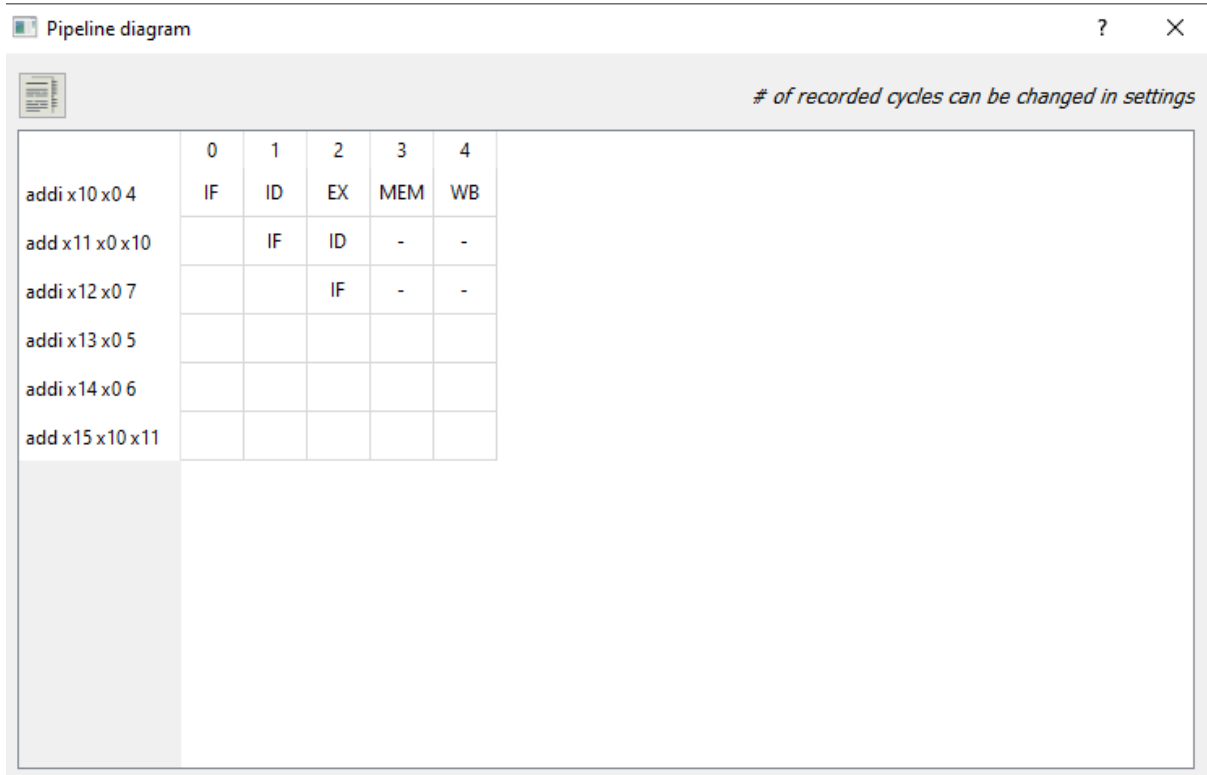
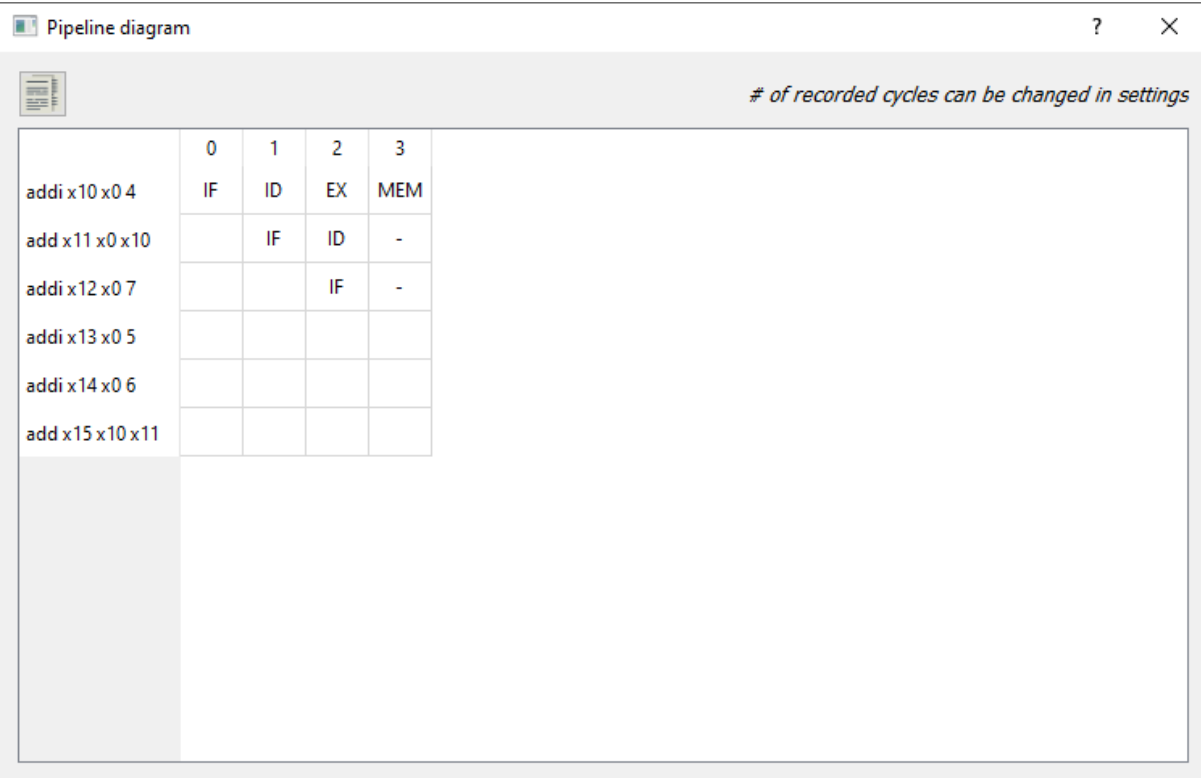
? ✕

