NEANDER
COMPUTADOR TEÓRICO

Sistemas Digitais - Etapa 04
Alunos: Gabriel Tadioto, Isadora Coelho e Leonardo Calsavara.

ULA (UNIDADE DE LÓGICA E ARITMÉTICA)

Componentes:
- Registrador com carga de 8 bits (AC);
- Registrador com carga de 2 bits (FLAGS);
- Multiplexador 2x8 bits;
- ULA(ALU) - Multiplexador 5x8 bits.

Registrador AC: Armazena a memória vinda da ULA, que futuramente será transformada em x, uma entrada da ULA, e enviado para o multiplexador especial.

Registrador FLAGS: Reset = “01”, em resumo, servem para registrar o sinal da informação vinda da ULA.

ULA (ALU): Realiza todas as operações possíveis, sendo elas selecionadas a partir do dado vindo de Ula_op que é o seletor de seu multiplexador.

Multiplexador 2x8:

MEMÓRIA

Componentes:
- Multiplexador 2x8;
- REM (Registrador com carga de 8 bits);
- Memória 256x8;
- RDM (Registrador com carga de 8 bits).

UNIDADE DE CONTROLE

Componentes
- Program Counter PC (RIP)
- Contador de 0 a 7;
- Registrador (RI)
- Decodificador 8x11:
- Control Unit (UC):

- Aciona os sinais de controle no tempo correto, sendo elas:
  - ➢ NOP;
  - ➢ STA;
  - ➢ LDA;
  - ➢ ADD;
  - ➢ AND;
  - ➢ OR;
  - ➢ NOT;
  - ➢ JMP;
  - ➢ JN;
  - ➢ JZ;
  - ➢ HLT.

Como apenas os passos realizados por cada operação foram fornecidos e não suas simplificações, foi necessário montar suas tabelas verdades para obter uma simplificação:

NOP:

q2	q1	q0	barinc	barpc	ulaop2	ulaop 1	ulaop 0	pc_rw	ac_r w	mem_rw	rem_rw	rdm_rw	ri_rw
0	0	0	1	1	0	0	0	0	0	0	1	0	0
0	0	1	1	1	0	0	0	1	0	0	0	1	0
0	1	0	1	1	0	0	0	0	0	0	0	0	1
0	1	1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0

barinc = (not q2 . not q(1) + not q(2) . not q(0);
barpc = not(q(2) . not q(1) + not q(2) . not q(0);
ulaop = "000";
pc_rw = not q(2) . not q(1) . q(0);
ac_rw = q(2) . q(1) . q(0);
mem_rw = '0';
rem_rw = not q(2) . not q(1) . not q(0);
rdm_rw = not q(2) . not q(1) . q(0);
ri_rw = not q(2) . q(1) and . q(0);

ADD:

b2	b1	b0	barinc	barpc	ulaop2	ulaop1	ulaop0	pc_rw	ac_rw	mem_rw	rem_rw	rdm_rw	ri_rw
0	0	0	1	1	0	0	1	0	0	0	1	0	0
0	0	1	1	1	0	0	1	1	0	0	0	1	0
0	1	0	1	1	0	0	1	0	0	0	0	0	1
0	1	1	1	0	0	0	1	0	0	0	1	0	0
1	0	0	1	0	0	0	1	1	0	0	0	1	0
1	0	1	1	0	0	0	1	0	0	0	1	0	0
1	1	0	1	0	0	0	1	0	0	1	0	1	0
1	1	1	1	0	0	0	1	0	0	0	0	0	0

barinc = ‘1;
barpc = not q(2) + not q(0) + q(1);
ulaop = "001";
pc_rw =(not q(2) . not q(1) . q(0)) + (q(2) . not q(1) not q(0));
ac_rw = q(2) . q(1) . q(0);
mem_rw = '0';
rem_rw = (not q(2) . not q(1) . not q(0)) + (q(2) . not q(1) . q(0)) + (not q(2) . q(1) . q(0));
rdm_rw = (not q(2) . not q(1) . q(0)) + (q(2) . not q(1) . not q(0)) + (q(2) . q(1) . not q(0));
ri_rw = not q(2) . q(1) . not q(0);

AND E OR
Esses dois componentes possuem as mesmas simplificações da ADD, mudando somente a ulaop

NOT:

b2	b1	b0	barinc	barpc	ulaop2	ulaop 1	ulaop0	pc_rw	ac_rw	mem_rw	rem_rw	rdm_rw	ri_rw
0	0	0	1	1	1	0	0	0	0	0	1	0	0
0	0	1	1	1	1	0	0	1	0	0	0	1	0
0	1	0	1	1	1	0	0	0	0	0	0	0	1
0	1	1	0	1	1	0	0	0	0	0	0	0	0
1	0	0	1	1	1	0	0	0	0	0	0	0	0

