SISTEMAS DIGITAIS

NÍVEL DE TRANSFERÊNCIA ENTRE REGISTRADORES

Prof. Luís Caldas

2017

ÍNDICE

FLUXO DE DADOS – PROJETO RTL

Introdução: O estudo de circuito digital básico focaliza o desenvolvimento de subsistemas digitais combinacionais e seqüenciais. As unidades funcionais são os meios para a realização de operações complexas, como, por exemplo, a adição de vários números ou mesmo a comparação entre números e as suas ordenações. A questão é como projetar um circuito, a fim de realizar operações mais complexas ou realizar operações que envolvam múltiplos passos?
Um exemplo de um problema mais complexo é como desenvolver um circuito capaz de adicionar quatro números ou ainda a questão mais radical, como adicionar um milhão de números?
Para adicionar quatro números é possível conectar quatro somadores juntos, mas para adicionar um milhão de números, não se pode adicionar um milhão de somadores juntos. A resposta para as operações de múltiplos passos é "fluxo de dados", onde é possível resolver realmente o problema e realizar e coordenar as operações, as quais envolvem múltiplos passos. A partir de algoritmos detalhados, é possível desenvolver sistemas digitais completos partindo inicialmente na adequação do fluxo de dados para solução do algoritmo e implementar em seguida a unidade de controle para uma função dedicada, uma FSM ou máquina de estados finitos. Essa abordagem de implementação é chamada de sistema RTL, sistemas em nível de transferência entre registradores. A fim de estudar como desenvolver e projetar máquina para esta finalidade, inicialmente temos que conhecer funcionalmente cada componente do sistema digital completo e iniciaremos o projeto RTL com o estudo do grafo de execução.

GRAFO DE EXECUÇÃO

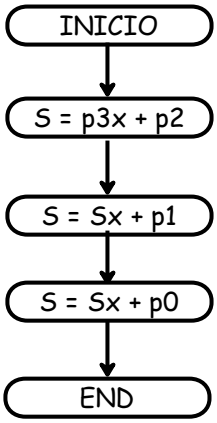
A abordagem RTL é dividida em dois subsistemas digitais: subsistema "datapath" ou fluxo de dados e o subsistema "unidade de controle". O conteúdo do registrador determina o estado do sistema. A função do sistema é realizada através de passos em uma seqüência de transferências entre registradores e em muitos ciclos de relógio. Essa seqüência de transferência dos dados é realizada de um registrador para outro. A unidade de controle comanda através de sinais de controle a seqüência de operações no fluxo de dados. A seguir apresentamos tipos de grafos de execuções para a realização de um sistema s(x).

Exemplo: Encontrar uma solução para o sistema $s(x) = p_3x^3 + p_2x^2 + p_1x + p_0$, Usar grafo de execução do tipo seqüencial:

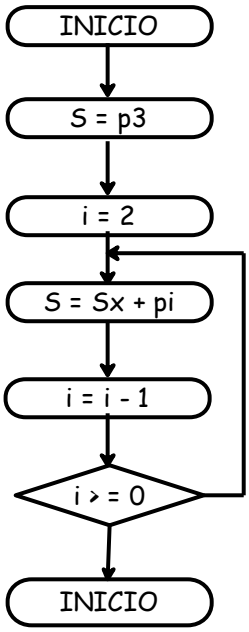
a) Desdobrando a expressão – Operação seqüencial

$s(x) = ((p_3x + p_2)x + p_1)x + p_0$.

a.1) Grafo de execução

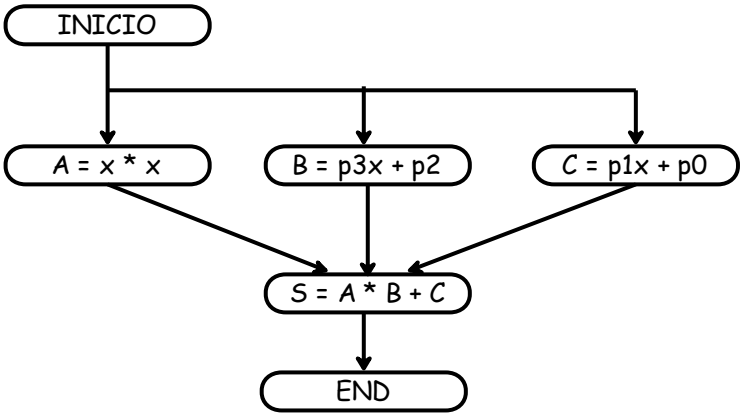


a.2) Loop



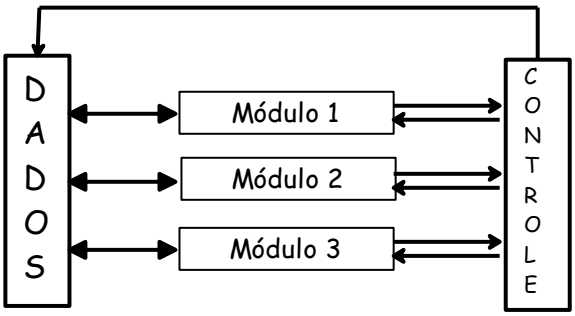
b) Evidenciando x na expressão s(x) – Operação concorrente

$s(x) = x^2(p_3x + p_2) + (p_1x + p_0)$

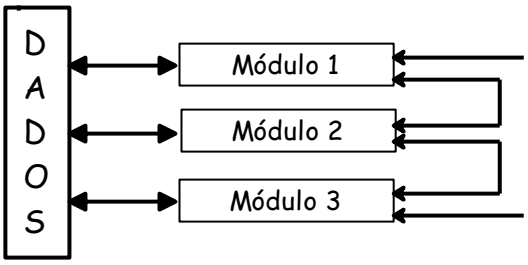


ORGANIZAÇÃO DE SISTEMAS – Um sistema pode ser organizado quanto a forma de controle em centralizado, descentralizado e semicentralizado.

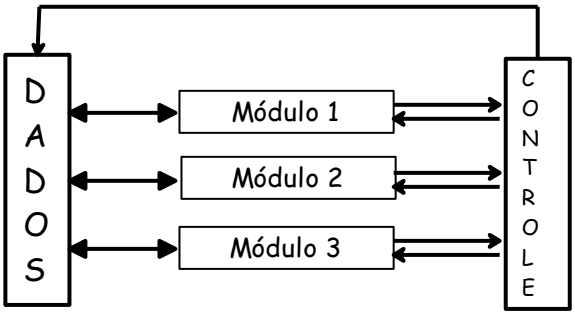
a) Centralizado – A unidade de controle controla todo o sistema, gerando sinais de controle para as unidades funcionais, como, multiplexadores, unidades aritméticas, registradores entre outros e recebe sinais que evidenciam condições.



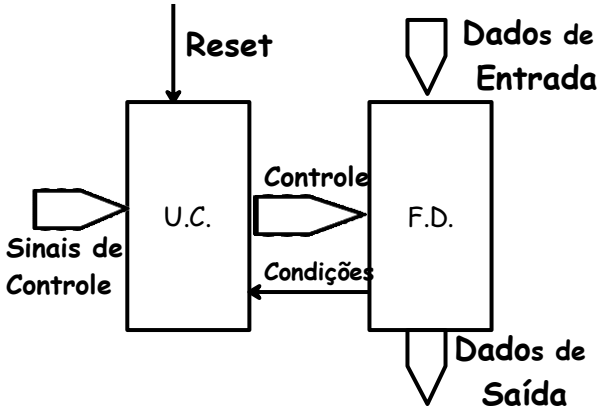
b) Descentralizado – Cada módulo no sistema contém mecanismos para controlar a própria operação no modo seqüencial. A unidade de controle está dentro de cada módulo e individual.



c) Semidescentralizado – É uma combinação dos tipos centralizada e descentralizada, onde o controle do módulo é interno ao módulo e individual, mas a unidade de controle controla os dados de entrada e saída dos módulos como um todo e centralizado.



Nota: Só estudaremos controle centralizado. A execução dos grafos é seqüencial ou grupo seqüencial. A organização RTL consiste em dois subsistemas "fluxo de dados" e unidade de controle.



NÍVEL DE TRANSFERÊNCIA ENTRE REGISTRADORES

INTRODUÇÃO: Nos projetos de circuitos digitais e integrados, a descrição da operação de um circuito digital síncrono é denominada como operação em RTL (do inglês Register Transfer Level). No projeto digital completo, duas unidades definidas como fluxo de dados e de controle compõe o sistema. O fluxo de dados é responsável por toda a operação e manipulação dos dados e gera sinais necessários para a unidade de controle enquanto essa unidade é responsável pela evolução e coordenação do fluxo de dados gerando sinais para as unidades funcionais do fluxo de dados. O projeto deve ser implementado como uma seqüência de operações realizadas as quais denominaremos de instruções. Cada instrução realizada será feita num ciclo de relógio, entrada para a unidade de controle e fluxo de dados.

PROCESSO DE PROJETO CIRCUITOS DIGITAIS

O processo de projeto de circuitos digitais pode ser descrito em duas fases sendo captura e conversão para circuito. Por exemplo:

Tipo de circuito	Captura	Conversão
Combinatório	Comportamento do circuito por: Equação booleana ou tabela da verdade	Converte o comportamento Para circuito.
Seqüencial	Comportamento do circuito por: equação de estados e saída (FSM)	Converte o comportamento Para circuito.
RTL	Comportamento do processador por: FSM de alto nível RTL	Converte o comportamento Para circuito.

MÉTODO DE PROJETO DO RTL.

O método de projeto RTL será realizado, conforme acima em quatro passos como descritos a seguir na tabela a seguir.

Passo 1: Processo de captura o qual descreve o comportamento do circuito através de um algoritmo com operações a serem realizadas dentro de uma seqüência desejada;

Passo 2: Definição do fluxo de dados;

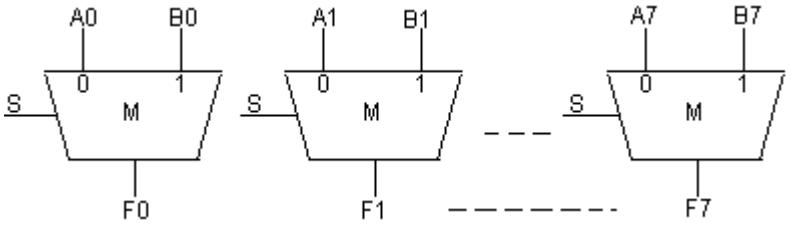
Passo 3: Projeto do controlador ou unidade de controle;

Passo 4: Conversão do algoritmo em instruções da FSM de alto nível e definindo as palavras de controle a serem geradas pela unidade de controle ao fluxo de dados e os sinais recebidos pela unidade de controle do fluxo de dados.

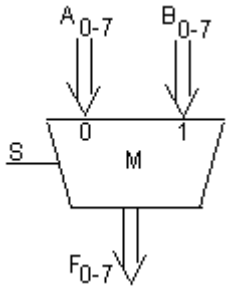
Obs.: Um quinto passo é necessário para definir a freqüência do relógio e o tipo de transição das unidades funcionais, subida ou descida.

UNIDADES FUNCIONAIS

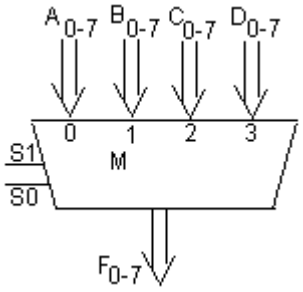
As unidades funcionais são partes integrantes do fluxo de dados e podem ser do tipo: comparadores. ULA, registradores de arquivos, registradores de propósito geral, contadores, multiplexadores e outros. Embora são circuitos estudados em circuitos digitais combinatórios e seqüenciais, vamos mostrar cada um deles começando pelo multiplexador. Para a construção do multiplexador para N bits deve-se associar MUX para a sua construção. Por exemplo associando MUX de 2 x 1, para a construção de um multiplexador de entradas A e B, sendo (A_{0-7} e B_{0-7}) de 8bits.



A representação do bloco multiplexador para 2 entradas A e B de 8 bits cada.

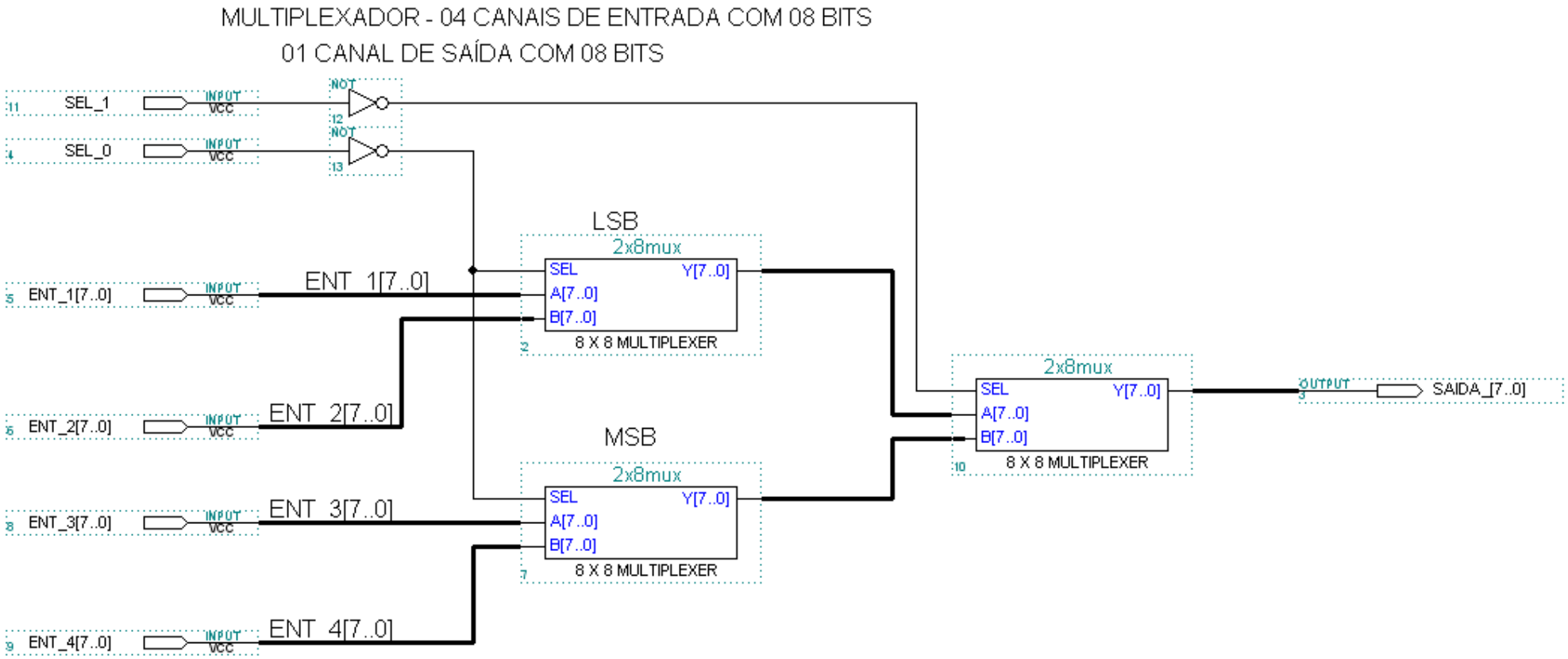


A representação do bloco multiplexador para 4 entradas A,B,C e D de 8 bits cada.

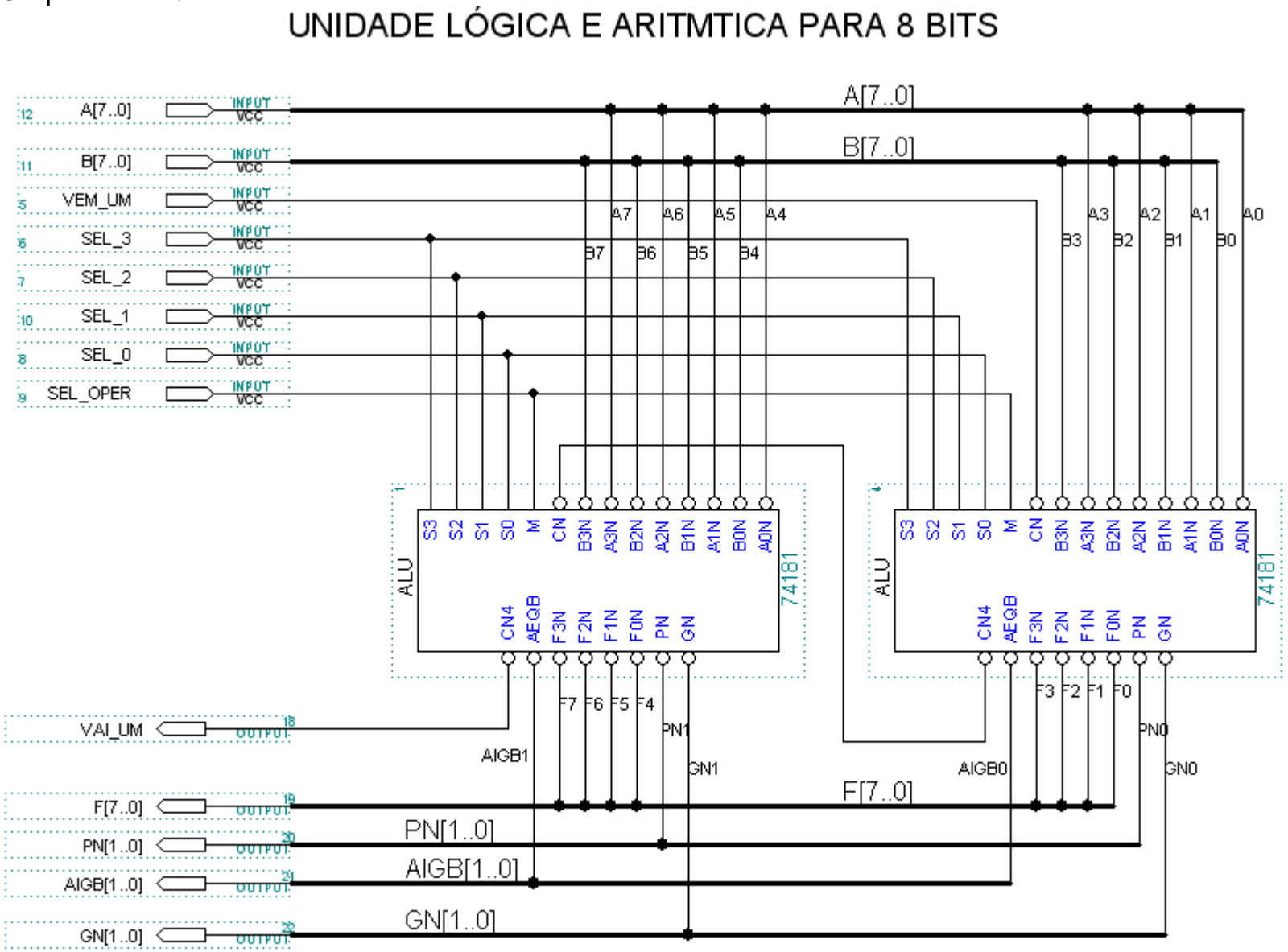


A seguir apresentamos a associação de MUX de 4 entradas de um circuito comercial,

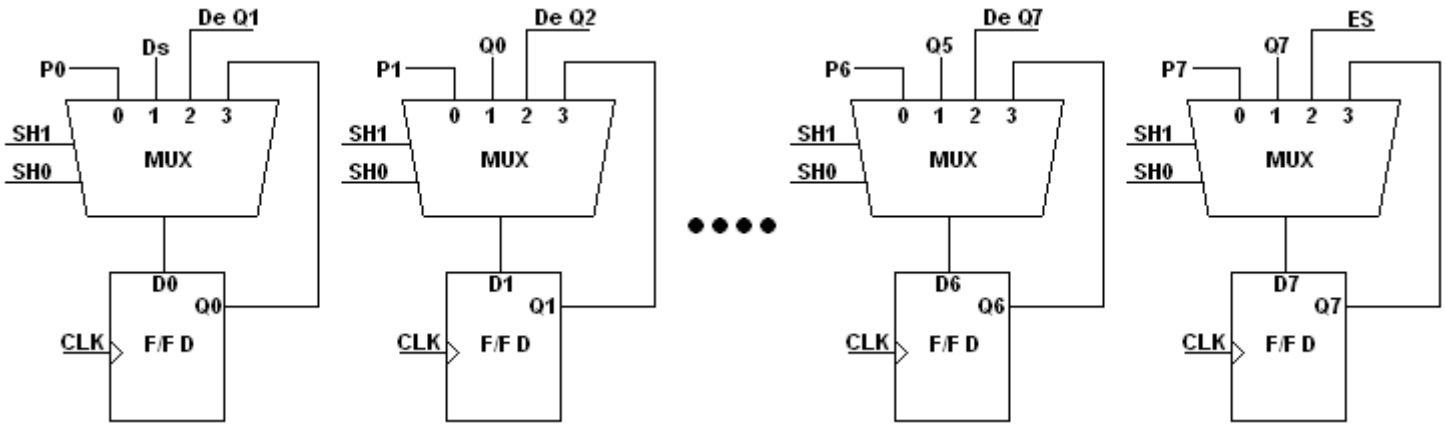
a) MULTIPLEXADOR – É um circuito combinatório e de seleção de dados com capacidade para selecionar até quatro canais de entrada de 4 a 8 bits.



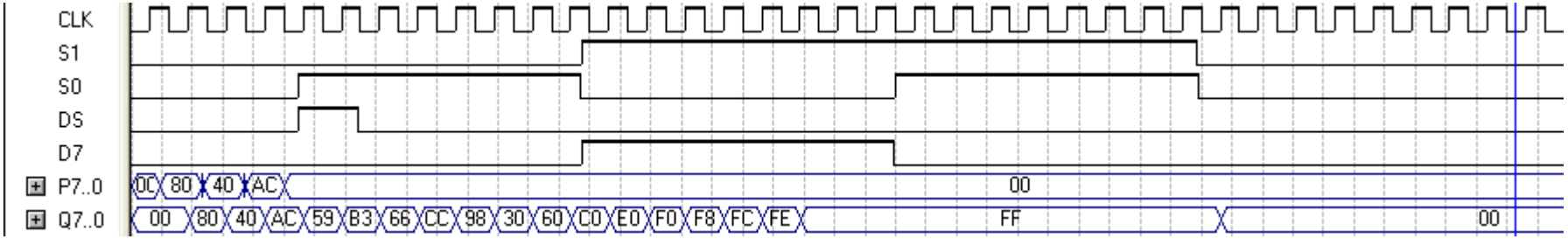
b) ULA para 8 bits.



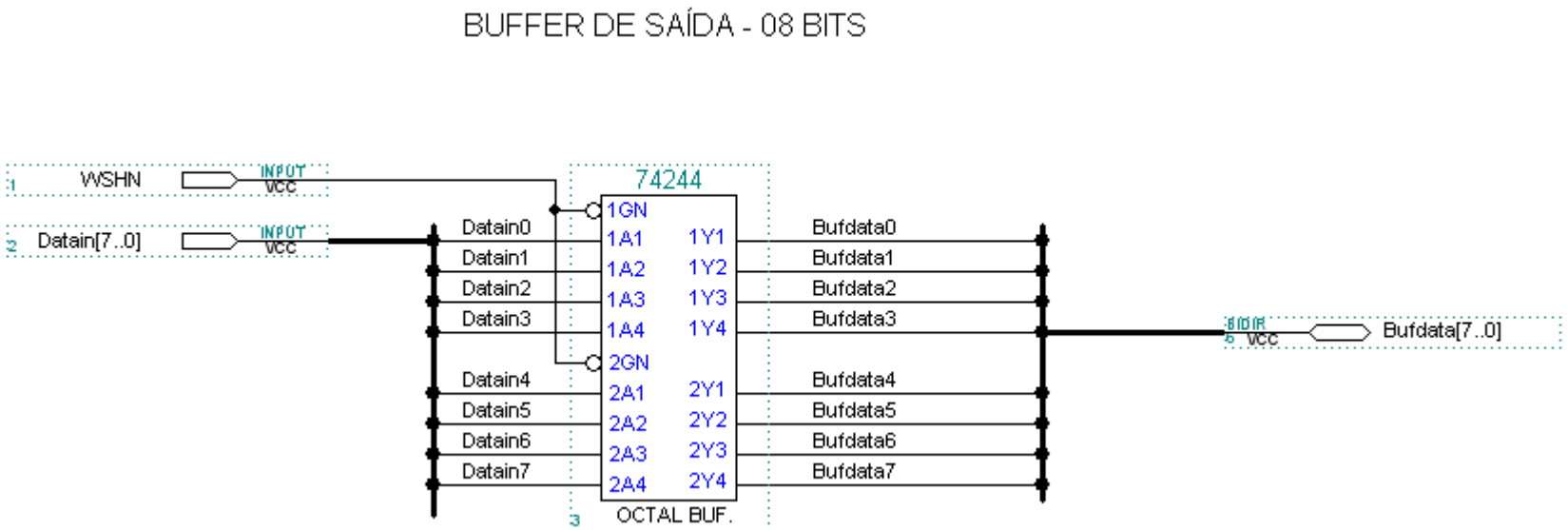
c) SHIFTER de 8 bits



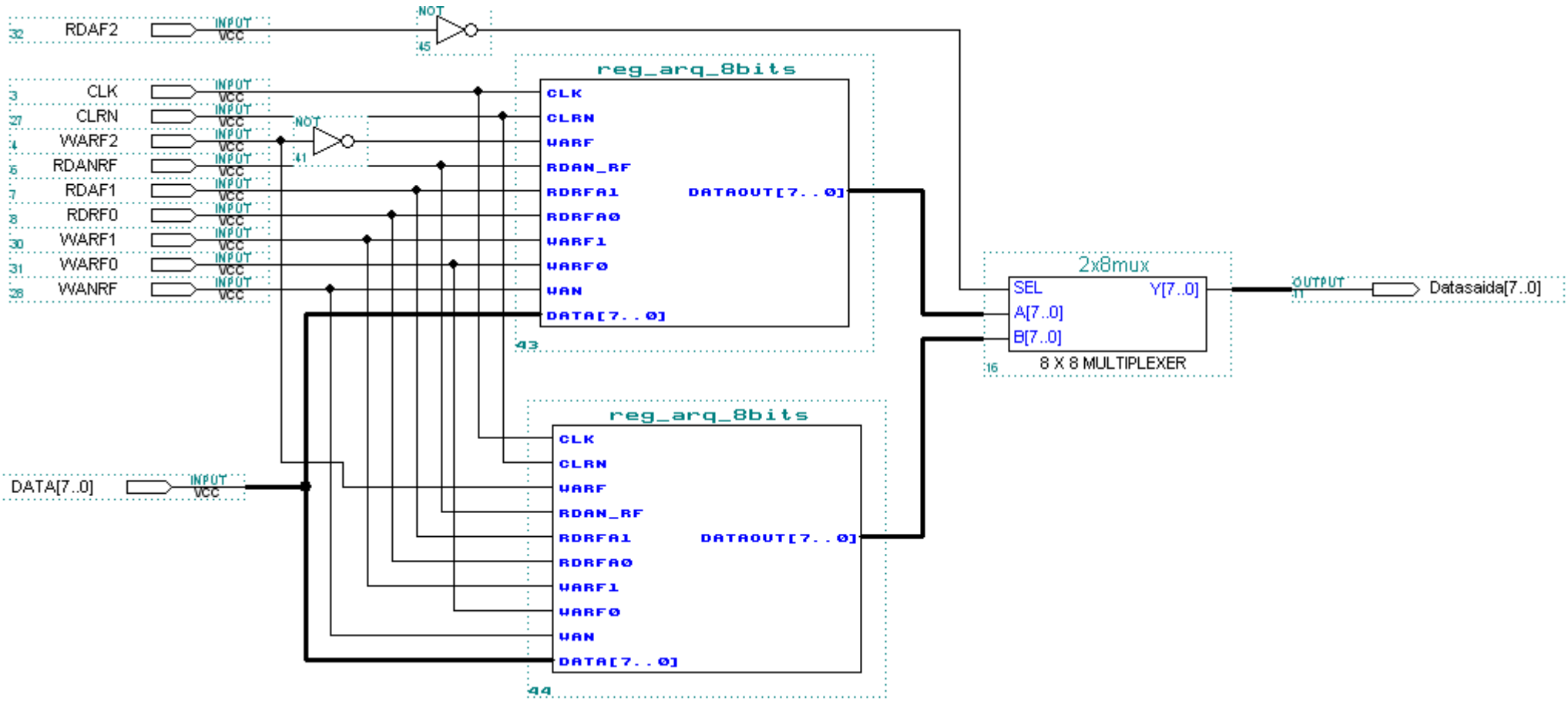
Formas de ondas.



e) Saída 8 bits.



d) Registrador de arquivos de 8 bits.



NÍVEL DE ABSTRAÇÃO DO RTL.

