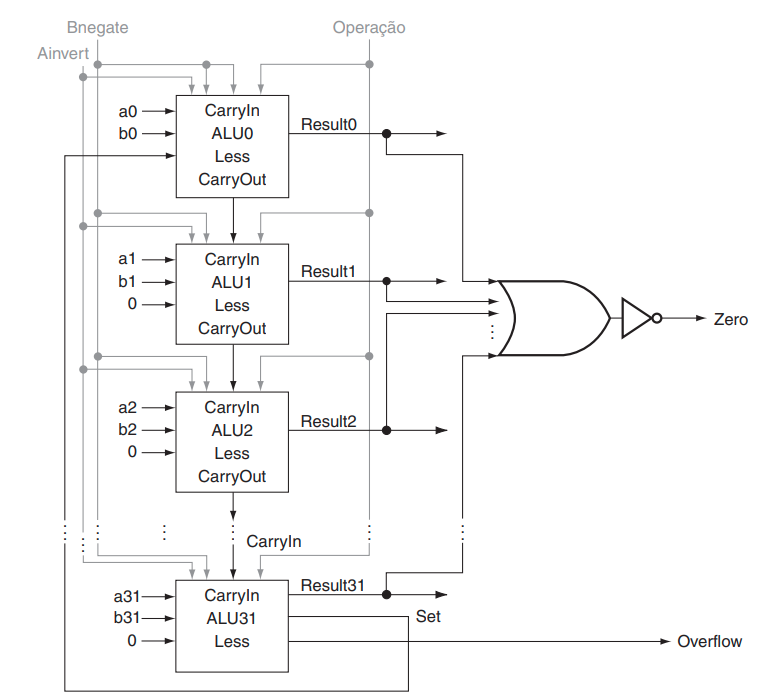
# Implementação do processador MIPS

Neste trabalho, cada grupo, deverá implementar o processador MIPS monociclo considerando a especificação dos módulos apresentada a seguir. Esses módulos serão implementados ao longo das aulas de laboratório (pode ser necessário dedicar outros momentos além do horário da aula para a realização do trabalho). A cada aula, será explicado um novo módulo e seus componentes, portanto, não deixe que o trabalho acumule entre uma aula e outra. Além do projeto do ISE contendo todos os módulos implementados (arquivos .vhd), cada grupo deverá entregar também uma documentação descrita ao final desse documento.

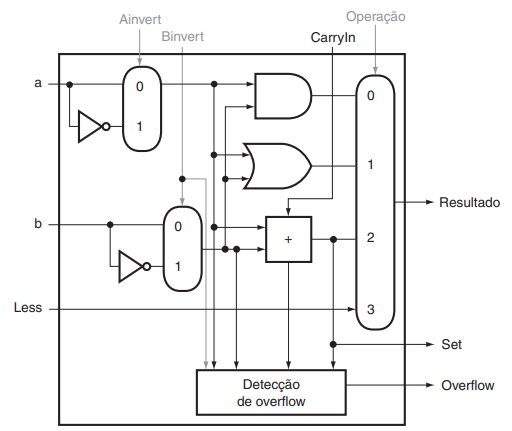
## Descrição dos módulos

### ULA de 32 bits



Não é necessário implementar o sinal overflow. As ULAs de 1 bit devem ser instanciadas através do comando generate. Além da ULA de 1 bit você deve utilizar a OR de 32 entradas. As entradas A e B e a saída Result devem ser representadas por vetores.

### ULA de 1 bit



Não é necessário implementar o módulo “Detecção de overflow” nem o sinal overflow. Você deverá utilizar os módulos somador completo e multiplexador de 2 e 4 bits para essa implementação.

### OR de 2 entradas

Implemente de forma estrutural (utilizando a expressão lógica) uma porta lógica OR de 2 entradas.

### OR de 32 entradas

Utilize a OR de 2 entradas para implementar uma de 32 entradas utilizando o comando generate. As entradas devem ser um único vetor.

### Multiplexador de 2 entradas de 1 bit

Você pode utilizar o código disponibilizado em sala de aula, no entanto, se necessário, adapte-o.

### Multiplexador de 4 entradas de 1 bit

Você pode utilizar o código disponibilizado em sala de aula, no entanto, se necessário, adapte-o.

