MAC0329 – Álgebra booleana e aplicações

DCC / IME-USP — Primeiro semestre de 2018

Projeto de circuito 3 – enunciado completo

O objetivo do EP3 é construir o circuito de uma CPU simples, inspirado no HIPO (https://www.ime.usp.br/~jstern/software/hipo/Hipo.pdf). No EP3a foram trabalhados alguns dos componentes desse circuito. Neste enunciado encontra-se a descrição completa do EP3. A entrega é dividida em duas partes:

1. **EP3b** – circuito que implementa um ciclo de instrução da CPU, testado com duas instruções
2. **EP3c** – circuito da CPU final, com todas as instruções.

O trabalho deve ser desenvolvido em grupo, mantendo-se preferencialmente os mesmos membros do EP anterior. Caso o EP anterior tenha sido desenvolvido individualmente ou em grupo de dois membros, no EP3 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.

1 Especificação da nossa CPU

No nosso processador consideraremos instruções e endereços de 8 *bits* e dados (números em notação complemento de dois) de 16 *bits* (note que a ULA que construímos no EP anterior trabalha com 8 *bits*. Desta forma, tecnicamente precisaríamos alterar a ULA para que ela opere com dados de 16 *bits*. Em relação a isso, será permitida uma de duas opções, descritas na seção 3, em "Observações Finais"). Assim, a memória terá 256 posições, com endereços de 0 a 255 (ou `00` a `FF`, em notação hexadecimal), e cada posição consiste de uma palavra de 16 *bits*.

1.1 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 Projeto 2. Veja também a observação 1 mais adiante na seção 3.

Memória RAM: Os endereços serão de 8 *bits* (portanto 256 posições) e cada posição corresponderá a 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, deverá ser interpretado como um número na notação complemento de dois.

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

Registradores: Os seguintes registradores serão necessários.

AC (Acumulador): o acumulador é um registrador utilizado para o armazenamento temporário de dados durante a execução de 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. Deve haver um sinal `inc` que faz o seu valor ser incrementando em 1 sempre que pertinente. Deve haver também um sinal `load` que carrega um certo valor de forma assíncrona (será útil para as instruções de desvio). 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, distribuidores ou *buffers* controlados serão necessários para garantir o correto tráfego dos dados e sinais de controle.

1.2 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 o mesmo ser desligado. Aliás, no nosso caso 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. *Fetch*: busca de uma instrução que está na memória RAM

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

2. *Decode*: decodificação da instrução (determinar as ações exigidas pela mesma)

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

3. *Execute*: execução das ações determinadas pela instrução

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

Um ciclo de instrução é tipicamente executado 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 (*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 buscada e armazenada no IR. 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. 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

uma 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. Portanto, um ciclo de instrução na nossa CPU pode ser implementado de forma a ser completado em dois ciclos do *clock*.

1.3 Conjunto de instruções

O conjunto de instruções da nossa CPU é inspirado no 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. No nosso caso usamos códigos próprios, que são mostrados na tabela abaixo em notação decimal e hexadecimal. 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 da nossa CPU

Código		Descrição
base 10	base 16	
01	01	Copie [EE] para o AC
02	02	Copie [AC] para a posição de endereço EE
03	03	Some [EE] com [AC] e guarde o resultado em AC
04	04	Subtraia [EE] de [AC] e guarde o resultado em AC
07	07	Leia um número e guarde-o na posição de endereço EE
08	08	Imprima [EE]
09	09	Pare
10	0A	Desvie para EE (desvio incondicional)
11	0B	Desvie para EE se [AC] > 0
13	0D	Desvie para EE se [AC] $= 0$
15	0F	Desvie para EE se [AC] < 0
00	00	NOP (no operation)
05	05	Multiplique [AC] por [EE] e guarde o resultado em AC
06	06	Divida [AC] por [EE] e guarde o resultado em AC
12	0C	Desvie para EE se [AC] ≤ 0
14	0D	Desvie para EE se [AC] $\neq 0$
16	10	Desvie para EE se [AC] ≥ 0

Note que os códigos adotados são arbitrários. Quando há uma grande quantidade de instruções, pode-se definir os códigos de forma que o circuito decodificador de instruções fique mais simples (por exemplo de forma que certos *bits* do código possam ser usados diretamente como um sinal de controle). Como temos apenas 16 instruções (o código `00` não está sendo usado), os códigos não foram definidos com essa preocupação. (As últimas 6 instruções estão separadas pois são opcionais para a entrega.)

