Projeto: implementação de um processador simples

MAC0329 – Álgebra booleana e aplicações (BCC / IME-USP — 2024)

(Enunciado final – em relação à versão anterior, foi acrescida a seção 3)

Objetivo

Como parte prática da disciplina, vamos projetar um circuito lógico que implementa um processador simples e em seguida fazer simulações de execução de programas (sequências de instruções de máquina) simples no processador implementado.

Para o desenvolvimento dos circuitos, usaremos o Logisim-evolution (`https://github.com/reds-heig/logisim-evolution`).

Entregas

O projeto está dividido em 3 etapas, que serão detalhadas mais adiante.

O projeto pode ser desenvolvido em grupos de 3. É esperado que todos os integrantes saibam explicar o que foi implementado e como foi feita a implementação.

Para cada etapa, haverá uma tarefa correspondente no e-disciplinas. Lá deverá ser entregue o arquivo com a extensão `.circ`, gerado pelo `logisim-evolution`, contendo o circuito desenvolvido. BASTA A ENTREGA POR APENAS UM MEMBRO DO GRUPO; não esqueçam de incluir no próprio arquivo do circuito o nome dos integrantes do grupo.

Critérios de avaliação

Serão avaliados os seguintes aspectos:

- corretude dos circuitos
- aderência às especificações
- organização do circuito, identificação clara dos pinos de entrada e saída em cada subcircuito, modularização, documentação, etc

Características do nosso processador

O processador a ser implementado deverá ser capaz de executar instruções (à la HIPO) armazenadas na memória RAM. O processador será composto por uma memória RAM, uma ULA (unidade lógico-aritmética), uma parte responsável por controlar o ciclo de instruções, e mais alguns componentes. Todos esses componentes serão detalhados ao longo do semestre.

Especificações:

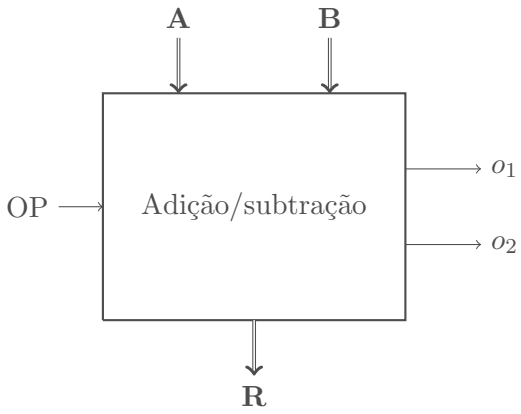
- **Palavra:** serão de 16 *bits*
- **Endereço:** serão de 8 *bits*. Portanto, a RAM terá $2^8 = 256$ posições de 16 *bits* cada.
- **Instrução:** os 8 *bits* mais significativos correspondem ao código da instrução, e os 8 *bits* menos significativos a um endereço (quando for o caso)
- **Dado:** como a RAM é de 16 *bits*, poderíamos considerar dados de 16 *bits*. Mas, para simplificar, vamos considerar dados de 8 *bits* e supor que eles estarão sempre armazenados nos 8 *bits* menos significativos quando estiverem aramazenados em palavras de 16 *bits*. Dados, quando representam números inteiros, podem ser interpretados como inteiros sem ou com sinal, sempre de acordo com a notação complemento de 2.
- **ULA:** Por conta de considerarmos dados de 8 *bits*, a ULA (unidade lógico-aritmética) precisará operar dados de 8 *bits*
- **Registradores:** nosso processador terá apenas alguns registradores.
  - **AC** (acumulador) irá basicamente armazenar dados e, portanto, deverá comportar 8 *bits*.
  - **IR** (registrador de instruções) irá armazenar uma instrução e portanto deverá comportar palavras de 16 *bits*.
  - **PC** (*Program counter*, ou apontador de instruções) armazena o endereço da próxima instrução a ser executada e, portanto, deve comportar 8 *bits*.

---

1 Etapa 1 – implementação de um somador de 8 *bits*

Na **Etapa 1** do projeto, o objetivo é a implementação de uma parte da ULA, a parte responsável pelas operações de adição e subtração. Note que o circuito deve ser capaz de realizar tanto a adição como a subtração, e deverá também indicar se ocorreu *overflow*, para as duas interpretações (sem sinal e com sinal) dos binários de entrada. Quanto às informações sobre as características do processador listadas acima, por enquanto a única que nos interessa diz respeito ao **dado** (adotaremos 8 *bits*). Caso tenha interesse em fazer a implementação para dados de 16 *bits*, sinta-se à vontade (apenas não esqueça de adequar todas as partes do processador que lidam com dados para elas comportarem palavras de 16 *bits*).

