Desenvolvimento de Sistemas Embarcados em Tempo Real

Prof. Hermano Cabral

Departamento de Eletrônica e Sistemas — UFPE

13 de junho de 2024

Plano de Aula

Tema central

- Programação reativa e Máquinas de estados

Plano de Aula

Tema central

- Programação reativa e Máquinas de estados

Objetivos

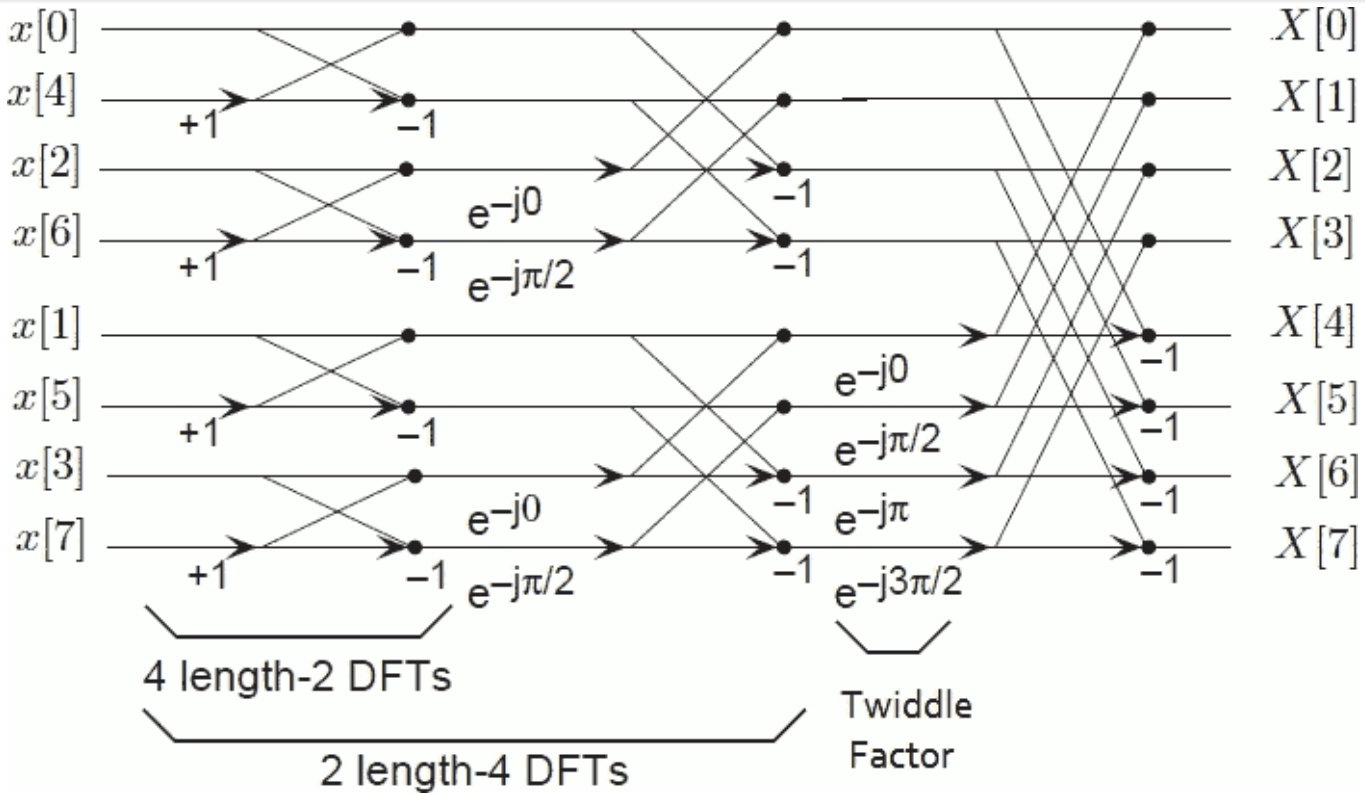
- Conhecer os principais modelos de computação
- Identificar as principais características de programação reativa
- Desenvolver programas usando os conceitos de programação reativa
- Conhecer as características de uma máquina de estados
- Programar uma máquina de estados

Modelos de computação

Introdução

- Na sua grande maioria, podemos dividir os programas em 2 classes:
  - Fluxo de dados
  - Fluxo de controle

