µProcessador 5 Condicionais e Desvios

Incluir no circuito instruções de *jump* (com endereço de salto absoluto) e *branch* (com endereço de salto relativo). Obviamente há várias formas de fazer isso.

Não há restrições para os circuitos: podem colocar uma ULA adicional, reutilizar a que está presente ou colocar um comparador dedicado. Ou então pode colocar uma *flag* (sinalizador, é um flip-flop independente que guarda uma condição). Veja como funciona uma comparação no 8051:

```
        CMP A,32      ; compara o reg. A com o número 32, setando a flag Z se iguais
        JZ IGUAL      ; salta para IGUAL apenas se a flag Z está setada
DIF:    NOP           ; entra nesta linha só se A ≠ 32
```

Para maior e menor é usada outra *flag* (C, de *carry)* e instrução JC; as negativas (JNZ e JNC) também estão presentes.

Implementação Mínima

Note que o jump deve usar um **endereço absoluto**, o que quer dizer que "jump 34" vai pular para o endereço 34 na memória. O branch, por outro lado, deve usar um **endereço relativo**, o que significa que um "branch 5" vai pular cinco instruções pra frente de onde ele está. Atenção: o relativo tem que pular tanto para frente como para trás ("branch -5" volta cinco instruções).

A exigência desta tarefa é fazer apenas uma condicional (pode ser só *beq* ou *bne*, por exemplo) e uma incondicional (o nosso *j* do MIPS), mas podem fazer mais se quiserem. Um meio qualquer de detectar qual o maior de dois números deve ser provido, mesmo que gaste várias instruções. Pode alterar sua ULA se precisar.

Notas

- A execução do programa deverá começar obrigatoriamente pela instrução no endereço zero **(Importante!)**
- Não é necessário finalizar formalmente o programa. Podem deixar rodando randomicamente, ou fazer um loop infinito, ou encher de *nops* (se houver a instrução *nop).* Mais elegante, pra quem quiser: criar uma instrução *halt* que congela o processador.
- Usualmente o endereço destino de um *branch* relativo é calculado a partir do endereço da instrução seguinte: um *beq $0,$0,30* provavelmente pula para PC+1+30 neste projeto. Mas não há problema em se fazer a partir do endereço do próprio *branch* (isto é, pular para PC+30).

Sugestões de Debug

Talvez seja uma boa ideia colocar como pinos no *top-level* tanto o valor do *Program Counter* quanto a saída da ROM (ou do Registrador de Instrução, se você usou), pra ajudar a verificação no *Waveform Simulator*. Pense num jeito em que se possa acompanhar a execução dos programas com facilidade na tela de simulação.

Sugiro que produzam uma tabela com o esquema de codificação de todas as instruções, de forma clara (isto é, que daqui a um ano você olhe e entenda tudo sem precisar olhar no circuito). Isso inclui largura de registradores, tamanho da palavra da memória, formatos de instrução...

Testem bem este circuito. Ele às vezes dá uns bugs safados de perceber com um testezinho mixuruca apenas. E são pentelhos de resolver.

Especial: o Mundo das Gambiarras

Neste laboratório, devido ao acréscimo de instruções que devem ser decodificadas, o circuito ganha uma complexidade inesperada. Os bits da instrução correm o circuito inteiro, entrando em

diversas portas lógicas que geram sinais intermediários usados para gerar os *enables* do resto do circuito.

Sugestões camaradas:

- *nomeie* adequadamente os sinais, de forma explícita;
- *renomeie* os bits da instrução (use o componente WIRE se for o caso) para clarear;
- faça conexões de forma organizada espacialmente;
- use retângulos e círculos (*Rectangle Tool* e *Oval Tool*, na barra de ferramentas) para delimitar subcircuitos importantes;
- coloque trechos de texto (sim, você adivinhou: com *Text Tool*) para documentar no esquemático itens de lógica, descrição de funcionalidade, *tabelas de decodificação* (tabelas verdade) e, *principalmente,* coisas como "gambiarra para atrasar meio clock e evitar erro de transitório";
- nomeie *sinais intermediários* da unidade de controle (como por exemplo um fio cujo sinal identifica "instrução no formato Imediato" quando em 1);
- use *cores diferentes* para identificar barramentos ou sinais específicos! Clock, instrução e sinais de controle são bons candidatos. (Clica no fio e vai no propriedades).

Eu não me importo tanto se o esquemático estiver parecendo macarronada de domingo, mas *vocês* vão sentir na pele na hora de debugar e alterar. Acreditem.

Entrega Eletrônica

O prazo para entrega é na 6a feira daqui a duas semanas às 13h00, mas *semana que vem teremos outra tarefa* (a RAM), então se espertem.

Sugiro fortemente escrever no papel programas de teste, detalhando em cada linha a instrução em assembly e a codificação hexadecimal e endereço na memória respectivos. Ajuda quando dá pau.

Os dois arquivos abaixo devem ser atualizados conforme a descrição.

- "**programa.mif**": contém o programa em ROM em que um registrador x qualquer deve começar com um valor 2. O programa deverá ser um simples *loop* que repete três vezes a acumulação deste número, ou seja, calcula x*3. Os valores devem ser enviados ao pino *out* no começo, a cada iteração e no final.
- "**opers.txt**": deve conter os seguintes nove trechos de opcodes hexadecimais para o seu processador *na ordem em que estão listados,* com um opcode por linha. Os trechos devem ser separados por linhas em branco.

1. nop
2. A <= 5
3. A <= A-1
4. salta oito endereços para trás se A for diferente de zero
5. salta sete endereços para frente se A for diferente de zero
6. salta para o endereço oito, incondicional
7. B <= 0
8. B <= B+2
9. se A>B, zera A

onde A e B representam dois registradores do seu processador à sua escolha. Mantenha a consistência: se A é R1, é sempre R1; se B é $k9, vai ser sempre $k9.

Use novamente o padrão do laboratório passado. Um arquivo pode começar assim:

```
00

03
05

E2
00 chuncho pra zerar r3
```

```
05
02
E2 subtrai
32

F8 bnz a,-8
...
```

Pode colocar comentários pra si mesmo após o opcode desde que deixe um espaço entre eles, e comentários após o final do código, deixando uma linha em branco depois do último opcode.

Checklist de Entrega

Circuitaria:

- ✔ Não há pinos de entrada deixados em aberto (i.e., desconectados)
- ✔ Todos os flip-flops/contadores possuem pino de RESET
- ✔ Não há acentuação ou espaços nos componentes
- ✔ Não há acentuação ou espaços nos nomes dos arquivos nem no caminho das pastas

Arquivos anexados no email:

- ✔ O arquivo de projeto compactado **.qar** *atualizado*
- ✔ O **projeto.txt** com as configurações *atualizadas*
- ✔ O **opers.txt** como descrito anteriormente
- ✔ Os códigos de ROM para realizar o cálculo pedido, **programa.mif**
- ✔ Os testes **.vwf** que vocês utilizaram, para me ajudar caso preciso
- ✔ Os outros arquivos **.mif** com os programas testados nos .vwf