Arquitetura de Computadores

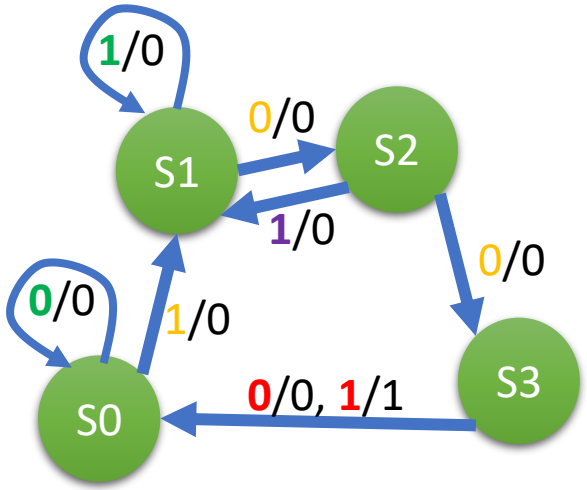
PVP 13 – Capítulo 7
Realização de Circuitos Sequenciais Complexos

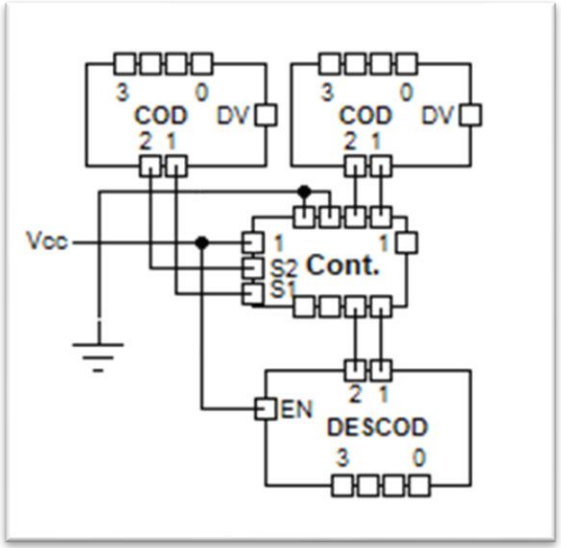
José Coelho,
Gracinda Carvalho 2023

Índice

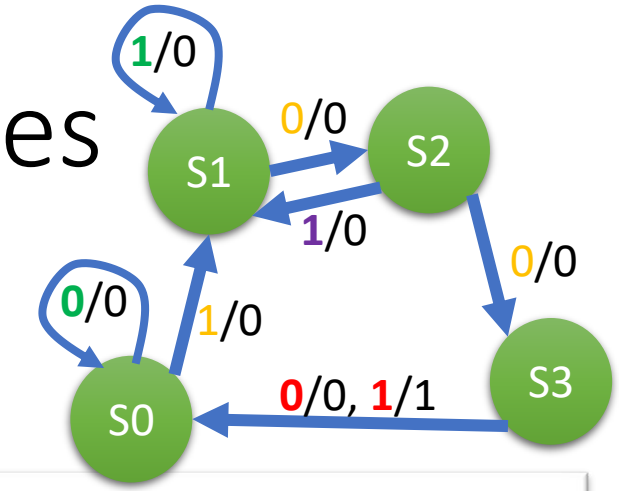
Circuitos Sequenciais com Contadores

- Complexo:
  - Muitas entradas / saídas
  - Simplificações complexas
  - Método pouco modular
- Notar que:
  - Registo = conjunto de básculas
  - Há quase sempre um salto que incrementa o estado
  - Há laços em que o estado mantém-se
  - Porque não utilizar um: Contador!
- Contador PVP 9
  - Controlo S:
    - **00 – clear - limpa**
    - **01 – load - carrega**
    - **10 – incrementa**
    - **11 – mantém**
  - Codificador para controlo S
  - Codificador para valor a carregar
  - Descodificador para descodificar o estado