A seguir está uma representação diagramática e a especificação das entradas e saídas desse circuito:



As entradas são:

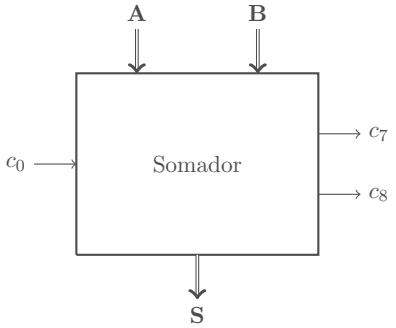
- **A** e **B** são os operandos, ambos de 8 *bits*;
- OP é um *bit* para indicar o tipo de operação: se 0, deve ser calculada a adição $\mathbf{A} + \mathbf{B}$ e, se 1, deve ser calculada a subtração $\mathbf{A} - \mathbf{B}$;

e as saídas são:

- **R** é o resultado da operação (8 *bits*);
- o_1 é um *bit*; se for 1, indica que a operação calculada, interpretando-se os números como sem sinal, resultou em *overflow*;
- o_2 é um *bit*; se for 1, indica que a operação calculada, interpretando-se os números como com sinal (notação complemento de dois), resultou em *overflow*.

Dicas de *design*: Em geral, um circuito complexo pode ser decomposto em subcircuitos. No caso do circuito acima, podemos considerar que um sub-componente principal dele é um somador de 8 *bits*. Já o somador de 8 *bits* pode ser visto como a composição de múltiplos somadores de *bits*. Os diagramas a seguir ilustram esses componentes.

Somador de 8 bits



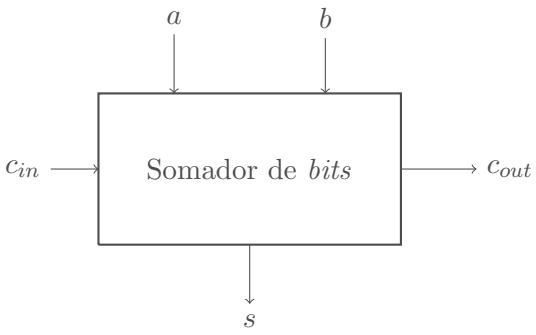
Entradas:

- **A** e **B** (operandos 1 e 2)
  $\underbrace{(a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)}_{\mathbf{A}}$ $\underbrace{(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)}_{\mathbf{B}}$
- c_0 (*carry* na coluna 0)

Saídas:

- $\mathbf{S} = \mathbf{A} + \mathbf{B}$ (8 *bits* da soma) $\underbrace{(s_7, s_6, s_5, s_4, s_3, s_2, s_1, s_0)}_{\mathbf{S}}$
- c_7 (*carry* na coluna 7)
- c_8 (*carry* na coluna 8)

Somador de *bits*: recebe três *bits*, a, b e c_{in}, e devolve dois *bits*, s e c_{out} (conforme discutido em sala de aula)



Portanto, para construir o circuito aritmético, podemos primeiramente construir um subcircuito somador de *bits*, e em seguida usá-los para a construção do circuito somador de 8 *bits*. Por fim, o circuito principal (*main*) usará o somador de 8 *bits* para implementar as operações de adição e subtração.

Em termos de organização dos subcircuitos, a posição dos pinos de entrada e saída pode ser modificada livremente. Apenas certifique-se que os pinos estão claramente identificados.

---

2 Etapa 2 – implementação da ULA

ULA é a contração de "Unidade Lógico-Aritmética". Ela é a parte da CPU que é responsável por executar as operações aritméticas e lógicas. Nesta etapa, construiremos um **comparador** e juntaremos com o somador feito na primeira etapa para construir uma ULA (simplificada).

2.1 Especificação do comparador

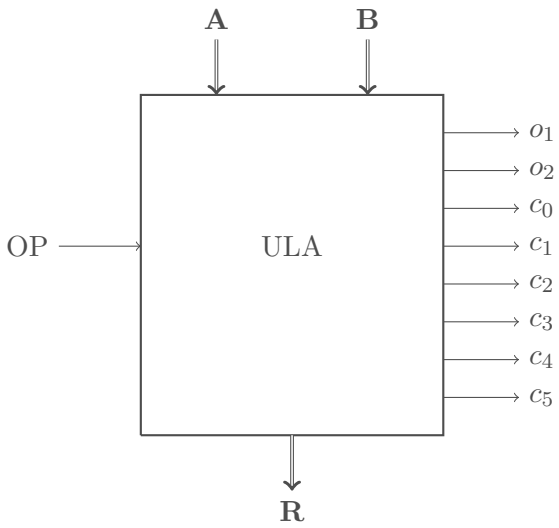
O nosso comparador é um circuito com duas entradas e 6 *bits* (*flags*) na saída. As entradas são dois binários de 8 *bits* cada, que denotamos **A** e **B**. As *flags* na saída são resultados de diferentes tipos de comparações entre **A** e **B** e estão especificadas a seguir:

- $c_0 = 1$ se e somente se $\mathbf{A} == \mathbf{0}$ (isto é, se o valor da primeira entrada é zero; note que aqui também a interpertação sem/com sinal não é relevante. Também não é relevante o valor de **B**),
- $c_1 = 1$ se e somente se $\mathbf{A} == \mathbf{B}$ (isto é, se ambas as entradas são iguais; note que aqui a interpertação sem/com sinal não é relevante)
- $c_2 = 1$ se e somente se $\mathbf{A} > \mathbf{B}$, considerando-se a interpretação sem sinal de seus valores
- $c_3 = 1$ se e somente se $\mathbf{A} < \mathbf{B}$, considerando-se a interpretação sem sinal de seus valores
- $c_4 = 1$ se e somente se $\mathbf{A} > \mathbf{B}$, considerando-se a interpretação com sinal de seus valores
- $c_5 = 1$ se e somente se $\mathbf{A} < \mathbf{B}$, considerando-se a interpretação com sinal de seus valores

2.2 Especificação da ULA

Nossa ULA deverá ser capaz de realizar duas operações aritméticas (adição e subtração) e comparações de dois números, todos inteiros representados em 8 *bits*. Ela deve ser construída usando-se o circuito construído na Etapa 1 e o comparador descrito acima.

O diagrama abaixo indica as entradas e saídas da ULA:



Entradas:

- **A** e **B** são os operandos, ambos de 8 *bits*;
- OP é um *bit* para indicar o tipo de operação: se 0, deve ser calculada a adição $\mathbf{A} + \mathbf{B}$ e, se 1, deve ser calculada a subtração $\mathbf{A} - \mathbf{B}$; as comparações serão sempre calculadas;

Saídas:

- **R** é o resultado (8 *bits*) da operação aritmética, conforme especificado na Etapa 1;
- o_1 e o_2 são as *flags* que indicam *overflow* na operação aritmética, para os casos de interpretação sem sinal e com sinal, respectivamente (conforme especificado na Etapa 1);
- c_0 a c_5 são as *flags* que indicam os resultados das comparações conforme especificados acima.

3 Etapa 3 – implementação da CPU

3.1 Visão geral

O circuito do processador possui como componentes principais a ULA (no nosso caso, feita nos EP1 e EP2), uma RAM e uma UC (unidade de controle).

Vamos supor que a sequência de instruções de máquina que correspondem ao programa a ser executado no nosso processador estará sempre armazenado a partir do endereço zero da RAM (ou seja, a primeira instrução do programa estará na posição de memória com endereço zero).

A execução de um programa ocorre repetindo-se um **ciclo de instrução** (em inglês, *fetch-decode-execute cycle* ou FDX). Em cada ciclo de instrução, o processador executa três passos:

1. *Fetch*: buscar a próxima instrução a ser executada (que está na RAM)
2. *Decode*: decodificar a instrução
3. *Execute*: executar a instrução

Em resumo, a cada ciclo de instrução uma instrução é executada. No ciclo subsequente, a próxima instrução deverá ser executada e assim por diante.

O processador utiliza registradores auxiliares, já mencionados no início do enunciado. Aqui detalhamos mais um pouco qual o papel deles em cada ciclo de execução:

- PC (*program counter*): contém o endereço na RAM onde está a próxima instrução a ser executada
- IR (registrador de instrução): No passo *Fetch*, a instrução a ser executada é copiada da RAM para esse registrador
- AC (Acumulador): é um registrador que armazena temporariamente dados (relacionados às instruções que envolvem o uso da ULA ou do I/O)

3.2 Conjunto de instruções de máquina

Programas a serem executados são sequências de instruções de máquina. O conjunto de instruções do nosso processador está especificado a seguir. Os códigos das instruções estão em notação hexadecimal. O conjunto de instruções é inspirado no HIPO e aqui consideramos um subconjunto pequeno suficiente para fazer alguns programas simples. Os códigos das instruções não são as mesmas do HIPO (que aqueles que estão cursando MAC0110 devem ter simulado no início do semestre).

Código	Descrição da instrução
00	NOP (*no operation*)
01	Pare
03	Copie [EE] para o AC
04	Copie [AC] para a posição de endereço EE
05	Some [EE] com [AC] e guarde o resultado em AC
06	Subtraia [EE] de [AC] e guarde o resultado em AC
08	Leia um número e guarde-o na posição de endereço EE
09	Imprima [EE]
0B	Desvie para EE (desvio incondicional)
0C	Desvie para EE se [AC] > 0
0D	Desvie para EE se [AC] $= 0$
0E	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 01 (Pare) faz o processador voltar ao estado correspondente ao do início de execução de um programa.

