Nome: Rafaelle

Atividade Pipeline

a)

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int i;
    int vetor[1000];
    for(i = 0; i < 1000; i+=2) {
        vetor[i] = 2;
        vetor[i+1] = 1;
    }
}
```

b)

```
main:                                   # @main
        addi    sp, sp, -2032
        sw      ra, 2028(sp)                    # 4-byte Folded Spill
        sw      s0, 2024(sp)                    # 4-byte Folded Spill
        addi    s0, sp, 2032
        addi    sp, sp, -2000
        li      a0, 0
        sw      a0, -16(s0)
        sw      a0, -20(s0)
        j       .LBB0_1
.LBB0_1:                                # =>This Inner Loop Header: Depth=1
        lw      a1, -20(s0)
        li      a0, 999
        blt     a0, a1, .LBB0_4
        j       .LBB0_2
.LBB0_2:                                #   in Loop: Header=BB0_1 Depth=1
        lw      a0, -20(s0)
        slli    a0, a0, 2
        lui     a1, 1048575
        addi    a1, a1, 76
        add     a1, a1, s0
        add     a2, a1, a0
        li      a0, 2
        sw      a0, 0(a2)
        lw      a0, -20(s0)
        slli    a0, a0, 2
        add     a1, a1, a0
        li      a0, 1
        sw      a0, 4(a1)
        j       .LBB0_3
.LBB0_3:                                #   in Loop: Header=BB0_1 Depth=1
        lw      a0, -20(s0)
        addi    a0, a0, 2
        sw      a0, -20(s0)
        j       .LBB0_1
.LBB0_4:
        lw      a0, -16(s0)
        addi    sp, sp, 2000
        lw      ra, 2028(sp)                    # 4-byte Folded Reload
        lw      s0, 2024(sp)                    # 4-byte Folded Reload
        addi    sp, sp, 2032
        ret
```

d) Avaliar a quantidade de stalls e flushes do programa utilizando diferentes

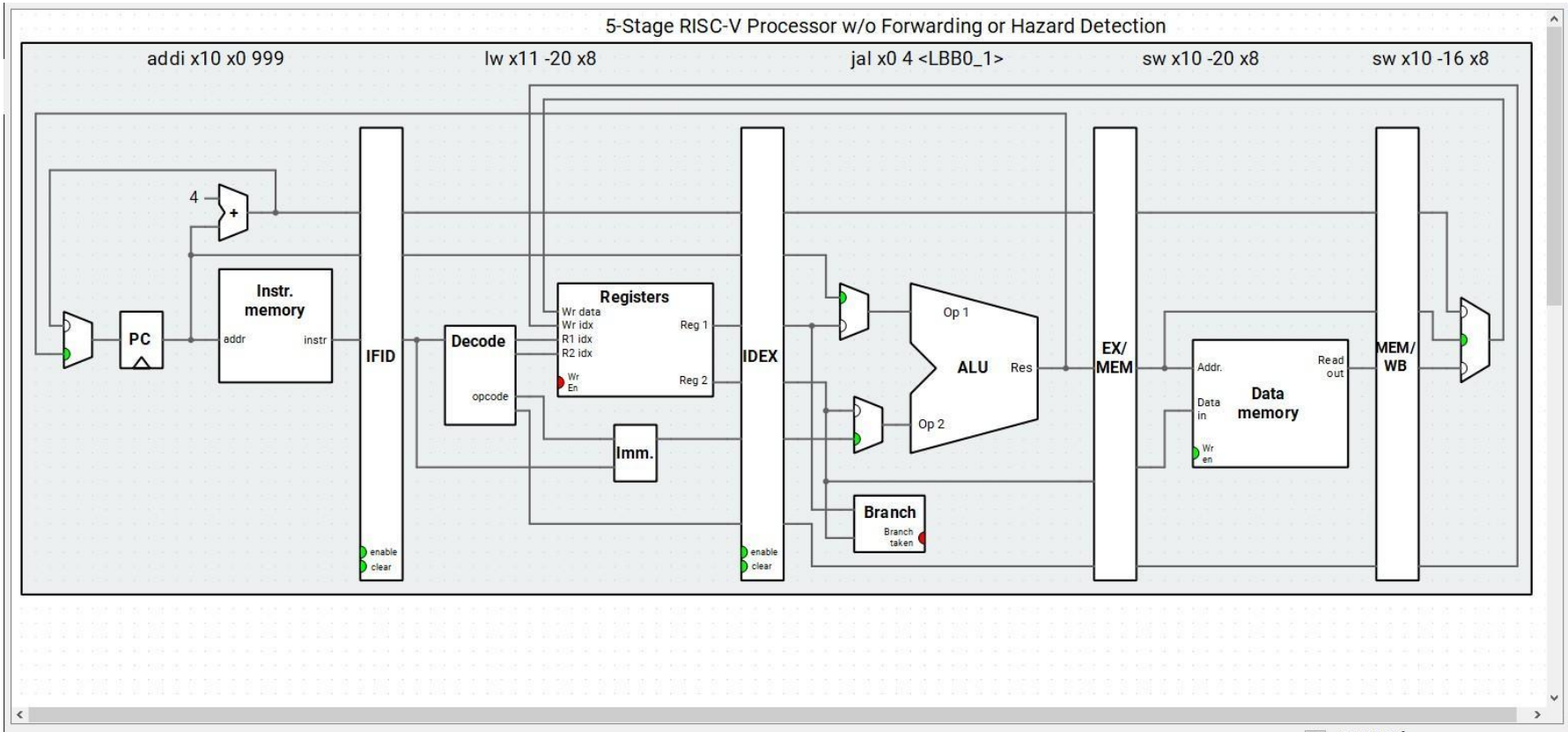
microarquiteturas: d1) pipeline sem detecção de hazard e sem adiantamento.

Resposta: Foi analisado 20 ciclos de clock do algoritmo em assembly, e foi constato que no ciclo 10 do processamento fica estável e sem conflitos de dados esperados. Mas quando vai para o próximo estado de ciclo 11,12,15,16,17,19,20 temos erro de processamento que não garante o resultado absoluto para o próximo ciclo de clock, as variáveis com problema são (x2, sp, 0x7FFFF030 e x10, a0 , 0x00003e7).

Podemos concluir que tivemos 8 stalls e flushes.