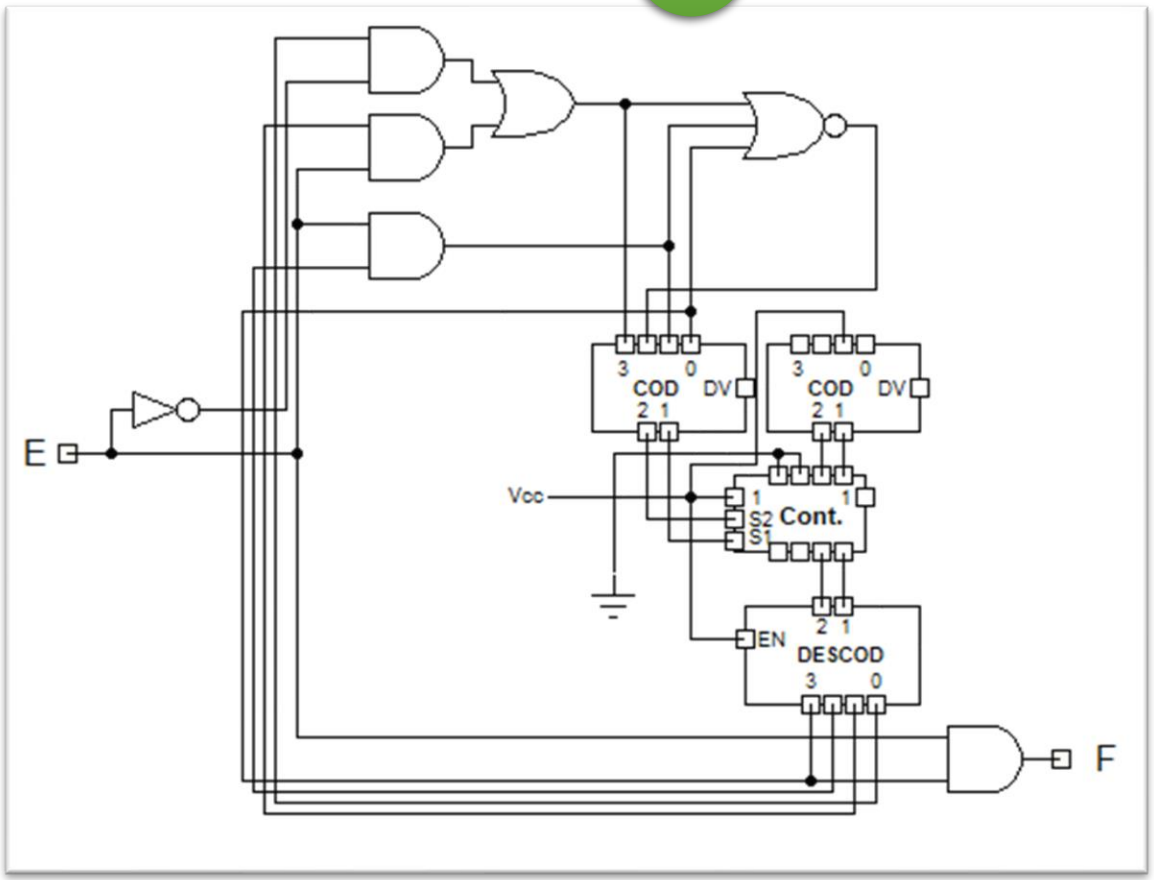




Circuitos Sequenciais com Contadores

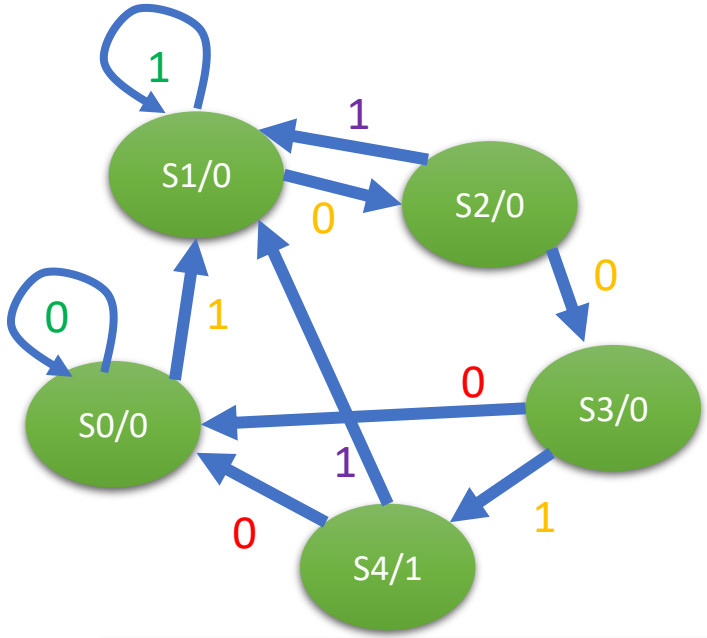


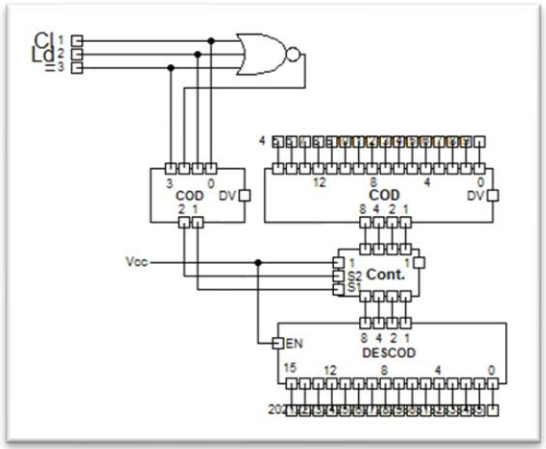
- Necessário:
  - Lógica a colocar no codificador
- Carregar:
  - 1 – para o salto de S2 para S1
- Controlo S:
  - 00 – clear – limpa
    - S_3 – sem qualquer outra condição
  - 01 – load – carrega
    - S_2E – deve ser carregado o valor fixo 1
  - 10 – incrementa