3.3 Detalhes sobre o ciclo de instrução

Os três passos do ciclo FDX são detalhados a seguir. Cabe à UC acertar os sinais de controle para que a execução da instrução ocorra corretamente.

1. *Fetch*: buscar a próxima instrução a ser executada

   - a instrução que se encontra na RAM, na posição apontada pelo PC, deve ser lida e carregada par o IR. Além disso, o valor do PC deve ser incrementado em 1.

2. *Decode*: decodificar a instrução

   - 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 RAM e dos registradores, ajustar o valor dos sinais que controlam os pinos seletores dos MUXes, etc). Esse processamento é assíncrono.

3. *Execute*: executar as ações determinadas pela instrução (basicamente, mudança de estado em componentes de memória)

   - Além disso, o ciclo deve ser "resetado" (voltar para a configuração de antes do passo *Fetch*.

Um ciclo de instrução é tipicamente executado em um número fixo de períodos do *clock*. Lembre-se que o *clock* é utilizado para sicronizar a mudança de estado. Observando o ciclo FDX, mudanças de estado acontecem nos passos 1 (*fetch*) e 3 (*execute*). Portanto, no nosso processador,

dois pulsos do *clock* são suficientes para a execução de um ciclo. O passo 2 (*Decode*) é executado assim que a instrução é carregada no IR; o decodificador de instruções deve cuidar para que os dados de entrada e sinais de controle em diferentes partes do circuito estejam corretamente configurados para que no segundo pulso do *clock* (isto é, no passo *Execute*) ocorra a mudança de estado correta.

3.4 Exemplos de programas

Nos exemplos a seguir, em cada linha encontram-se as seguintes informações: o endereço na RAM, a instrução (em notação hexadecimal), e a "tradução" dessa instrução.

Exemplo 1: Programa que lê um número e em seguida o imprime

```
00: 0805 -- Leia um número e guarde-o na posição de endereço 05
01: 0905 -- Imprima [05]
02: 0100 -- Pare
```

Exemplo 2: Programa que lê dois números e calcula e imprime a adição e subtração entre eles

```
00: 0810 -- Leia um número e guarde-o na posição de endereço 10
01: 0811 -- Leia um número e guarde-o na posição de endereço 11
02: 0310 -- Copie [10] para o AC
03: 0511 -- Some [11] com [AC] e guarde o resultado em AC
04: 0412 -- Copie [AC] para a posição de endereço 12
05: 0912 -- Imprima [12]
06: 0310 -- Copie [10] para o AC
07: 0611 -- Subtraia [11] de [AC] e guarde o resultado em AC
07: 0413 -- Copie [AC] para a posição de endereço 13
08: 0913 -- Imprima [13]
09: 0100 -- Pare
```

Exemplo 3: Programa que envolve laço

```
00: 0310 -- Copie [10] para o AC
01: 0411 -- Copie [AC] para a posição de endereço 11
02: 0911 -- Imprima [11]
03: 0812 -- Leia um número e guarde-o na posição de endereço 12
04: 0312 -- Copie [12] para o AC
05: 0D0A -- Desvie para 0A se [AC] = 0
06: 0511 -- Some [11] com [AC] e guarde o resultado em AC
07: 0411 -- Copie [AC] para a posição de endereço 11
08: 0911 -- Imprima [11]
```

```
09: 0B03 -- Desvie para 03
0A: 0100 -- Pare
    ...
0F: ...
10: 0000 -- Zero
11:      -- Soma
12:      -- Num
```

Exemplo 4: Como seria o código para o seguinte programa?

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

3.5 Sobre a implementação

Vocês podem usar a RAM, registrador e contadores disponíveis no `Logisim`. Assim como pinos de entrada, pinos de saída, portas lógicas, *clock*, multiplexadores, entre outros. A ULA deve ser aquele que foi projetado por vocês no EP1 e EP2 (se necessário, corrija ela).

Nas instruções de leitura e escrita, um 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.

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 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*)
03EE	Copie [EE] para o AC
04EE	Copie [AC] para a posição de endereço EE

Nas instruções 03EE e 04EE, 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.

Escrever as instruções na RAM manualmente é muito chato. Para evitar isso, crie um arquivo `txt` com a sequência de instruções do programa e então carregue as instruções para a RAM a partir do arquivo. Por exemplo, o conteúdo do arquivo `txt` correspondente ao "Exemplo 3" (acima) pode ser[1]:

```
v2.0 raw
0310 0411 0911 0812 0312 0d0a 0511 0411 0911 0b03 0100 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...`

Se houver necessidade, esse enunciado será atualizado e um aviso será enviado pelo e-disciplinas.

[1] Ou assim era.