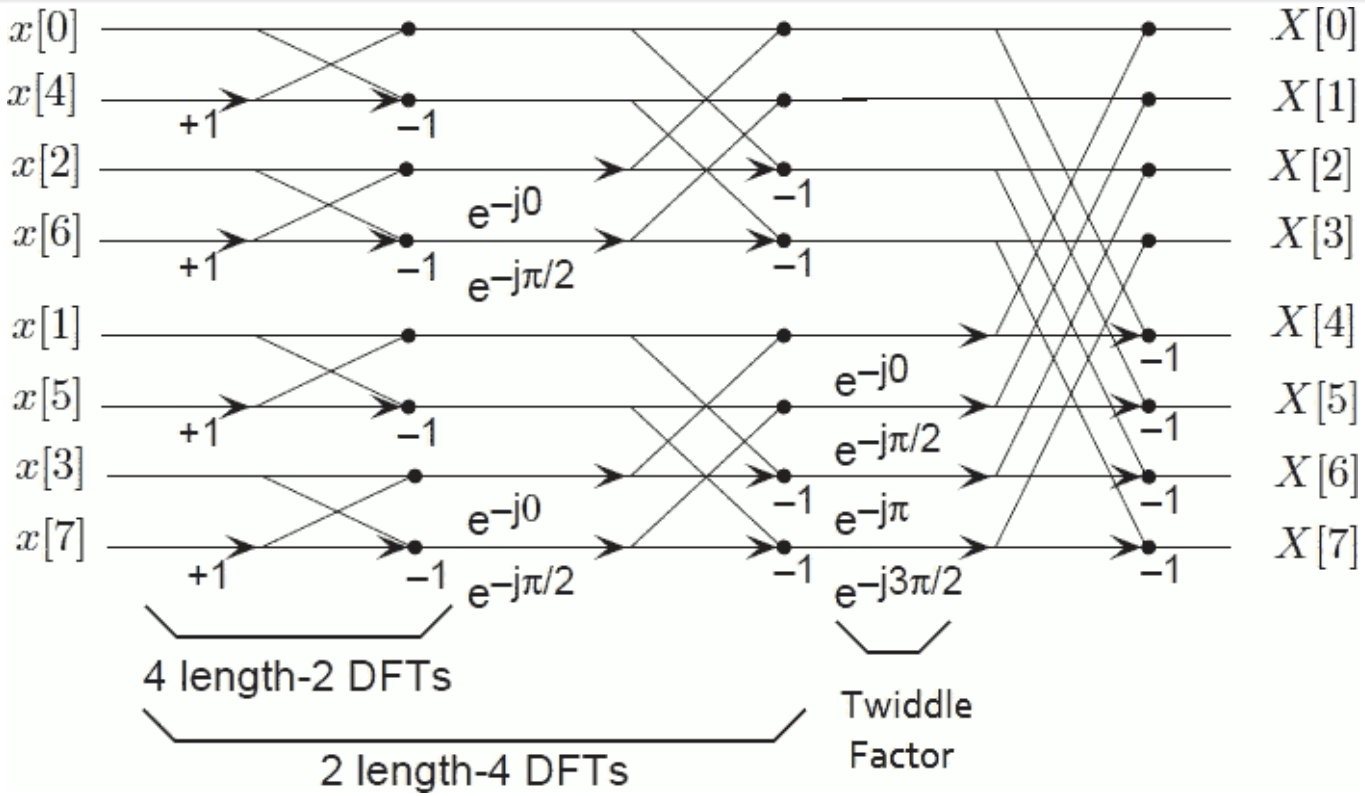
Modelos de computação



Fluxo de dados

- No caso da classe de fluxo de dados, o processamento é simples.