    - Restantes casos falsos
  - 11 – mantém
    - $S_0\bar{E} + S_1E$

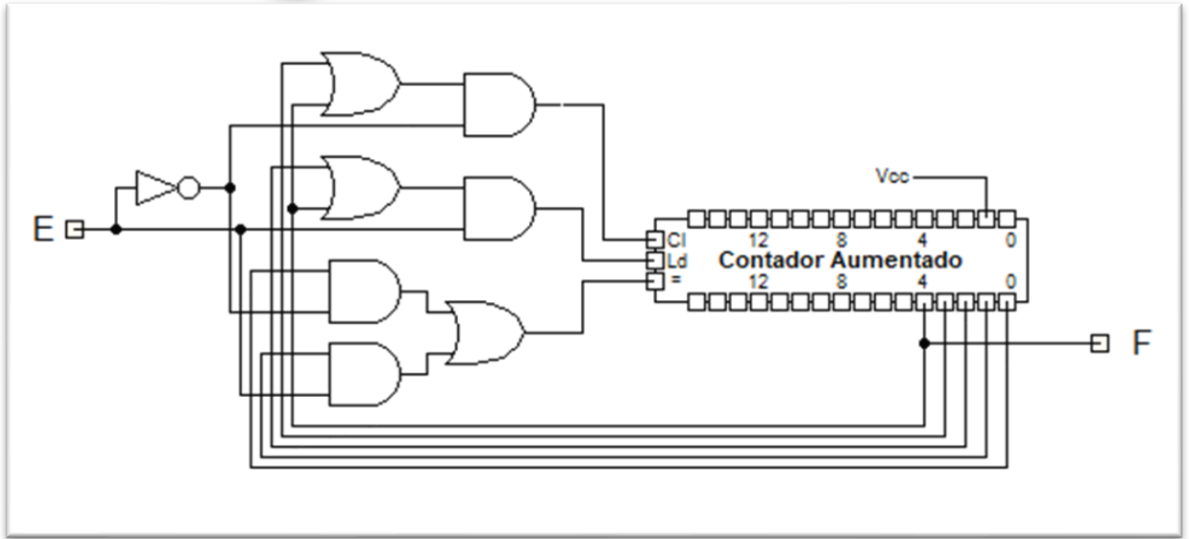


Circuitos Sequenciais com Contadores

- Necessário:
  - Expandir solução a 16 estados
- Carregar:
  - 1 – para os saltos de S2 e S4 para S1
- Controlo S:
  - 00 – clear – limpa
    - $(S_3 + S_4)\bar{E}$
  - 01 – load – carrega
    - $(S_2 + S_4)E$
  - 10 – incrementa
    - Resto
  - 11 – mantém
    - $S_0\bar{E} + S_1E$