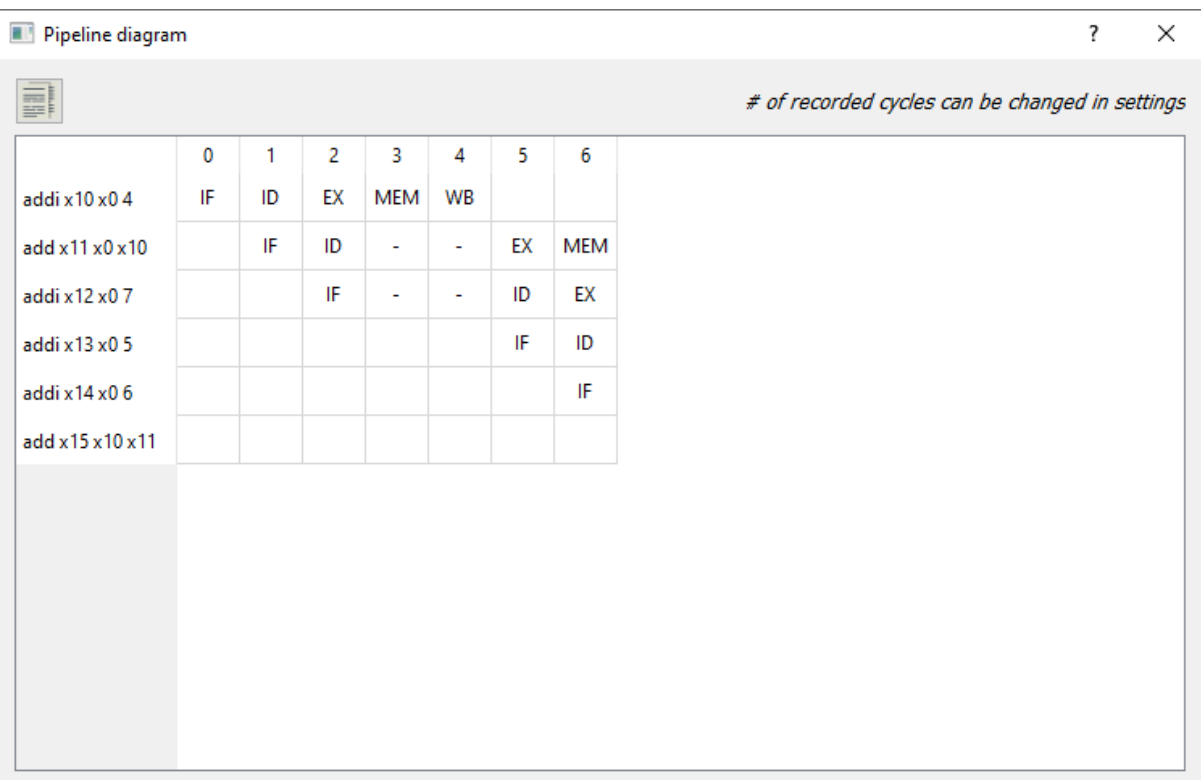
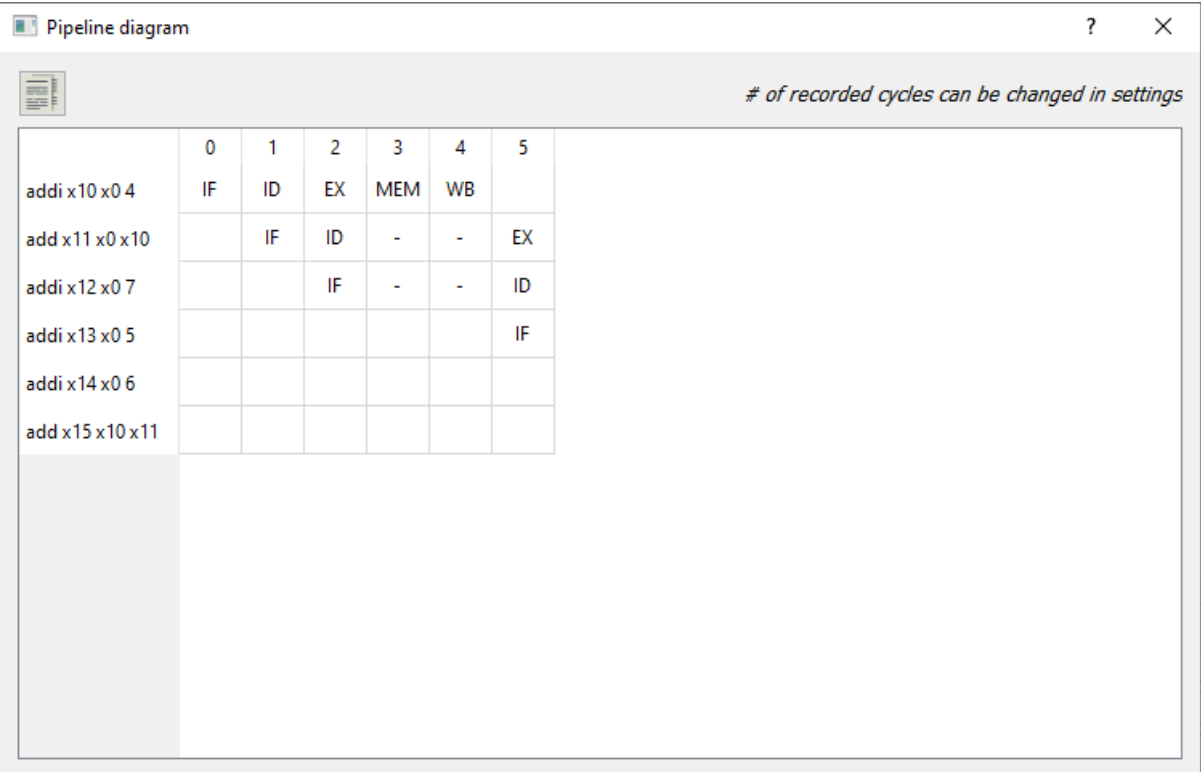


