

Projeto de circuito (parte final)

MAC0329 – Álgebra booleana e aplicações (DCC / IME-USP — 2020)

– Este EP deverá ser feito com o *Logisim* (<http://www.cburch.com/logisim/>) –

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

(se feito em grupo, apenas um membro deve fazer a entrega;
os nomes devem constar no circuito principal)

Parte 3: Processador – entrega no moodle, até 20/07

O objetivo deste EP é integrar vários dos componentes que fazem parte de um processador e implementar o ciclo de execução para simular o funcionamento de um computador hipotético (mais detalhes abaixo).

Usaremos o *Logisim* pois o *CircuitVerse* aparentemente tem algumas limitações. Consulte o outro documento, *logisim.pdf* disponibilizado no moodle, assim como a *Ajuda* disponível no *Logisim* para fazer a transição do *CircuitVerse* para o *Logisim*. É esperado que essa transição seja “suave”.

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 de 8 *bits* (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): (i) 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; (ii) 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).

Registadores: 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 (notadamente, as de I/O e as da ULA).

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*, cujo valor pode ser incrementado com um pulso do clock ou alterado de forma assíncrona.

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), *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). Esse processamento é assíncrono.
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, apenas o passo 1 (*fetch*) e o passo 3 (*execute*) envolvem atualização de memória ou registrador. Portanto, dois pulsos do *clock* são suficientes para a execução de um ciclo.

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. Então, 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.

3 Conjunto de instruções

O conjunto de instruções que do nosso processador está especificado a seguir (essas instruções foram inspiradas no HIPO¹, mas os códigos não são os mesmos). Embora seja mostrado o código em notação decimal, todos os endereços e dados serão em seguida expressos em notação hexadecimal.

Código		Descrição
base 10	base 16	
00	00	NOP (no operation)
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

[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*.

Observação: A instrução 00 (NOP) corresponde a fazer nada no passo 3 do ciclo FDX. A instrução 09 (Pare) deverá fazer com que o processador volte ao estado correspondente ao do início de execução de um programa. Isto é, deve zerar o PC (*Program Counter*) e voltar à configuração de início de um ciclo de execução.

4 Tarefa, entrega e avaliação

Neste EP podem ser usados os componentes disponíveis no **Logisim**. A ULA deve ser a desenvolvida por vocês (como ela foi feita no **CircuitVerse**, sugerimos que a primeira atividade no EP3 seja refazer a ULA no **Logisim**, como forma de se familiarizarem com ele. Em princípio, isto não deveria exigir muito esforço).

Planeje a organização do processador antes de começar a trabalhar no Logsim. Em relação às instruções, implemente inicialmente uma ou outra, teste a sua execução, e uma vez que você tenha entendido a dinâmica, implemente o restante das instruções. Por exemplo, inicialmente podem ser implementadas as seguintes instruções:

Código na base 16	Descrição
00--	NOP (no operation)
01EE	Copie [EE] para o AC
02EE	Copie [AC] para a posição de endereço EE

Nas instruções 01EE e 02EE, use endereços distintos (por exemplo EE=07 e EE=08). Para simular o circuito, escreva as instruções nas posições 00 a 02 e um valor qualquer no *byte* menos

¹<https://www.ime.usp.br/~jstern/software/hipo/Hipo.pdf>

significativo na posição EE 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.

Entrega: entregar via moodle um arquivo `cpu.circ`, contendo o processador descrito acima. Caso ache adequado, entregue também um relatório sucinto sobre como está organizado o processador implementado.

Serão avaliados os seguintes aspectos:

- completude e 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)

5 Exemplos para teste do processador

Para testar o circuito, crie pequenas sequências de instruções e gere os pulsos do *clock* manualmente, de forma que seja possível acompanhar as alterações que ocorrem em diferentes partes do circuito a cada pulso. Lembre-se que iremos supor que a primeira instrução a ser executada estará sempre no endereço 00.

A seguir estão três exemplos de programas, que podem ser usados para testar o processador. Para cada linha do exemplo, estão presentes o endereço na RAM, o código da instrução, e o significado da instrução.

Primeiro exemplo: teste de leitura e impressão. O dado de entrada deve ser representado por um pino de entrada. Antes de executar uma instrução de leitura, insira o valor a ser lido no pino de entrada (podemos supor que esse é o número que foi digitado pelo usuário). No caso da impressão, o valor a ser impresso pode ser enviado para um pino de saída ou então mostrado usando componentes *display*.

```
00: 0705 -- Leia um número e guarde-o na posição de endereço 05
01: 0805 -- Imprima [05]
02: 0900 -- Pare
```

Segundo exemplo: teste de adição e subtração

```
00: 0710 -- Leia um número e guarde-o na posição de endereço 10
01: 0711 -- Leia um número e guarde-o na posição de endereço 11
02: 0110 -- Copie [10] para o AC
03: 0311 -- Some [11] com [AC] e guarde o resultado em AC
04: 0212 -- Copie [AC] para a posição de endereço 12
05: 0812 -- Imprima [12]
06: 0110 -- Copie [10] para o AC
07: 0411 -- Subtraia [11] de [AC] e guarde o resultado em AC
07: 0213 -- Copie [AC] para a posição de endereço 13
08: 0813 -- Imprima [13]
09: 0900 -- Pare
```

Terceiro exemplo: teste de laço (desvios)

```
00: 0110 -- Copie [10] para o AC
01: 0211 -- Copie [AC] para a posição de endereço 11
02: 0811 -- Imprima [11]
03: 0712 -- Leia um número e guarde-o na posição de endereço 12
04: 0112 -- Copie [12] para o AC
05: 0D0A -- Desvie para 0A se [AC] = 0
06: 0311 -- Some [11] com [AC] e guarde o resultado em AC
07: 0211 -- Copie [AC] para a posição de endereço 11
08: 0811 -- Imprima [11]
09: 0A03 -- Desvie para 03
0A: 0900 -- Pare
    ...
0F: ...
10: 0000 -- Zero
11:      -- Soma
12:      -- Num
```

Estes programas podem ser gravados em um arquivo `txt` e carregados para a RAM. Por exemplo, o conteúdo do arquivo `txt` correspondente ao terceiro exemplo deve ser

```
v2.0 raw
0110 0211 0811 0712 0112 0d0a 0311 0211 0811 0a03 0900 0000 0000 0000 0000 0000
```

Para carregar essas instruções na RAM, basta clicar sobre ela e usar a opção `Carregar imagem...`. Para ver todo o conteúdo da RAM ou editá-los, basta usar a opção `Editar conteúdos...`

Postem as dúvidas/comentários/sugestões/correções no Fórum de discussões. Obrigada!