Trabalho Final

1 Introdução

O objetivo é desenvolver em VHDL um processador digital simples segundo as especificações contidas neste documento. Deve ser entregue os códigos desenvolvidos, um relatório e apresentar o processador digital funcionando no kit de FPGA do laboratório.

2 Especificação

2.1 Barramentos

Na arquitetura proposta, utilizaremos três barramentos principais, conforme ilustrado na imagem anexa:

- Barramento de Endereço: Indica o endereço que será acessado para leitura ou escrita, seja na memória, entrada ou saída.
- Barramento de Dados: Transporta os dados a serem lidos ou escritos no endereço apontado.
- Barramento de Controle: Transporta os sinais de controle, como:
  - Read: Indica que será realizada uma leitura de memória (ativa apenas se Memory Enable estiver ativa).
  - Write: Indica que será realizada uma escrita de memória (ativa apenas se Memory Enable estiver ativa).
  - Memory Enable: Habilita a leitura ou escrita na memória.
  - Input Enable: Habilita a leitura dos dispositivos de entrada.
  - Output Enable: Habilita a escrita nos dispositivos de saída.
  - Clock: Controle de ciclo do processador
  - Reset: reset para o estado inicial.

A conexão entre os barramentos, dispositvos e CPU pode ser visualizado na imagem abaixo.

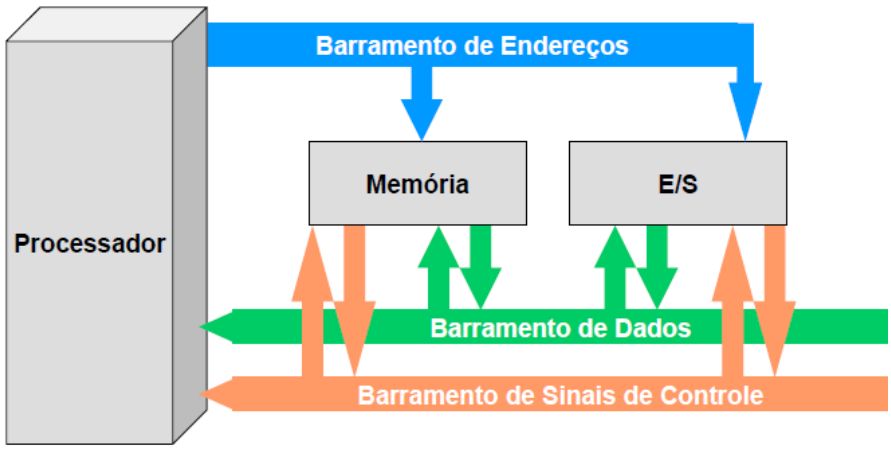

Figure 1: Barramentos

2.2 Componentes Internos da CPU

A CPU é composta pelos seguintes módulos:

- Input Unit: Utiliza 8 chaves do kit FPGA como entrada.
- Output Unit: Exibe dados de saída em 8 LEDs do kit FPGA.
- Memory Unit: Memória de 8 bits, utilizando módulos desenvolvidos em atividades anteriores.
- CPU:
  - ALU: Realiza operações lógicas e aritméticas (soma, subtração, AND, OR, NOT).
  - Control Unit:
    - Decodificação de Instruções: Identifica a instrução e ativa os componentes necessários (ALU, registradores, etc.).
    - Coordenação de Operações: Controla o fluxo de dados entre CPU, memória, e unidades de entrada e saída.
    - Sinais de Controle: Gera sinais para sincronizar as operações entre ALU, memória, entrada e saída

2.3 Registradores da CPU

A CPU terá os seguintes registradores:

- Flags (1 bit): Zero, Sinal, Carry, Overflow.

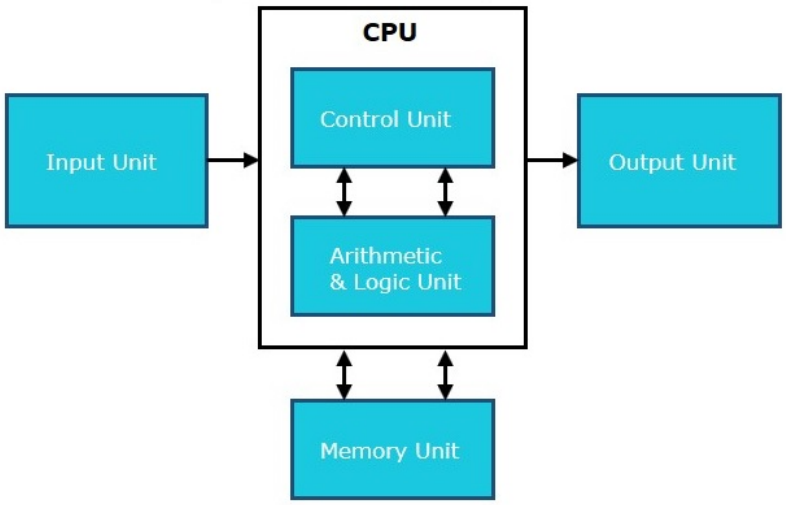

Figure 2: Componentes internos da CPU

- A e B (8 bits): Registradores de propósito geral.
- R (8 bits): Registrador para armazenar resultados da ALU.
- Program Counter (PC) (8 bits): Contador de instruções.
- Instruction Register (IR) (4 bits): Armazena a instrução atual.

