Projeto de circuito

MAC0329 – Álgebra booleana e aplicações (DCC / IME-USP — 2019)

– Todas as etapas do projeto deverão ser feitas com o *Logisim* (http://www.cburch.com/logisim/) –

– O projeto poderá ser feito em grupo de até três pessoas –

Parte 3: Processador Vzero – entrega no PACA, até 11/06

O objetivo deste EP é integrar vários dos componentes que fazem parte de um processador e implementar o ciclo de execução de modo que ele seja capaz de executar algumas poucas instruções específicas. No próximo EP, o repertório de instruções será ampliado para que seja possível a execução de um programa completo.

1 Componentes de um processador

Os principais **componentes de um processador** são a unidade central de processamento (CPU) – formada pela unidade de controle (UC) e a unidade lógico-aritmética (ULA) –, e a memória (RAM). Abaixo está uma breve descrição destes e de outros componentes do processador que iremos construir.

ULA: a ULA da nossa CPU será capaz de efetuar as operações de adição e subtração e comparações de números inteiros sem sinal.

Memória RAM: Os endereços serão de 8 *bits* (portanto 256 posições) e cada posição é formada por uma palavra de 16 *bits*. Cada posição da RAM pode armazenar uma instrução ou um dado (número). Se for uma instrução, o *byte* mais significativo conterá o código de uma instrução e o *byte* menos significativo poderá conter o endereço de uma posição de memória. Se for um dado, apenas os 8 *bits* menos significativos serão considerados e interpretados como número sem sinal.

UC: a unidade de controle é a responsável por controlar a execução das instruções. Dentre as principais tarefas está a de decodificar uma instrução (mais detalhes abaixo).

Registradores: Serão utilizados os seguinte registradores.

- **AC (Acumulador):** o acumulador é um registrador de 8 *bits* utilizado para o armazenamento temporário de dados durante a execução de algumas instruções.
- **IR (registrador de instrução):** A instrução a ser executada encontra-se na RAM e deve ser copiada para o IR antes de ser executada. O IR deverá ter 16 *bits*.
- **PC (program counter):** PC é um contador (também denominado apontador de instruções). Seu valor deve ser o endereço da posição de memória que contém a próxima instrução a ser executada. No início da simulação, o seu valor deve ser zero. O PC é um contador de 8 *bits*.

***Clock*:** o papel do *clock* é a sincronização da mudança de estados (memória RAM, registradores e contadores, basicamente).

Outros: outros componentes como seletores (MUX), distribuidores (DMUX), ou *buffers* controlados serão necessários para garantir o correto tráfego dos dados. Pode-se também usar um *display* para mostrar valores de pontos estratégicos do processador.

2 Ciclo de instrução

Toda CPU executa ciclos de instrução (em inglês, *fetch-decode-execute cycle* ou FDX) de forma contínua e sequencial, desde o momento em que o computador é inicializado até quando ele é desligado.

Um **ciclo de instrução** consiste dos três passos a seguir e cabe à UC acertar os sinais de controle para que a execução ocorra corretamente.

1. *Fetch*: busca-se uma instrução que está na memória RAM

   - a instrução na posição apontada pelo PC deve ser lida da memória e carregada no IR. Além disso, o valor do PC deve ser incrementado em 1.

2. *Decode*: decodifica-se a instrução (determina-se as ações exigidas pela mesma)

   - o valor dos diversos sinais de controle deve ser ajustado conforme a instrução a ser executada (por exemplo, pode ser necessário definir o modo de operação – leitura/escrita – da memória e dos registradores, os sinais que controlam os pinos seletores dos MUXes, etc).

3. *Execute*: executa-se as ações determinadas pela instrução

   - Além disso, o ciclo deve ser "resetado".

Um ciclo de instrução é tipicamente executado em um número fixo de períodos do *clock*. No nosso modelo simplificado, dos passos acima, apenas o passo 1 (*fetch*) e o passo 3 (*execute*) envolvem atualização de memória ou registrador.