1.4 Exemplo de um programa

Um exemplo de programa (sequência de instruções) que inclui apenas as instruções de implementação obrigatória, pode ser encontrado no arquivo `programa1`, e está transcrito abaixo.

```
0110 0211 0712 0812 0212 0F0a 0111 0312 0211 0a02 0811 0900
```

A seguir está a "tradução" do programa. Todos os endereços estão em notação hexadecimal. Note também que a posição de memória 10 deve conter o valor `00 00`.

Endereço	Instrução	Tradução
00	01 10	Copie [10] para o AC
01	02 11	Copie [AC] para a posição de endereço 11
02	07 12	Leia um número e guarde-o na posição de endereço 12
03	08 12	Imprima [12]
04	01 12	Copie [12] para o AC
05	0f 0a	desvie para 0a se [AC] < 0
06	01 11	Copie [11] para o AC
07	03 12	Some [AC] com [12] e guarde o resultado em AC
08	02 11	Copie [AC] para a posição de endereço 11
09	0a 02	Desvie para 02
0a	08 11	Imprima [11]
0b	09 00	Pare

Em linguagem um pouco mais alto nível, temos

```
10 ~ Zero
11 ~ Soma
12 ~ Num

     AC <-- Zero
     Soma <-- AC
Ler: Leia Num
     Imprima Num
     AC <-- Num
     Se AC < 0, desvie para Fim
     AC <-- Soma
     AC <-- AC + Num
     Soma <-- AC
     Desvie para Ler
Fim: Imprima Soma
     Pare
```

2 Tarefas

1. Parte (1) – **EP3b**: Projetar e implementar a CPU de acordo com as especificações acima. A sua CPU deve ser capaz de executar ao menos duas instruções:

   (a) instrução 08: Imprima [EE]

   (b) instrução 09: Pare

   Teste a sua CPU para a seguinte configuração

Endereço	Instrução	Tradução
00	08 03	Imprima [03]
01	09 00	Pare
03	00 0F	número 15 (em hexadecimal)

   A instrução `Imprima` deve simplesmente enviar o dado para um pino de saída. A instrução `Pare` deve resetar a execução do programa (não deve limpar a memória RAM).

2. Parte (2) – **EP3c**: a CPU final; ela deve ser capaz de executar todas as 11 primeiras instruções da tabela de instruções.

   Para que o funcionamento da CPU possa ser simulado, adicione um pino *reset* que deve ser ativado para colocar a CPU num estado inicial, com a RAM "zerada". Após "carregar" um programa, ela deve estar pronta para a simulação. Sua CPU deve também ter um *clock* (ele pode ser usado para simular a CPU passo-a-passo).

   O programa a ser simulado pode ser carregado para a RAM a partir de um arquivo (por exemplo, o programa dado acima e disponibilizado no arquivo `programa1`). As instruções podem também ser digitadas posição por posição na RAM do `Logisim`.

3 Observações Finais

1. Os dados em nosso computador são de 16 *bits*. Porém, a ULA construída no EP2 é de 8 *bits*. Diante disso, podemos considerar duas opções:

   (a) alterar a ULA para manipular números de 16 *bits* (números no intervalo de -2^{15} a $+2^{15}-1$)

   (b) aproveitar a ULA já feita no EP2. Para manter a consistência, devemos garantir que a ULA recebe apenas os 8 *bits* menos significativos dos números a serem operados e, similarmente, que a saída da ULA será completada com 8 *bits* iguais a zero. Neste caso, vamos supor que os números "válidos" são aqueles no intervalo de -128 a +127.

   Esta observação não é relevante no **EP3b** pois ele não inclui instruções que envolvem a ULA.

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

   Caso vocês queiram implementar as instruções de multiplicação e divisão, podem usar os circuitos multiplicadores e divisores do Logisim (mas não sei se eles assumem inteiros na notação complemento de dois ou inteiros sem sinal ...)

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. Utilize recursos que facilitam a organização do circuito: *input/output* com diferentes larguras (em termos de números de *bits*), *label* para cada um dos componentes, *data splitters*, e tunelamento (*Tunnel* no menu `Wiring`).

5. Eventuais inconsistências nas especificações ou outros detalhes (que sempre existem) serão consertados à medida que forem identificados ... (Desde já, agradecimentos àqueles que encontrarem alguma insconsistência/erro.)

6. Entregar: o circuito (arquivo `.circ`) mais um relatório descrevendo a organização geral do circuito e o passo a passo para fazer uma simulação. Deixe claro se você usa uma ULA de 8 ou de 16 *bits*. Datas de entrega no PACA.

7. **Dúvidas?** Perferencialmente, poste suas dúvidas no Fórum de discussão no PACA ou traga-as para a sala de aula.