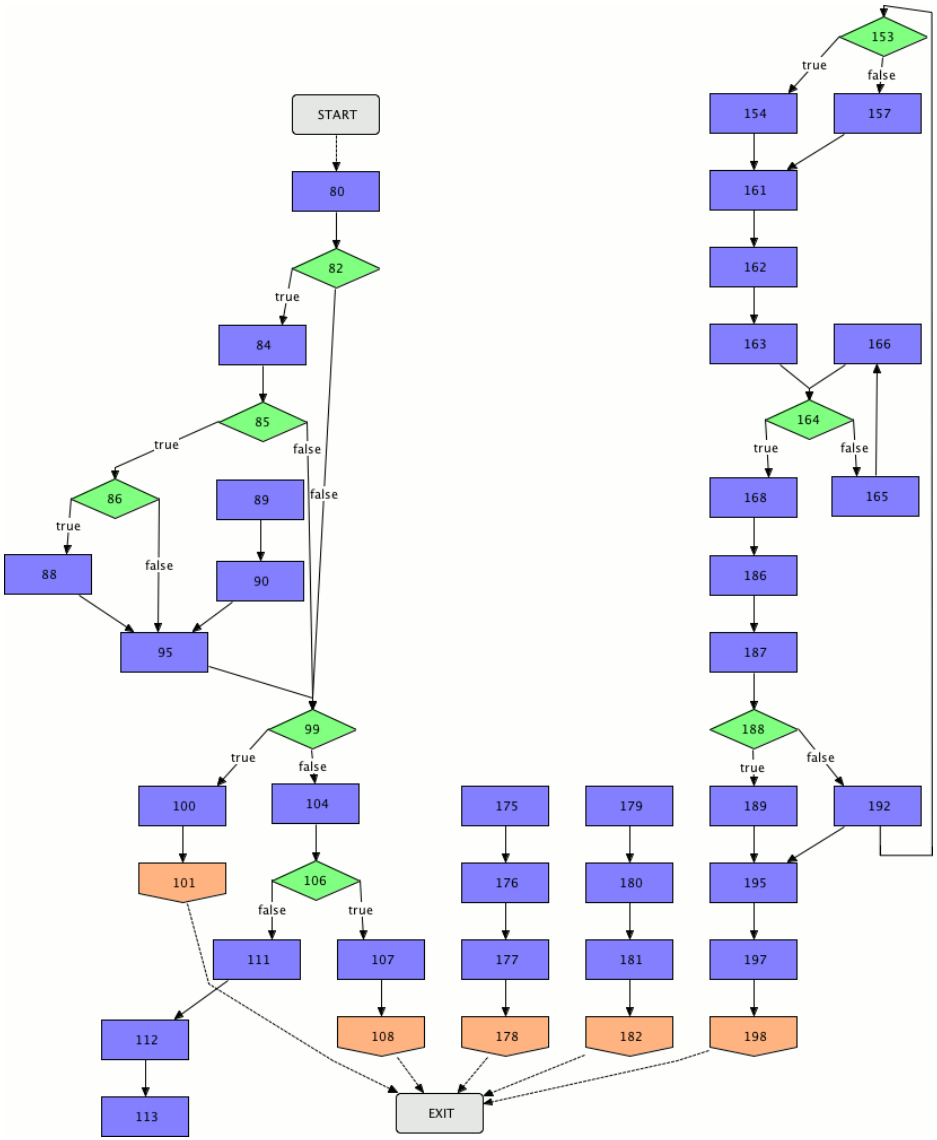
Modelos de computação



Fluxo de dados

- No caso da classe de fluxo de dados, o processamento é simples.
- A questão é garantir o fluxo contínuo de dados.

Modelos de computação



Fluxo de controle

- Na classe de fluxo de controle, o foco é nas instruções que devem ser executadas baseado em condições internas e externas.

Observações

- Existem programas que tem uma estrutura de fluxo de controle onde certas partes são do tipo fluxo de dados.

Observações

- Existem programas que tem uma estrutura de fluxo de controle onde certas partes são do tipo fluxo de dados.
- Além disso, em geral parte da estrutura de fluxo é em uma espera por uma condição ocorrer.

Observações

- Existem programas que tem uma estrutura de fluxo de controle onde certas partes são do tipo fluxo de dados.
- Além disso, em geral parte da estrutura de fluxo é em uma espera por uma condição ocorrer.
- Isto nos leva ao conceito de programação reativa.