A abstração RTL é usada em linguagens de descrição de hardware como Verilog e VHDL na representação do circuito de baixo nível do qual se derivam as conexões do hardware.
A descrição RTL é usualmente convertida para a descrição de circuitos no nível de porta por uma ferramenta de síntese lógica. Esta síntese resultante representa a descrição física do circuito. As ferramentas de simulação lógica podem utilizar a descrição RTL para verificar sua funcionalidade. Se existe um caminho lógico cíclico da saída de um registrador para a sua entrada (ou da saída de um conjunto de registradores para suas entradas), este circuito é chamado máquina de estados. Se existem caminhos lógicos de um registrador para outro, que operam sem a necessidade de ciclos de relógio, estes blocos são chamados "pipelines".

FLUXO DE DADOS E UNIDADE DE CONTROLE

I - INTRODUÇÃO

Em circuitos digitais o objetivo é estudar e projetar unidades funcionais para realizar operações na ULA como, por exemplo, a adição de dois números ou a comparação entre dois valores. A questão é como projetar um circuito a fim de realizar operações mais complexas ou realizar operações que envolvam múltiplos passos?

Um exemplo pode ser o desenvolvimento de um circuito capaz de adicionar quatro números ou ainda a questão mais radical, como adicionar um milhão de números? Para adicionar quatro números é possível conectar quatro somadores juntos, mas para adicionar um milhão de números, não se pode adicionar um milhão de somadores juntos.

A resposta para as operações de múltiplos passos é utilizar um "fluxo de dados", onde é possível resolver adaptar o circuito a fim de realizar as operações que envolvem múltiplos passos. A arquitetura será dedicada a sua realização.

A partir de algoritmos detalhados, é possível desenvolver sistemas digitais complexos começando pela adequação do fluxo de dados na solução do algoritmo e projeto em seguida a unidade de controle.

A fim de estudar como desenvolver e projetar um sistema digital completo com unidade de controle comandando o fluxo de dados para uma dedicada finalidade, inicialmente temos que conhecer funcionalmente cada componente do sistema digital completo e iniciaremos pelo estudo do fluxo de dados.

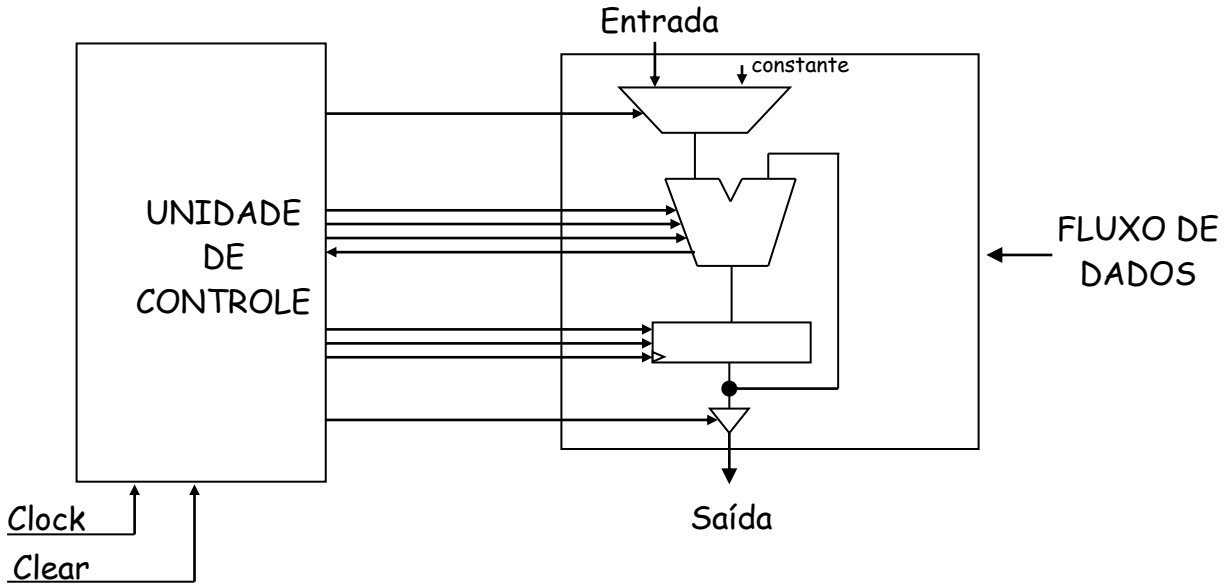
Composição funcional do Fluxo de dados

A arquitetura do fluxo de dados dentro de um sistema digital completo é responsável por :

1) Unidades funcionais como somadores, multiplicadores, ULAs, comparadores e manipuladores de dados;

2) Registradores e outros elementos para passagem, armazenagem temporária dos dados e outras;

3) Canais de comunicações (barramentos), multiplexadores para a transferência de dados entre os diversos componentes no fluxo de dados. Os dados externos podem ser colocados no fluxo de dados através das linhas de entradas dos dados. Os resultados da computação são apresentados através das linhas de saída de dados.
A arquitetura apresentada a seguir é de um sistema digital completo, onde a unidade de controle gerencia e controla o fluxo de dados.



Sinais de controle e de habilitação e status do Fluxo de dados

Para o funcionamento correto do fluxo de dados, os sinais próprios da unidade de controle devem ser definidos em uma base de tempo correta. Os sinais da unidade de controle são enviados para todas as linhas de controle e de seleção, ou seja para todos os componentes do fluxo de dados. Isto inclui todas as linhas de seleções para os multiplexadores, ULAs e outras unidades funcionais que realizam operações múltiplas, as linhas de habilitação de leitura/ escrita dos registradores, as linhas de habilitação dos registradores de arquivos, as linhas de endereçamento para cada uma das locações destes registradores e os sinais de habilitação dos buffers tri-state.

A operação do fluxo de dados é determinada pelos sinais da unidade de controle, os quais são assertivos por um período de tempo. Nos microprocessadores estes sinais de controle são gerados por uma unidade de controle através da seqüência de operações gravadas em uma memória com uma programação interna, o qual coordenada a execução de uma instrução.

Em contrapartida o fluxo de dados necessita fornecer sinais de "status" que retornam para a unidade de controle, a fim de decidir a próxima operação a ser realizada e enfim poder operar correntemente. Os sinais de status são usualmente vindos das saídas dos comparadores, da ULA ou de unidades de contagem. Por exemplo, um sinal de status do comparador no teste de comparação entre dois valores; o resultado da operação é um bit que

indica certa condição lógica e assim a unidade de controle quando lê este bit de teste poderá decidir qual o próximo caminho a evoluir. Os sinais de status apresentam a informação de entrada para a unidade de controle determinar qual a próxima operação a ser realizada. Outro exemplo do sinal de status gerado por unidades de contagem é numa situação de loop condicional, o bit de status de contagem informará à unidade de controle se a operação deve se repetir ou deixar o loop.

Para a operação no fluxo de dados, os dados podem ser obtidos dos elementos de memória ou diretamente da saída das unidades funcionais ou através de elementos ligados no circuito como constantes. Uma vez que o fluxo de dados realiza todas as operações que um microprocessador necessita e o microprocessador foi implementado para a solução de problemas, o fluxo de dados deverá ser capaz de realizar todas as operações requeridas a fim de resolver um determinado problema. Quais são os componentes então que um fluxo de dados deverá conter para a solução de um problema específico?

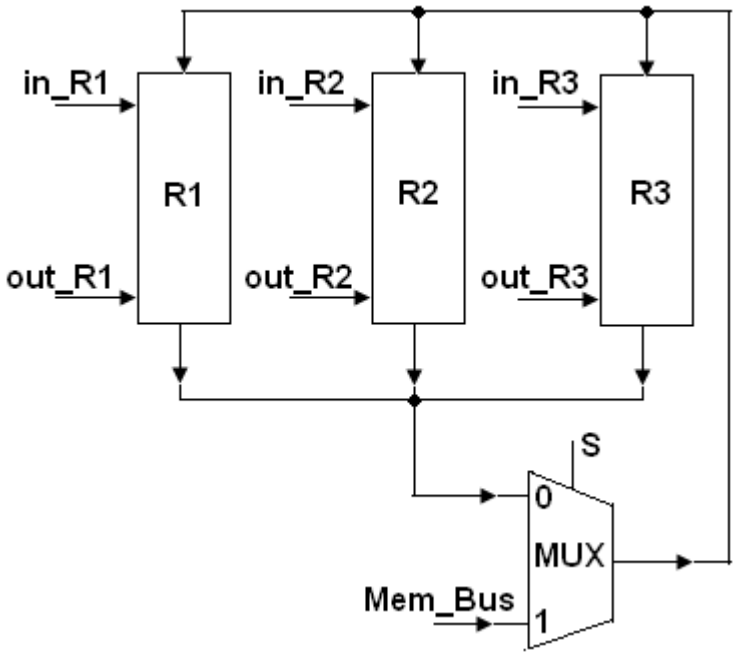
- Se o problema requer a adição de dois números, o fluxo de dados deverá conter um somador;

  - Se o problema requer armazenagem temporária de três variáveis, o fluxo de dados deve conter três registradores.

O fluxo de dados deve-se adequar ao problema e deve ser entendido como uma determinada roupa que serve sob medida a uma determinada pessoa. É com esse intuito que são construídas as máquinas dedicadas. Em seguida apresentamos os passos para o projeto do fluxo de dados.

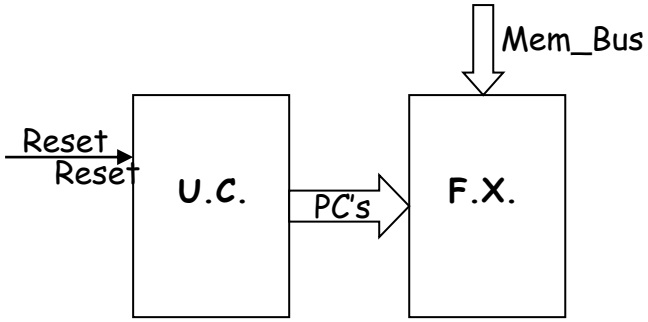
Exemplo: Transferência entre registradores de dados executar na seqüência as instruções:

1. Mem_Bus $\longrightarrow$ R_1
2. R_3 $\longrightarrow$ R_2
3. R_3 $\longleftarrow$ Mem_Bus.

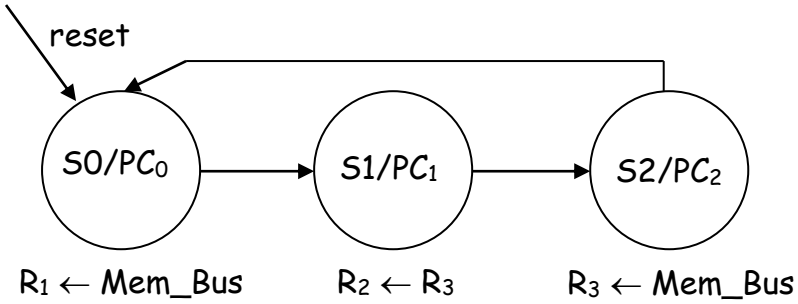


Pede-se:
a) Representação esquemática do sistema digital completo.
b) Projeto da UC
c) Quadro de instruções.

a) Representação esquemática do sistema digital completo.



b) Diagrama de estado de descrição da F.S.M. da U.C.



c) Quadro de instruções

item	instrução	S	in_R_1	in_R_2	in_R_3	out_R_1	out_R_2	out_R_3	PC's
1	$R_1 \leftarrow$ Mem_Bus	1	1	0	0	0	0	0	PC_0
2	$R_2 \leftarrow R_3$	0	0	1	0	0	0	1	PC_1
3	$R_3 \leftarrow$ Mem_Bus	1	0	0	1	0	0	0	PC_2

Implementação da F.S.M.

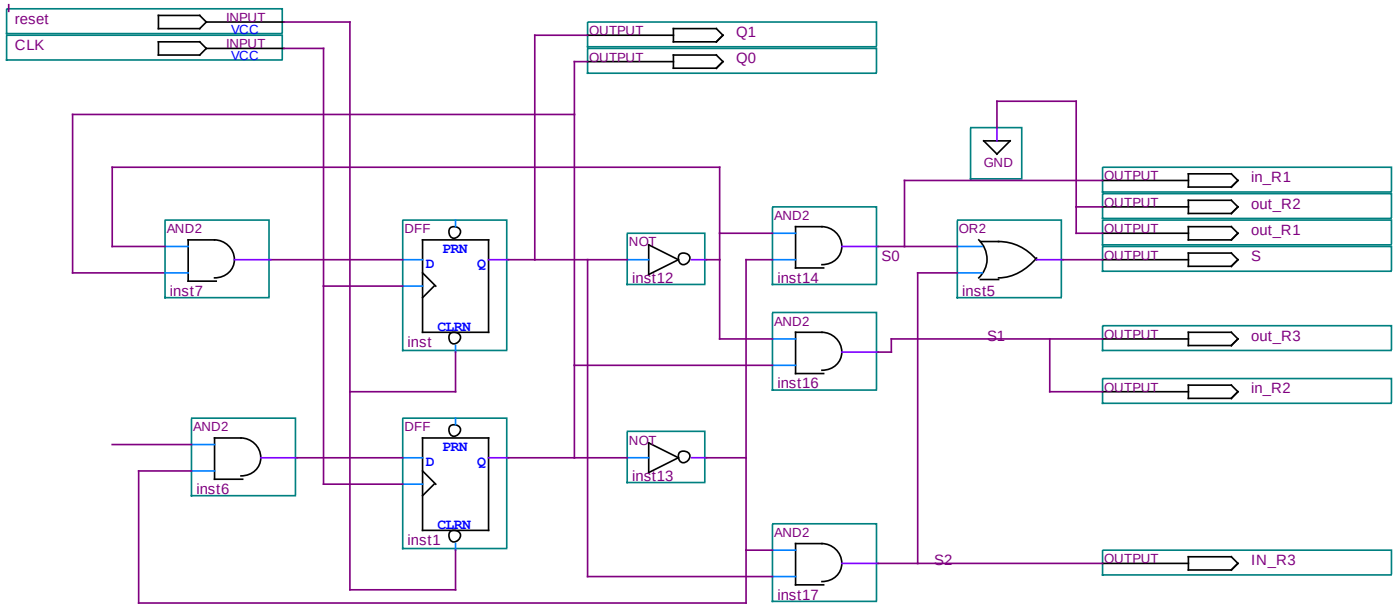
Atual	Futuro	-
Q_1Q_0	Q_1Q_0	PC's
S0 - 00	01	PC_0
S1 - 01	10	PC_1
S2 - 10	00	PC_2
S3 - 11	-	-

Equações booleanas entradas/saídas.

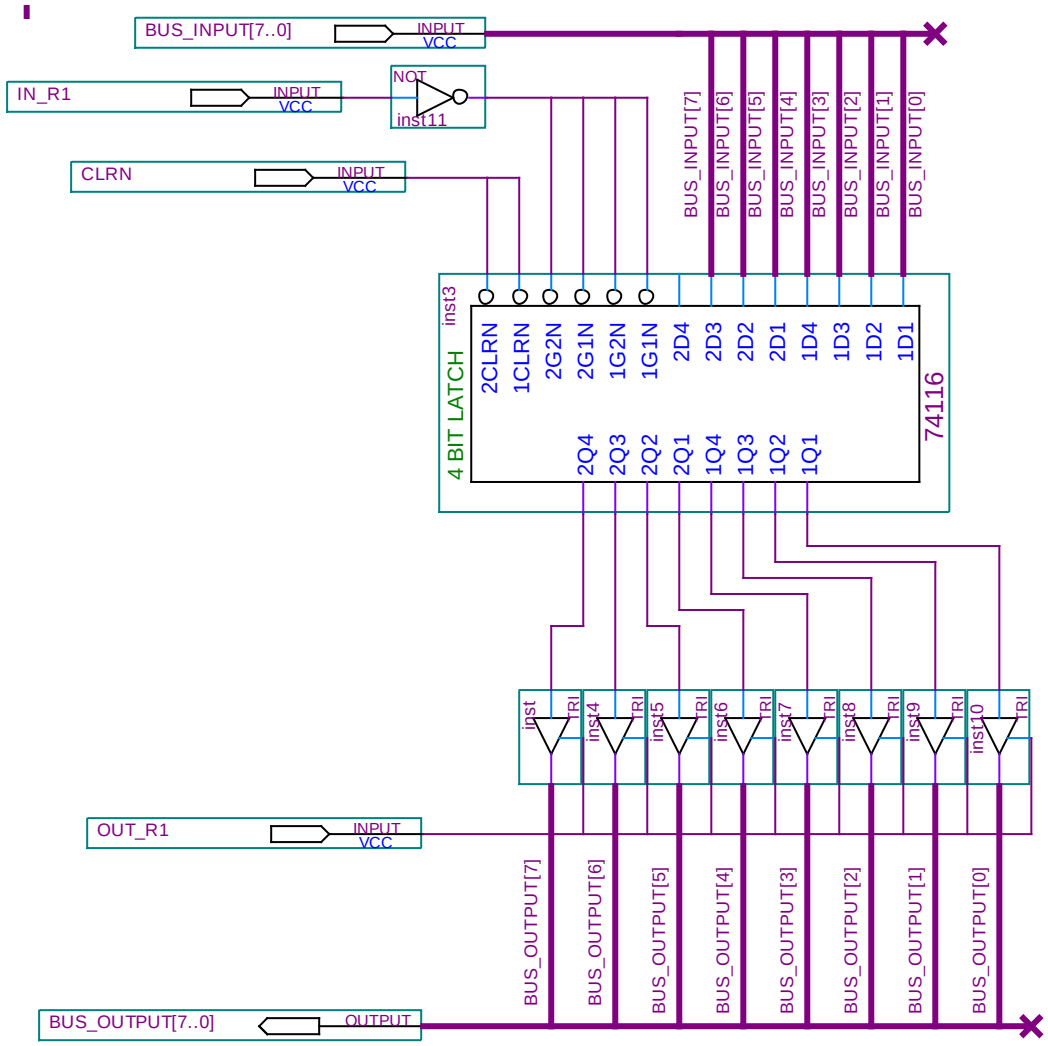
$D_1 = Q_1'Q_0$
$D_0 = Q_0'Q_1'$
$PC_0 = (60)_H$
$PC_1 = (11)_H$
$PC_2 = (48)_H$

$S = S0 + S2;$
$in\_R_1 = S0;$
$in\_R_2 = S1;$
$in\_R_3 = S_2;$
$out\_R_1 = 0;$
$out\_R_2 = 0;$
$out\_R_3 = S1$

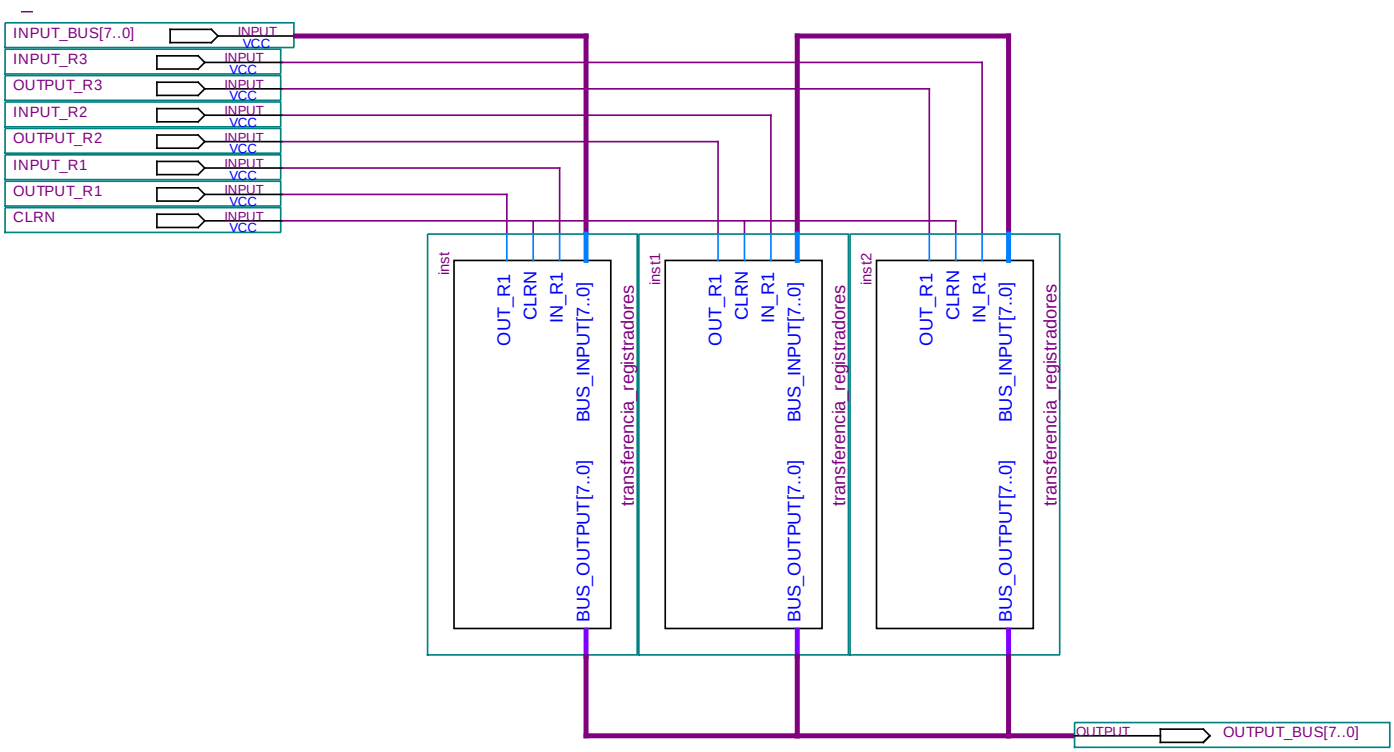
Circuito final da U.C.



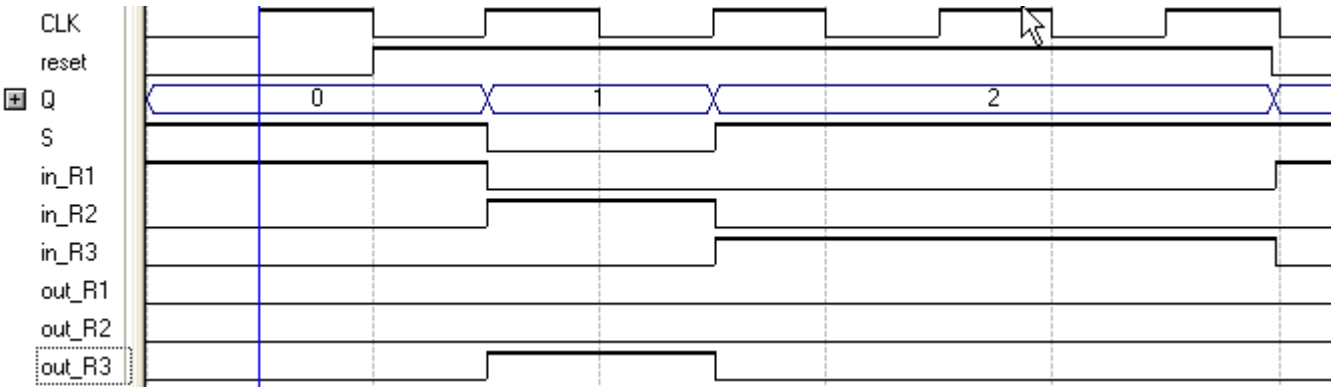
Circuito registrador 8 bits.



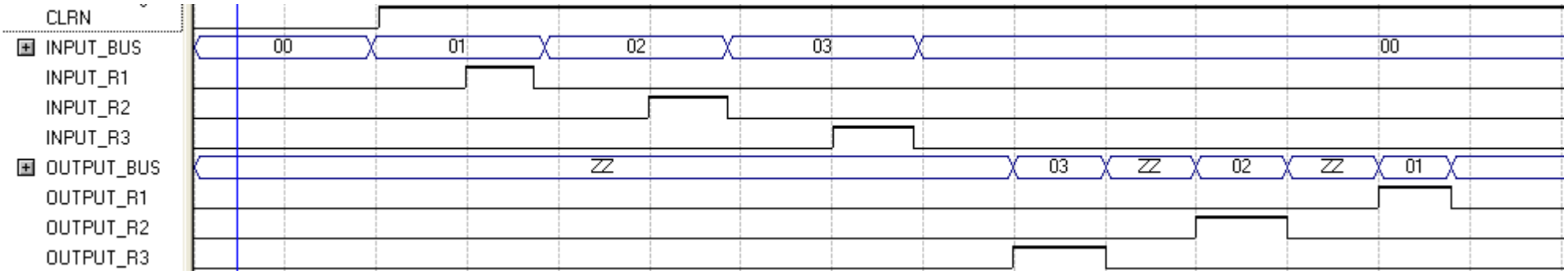
Circuito final transferência entre três registradores.



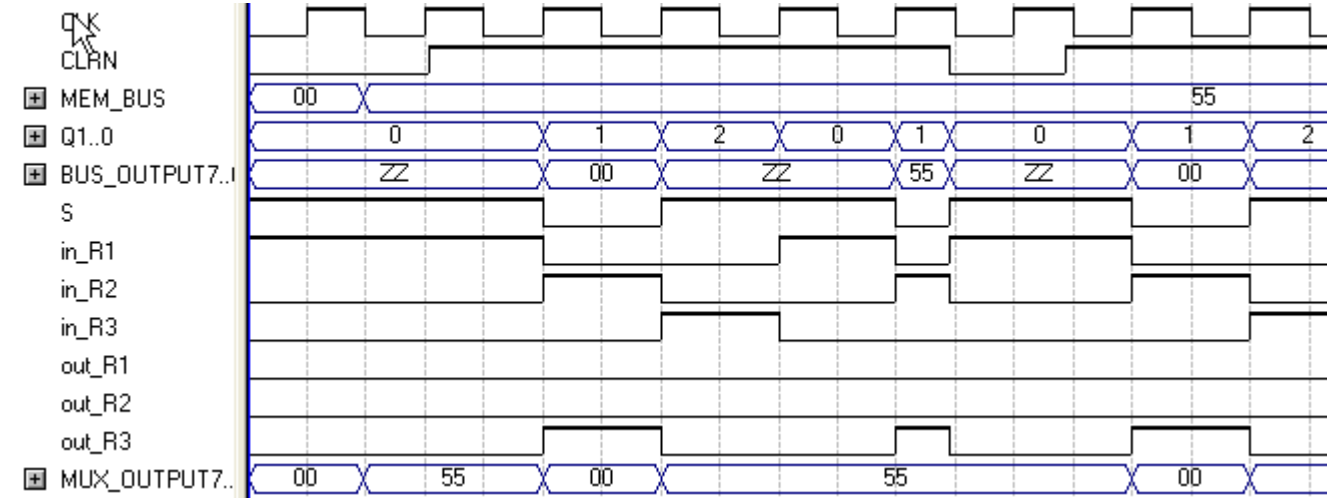
Formas de ondas da U.C.



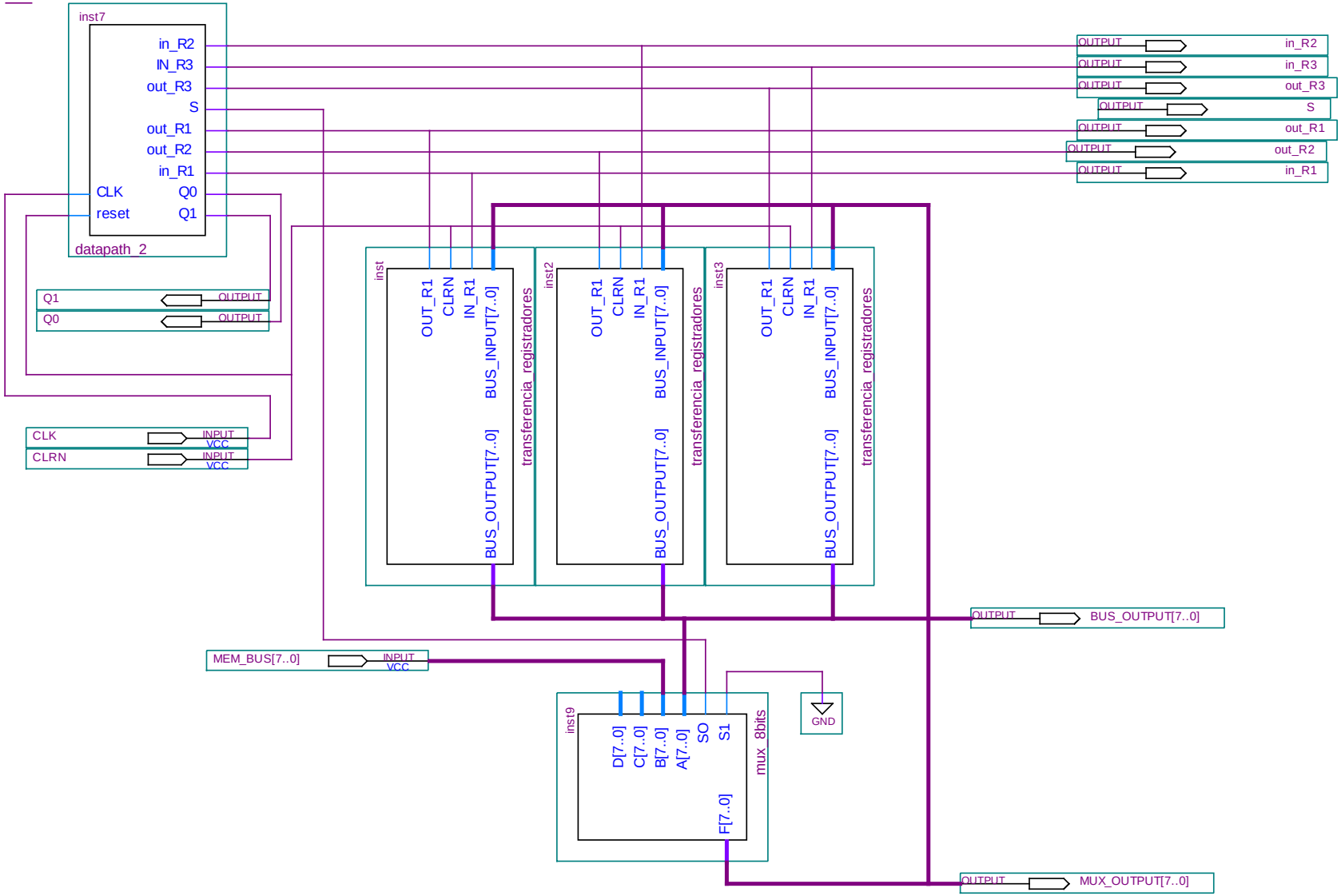
Formas de ondas do fluxo de dados.



