Laboratório 5 - CPU MIPS Pipeline

Grupo 6
Alunos:

- Caio Couto Moreira (15/0155433)
- Gabriel Alcantara (15/0126701)
- Breno Felipe (16/0003318)
- João Pedro Ramos do Amaral (14/0056394)

PARTE A: Apresentação do processador MIPS Pipeline

1)

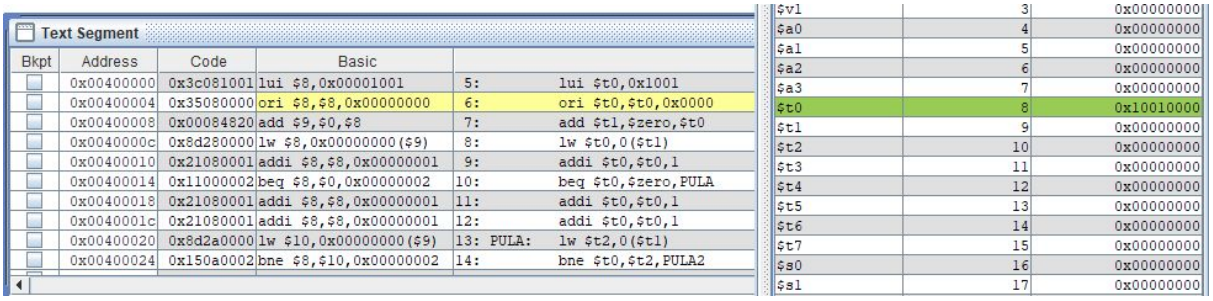
O seguinte código em assembly foi usado para teste dos processadores:

```
# Teste para verificacao da simulacao por forma de onda e na DE1-SoC
.data
        WORD: .word 0xB0DEB0B0, 0xF0CAF0FA, 0x00400058
.text
        lui $t0,0x1001
        ori $t0,$t0,0x0000
        add $t1,$zero,$t0
        lw $t0,0($t1)
        addi $t0,$t0,1
        beq $t0,$zero,PULA
        addi $t0,$t0,1
        addi $t0,$t0,1
PULA:   lw $t2,0($t1)
        bne $t0,$t2,PULA2
        addi $t0,$t1,-1
        addi $t0,$t0,-1
PULA2:  addi $t0,$t0,1
        jal PROC
        lw $t2,8($t1)
        jr $t2
VOLTA:  addi $t0,$t0,1
        j FIM1
        addi $t0,$t0,-1
        addi $t0,$t0,-1
PROC:   jr $ra
        addi $t0,$t0,-1
PROC1:  lw $t1,4($t1)
        div $t0,$t1
        mfhi $t1
        add $t0,$t0,$t1
        j VOLTA
        addi $t0,$t0,-1
FIM1:   j FIM1
        addi $t0,$t0,-1
```

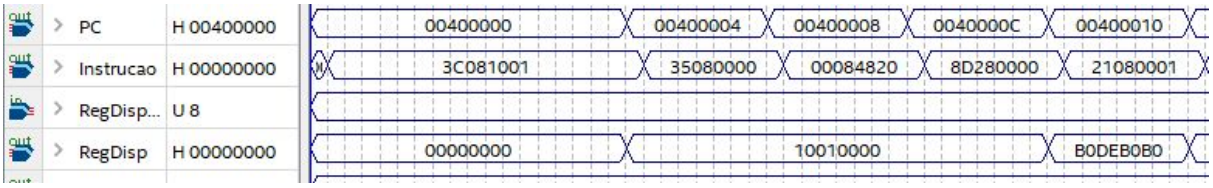
a) Uniciclo

Após a fazer compilação apenas da Análise Síntese do processador uniciclo, executamos a waveform2.vwf e obtemos o seguinte resultado:

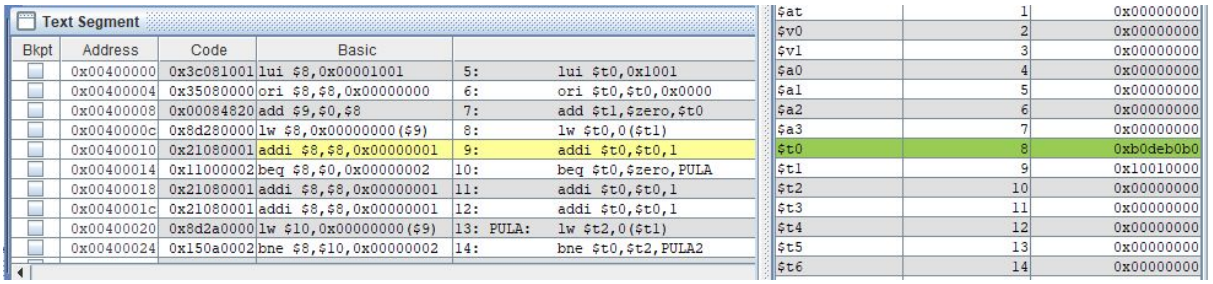
Na imagem acima vemos na coluna esquerda na linha do **RegDisp** e em seguida **U8** que nos diz qual registrador estaremos usando para imprimir seus valores, que nesse caso é o registrador 8, onde no mips esse registrador é o **$t0.** Podemos verificar a corretude do programa acompanhando a impressão dos valores de $t0 comparando diretamente no mars.



No mars verificamos que quando estamos no endereço 0x00400004 o valor de $t0 é 0x10010000.



E esse é o valor que $t0 mantém até o endereço 0x0040010 na waveform também. Quando estamos no endereço 0x00400010 podemos ver que o valor de $t0 agora é B0DEB0B0.

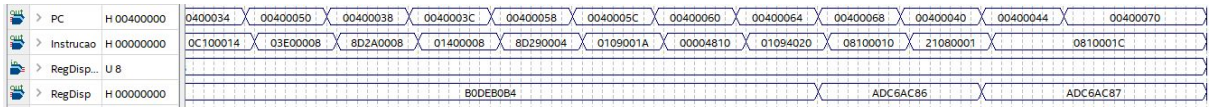


Text Segment

Bkpt	Address	Code	Basic		
	0x00400000	0x3c081001	lui $8,0x00001001	5:	lui $t0,0x1001
	0x00400004	0x35080000	ori $8,$8,0x00000000	6:	ori $t0,$t0,0x0000
	0x00400008	0x00084820	add $9,$0,$8	7:	add $t1,$zero,$t0
	0x0040000c	0x8d280000	lw $8,0x00000000($9)	8:	lw $t0,0($t1)
	0x00400010	0x21080001	addi $8,$8,0x00000001	9:	addi $t0,$t0,1
	0x00400014	0x11000002	beq $8,$0,0x00000002	10:	beq $t0,$zero,PULA
	0x00400018	0x21080001	addi $8,$8,0x00000001	11:	addi $t0,$t0,1
	0x0040001c	0x21080001	addi $8,$8,0x00000001	12:	addi $t0,$t0,1
	0x00400020	0x8d2a0000	lw $10,0x00000000($9)	13: PULA:	lw $t2,0($t1)
	0x00400024	0x150a0002	bne $8,$10,0x00000002	14:	bne $t0,$t2,PULA2

$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0xb0deb0b0
$t1	9	0x10010000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000

E é exatamente esse valor que vemos neste endereço em mars.
Fazemos isso durante toda a execução do programa e então podemos confirmar olhando os valores finais tanto na waveform quanto na execução do rars, o valor final de $t0.