Programação reativa

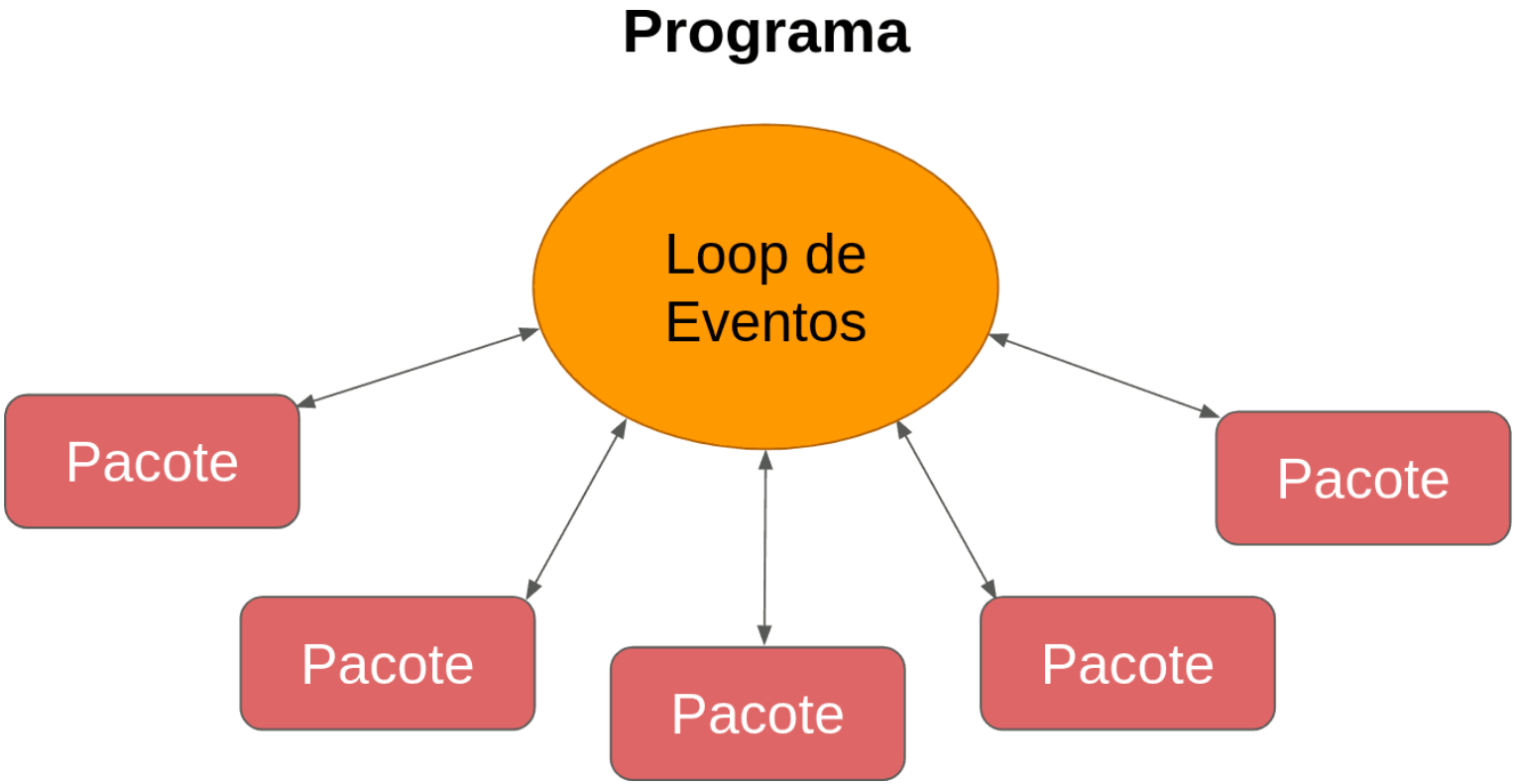
Introdução

- A ideia da programação reativa é reagir a eventos, ou condições.

Programação reativa

Introdução

- A ideia da programação reativa é reagir a eventos, ou condições.
- Não devemos pensar em um programa como uma sequência de instruções a serem realizadas.



Introdução

- Um programa é uma coleção de "pacotes" de instruções onde cada pacote deve ser executado se uma ou mais condições (ou eventos) acontecerem.

Desenvolvimento

- Muitas dessas condições são eventos relativos aos periféricos.

Desenvolvimento

- Muitas dessas condições são eventos relativos aos periféricos.
- Ao receber a ação, realizamos um "pacote" de instruções (a tarefa) ao término do qual voltamos a esperar uma outra ação.

Desenvolvimento

- Os eventos podem ser gerados externamente ou internamente.

Desenvolvimento

- Os eventos podem ser gerados externamente ou internamente.
- Se forem gerados externamente, usamos interrupções ou threads para assimilar o evento.

