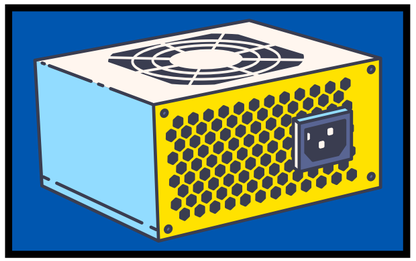
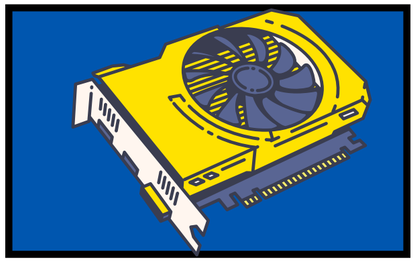
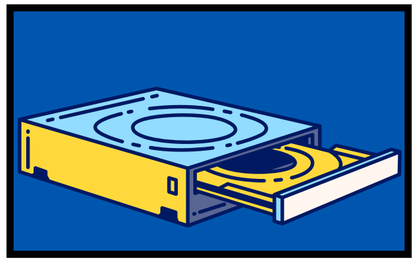
1 CHECK POINT
ARQUITETURA



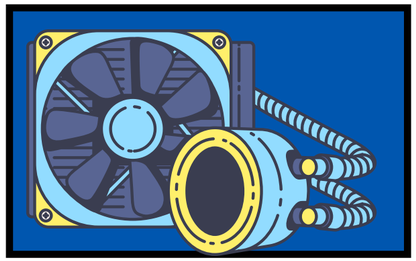
MARIA EDUARDA



CAIO



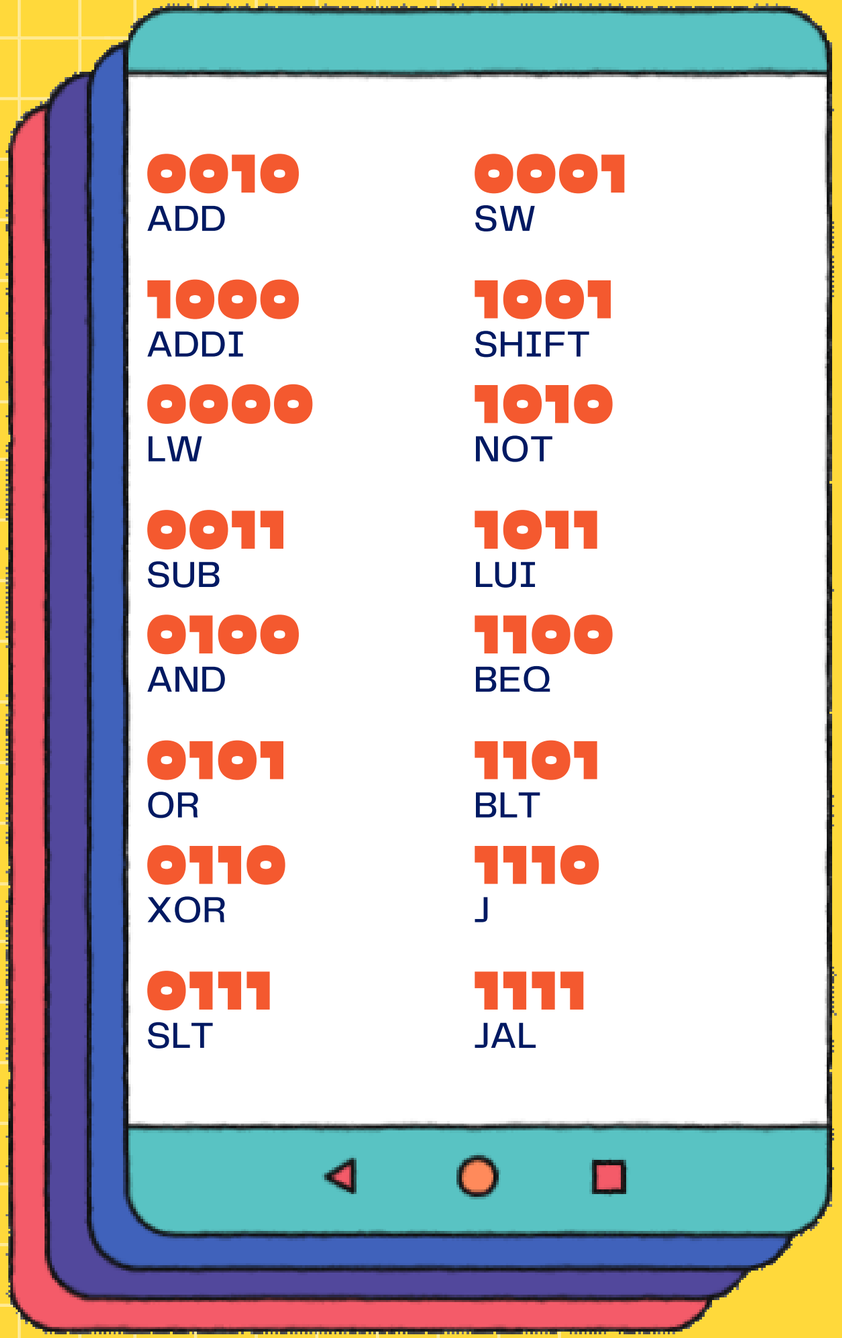
GUILHERME



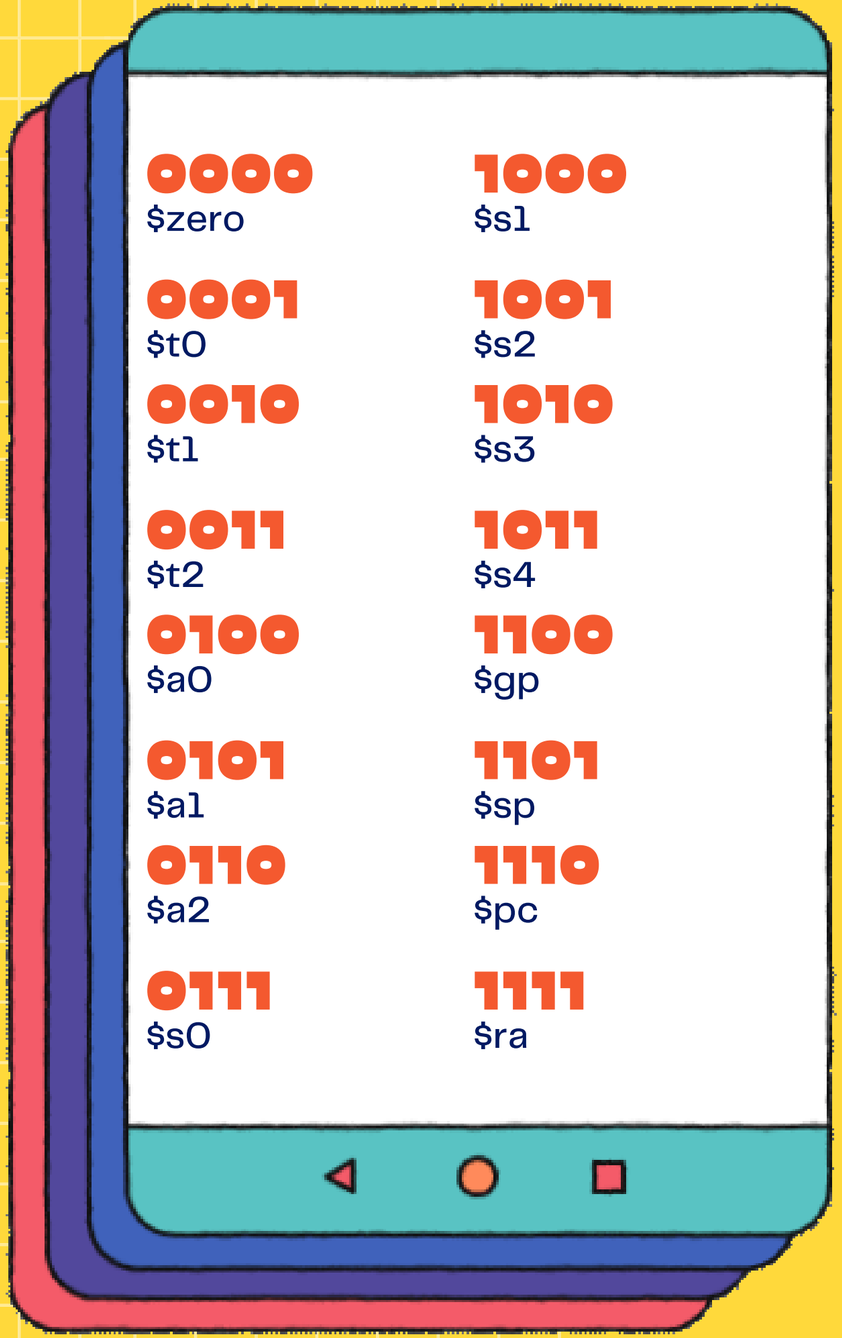
RODRIGO

PROCESSADOR

upcodes



Categoria	Instrução	Opcode	Exemplo
Aritmética	Add	0010_2	Add $s1,$s2,$s3
	Sub	0011_2	sub $s1,$s2,$s3
	Addi	1000_2	addi $s1,100
	Shift	1001_2	Sft $s1,8
Lógica	And	0100_2	And $s1,$s2,$s3
	Or	0101_2	or $s1,$s2,$s3
	Not	1010_2	Not $s1
	Xor	0110_2	xor $s1,$s2,$s3
	Slt	0111_2	Slt $s1,$s2,$s3
Transferência	Lw	0000_2	lw $s1,$s2,$s3
	Sw	0001_2	sw $s1,$s2,$s3
	Lui	1011_2	Lui $s1,100
Desvio Condicional	Beq	1100_2	beq $s1,$s2,5
	Blt	1101_2	blt $s1,$s2,5
Desvio incondicional	J	1110_2	J $s1,100
	Jal	1111_2	Jal $s1,100



register

Código	Símbolo	Função	Descrição
0000_2	$zero	Constante zero	Constante 0 de 16 bits
0001_2 0010_2 0011_2	$t0 $t1 $t2	Temporários	Registradores Auxiliares
0100_2 0101_2 0110_2	$a0 $a1 $a2	Argumento	Argumentos para operações aritméticas e procedimentos
