



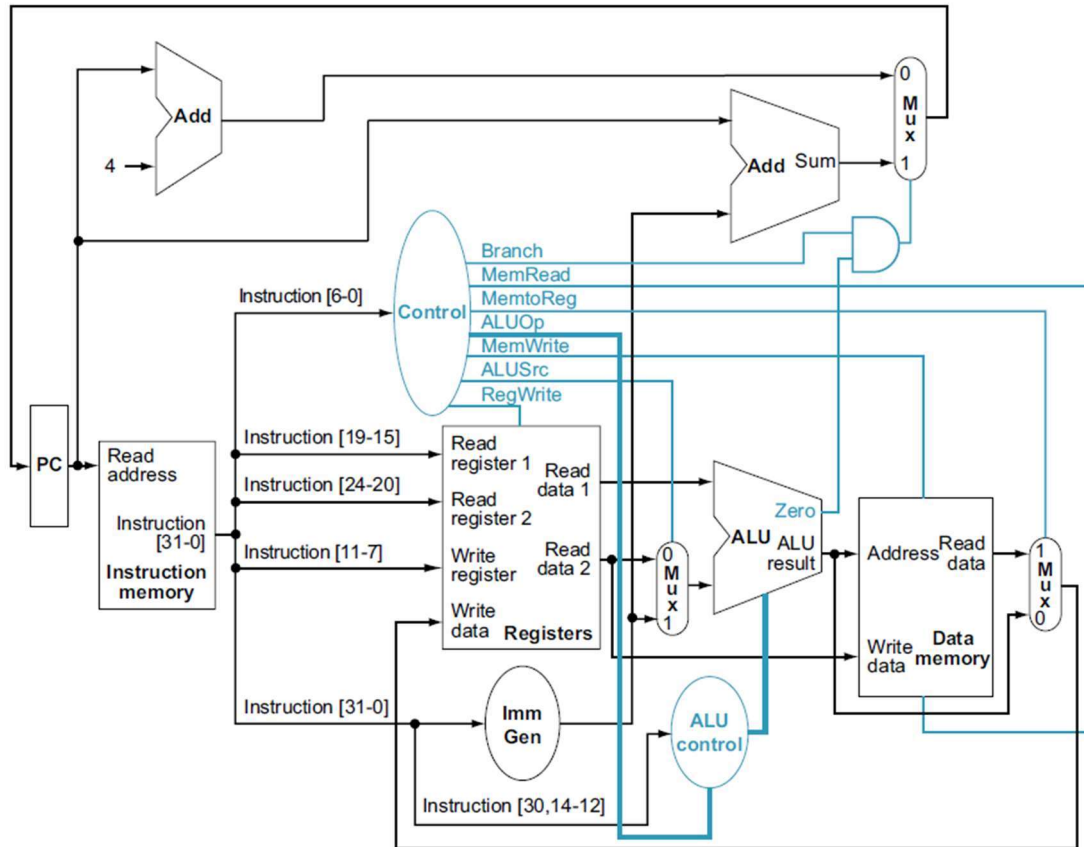
UNIVERSIDADE DE BRASÍLIA
DISCIPLINA: ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES
TURMA: C
2º SEMESTRE DE 2021

RISC-V UNICICLO

FELIPE FARIAS DA COSTA – 190027592

Descrição

O projeto consiste na construção de um processador RISC-V utilizando a linguagem VHDL. Para a compilação das entidades e realização dos testes, foi utilizado o ModelSim. O diagrama inicial do caminho de dados do RISC-V é apresentado na figura abaixo:



Porém, algumas modificações precisam ser feitas para que possamos implementar todas instruções desejadas.

Instruções

Lógico-Aritméticas: ADD, SUB, AND, OR, XOR, SLT, SLTU

Lógico-Aritméticas com imediato: ADDi, ANDi, ORi, XORi, SLLi, SRLi, SRAi, SLTi, SLTUi, LUI, AUIPC

Subrotinas: JAL, JALR

Saltos: BEQ, BNE, BLT, BGE, BGEU, BLTU

Memória: LW, SW

As instruções destacadas exigiram mudanças no caminho de dados para que pudessem ser implementadas.

Para verificar se a arquitetura foi implementada corretamente para todas as instruções, vamos executar um programa de teste e observar os valores de saída obtidos nos módulos.

Instruções ORI, ANDI, LUI, LW, ADD, SW, LW, ADDI

		.text		#	PC	MI	ULA	MD			
		ori	t0, zero, 0xFF	#	00000000	0ff06293	000000ff	00000000			
		andi	t0, t0, 0xF0	#	00000004	0f02f293	000000f0	00000000			
		lui	s0, 2	#	00000008	00002437	00002000	00000000			
		lw	s1, 0(s0)	#	0000000c	00042483	00002000	0000000f			
		lw	s2, 4(s0)	#	00000010	00442903	00002004	0000003f			
		add	s3, s1, s2	#	00000014	012489b3	0000004e	00000000			
		sw	s3, 8(s0)	#	00000018	01342423	00002008	0000004e			
clk	0										
dataout_pc	00000020	00000000	00000004	00000008	0000000c	00000010	00000014	00000018	0000001c	00000020	00000024
instruction	7F000A13	0FF06293	0F02F293	00002437	00042483	00442903	012489B3	01342423	00842503	7F000A13	0FF00A93
z_alu	000007F0	000000FF	000000F0	00002000	00002000	00002004	0000004E	00002008	000000F0	000000FF	
dataout_ram	00000000	00000000	00000000	0000000F	00000000	0000003F	00000000	00000000	0000004E	00000000	