Formas de onda do projeto do sistema digital completo.

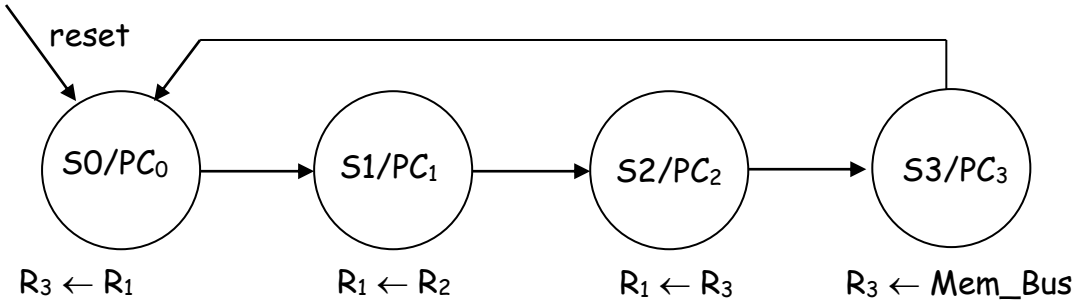


Circuito do sistema digital completo transferência entre registradores.



Exemplo: Repetir o problema anterior para:
1. $R_1 \Leftrightarrow R_2$;
2. $R_3 \leftarrow$ Mem_Bus.

b) Diagrama de estado de descrição da F.S.M. da U.C.



c) Quadro de instruções

item	instrução	S	in_R_1	in_R_2	in_R_3	out_R_1	out_R_2	out_R_3	PC's
1	$R_3 \leftarrow R_1$	0	0	0	1	1	0	0	PC_0
2	$R_1 \leftarrow R_2$	0	1	0	0	0	1	0	PC_1
3	$R_2 \leftarrow R_3$	0	0	1	0	0	0	1	PC_2
4	$R_3 \leftarrow$ Mem_Bus	1	0	0	1	0	0	0	PC_3

Implementação da F.S.M.

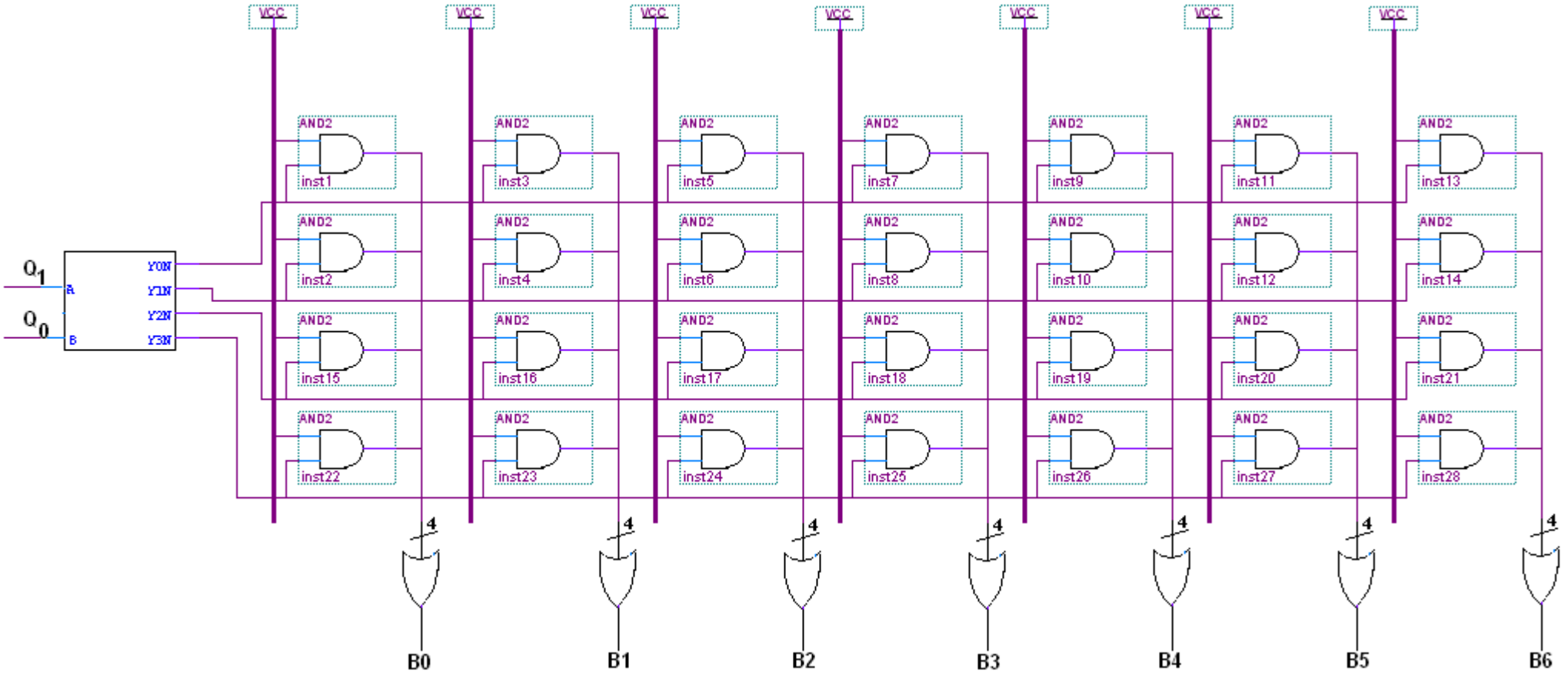
Atual	Futuro	-
Q_1Q_0	Q_1Q_0	PC's
S0 - 00	01	PC_0
S1 - 01	10	PC_1
S2 - 10	11	PC_2
S3 - 11	11	PC_3

Equações booleanas entradas/saídas.

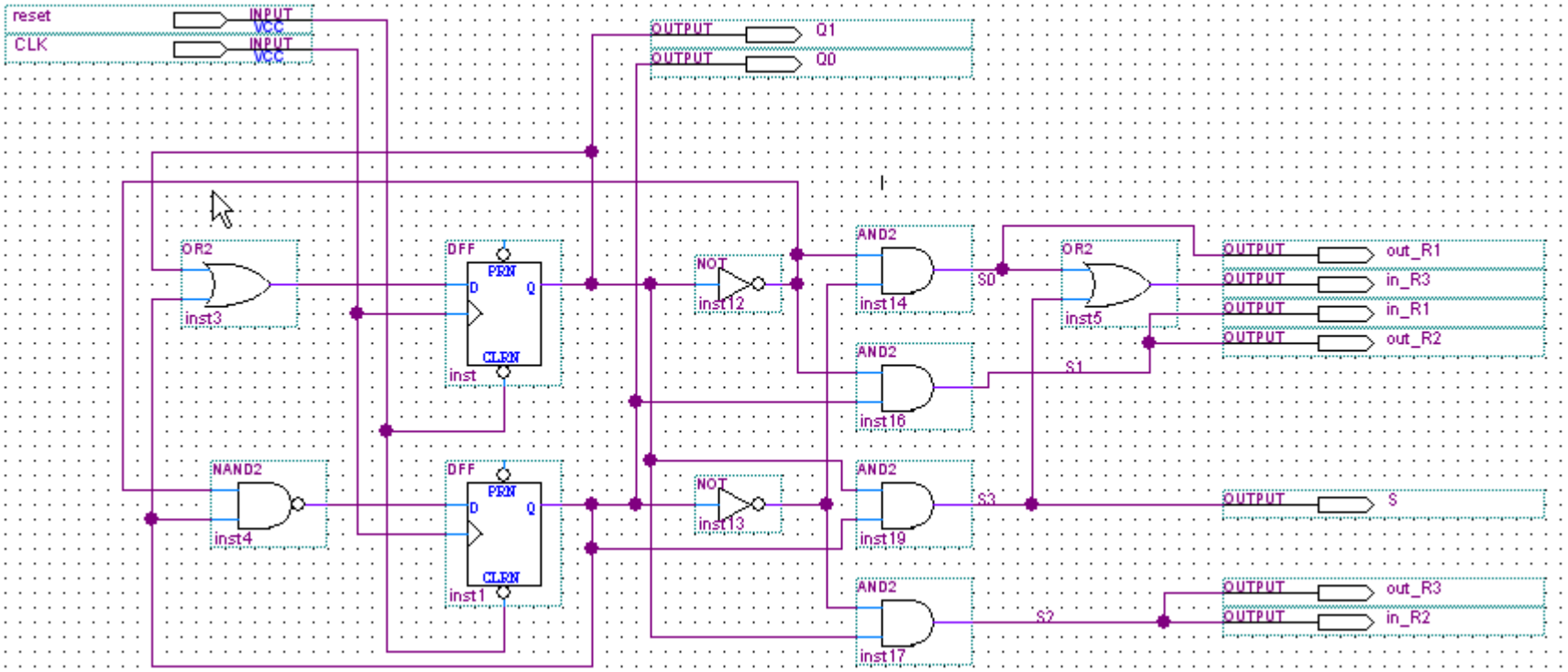
$D_1 = Q_0 + Q_1$
$D_0 = (Q_1'Q_0)'$
$PC_0 = (0C)_H$
$PC_1 = (22)_H$
$PC_2 = (11)_H$
$PC_3 = (48)_H$

S = S3;
in_R_1 = S1;
in_R_2 = S2;
in_R_3 = S0 + S3;
out_R_1 = S0;
out_R_2 = S1;
out_R_3 = S2.

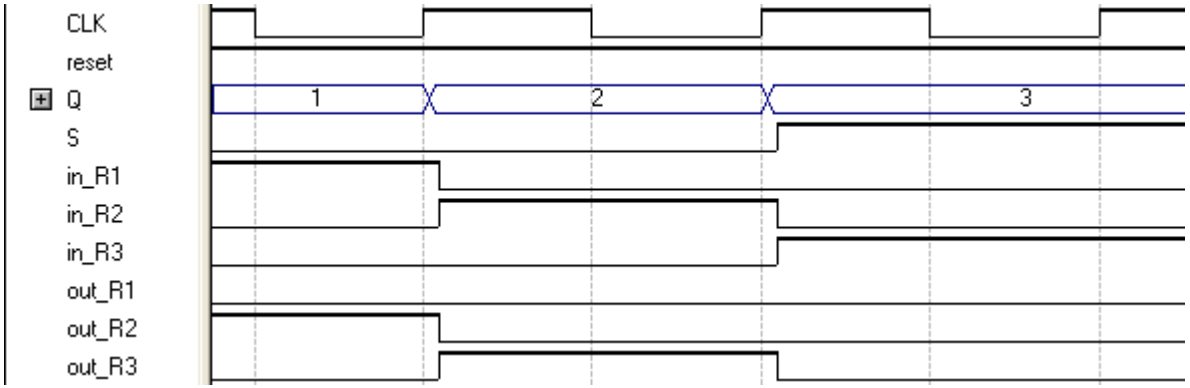
Geração das palavras de controle e programar somente as portas E onde a saída é igual a "1".



Circuito final.

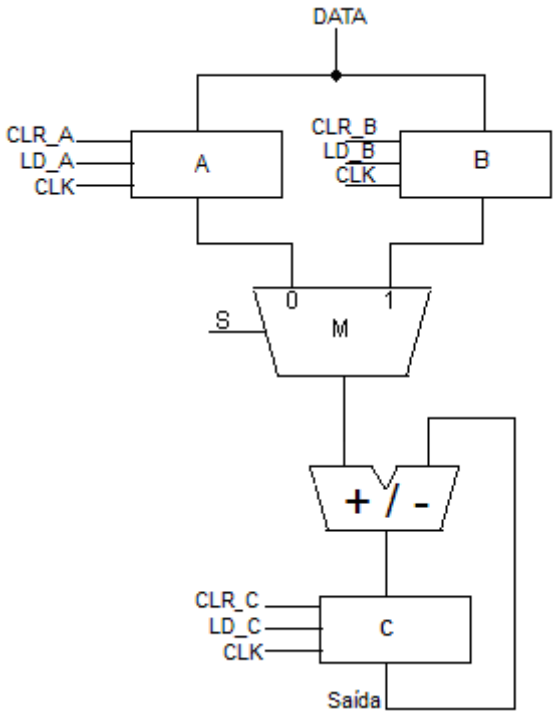


Formas de ondas



FLUXO DE DADOS – "DATAPATH"

Exemplo: Realizar a operação *C = A + B*.



Pede-se:

a) Representação esquemática do sistema digital completo.

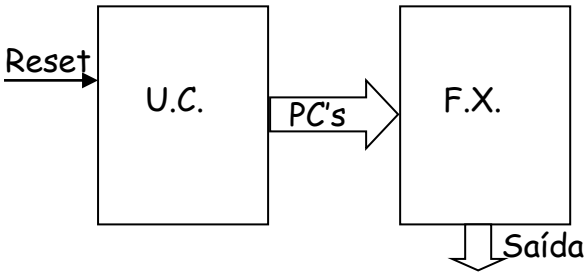
b) Projeto da unidade de controle.

c) Quadro de instruções

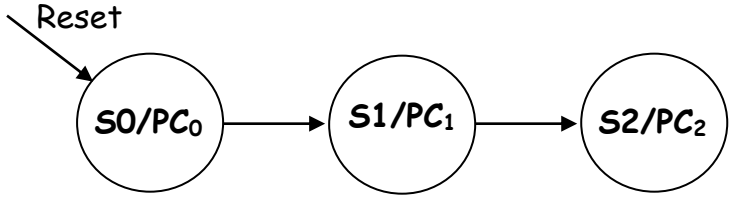
d) Circuito final

e) Formas de ondas.

a) Representação esquemática do sistema digital.



b) Projeto da U.C.



c) Quadro de instruções.

item	instrução	S	LDA	LDB	LDC	CLRA	CLRB	CLRC	PC
1	C = A	0	0	0	1	0	0	0	0
2	C = C + B	1	0	0	1	0	0	0	1
3	FIM	X	0	0	0	0	0	0	2

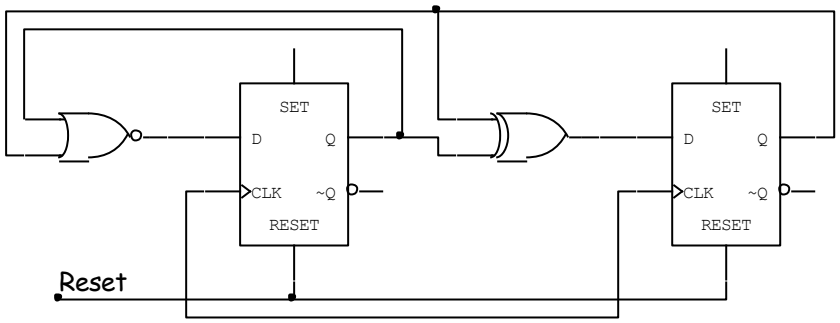
d) Tabela de estados e saída.

Q_1	Q_0	Q_1	Q_0	PC
0	0	0	1	0
0	1	1	0	1
1	0	1	0	2
1	1	-	-	-

e) Implementação da F.S.M.

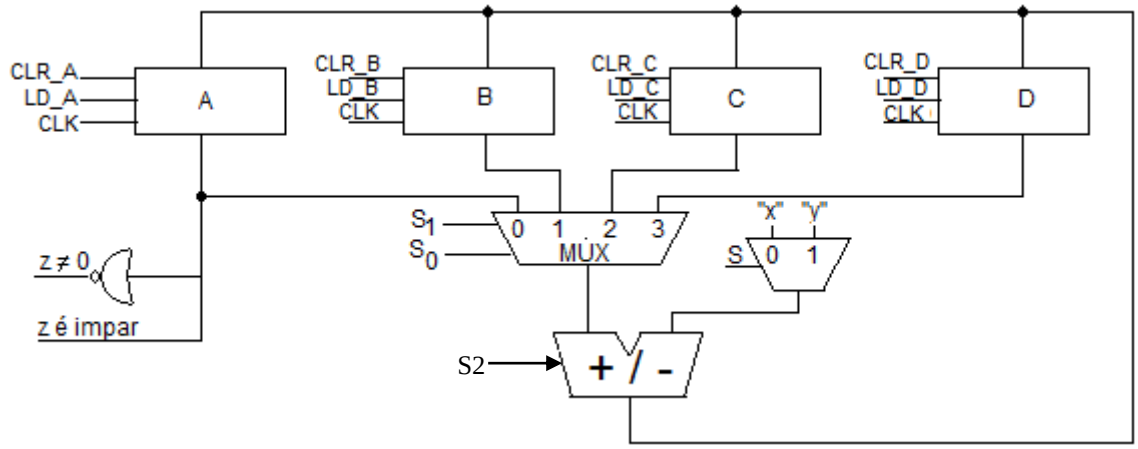
$D_0 = (Q_1 + Q_0)'$ e $D_1 = Q_1 \oplus Q_0$.

d) Circuito final



Exemplo: Montar um quadro de instruções para calcular as expressões a seguir.

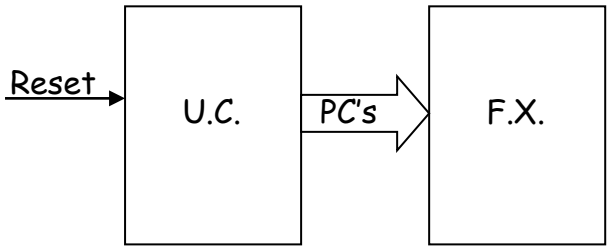
A = A - 1, B = B + 2 e C = C - 1, faça as operações enquanto A ≠ 0.
Cond.: A,B e C ≥ 0



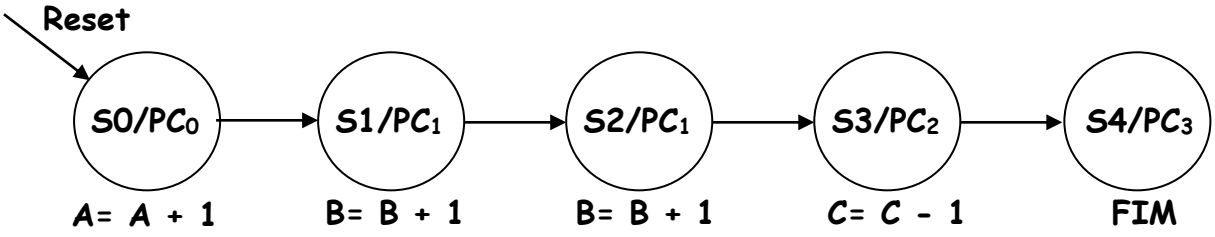
Obs.: A variável z = 1 sempre que A = 0, x = 1 e y = 0

Pede-se:
a) Representação esquemática do sistema digital completo.
b) Projeto da unidade de controle.
c) Quadro de instruções
d) Circuito final
e) Formas de ondas.

a) Representação esquemática do sistema digital.



b) Projeto da U.C.



c) Quadro de instruções.

item	instrução	LDA	LDB	LDC	LDD	CLRA	CLRB	CLRC	CLRD	S1	S0	S2	S	PC'S
1	A = A – 1	1	0	0	0	0	0	0	0	0	0	1	0	0
2	B = B + 1	0	1	0	0	0	0	0	0	0	1	0	0	1
3	C = C – 1	0	0	1	0	0	0	0	0	1	0	0	0	1
4	FIM	0	0	0	0	0	0	0	0	0	0	0	0	2

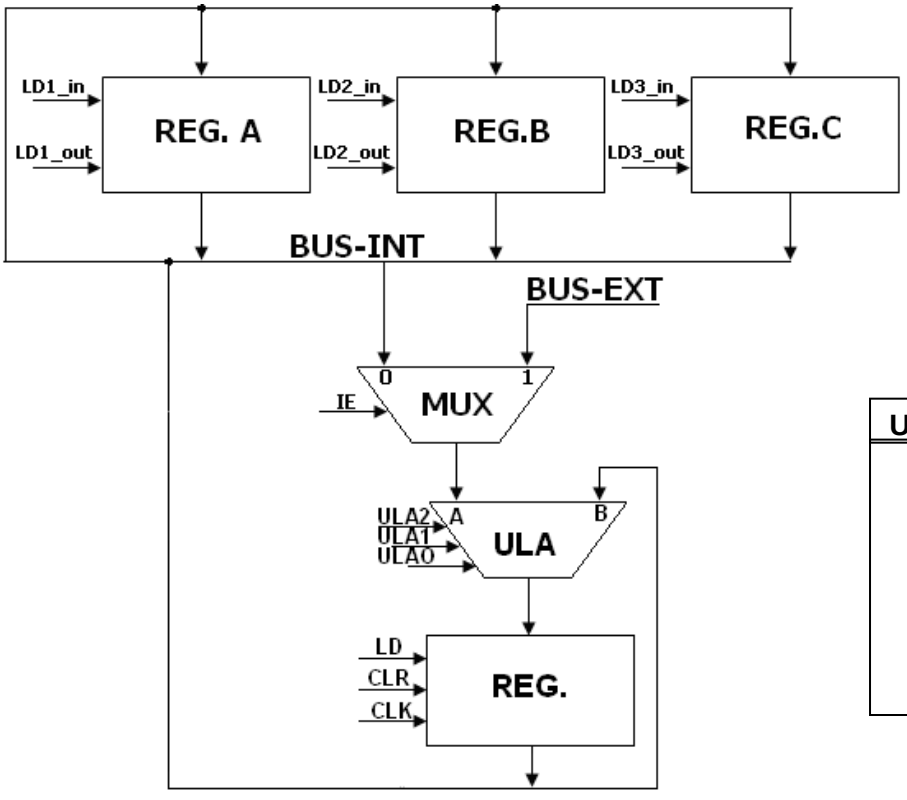
Implementação da F.S.M.

Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	PC
0	0	0	0	0	1	0
0	0	1	0	1	0	1
0	1	0	0	1	1	1
0	1	1	1	0	0	2
1	0	0	1	0	0	3

d) Circuito final

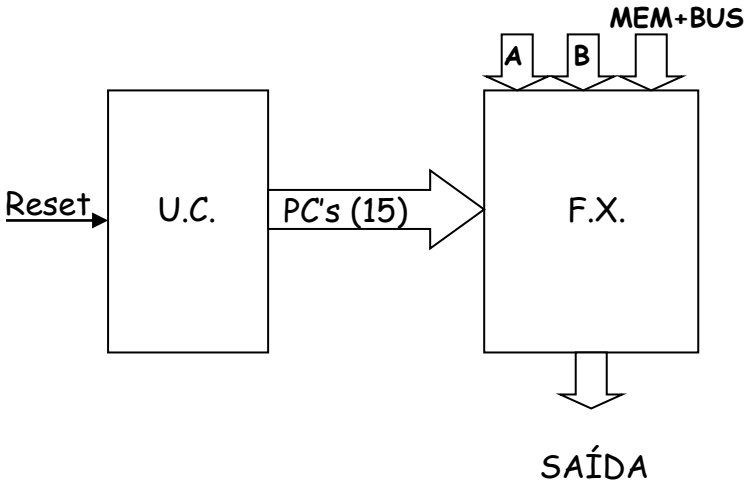
$D_2 = Q_2 \oplus Q_1$ e $D_1 = Q_1 \oplus Q_0$ e $D_0 = Q_2'Q_0'$.

Exemplo: Realizar C = A + 2*B – BUS-EXT.

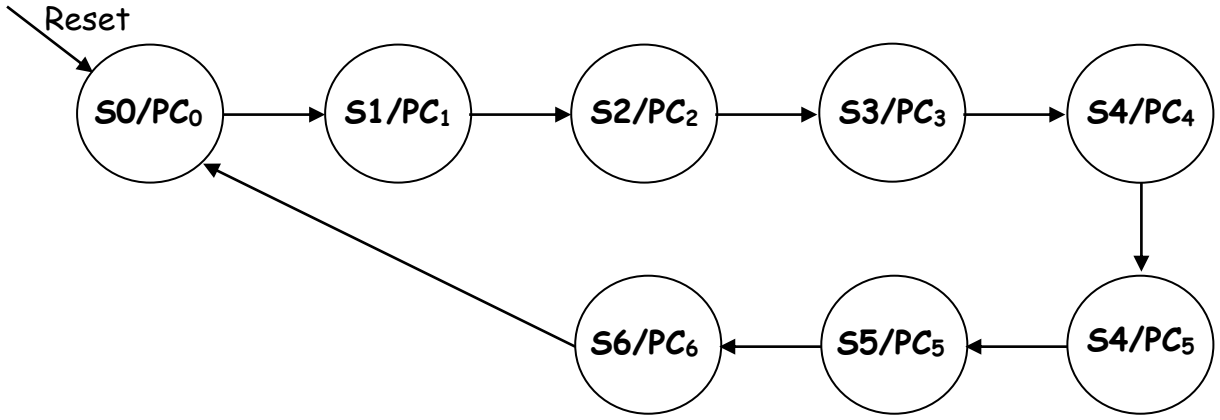


ULA_2	ULA_1	ULA_0	Operação
0	0	0	Passa A
0	0	1	A ∧ B
0	1	0	A ∨ B
0	1	1	A'
1	0	0	A + B
1	0	1	A - B
1	1	0	A + 1
1	1	1	A - 1

a) Representação esquemática do sistema digital.



b) Projeto da U.C.



c) Quadro de instruções.

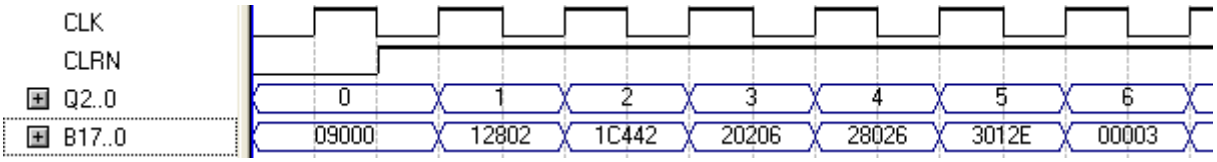
item	Instrução	$S_{1..0}$	In_R1	In_R2	In_R3	Out_R1	Out_R2	Out_R3	IE	$ULASEL_{2..0}$	LD_reg	CLRN	OE	PC's
1	LEIA A	00	1	0	0	0	0	0	0	XXX	0	0	0	0
2	LEIA B	01	0	1	0	0	0	0	0	XXX	0	1	0	1
3	LEIA C	10	0	0	1	0	0	0	0	XXX	0	1	0	2
4	Mem_bus → Reg	XX	0	0	0	0	0	0	1	000	1	1	0	3
5	Reg ← A – Reg	XX	0	0	0	1	0	0	0	101	1	1	0	4
6	Reg ← Reg + B	XX	0	0	0	0	1	0	0	100	1	1	0	5
7	Reg ← Reg + B	XX	0	0	0	0	1	0	0	100	1	1	0	5
8	Saída ← Reg	XX	0	0	0	0	0	0	0	XXX	0	1	1	6

Implementação da F.S.M.
S0 = 000, S1 = 001, S2 = 010, S3 = 011, S4 = 100, S5 = 101, S6 = 110
PC0 = 001000000XXX000; PC1 = 010100000XXX010; PC2 = 100010000XXX010; PC3 = XX0000001000110; PC4 = XX0001000101110;
PC5 = XX0000100100110; PC6 = XX0000000XXX011.