Desenvolvimento

- Os eventos podem ser gerados externamente ou internamente.
- Se forem gerados externamente, usamos interrupções ou threads para assimilar o evento.
- Por outro lado, uma tarefa pode enviar um evento para acionar outra tarefa.

Desenvolvimento

- O tratamento de eventos pode ser feito de forma sequencial ou paralela.

Desenvolvimento

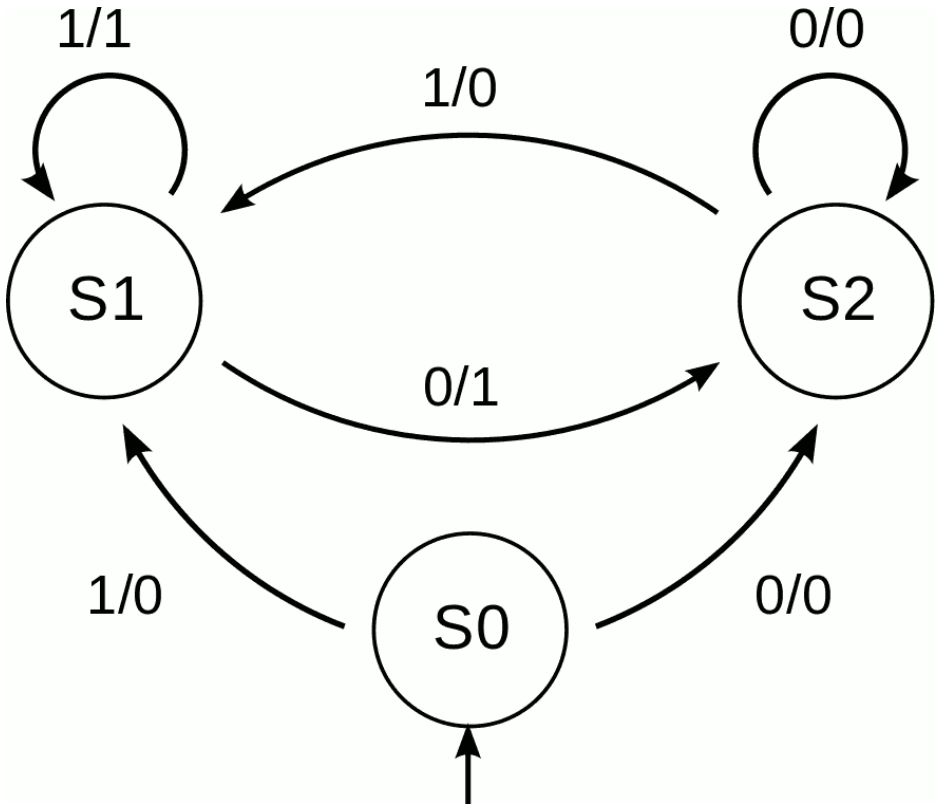
- O tratamento de eventos pode ser feito de forma sequencial ou paralela.
- O mais simples, entretanto, é o sequencial.

Programação reativa

Desenvolvimento

- O tratamento de eventos pode ser feito de forma sequencial ou paralela.
- O mais simples, entretanto, é o sequencial.
- No caso de sistemas de tempo real, entretanto, pode ser necessário programar em paralelo.