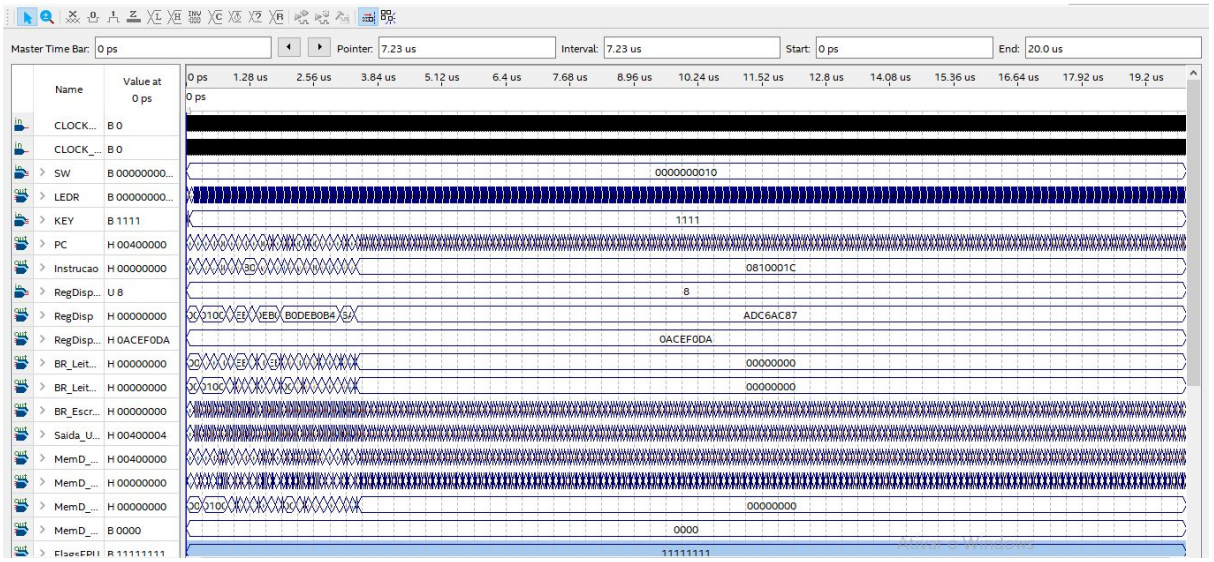
Bkpt	Address	Code	Basic	Source
	0x0040004c	0x2108ffff	addi $8,$8,0xffffffff	24: addi $t0,$t0,-1
	0x00400050	0x03e00008	jr $31	25: PROC: jr $ra
	0x00400054	0x2108ffff	addi $8,$8,0xffffffff	26: addi $t0,$t0,-1
	0x00400058	0x8d290004	lw $9,0x00000004($9)	27: PROC1: lw $t1,4($t1)
	0x0040005c	0x0109001a	div $8,$9	28: div $t0,$t1
	0x00400060	0x00004810	mfhi $9	29: mfhi $t1
	0x00400064	0x01094020	add $8,$8,$9	30: add $t0,$t0,$t1
	0x00400068	0x08100010	j 0x00400040	31: j VOLTA
	0x0040006c	0x2108ffff	addi $8,$8,0xffffffff	32: addi $t0,$t0,-1
	0x00400070	0x0810001c	j 0x00400070	33: FIM1: j FIM1

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0xadc6ac87
$t1	9	0xfce7fbd2
$t2	10	0x00400058
$t3	11	0x00000000

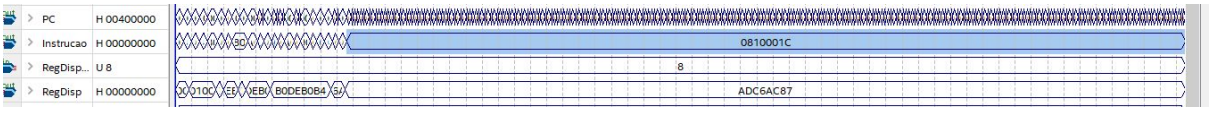
Verificamos então que nosso programa **testeSIMPLESPIPE.s** está funcionando corretamente no processador uniciclo.

Verificando manualmente na fpga a frequência máxima com a qual nosso programa ainda é executado corretamente no uniciclo é quando o divisor de frequência é dividido por 2.

b) Multiciclo:
Após a fazer compilação apenas da Análise Síntese do processador multiciclo, executamos a waveform2.vwf e obtemos o seguinte resultado:



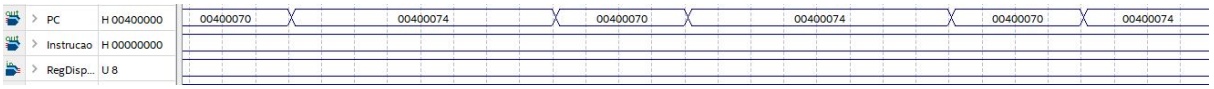
Seguindo a mesma ideia de teste feito no item anterior podemos ver que pela waveform gerada, o pc está indo até o endereço 0x00400074, o qual já é um ponto bom uma vez que no mars o programa está terminando neste endereço.