Pipeline diagram

? ✕







of recorded cycles can be changed in settings

of recorded cycles can be changed in settings



of recorded cycles can be changed in settings

	0	1	2	3	4	5	6	7	8	9
addi x10,x0,4	IF	ID	EX	MEM	WB					
add x11,x0,x10		IF	ID	-	-	EX	MEM	WB		
addi x12,x0,7			IF	-	-	ID	EX	MEM	WB	
addi x13,x0,5						IF	ID	EX	MEM	WB
addi x14,x0,6							IF	ID	EX	MEM
add x15,x10,x11								IF	ID	EX



of recorded cycles can be changed in settings

	0	1	2	3	4	5	6	7	8	9	10
addi x10,x0,4	IF	ID	EX	MEM	WB						
add x11,x0,x10		IF	ID	-	-	EX	MEM	WB			
addi x12,x0,7			IF	-	-	ID	EX	MEM	WB		
addi x13,x0,5						IF	ID	EX	MEM	WB	
addi x14,x0,6							IF	ID	EX	MEM	WB
add x15,x10,x11								IF	ID	EX	MEM

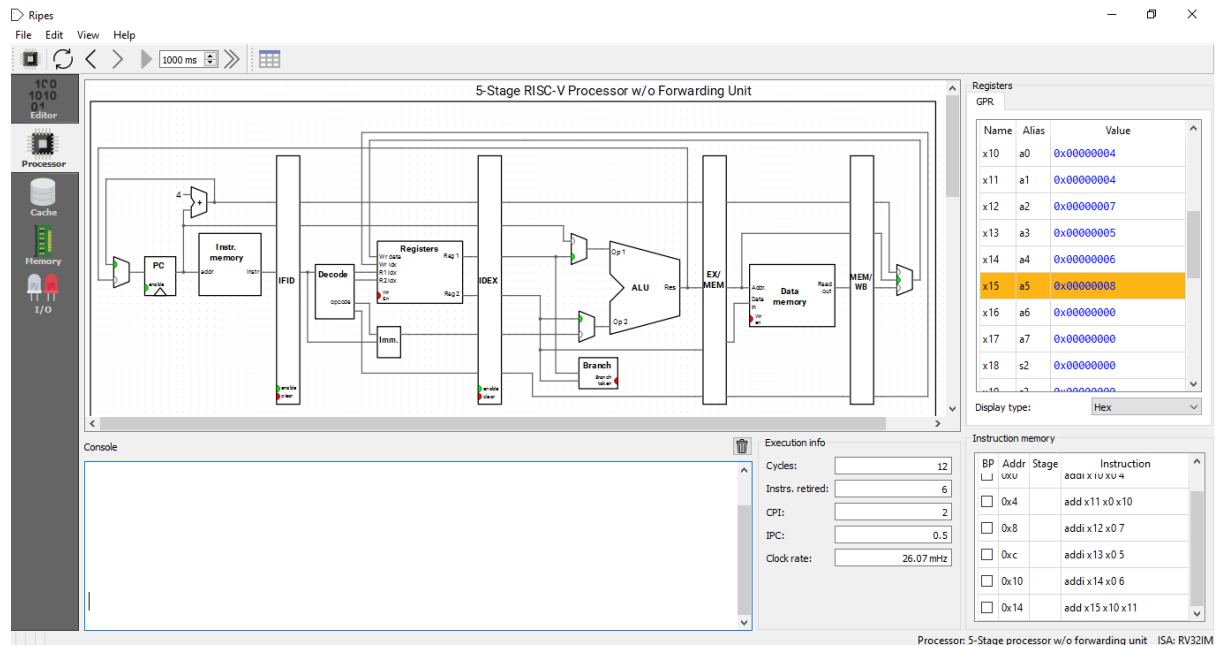


of recorded cycles can be changed in settings

[illegible]

of recorded cycles can be changed in settings

	0	1	2	3	4	5	6	7	8	9	10	11	12
addi x10,x0,4	IF	ID	EX	MEM	WB								
add x11,x0,x10		IF	ID	-	-	EX	MEM	WB					
addi x12,x0,7			IF	-	-	ID	EX	MEM	WB				
addi x13,x0,5						IF	ID	EX	MEM	WB			
addi x14,x0,6							IF	ID	EX	MEM	WB		
add x15,x10,x11								IF	ID	EX	MEM	WB	




Durante a execução do código foram necessários 12 pulsos de clock para a execução do código assembly do RISC-V. Diferentemente do código 1, no código 2 temos dependência de instruções o que não pode ocorrer no pipeline. Na segunda instrução é possível notar a dependência do registrador x10 para poder ser executado. Então, se cria uma bolha para esperar a execução das instruções dependentes para evitar propagação de erros.

Agora vamos utilizar o Pipeline com tratamento de erros.