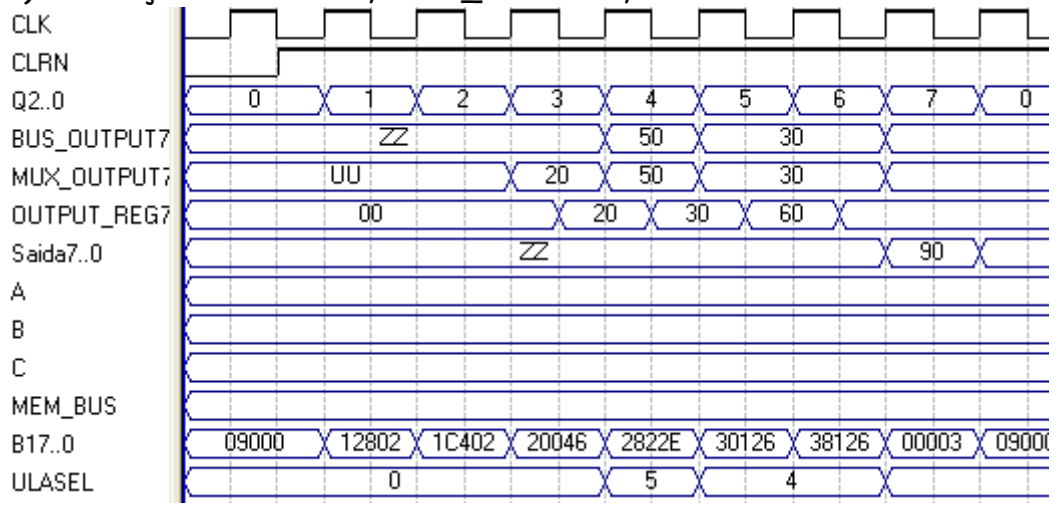
Tabela de estados completa entrada e saída com endereço e conteúdo da ROM.

A_2	A_1	A_0	B_{17}	B_{16}	B_{15}	B_{14}	B_{13}	B_{12}	B_{11}	B_{10}	B_9	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	$A_{2..0}$	$B_{17..0}$
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	S_1	S_0	In_R1	In_R2	In_R3	Out_R1	Out_R2	Out_R3	IE	ULA_2	ULA_1	ULA_0	LD	CLRN	OE	Ender.	Conteúdo
0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	X	X	X	0	0	0	**0**	**09000**
0	0	1	0	1	0	0	1	0	1	0	0	0	0	0	X	X	X	0	1	0	**1**	**12802**
0	1	0	0	1	1	1	0	0	0	1	0	0	0	0	X	X	X	0	1	0	**2**	**1C402**
0	1	1	1	0	0	X	X	0	0	0	0	0	0	1	0	0	0	1	1	0	**3**	**20046**
1	0	0	1	0	1	X	X	0	0	0	1	0	0	0	1	0	1	1	1	0	**4**	**2842E**
1	0	1	1	1	0	X	X	0	0	0	0	1	0	0	1	0	0	1	1	0	**5**	**30126**
1	1	0	1	1	1	X	X	0	0	0	0	1	0	0	1	0	0	1	1	0	**6**	**38126**
1	1	1	0	0	0	X	X	0	0	0	0	0	0	0	0	0	0	0	1	1	**7**	**00003**

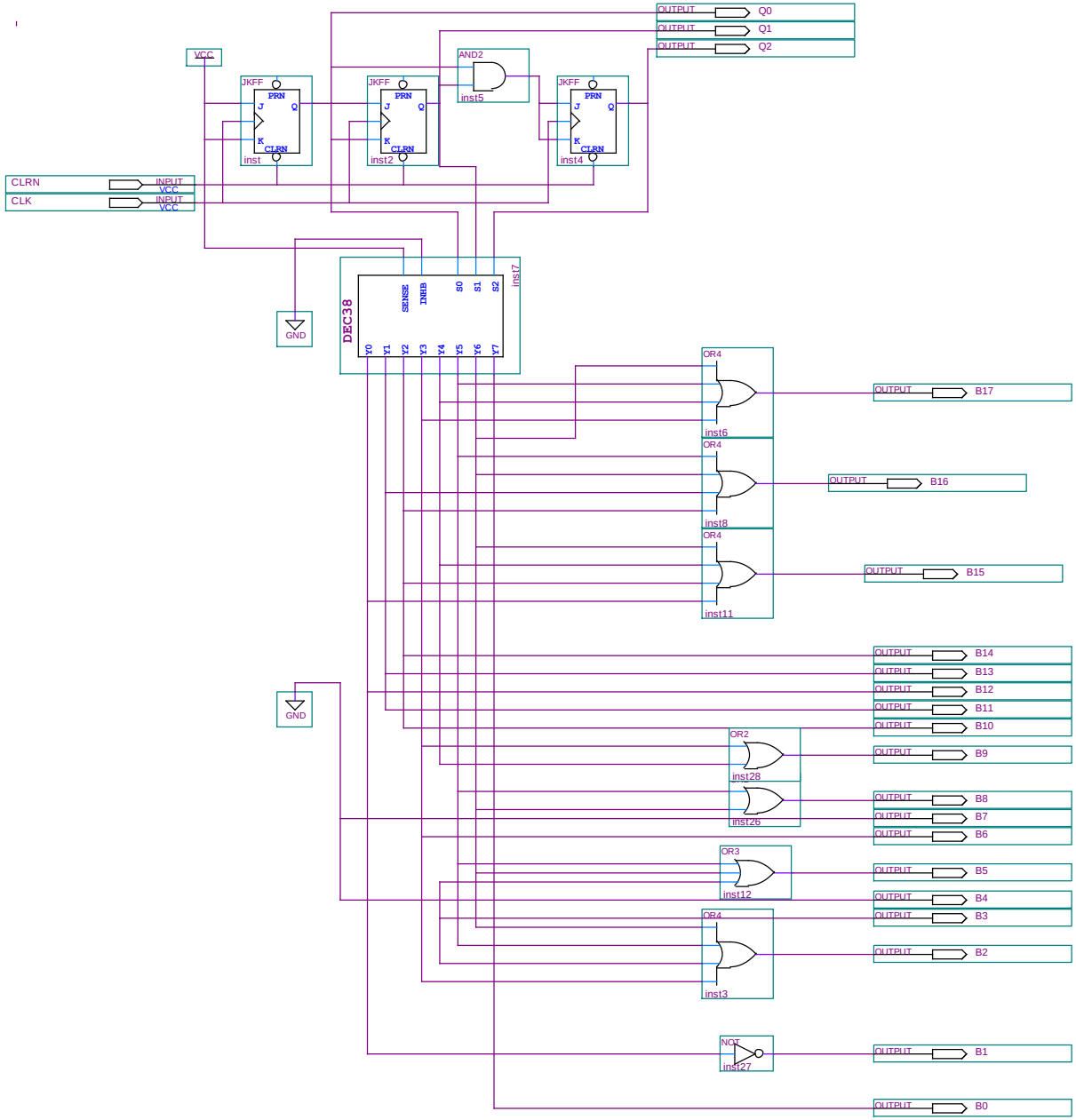
a) Formas de ondas U.C. com geração das saídas PC's.



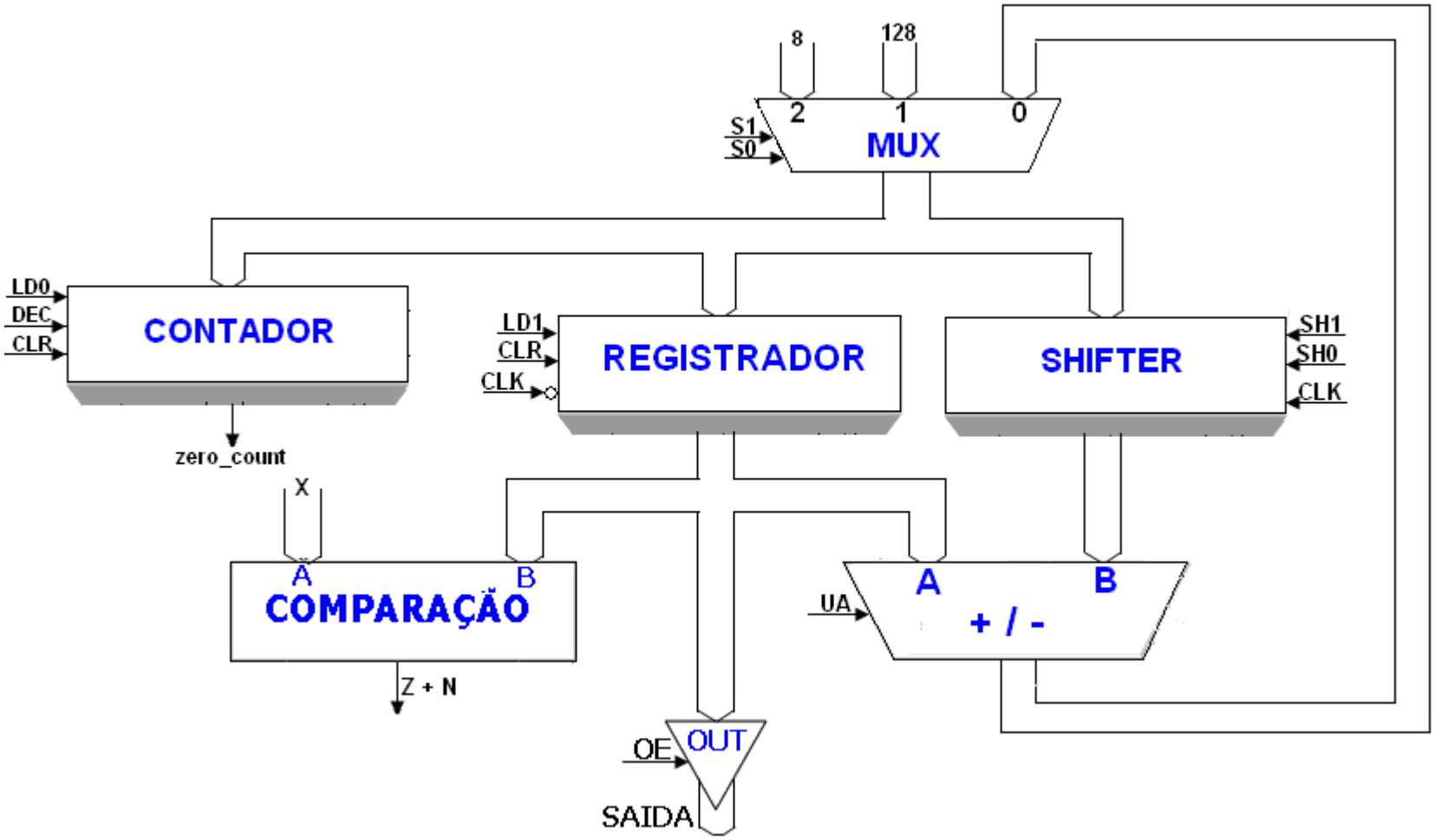
b) Simulação com *A* = 50, MEM_BUS = 20, B = 30.



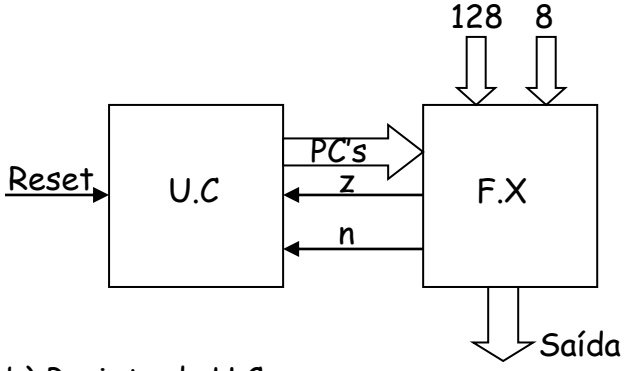
Circuito U.C. e saídas.



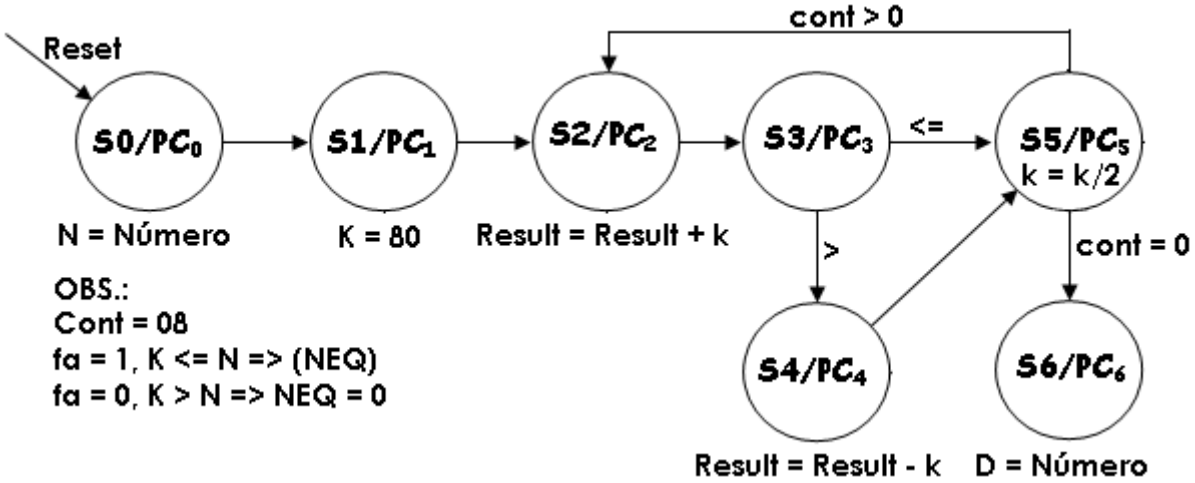
Exemplo: Determinar o valor de x, sendo $x \geq 0$, e x é um valor dado.



a) Representação esquemática do sistema digital



b) Projeto da U.C.



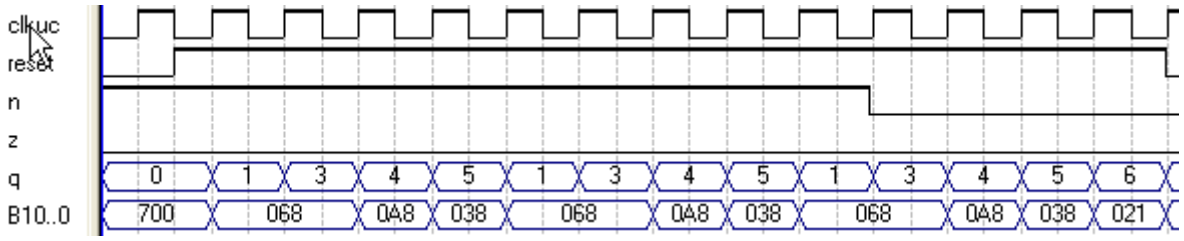
b.1) Quadro de instruções.

Item	Instrução	S_1 B_{10}	S_0 B_9	LD1 B_8	LD0 B_7	SH1 B_6	SH0 B_5	CLR B_4	$ULA_{2..0}$ $B_{3..1}$	OE B_0	PC'S	Valor
1	INICIO_NUM	1	1	0	1	0	0	0	000	0	0	680
2	INICIO_K	0	1	0	0	0	1	1	000	0	1	230
3	SOMA	0	0	1	0	0	0	1	100	0	2	118
4	COMPARA	0	0	0	0	0	0	1	000	0	3	010
5	AJUSTA	0	0	1	0	0	0	1	101	0	4	11A
6	DESLOCA	0	0	0	0	1	0	1	000	0	5	050
7	PÁRA	0	0	0	0	0	0	1	000	1	6	011

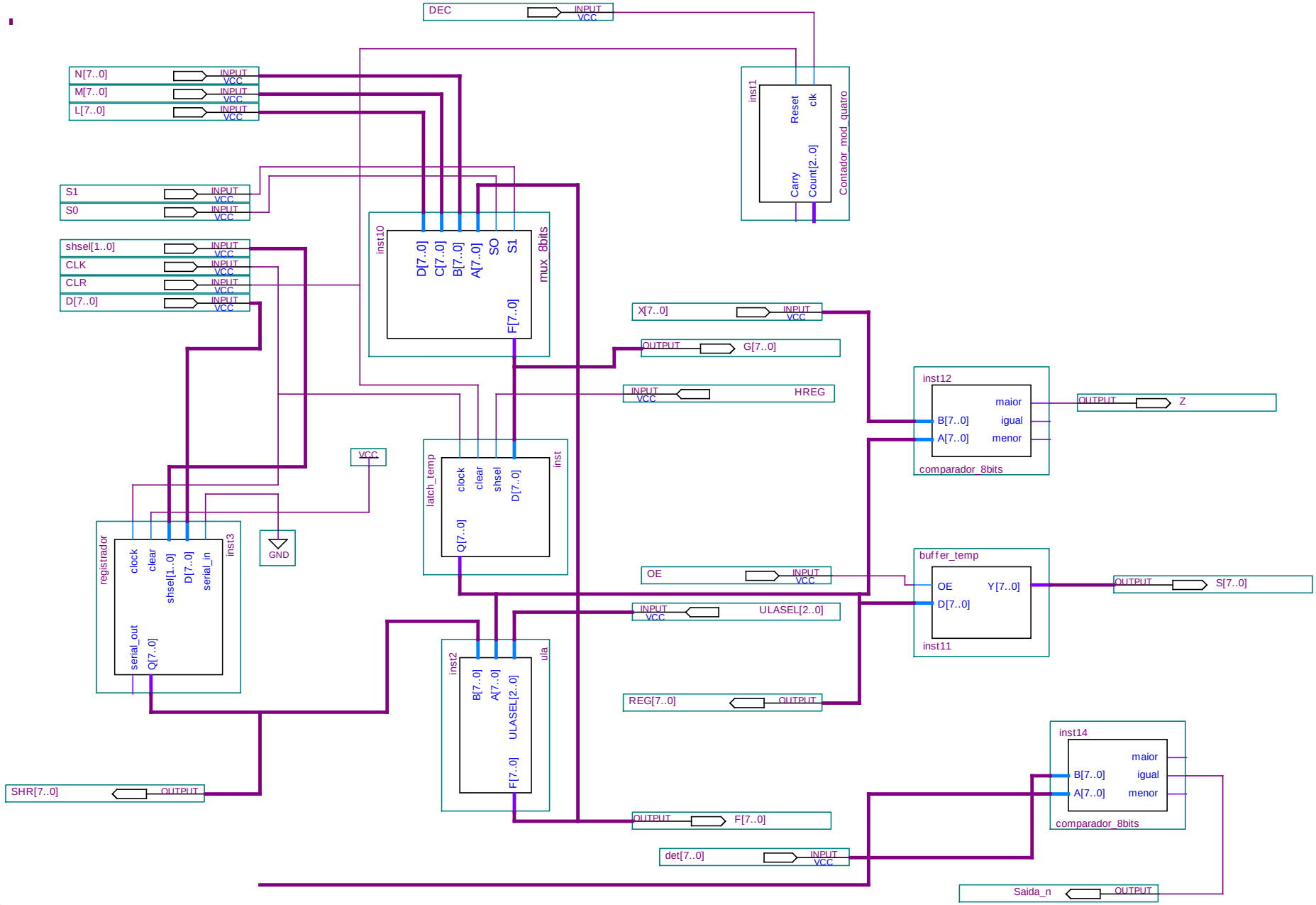
b,2) Implementação da U.C.

			Cont'fa'	Cont'fa	Contfa'	Contfa	-
Q_2	Q_1	Q_0	00	01	10	11	Saída
0	0	0	001	001	001	001	PC0
0	0	1	010	010	010	010	PC1
0	1	0	011	011	011	011	PC2
0	1	1	100	101	100	101	PC3
1	0	0	101	101	101	101	PC4
1	0	1	110	110	110	110	PC5
1	1	0	110	110	110	110	PC6
1	1	1	000	000	000	000	-

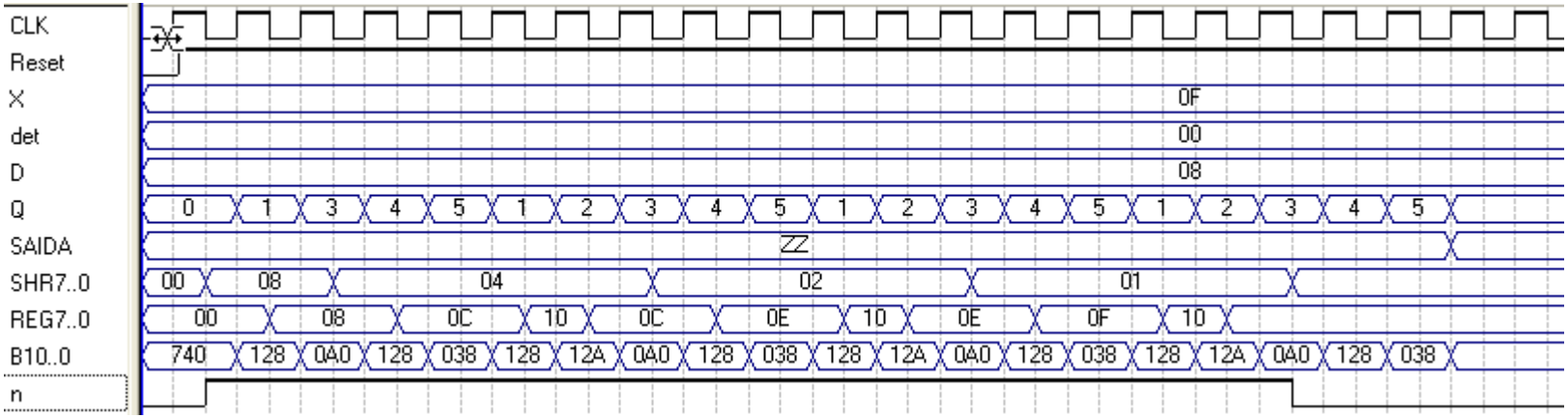
Formas de ondas da U.C. com saídas PC's

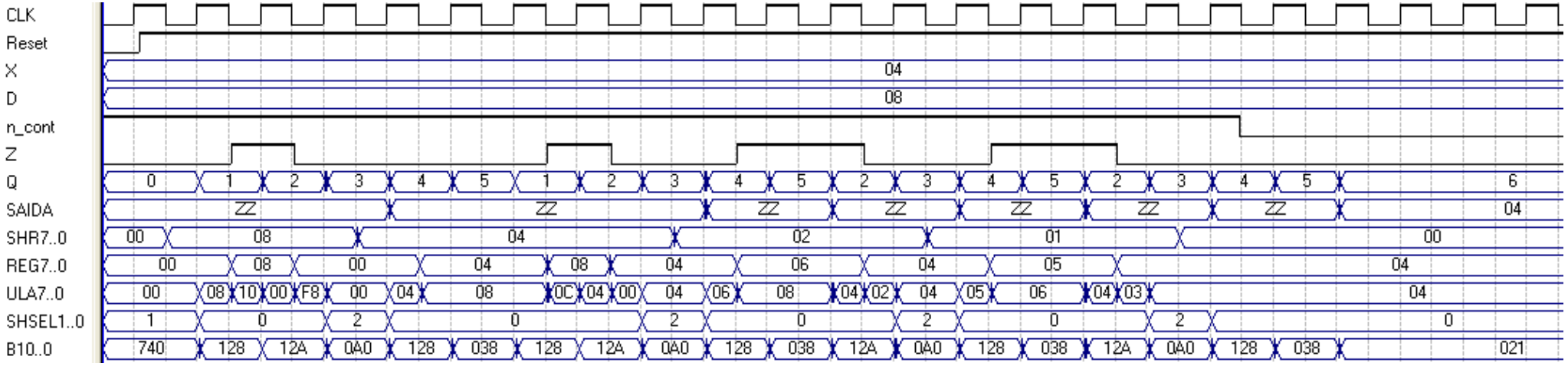


Circuito do fluxo de dados.

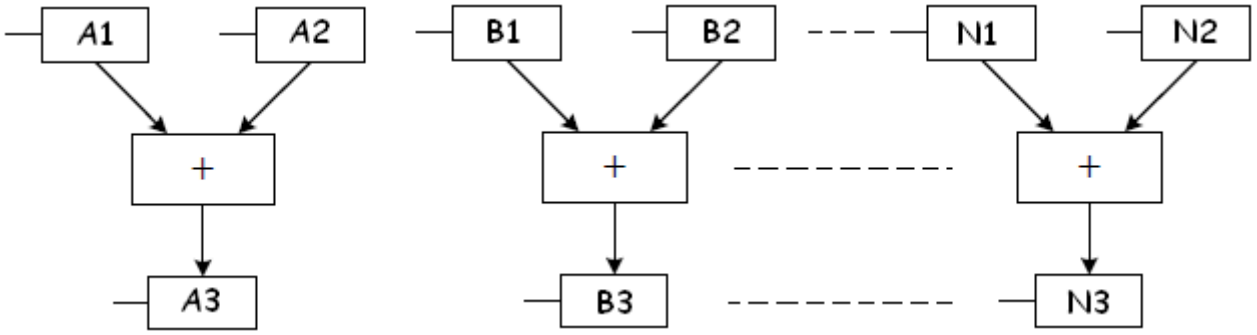

DEC
INPUT
VCC
N[7..0]
M[7..0]
L[7..0]
S1
S0
shsel[1..0]
CLK
CLR
D[7..0]
inst1
Reset
clk
Carry
Count[2..0]
Contador_mod_quatro
inst10
D[7..0]
C[7..0]
B[7..0]
A[7..0]
SO
S1
F[7..0]
mux_8bits
X[7..0]
OUTPUT
G[7..0]
INPUT
HREG
inst12
maior
igual
menor
B[7..0]
A[7..0]
comparador_8bits
OUTPUT
Z
VCC
latch_temp
clock
clear
shsel
D[7..0]
Q[7..0]
inst
GND
registrador
clock
clear
shsel[1..0]
D[7..0]
serial_in
serial_out
Q[7..0]
inst3
OE
INPUT
VCC
buffer_temp
OE
Y[7..0]
D[7..0]
inst11
OUTPUT
S[7..0]
INPUT
ULASEL[2..0]
inst2
B[7..0]
A[7..0]
ULASEL[2..0]
F[7..0]
ula
REG[7..0]
OUTPUT
inst14
maior
igual
menor
B[7..0]
A[7..0]
comparador_8bits
SHR[7..0]
OUTPUT
OUTPUT
F[7..0]
det[7..0]
INPUT
VCC
Saida_n
OUTPUT

Simulação das formas de ondas do fluxo de dados.

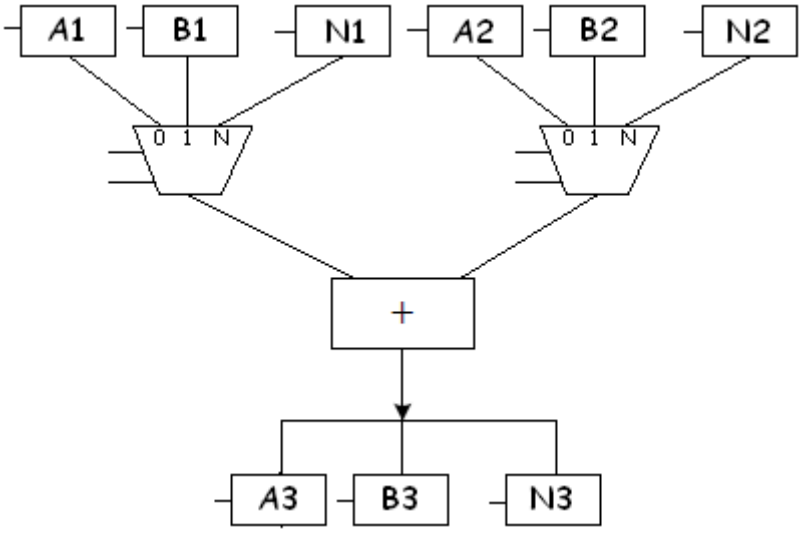




Exemplo: Somador de N números usando N somadores.



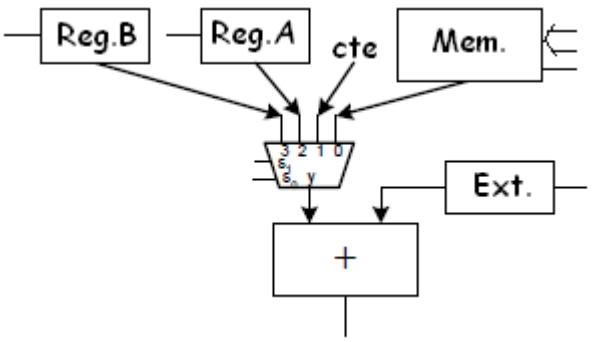
Exemplo: Somador de N números usando somente 1 somador.



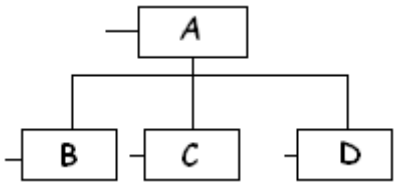
Fluxo de dados

Existem vários métodos para transferências entre unidades funcionais e registradores. A seguir apresentamos os métodos.