Como a instrução está num pipeline, ela caminha para o estágio EX, que precisa executar algo, mas não pode ser a instrução pois não tem os dados necessários, executa uma instrução que não tem efeito: nop e para inserir essa bolha no pipeline basta desativar todos os sinais de controle dos estágios EX, MEM e WB. Concluímos que, se deixar de compilar mais de vinte, iria em loop infinito este algoritmo.

Obs.: tem prints da tela onde foi realizada análise técnica, e mostra a quantidade de instruções executadas e quantos ciclos realizados.



Registers

GPR

Name	Alias	Value
x0	zero	0x00000000
x1	ra	0x00000000
x2	sp	0x7ffff030
x3	gp	0x10000000
x4	tp	0x00000000
x5	t0	0x00000000
x6	t1	0x00000000
x7	t2	0x00000000
x8	s0	0x7ffffff0
x9	s1	0x00000000
x10	a0	0x00000000
x11	a1	0x00000000
x12	a2	0x00000000
x13	a3	0x00000000
x14	a4	0x00000000
x15	a5	0x00000000
x16	a6	0x00000000
x17	a7	0x00000000
x18	s2	0x00000000
x19	s3	0x00000000

Display type: Hex

Console

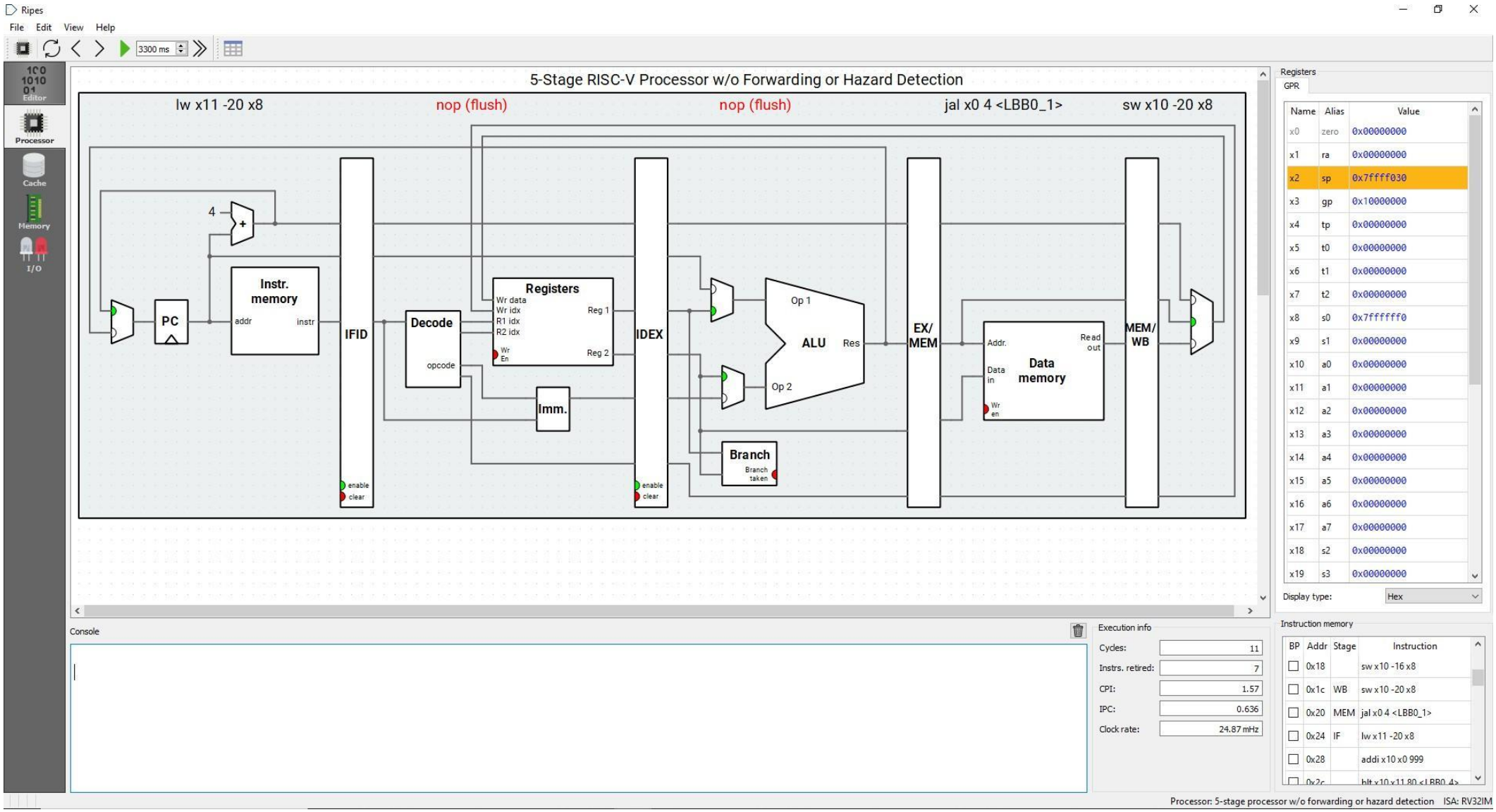
Execution info

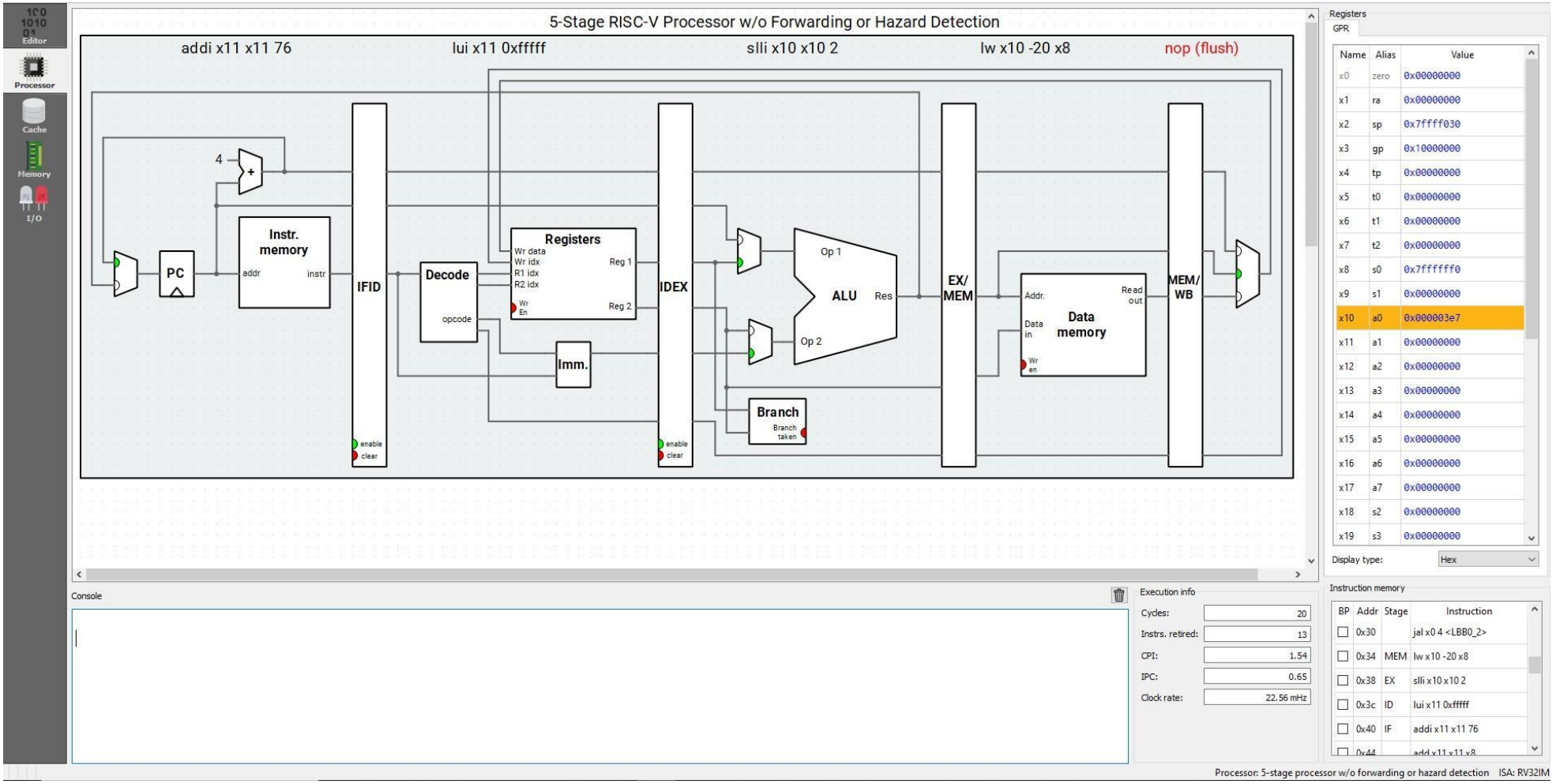
Cycles:	10
Instrs. retired:	6
CPI:	1.67
IPC:	0.6
Clock rate:	331.24 mHz

Instruction memory

BP	Addr	Stage	Instruction
☐	0x0		addi x2 x2 -2032
☐	0x4		sw x1 2028 x2
☐	0x8		sw x8 2024 x2
☐	0xc		addi x8 x2 2032
☐	0x10		addi x2 x2 -2000
☐	0x14		addi x10 x0 0

Processor: 5-stage processor w/o forwarding or hazard detection ISA: RV32


Ripes
File Edit View Help
3300 ms
Editor
Processor
Cache
Memory
I/O