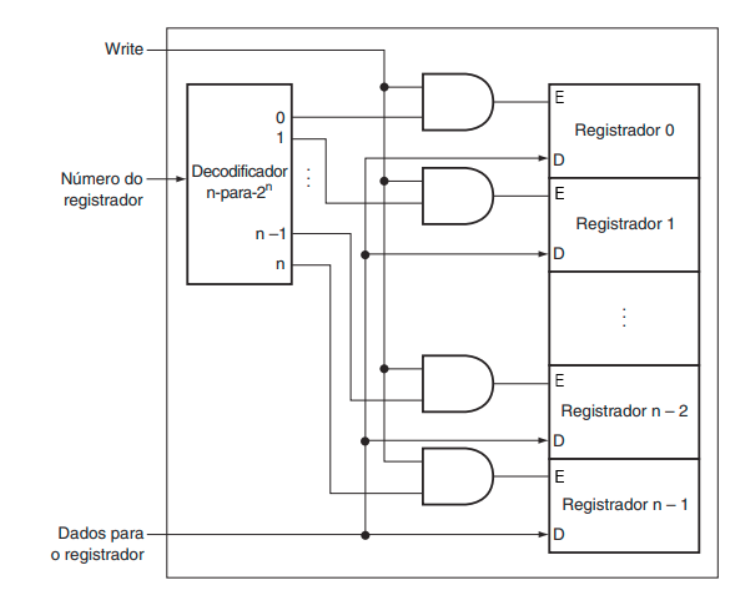
### Somador completo

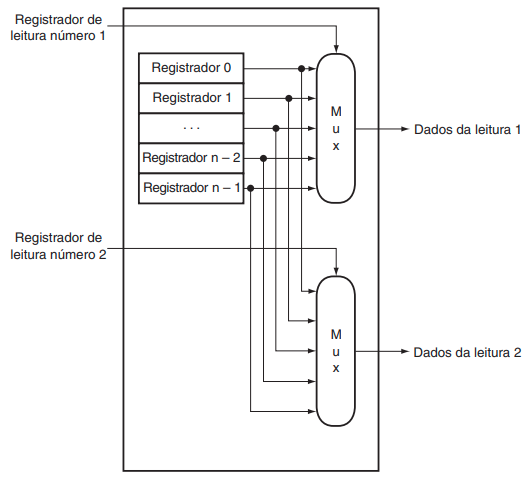
Você pode utilizar o código que você já fez em aula.

### Somador de 32 bits

Utilizando o somador completo e o comando generate implemente um somador onde cada uma de suas entradas e sua saída têm 32 bits. As entradas e a saída devem ser representadas por vetores.

### Banco de registradores

Implemente um banco de registradores que tenha 32 registradores, cada um 32 bits. Para isso, você deverá implementar o registrador de 32 bits com clear, entrada de enable e ativo em borda de descida, o decodificador 5:32 e o multiplexador de 32 entradas de 32 bits, cada. Esses módulos são descritos a seguir. Para essa implementação, você deve utilizar o comando generate para gerar os 32 registradores e as 32 portas AND da figura a seguir (você pode utilizar uma expressão lógica dentro do generate dos registradores, mas fora do port map deles).



No lado esquerdo da figura está representada somente a parte do banco de registradores responsável pela escrita em um registrador. Quando o sinal de controle Write está em nível lógico alto e ocorre uma borda de descida no *clock* do processador, o registrador indicado na entrada Número do registrador recebe o dado da entrada Dado para o registrador. Nessa figura foram omitidas as entradas Clock e Clear que são comuns a todos os registradores e são também entradas do banco de registradores. No lado direito da figura está representada a lógica necessária para permitir a leitura de 2 registradores em paralelo. Para essa implementação, no generate dos registradores, você deve mapear as saídas dos registradores para um sinal que, no momento do mapeamento dos multiplexadores, será utilizado como entrada dos mesmos. Observe que esse sinal deverá ser um vetor de vetores. Para fazer esse vetor, você deverá utilizar o pacote tipo.vhd disponibilizado no portal didático. Lembre-se de indicar a utilização do mesmo na seção de bibliotecas do seu código. Esse mesmo pacote foi utilizado na implementação do multiplexador de 32 palavras de 32 bits. Utilize este módulo como uma referência para sua implementação.

### Multiplexador de 32 entradas de 32 bits

Utilize a implementação disponibilizada no Portal Didático. Lembre-se que é necessário utilizar o pacote implementado no arquivo tipo.vhd. Observe que na implementação do multiplexador, foram utilizadas as funções to\_integer e unsigned da biblioteca IEEE.NUMERIC\_STD. Essas funções também podem ser utilizadas dentro de processos.