0111_2 1000_2 1001_2 1010_2 1011_2	$s0 $s1 $s2 $s3 $s4	salvos	Armazena valores durante chamadas de procedimento
1100_2	$gp	Apontador global	Aponta para as variáveis globais na pilha
1101_2	$sp	Apontador pilha	Aponta para o topo da pilha
1110_2	$pc	Contador de programa	Aponta para a próxima instrução
1111_2	$ra	Endereço de Retorno	Armazena o endereço de retorno de uma rotina

Registrar

```
-- Criando matrizes de Memória de 16 endereços
        type RAM_ARRAY is array (0 to 15) of std_logic_vector (0 to 15);
        type Inst_ARRAY is array (0 to 15) of std_logic_vector (0 to 15);
        type Reg_ARRAY is array (0 to 15) of std_logic_vector (0 to 15);
-- Declarando Banco de Registradores
        signal Reg_Bank: Reg_ARRAY:=(
        "0000000000000000","0000000000000100","0000000000000001","0000000000001000", --  0(0000) |  1(0001) | 2(0010)  | 3(0011)
        "0000000000000000","0000000000000000","0000000000000010","0000000000000000", --  4(0100) |  5(0101) | 6(0110)  | 7(0111)
        "0000000000000000","0000000000000000","0000000000000000","0000000000000000", --  8(1000) |  9(1001) | 10(1010) | 11(1011)