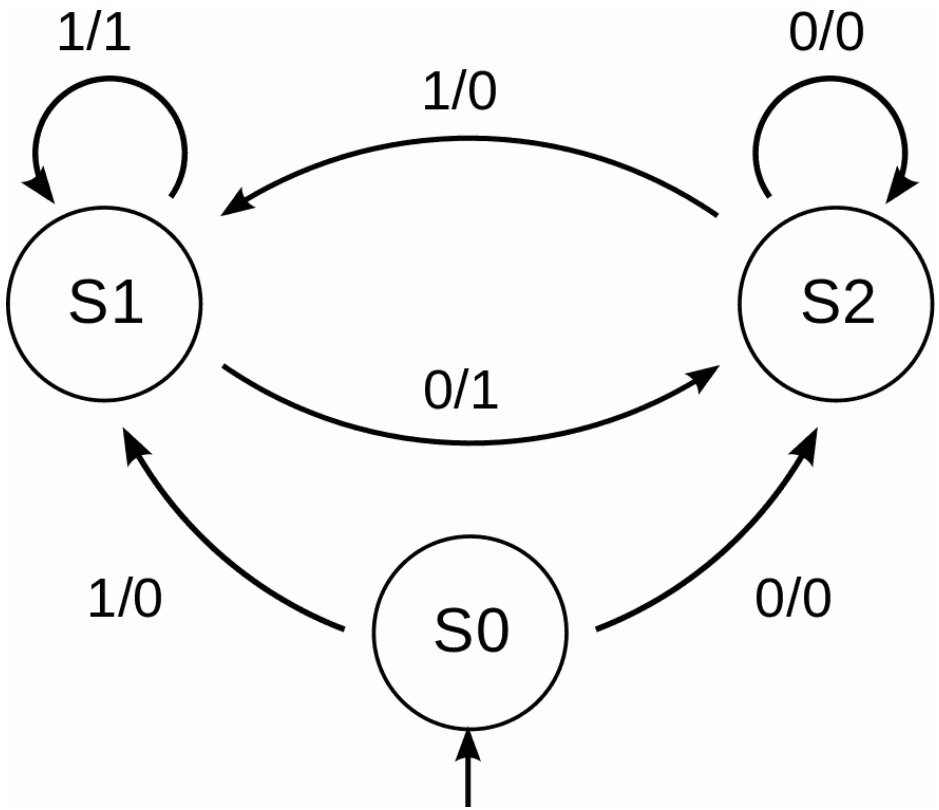
Máquina de estados



Definição

- Uma máquina de estados é um sistema com conjuntos finitos de estados, entradas e saídas e uma função de transição de estados.

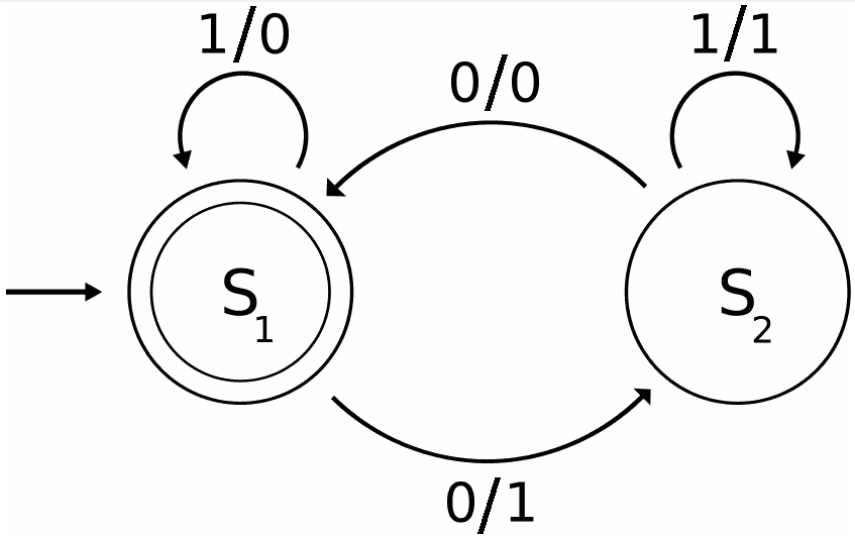
Máquina de estados



Definição

- Cada transição de um estado a outro é rotulado na forma e/s, onde e é a entrada do sistema e s é a saída.