Verificando valor final do registrador $t0, temos que o resultado é ADC6AC87, o qual é o valor final no programa **testeSIMPLESPIPE.v** no mars.

Bkpt	Address	Code	Basic	Source
	0x0040004c	0x2108ffff	addi $8,$8,0xffffffff	24: addi $t0,$t0,-1
	0x00400050	0x03e00008	jr $31	25: PROC: jr $ra
	0x00400054	0x2108ffff	addi $8,$8,0xffffffff	26: addi $t0,$t0,-1
	0x00400058	0x8d290004	lw $9,0x00000004($9)	27: PROC1: lw $t1,4($t1)
	0x0040005c	0x0109001a	div $8,$9	28: div $t0,$t1
	0x00400060	0x00004810	mfhi $9	29: mfhi $t1
	0x00400064	0x01094020	add $8,$8,$9	30: add $t0,$t0,$t1
	0x00400068	0x08100010	j 0x00400040	31: j VOLTA
	0x0040006c	0x2108ffff	addi $8,$8,0xffffffff	32: addi $t0,$t0,-1
	0x00400070	0x0810001c	j 0x00400070	33: FIM1: j FIM1

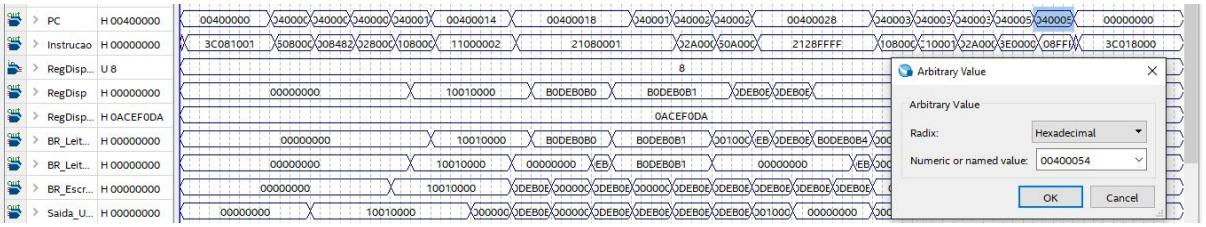
Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0xadc6ac87
$t1	9	0xfce7fbd2
$t2	10	0x00400058
$t3	11	0x00000000



Verificando manualmente na fpga a frequência máxima com a qual nosso programa ainda é executado corretamente no multiciclo é quando o divisor de frequência é dividido por 3.

c) Pipeline:

Após a fazer compilação apenas da Análise Síntese, executamos a waveform2.vwf e obtemos o seguinte resultado:



Com o resultado da waveform vamos fazer a comparação para vê se o processador está executando as instruções de acordo com o programa **testeSIMPLESPIPE.s**

Originalmente o programa é executado no mars do endereço 0x00400000 até 0x00400074, como pode ser visto na imagem abaixo:

Address	Code
0x00400000	0x3c081001
0x00400004	0x35080000
0x00400008	0x00084820
0x0040000c	0x8d280000
0x00400010	0x21080001
0x00400014	0x11000002
0x00400018	0x21080001
0x0040001c	0x21080001
0x00400020	0x8d2a0000
0x00400024	0x150a0002
0x00400028	0x2128ffff
0x0040002c	0x2108ffff
0x00400030	0x21080001
0x00400034	0x0c100014
0x00400038	0x8d2a0008
0x0040003c	0x01400008
0x00400040	0x21080001
0x00400044	0x0810001c
0x00400048	0x2108ffff
0x0040004c	0x2108ffff
0x00400050	0x03e00008
0x00400054	0x2108ffff
0x00400058	0x8d290004
0x0040005c	0x0109001a
0x00400060	0x00004810
0x00400064	0x01094020
0x00400068	0x08100010
0x0040006c	0x2108ffff
0x00400070	0x0810001c
0x00400074	0x2108ffff