### Decodificador 5:32

O decodificador 5:32 tem 5 entradas de endereço que determina qual das 32 saídas estará ativada (as demais saídas deverão estar em 0). Para implementá-lo, utilize um processo que contenha um laço de repetição (for) que compara o contador do laço com o endereço convertido para inteiro (com as funções to\_integer e unsigned da biblioteca IEEE.NUMERIC\_STD). Com base no resultado da comparação, você deve atribuir 0 ou 1 à saída correspondente ao índice.

### Registrador de 32 bits

Utilize o generate e o *Flip flop D com Clear e enable, ativo em borda de descida* descrito a seguir para implementar um registrador de 32 bits com as entradas Clock, Clear, Enable cada uma com 1 bit e o dado de entrada e dado de saída com 32 bits. A saída Q’ dos flip flops pode ser ignorada no momento do mapeamento dos mesmos.

### Flip flop D com Clear e enable, ativo em borda de descida

Modifique o código do flip flop D fornecido em uma aula anterior para implementar um flip flop D que tenha, além das entradas D e Clock, uma entrada assíncrona Clear, ativa em nível lógico alto, que coloque a saída Q em zero. Além dessa entrada, esse flip flop deve ter uma entrada enable que determina quando a entrada D será escrita ou não: a entrada D é escrita no flip flop somente quando acontece a borda de descida do clock e a entrada enable está ativa (está em 1).

### Flip flop D com Clear e enable, ativo em borda de subida

Modifique o código do flip flop D fornecido em uma aula anterior para implementar um flip flop D que tenha, além das entradas D e Clock, uma entrada assíncrona Clear, ativa em nível lógico alto, que coloque a saída Q em zero. Além dessa entrada, esse flip flop deve ter uma entrada enable que determina quando a entrada D será escrita ou não: a entrada D é escrita no flip flop somente quando acontece a borda de subida do clock e a entrada enable está ativa (está em 1).

### Flip flop D com Preset e enable, ativo em borda de descida

Modifique o código do flip flop D fornecido em uma aula anterior para implementar um flip flop D que tenha, além das entradas D e Clock, uma entrada assíncrona Preset, ativa em nível lógico alto, que coloque a saída Q em um. Além dessa entrada, esse flip flop deve ter uma entrada enable que determina quando a entrada D será escrita ou não: a entrada D é escrita no flip flop somente quando acontece a borda de subida do clock e a entrada enable está ativa (está em 1).

### Registrador PC (Program Counter)

Utilize o generate e os *Flip flops D* sensíveis em borda de subida para implementar um registrador de 32 bits com as entradas Clock, Reset, Enable cada uma com 1 bit e o dado de entrada e dado de saída com 32 bits. A saída Q’ dos flip flops pode ser ignorada no momento do mapeamento dos mesmos. A entrada de Reset, quando ativada, deve escrever o valor 0x00400000 no registrador. Dica: use 2 generates para mapear os flip flops que terão a entrada Clear ativada e faça um mapeamento do flip flop que terá a entrada Preset ativada.

### Multiplexador de 2 entradas de 32 bits

Esse mux é utilizado para determinar a entrada do PC, a entrada da ULA e a entrada de dados do banco de registradores (observe na figura que representa todo o processador). Nesta implementação, não utilize um vetor para representar as entradas de dados, utilize 2 entradas separadas, cada uma deve ser um vetor de 32 bits. Você deve utilizar um generate e o multiplexador de 2 entradas de 1 bit.

### Multiplexador de 2 entradas de 5 bits

Esse mux é utilizado para determinar o registrador que é escrito no banco de registradores (observe na figura que representa todo o processador). Nesta implementação, não utilize um vetor para representar as entradas de dados, utilize 2 entradas separadas, cada uma deve ser um vetor de 5 bits. Você deve utilizar um generate e o multiplexador de 2 entradas de 1 bit.

### Extensor de sinal

Esse módulo recebe como entrada um sinal de 16 bits e tem como saída um sinal de 32 bits. Os 16 bits menos significativos da saída são os mesmos da entrada e os demais bits da saída são iguais ao bit mais significativo da entrada.

### Deslocador de 2 bits à esquerda

Esse módulo não precisa ser implementado, seu comportamento pode ser obtido diretamente no mapeamento dos sinais que passam por ele. Para isso, utilize a técnica de *slice* do vetor (acessar apenas uma parte do vetor de cada vez).

