MAC0329 – Álgebra booleana e circuitos digitais

DCC / IME-USP — Primeiro semestre de 2017

Projeto de circuito 2 – CPU/MAC0329

Data de entrega: até 30/06/2017

Neste projeto, o objetivo é a construção do circuito de um processador. No primeiro projeto construímos uma ULA. Agora iremos acrescentar as demais partes. O circuito final deverá ser capaz de executar uma sequência de instruções armazenadas na memória (RAM). As especificações encontram-se mais adiante.

Para o desenvolvimento do projeto deve ser usado o software `Logisim` (http://www.cburch.com/logisim/).

O trabalho deve ser desenvolvido em grupo, mantendo-se os mesmos membros do projeto anterior. Caso o projeto anterior tenha sido desenvolvido individualmente ou em grupo de dois membros, neste projeto o arranjo pode ser alterado para se formar grupos de 3 membros. O trabalho pode ser dividido entre os membros, mas todos devem ter ciência sobre os detalhes do projeto.

A entrega do projeto será similar ao do projeto anterior:

- entrega via PACA
- entregar um arquivo `.circ`, criado com o `Logisim`
- entregar um relatório contendo
  - nome dos membros do grupo
  - descrição sucinta e clara de como está organizado o circuito (partes principais e como se relacionam)
  - descrição sucinta e clara sobre como cada parte foi feita e por que ela foi feita
  - descrição de dificuldades enfrentadas
  - instruções sobre como simular o circuito (é preciso "setar" alguma parte do circuito? O que deve ser clicado? Etc.)
- Os dois arquivos devem ser empacotados em um único arquivo (`.zip` ou `.tar.gz` ou `tgz`).
- basta que apenas um dos membros do grupo faça a entrega.

Postem suas dúvidas ou descobertas no Fórum da disciplina.

---

1 Formato dos dados

No nosso processador consideraremos instruções de 8 *bits*, endereçamento também de 8 *bits* e dados (números em notação complemento de dois) de 16 *bits*. Assim, a memória poderá ter até 256 posições, com endereços de 0 a 255, e cada posição consistindo de uma palavra de 16 *bits*.

2 Conjunto de instruções

Adotaremos parte do conjunto de instruções do HIPO (o computador hipotético), o qual simulamos em sala de aula. Mais informações sobre o HIPO podem ser encontrados por exemplo em www.ime.usp.br/~jstern/miscellanea/MaterialDidatico/hipo.htm.

No HIPO, os códigos de instrução, os endereços e os dados são representados na base 10. Porém, uma vez que os valores armazenados na memória do `Logisim` são exibidos em hexadecimal, na tabela abaixo mostramos o código das instruções do HIPO em hexadecimal (base 16). AC refere-se ao acumulador (um registrador), EE refere-se a um endereço qualquer e [EE] ao conteúdo de EE (i.e., o dado armazenado na posição de memória cujo endereço é EE).

Tabela de instruções do HIPO

Código		Descrição
base 10	base 16	
11	0B	Copie [EE] no AC
12	0C	Copie [AC] no endereço EE
21	15	Some [EE] com [AC] e guarde o resultado no AC
22	16	Subtraia [EE] de [AC] e guarde o resultado no AC
31	1F	Leia um número e guarde-o no endereço EE
41	29	Imprima [EE]
51	33	Desvio não condicional
56	38	Desvie se [AC] < 0
70	46	Pare
23	17	Multiplique [AC] por [EE] e guarde o resultado no AC
24	18	Divida [AC] por [EE] e guarde o resultado no AC
50	32	NOP (*no operation*)
52	34	Desvie se [AC] ≤ 0
53	35	Desvie se [AC] $\neq 0$
54	36	Desvie se [AC] > 0
55	37	Desvie se [AC] $= 0$
57	38	Desvie se [AC] ≥ 0

As 9 primeiras instruções devem ser implementadas. As demais são opcionais. Caso opte por implementar a multiplicação e a divisão, use os circuitos multiplicador e divisor disponíveis no `Logisim`.