No resultado da nossa waveform o programa está indo do endereço 0x00400000 até 0x00400054, nesse caso temos que nosso programa não executando por completo, apesar do registrador $t0, que está sendo usado para debugger está

funcionando corretamente até a instrução 0x00400054. Esse erro acontece pois no programa em assembly contém hazard de instruções.

d) Compare os requerimentos físicos:

i) Número de Elementos Lógicos
ii) Número de Registradores
iii) Quantidade de bits de memória
iv) Número de blocos DSP

Uniciclo:

Total registers	4432
Total pins	305
Total virtual pins	347
Total block memory bits	934,912
Total DSP Blocks	18

Multiciclo:

Total registers	4641
Total pins	273
Total virtual pins	379
Total block memory bits	934,912
Total DSP Blocks	18

Pipeline:

Total registers	4589
Total pins	305
Total virtual pins	347
Total block memory bits	934,912
Total DSP Blocks	18

Compare os requerimentos temporais:
i) caminho de maior atraso tpd
ii) caminhos com piores tempos th, tco, tsu e slacks
iii) máxima frequência de clock obtida pelo TimeQuest

Uniciclo:

Minimum Propagation Delay

	Input Port	Output Port	RR	RF	FR	FF
1	SW[7]	VGA_G[1]	16.468	18.622	16.796	18.950

Slow 1100mV 85C Model

	Clock	Slack	End Point TNS
1	altera_reserved_tck	9.229	0.000

Clock to Output Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	altera_reserved_tdo	altera_reserved_tck	5.883	5.846	Fall	altera_reserved_tck

Setup Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	altera_reserved_tms	altera...ed_tck	3.085	3.815	Rise	altera_reserved_tck

Hold Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	altera_reserved_tdi	altera_reserved_tck	0.762	0.614	Rise	altera_reserved_tck

Slow 1100mV 85C Model

	Fmax	Restricted Fmax	Clock Name	Note
1	67.23 MHz	67.23 MHz	altera_...ved_tck	

Multiciclo:

Propagation Delay

	Input Port	Output Port	RR	RF	FR	FF
1	KEY[0]	MemD_...a[28]	27.695	30.123	27.152	29.531

Slow 1100mV 85C Model

	Clock	Slack	End Point TNS
1	altera_reserved_tck	10.088	0.000

Clock to Output Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	MemD_Da...tura[*]	altera_reserved_tck	16.694	19.488	Rise	altera_reserved_tck

Setup Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	altera_reserved_tms	altera...ed_tck	3.019	3.553	Rise	altera_reserved_tck

Hold Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	altera_reserved_tdi	altera_reserved_tck	0.762	0.614	Rise	altera_reserved_tck

Slow 1100mV 85C Model

	Fmax	Restricted Fmax	Clock Name	Note
1	76.01 MHz	76.01 MHz	altera_...ved_tck	

Pipeline:

Propagation Delay

	Input Port	Output Port	RR	RF	FR	FF
1	SW[7]	VGA_G[2]	23.775	25.817	23.543	25.585

Slow 1100mV 85C Model

	Clock	Slack	End Point TNS
1	altera_reserved_tck	10.542	0.000

Clock to Output Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	altera_reserved_tdo	altera_reserved_tck	5.883	5.846	Fall	altera_reserved_tck

Setup Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	altera_reserved_tms	altera...ed_tck	2.981	3.737	Rise	altera_reserved_tck

Hold Times

	Data Port	Clock Port	Rise	Fall	Clock Edge	Clock Reference
1	altera_reserved_tdi	altera_reserved_tck	0.762	0.614	Rise	altera_reserved_tck