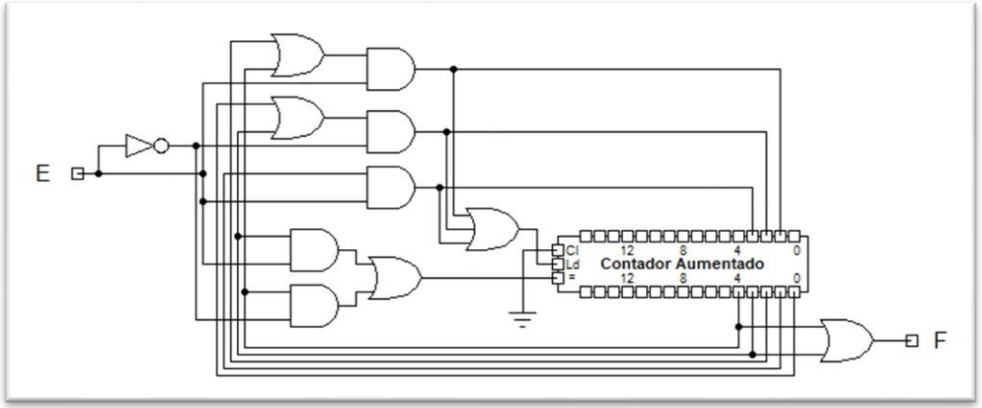


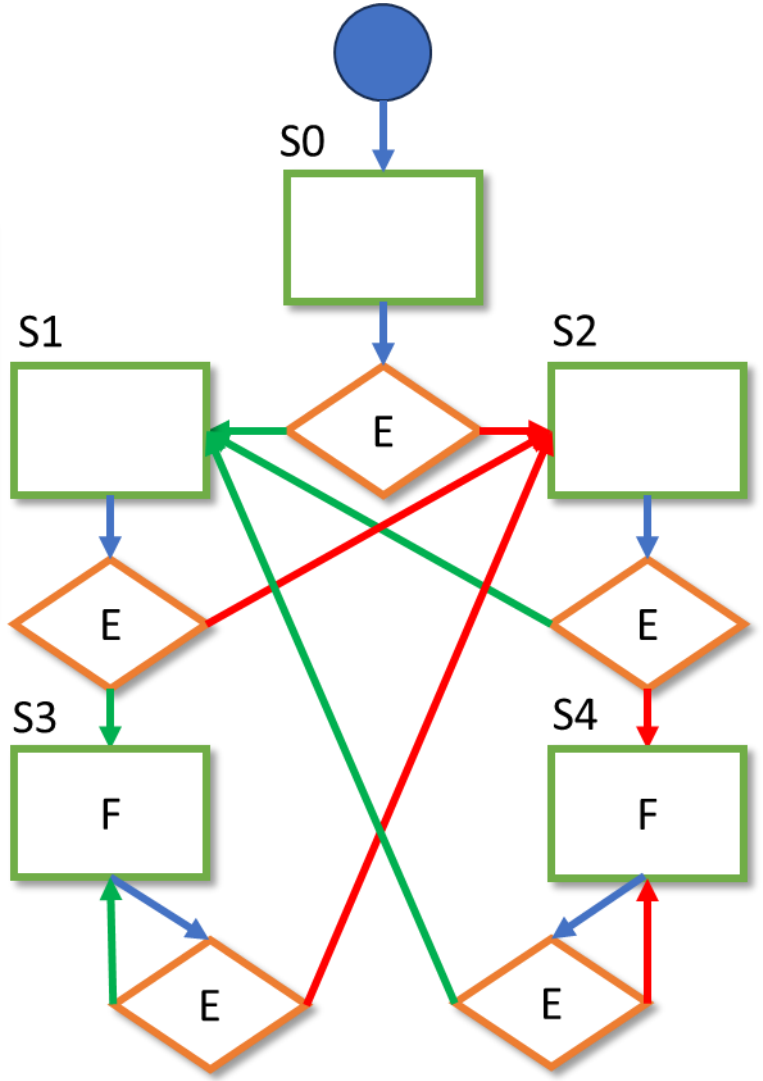




Circuitos Sequenciais com Contadores

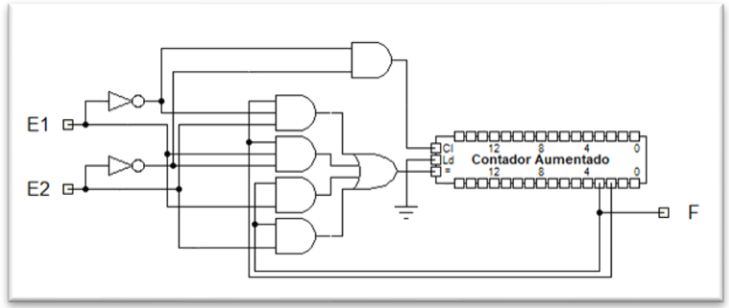
- Fluxogramas: idêntico + simples
- Carregar:
  - 1 - $(S_2 + S_4)E$
  - 2 - $(S_0 + S_3)\bar{E}$
  - 3 - $S_1 E$
- Controlo S:
  - 00 – clear – limpa
    - -
  - 01 – load – carrega
    - $(S_1 + S_2 + S_4)E + (S_0 + S_3)\bar{E}$
  - 10 – incrementa
    - Restantes casos falsos
  - 11 – mantém
    - $S_3 E + S_4 \bar{E}$





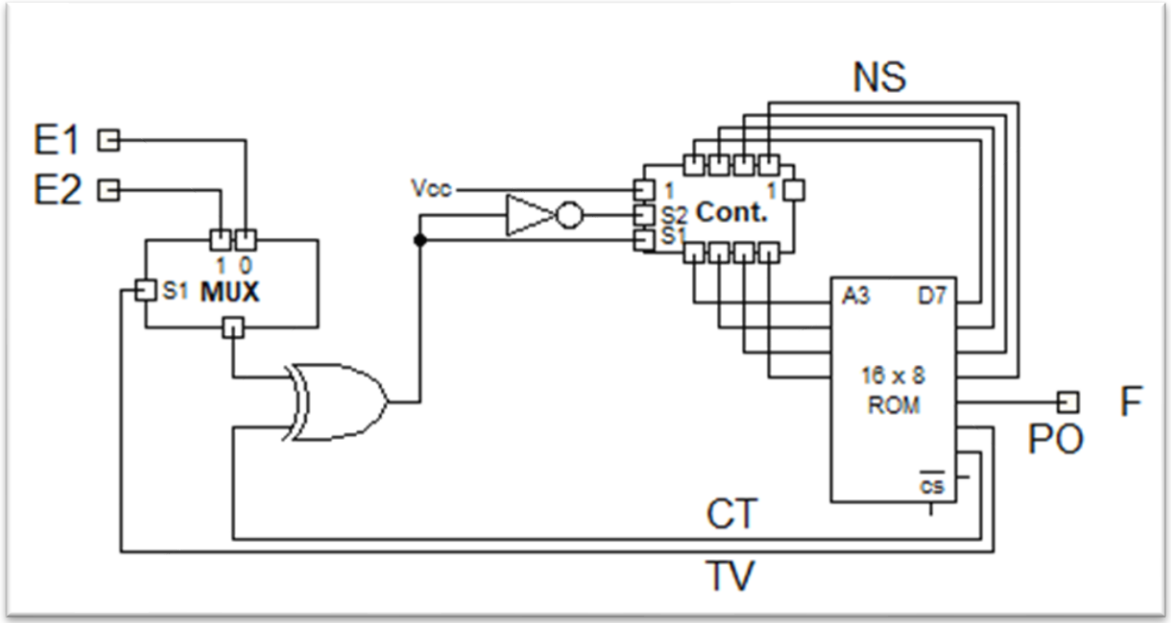
Circuitos Sequenciais com Contadores

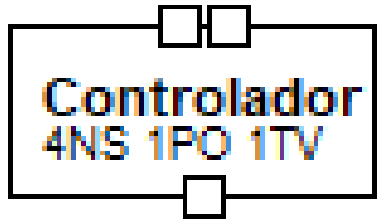
- Fluxograma mais complexo
- Carregar:
- Controlo S:
  - 00 – clear – limpa
    - $\overline{E_1} \cdot \overline{E_2}$
  - 01 – load – carrega
    - -
  - 10 – incrementa
    - Restantes casos falsos
  - 11 – mantém
    - $S_2\overline{E_1}E_2 + S_2E_1\overline{E_2} + S_3E_1 + S_3E_2$

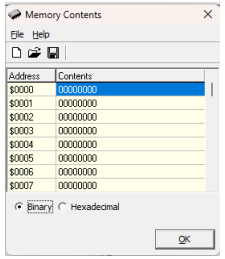


Circuitos Sequenciais Microprogramados

- Máquina de Moore
- Contador:
  - Testa uma só variável
    - 10 - Incrementa se falso
    - 01 - Carrega se verdadeiro
- Utilização de memória
  - NS – valor a carregar
  - PO – variáveis de saída
  - TV – seleção da variável de teste
  - CT – complementar teste
- Editar memória para construir funcionalidade