Observando as ondas no Modelsim, podemos verificar os 4 dados de saída para cada instrução:

Instruções AND, OR, XOR, SLLI, SRLI, SRAI, SLT, SLTU

```

and    s6, s5, s4      # 00000028 014afb33 000000f0 00000000
or     s7, s5, s4      # 0000002c 014aebb3 000007ff 00000000
xor    s8, s5, s4      # 00000030 014acc33 0000070f 00000000
slli   t1, s5, 4        # 00000034 004a9313 00000ff0 00000000
lui    t2, 0xFF000      # 00000038 ff0003b7 ff000000 00000000
srli   t3, t2, 4        # 0000003c 0043de13 0ff00000 00000000
srai   t4, t2, 4        # 00000040 4043de93 fff00000 00000000
slt    s0, t0, t1       # 00000044 0062a433 00000001 00000000
slt    s1, t1, t0       # 00000048 005324b3 00000000 00000000
sltu   s3, zero, t0     # 0000004c 005039b3 00000001 00000000
sltu   s4, t0, zero     # 00000050 0002ba33 00000000 00000000

```

clk	0												
dataout_pc	00000050	0...	00000028	0000002C	00000030	00000034	00000038	0000003C	00000040	00000044	00000048	0000004C	00000050
instruction	0002BA33	0...	014AFB33	014AE8B3	014ACC33	004A9313	FF0003B7	0043DE13	4043DE93	0062A433	005324B3	005039B3	0002BA33
z_alu	00000000	0...	000000F0	000007FF	0000070F	00000FF0	FF000000	0FF00000	FFF00000	00000001	00000000	00000001	00000000
dataout_ram	0000000F	0...	00000000	00000000	00000000	00000000	0000000F	00000000	00000000	00000000	00000000	00000000	00000000

Instruções JAL, SUB, JALR, BEQ, BNE

jal	x0, next	# 00000058	00c0006f	00000000	00000000 => 64
testasub:					
sub	t3, t0, t1	# 0000005c	40628e33	ffffff100	00000000
jalr	x0, ra, 0	# 00000060	00008067	00000058	00000000 => 58
next:					
addi	t0, zero, -2	# 00000064	ffe00293	fffffffe	00000000
beqsim:					
addi	t0, t0, 2	# 00000068	00228293	0000000*	00000000 * t0 = 0, 2
beq	t0, zero, beqsim	# 0000006c	fe028ee3	00000000	00000000 => 68, 70
bnesim:					
addi	t0, t0, -1	# 00000070	fff28293	0000000*	00000000 * t0 = 1, 0
bne	t0, zero, bnesim	# 00000074	fe029ee3	00000000	00000000 => 70, 78
	addi t0, zero, 1	# 00000078	00100293	00000001	xxxxxxxx

[illegible]

Instruções BLT, BGE, AUIPC

```

bltadd:
    addi t0, t0, -1          # 0000007c fff28293 00000000 xxxxxxxx
    blt  t0, zero, blton     # 00000080 0002c463 xxxxxxxx xxxxxxxx => 84, 88
    j     bltadd             # 00000084 ff9ff06f xxxxxxxx xxxxxxxx => 7c
blton:
    bge  t0, zero, end       # 00000088 0002d663 xxxxxxxx xxxxxxxx => 8c, 94
    addi t0, t0, 1           # 0000008c 00128293 00000000 xxxxxxxx
    j     blton              # 00000090 ff9ff06f xxxxxxxx xxxxxxxx => 88
end:
    auipc t1, 10             # 00000094 0000a317 00000000 xxxxxxxx
    addi  t0, x0, 10         # 00000098 00a00293 00000000 xxxxxxxx
    addi  t1, x0, 20         # 0000009c 01400313 00000000 xxxxxxxx
    add   t2, t0, t1         # 000000a0 006283b3 00000000 xxxxxxxx

```

clk	0											
dataout_pc	0000009C	0000007C	00000084	0000007C	00000080	00000088	0000008C	00000090	00000088	00000094	00000098	0000009C
instruction	01400313	FFF28293	0002C463	FF9FF06F	FFF28293	0002C463	0002D663	00128293	FF9FF06F	0002D663	0000A317	00A00293
z_alu	00000014	00000000	00000000	FFFFFFFF	00000001	00000000	00000000	00000000	00000001	00000058	0000000A	00000014
dataout_ram	00000000	0000000F	0000000F	00000000	0000000F	0000000F	0000000F	0000000F	00000000	00000000	0000000E	00000000

Observando com cuidado o código (e seus comentários), vemos que o programa foi executado com sucesso. As instruções foram executadas na ordem correta respeitando as condições de salto, verificando, assim, o conteúdo dos registradores e a implementação dos saltos. Nota-se que alguns valores da saída da memória de dados (*dataout_ram*) estão diferentes do apresentado nos comentários do código, isso se deve ao fato da memória ser assíncrona para a leitura, mas é um detalhe de implementação que não afeta o funcionamento do programa no RISC-V Uniclo.

Para executar o simulador no Modelsim, deve-se criar um projeto que contenha todos os arquivos de código .vhd do RISC-V Uniclo (e o testbench *tb_riscv_singlecycle.vhd*) e os arquivos de texto utilizados para carregar as memórias de instruções e dados. O testbench *tb_riscv_singlecycle.vhd* é o ponto de partida para a simulação do programa armazenado na *rom*.