5-Stage RISC-V Processor w/o Forwarding or Hazard Detection
lw x11 -20 x8
nop (flush)
nop (flush)
jal x0 4 <LBB0_1>
sw x10 -20 x8
4
+
PC
Instr. memory
addr
instr
IFID
enable
clear
Decode
opcode
Registers
Wr data
Wr idx
R1 idx
R2 idx
Wr En
Reg 1
Reg 2
Imm.
IDEX
enable
clear
Op 1
ALU
Res
Op 2
Branch
Branch taken
EX/ MEM
Addr.
Data in
Wr en
Data memory
Read out
MEM/ WB
Registers
GPR
Name Alias Value
x0 zero 0x00000000
x1 ra 0x00000000
x2 sp 0x7ffff030
x3 gp 0x10000000
x4 tp 0x00000000
x5 t0 0x00000000
x6 t1 0x00000000
x7 t2 0x00000000
x8 s0 0x7fffffff0
x9 s1 0x00000000
x10 a0 0x00000000
x11 a1 0x00000000
x12 a2 0x00000000
x13 a3 0x00000000
x14 a4 0x00000000
x15 a5 0x00000000
x16 a6 0x00000000
x17 a7 0x00000000
x18 s2 0x00000000
x19 s3 0x00000000
Display type: Hex
Console
Execution info
Cycles: 11
Instrs. retired: 7
CPI: 1.57
IPC: 0.636
Clock rate: 24.87 mHz
Instruction memory
BP Addr Stage Instruction
0x18 sw x10 -16 x8
0x1c WB sw x10 -20 x8
0x20 MEM jal x0 4 <LBB0_1>
0x24 IF lw x11 -20 x8
0x28 addi x10 x0 999
Processor: 5-stage processor w/o forwarding or hazard detection ISA: RV32IM


Editor
Processor
Cache
Memory
I/O
5-Stage RISC-V Processor w/o Forwarding or Hazard Detection
addi x11 x11 76
lui x11 0xfffff
slli x10 x10 2
lw x10 -20 x8
nop (flush)
4
+
PC
Instr. memory
addr
instr
IFID
enable
clear
Decode
opcode
Registers
Wr data
Wr idx
R1 idx
R2 idx
Wr En
Reg 1
Reg 2
Imm.
IDEX
enable
clear
Op 1
Op 2
ALU
Res
Branch
Branch taken
EX/ MEM
Addr.
