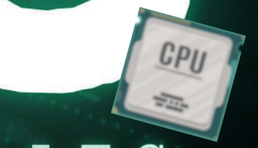
UNIVERSIDADE FEDERAL DE RORAIMA

UFRR

Conheça ASES



PROCESSADOR RISC 8 BITS

POR: ANDERSSON SILVA, ANDREZA GONÇALVES E ARTHUR RAMOS

ORIENTADOR: DR. PROF. HEBERT ROCHA

UNIVERSIDADE FEDERAL DE RORAIMA

ASES ®

ASES CPU

ASES ® CORE™ A3

A3 - 3625C

GONÇALVES,
RAMOS,
SILVA.

- PROCESSADOR RISC
- 8 BITS
- BASEADO NO MIPS

O melhor e mais confiável processador 8 bits da atualidade!

INSTRUÇÕES

Tipo R: Operam diretamente nos registradores.			
Opcode	Registrador 1	Registrador 2	Funct
3 bits	2 bits	2 bits	1 bit
7-5	4-3	2-1	0
Tipo I: Instruções que envolvem memória.			
Opcode	Registrador 1	Immediate	
3 bits	2 bits	3 bits	
7-5	4-3	2-0	
Tipo J: Saltos incondicionais.			
Opcode		Address	
3 bits		5 bits	
7-5		4-0	