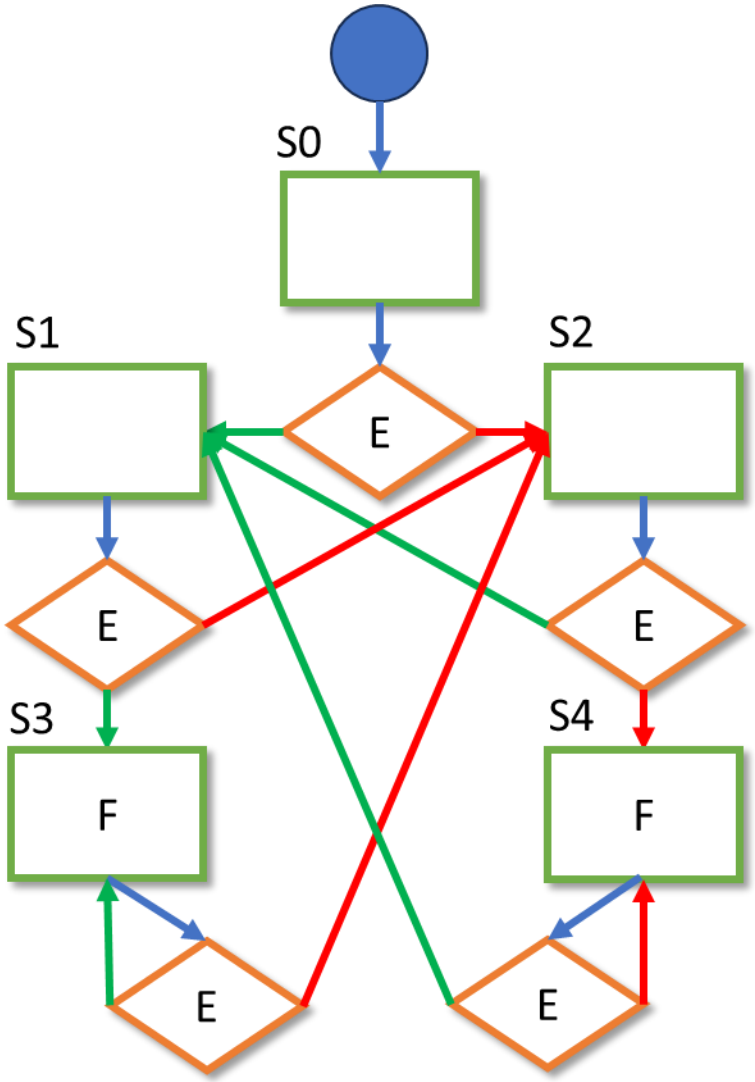






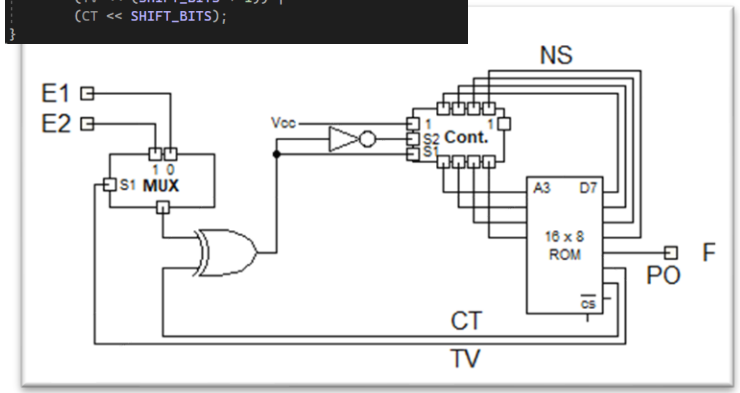
Circuitos Sequenciais Microprogramados

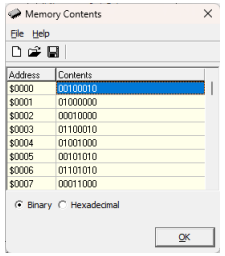
- Editar memória no Digital Works
  - Construção de uma microinstrução
  - Construção de um microprograma
- Duas microinstruções por estado:
  - Estados não incrementais