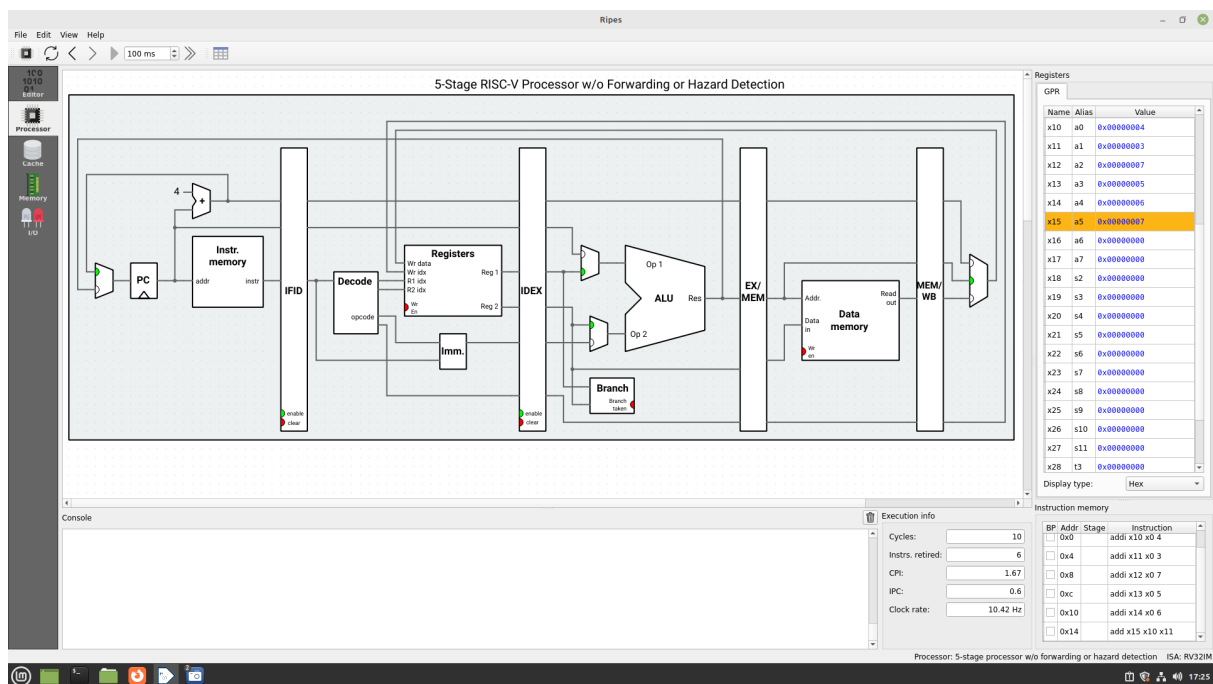
Para o código 1 o resultado foi o mesmo, 10 pulsos de clock e sem conflito sobre o uso de registradores.