2.4 Instruções

As instruções planejadas incluem operações aritméticas, lógicas e de controle de fluxo:

- ADD Reg1, Reg2: Soma valores de Reg1 e Reg2, armazena o resultado em R.
- SUB Reg1, Reg2: Subtrai Reg1 de Reg2, armazena o resultado em R.
- AND Reg1, Reg2: Realiza operação AND entre Reg1 e Reg2, armazena o resultado em R.
- OR Reg1, Reg2: Realiza operação OR entre Reg1 e Reg2, armazena o resultado em R.
- NOT Reg1: Inverte os bits de Reg1, armazena o resultado em R.
- CMP Reg1, Reg2: Compara Reg1 e Reg2, atualizando os registradores de flags.

- JMP addr: Salta para o endereço addr.
- JEQ addr: Salta para addr se o último resultado comparado for igual.
- JGR addr: Salta para addr se o último resultado comparado for maior.
- LOAD Reg1, addr: Carrega o valor do endereço addr para Reg1.
- STORE Reg1, addr: Armazena o valor de Reg1 no endereço addr.
- MOV Reg1, Reg2: Move o valor de Reg2 para Reg1.
- IN Reg1: Lê o valor das chaves do FPGA e armazena em Reg1.
- OUT Reg1: Exibe o valor de Reg1 nos LEDs.
- WAIT: (Possível inclusão) Aguarda um sinal de entrada para continuar

Note que as instruções como ADD podem usar os Registradores A,B, R e também podem receber números de 8 bits. Exemplos:

```
ADD A B      ; Soma A e B e Armazena em R (R=A+B)
ADD A R      ; Soma A e R e Armazena em R (R=A+R)
ADD B R
ADD R R      ;
ADD A 255    ; Soma A e 255 e Armazena em R (R=A+255)
```

A instruções STORE podem armazenar valores de um Registrador no endereço específicado. O endereço pode ser um número de 8 bits ou um Registrador. A instrução LOAD carrega em um registrador o valor de um endereço na memória que pode ser um número de 8 bits ou um registrador. Exemplos

```
STORE A 00000000
STORE R 00000010
STORE B R
LOAD A B
LOAD A 00000000
```

A implementação das instruções e codificação fica em aberto para implementar da forma que acharem mais conveniente.

3 Exemplos de programas para CPU

Os três exemplos a seguir servem para ajudar a entender como as instruções da CPU são utilizadas.

3.1 Soma

```
IN A ; Lê o que está no input para Registrador A
IN B ; Lê o que está no input para Registrador B
ADD A, B ; Realiza operação de soma de A+B armazenadno em R
OUT R ; Coloca o valor de R na saída
WAIT
```

3.2 Multiplicação

```
IN A ; Lê o que está no input para Registrador A
IN B ; Lê o que está no input para Registrador B
MOV R, 0 ; Coloca o valor zero no Registrador R
LOOP_START:
    CMP B, 0        ; Compara B com 0
    JEQ END_LOOP    ; Se B for igual a 0, termina o loop
    ADD R, A        ; Soma A ao valor acumulado em R (R = R +
        A)
    SUB B, 1        ; Decrementa o multiplicador B (B = B -
       1)
    JMP LOOP_START ; Repete o loop
END_LOOP:
    OUT R
WAIT
```

3.3 Série de Fibonacci

```
MOV A, 0           ; A = 0 primeiro valor da sequência
MOV B, 1           ; B = 1 segundo valor da sequência
MOV R, 0x10        ; Endereço base da memória para armazenar
    a sequência

STORE A, R         ; Armazena A no endereço R (0x10)
ADD R, 1           ; Incrementa o endereço para o próximo nú
    mero
STORE B, R         ; Armazena B no próximo endereço

; Define o endereço limite para o cálculo (0x19 para 10 nú
    meros de Fibonacci)
MOV R, 0x12        ; Ajusta R para o próximo endereço onde
    será armazenado o próximo número

LOOP_START:
    ; Calcula o próximo número da sequência
    ADD A, B       ; Soma A e B, resultado em R (R = A + B)
    STORE R, R     ; Armazena o valor de R no endereço da
       memória apontado por R
```

```
16
17      ; Atualiza valores para o próximo cálculo
18      MOV A, B        ; Move B para A (A = B)
19      MOV B, R        ; Move o novo valor de R para B (B = R)
20      ADD R, 1        ; Incrementa o endereço para o próximo
            armazenamento
21
22      CMP R, 0x1A     ; Compara o endereço R com o limite 0x1A
23      JGR LOOP_START ; Se R < 0x1A, continua o loop
24
25  END_LOOP:
26      ; Exibe a sequência a partir da memória
27      MOV R, 0x10     ; Redefine R para o endereço base (0x10)
28
29  DISPLAY_LOOP:
30      LOAD A, R       ; Carrega o valor armazenado no endereço
            R para A
31      OUT A           ; Exibe o valor em A nos LEDs
32      ADD R, 1        ; Avança para o próximo endereço
33
34      CMP R, 0x1A     ; Verifica se o endereço ultrapassou o
            limite 0x1A
35      JGR DISPLAY_LOOP ; Se R < 0x1A, continua exibindo
36
37      ; Fim do programa
38      WAIT            ; Espera por uma ação para resetar
```