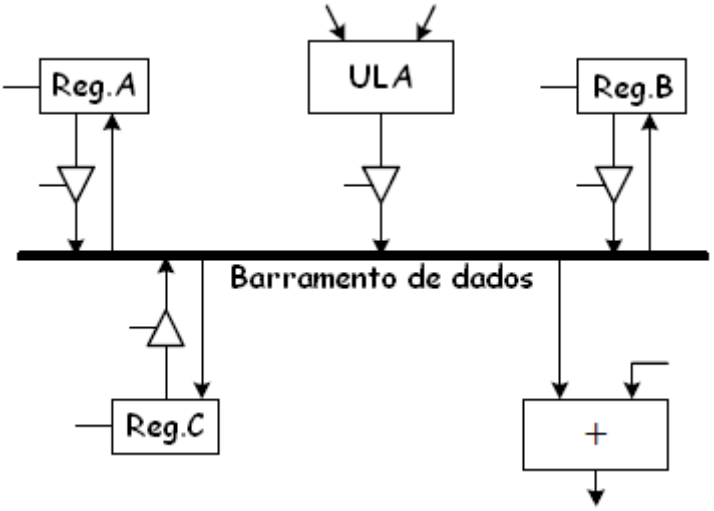
1. Múltiplas fontes – São utilizadas unidades funcionais como multiplexadores quando o número de fontes de origem é muito. O exemplo a seguir mostra múltiplas fontes e um destino.



2. Múltiplos destinos – São utilizadas muitas unidades de destino selecionadas por uma unidade de controle. O exemplo a seguir mostra múltiplos destinos de uma fonte de origem.

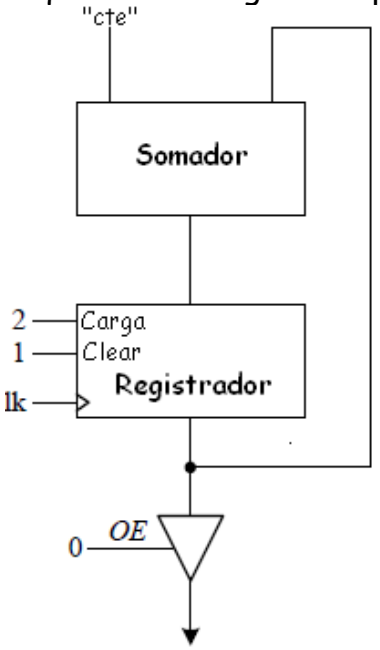


3 - Terceiro estado – O fluxo de dados é um caminho de dados para muitas fontes e muitos destinos e uma forma de conexão é através de um meio comum conhecido como "data-bus" ou barramento de dados. Para que não haja um conflito entre saídas de cada uma das fontes de dados, um circuito buffer com saída terceiro estado é colocado à saída de cada uma das fontes e interligados ao barramento de dados.



4 - Dedicados fluxo de dados

Quando a aplicação requer específicas unidades funcionais para a realização de uma tarefa única, por exemplo, uma multiplicação, um cálculo fatorial, um cálculo de um seno, então um fluxo de dados dedicados é montado com uma arquitetura dedicada. Por exemplo, a arquitetura a seguir é capaz de realizar a operação de soma entre as fontes de origens.



5 - **Métodos utilizados na escolha da arquitetura do fluxo de dados**

Partindo da premissa de projeto onde inicialmente o fluxo de dados é projetado para resolver certo problema existem dois métodos que poderão ser utilizados na escolha da arquitetura mais adequada para o fluxo de dados. São eles:

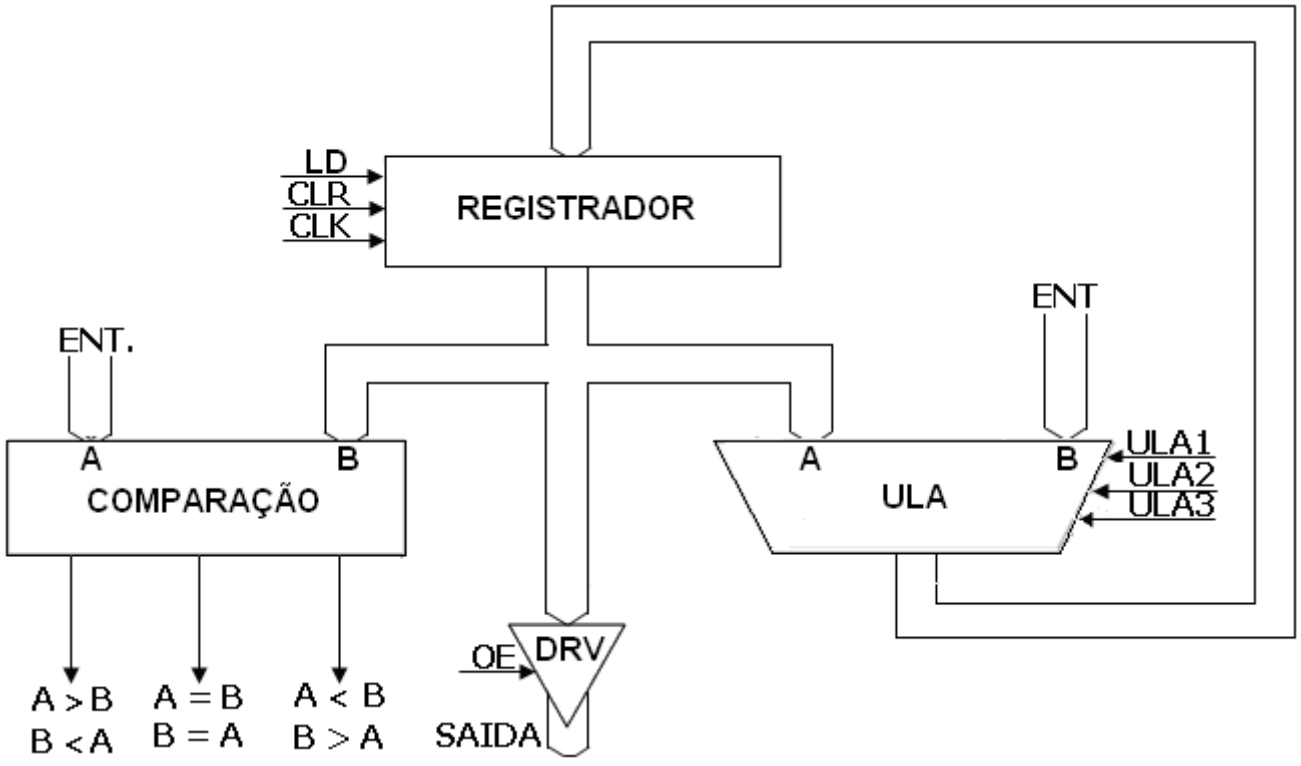
- Fluxo de dados geral;
- Fluxo de dados customizado ou dedicado.

6 - Fluxo de Dados Geral

A arquitetura de fluxo de dados geral, como apresentada a seguir, tem a capacidade de resolver vários algoritmos simples, onde o número de variáveis que se manipula internamente é limitado a uma única variável. Esta arquitetura se presta a solução de problemas cujos operandos de entrada realizam com os operandos internos operações simples na ULA e o resultado é apresentado na saída. Uma vez que o fluxo de dados contém apenas um único registrador de armazenagem temporária, os operandos vêem da entrada externa do fluxo de dados e do operando acumulado no fluxo de dados e o resultado da operação é armazenado temporariamente no registrador.

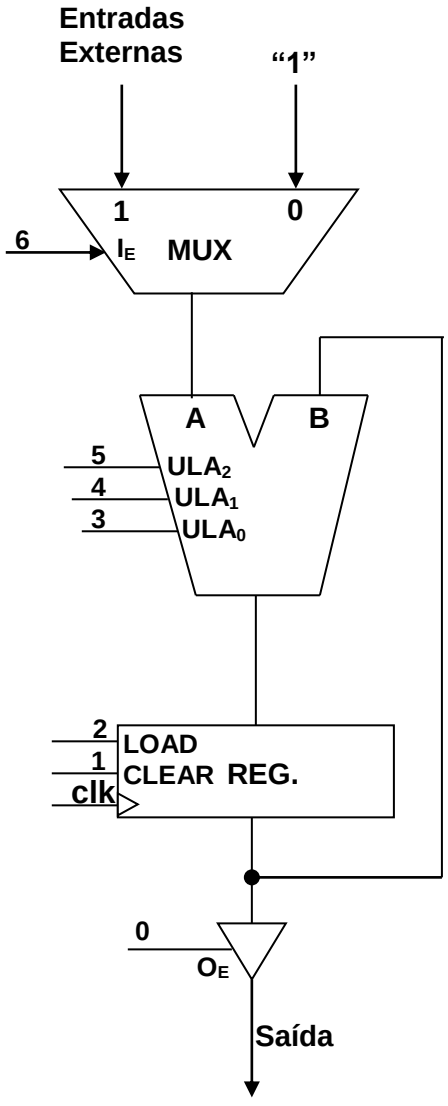
Para a implementação de diversos algoritmos o fluxo de dados geral contém: uma unidade funcional de operações lógicas e aritméticas a ULA e um registrador para armazenagem temporária dos dados.

7 - FLUXO DE DADOS GERAL



A arquitetura a seguir é utilizada quando se têm operações entre um mais do que uma entrada de dados, embora não possa armazenar mais do que um conteúdo.

A entrada na ULA do operando A pode ser feita através de dois caminhos sendo o primeiro pela entrada externa e o segundo por uma entrada constante igual "1", selecionados no multiplexador através da linha de seleção I_E, como apresentada na figura a seguir. O operando B da ULA será sempre oriundo do conteúdo do registrador de armazenagem temporária. A operação da ULA é determinada por três linhas de controle ULA_2, ULA_1, ULA_0, como definida na tabela funcional apresentada a seguir. O registrador possui uma entrada para a carga de dados paralela e recebe os dados da saída da ULA a fim de armazenar. O registrador pode ser resetado para zero através de um sinal de controle **CLEAR** vindo da unidade de controle. O conteúdo do registrador temporário pode ser transferido diretamente com passagem direta para a saída externa, somente habilitando-se a linha de controle O_E do buffer tri-estado. Assume-se que os canais de comunicação ou barramentos para a transferência de dados são de largura igual a oito bits. Todas as linhas de controle são apenas de um bit. A seguir apresentamos a arquitetura do fluxo de dados geral e um exemplo de implementação.



ULA_2	ULA_1	ULA_0	Operação
0	0	0	Passa A
0	0	1	A $\wedge$ B
0	1	0	A $\vee$ B
0	1	1	A'
1	0	0	A + B
1	0	1	A - B
1	1	0	A + 1
1	1	1	A - 1

$\wedge \rightarrow$ Operação A AND B.
$\vee \rightarrow$ Operação A OR B.

A arquitetura do fluxo de dados, apresenta sete linhas de controle (0 – 6) para as operações neste fluxo de dados simples. Várias operações podem ser realizadas neste fluxo

de dados aplicando sinais **SET/RESET** nos sinais de controle em tempos distintos. Estas linhas de controle agrupadas formam uma **PALAVRA DE CONTROLE**. Uma operação do fluxo de dados, entretanto é determinada pelos valores definidos na palavra de controle e serão válidas em um ciclo de relógio. Pela combinação de múltiplas palavras de controle e numa certa seqüência, o fluxo de dados realizará as operações especificadas em uma dada ordem cumprindo a seqüência do algorítmo. O resultado é uma máquina dedicada que resolve aquele problema.

Por exemplo, se uma operação a ser realizada é para carregar um valor de uma entrada externa para o registrador temporário, a palavra de controle é definida como a seguir.

Linha Controle Número Entrada	I_E 6	$ULA_2\ ULA_1\ ULA_0$ 5 - 3	LOAD 2	CLEAR 1	O_E 0
Valor setado	1	000 (Passa)	1	0	0

Como a variável de seleção de entrada, $I_E = 1$, a entrada selecionada no multiplexador é a entrada externa ao MUX. Da tabela, as linhas de seleção de operações da ULA, ULA_{2-0} devem ser setadas em **000** selecionando a operação de passagem direta dos dados. Finalmente a variável LOAD = 1, carrega o valor da saída da ULA para dentro do registrador temporário. Se o dado armazenado não é para ser apresentado na saída o sinal de controle O_E deve ser ressetado fazendo $O_E = 0$.

Note que a carga da variável no registrador deve ser síncrona com a borda de subida do sinal de relógio (clk), enquanto que as demais variáveis são assíncronas.

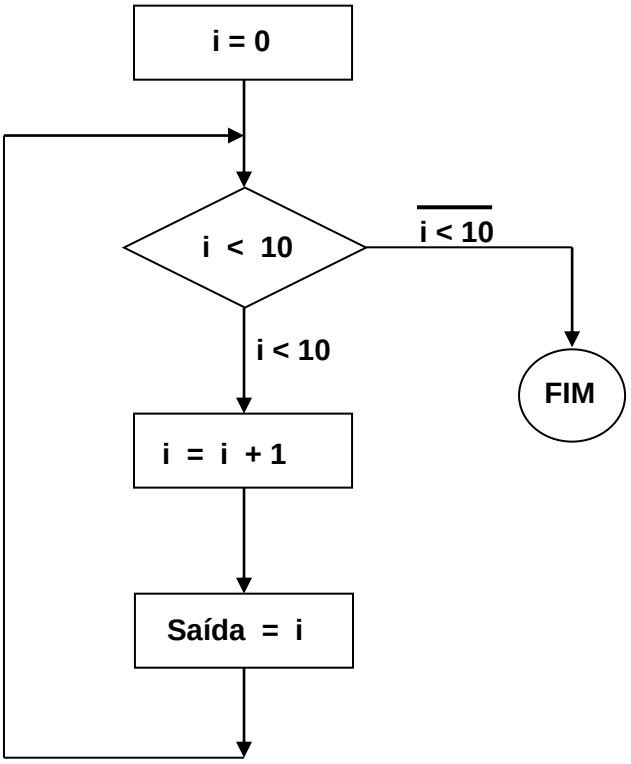
Como o próximo evento é a apresentação na saída do dado armazenado no registrador, a seqüência será uma operação de leitura do registrador e concomitante a habilitação do buffer de saída. O evento só acontecerá na próxima borda do relógio, assim a carga no registrador ocorre em um ciclo de relógio e a leitura na saída será no próximo ciclo de relógio e da habilitação do sinal $O_E = 1$.

Exemplo 1: Utilizando-se do fluxo de dados geral determinar as palavras de controle para gerar e apresentar na saída números de 1 a 10.

PASSO 1: Descrição do problema.
O algoritmo para realizar esta operação é mostrado a seguir.

```
1    i= 0
2    do while (i< 10)
3    {i= i +1
4    output i}
```

O fluxograma a seguir descreve o algoritmo de geração e apresentação de números e 1 a 10 no fluxo de dados.



PASSO 2: Definição do fluxo de dados.

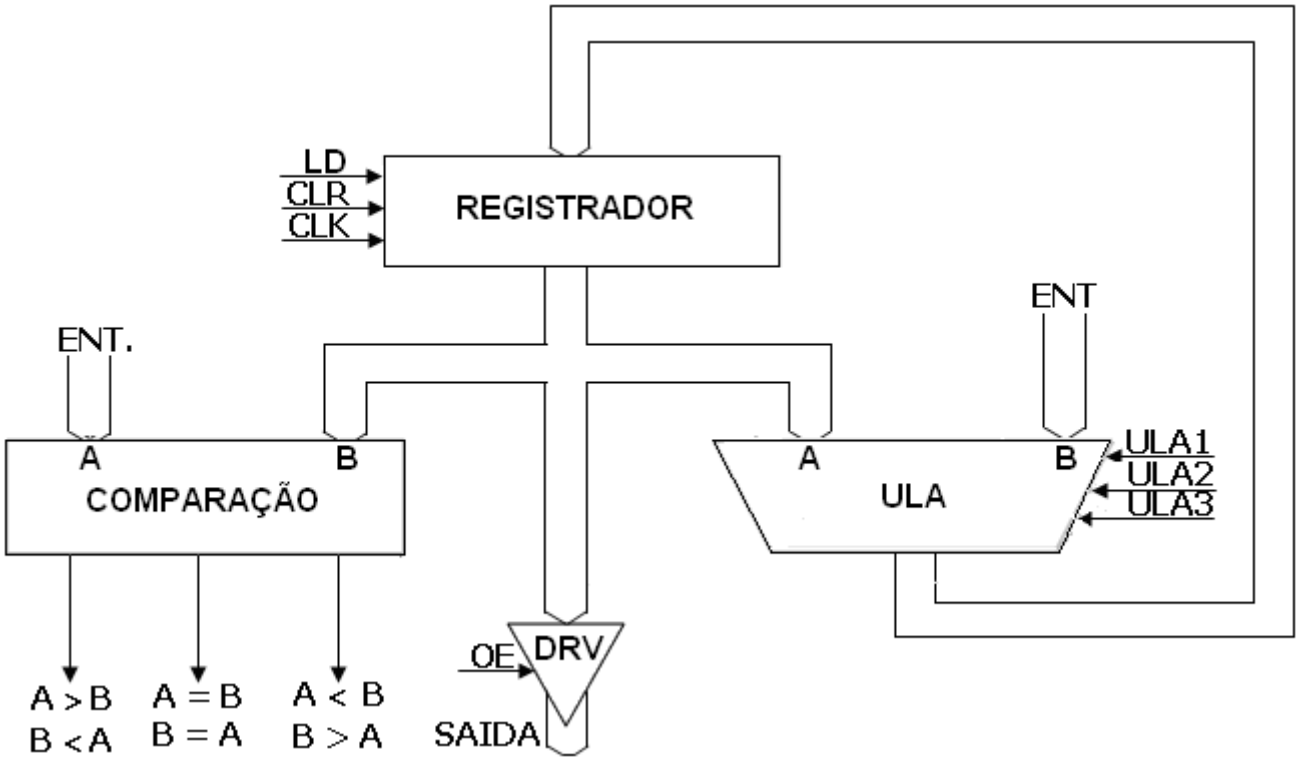
Para traduzir o algoritmo em palavras de controle, as operações realizadas no fluxo de dados e operações realizadas pela unidade de com trole, necessita-se observar no algoritmo:

1) As operações nos retângulos, tais como, as operações de leitura, escrita lógicas ou aritméticas com os dados, são elas as linhas 1, 3 e 4 e realizadas no fluxo de dados;
2) As operações de controle, tais como comparações, por exemplo do valor de i na linha 2 realizado na unidade de controle.

Esta condição é apresentada no fluxo de dados através do sinal de status (sinal de status = 0 quando a condição é falsa e igual a "1" quando verdadeira). Esta condição gerada será enviada para a unidade de controle.

Dependendo da condição deste sinal de status, a unidade de controle decidirá se realizará ou não o loop novamente.

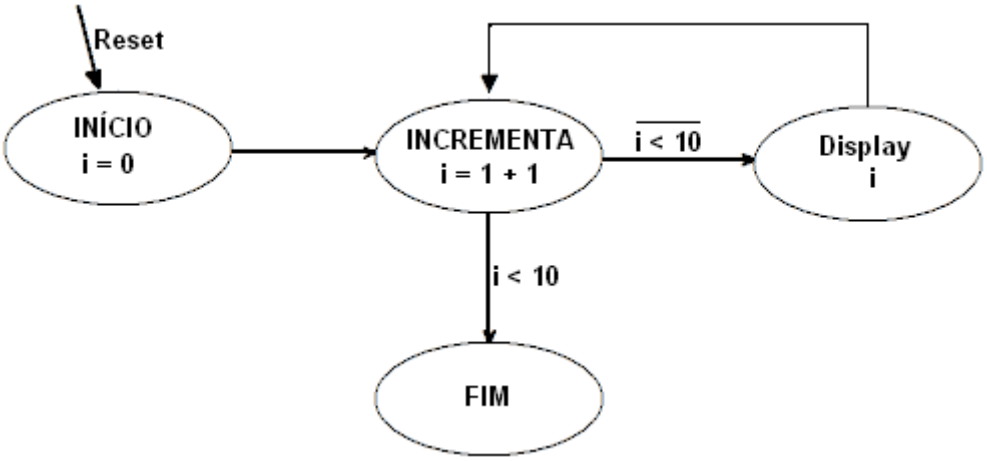
A arquitetura apresentada a seguir é adaptada para a realização do algoritmo descrito.



Na arquitetura acima, o fluxo de dados possui duas entradas de dados, sendo uma para a unidade de comparação e a segunda para a ULA. O funcional da ULA.

PASSO 3: Projeto da unidade de controle. Descrição do algoritmo através do diagrama de estados pelo modelo de Moore.

A designação de três estados sendo S0 = Estado inicial, S1 = Estado Incremento, S2 = Estado display e S3 = FIM.



8 - Implementação da F.S.M.

A designação de estados por codificação de estados é: S0 = 00, S1 = 01, S2 = 10, S3 = 11.

Tabela de estados

Q_1	Q_0	I < 10 = 0	i < 10 = 1	Saída
0	0	01	01	PC_0
0	1	10	11	PC_1
1	0	01	01	PC_2
1	1	11	11	PC_3

PASSO 4: Projeto nível RTL. Desenvolvimento do quadro de instruções e definição do sinais de controle gerados pela U.C. e que comandam o fluxo de dados. As palavras de controle PC's são:

As palavras de controle para as três instruções são apresentadas a seguir:

Palavra Controle	Instrução	I_E 6	ULA_2 ULA_1 ULA_0 5 – 3	LOAD 2	CLEAR 1	O_E 0
1	i= 0	X	X X X	0	1	0
2	i= i+1	0	100 (adr)	1	0	0
3	Saída i	X	X X X	0	0	1

As operações 1, 2 e 3 são realizadas em 3 ciclos sucessivos de relógio. A evolução para a próxima instrução será incondicional e inicia com a palavra de controle 1 que inicializa i em 0 (i = 0).

O valor de i é armazenado no registrador do fluxo de dados. Uma vez que o registrador possui a função **CLEAR**, esta pode ser utilizada para zerar o registrador. A ULA não é necessária nesta operação e assim é irrelevante a função selecionada para a ULA realizar. Desta forma as linhas de controle ULA_2, ULA_1, ULA_0 e I_E, poderão ser irrelevantes (don't care). A linha de controle do registrador **LOAD** também será zero porque não é para armazenar o dado de saída da ULA no registrador. Como o algoritmo manda apresentar números de 1 a 10, então para i = 0 não deve ter saída e o valor de saída do registrador não passa para a saída pois a linha de controle será selecionada fazendo O_E = 0.

A palavra de controle 2, incrementa o valor de **i** e é necessário adicionar o valor "1" ao operando que está armazenado no registrador. A ULA tem esta função de incremento somente com o operando A e não com o operando B. Pela arquitetura do fluxo de dados, vemos que o registrador realimenta o dado a ser incrementado no operando B e será com B a realização desta operação de incremento. Para a realização desta operação de incremento diretamente em B com a ULA, deverá haver uma modificação ou na ULA, criando a função incremento em B ou mesmo modificando o fluxo de dados para o operando no registrador ser roteado para o operando A da ULA. Tudo isso requer modificações no fluxo geral de dados o que neste momento não é possível.

A operação escolhida foi entrar com o valor constante "1" do MUX como o operando A pelo pela variável I_E= 0 . A ULA realiza esta operação setando as linhas de controle em **100** e somará o operando A com o operando B e o resultado da operação será armazenada no

registrador pelo sinal de controle **LOAD** do registrador, e concomitantemente o resultado será transportado para o operando B.

A palavra de controle 3 seta a linha de controle O_E e o valor incrementado é apresentado para ser lido. Novamente serão irrelevantes os valores das linhas de controle da ULA porque não existe um valor novo para carregar no registrador. Como o objetivo é apresentar o valor do registrador na saída, um $O_E = 1$ já apresenta o valor atual.

As palavras 2 e 3 devem ser executadas 10 vezes, a fim de apresentar os números de 1 a 10. A instrução "**DO WHILE"** o loop de contagem de 1 a 10 é implementado na unidade de controle. Conforme a seqüência de operações mostradas pelo algoritmo, apresentaremos a implementação da Unidade de Controle para a realização do algoritmo com o fluxo de dados.

9 - Implementação da unidade de controle

A escolha de um mecanismo de descrição deste problema seqüencial foi por uma máquina de estados. Existem alguns mecanismos de descrição de sistemas seqüenciais como : a tabela de fluxo, algoritmo máquina de estados ASM, redes de Petri e foi escolhido o diagrama de estados como ferramenta pois se trata de um problema tipicamente seqüencial e o diagrama de estados é de mais fácil implementação por máquina de estados finitos – FSM, pois cada nó do diagrama é um estado do sistema.

10 - Implementação por diagrama de estados

O diagrama de estados a seguir utiliza um estado inicial S0, que é o estado de partida e uma chave de entrada Start, que inicializa a máquina. O segundo estado S1 é responsável pela primeira palavra de controle para o fluxo de dados, zerando o registrador e fazendo i = 0. A terceira palavra de controle é executada em S2 incrementando o valor de i (i=i+1). O próximo estado é de apresentação do número incrementado na saída. A Unidade de controle sempre testa neste estado S3 se a contagem executou os 10 números apresentados. Caso negativo o processo de incremento continua indo para o estado S2 e caso positivo o processo retorna para o esta S0.

Observando o diagrama de estados vemos então que existem quatro estados S0,S1,S2 e S3, e portanto são necessários 02 bits Q_1 e Q_0 e duas variáveis de entrada Start, i < 10, o que possibilita quatro combinações possíveis.

A seguir montamos a tabela de estados presentes, futuros e saídas para definir a evolução do diagrama de estados. A saída na tabela abaixo são as palavras de controles 1, 2 e 3 e serão mostradas quando da implementação por máquina de estados.

11 - Diagrama de Estados

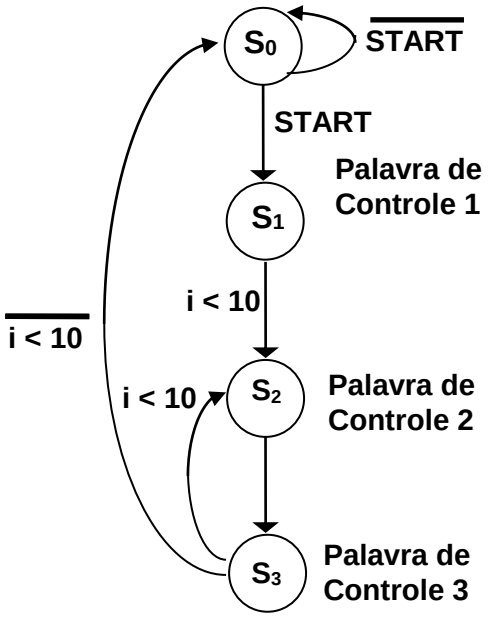


TABELA DE ESTADOS

Atual	Entradas Start, i<10				Saídas
$Q_1\,Q_0$	00	01	10	11	Palavras
00	00	00	01	01	—
01	10	10	10	10	Palavra 1
10	11	11	11	11	Palavra 2
11	00	10	00	10	Palavra 3

Designação das Variáveis

Palavra $_i$	Conteúdo $B_6\,B_5\,B_4\,B_3\,B_3\,B_1\,B_0$
1	X X X X 0 1 0
2	0 1 0 0 1 0 0
3	X X X X 0 0 1

A_3	A_2	A_1	A_0	B_8	B_7
Q_1	Q_0	START	i<10	Q_1	Q_0

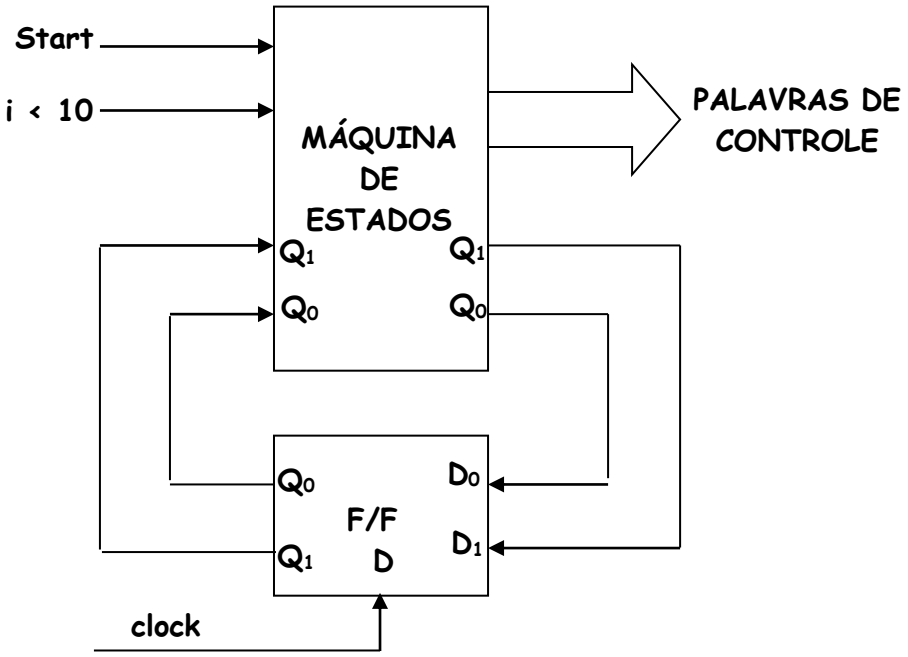
12 - Implementação da máquina de estados finitos - FSM

A montagem da tabela de estados, entradas e saídas permite a construção da tabela PAL (Arranjo lógico programável), conforme é apresentada. A seguir as entradas e as saídas da PAL foram designadas e a tabela resultante apresentada.