Máquina de estados

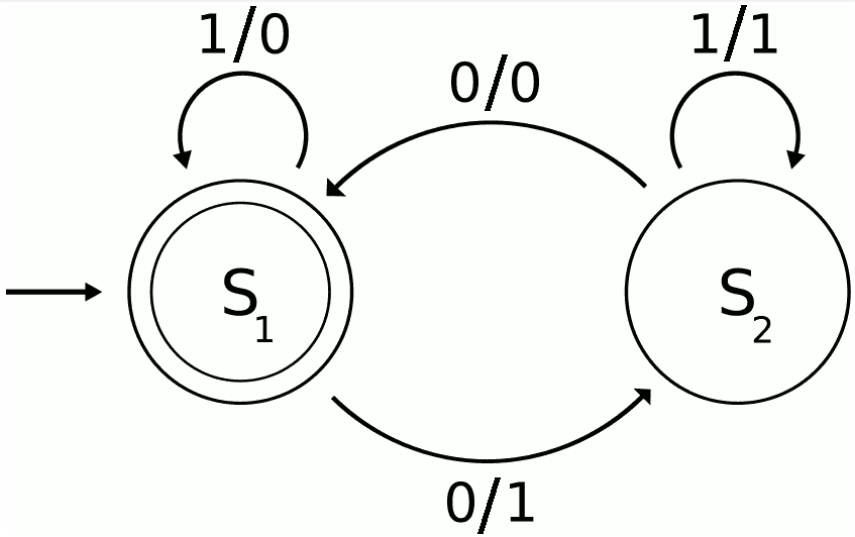


Máquina de estados para determinar se uma sequência de bits possui um número par ou ímpar de 0

Características

- Uma máquina de estados é útil na representação de um sistema onde um estado pode representar todo o histórico de eventos ocorridos do sistema.

Máquina de estados

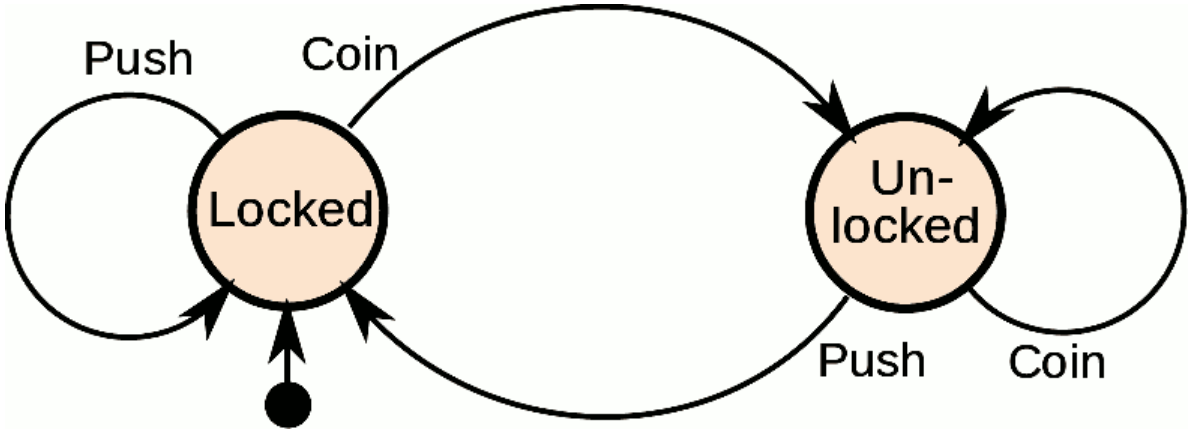


Máquina de estados para determinar se uma sequência de bits possui um número par ou ímpar de 0

Características

- Uma máquina de estados é útil na representação de um sistema onde um estado pode representar todo o histórico de eventos ocorridos do sistema.
- Isto significa que a resposta do sistema a um evento depende de um contexto que é representado por um estado.

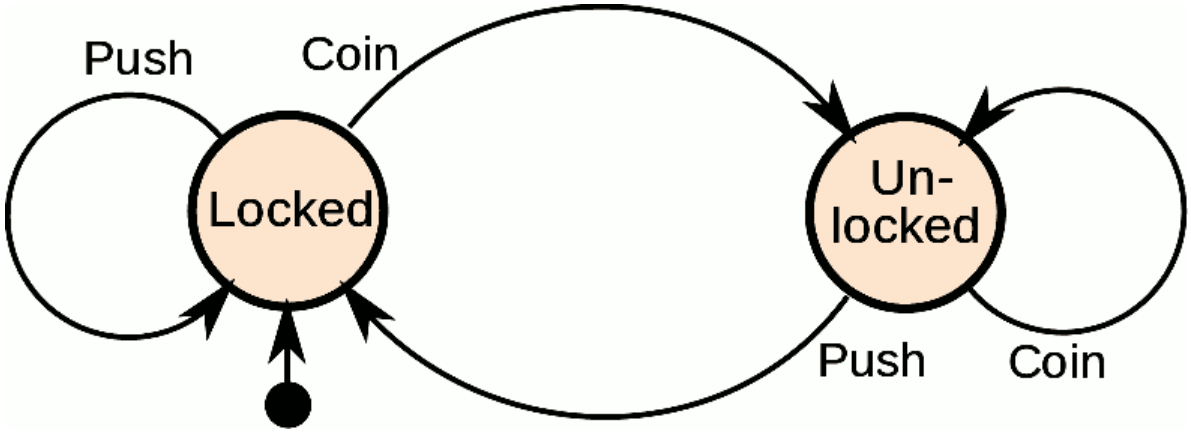
Máquina de estados



Características

- Um outro exemplo é o de uma catraca de metrô.