Data in
Wr en
Data memory
Read out
MEM/ WB
Registers
GPR
Name
Alias
Value
x0 zero 0x00000000
x1 ra 0x00000000
x2 sp 0x7ffff030
x3 gp 0x10000000
x4 tp 0x00000000
x5 t0 0x00000000
x6 t1 0x00000000
x7 t2 0x00000000
x8 s0 0x7fffffc0
x9 s1 0x00000000
x10 a0 0x000003e7
x11 a1 0x00000000
x12 a2 0x00000000
x13 a3 0x00000000
x14 a4 0x00000000
x15 a5 0x00000000
x16 a6 0x00000000
x17 a7 0x00000000
x18 s2 0x00000000
x19 s3 0x00000000
Display type:
Hex
Console
Execution info
Cycles: 20
Instrs. retired: 13
CPI: 1.54
IPC: 0.65
Clock rate: 22.56 mHz
Instruction memory
BP
Addr
Stage
Instruction
0x30 jal x0 4 <LBB0_2>
0x34 MEM lw x10 -20 x8
0x38 EX slli x10 x10 2
0x3c ID lui x11 0xfffff
0x40 IF addi x11 x11 76
Processor: 5-stage processor w/o forwarding or hazard detection
ISA: RV32IM

e) Avaliar a performance

e1) apresentar todos os stalls ocorridos na execução esclarecendo a razão pelo qual o mesmo aconteceu

Resposta: Neste caso, como é detecção de hazard e sem adiantamento no ripes. Percebi que não tem efeito bolha e executa as instruções mesmo tendo problemas de dados erros, deixando estável e não resolvendo os conflitos que geram em cada estágio do processamento.

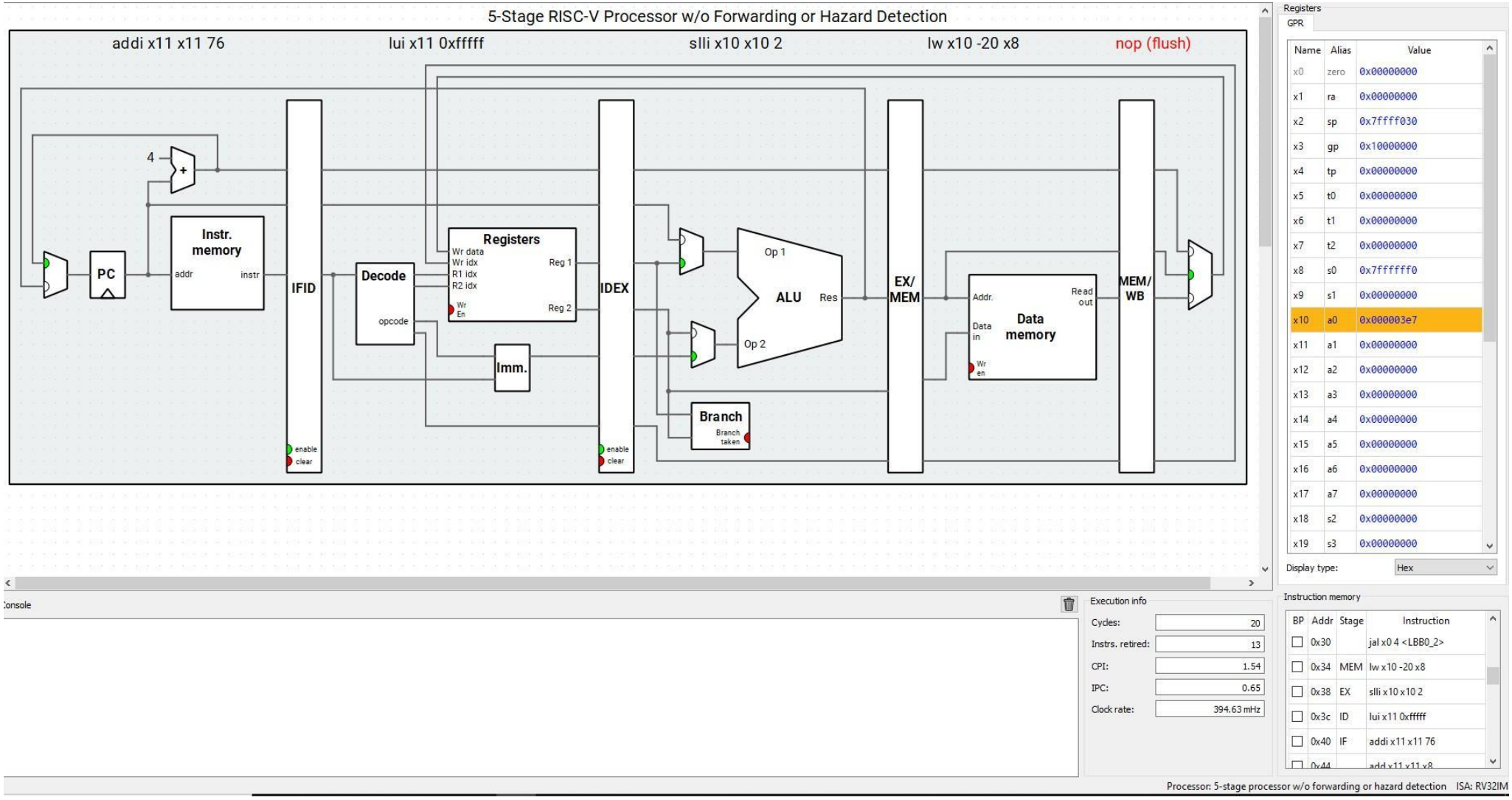
e2) apresentar todos os flushes ocorridos na execução esclarecendo a razão pelo qual o mesmo aconteceu.

Resposta: Foi realizados 20 ciclos de clock onde constatei 15 flushes ocorridos em cada execução de estágios no processador. Esse motivo ocorre, quando temos erros no próprio código onde não foi tratado de maneira eficaz e garantindo um resultado não absoluto para o próximo ciclo, tendo problema de dados e tanto estruturais. Recomenda-se que altere a sequência de instruções para não houve algum conflito. Mais uma observação, decidir iniciar com 20 ciclos no simulador, se deixasse o programa compilar por mim, irá até o infinito e teríamos N flush ocorrendo no processador.

e3) apresentar a quantidade total de stall e flushs presentes no código.

Resposta: não obteve nenhum stall, já em flush tivemos 15. (Realizado 20 ciclos de clock)

e4) apresentar o total de instruções executadas e a quantidade de ciclos de clock



Resposta: 42 instruções, foram 20 ciclos de clock realizada.

e5) apresentar a CPI média

Resposta: ciclos de clock /quantidade de instruções

Média-> 20/42 = 0,47

e6) alterar a sequência de instruções manualmente de forma a diminuir a quantidade de stalls e/ou flushes mantendo a execução correta do programa.