        "0000000000000000","0000000000000000","0000000000000000","0000000000000000"  -- 12(1100) | 13(1101) | 14(1110) | 15(1111)
);
-- Declarando memoria de instruções
        signal Inst_Bank: Inst_ARRAY:=(
        "0010000100001010","0010001100100100","0011001101100111","0011010000111001", --  0 |  1 | 2  | 3 1000000001100101
        "0000000000000000","0000000000000000","0000000000000000","0000000000000000", --  4 |  5 | 6  | 7 1001000000010001
        "0000000000000000","0000000000000000","0000000000000000","0000000000000000", --  8 |  9 | 10 | 11
        "0000000000000000","0000000000000000","0000000000000000","0000000000000000"  -- 12 | 13 | 14 | 15
);
-- Declarando valores da RAM
        signal RAM: RAM_ARRAY :=(
        "0000000000000000","0000000000000000","0000000000000000","0000000000000000", --  0 |  1 | 2  | 3
        "0000000000000000","0000000000000000","0000000000000000","0000000000000000", --  4 |  5 | 6  | 7
        "0000000000000000","0000000000000000","0000000000000000","0000000000000000", --  8 |  9 | 10 | 11
        "0000000000000000","0000000000000000","0000000000000000","0000000000000000"  -- 12 | 13 | 14 | 15
);

begin
```