Pipeline diagram

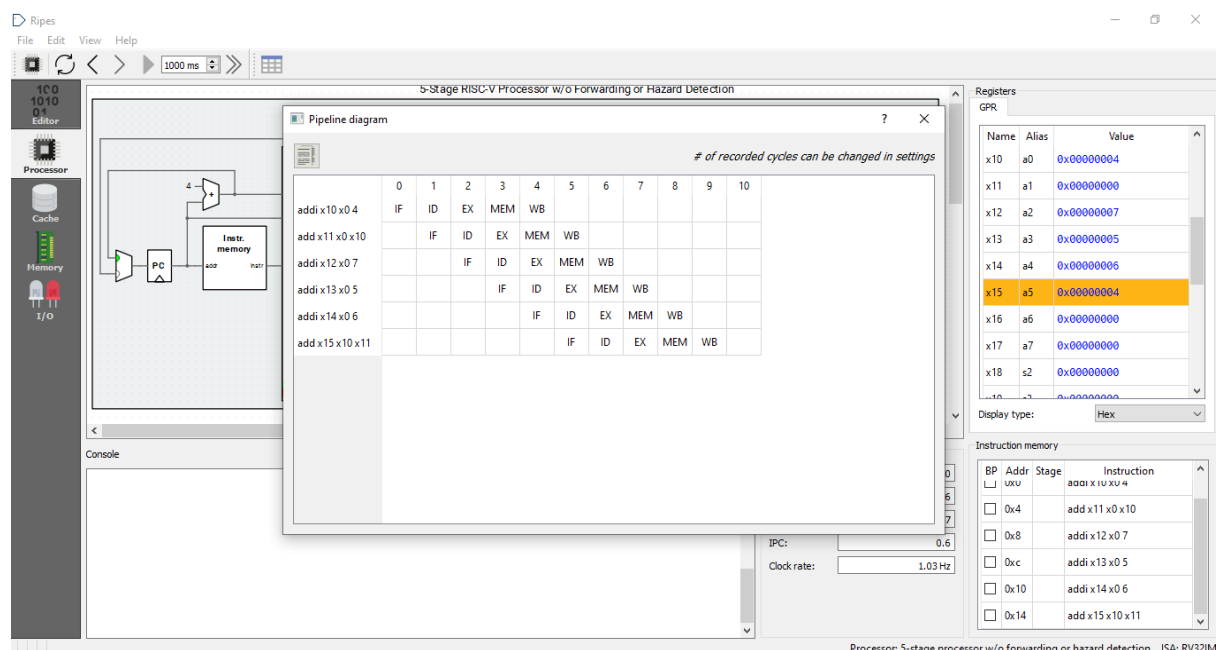
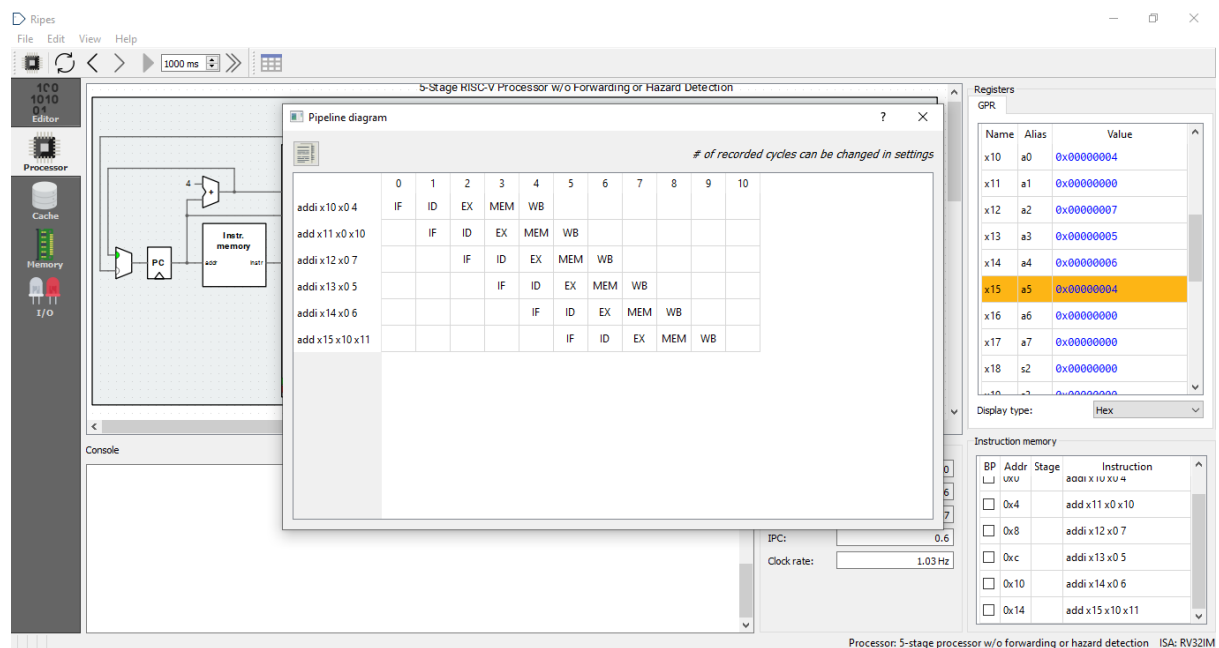


of recorded cycles can be changed in settings

	0	1	2	3	4	5	6	7	8	9	10
addi x10 x0 4	IF	ID	EX	MEM	WB						
addi x11 x0 3		IF	ID	EX	MEM	WB					
addi x12 x0 7			IF	ID	EX	MEM	WB				
addi x13 x0 5				IF	ID	EX	MEM	WB			
addi x14 x0 6					IF	ID	EX	MEM	WB		
add x15 x10 x11						IF	ID	EX	MEM	WB	



Já paga o código 2 o resultado foi o seguinte:



Foram necessários apenas 10 pulsos de clock na versão Pipeline sem propagação de erros, enquanto na versão com propagação de erros foram necessários 12 pulsos de clock para o nosso código 2. O que aconteceu foi que na versão sem propagação de erros, ele trata a código, ele executa outra instrução no lugar da instrução de depende de um registrador que ainda não tem um valor guardado. Então, ele troca a instrução que depende de algum registrador que ainda não tem valor por outra para que não haja a bolha criada no Pipeline que não trata propagação de erros.