Resposta:

Segue o código com nop's inseridos no código.

```
function:                               # @function
        addi    sp, sp, -2032
        sw      ra, 2028(sp)                    # 4-byte Folded Spill
        sw      s0, 2024(sp)                    # 4-byte Folded Spill
        addi    s0, sp, 2032
        addi    sp, sp, -2000
        li      a0, 0
        sw      a0, -20(s0)
        j       LBB0_1
LBB0_1:                                 # =>This Inner Loop Header: Depth=1
        lw      a1, -20(s0)
        nop
        nop
        nop
        li      a0, 999
        nop
        nop
        blt     a0, a1, LBB0_7
```

```
        j       LBB0_2
LBB0_2:                                 #   in Loop: Header=BB0_1 Depth=1
        lw      a0, -20(s0)
        nop
        nop
        srli    a1, a0, 31
        nop
        nop
        add     a1, a1, a0
        nop
        nop
        andi    a1, a1, -2
        nop
        nop
        sub     a0, a0, a1
        li      a1, 0
        nop
        nop
        bne     a0, a1, LBB0_4
        j       LBB0_3
LBB0_3:                                 #   in Loop: Header=BB0_1 Depth=1
        lw      a0, -20(s0)
        nop
        nop
        slli    a1, a0, 2
        lui     a0, 1048575
        nop
        nop
        addi    a0, a0, 72
        nop
        nop
```

```
        add     a0, a0, s0
        nop
        nop
        add     a1, a1, a0
        nop
        nop
        li      a0, 2
        nop
        nop
        sw      a0, 0(a1)
        j       LBB0_5
LBB0_4:                                 #   in Loop: Header=BB0_1 Depth=1
        lw      a0, -20(s0)
        nop
        nop
        slli    a1, a0, 2
        lui     a0, 1048575
        nop
        nop
        addi    a0, a0, 72
        nop
        nop
        add     a0, a0, s0
        nop
        nop
        add     a1, a1, a0
        li      a0, 1
        nop
        nop
        sw      a0, 0(a1)
        j       LBB0_5
```

```
LBB0_5:                 #   in Loop: Header=BB0_1 Depth=1
        j       LBB0_6
LBB0_6:                 #   in Loop: Header=BB0_1 Depth=1
        lw      a0, -20(s0)
        nop
        nop
        addi    a0, a0, 1
        nop
        nop
        sw      a0, -20(s0)
        j       LBB0_1
LBB0_7:
        lw      a0, -16(s0)
        addi    sp, sp, 2000
        lw      ra, 2028(sp)            # 4-byte Folded Reload
        lw      s0, 2024(sp)            # 4-byte Folded Reload
        addi    sp, sp, 2032
```

e7) após ajuste indicar quais stalls e flushes foram suprimidos em relação aos apontados em e1 e e2.

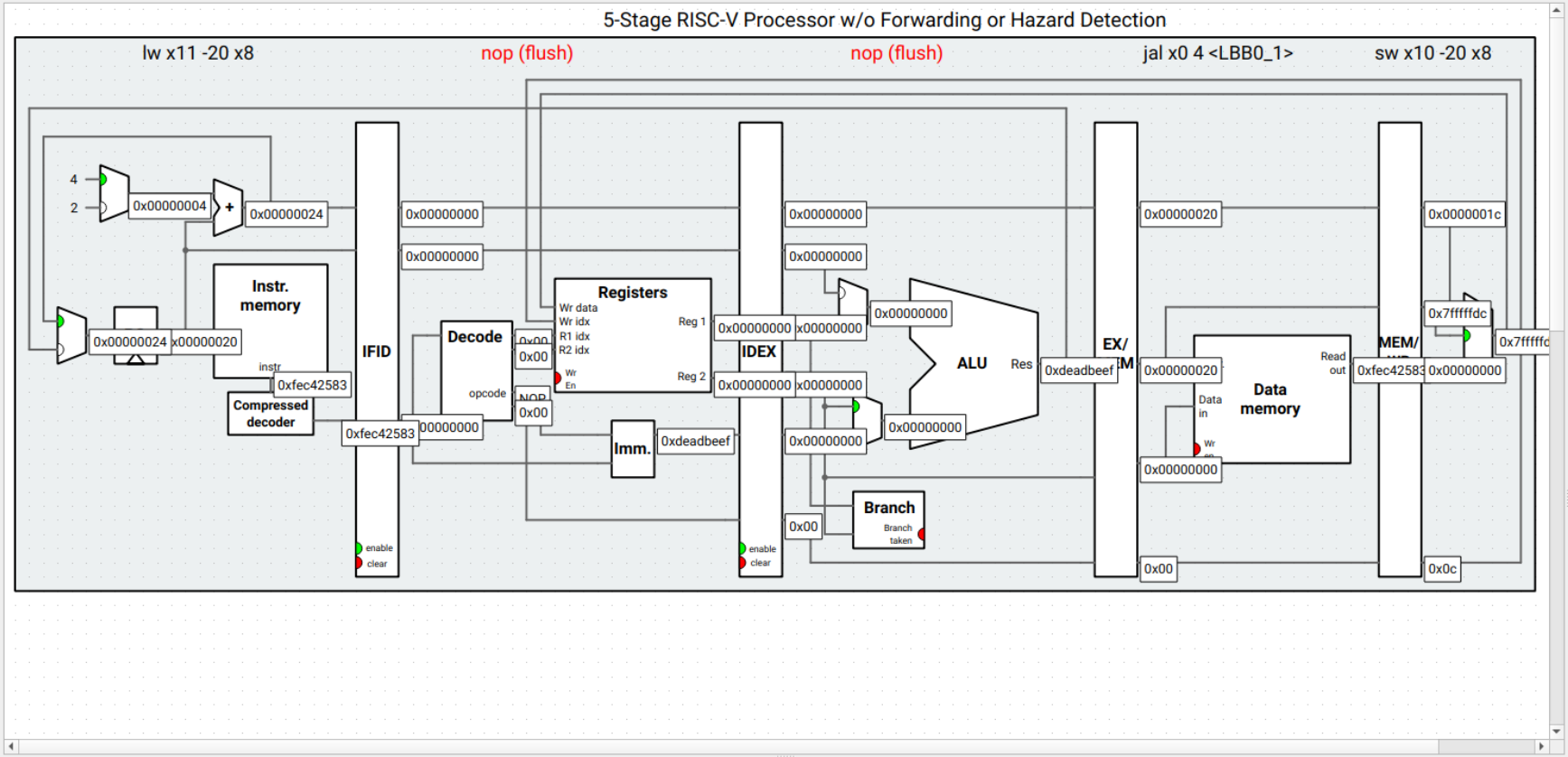
Resposta: Foi obtido 7 nops(flush) , com duração de 20 ciclos de clock. Abaixo tem um print aonde acontece a interrupção dos nops em passo a passo que foi realizado.