ADD

```
when "0010" =>
Reg_bank(to_integer(unsigned(instruct(12 to 15)))) <=
Reg_bank(to_integer(unsigned(instruct(4 to 7)))) +
Reg_bank(to_integer(unsigned(instruct(8 to 11))));
destino <= to_integer(unsigned(instruct(12 to 15)));
DECOP2 <= "0001000"; -- 'A' de add;
```

SUB

```
when "0011" =>
Reg_bank(to_integer(unsigned(instruct(12 to 15)))) <=
  Reg_bank(to_integer(unsigned(instruct(4 to 7)))) -
Reg_bank(to_integer(unsigned(instruct(8 to 11))));
  destino <= to_integer(unsigned(instruct(12 to 15)));
  DECOP2 <= "0100100"; -- 'S' de sub;
```

SHIFT

```
when "1001" =>
Reg_bank(to_integer(unsigned(instruct(8 to 11)))) <=
std_logic_vector(shift_left(unsigned(Reg_bank(to_integer(unsigned(instruct(8
to 11))))),to_integer(unsigned(instruct(12 to 15))));
destino <= to_integer(unsigned(instruct(8 to 11)));
DECOP2 <= "0111100"; -- 'S' de shift;
```

ADDI

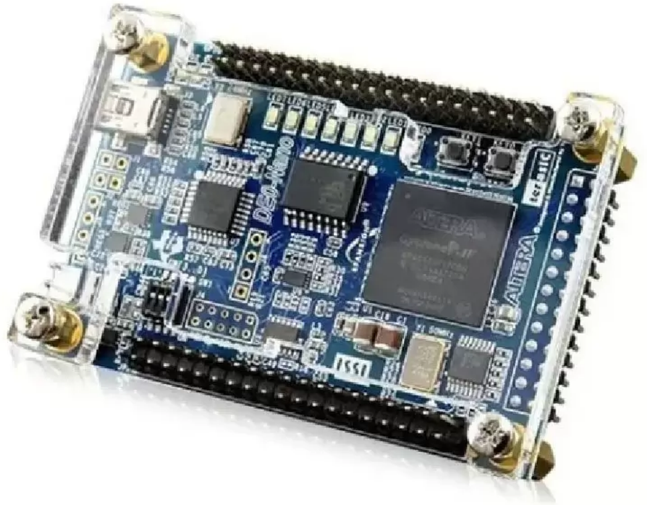
```
when "1000" =>
Reg_bank(to_integer(unsigned(instruct(8 to 11)))) <=
Reg_bank(to_integer(unsigned(instruct(4 to 7)))) +
x"000" & instruct(12 to 15);
destino <= to_integer(unsigned(instruct(8 to 11)));
DECOP2 <= "1001001"; -- 'A' de addi;
```

OUTROS

```
                    when others =>
output  <= 0        ; -- '-' de passagem intermediaria
destino <= 0    ; -- '-' de passagem intermediaria
DECOP2  <= "1110111"; -- '-' de passagem intermediaria
```

NUMEROS DA VARIAVEL SAIDA

```
case(output) is
when 0   => DECOP <= "0000001"; --'0'
 when 1   => DECOP <= "1001111"; --'1'
when 2   => DECOP <= "0010010"; --'2'
when 3   => DECOP <= "0000110"; --'3'
when 4   => DECOP <= "1001100"; --'4'
when 5   => DECOP <= "0100100"; --'5'
when 6   => DECOP <= "0100000"; --'6'
 when 7   => DECOP <= "0001111"; --'7'
when 8  => DECOP <= "0000000"; --'8'
when 9   => DECOP <= "0000100"; --'9'
when others => DECOP <= "1110111"; --'-'
end case;
 end if;
led_out <= ciclo and ENABLE;
-- DECOP  <= DEC;
-- DECOP2 <= DEC2;
leds  <= instruct(0 to 3);
leds2 <= instruct(4 to 7);
if(to_integer(unsigned(Reg_bank(14))) = 15) then
DECOP  <= "1110111";
DECOP2 <= "1110111";
end if;
```



processador funcionando

OBRIGADA!

Dúvidas?