### Unidade de controle

Para a implementação da unidade de controle você deverá utilizar um processo com diversos ifs, um para cada opcode listado na tabela a seguir. Em cada um dos ifs, você deve atribuir valores a todos os demais sinais listados na tabela. Quando o sinal estiver identificado com X, não é necessário fazer a atribuição a esse sinal.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instru-**  **ção** | **Opcode** | **ALU**  **Op** | **Reg**  **Write** | **Reg**  **Dst** | **ALU**  **Scr** | **Branch** | **Mem**  **Write** | **Mem**  **ToReg** | **Jump** | **Mem**  **Read** |
| Tipo R | 000000 | 10 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| addi | 001000 | 00 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| lw | 100011 | 00 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| sw | 101011 | 00 | 0 | X | 1 | 0 | 1 | X | 0 | 0 |
| beq | 000100 | 01 | 0 | X | 0 | 1 | 0 | X | 0 | 0 |
| j | 000010 | XX | 0 | X | X | 0 | 0 | X | 1 | 0 |

### Unidade de controle da ULA

A unidade de controle da ULA recebe como entradas o sinal ALUOp vindo da unidade de controle e o campo funct vindo da instrução (bits 5 a 0). Para implementar essa unidade, utilize a mesma técnica de ifs utilizada para a implementação da unidade de controle do processador. Em cada if você deve analisar a operação indicada pelo sinal ALUOp e, quando esse sinal tiver o valor 10, você deverá observar na tabela do lado esquerdo a operação que deve ser realizada pela ULA. A tabela do lado direito apresenta os valores que devem ser colocados nos sinais de controle da ULA de 32 bits para a realização de cada operação.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Instrução do tipo R** | **Funct** | | add | 100000 | | and | 100100 | | nor | 100111 | | or | 100101 | | slt | 101010 | | sub | 100010 | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Operação** | **AInverte** | **BInverte** | **Operação** | **Primeiro Vem1** | | Soma | 0 | 0 | 10 | 0 | | Subtração | 0 | 1 | 10 | 1 | | And | 0 | 0 | 00 | X | | Nor | 1 | 1 | 00 | X | | Or | 0 | 0 | 01 | X | | Slt | 0 | 1 | 11 | 1 | | Beq | 0 | 1 | XX | 1 | | LW e SW | 0 | 0 | 10 | 0 | |

### Memórias de dados

A memória de dados será fornecida no Portal Didático. Ela consiste em um vetor de 256 palavras cada uma de 32 bits. Além dos sinais de controle e de dados, ela possui os sinais DebugEndereco e Debug palavra que são utilizados para permitir a leitura de uma palavra independentemente da execução do processador, permitindo que seja analisado o programa executado por meio da leitura das palavras escritas na memória de dados.