Entradas da Tabela: Start, i<10, Q_1 e Q_0
Conteúdo ou Saídas da Tabela: Q_1 e Q_0 , Palavra de Controle.

A_3	A_2	A_1	A_0	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	X	X	X	X	X	X	X
0	0	0	1	0	0	X	X	X	X	X	X	X
0	0	1	0	0	1	X	X	X	X	X	X	X
0	0	1	1	0	1	X	X	X	X	X	X	X
0	1	0	0	1	0	X	X	X	X	0	1	0
0	1	0	1	1	0	X	X	X	X	0	1	0
0	1	1	0	1	0	X	X	X	X	0	1	0
0	1	1	1	1	0	X	X	X	X	0	1	0
1	0	0	0	1	1	0	1	0	0	1	0	0
1	0	0	1	1	1	0	1	0	0	1	0	0
1	0	1	0	1	1	0	1	0	0	1	0	0
1	0	1	1	1	1	0	1	0	0	1	0	0
1	1	0	0	0	0	X	X	X	X	0	0	1
1	1	0	1	1	0	X	X	X	X	0	0	1
1	1	1	0	0	0	X	X	X	X	0	0	1
1	1	1	1	1	0	X	X	X	X	0	0	1

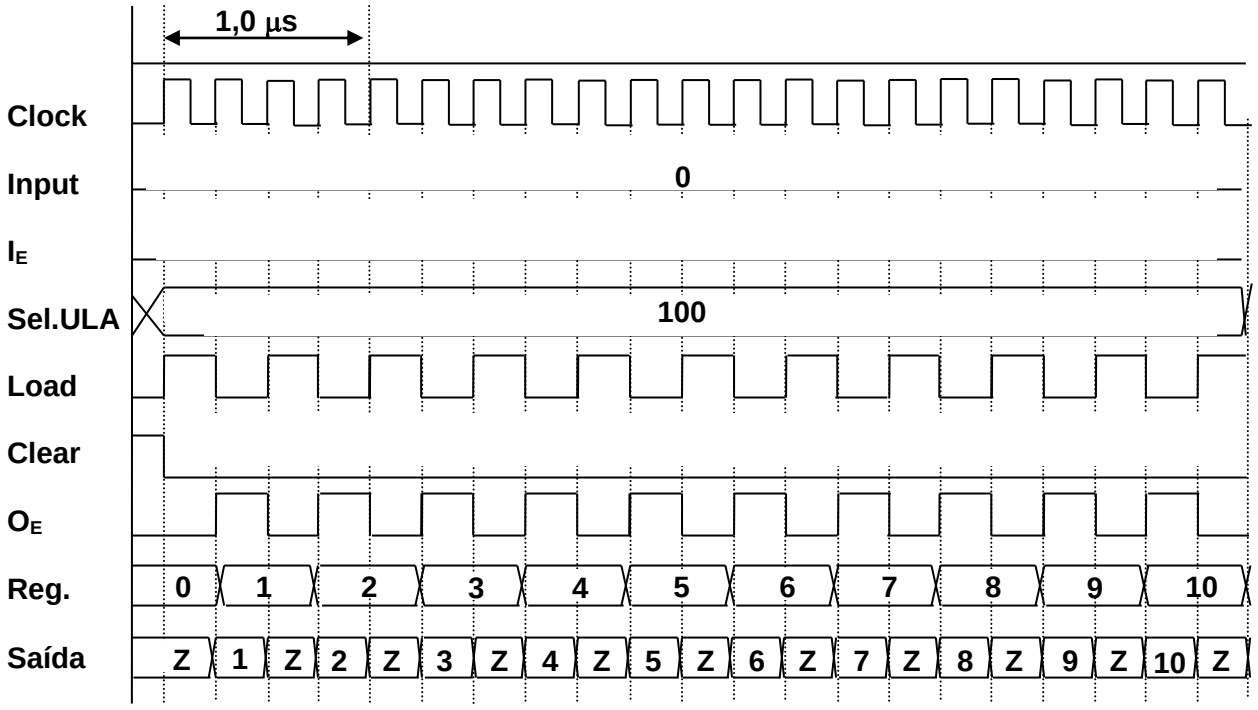
A implementação por máquina de estados da U.C. é mostrada a seguir.



13 - Diagrama de tempo

A seguir apresentamos as formas de onda do fluxo de dados para a realização do algoritmo. As formas de onda apresentadas a seguir mostram os ciclos de relógios necessários para a execução de cada instrução do fluxo de dados.

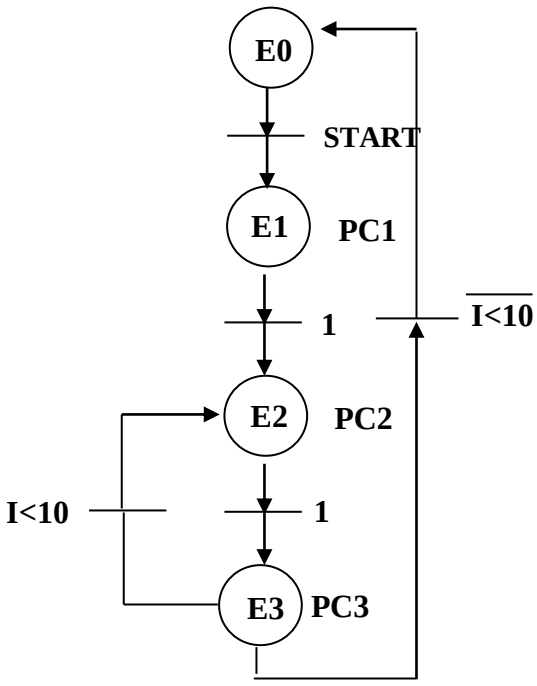
Notar que são necessários dois ciclos para cada contagem, sendo o primeiro ciclo para a palavra de controle 2 e o segundo ciclo para a palavra de controle 3. Estes dois ciclos são repetidos por 10 vezes.



A base de tempo utilizada na simulação do fluxo de dados para a apresentação do algoritmo é de 250 ns, por ciclo de relógio.

14 - Implementação do algoritmo por rede de Petri

A Rede de Petri correspondente ao sistema de controle descrito pode ser expressa como se segue:



Equações de Estado

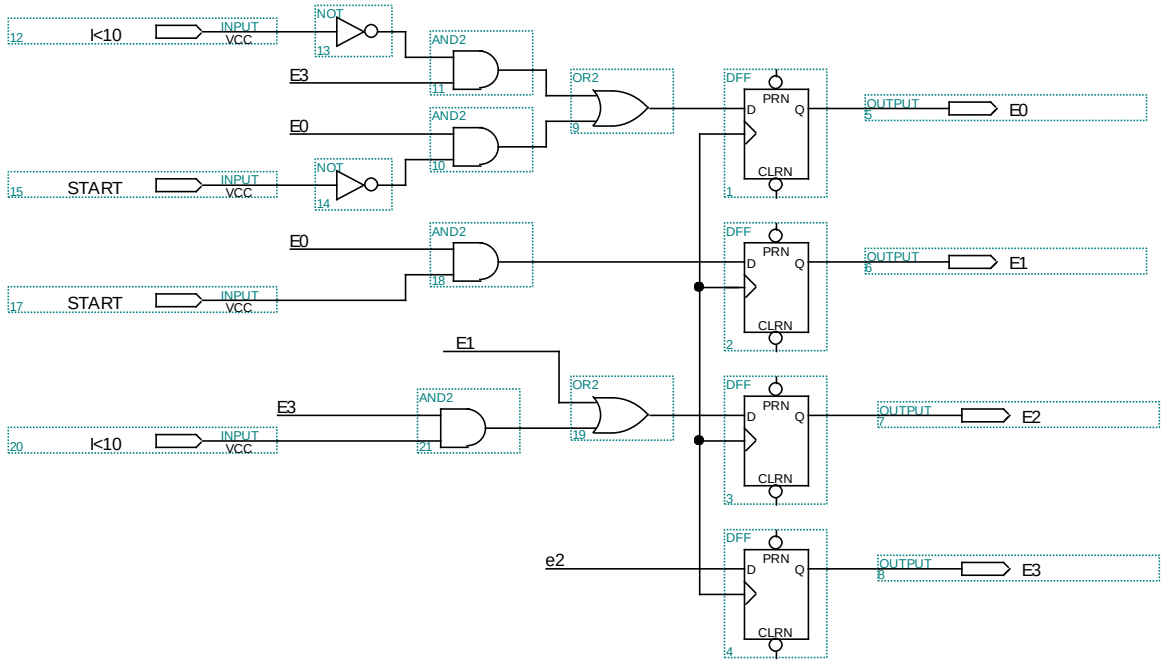
E0=E3*(I<10) + E0*START
E1=E0*START
E2=E1 + E3*(I<10)
E3=E2

Equações de Saída – considerando que PCn corresponde a cada uma das palavras de controle, para definirmos as equações de saída devemos considerar cada variável independentemente. Assim, considerando que os casos irrelevantes (X) assumam valor lógico zero teremos:

LOAD=E2
CLEAR=E1
O_E=E3
ULA_2=E2

As demais variáveis (I_E, ULA_1 e ULA_0) apresentam valor lógico zero

A implementação desta rede através de um circuito utilizando flip-flops resulta no diagrama abaixo:



15 - Fluxo de dados mais complexos

Quando um fluxo de dados não contém todas as unidades funcionais necessárias para realizar todas as operações requeridas, especificadas pelo algoritmo, para a solução de um problema, então a solução é selecionar um fluxo de dados mais complexo.

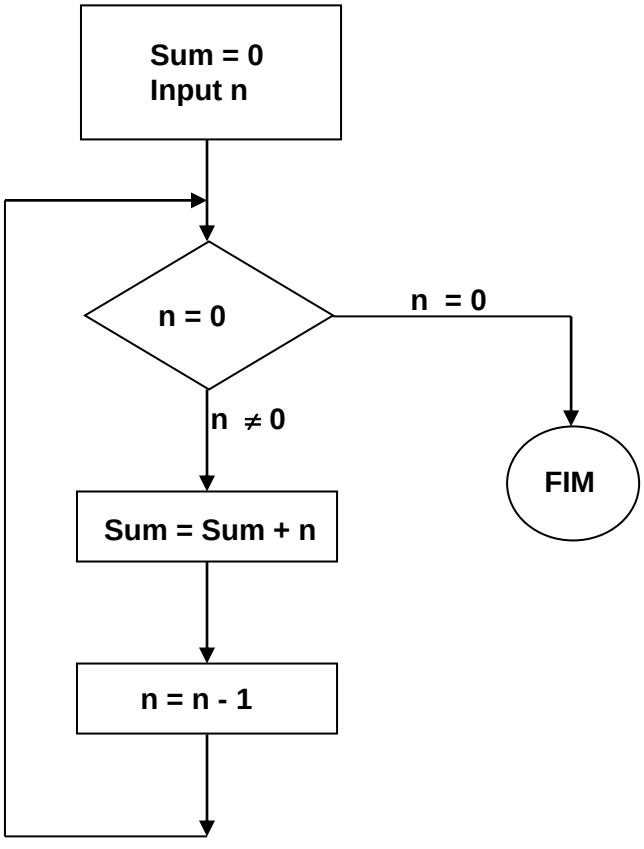
No projeto do fluxo de dados, a meta é encontrar um fluxo de dados simples e reduzido e que possibilite realizar todos os requisitos de um problema e que seja o mais adequado e justo.

A seguir apresenta-se um fluxo de dados mais complexo e que é capaz de somar números **n** até 1 decrescentemente, onde n é um número de entrada e cuja saída é a soma destes números.

O algoritmo é apresentado a seguir:

```
1     Sum = 0
2     input n
3     do while (n ≠ 0)
4     {Sum= Sum + n
5     n= n – 1}
6     output Sum
```

O fluxograma a seguir descreve o desenvolvimento do algoritmo para realizar a operação de soma de n números até 1 na ordem decrescente.



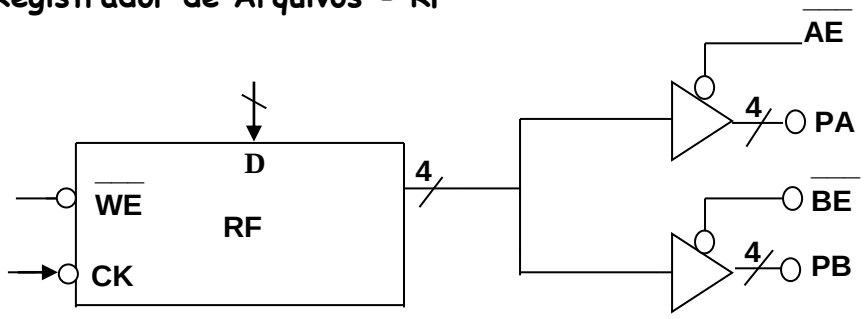
Deve-se incluir no projeto do fluxo de dados, para a solução deste problema, pelo menos 02 registradores. Uma proposta de solução capaz de realizar o algoritmo é apresentada com um fluxo de dados complexo. A diferença fundamental entre o fluxo de dados geral e este novo fluxo de dados complexo é a introdução do registrador de arquivos (RF) com quatro locações. O RF possui um porte de escrita e dois portes de leitura. Para o acesso a um dos portes, uma linha de habilitação do porte deve ser setada juntamente com o endereço devido.

As linhas de habilitação para escrita nos portos são WE para a habilitação da escrita e WA_{1-0} para o endereço do porto para escrita. Para as leituras são RAE para habilitação da leitura do porto A e RBE para habilitação da leitura do porto B. Os sinais RAA_{1-0} para o endereço do porto A e RBA_{1-0} para o endereço de leitura para o porto B. Os portos A e B podem ser lidos simultaneamente e estes portos são conectados respectivamente aos operandos A e B da ULA. A saída da ULA é passada através de um registrador (deslocador), cuja função é especificada na tabela da verdade a seguir. A saída do deslocador é roteada de volta para o RF, via MUX e também pode ser enviada à saída externa habilitando o controle do tri-estado do buffer. A largura do fluxo de dados é de oito bits.

Vale ressaltar que o processo de escrita desse registrador de arquivos é síncrono com a borda de descida do relógio (clock), possibilitando como veremos nas palavras de controle, operações concomitantes como a leitura e o processamento de um dado de um certo endereço para a escrita do resultado no mesmo ou outro endereço e tudo realizado durante um mesmo ciclo de relógio (clock).

SH_1	SH_0	**Operação do deslocador**
0	**0**	**Passagem**
0	**1**	**Desloca a esquerda e preenche com 0**
1	**0**	**Desloca a direita e preenche com 0**
1	**1**	**Rotate a direita**

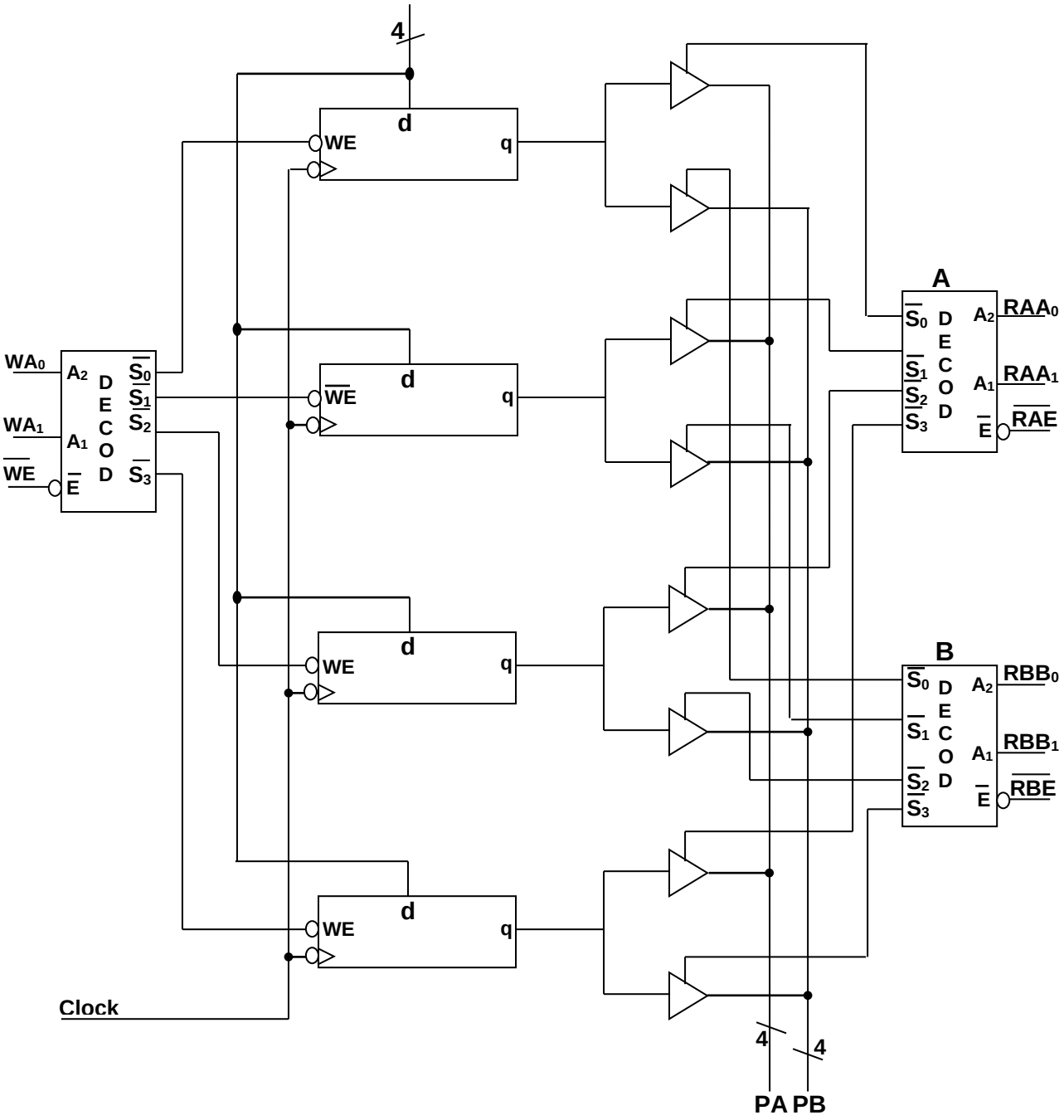
Bloco do Registrador de Arquivos - RF



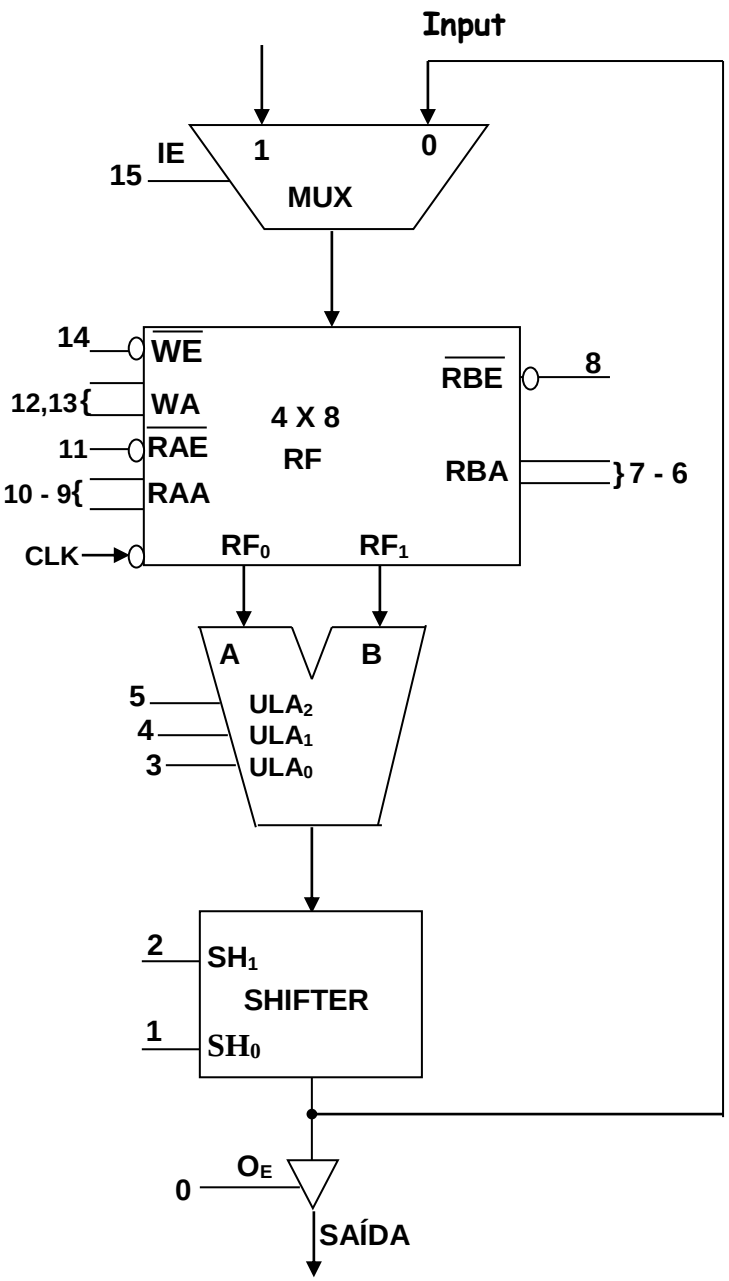
PA= Porto A (saída)
PB= Porto B (saída)
$\overline{WE}$= Habilita reg.
D= Entrada de dados

$\overline{AE}$= Habilita porto A
$\overline{BE}$= Habilita porto B

Diagrama de blocos do registrador de arquivos - RF



Fluxo de Dados



ULA_2	ULA_1	ULA_O	OPERAÇÃO
0	0	0	passagem A
0	0	1	$A \wedge B$
0	1	0	$A \vee B$
0	1	1	A'
1	0	0	A + B
1	0	1	A – B
1	1	0	A + 1
1	1	1	A - 1

V → Lógica OR

∧ → Lógica AND

' → Complemento lógico

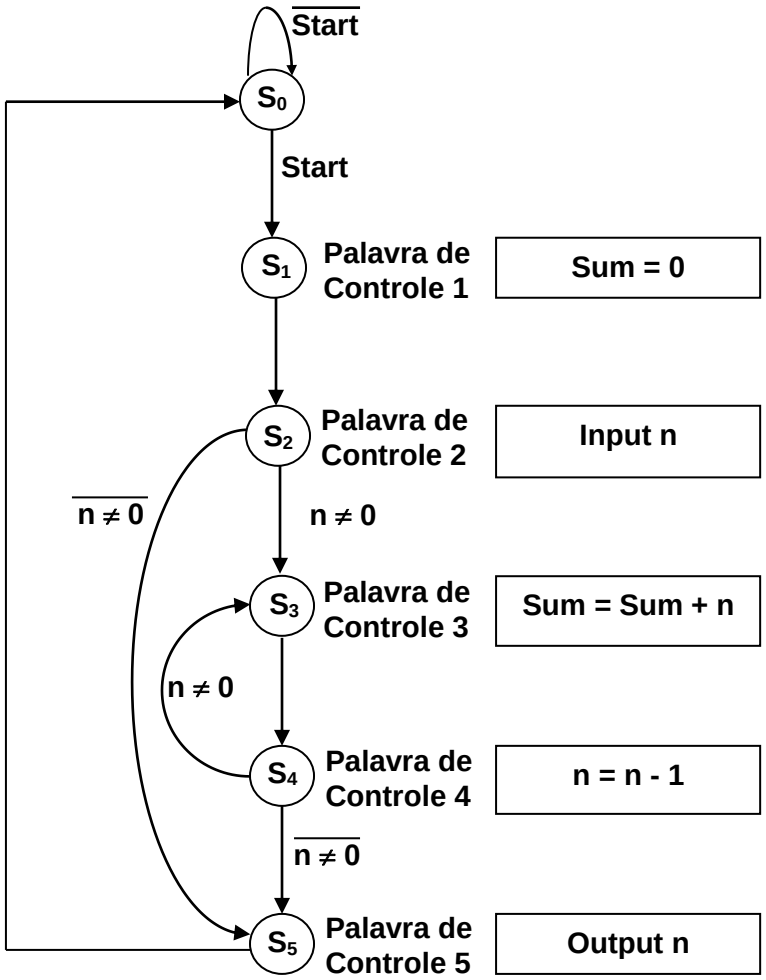
A unidade de controle deverá gerar as seguintes palavras de controle para o fluxo de dados complexo, a fim de modelar o algoritmo.

Palavras de Controle	INSTRUÇÕES	I_E	$\overline{WE}$ 14	$WA_{1,0}$ 13,12	$\overline{RAE}$ 11	$RAA_{1,0}$ 10 - 9	$\overline{RBE}$ 8	$RBA_{1,0}$ 7 – 6	$ULA_{2,1,0}$ 5 – 3	$SH_{1,0}$ 2 – 1	O_E 0
1	sum= o	0	0	00	0	00	0	00	101	00	0
2	input n	1	0	01	1	XX	1	XX	XXX	XX	0
3	sum= sum + n	0	0	00	0	00	0	01	100	00	0
4	n= n – 1	0	0	01	0	01	1	XX	111	00	0
5	output sum	X	1	XX	0	00	1	XX	000	00	1

ULA – 101 s – subtrai, 100 a – adiciona, 111 d – decrementa, 000 p – passagem A.

Implementação por diagrama de estados

A implementação do problema pelo diagrama de estados, apresenta o estado S_0, como um estado inicial que só inicializa quando o operador pressiona uma tecla de Start. O diagrama de estados poderia inicializar em S_1. Para os estados conectados às transições que não são condicionadas à qualquer variável, a evolução para o outro estado é dita incondicional e depende somente do clock de entrada.



Designando os estados S_0 a S_5 do diagrama de estados apresentado por:

S_0 = 000 ; S_1 = 001; S_2 = 010; S_3 = 011; S_4 = 100; S_5 = 101.

Para a implementação da FSM, são necessários três F/Fs, designados por Q_2, Q_1 e Q_0, sendo as variáveis externas Start, n ≠ 0, as quais combinadas resultam em quatro condições. As condições das entradas serão 1-00, 2-01, 3-10 e 4-11.

Na designação dos estados de S_0 a S_5, não houve qualquer preocupação com os Hazares e glitches, os quais podem ser evitados na designação obedecendo o princípio da adjacência.

Implementação da máquina de estados finitos - FSM

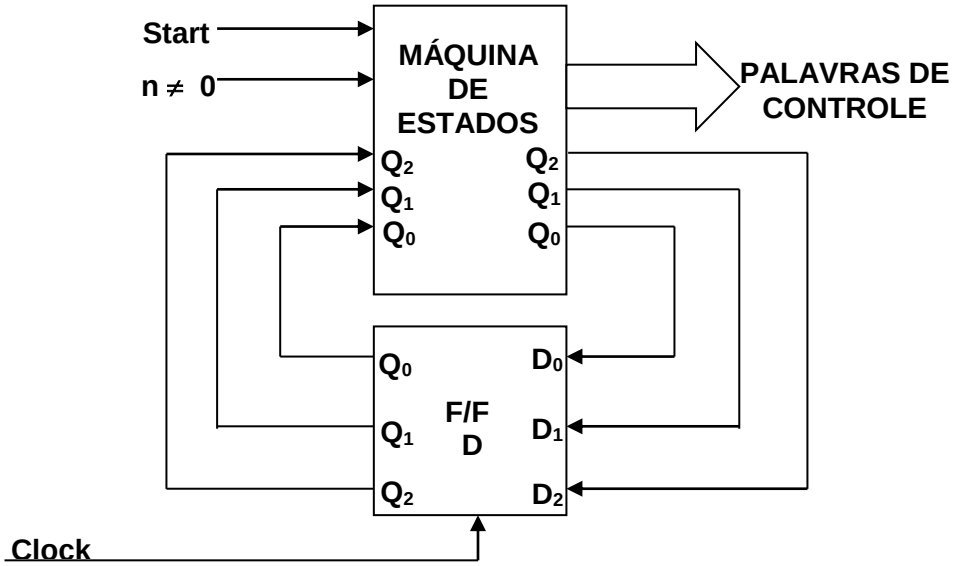
Podemos montar a tabela de estados, entradas e saídas, conforme será apresentado a seguir, definindo os estados futuros pelo diagrama de estados.