Com a manipulação no código fonte, observa que adicionando o nop para as instruções podemos melhorar o desempenho das instruções e evitando possíveis conflitos para que o resultado seja endereçado para o próximo registrador e assim efetuar seu processo de execução.

File Edit View Help

1 ms

Editor
Processor
Cache
Memory
I/O



Registers

GPR

Name	Alias	Value
x0	zero	0x00000000
x1	ra	0x00000000
x2	sp	0x7ffff030
x3	gp	0x10000000
x4	tp	0x00000000
x5	t0	0x00000000
x6	t1	0x00000000
x7	t2	0x00000000
x8	s0	0x7ffffff0
x9	s1	0x00000000
x10	a0	0x00000000
x11	a1	0x00000000
x12	a2	0x00000000
x13	a3	0x00000000
x14	a4	0x00000000
x15	a5	0x00000000
x16	a6	0x00000000
x17	a7	0x00000000
x18	s2	0x00000000

Display type: Hex

Instruction memory

BP	Addr	Stage	Instruction
	0x18	WB	sw x10 -20 x8
	0x1c	MEM	jal x0 4 <LBB0_1>
	0x20	IF	lw x11 -20 x8
	0x24		addi x0 x0 0
	0x28		addi x0 x0 0
	0x2c		addi x0 x0 0

Console

Execution info

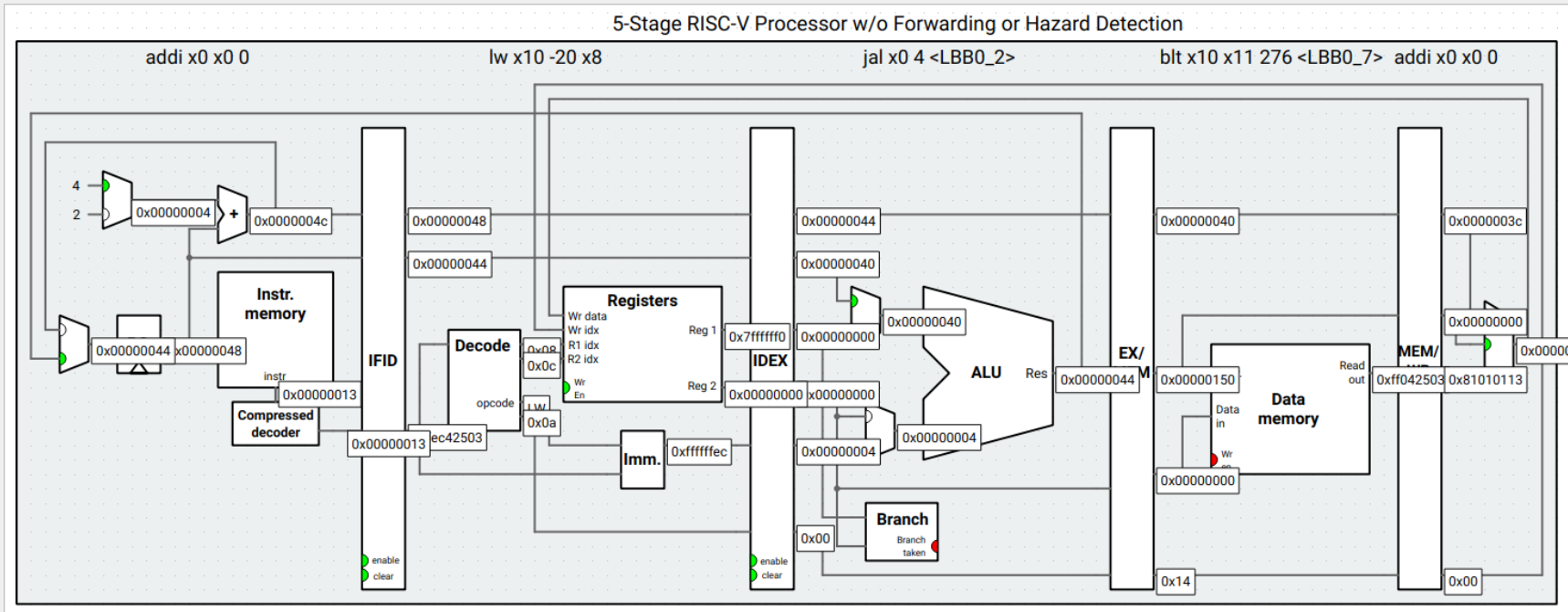
Cycles:	10
Instrs. retired:	6
CPI:	1.67
IPC:	0.6
Clock rate:	1.02 Hz

Processor: 5-stage processor w/o forwarding or hazard detection ISA: RV32IMC

File Edit View Help

1 ms

Editor
Processor
Cache
Memory
I/O



Console

Execution info

Cycles:	20
Instrs. retired:	14
CPI:	1.43
IPC:	0.7
Clock rate:	-60.29 mHz

Registers

GPR

Name	Alias	Value
x0	zero	0x00000000
x1	ra	0x00000000
x2	sp	0x7ffff030
x3	gp	0x10000000
x4	tp	0x00000000
x5	t0	0x00000000
x6	t1	0x00000000
x7	t2	0x00000000
x8	s0	0x7fffffff0
x9	s1	0x00000000
x10	a0	0x000003e7
x11	a1	0x00000000
x12	a2	0x00000000
x13	a3	0x00000000
x14	a4	0x00000000
x15	a5	0x00000000
x16	a6	0x00000000
x17	a7	0x00000000
x18	s2	0x00000000

Display type: Hex

Instruction memory

BP	Addr	Stage	Instruction
	0x3c	MEM	blt x10 x11 276 ...
