Projeto de circuito

MAC0329 – Álgebra booleana e aplicações (DCC / IME-USP — 2021)

– Todas as etapas do projeto deverão ser feitas usando o Logisim-evolution
https://github.com/reds-heig/logisim-evolution –

Parte 3: Processador (CPU MAC0329) – entrega no e-disciplinas, até 25/07

O objetivo deste EP é implementar o modelo de processador que discutimos em aula. Nesta etapa, os desafios são integrar os diversos componentes do processador, e implementar o ciclo FDX, e simular a execução de programas simples. Veja detalhes abaixo.

1 Componentes de um processador

O circuito terá como componentes principais a ULA (feita no EP2), uma RAM e uma UC (unidade de controle). Adicionalmente, teremos o PC (*program counter*, que será um contador no nosso caso), e os registradores AC e IR. A UC é um circuito combinacional encarregado de fazer, basicamente, a decodificação da instrução e garantir que os sinais de controle estão com os valores corretos para que as instruções sejam executadas corretamente. A RAM, contador e registadores não precisam ser projetados por vocês. Vocês podem usar esses componentes disponíveis no `Logisim`.

Abaixo uma breve explicação de cada um dos componentes. Veja mais detalhes nas notas de aula.

ULA: a ULA deve ser capaz de efetuar as operações de adição e subtração e comparações de números de 8 *bits*, conforme especificado no EP2.

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 deverão ser considerados.

UC: já comentado acima, trata-se de um decodificador de instruções que recebe na entrada o código de uma instrução e cuja saída são os diversos sinais de controle. Cada um dos sinais de controle deve ser ativado/desativado de acordo com a instrução na entrada, para garantir a correta execução da instrução.

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 (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 (ou seja, vamos supor que a primeira instrução de um programa estará sempre na posição de endereço 0x00). 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.

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 devem ser ajustados 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. Mais detalhes podem ser vistos nas notas de aula.

3 Conjunto de instruções

O conjunto de instruções do nosso processador está especificado a seguir. São os mesmos códigos que estão nas notas de aula. Os códigos estão em hexadecimal.

Código	Descrição da instrução
00	NOP (no operation)
01	Copie [EE] para o AC
02	Copie [AC] para a posição de endereço EE
03	Some [EE] com [AC] e guarde o resultado em AC
04	Subtraia [EE] de [AC] e guarde o resultado em AC
07	Leia um número e guarde-o na posição de endereço EE
08	Imprima [EE]
09	Pare
0A	Desvie para EE (desvio incondicional)
0B	Desvie para EE se [AC] > 0
0D	Desvie para EE se [AC] $= 0$
0F	Desvie para EE se [AC] < 0

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

[AC] significa o conteúdo do AC

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 Dicas

Planeje a organização do processador antes de começar a trabalhar no `Logsim`. Em relação às intruções, 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 intruções:

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 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.

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.

```
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: Como seria o código para o seguinte programa?

```
    Leia um número
    se número < 0:
       imprima -1
    senão
       imprima 1
```

Quarto 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 quarto exemplo pode ser:

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

6 Entrega

Entregar via e-disciplinas um arquivo `cpu.circ`, contendo o processador descrito acima. Capriche na organização do circuito/subcircuitos.

Este enunciado foi baseado em testes realizados com uma versão antiga do `Logisim` (antes da versão *evolution*). Se houver algo inconsistente com o enunciado, avise o mestre!

Postem as dúvidas/cometários/sugestões/correções no Fórum de discusssões. Obrigada!