3 Componentes

Os principais componentes de um computador são a unidade central de processamento (CPU), formada pela unidade de controle (UC) e a unidade lógico-aritmética (ULA), e a memória (RAM). A CPU utiliza memórias especiais, denominadas registradores, na execução das instruções. Abaixo está uma breve descrição de alguns componentes que deverão fazer parte do seu circuito.

ULA: Na nossa CPU, a ULA deve efetuar as mesmas operações especificadas no projeto1. Ela deve ser alterada para operar com dados de 16 *bits* (veja observação 1).

Memória RAM: Os endereços serão de 8 bits (256 posições) e cada posição corresponderá a uma palavra de 16 bits. Cada posição da memória 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 conterá o endereço de uma posição de memória. Se for um dado, deverá ser interpretado como um número de 16 *bits*, na notação complemento de dois.

UC: a unidade de controle é a responsável por controlar a execução das instruções.

Registradores: Os seguintes registradores serão necessários.

ACC (Acumulador): o acumulador é um registrador utilizado para o armazenamento temporário de dados durante a execução de instruções. Deverá ter 16 *bits*.

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. Deve haver um sinal "inc" que faz o seu valor ser incrementando em 1 sempre que pertinente. Deve haver também um comando *load* que carrega um certo valor de forma assíncrona (será útil para as instruções de desvio). PC será um contador de 8 *bits*.

Outros: outros componentes como seletores, distribuidores ou *buffers* controlados serão necessários para garantir o correto tráfego dos dados e sinais de controle.

4 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 o mesmo ser desligado. Aliás, a CPU só faz isso!

A UC é responsável por controlar a execução dos ciclos de instrução. Um ciclo de instrução consiste dos três passos a seguir:

1. Busca de uma instrução na memória (*fetch*)
2. Decodificação da instrução (determinar as ações exigidas pela mesma)
3. execução das ações determinadas pela instrução

Especificamente, a UC deve:

1. no passo 1, ler da memória a instrução na posição apontada pelo PC e carregá-la no IR. Além disso, deve incrementar o valor de PC.
2. no passo 2, deve definir o modo de operação (leitura/escrita) da memória e dos registradores e acertar todas as *flags* de controle com valores pertinentes.
3. no passo 3 ocorre a execução da instrução. Além disso, o ciclo deve ser "resetado".

Um ciclo de instrução é tipicamente executada em um número fixo de ciclos do *clock*. Lembre-se que o estado das partes sequenciais do circuito (isto é RAM e registradores) é atualizado apenas num pulso do *clock*. Por outro lado, as partes combinacionais do circuito alteram-se imediatamente após a alteração de suas entradas.

No nosso modelo simplificado, dos passos acima, apenas o passo 1 (*fetch*) e o passo 3 (execução) envolvem atualização de memória ou registrador. Logo, um ciclo de instrução na nossa CPU pode ser implementado de forma a ser completado em dois ciclos do *clock*. Antes do primeiro pulso do *clock* deve-se garantir que todos os sinais de controle, assim como os dados pertinentes, estão ajustados

adequadamente para que a próxima instrução seja buscada e armazenada no IR. O passo 2 do ciclo de instrução (decodificação) depende da instrução a ser executada. Os sinais de controle deve ser ajustados de acordo com a instrução. Para fazer essa parte, convém projetarmos um circuito combinacional que ativa/desativa as *flags* pertinentes de acordo com a instrução sendo decodificada. Note que esse passo não requer um pulso do *clock*. Ele será executado assim que a instrução for carregada no IR.

O terceiro passo 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.

5 Simulação

Para que o funcionamento da CPU possa ser simulado, adicione um pino *reset* que deve ser ativado para colocar a CPU num estado inicial, pronto para a simulação. Sua CPU deve também ter um *clock*, para coordenar os passos de execução.

O programa a ser simulado pode ser carregado para a RAM a partir de um arquivo. Pode também ser digitado posição por posição.