Máquina de estados



Características

- Um outro exemplo é o de uma catraca de metrô.
- Observe que no exemplo acima os eventos são {moeda inserida, catraca empurrada}, e não números.

Máquina de estados

Representação

- Além da representação visual, uma máquina de estados também pode ser representada por uma tabela.

Representação

- Além da representação visual, uma máquina de estados também pode ser representada por uma tabela.
- No caso do número par ou ímpar de zeros:

	S_1	S_2
0	$S_2/1$	$S_1/0$
1	$S_1/0$	$S_2/1$

Máquina de estados

Implementação

- Uma máquina de estados pode ser implementada de diferentes maneiras em C.

Máquina de estados

Implementação

- Uma máquina de estados pode ser implementada de diferentes maneiras em C.
- Duas formas mais usuais são:
  - Usando a instrução `switch`
  - Usando um vetor de ponteiros para funções

Máquina de estados

```
 1 typedef enum {
 2     STATE1 = 0, STATE2
 3
 4 } estados;
 5 typedef enum {
 6     INPUT1 = 0, INPUT2
 7 } entradas;
 8
 9 int main()
10 {
11     estados state  = STATE1;
12     entradas input = INPUT1;
13
14     switch (state) {
15         case STATE1:
16             switch (input) {
17                 case INPUT1:
18                     break;
19                 case INPUT2:
20                     break;
21                 default:
22                     break;
23             }
24             break;
25         case STATE2:
26             switch (input) {
27                 case INPUT1:
28                     break;
29                 case INPUT2:
30                     break;
31                 default:
32                     break;
33             }
34             break;
35         default:
36             break;
37     }
38
39     return 0;
40 }
```

implementação – usando switch

- A implementação por um switch duplo está mostrada ao lado.

Máquina de estados

```
 1 typedef enum {
 2     STATE1 = 0, STATE2
 3
 4 } estados;
 5 typedef enum {
 6     INPUT1 = 0, INPUT2
 7 } entradas;
 8
 9 int main()
10 {
11     estados state  = STATE1;
12     entradas input = INPUT1;
13
14     switch (state) {
15         case STATE1:
16             switch (input) {
17                 case INPUT1:
18                     break;
19                 case INPUT2:
20                     break;
21                 default:
22                     break;
23             }
24             break;
25         case STATE2:
26             switch (input) {
27                 case INPUT1:
28                     break;
29                 case INPUT2:
30                     break;
31                 default:
32                     break;
33             }
34             break;
35         default:
36             break;
37     }
38
39     return 0;
40 }
```

implementação – usando switch

- A implementação por um switch duplo está mostrada ao lado.
- Note que podemos trocar a instrução `switch` por uma sequência de `ifs`.

Máquina de estados

```
 1 typedef enum {
 2     STATE1 = 0, STATE2, MAX_STATE
 3 } estados;
 4 typedef enum {
 5     INPUT1 = 0, INPUT2, MAX_INPUT
 6 } entradas;
 7 typedef void (*cb_t)(void);
 8
 9 cb_t machine[MAX_STATE][MAX_INPUT] = {
10     0, 0,  /* funções para estado 1 */
11     0, 0   /* funções para estado 2 */
12 };
13 estados next_state[MAX_STATE][MAX_INPUT] = {
14     0, 0,  /* próximos estados para estado 1 */
15     0, 0   /* próximos estados para estado 2 */
16 };
17 estados state  = STATE1;
18
19 int main()
20 {
21     entradas input;
22
23     while(1) {
24         input = wait_for_events();
25         machine[state][input];
26         state = next_state[estado];
27     }
28
29     return 0;
30 }
```

Implementação — usando ponteiros

- Ao invés de um switch duplo, podemos usar uma tabela de ponteiros para funções.

Máquina de estados

Exemplo

- Implemente um programa que retire as linhas de comentário de um programa C.