Estado	Próximo Estado Q_{2+1} Q_{1+1} Q_{O+1}				
Atual $Q_2 Q_1 Q_0$	Entradas start, n ≠ 0 00	01	10	11	Palavra de controle
S_0 - 000	000	000	001	001	Palavra de controle 1
S_1 - 001	010	010	010	010	Palavra de controle 2
S_2 - 010	011	011	011	011	Palavra de controle 3
S_3 - 011	100	100	100	100	Palavra de controle 4
S_4 - 100	101	011	101	011	Palavra de controle 5
S_5 - 101	000	000	000	000	
S_6- 110	000	000	000	000	
S_7 - 111	000	000	000	000	

A partir da tabela de estados, entradas e saídas, a seguir implementaremos a FSM utilizando uma PAL, como lógica combinatória.

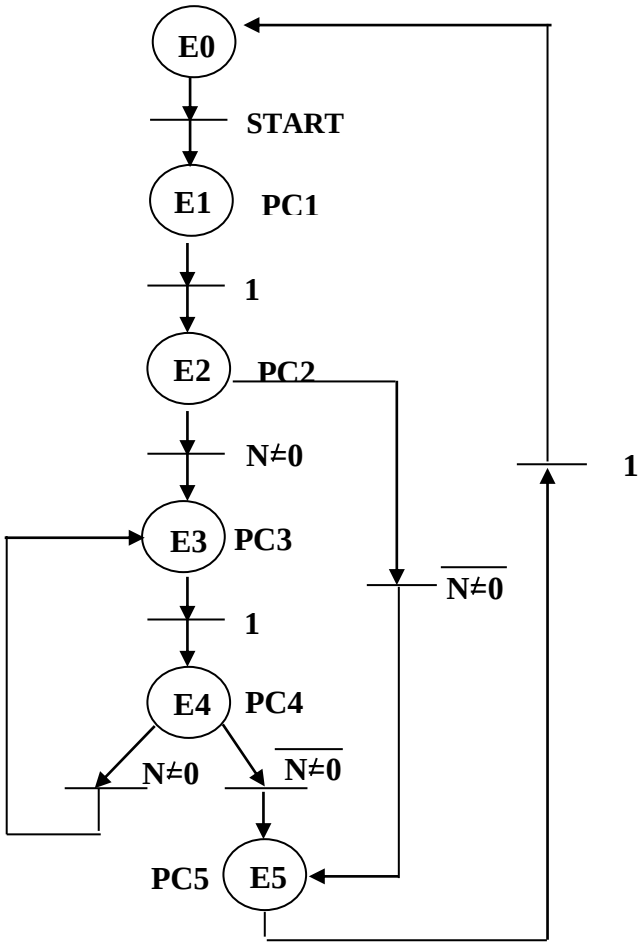
Endereços					Conteúdo												
					Q_2	Q_1	Q_0	I_E	W_E	WA_{1-0}	RAE	RAA_{1-0}	RBE	RBB_{1-0}	$ULA_{2,1,0}$	$SH_{1,0}$	O_E
Q_2	Q_1	Q_0	ST	n	B_{18}	B_{17}	B_{16}	B_{15}	B_{14}	$B_{13}B_{12}$	B_{11}	$B_{10}B_9$	B_8	B_7B_6	$B_5B_4B_3$	B_2B_1	B_0
0	0	0	0	0	0	0	0	X	X	XX	X	XX	X	XX	XXX	XX	X
0	0	0	0	1	0	0	0	X	X	XX	X	XX	X	XX	XXX	XX	X
0	0	0	1	0	0	0	1	X	X	XX	X	XX	X	XX	XXX	XX	X
0	0	0	1	1	0	0	1	X	X	XX	X	XX	X	XX	XXX	XX	X
0	0	1	0	0	0	1	0	0	0	00	0	00	0	00	101	00	0
0	0	1	0	1	0	1	0	0	0	00	0	00	0	00	101	00	0
0	0	1	1	0	0	1	0	0	0	00	0	00	0	00	101	00	0
0	0	1	1	1	0	1	0	0	0	00	0	00	0	00	101	00	0
0	1	0	0	0	0	1	1	1	0	01	1	XX	X	XX	XXX	XX	0
0	1	0	0	1	0	1	1	1	0	01	1	XX	X	XX	XXX	XX	0
0	1	0	1	0	0	1	1	1	0	01	1	XX	X	XX	XXX	XX	0
0	1	0	1	1	0	1	1	1	0	01	1	XX	X	XX	XXX	XX	0
0	1	1	0	0	1	0	0	0	0	00	0	00	0	01	100	00	0
0	1	1	0	1	1	0	0	0	0	00	0	00	0	01	100	00	0
0	1	1	1	0	1	0	0	0	0	00	0	00	0	01	100	00	0
0	1	1	1	1	1	0	0	0	0	00	0	00	0	01	100	00	0
1	0	0	0	0	1	0	1	0	0	01	0	01	1	XX	111	00	0
1	0	0	0	1	0	1	1	0	0	01	0	01	1	XX	111	00	0
1	0	0	1	0	1	0	1	0	0	01	0	01	1	XX	111	00	0
1	0	0	1	1	0	1	1	0	0	01	0	01	1	XX	111	00	0
1	0	1	0	0	0	0	0	X	1	XX	0	00	1	XX	000	00	1
1	0	1	0	1	0	0	0	X	1	XX	0	00	1	XX	000	00	1
1	0	1	1	0	0	0	0	X	1	XX	0	00	1	XX	000	00	1
1	0	1	1	1	0	0	0	X	1	XX	0	00	1	XX	000	00	1
1	1	0	0	0	0	0	0	X	X	XX	X	XX	X	XX	XXX	XX	X
1	1	0	0	1	0	0	0	X	X	XX	X	XX	X	XX	XXX	XX	X
1	1	0	1	0	0	0	0	X	X	XX	X	XX	X	XX	XXX	XX	X
1	1	0	1	1	0	0	0	X	X	XX	X	XX	X	XX	XXX	XX	X
1	1	1	0	0	0	0	0	X	X	XX	X	XX	X	XX	XXX	XX	X
1	1	1	0	1	0	0	0	X	X	XX	X	XX	X	XX	XXX	XX	X
1	1	1	1	0	0	0	0	X	X	XX	X	XX	X	XX	XXX	XX	X
1	1	1	1	1	0	0	0	X	X	XX	X	XX	X	XX	XXX	XX	X

A seguir é apresentada a implementação da unidade de controle para o fluxo de dados complexo por máquina de estados



Implementação do algoritmo por rede de Petri

A Rede de Petri correspondente ao sistema de controle descrito pode ser expressa como se segue:



Equações de Estado

E0=E5+E0*~~START~~
E1=E0*START
E2=E1
E3=E2*(N=0)+E4*(N=0)
E4=E3
E5=E4*(N≠0)+E2*(N=0)

Equações de Saída de maneira semelhante ao exercício anterior teremos:

I_E = E2
WE = E5
$\overline{WA_0}$ = E2+E4
$\overline{RAE}$ = E2
RAA_0 = E4
~~RB~~E = E2+E4+E5
RBA_0 = E3
ULA_2 = E1+E3+E4
ULA_1 = E4
ULA_0 = E1+E4
$\overline{O_E}$ = E5

A implementação desta rede através de um circuito utilizando flip-flops resulta no diagrama abaixo:

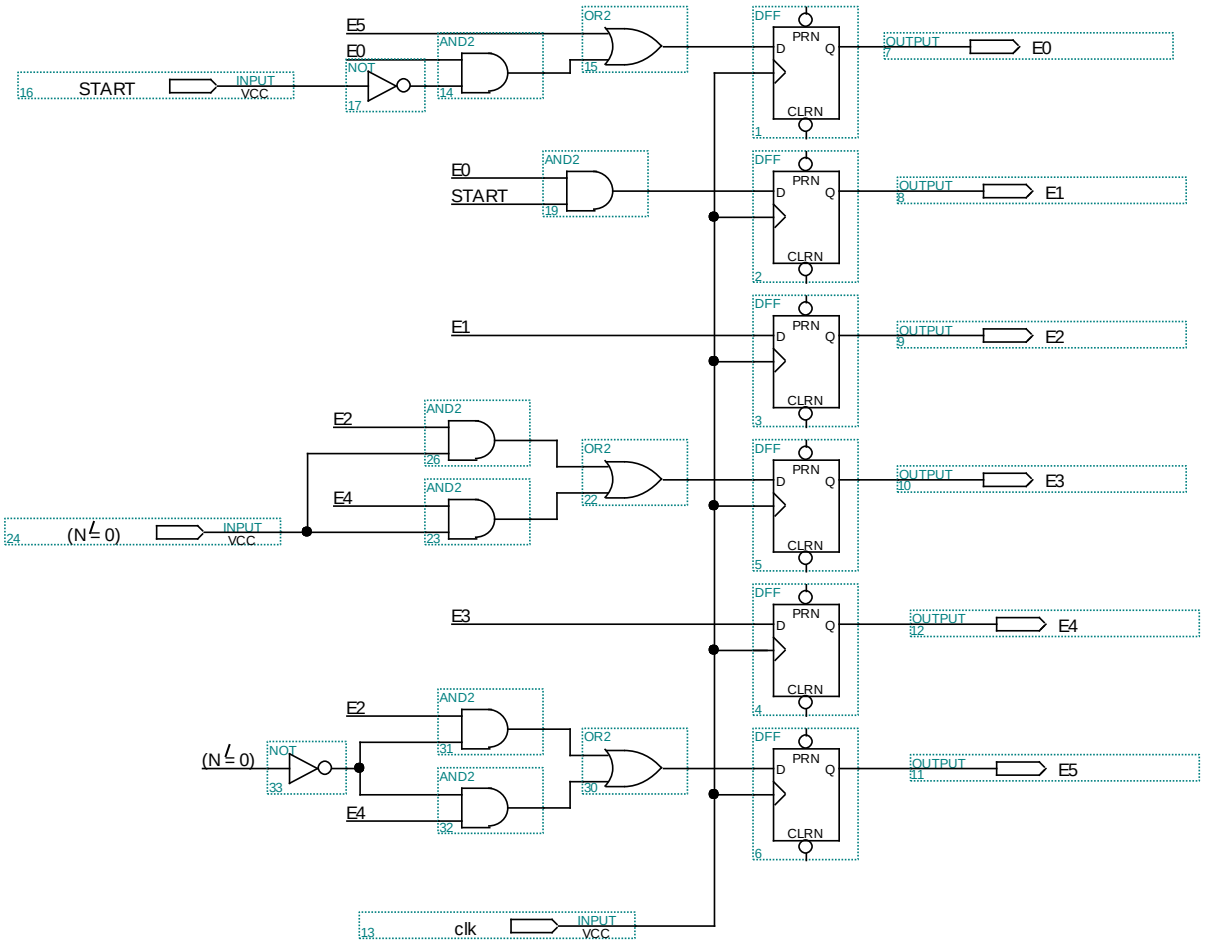
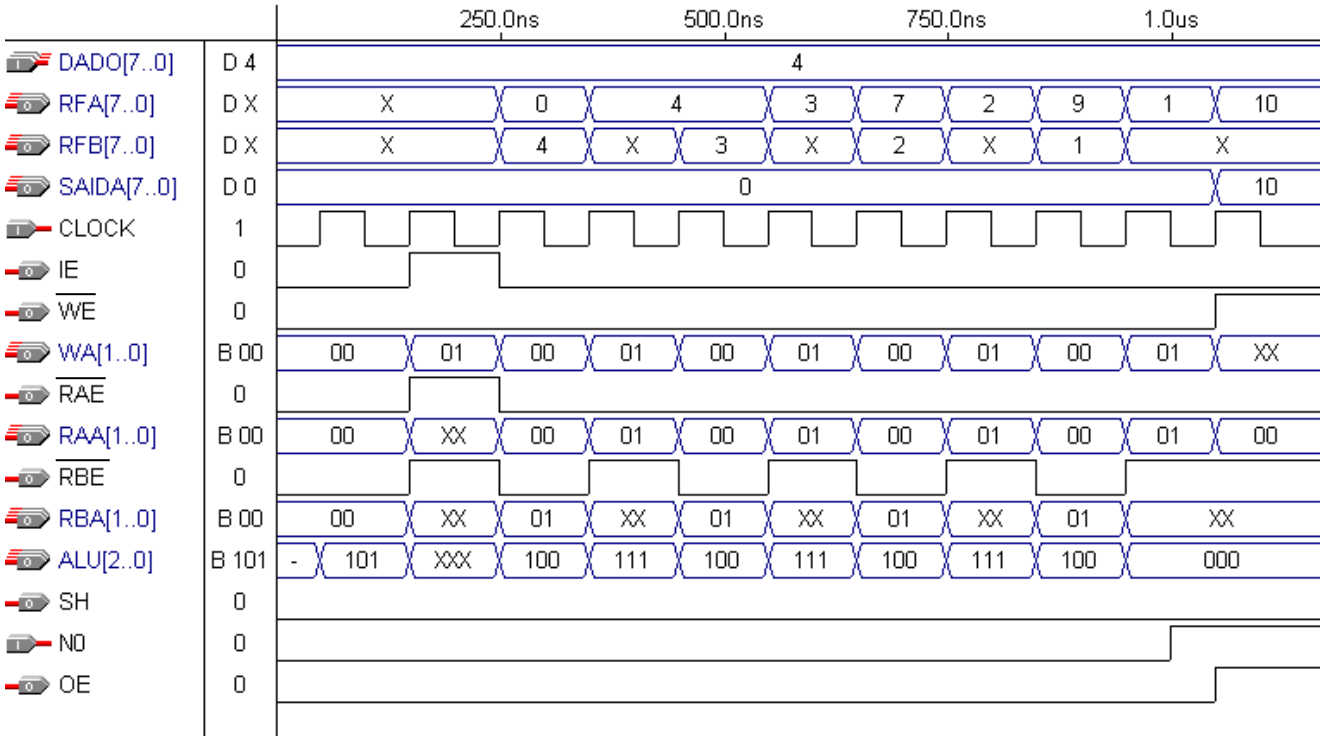


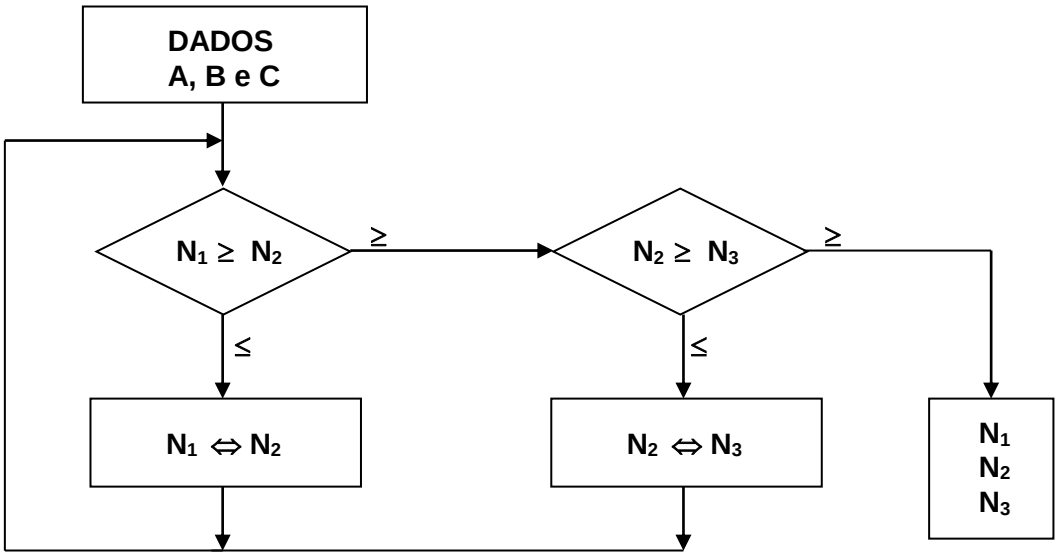
Diagrama de tempo

O diagrama de tempo a seguir, entra com o número 4 e apresenta na saída a somatória dos números de 4 até 1, com resultado 10. A apresentação na saída do número 10 é a solução final do problema. O relógio marca o tempo e para cada ciclo de relógio teremos uma instrução executada. O tempo total do programa é de 1,1μs. O período de cada ciclo de relógio é de 100ns.



V - EXERCÍCIOS DE APLICAÇÃO

EXEMPLO 1 :Colocar 3 números N_1, N_2 e N_3 em ordem decrescente.
O algoritmo pode ser descrito conforme o fluxograma apresentado a seguir.



Utilizando a arquitetura do fluxo de dados complexo, tem-se que inicialmente entrar com os números N_1, N_2 e N_3. Os números ficarão armazenados no registrador de arquivos nos endereços 0-00, 1-01, 2-02 e o endereço 3-11 servirá como armazenagem temporária.

Na comparação dos números N_1 com N_2 , sendo o resultado positivo, o algoritmo manda realizar a segunda comparação entre N_2 e N_3 e se ainda o resultado for positivo então os números no registrador de arquivos os quais permaneceram nas suas locações iniciais serão

apresentados na saída realizando a leitura dos números N_1, N_2 e N_3 no registrador de arquivos na ordem 0-00, 1-01 e 2-02 pois $N_1 \geq N_2$ e $N_2 \geq N_3$.

Se na comparação entre N_1 e N_2, o resultado for negativo, o algoritmo manda trocar N_1 com N_2. A forma de troca de locações entre N_1 e N_2 é realizada utilizando a locação 1-11 do registrador de arquivos como armazenagem temporária. Inicialmente transfere-se o N_1 para esta locação para preservar o número, em seguida transfere-se o número N_2, para a locação antiga de N_1 e uma vez salvo o número, transfere-se o número N_1 para a antiga locação de N_2. O mesmo processo de transferência é utilizado entre N_2 e N_3 caso o resultado da comparação seja negativo. Nesse caso, deve-se realizar uma nova comparação entre N_1 e N_3. E nesta última comparação fica determinada a ordem dos números.

Palavra de controle	Instrução	I_E 15	WE 14	$WA_{1,0}$ 13 -12	RAE 11	$RAA_{1,0}$ 10 - 9	RBE 8	$RBA_{1,0}$ 7 – 6	ULA 5 - 3	SH 2- 1	OE 0
01	Input N_1	1	0	00	1	XX	1	XX	XXX	XX	0
02	Input N_2	1	0	01	1	XX	1	XX	XXX	XX	0
03	Input N_3	1	0	10	1	XX	1	XX	XXX	XX	0
04	N_1 : N_2	0	1	XX	0	00	0	01	101	00	0
05	N_2 : N_3	0	1	XX	0	01	0	10	101	00	0
06	Output N_1	X	1	XX	0	00	1	XX	000	00	1
07	Output N_2	X	1	XX	0	01	1	XX	000	00	1
08	Output N_3	X	1	XX	0	10	1	XX	000	00	1
09	(Temp -11) ← N_1	0	0	11	0	00	1	XX	000	00	0
10	(N_1- 00) ← N_2	0	0	00	0	01	1	XX	000	00	0
11	(N_2-01) ← (Temp-11)	0	0	01	0	11	1	XX	000	00	0
12	(Temp-11) ← N_2	0	0	11	0	01	1	XX	000	00	0
13	(N_2-01) ← N_3	0	0	01	0	10	1	XX	000	00	0
14	(N_3-10) ← (Temp-11)	0	0	10	0	11	1	XX	000	00	0

A unidade de controle determina qual das instruções devem ser executadas de acordo com os sinais de status da ULA. São eles: Positivo e negativo.

As instruções 1,2 e 3 são de entrada dos números no registrador de arquivos respectivamente nas locações 0-00, 1-01 e 2-10. A instrução 4 é executada e dependendo do resultado da ULA, se positivo executa 5, se negativo executa as instruções 9, 10 e 11 e em seguida retorna à execução da próxima instrução a 4. O resultado da ULA na execução 5 pode ter 02 caminhos, no primeiro se o resultado for positivo o programa salta para a execução das instruções 6,7 e 8 e o programa finaliza. No segundo se o resultado da ULA for negativo o programa salta para a execução das instruções 12, 13 e 14 e retorna para a instrução 4 fazendo o ciclo novamente.

Implementação do diagrama de Estados

A implementação do problema pelo diagrama de estados, apresenta o estado S_0, como um estado inicial e o programa só inicializa quando o operador pressionar a tecla de Start. O diagrama de estados poderia inicializar em S_1. Para os estados conectados pelas transições as quais não são condicionadas à qualquer variável, a evolução para o outro estado é dita incondicional e depende somente do clock de entrada.

Conforme apresentado no fluxograma do algoritmo que ordena os números em ordem decrescente, a evolução de um estado para outro com as transições condicionais é dependente do status da ULA, onde o resultado da operação de comparação é realizada e

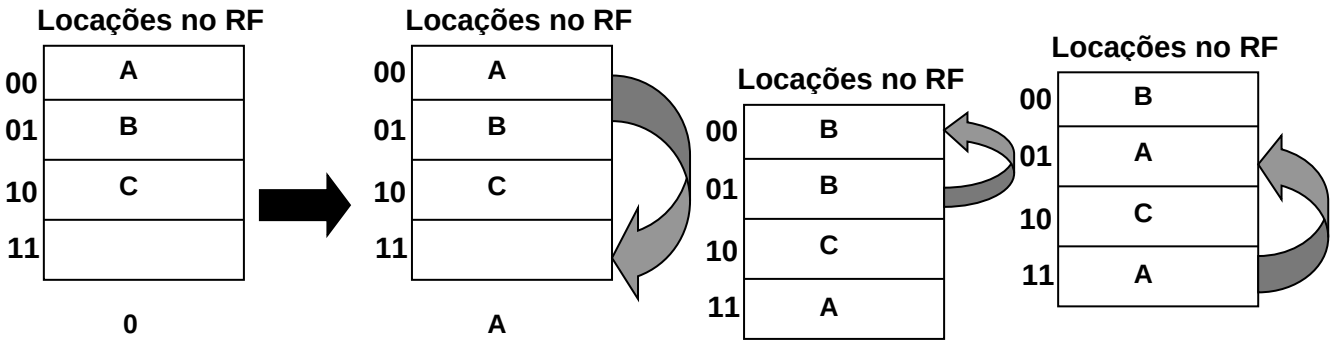
indicada pelo bit de status P, sendo igual a zero quando o resultado é positivo ou igual a um quando o resultado é negativo. O bit P de status é indicado na ULA após a subtração realizada entre os dois operandos que podem ser ($N_1 - N_2$) ou ($N_2 - N_3$).

Quando os números ficarem ordenados no registrador de arquivos, sendo o maior dos 03 números alocado no endereço 00, o número intermediário alocado no endereço 01 e o menor dos 03 números alocado no endereço 10 o processo está finalizado.

O endereço 11 é utilizado na transferência dos números para as locações e serve como armazenagem temporária de um dos números. No diagrama de estados esta locação é identificada pela variável Temp.

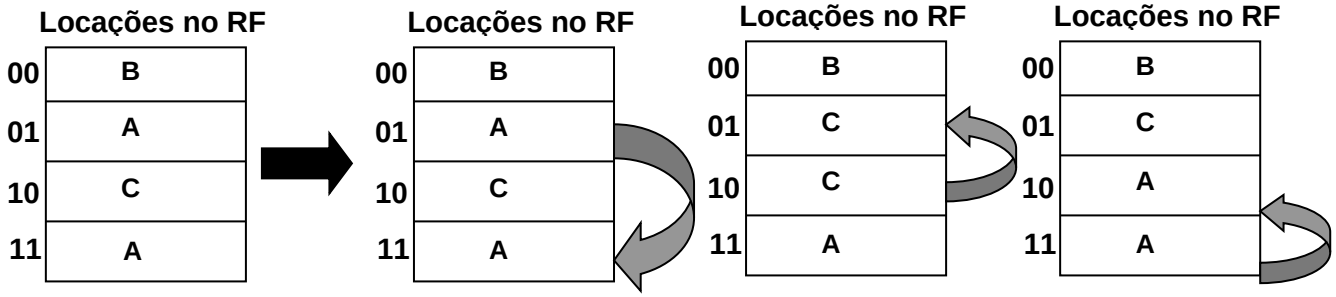
O exemplo a seguir mostra 03 números A, B e C, como a transferência é realizada e como os números trocam de locações para a ordenação destes números no registrador de arquivos. O exemplo demonstra como é a transferência dos números nas locações do registrador de arquivos quando entra-se com os números A, B e C sendo C o maior número, B o número intermediário e A o menor número.

1) B maior ou igual A ($B \geq A$).



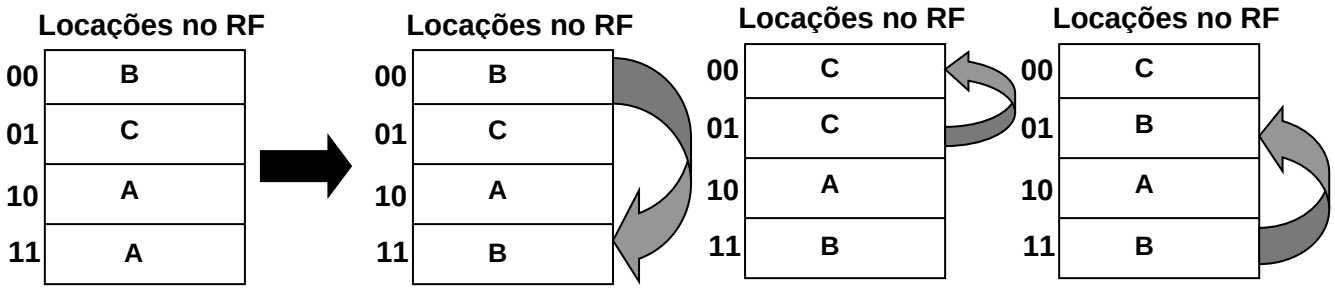
O número A é transferido para a locação temporária 11, em seguida o número B é transferido para a antiga locação de A e por fim o número A é transferido para a locação antiga de B. O passo seguinte do algoritmo é a segunda comparação entre os números A e C. Caso A fosse maior do que C o algoritmo se encerraria, mas como não é segue então os próximos passos.

2) C maior ou igual a A ($C \geq A$).



O número A é transferido para a locação temporária 11, em seguida o número C é transferido para a antiga locação de A e por fim o número A é transferido para a locação antiga de C. O passo seguinte do algoritmo é a terceira comparação entre os números B e C. Caso B fosse maior do que C o algoritmo se encerraria, mas como não é segue então os próximos passos. Finalizando, procede-se a transferência entre as locações e ordenação dos números decrescentemente, conforme demonstrado abaixo.

3) C maior ou igual a B ($C \geq B$).

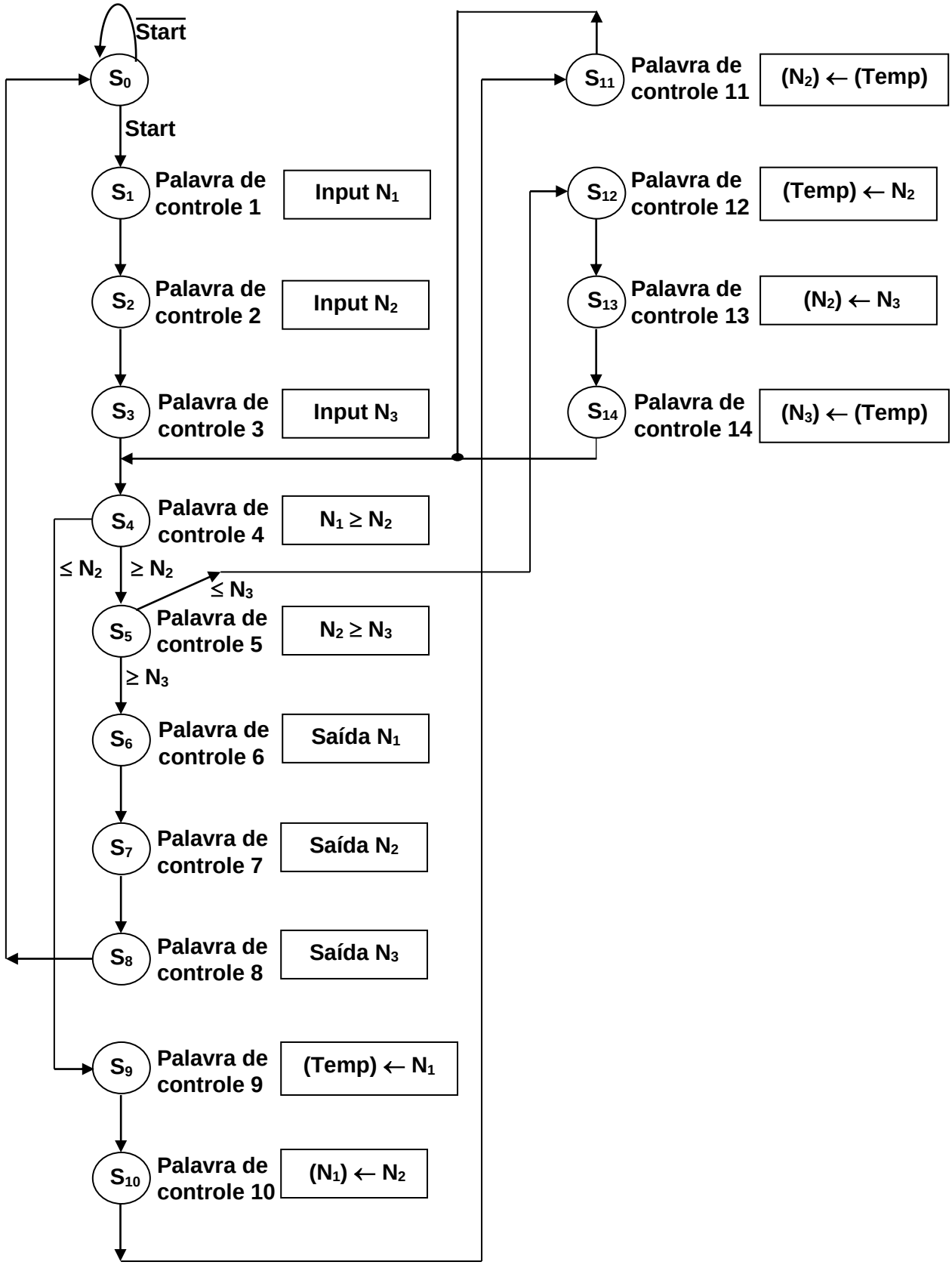


A apresentação dos números C, B e A nas locações 00, 01 e 10 respectivamente encerra o algoritmo.