Os dados de entrada (comando de leitura) podem ser disponibilizados (para a CPU) por meio de pinos de entrada (16 bits). Quando da execução da instrução de leitura, deve-se garantir que o dado de entrada está disponível no pino de entrada antes da execução propriamente dita.

Quanto à saída (comando de impressão), ela pode ser enviada para pinos de saída. Se vocês conseguirem fazer um terminal para mostrar a saída das impressões, compartilhem no fórum.

6 Exemplo de um programa

Um exemplo de programa (sequência de instruções do) HIPO, que inclui apenas as instruções de implementação obrigatória, pode ser encontrado no arquivo `programa1`. Este pode ser carregado na RAM, antes da simulação.

Abaixo está a "tradução" do programa. Todos os endereços estão em notação hexadecimal.

Endereço	Instrução	Tradução
00	0b 1e	Copie [1e] no AC
01	0c 28	Copie [AC] no endereço 28
02	1f 2d	Leia num e copie no endereço 2d
03	29 2d	Imprima [2d]
04	0b 2d	Copie [2d] no AC
05	38 0b	se [AC] < 0 desvie para o endereço 0a
06	0b 28	Copie [28] no AC
07	15 2d	Some [AC] com [2d] e copie o resultado em AC
08	0c 28	Copie [AC] no endreço 28
09	33 03	Desvie para o endereço 02
0a	29 28	Imprima [28]
0b	46 00	Pare

7 Observações e dicas

1. Os dados em nosso computador são de 16 bits. Para aproveitarmos a ULA já feita no projeto1, podemos considerar que apenas o *byte* menos significativo dos dados são passados para a ULA. Para manter a consistência, à saída da ULA deverá ser completada com 8 *bits* iguais a zero. Porém, se possível a ULA deve ser alterada para que ela opere com dados de 16 *bits*.

   No relatório final você deve indicar de forma clara se a ULA opera com 8 ou 16 *bits*. Em ambos os casos, deverá ser considerada a notação complemento de dois (i.e., no caso de 8 *bits*, os números válidos são aqueles no intervalo de -128 a +127. No caso de 16 *bits*, são números no intervalo de -2^{15} a $+2^{15} - 1$).

2. Podem ser utilizados todos os componentes disponíveis no `Logisim`, exceto o somador/subtrator. Podem ser utilizados RAM, registradores, contadores, *clock*, portas lógicas, multiplexadores, ...

   Leia o manual e entenda como funciona o componente a ser usado.

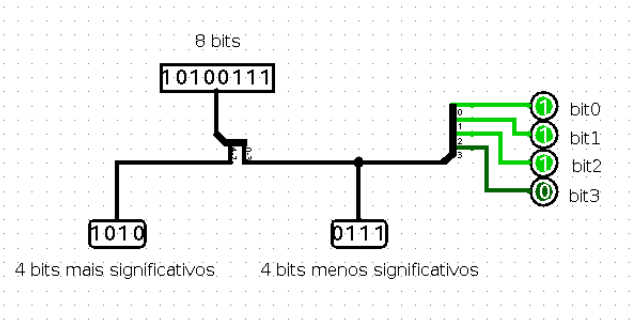
   Acrescente comentários no circuito se os mesmos facilitarem o entendimento.

3. Planeje a organização do circuito antes de começar a desenhá-lo no `Logisim`.

   Uma parte importante do circuito são os sinais de controle. Liste os sinais de controle do seu circuito e analise quando eles devem estar ativos/inativos.

4. Em certas partes do circuito pode ser conveniente utilizar

   bits de dados "largos", assim como os *splitters*. A figura a seguir ilustra um pino de entrada com 8 *bits* e o uso de *splitters* para separar (sub)grupos de *bits*.



   Outro item útil são os túneis (para evitar ligação explícita por meio de linhas). Use rótulos auto-explicativos para os túneis.

Dúvidas? Poste suas dúvidas no Fórum de discussão no PACA ou traga-as para a sala de aula.