Antes do primeiro pulso do *clock* deve-se garantir que todos os sinais de controle, assim como os endereços pertinentes, estão ajustados adequadamente para que a próxima instrução seja lida da memória e armazenada no IR. No primeiro pulso do *clock* o **passo 1** (*fetch*) é executado. O **passo 2** do ciclo de instrução (*decode*) depende da instrução a ser executada. Os sinais de controle devem ser ajustados de acordo com a instrução (8 *bits* mais significativos do IR). Para fazer essa parte, deve-se construir um circuito combinacional que ativa/desativa as *flags* relevantes de acordo com a instrução sendo decodificada. Note que esse passo não requer um pulso do *clock*. Ele será executado assim que uma instrução for carregada no IR. O **passo 3** é executado com um segundo pulso do *clock* e corresponde à execução propriamente dita da instrução. Após a execução desse passo, o circuito deve voltar à configuração do início do ciclo de instrução, pronto para a execução da próxima instrução. Portanto, um ciclo de instrução na nossa CPU pode ser implementado de forma a ser completado em dois pulsos do *clock*.

3 Tarefa, entrega e avaliação

Implementar um processador, que seja capaz de executar as três seguintes instruções:

Código na base 16	Descrição
0x00--	NOP (no operation)
0x01EE	Copie [EE] para o AC
0x02EE	Copie [AC] para a posição de endereço EE

[EE] significa o conteúdo na posição de endereço EE na RAM

[AC] significa o conteúdo do AC

Note que o AC tem 8 *bits* enquanto uma posição na RAM tem 16 *bits*.

Entregar via PACA um arquivo `cpuV0.circ`, contendo o processador descrito acima. Desta vez, a organização do circuito fica por conta de vocês. Uma dica é o uso de túneis (um dos items disponíveis no Logisim) que permite conectar logicamente dois extremos de fios (mesmo eles não estando diretamente conectados no desenho).

Serão **avaliados os seguintes aspectos**:

- corretude dos circuitos
- clareza na organização do circuito (se desejar, entregue um documento à parte, com as informações que ajudem a entender a organização e lógica dos circuitos)
- cumprimento do prazo

Neste EP podem ser usados os componentes disponíveis no `Logisim`. Para simular o circuito, escreva as instruções nas posições 0x00 a 0x02 da memória RAM, gere os pulsos do *clock* manualmente (basta clicar sobre ele para mudar o valor), e certifique-se que as mudanças de estado corretas estão ocorrendo. Use endereços distintos para as instruções 0x01EE e 0x02EE (por exemplo EE=0x07 e EE=0x08).

3.1 Exemplo

Suponha que a instrução a ser executada está no endereço 0x00 da memória RAM e que trata-se da instrução "Copie o número que está na posição 0x28 para o acumulador". De acordo com a tabela acima o código da instrução é 0x0128. No primeiro pulso, a instrução que está na posição 0x00 da memória RAM deve ser carregada para o IR. Este processo corresponde ao passo *fetch*. No segundo pulso, o número que está na posição 0x28 da RAM deve ser carregado no AC. Este processo corresponde ao passo *execute*. Entre o primeiro e o segundo pulsos ocorre o passo *decode*.

Como deve estar o estado do circuito antes do primeiro pulso do *clock*:

- o PC deve estar com valor 0x00
- o valor do PC deve estar disponível na entrada de endereço da RAM
- a RAM deve estar em modo de leitura (*read*)
- os *flags* devem ser ajustados de forma que a saída da RAM esteja disponível na entrada do IR

Um pulso fará com que a instrução seja carregada no IR. A UC, no passo *decode*, deve tratar de acertar as *flags* para a execução da instrução. Assim, antes do segundo pulso do *clock*, o estado do circuito deve satisfazer:

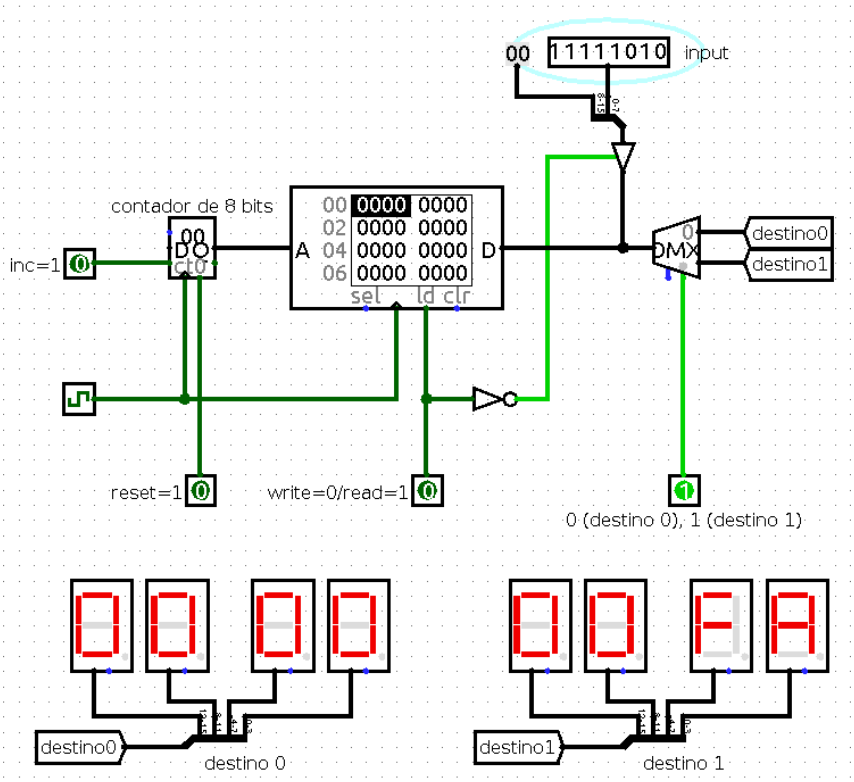
- o PC deve estar com valor 0x01
- o valor 0x28 (vindo do IR) deve estar disponível na entrada de endereço da RAM
- a RAM deve estar em modo de leitura (read)

- os *flags* devem ser ajustados de forma que a saída da RAM esteja disponível na entrada do AC

Um pulso do *clock* fará com que o dado do endereço 0x28 seja carregado no AC. Além disso, deve-se preparar o circuito para iniciar o próximo ciclo de execução.

4 Apêndice

Para servir de ponto de partida, vocês podem reproduzir o circuito abaixo que inclui uma RAM, *clock*, contador, e um DMUX, e explorar o seu funcionamento.



Procure no *Help* do Logisim a documentação sobre os diferentes pinos de cada componente para entender como esse circuito funciona.

Note que as linhas que conectam os diversos componentes transportam *bits*. Essas linhas são chamadas de barramentos e possuem largura (em *bits*) que depende do tipo da informação transportada. No nosso exemplo, a linha conectando a porta `D` da RAM com o `input` e com o DMUX é um **barramento de dados** de largura 16 *bits*. Já a linha ligando a saída do contador à porta `A` da RAM é um **barramento de endereço** de 8 *bits*. As linhas ligadas aos pinos de controle são os **barramentos de controle**, no caso todos de largura 1 *bit*. (Note que o AC e a ULA trabalham com palavras de 8 *bits* e portanto a largura das palavras (de dados) deve ser ajustada adequadamente.)

O contador no exemplo acima pode ser incrementado a cada pulso do *clock* quando inc=1. Há também um outro pino `load` para permitir o carregamento de um valor de forma assíncrona (será útil para as instruções de desvio, na próxima etapa do EP). Para detalhes, veja a documentação do Logisim.

Postem as dúvidas no Fórum de discusssões no PACA.