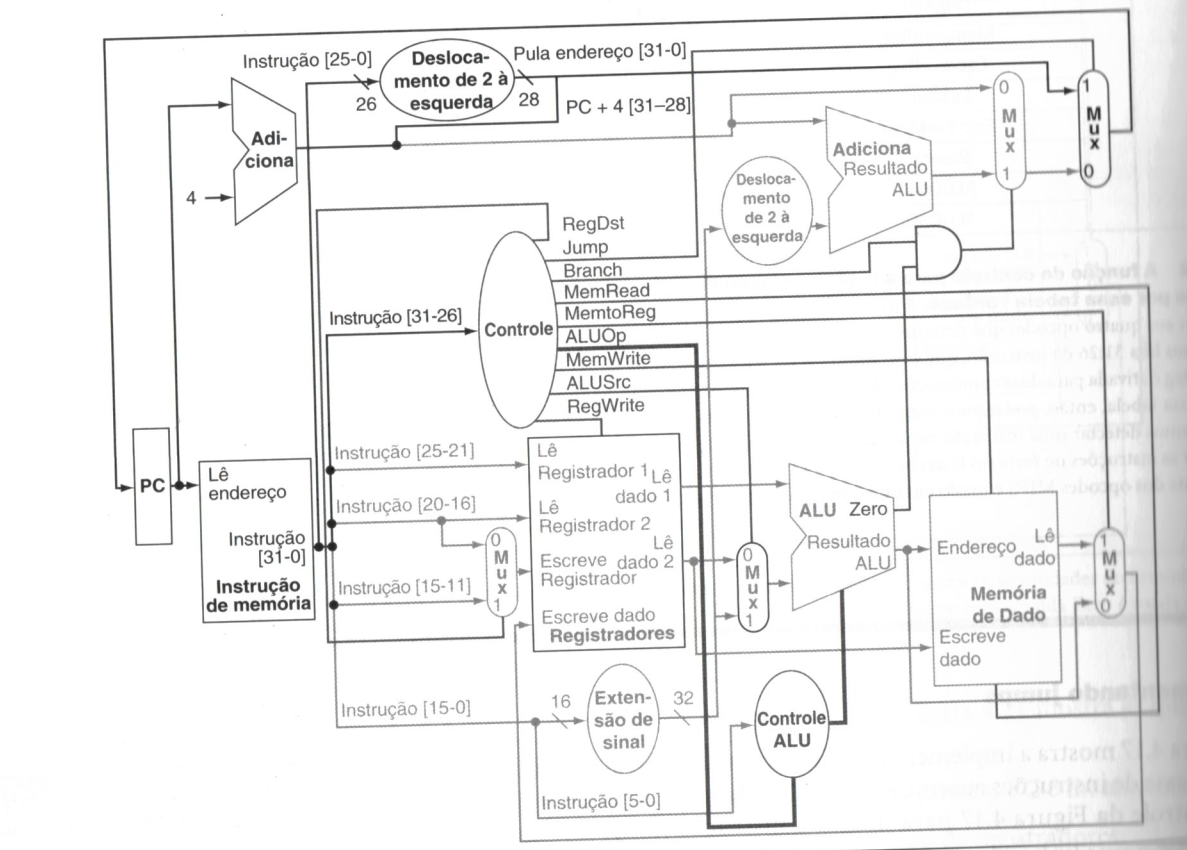
### Memórias de dados

A memória de dados será fornecida no Portal Didático. Ela consiste em um vetor de 256 palavras cada uma de 32 bits. As instruções devem ser colocadas em binário no vetor memória e você deve ajustar seu tamanho de acordo com o número de instruções que você utilizar em seu programa. Para gerar o programa, utilize o simulador MARS. Depois de implementar o código, acesse o menu Run -> Assemble. Depois disso, acesse o menu File -> Dump memory, selecione .text em Memory segment e Binary Text em Dump Format. Para finalizar, clique em Dump to file e salve o arquivo.

### Caminho de dados do MIPS

O caminho de dados do MIPS deve ser implementado por meio do port map de cada um dos componentes anteriormente descritos segundo a figura a seguir.

## Documentação



Além do projeto do ISE contendo todos os módulos implementados (arquivos .vhd), cada grupo deverá entregar também uma documentação que contenha as seguintes informações:

* Função do módulo
* Lista e descrição das entradas e saídas do módulo

Além disso, deve ser preenchida a tabela a seguir com a identificação do nome do módulo. Caso o grupo tenha implementado mais módulos que os listados, estes deverão ser incluídos ao final da tabela. Caso o módulo não tenha sido implementado, deixe os campos correspondentes em branco.

Ao final da documentação comente as principais dificuldades na realização do trabalho e o que o grupo achou de mais interessante em sua execução.

|  |  |  |
| --- | --- | --- |
| Módulo | Nome do arquivo VHDL | Aluno(s) responsável(eis) pela implementação |
| PC |  | FEITO PELOS 3 |
| Flip flop D com Clear |  | FEITO PELOS 3 |
| Flip flop D com Preset |  | FEITO PELOS 3 |
| Banco de registradores |  | FEITO PELOS 3 |
| Decodificador 5:32 |  | FEITO PELOS 3 |
| Registrador de 32 bits |  | FEITO PELOS 3 |
| Multiplexador de 32 entradas de 32 bits |  | FEITO PELOS 3 |
| Multiplexador de 2 entradas de 32 bits |  | FEITO PELOS 3 |
| Multiplexador de 2 entradas de 5 bits |  | FEITO PELOS 3 |
| ULA de 32 bits |  |  |
| ULA de 1 bit |  |  |
| OR de 2 entradas |  | FEITO PELOS 3 |
| OR de 32 entradas |  |  |
| Multiplexador de 2 entradas de 1 bit |  | FEITO PELOS 3 |
| Multiplexador de 4 entradas de 1 bit |  | FEITO PELOS 3 |
| Somador completo |  | FEITO PELOS 3 |
| Somador de 32 bits |  | FEITO PELOS 3 |
| Extensor de sinal |  | FEITO PELOS 3 |
| Unidade de controle |  | FEITO PELOS 3 |
| Unidade de controle da ULA |  |  |
| Caminho de dados do MIPS |  |  |