1	0	1	1	1	1	0	0	0	0	0	0	0	0
1	1	0	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0

barinc = 1;
barpc =1;
ulaop = "100";
pc_rw = not q(2) . not q(1) . q(0);
ac_rw = q(2) . q(1) . q(0);
mem_rw = '0';
rem_rw = not q(2) . not q(1) . not q(0);
rdm_rw = not q(2) . not q(1) . q(0);
ri_rw = not q(2) . q(1) . not q(0);

STA

b2	b1	b0	barinc	barpc	ulaop 2	ulaop 1	ulaop 0	pc_rw	ac_rw	mem_rw	rem_rw	rdm_rw	ri_rw
0	0	0	1	1	0	0	0	0	0	0	1	0	0
0	0	1	1	1	0	0	0	1	0	0	0	1	0
0	1	0	1	1	0	0	0	0	0	0	0	0	1
0	1	1	1	1	0	0	0	0	0	0	1	0	0
1	0	0	1	1	0	0	0	1	0	0	0	1	0
1	0	1	1	0	0	0	0	0	0	0	1	0	0
1	1	0	1	0	0	0	0	0	0	1	1	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0

barinc = '1';
barpc = not q(2) + q(1) + not q(0);
ulaop = "000";
pc_rw = not q(1) . (q(2) xor q(0));
ac_rw = '0';
mem_rw = q(2) and q(1) and not q(0);
rem_rw = (not(q(1)) . (q(2) xnor q(0))) + (not(q(2)) . q(1) . q(0));
rdm_rw = not q(1) . (q(2) xor q(0));
ri_rw = not q(2) . q(1) . not q(0);

JMP

b2	b1	b0	barinc	barpc	ulaop2	ulaop 1	ulaop 0	pc_rw	ac_rw	mem_rw	rem_rw	rdm_rw	ri_rw
0	0	0	1	1	0	0	0	0	0	0	1	0	0
0	0	1	1	1	0	0	0	1	0	0	0	1	0
0	1	0	1	1	0	0	0	0	0	0	0	0	1
0	1	1	1	1	0	0	0	0	0	0	1	0	0
1	0	0	1	1	0	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	1	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0

barinc = not q(2) + q(1) + not q(0);
barpc = not q(2) + q(1) + not q(0);
ulaop = "000";
pc_rw = not q(1) . q(0);
ac_rw = '0';
mem_rw = '0';
rem_rw = (not q(1) . (q(2) xnor q(0))) + (not q(2) . q(1) . q(0));
rdm_rw = not q(1) . (q(2) xor q(0));
ri_rw = not q(2) . q(1) . not q(0);

JMP FALSO

b2	b1	b0	barinc	barpc	ulaop2	ulaop 1	ulaop0	pc_rw	ac_rw	mem_rw	rem_rw	rdm_rw	ri_rw
0	0	0	1	1	0	0	0	0	0	0	1	0	0
0	0	1	1	1	0	0	0	1	0	0	0	1	0
0	1	0	1	1	0	0	0	0	0	0	0	0	1
0	1	1	1	1	0	0	0	1	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0

barinc = 1;

barpc = 1;
ulaop = "000";
pc_rw = not q(2) . q(0);
ac_rw = '0';
mem_rw = '0';
rem_rw = not q(2) . not q(1) . not q(0);
rdm_rw = not q(2) . not q(1) . q(0);
ri_rw = not q(2) . q(1) . not q(0);

HLT
Não executa nenhum passo, portanto todos os seus bits recebem 0.

LDA

b2	b1	b0	barinc	barpc	ulaop2	ulaop 1	ulaop 0	pc_rw	ac_rw	mem_rw	rem_rw	rdm_rw	ri_rw
0	0	0	1	1	0	0	0	0	0	0	1	0	0
0	0	1	1	1	0	0	0	1	0	0	0	1	0
0	1	0	1	1	0	0	0	0	0	0	0	0	1
0	1	1	1	1	0	0	0	0	0	0	1	0	0
1	0	0	1	1	0	0	0	1	0	0	0	1	0
1	0	1	1	0	0	0	0	0	0	0	1	0	0
1	1	0	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	1	0	0	0	0	1	0	0	0	0

barinc = '1';
barpc = not q(2) + q(1) + not q(0);
ulaop = "000";
pc_rw = not q(1) . (q(2) . q(0));
ac_rw = q(2) . q(1) . q(0);
mem_rw = '0';
rem_rw = (not q(1) . (q(2) xnor q(0))) + (not q(2) . q(1) . q(0));
rdm_rw = (q(2) and . q(0)) + (not q(2) . not q(1) . q(0));
ri_rw = not q(2) . q(1) . not q(0);

OBS: Para a realização do jn e jz (jmp falso) foi realizado algumas linhas de código que simulam um multiplexador nos seguintes barramentos de controle: barinc, pc_rw, rem_rw e rdm_rw. Logo, quando o seletor é ‘1’, o jmp é verdadeiro, caso contrário, o falso.
O seletor é referente a saída da flag NZ.