	0x40	EX	jal x0 4 <LBB0_2>
	0x44	ID	lw x10 -20 x8
	0x48	IF	addi x0 x0 0
	0x4c		addi x0 x0 0
	0x50		srli x11 x10 31

Processor: 5-stage processor w/o forwarding or hazard detection ISA: RV32IM

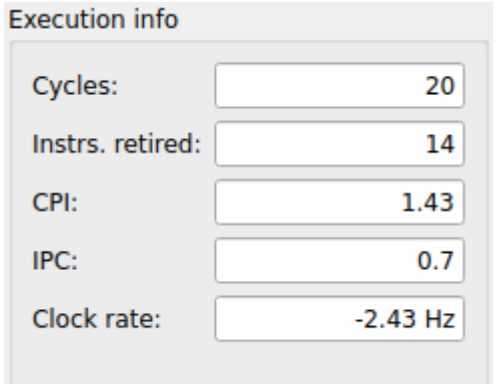
e8) apresentar a quantidade total de stall e flushs presentes após ajuste

Reposta: nenhum stall e 7 flushs.

e9) apresentar o total de instruções executadas e a quantidade de ciclos de clock após ajuste

Reposta:

97 instruções + 20 ciclos.



e10) apresentar a CPI média após ajuste.

Reposta: 20 cliclo de clock / número de instruçõe 97

Média = 20/97 =0.20

Pipeline com detecção de hazard e com adiantamento.

e1) apresentar todos os stalls ocorridos na execução esclarecendo a razão pelo qual o mesmo aconteceu

Resposta: Neste caso, como é com detecção de hazard e com adiantamento no ripes. Percebi que o programa detecta e esperar o registrador ficar disponível para inserir uma bolha, para evitar algum conflito de dados e adianta o dado para a instrução dependente para próximo ciclo de clock. Observa-se que, prever a necessidade do valor e executar de forma especulativa verificando se acertou a previsão ou não. Em passo a passo de compilação em 35 ciclos de clock, obtive 6 stalls em cada execução, se puder deixar compilar normal sem passos irei obter n stalls no programa.

e2) apresentar todos os flushes ocorridos na execução esclarecendo a razão pelo qual o mesmo aconteceu

Resposta: Obtive 16 flushes ocorridos, pelo fato de conflito de dados quando vai para o próximo ciclo de clock e o resultado não é o esperado pelo usuário.

e3) apresentar a quantidade total de stall e flushes presentes no código

Resposta: 16 flushes e 6 stall, foi feita uma compilação de passo a passo com 35 ciclos de clock, caso deixasse o software compilar, teríamos stall e flushes.

e4) apresentar o total de instruções executadas e a quantidade de ciclos de clock

Resposta: 45 instruções e 35 ciclos de clock realizado.

e5) apresentar a CPI média

Resposta: 35/45 =0,83

e6) alterar a sequência de instruções manualmente de forma a diminuir a quantidade de stalls e/ou flushes mantendo o execução correta do programa

Resposta:

Segue o código com nop's inseridos no código.

```
function:                               # @function
        addi    sp, sp, -2032
        sw      ra, 2028(sp)                    # 4-byte Folded Spill
        sw      s0, 2024(sp)                    # 4-byte Folded Spill
        addi    s0, sp, 2032
        addi    sp, sp, -2000
        li      a0, 0
        sw      a0, -20(s0)
        j       LBB0_1
LBB0_1:                                 # =>This Inner Loop Header: Depth=1
        lw      a1, -20(s0)
        nop
        nop
        nop
        li      a0, 999
        nop
        nop
        blt     a0, a1, LBB0_7
        j       LBB0_2
LBB0_2:                                 #   in Loop: Header=BB0_1 Depth=1
        lw      a0, -20(s0)
```

```
        nop
        nop
        srli    a1, a0, 31
        nop
        nop
        add     a1, a1, a0
        nop
        nop
        andi    a1, a1, -2
        nop
        nop
        sub     a0, a0, a1
        li      a1, 0
        nop
        nop
        bne     a0, a1, LBB0_4
        j       LBB0_3