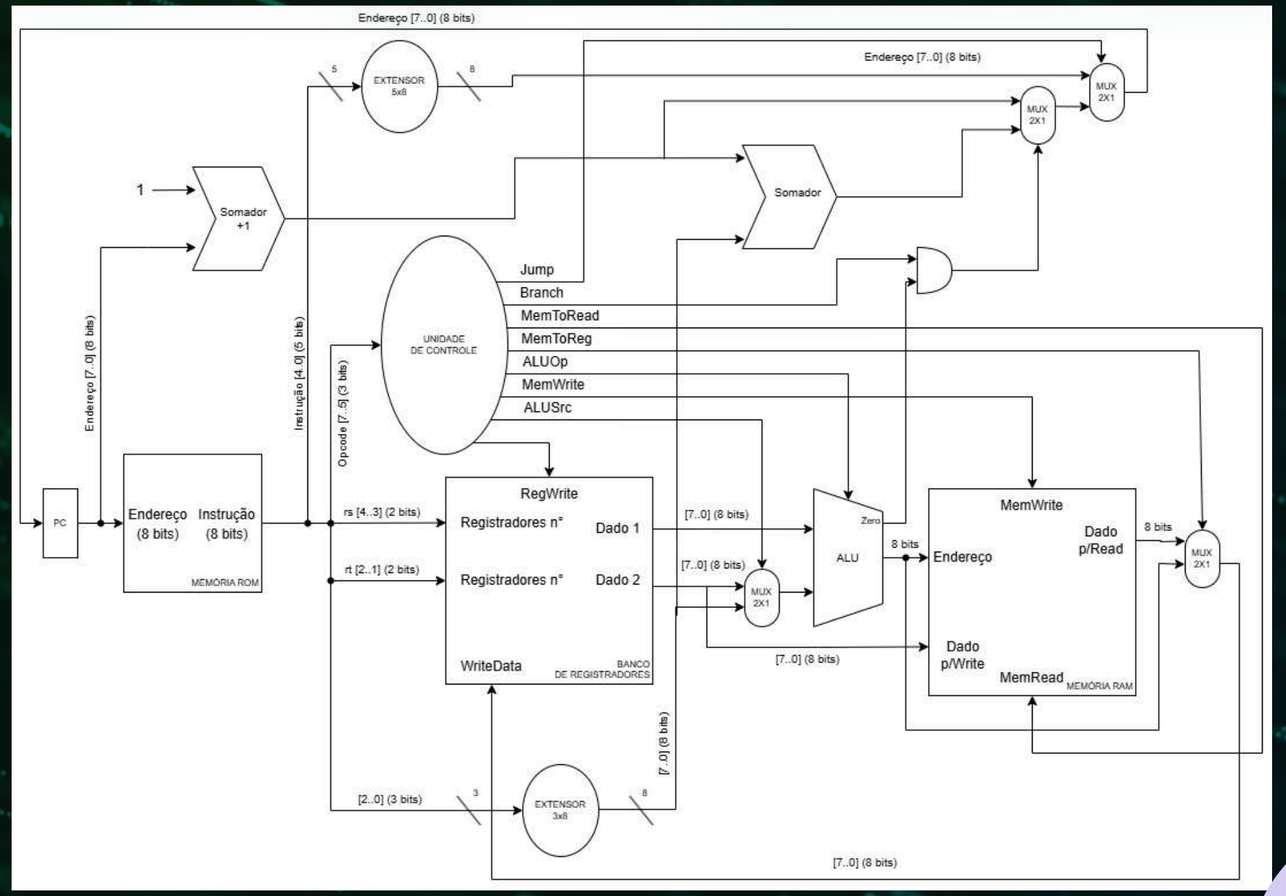
DIVISÃO DE BITS

A divisão dos bits foi feita para um processador de 8 bits com suporte a instruções do tipo R, I e J. A estrutura do Tipo R permite 4 registradores por causa do número de bits reservados para identificar os registradores. O imediato tem 3 bits, permitindo acessar endereços pequenos de memória de 0 a 7. O endereço de salto tem 5 bits, permitindo um alcance de 2^5 = 32 endereços na memória de instruções.

OPERAÇÕES SUPORTADAS

Opcode	Sintaxe	Nome	Exemplo
000	J	Jump	J ADRESS
001	BEQ	Branch if equal	BEQ ADRESS
010	LW	Load word	LW $S0 ADRESS
011	SW	Store word	SW $S1 ADRESS
100	ADD	Soma	ADD $S0 $S1
101	SUB	Subtração	SUB $S3 $S2
110	LI	Load imediato	LI $S0 1
111	ADDI	Soma imediata	ADDI $S0 2

DATAPATH DO PROCESSADOR



TESTES

```
case address is
    -- Teste ADD e ADDI
    when "00000000" => data_out <= "11000010";  -- LI R1, 2
    when "00000001" => data_out <= "11001100";  -- LI R2, 4
    when "00000010" => data_out <= "10010110";  -- ADD R3, R4 (R3 = R1 + R2)
    when "00000011" => data_out <= "11110101";  -- ADDI R3, 5 (R3 = R3 + 5)

    when others => data_out <= "00000000";  -- Default (NOP)
```

```
-- Teste SUB e SUBI
when "00000000" => data_out <= "11001111";  -- LI R1, 8
when "00000001" => data_out <= "11000101";  -- LI R2, 5
when "00000010" => data_out <= "10110110";  -- SUB R3, R4 (R3 = R1 - R2)
when "00000011" => data_out <= "00110001";  -- SUBI R3, 1 (R3 = R3 - 1)

when others => data_out <= "00000000";  -- Default (NOP)
```

Testes das instruções:

1. ADD e ADDI;
2. SUB e SUBI;
3. BEQ e JUMP;
4. LW, SW e LI;
5. Fibonacci.

TESTES

```
-- Teste BEQ e JUMP
when "00000000" => data_out <= "11001110";  -- LI R1, 7
when "00000001" => data_out <= "11001110";  -- LI R2, 7
when "00000010" => data_out <= "00101011";  -- BEQ R1, R2 (Se R1 == R2, pula para endereço 11)
when "00000011" => data_out <= "11000001";  -- LI R1, 1 (Se não pulou, carrega 1 em R1)
when "00000100" => data_out <= "00001000";  -- JUMP 8 (Pula para endereço 8)
when "00000101" => data_out <= "11000011";  -- LI R1, 3 (Nunca será executado se o BEQ funcionou)

when others => data_out <= "00000000";  -- Default (NOP)
```

```
-- Teste LW, SW e LI
when "00000000" => data_out <= "11000010";  -- LI R1, 2
when "00000001" => data_out <= "01010000";  -- SW R1, 000 (Salva R1 na memória)
when "00000010" => data_out <= "01001100";  -- LW R3, 000 (Carrega R3 da memória)
when "00000011" => data_out <= "11000101";  -- LI R2, 5
when "00000100" => data_out <= "10010110";  -- ADD R3, R4 (R3 = R1 + R2)

when others => data_out <= "00000000";  -- Default (NOP)
```

Testes das instruções:

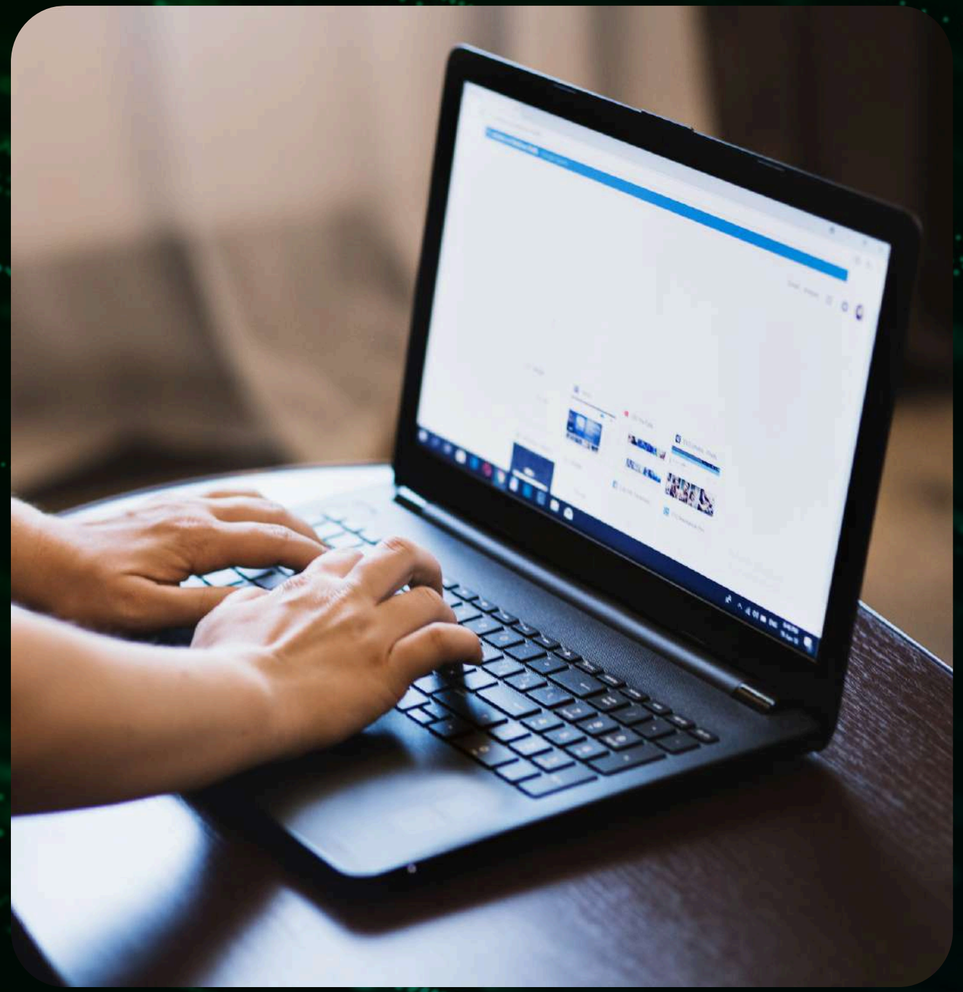
1. ADD e ADDI;
2. SUB e SUBI;
3. BEQ e JUMP;
4. LW, SW e LI;
5. Fibonacci.

TESTES

```
case address is
    when "00000000" => data_out <= "11000000";  -- LI R1, 0
    when "00000001" => data_out <= "11001111";  -- LI R2, 8
    when "00000010" => data_out <= "11010000";  -- LI R3, 0
    when "00000011" => data_out <= "11011001";  -- LI R4, 1
    when "00000100" => data_out <= "10010110";  -- ADD R3, R4 (Fibonacci inicial)

    -- LOOP
    when "00000101" => data_out <= "01110000";  -- SW R3, 000 (Salva Fibonacci)
    when "00000110" => data_out <= "10010110";  -- ADD R3, R4 (Próximo Fibonacci)
    when "00000111" => data_out <= "01011000";  -- LW R4, 000 (Carrega R4 da memória)
    when "00001000" => data_out <= "11100001";  -- ADDI R1, 1 (contador++)
    when "00001001" => data_out <= "00101001";  -- BEQ R2, FIM (Se contador == 8, fim)
    when "00001010" => data_out <= "00000101";  -- JUMP LOOP (volta para loop)

    -- FIM
    when "00001011" => data_out <= "00000000";  -- JUMP FIM (Loop infinito)

    when others => data_out <= "00000000";  -- Default (NOP)
end case;
```

Testes das instruções:

1. ADD e ADDI;
2. SUB e SUBI;
3. BEQ e JUMP;
4. LW, SW e LI;
5. Fibonacci.



REPOSITÓRIO

Link Github:
https://github.com/ArthurRamos26/AOC_Andersson_SilvaAndreza_GoncalvesArthur_Ramos_UFRR_2024/tree/main/processador8bits

OBRIGADO