Slow 1100mV 85C Model

	Fmax	Restricted Fmax	Clock Name	Note
1	81.64 MHz	81.64 MHz	altera_...ved_tck	

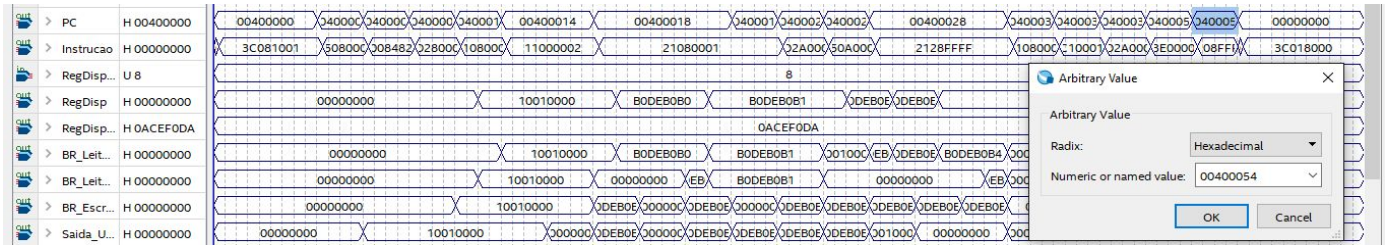
PARTE B: Análise do processador MIPS Pipeline

2 - a) Abaixo será descrito os resultados intermediários esperados da execução do programa testeSIMPLESPIPE.s no mars (obs: antes da correção do programa):

- $t0 recebe o valor do endereço do dado WORD. linha:5
- $t1 recebe esse valor de $t0 linha:7

- $t0 salva o primeiro valor(0xb0deb0b0) de WORD linha:8
- não entra no beq linha:10
- soma 3 em $t0
- $t2 salva o primeiro valor(0xb0deb0b0) de WORD linha:13
- compara $t2 e $t0 e entra no label(PULA2) no bne linha: 14
- entra no jal PROC linha:18, e em PROC já retorna pra linha abaixo que chamou jal PROC.
- salva o terceiro valor da WORD(0x00400058) em $t2
- pula para o endereço salvo em $t2
- divide $t0 e $t1 salva o high em $t1
- soma $t0 e $t1
- pula pra label VOLTA e em seguida pula pra label FIM1 e acaba a execução do programa.

b)



Pela fórmula de onda nosso programa executou somente até a instrução 0x00400054, observando suas instruções anteriores ele para na parte em que entra no jal PROC linha:18, e em PROC já retorna para linha abaixo que chamou jal PROC.

c) Filmagem do funcionamento na de1-soc:
Execução passo a passo:
https://www.youtube.com/watch?v=rtxGXtRcaDs

Execução rápida:
https://www.youtube.com/watch?v=UOnJ37GZpjs

d) Corrigindo instruções com bolhas que estão causando erros:

A instruções corrigidas foram as seguintes:

```
PROC:   nop
        nop
        nop
        jr $ra
```

Corrigimos com 3 nop, a chamada da instrução jr $ra, pois a anterior a ela é a instrução jal PROC, pois quando a instrução faz o jal PROC ainda não foi possível pegar do banco de registradores o valor correto do registrador $r0.

```
PULA2:  addi $t0,$t0,1
        jal PROC
        lw $t2,8($t1)
        nop
        nop
        jr $t2
```

Corrigimos com 2 nop a chamada do jr $t2, pois o lw causa também o hazard.

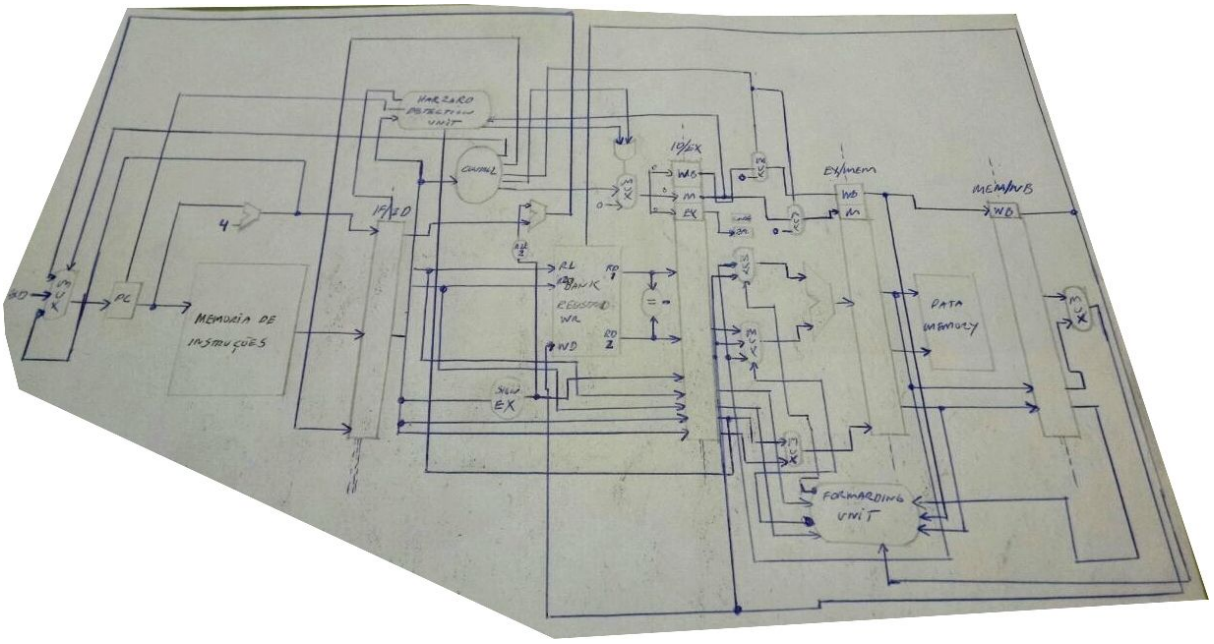
```
.data
        WORD: .word 0xB0DEB0B0, 0xF0CAF0FA, 0x00400068
```

