ESPECIFICAÇÃO DO PROJETO DE PROCESSADOR DE 16 BITS

O projeto de processadores é uma das etapas no projeto de circuitos específicos de aplicação. O projeto de processadores é composto, basicamente, por três etapas:

- Projeto das Instruções
- Projeto da Unidade de Controle
- Projeto do *Datapath*

Para nosso projeto, o processador processará palavras de 16 bits, dessa forma as etapas mencionadas devem levar em consideração essa restrição para o formato binário das instruções e as larguras de dados. A memória será contínua e dividida em área de dados e área de programa, a quantidade de palavras nas memórias fica a cargo do projeto desde que essas sejam de 16 bits.

O projeto das instruções consiste no projeto do conjunto de instruções que um processador deve suportar em seu funcionamento. Através do conjunto de instruções é possível extrair informações para as duas próximas etapas do projeto.

Para nosso processador as seguintes instruções devem ser implementadas:

mnemônico:	**`jumpeq, jumpneq, jumpgt, jumpgte, jumplt, jumplte`**
descrição:	Consistem em instruções de desvio de fluxo condicionais capazes de implementar laços e Ifs.
formato:	`jumpeq` RegNUM1, RegNUM2, CTE
semântica:	Desvia o fluxo para a posição de memória dada em CTE caso o valor contido no registrador RegNUM1 seja igual ao valor do Registrador RegNUM2.

mnemônico:	**`jump`**
descrição:	Instrução de desvio incondicional de fluxo capaz de implementar goto.
formato:	`jump` CTE
semântica:	Desvia o fluxo para a posição de memória dada em CTE.

mnemônico:	**`add, sub, addi, subi`**
descrição:	Instruções de adição e subtração.
formato:	`add` RegNUM1, RegNUM2, RegNUM3 `addi` RegNUM1, CTE, RegNUM3
semântica:	Adiciona o valor de registradores RegNUM1 e RegNUM2 ou RegNUM1 e CTE guardando o valor resultante em REGNUM3.

mnemônico:	**`shfl, shfr`**
descrição:	Instruções de deslocamento de bits para esquerda (shfl) ou para direita (shfr).
formato:	`shfl` RegNUM1, CTE, RegNUM2
semântica:	A instrução `shf` faz o deslocamento de bit do valor no RegNUM1, CTE vezes, armazenando o valor em RegNUM2.

mnemônico:	**`st`**
descrição:	instrução que guarda um valor dado por um registrador na memória.
formato:	`st` RegNUM1, RegNUM2
semântica:	guarda o valor do registrador RegNUM1 no endereço de memória indicado pelo valor do registrador RegNUM2.

mnemônico:	**`ld`**
descrição:	instrução que carrega um valor da memória em um dado registrador.
formato:	`ld` RegNUM2, RegNUM1
semântica:	carrega o valor do endereço da memória indicado pelo valor do registrador RegNUM2 no registrador RegNUM1.

mnemônico:	**`nop`**
descrição:	instrução *filler*, processador não executa operações nessa instrução
formato:	`nop`
semântica:	

O projeto da unidade de controle leva em conta o conjunto de instruções para especificar um formato binário que o represente. Por exemplo, nosso processador tem 16 instruções, portanto, 4 bits são suficientes para identificar univocamente todas elas. Dessa forma, a representação binária da instrução `nop` poderia ser '0000' que será o valor que deve estar guardado na memória de programa para que a unidade de controle identifique que a operação a ser executada é aquela que corresponde ao `nop`. No entanto, como nosso processador é de 16 bits, a instrução `nop` precisa ser representada com todos os bits, ficando '0000 0000 0000 0000', por isso, é comum que os primeiros n-bits das palavras correspondam ao código de instrução, onde `n` é a quantidade de bits suficiente para representar as mesmas.

Outras instruções podem não ser tão simples, a instrução `add`, por exemplo deve conter seu código de instrução, digamos '0001', além dos operandos. Suponha que no nosso caso temos 4 registradores de propósito geral, dessa forma 2 bits são suficientes para endereçá-los. Assim se queremos adicionar R1 com R2, guardando o resultado em R3, podemos escrever na memória de programa o valor: '0001 0001 0010 0011'.

Assim o projeto da unidade de controle consiste em mapear as instruções para formatos binários que possam servir como entrada que a Unidade de Controle usa para orquestrar o fluxo de dados do processador de forma a executar a instrução desejada. A unidade de controle também se encarrega de incrementar o PC após cada instrução, além de buscar a próxima instrução, na memória de programa, apontada pelo PC.

Por fim o projeto do *Datapath*, leva em conta as características que podemos explorar no processador, ficando a cargo do projetista, cruzar os dados extraídos das instruções com o custo de desenvolvimento visando maximizar as funcionalidades enquanto reduz o custo.

Neste projeto o *Datapath* pode ser o mínimo possível para executar as instruções listadas, assim menos trabalho de teste é necessário.

Além da especificação acima, o processador projetado deve ser capaz de acessar alguns GPIOs da placa através de mapeamento em memória. Essa técnica mapeia parte da memória acessível ao processador como elementos de acesso aos GPIOs.

Por exemplo, o endereço de memória 0xFF0001, pode ser mapeado para o GPIO 1, o endereço 0xFF0002 pode ser mapeado para o GPIO 2 e, assim por diante. Nesse caso, apenas o último bit nos 16 bits dos endereços 0xFF0001 e 0xFF0002 serão considerados para o valor do GPIO.

Para esse processador faremos GPIOs exclusivamente de entrada e exclusivamente de saída. Assim, o processador deve contar com, no mínimo, 8 GPIOs de entrada e 8 GPIOs de saída, que devem ser mapeados na memória a critério do projetista.

Os GPIOs serão usados, basicamente, para testar o processador através de dois programas simples:

Led Blink

```
main (){
      int i;
      while(true){
      // o objetivo aqui é que o led fique piscando,
      // aumentem/diminuam o valor 50000000 a critério de vocês
      for (i = 0; i< 50000000; i++);
      GPIO1 = 1;
      for (i = 0; i< 50000000; i++);
      GPIO1 = 0;
      }
}
```

Led Toogle

```
main (){
      int i;
      while(true){
      if(GPIO8 == 1)
            GPIO1 = 1;
      else
            GPIO1 = 0;
      }
}
```

A programação do processador pode ser feita através da inicialização estática da memória de dados/programa. No entanto, essa inicialização deve ser feita através de código binário. É aconselhável que coloquem comentários na inicialização da memória, descrevendo quais instruções/valores estão sendo escritos.

Embora as demais instruções não sejam testadas nos programas citados à cima, façam *test bench* para mostrar a execução de todas as instruções requeridas a fim de provar o funcionamento.