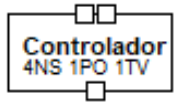



```c
void GravarMapa(char* nome, int *dados, int tamanho)
{
    int cabecalho[] = {43981, 256};
    FILE* f = fopen(nome, "wb");
    if (f != NULL) {
        cabecalho[1] = tamanho;
        fwrite(cabecalho, sizeof(int), 2, f);
        fwrite(dados, sizeof(int), tamanho, f);
        fclose(f);
    }
}
```

```c
#define NS_BITS 4
#define PO_BITS 1
#define TV_BITS 1
#define SHIFT_BITS 1
// Microinstrução
int MI(int NS, int PO, int TV, int CT)
{
    return (NS << (PO_BITS + TV_BITS + SHIFT_BITS + 1)) |
        (PO << (TV_BITS + SHIFT_BITS + 1)) |
        (TV << (SHIFT_BITS + 1)) |
        (CT << SHIFT_BITS);
}
```








```c
void MicroProgramaA() {
    enum valoresCT { CTVerdade = 0, CTFalso };
    enum testeVars { E1 = 0, E2 };
    enum estados { S0 = 0, S1, S2, S3 = 4, S4 = 6 };
    int memoria[16];
    int count = 0;

    // S0
    memoria[count++] = MI(S2, 0, E1, CTFalso);
    // S1
    memoria[count++] = MI(S3, 0, E1, CTVerdade);
    // S2
    memoria[count++] = MI(S1, 0, E1, CTVerdade);
    memoria[count++] = MI(S4, 0, E1, CTFalso);
    // S3
    memoria[count++] = MI(S3, 1, E1, CTVerdade);
    memoria[count++] = MI(S2, 1, E1, CTFalso);
    // S4
    memoria[count++] = MI(S4, 1, E1, CTFalso);
    memoria[count++] = MI(S1, 1, E1, CTVerdade);

    GravarMapa("MicroprogramaA.map", memoria, 16);
}
```

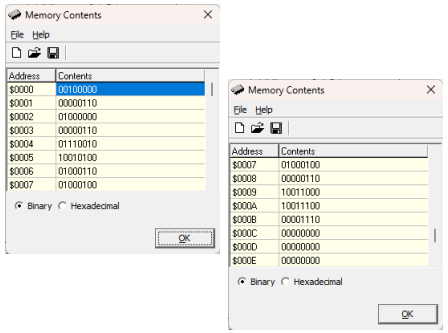
Circuitos Sequenciais Microprogramados

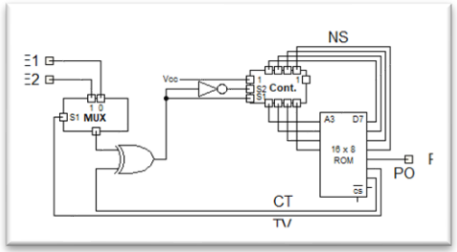
- Máquina de Moore
  - Cada teste uma instrução

```
void MicroProgramaB() {
    enum valoresCT { CTVerdade = 0, CTFalso};
    enum testeVars { E1 = 0, E2 };
    enum estados { S0 = 0, S1=2, S2=4, S3=9 };
    int memoria[16];
    int count = 0;

    // S0
    memoria[count++] = MI(S1, 0, E1, CTVerdade);
    memoria[count++] = MI(S0, 0, E2, CTFalso);
    // S1
    memoria[count++] = MI(S2, 0, E1, CTVerdade);
    memoria[count++] = MI(S0, 0, E2, CTFalso);
    // S2
    memoria[count++] = MI(S2+3, 0, E1, CTFalso);
    memoria[count++] = MI(S3, 0, E2, CTVerdade);
    memoria[count++] = MI(S2, 0, E2, CTFalso);
    memoria[count++] = MI(S2, 0, E2, CTVerdade);
    memoria[count++] = MI(S0, 0, E2, CTFalso);
    // S3
    memoria[count++] = MI(S3, 1, E1, CTVerdade);
    memoria[count++] = MI(S3, 1, E2, CTVerdade);
    memoria[count++] = MI(S0, 1, E2, CTFalso);

    GravarMapa("MicroprogramaB.map", memoria, 16);
}
```





Recursos utilizados

- Microsoft Power Point
- Clipchamp, voz de síntese Fernanda
- Vimeo
- G. Arroz, J. Monteiro, A. Oliveira (2020). Arquitectura de Computadores: dos Sistemas Digitais aos Microprocessadores (5ª edição). IST Press