LBB0_3:                                 #   in Loop: Header=BB0_1 Depth=1
        lw      a0, -20(s0)
        nop
        nop
        slli    a1, a0, 2
        lui     a0, 1048575
        nop
        nop
        addi    a0, a0, 72
        nop
        nop
        add     a0, a0, s0
        nop
        nop
```

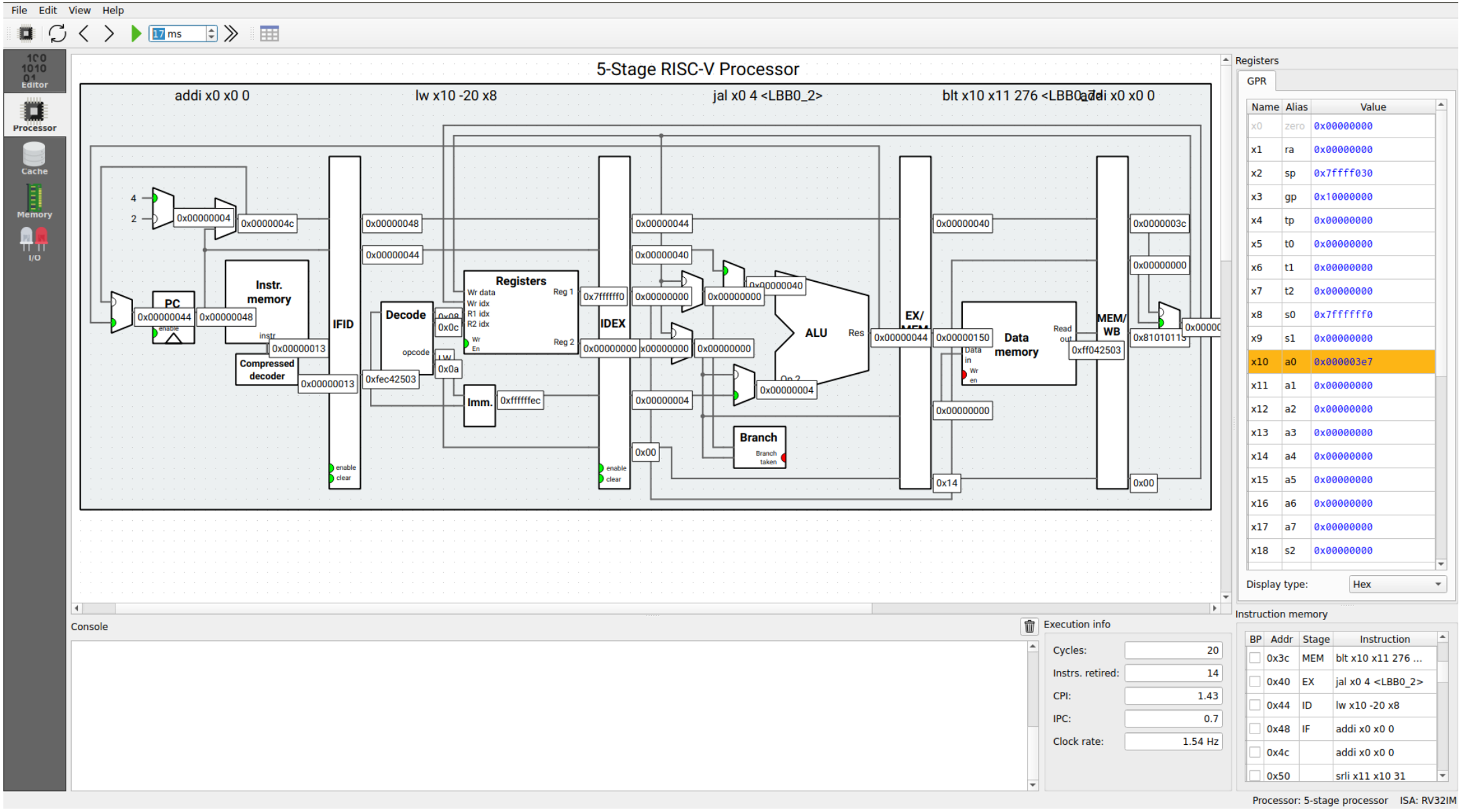
```
    add     a1, a1, a0
    nop
    nop
    li      a0, 2
    nop
    nop
    sw      a0, 0(a1)
    j       LBB0_5
LBB0_4:                 #   in Loop: Header=BB0_1 Depth=1
    lw      a0, -20(s0)
    nop
    nop
    slli    a1, a0, 2
    lui     a0, 1048575
    nop
    nop
    addi    a0, a0, 72
    nop
    nop
    add     a0, a0, s0
    nop
    nop
    add     a1, a1, a0
    li      a0, 1
    nop
    nop
    sw      a0, 0(a1)
    j       LBB0_5
LBB0_5:                 #   in Loop: Header=BB0_1 Depth=1
    j       LBB0_6
LBB0_6:                 #   in Loop: Header=BB0_1 Depth=1
```

```
        lw      a0, -20(s0)
        nop
        nop
        addi    a0, a0, 1
        nop
        nop
        sw      a0, -20(s0)
        j       LBB0_1
LBB0_7:
        lw      a0, -16(s0)
        addi    sp, sp, 2000
        lw      ra, 2028(sp)            # 4-byte Folded Reload
        lw      s0, 2024(sp)            # 4-byte Folded Reload
        addi    sp, sp, 2032
```

e7) após ajuste indicar quais stalls e flushes foram suprimidos em relação aos apontados em e1 e e2

Resposta: Foi obtido 7 nops(flush) , com duração de 20 ciclos de clock. Abaixo tem um print onde acontece a interrupção dos nops em passo a passo que foi realizado.

Com a manipulação no código fonte, observa que adicionando o nop para as instruções podemos melhorar o desempenho das instruções e evitando possíveis conflitos para que o resultado seja endereçado para o próximo registrador e assim efetuar seu processo de execução.


File Edit View Help
17 ms
Editor
Processor
Cache
Memory
I/O
5-Stage RISC-V Processor
addi x0 x0 0
lw x10 -20 x8
jal x0 4 <LBB0_2>
blt x10 x11 276 <LBB0_2>
addi x0 x0 0
PC
Instr. memory
Compressed decoder
IFID
Decode
Registers
IDEX
ALU
EX/MEM
Data memory
MEM/WB
Branch
Imm.
Console
Execution info
Cycles: 20
Instrs. retired: 14
CPI: 1.43
IPC: 0.7
Clock rate: 1.54 Hz
Registers
GPR
Name Alias Value
x0 zero 0x00000000
x1 ra 0x00000000
x2 sp 0x7ffff030
x3 gp 0x10000000
x4 tp 0x00000000
x5 t0 0x00000000
x6 t1 0x00000000
x7 t2 0x00000000
x8 s0 0x7fffffff0
x9 s1 0x00000000
x10 a0 0x000003e7
x11 a1 0x00000000
x12 a2 0x00000000
x13 a3 0x00000000
x14 a4 0x00000000
x15 a5 0x00000000
x16 a6 0x00000000
x17 a7 0x00000000
x18 s2 0x00000000
Display type: Hex
Instruction memory
BP Addr Stage Instruction
0x3c MEM blt x10 x11 276 ...
0x40 EX jal x0 4 <LBB0_2>
0x44 ID lw x10 -20 x8
0x48 IF addi x0 x0 0
0x4c addi x0 x0 0
0x50 srli x11 x10 31
Processor: 5-stage processor ISA: RV32IM

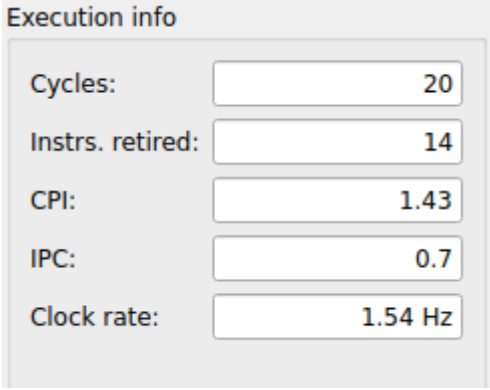
e8) apresentar a quantidade total de stall e flushs presentes após ajuste

Resposta: Foi obtido 7 nops(flush)

e9) apresentar o total de instruções executadas e a quantidade de ciclos de clock após ajuste

Resposta:

97 instruções + 20 ciclos.



e10) apresentar a CPI média após ajuste

Resposta: 20 cliclo de clock / número de instruçõe 97

Média = 20/97 =0.20