Feito essas modificações também foi necessário alterar o terceiro valor “salvo” em WORD, pois uma vez que adicionamos nop no nosso código, os endereços das instruções seguintes são alteradas, nesse caso devemos alterar esse valor para próxima instrução que temos que voltar.

E com isso conseguimos o resultado correto, como demonstrado no vídeo abaixo: **https://www.youtube.com/watch?v=rtxGXtRcaDs**

3 -

Caminho de dados:



Sinais de controle de dados:

Instrução	Escreve Reg	Mem para Reg
Formato R	1	0
lw	1	1
sw	0	X
beq	0	X

Instrução	RegDst	OpALU1	OpALU0	OrigALU
Formato R	1	1	0	0
lw	0	0	0	1
sw	X	0	0	1
beq	X	0	1	0

Instrução	Branch	LeMem	Escreve Mem
Formato R	0	0	0
lw	0	1	0
sw	0	0	1
beq	1	0	0

4 -

Hazard: Verifica a situação em que próxima instrução não pode ser executada no ciclo de clock seguinte.

No hazard de dados, o pipeline precisa ser interrompido porque os dados para executar a instrução ainda não estão disponíveis. Ex:

```
1  add $s0, $t0, $t1
2  sub $t2, $s0, $t3
```

Ele pode ser tratado com:

- Forwarding: ele corta caminho através do pipeline
- Reordenação do código
- Inserção de bolhas, então insere-se uma bolha no pipeline até que seja computado se o desvio vai ser tomado ou não

No hazard de controle acontece quando o processador precisa fazer um desvio de fluxo, quando precisamos alterar o PC(Program Counter). Ex:

```
bne $t1, $t2, algumLugar
```

Ele pode ser tratado com:

- inserção de bolha, então insere-se uma bolha no pipeline até que seja computado se o desvio vai ser tomado ou não

- nunca tomar desvio
- sempre tomar desvio, já começa a agendar as instruções posteriores ao desvio.

Forward: A detecção do forward

Na entrada da ula tem multiplexadores, onde eu devo escolher se o que vai vir da ula é aquilo que vem do banco de registradores, ou aquilo tá no registrador EX/MEM ou aquilo que tá gravado no registrador MEM/WB, e então eu faço o controle de sinais por forward criando uma unidade de forward. Posso identificar forward quando o **rd** da instrução anterior for igual ao **rs1** ou **rs2** da instrução atual.

5 -

Exemplo 1:

testeSIMPLESPIPE.s | test-hazard.asm | test-hazard2.asm*

```
1  .text
2          addi $t0, $t0, 10
3          addi $t1, $t0, 1
```

Exemplo 2:

testeSIMPLESPIPE.s | test-hazard.asm* | test-hazard2.asm

```
1  .data
2          WORD: .word 32
3  .text
4          lui $t0,0x1001
5          lw $t2, 0($t0)
```