Para a implementação da unidade de controle que realiza o algoritmo de comparação de 03 números, é apresentado o diagrama de estados a seguir. As locações 00, 01,10 e 11 denominadas como N_1, N_2, N_3 e Temp, são ilustradas no diagrama de estados. Quando utilizamos um parênteses na locação (N_i ou Temp), significa o endereço.

Implementação do Diagrama de Estados e da Máquina de Estados

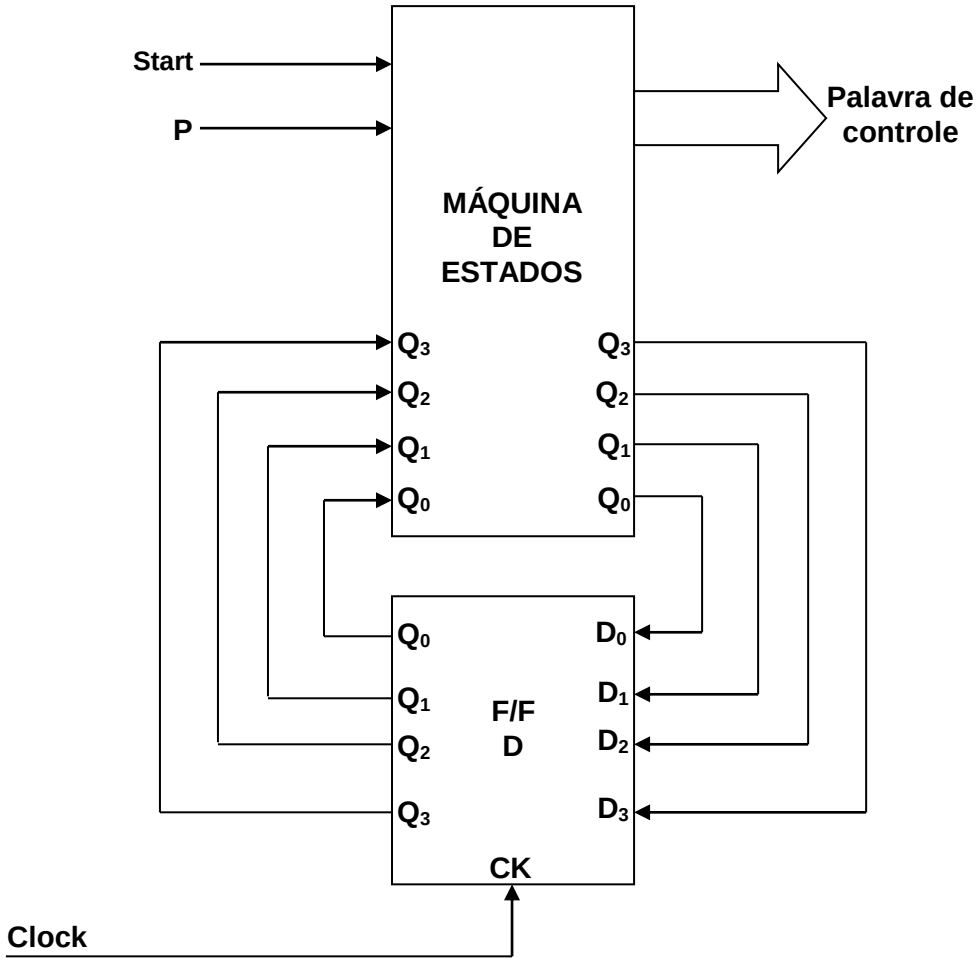
a) Diagrama de estados do algoritmo.



b) Tabela de Estados, presentes, futuros, entradas e saídas

ATUAL	ENTRADAS Start, P				Palavra de Controle
$Q_3Q_2Q_1Q_0$	00	01	10	11	CW
0000	0000	0000	0001	0001	CW = 00
0001	0010	0010	0010	0010	CW = 01
0010	0011	0011	0011	0011	CW = 02
0011	0100	0100	0100	0100	CW = 03
0100	0101	1001	0101	1001	CW = 04
0101	0110	1100	0110	1100	CW = 05
0110	0111	0111	0111	0111	CW = 06
0111	1000	1000	1000	1000	CW = 07
1000	0000	0000	0000	0000	CW = 08
1001	1010	1010	1010	1010	CW = 09
1010	1011	1011	1011	1011	CW = 10
1011	0100	0100	0100	0100	CW = 11
1100	1101	1101	1101	1101	CW = 12
1101	1110	1110	1110	1110	CW = 13
1110	0100	0100	0100	0100	CW = 14
1111	0000	0000	0000	0000	CW = 00

c) Representação por máquina de estados.



A seguir identificando nos endereços da PAL, os estados atuais, entradas, saídas e palavra de controle.
d) Mapa da PAL, para a implementação da tabela combinacional.

Q_3	Q_2	Q_1	Q_0	ST	P	Q_3	Q_2	Q_1	Q_0	I_E	WE	WA_{0-1}	RAE	RAA_{0-1}	RBE	RBA_{0-1}	ULA_3	ULA_2	ULA_1	SH	OE
A_5	A_4	A_3	A_2	A_1	A_0	B_{19}	B_{18}	B_{17}	B_{16}	B_{15}	B_{14}	$B_{13}B_{12}$	B_{11}	B_{10} B_9	B_8	B_7 B_6	B_5	B_4	B_3	B_2B_1	B_0
0	0	0	0	0	0	0	0	0	0	X	1	XX	1	XX	1	XX	X	X	X	XX	0
0	0	0	0	0	1	0	0	0	1	X	1	XX	1	XX	1	XX	X	X	X	XX	0
0	0	0	0	1	0	0	0	0	0	X	1	XX	1	XX	1	XX	X	X	X	XX	0
0	0	0	0	1	1	0	0	0	1	X	1	XX	1	XX	1	XX	X	X	X	XX	0
0	0	0	1	0	0	0	0	1	0	1	0	00	1	XX	1	XX	X	X	X	XX	0
0	0	0	1	0	1	0	0	1	0	1	0	00	1	XX	1	XX	X	X	X	XX	0
0	0	0	1	1	0	0	0	1	0	1	0	00	1	XX	1	XX	X	X	X	XX	0
0	0	0	1	1	1	0	0	1	0	1	0	00	1	XX	1	XX	X	X	X	XX	0
0	0	1	0	0	0	0	0	1	1	1	0	01	1	XX	1	XX	X	X	X	XX	0
0	0	1	0	0	1	0	0	1	1	1	0	01	1	XX	1	XX	X	X	X	XX	0
0	0	1	0	1	0	0	0	1	1	1	0	01	1	XX	1	XX	X	X	X	XX	0
0	0	1	0	1	1	0	0	1	1	1	0	01	1	XX	1	XX	X	X	X	XX	0
0	0	1	1	0	0	0	1	0	0	1	0	10	1	XX	1	XX	X	X	X	XX	0
0	0	1	1	0	1	0	1	0	0	1	0	10	1	XX	1	XX	X	X	X	XX	0
0	0	1	1	1	0	0	1	0	0	1	0	10	1	XX	1	XX	X	X	X	XX	0
0	0	1	1	1	1	0	1	0	0	1	0	10	1	XX	1	XX	X	X	X	XX	0
0	1	0	0	0	0	0	1	0	1	0	1	XX	0	01	0	01	1	0	1	00	0
0	1	0	0	0	1	1	0	0	1	0	1	XX	0	01	0	01	1	0	1	00	0
0	1	0	0	1	0	0	1	0	1	0	1	XX	0	01	0	01	1	0	1	00	0
0	1	0	0	1	1	1	0	0	1	0	1	XX	0	01	0	01	1	0	1	00	0
0	1	0	1	0	0	0	1	1	0	0	1	XX	0	10	0	10	1	0	1	00	0
0	1	0	1	0	1	1	1	0	0	0	1	XX	0	10	0	10	1	0	1	00	0
0	1	0	1	1	0	0	1	1	0	0	1	XX	0	10	0	10	1	0	1	00	0
0	1	0	1	1	1	1	1	0	0	0	1	XX	0	10	0	10	1	0	1	00	0
0	1	1	0	0	0	0	0	1	1	X	1	XX	0	00	1	XX	0	0	0	00	0
0	1	1	0	0	1	0	0	1	1	X	1	XX	0	00	1	XX	0	0	0	00	0
0	1	1	0	1	0	0	0	1	1	X	1	XX	0	00	1	XX	0	0	0	00	0
0	1	1	0	1	1	0	0	1	1	X	1	XX	0	00	1	XX	0	0	0	00	0
0	1	1	1	0	0	1	0	0	0	X	1	XX	0	00	1	XX	0	0	0	00	1
0	1	1	1	0	1	1	0	0	0	X	1	XX	0	00	1	XX	0	0	0	00	1
0	1	1	1	1	0	1	0	0	0	X	1	XX	0	00	1	XX	0	0	0	00	1
0	1	1	1	1	1	1	0	0	0	X	1	XX	0	00	1	XX	0	0	0	00	1
1	0	0	0	0	0	0	0	0	0	X	1	XX	0	00	1	XX	0	0	0	00	1

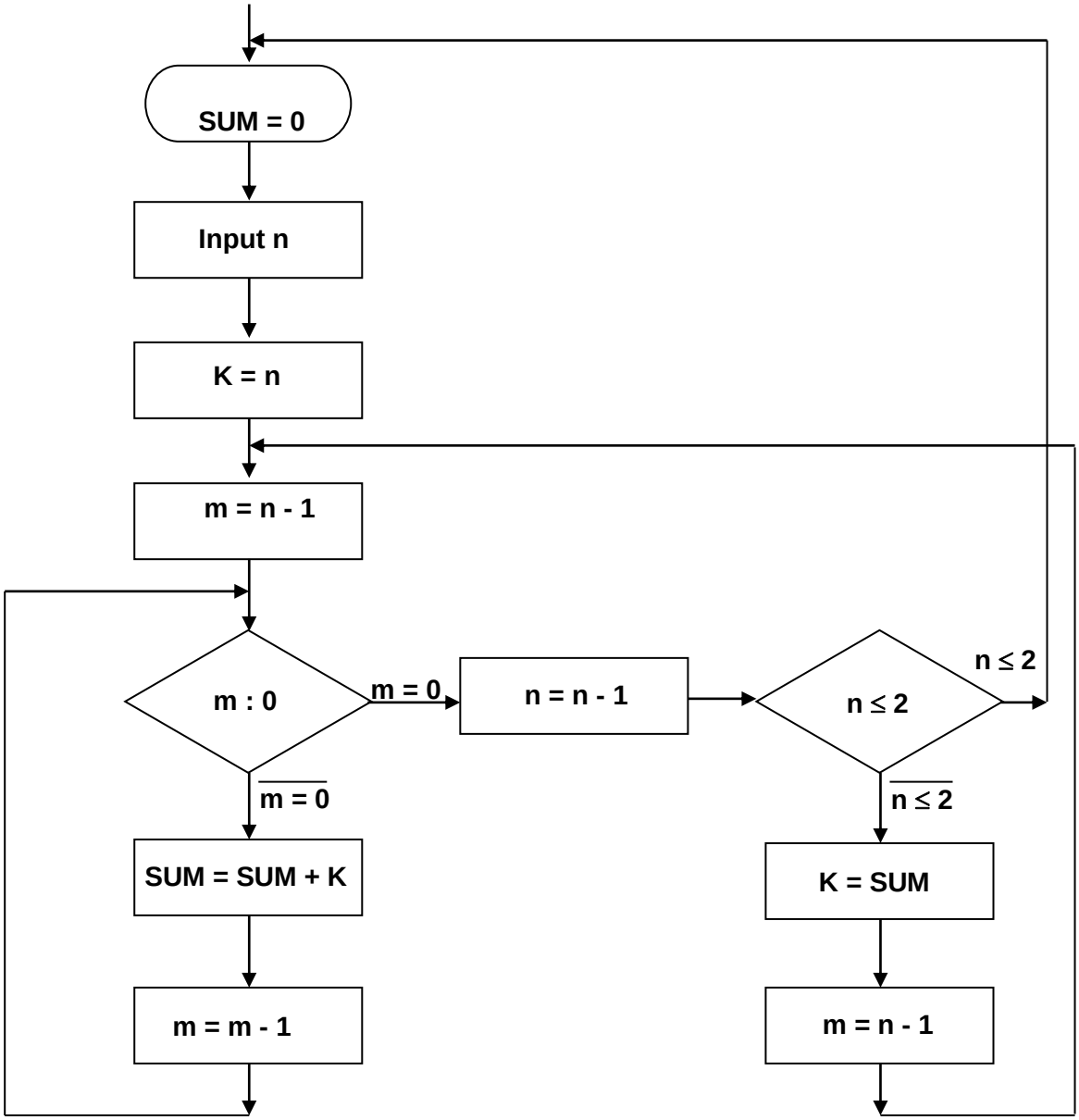
1	0	0	0	0	1	0	0	0	0	X	1	XX	0	00	1	XX	0	0	0	00	1
1	0	0	0	1	0	0	0	0	0	X	1	XX	0	00	1	XX	0	0	0	00	1
1	0	0	0	1	1	0	0	0	0	X	1	XX	0	00	1	XX	0	0	0	00	1
1	0	0	1	0	0	1	0	1	0	0	0	11	0	00	1	XX	0	0	0	00	0
1	0	0	1	0	1	1	0	1	0	0	0	11	0	00	1	XX	0	0	0	00	0
1	0	0	1	1	0	1	0	1	0	0	0	11	0	00	1	XX	0	0	0	00	0
1	0	0	1	1	1	1	0	1	0	0	0	11	0	00	1	XX	0	0	0	00	0
1	0	1	0	0	0	1	0	1	1	0	0	00	0	01	1	XX	0	0	0	00	0
1	0	1	0	0	1	1	0	1	1	0	0	00	0	01	1	XX	0	0	0	00	0
1	0	1	0	1	0	1	0	1	1	0	0	00	0	01	1	XX	0	0	0	00	0
1	0	1	0	1	1	1	0	1	1	0	0	00	0	01	1	XX	0	0	0	00	0
1	0	1	1	0	0	0	1	0	0	0	0	01	0	11	1	XX	0	0	0	00	0
1	0	1	1	0	1	0	1	0	0	0	0	01	0	11	1	XX	0	0	0	00	0
1	0	1	1	1	0	0	1	0	0	0	0	01	0	11	1	XX	0	0	0	00	0
1	0	1	1	1	1	0	1	0	0	0	0	01	0	11	1	XX	0	0	0	00	0
1	1	0	0	0	0	1	1	0	1	0	0	11	0	01	1	XX	0	0	0	00	0
1	1	0	0	0	1	1	1	0	1	0	0	11	0	01	1	XX	0	0	0	00	0
1	1	0	0	1	0	1	1	0	1	0	0	11	0	01	1	XX	0	0	0	00	0
1	1	0	0	1	1	1	1	0	1	0	0	11	0	01	1	XX	0	0	0	00	0
1	1	0	1	0	0	1	1	1	0	0	0	01	0	10	1	XX	0	0	0	00	0
1	1	0	1	0	1	1	1	1	0	0	0	01	0	10	1	XX	0	0	0	00	0
1	1	0	1	1	0	1	1	1	0	0	0	01	0	10	1	XX	0	0	0	00	0
1	1	0	1	1	1	1	1	1	0	0	0	01	0	10	1	XX	0	0	0	00	0
1	1	1	0	0	0	0	1	0	0	0	0	10	0	11	1	XX	0	0	0	00	0
1	1	1	0	0	1	0	1	0	0	0	0	10	0	11	1	XX	0	0	0	00	0
1	1	1	0	1	0	0	1	0	0	0	0	10	0	11	1	XX	0	0	0	00	0
1	1	1	0	1	1	0	1	0	0	0	0	10	0	11	1	XX	0	0	0	00	0
1	1	1	1	0	0	0	0	0	0	X	1	XX	1	XX	1	XX	X	X	X	XX	0
1	1	1	1	0	1	0	0	0	0	X	1	XX	1	XX	1	XX	X	X	X	XX	0
1	1	1	1	1	0	0	0	0	0	X	1	XX	1	XX	1	XX	X	X	X	XX	0
1	1	1	1	1	1	0	0	0	0	X	1	XX	1	XX	1	XX	X	X	X	XX	0

EXEMPLO 2: Para o fluxo de dados complexo, gerar a tabela de estados presentes e futuro, saída e palavra de controle, para cálculo fatorial de um número N de entrada.

N = 5 ! = 1x2x3x4x5 = 120

Como o exemplo mostra que para cálculo do N! é necessário um operador aritmético de multiplicação e o fluxo de dados complexo não possui este recurso para multiplicação direta entre números, a solução é substituir a multiplicação por operações por somas sucessivas. O número **n** é a entrada N! e o contador **m** é instalado para administrar o número de vezes de cada operação de multiplicação é realizada e a locação **SUM** armazena os resultados temporários. Ainda foi criada a variável temporária **K** que tem a finalidade de armazenar a constante que deverá ser somada aos resultados em **SUM** a cada vez que o programa contar. A seguir é apresentado o fluxograma do algoritmo para o cálculo de N !.

FLUXOGRAMA DO ALGORÍTMO PARA CÁLCULO DE N!



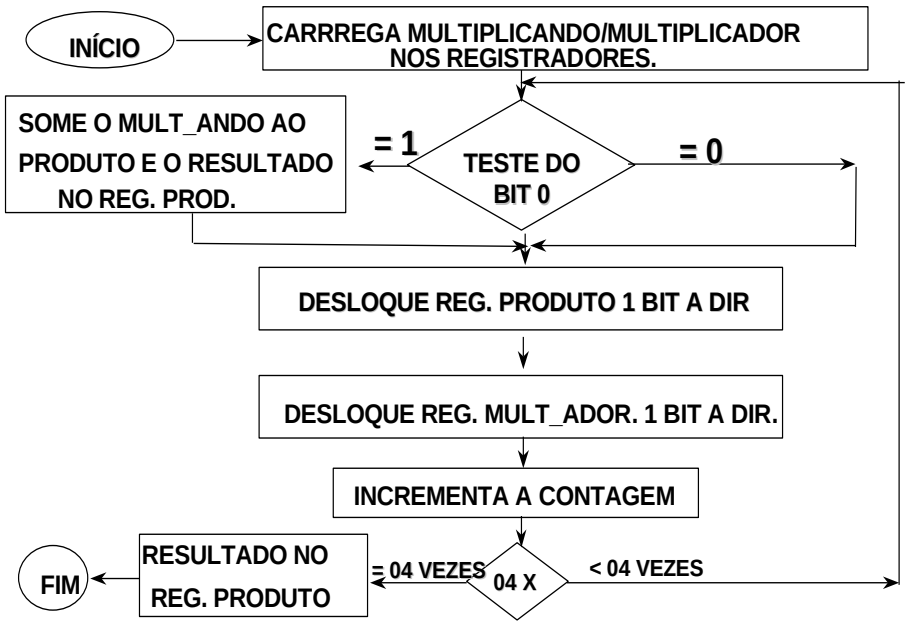
A seguir apresentamos a tabela de estados, presentes e palavra de controle para o algoritmo computacional N!.

b) Tabela de estados, presentes, futuros saída e palavra de controle.

Palavras de Controle	Instruções	IE	WE	WAA	RAE	RAA	RBE	RBB	ULA	SH	OE
01	SUM = 0	0	0	00	0	00	0	00	101	00	0
02	INPUT n	1	0	01	1	XX	1	XX	XXX	XX	0
03	K = n	0	0	11	0	01	1	XX	000	00	0
04	m = n – 1	0	0	10	0	01	1	XX	111	00	0
05	SUM = SUM + K	0	0	00	0	00	0	11	100	00	0
06	m = m –1	0	0	10	0	10	1	XX	111	00	0
07	n = n –1	0	0	01	0	01	1	XX	111	00	0
08	K = SUM	0	0	11	0	00	1	XX	000	00	0
09	m = n –1	0	0	10	0	01	1	XX	111	00	0
10	Display	X	1	XX	0	00	1	XX	111	00	1

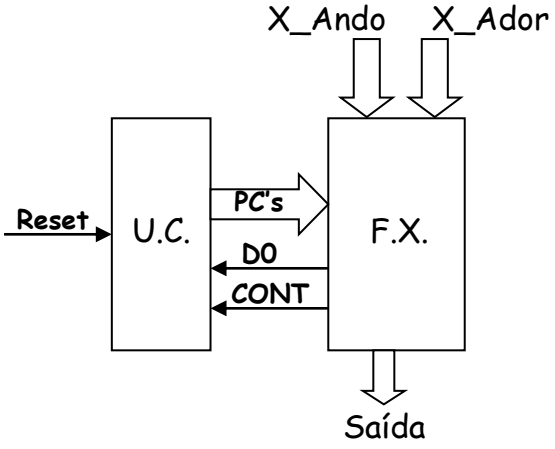
VI - EXERCÍCIOS PROPOSTOS

EXERCÍCIO1 - Dado o algoritmo abaixo, realizar a simulação dos números Multiplicando = (+ 6) e Multiplicador (+ 5). Apresentar toda a sequência do algoritmo.



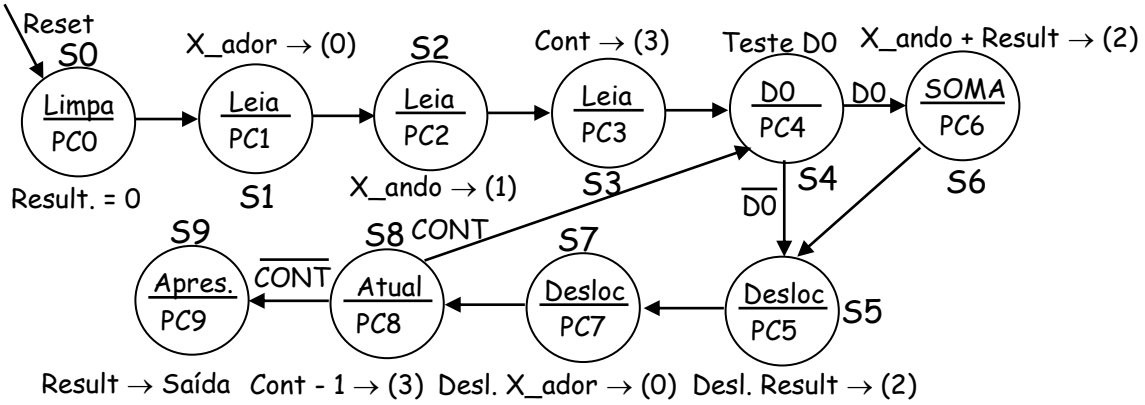
Multiplicando	Multiplicador	Produto	Bit(0)	Contagem
0111	0101	0000 0000	1	4
0111	0010	0011 1000	0	3
0111	0001	0001 1100	1	2
0111	0000	0100 0110	0	1
0111	0000	0010 0011	0	0

Representação esquemática do multiplicador.



Projeto da U.C.

a) Modelo de descrição do sistema por diagrama de estados.



S0 = 0000, S1 = 0001, S2 = 0010, S3 = 0011, S4 = 0100, S5 = 0101, S6 = 0110, S7 = 0111, S8 = 1000 e S9 = 1001.

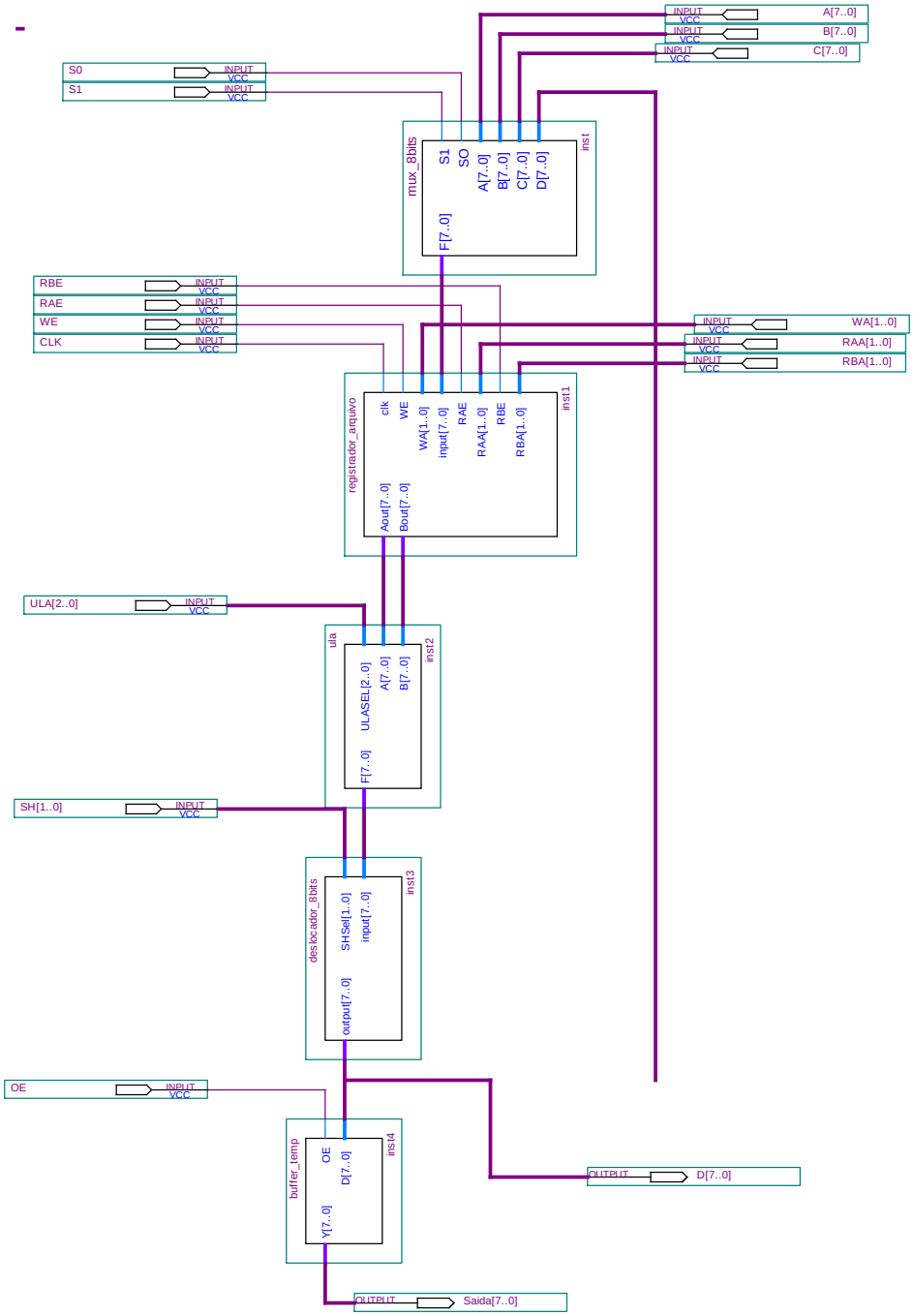
b) Tabela de estados e saída.

Estado	Q_3	Q_2	Q_1	Q_0	00	01	10	11	PC
S0	0	0	0	0	0001	0001	0001	0001	0
S1	0	0	0	1	0010	0010	0010	0010	1
S2	0	0	1	0	0011	0011	0011	0011	2
S3	0	0	1	1	0100	0100	0100	0100	3
S4	0	1	0	0	0101	0101	0110	0110	4
S5	0	1	0	1	0111	0111	0111	0111	5
S6	0	1	1	0	0101	0101	0101	0101	6
S7	0	1	1	1	1000	1000	1000	1000	7
S8	1	0	0	0	1001	0100	1001	0100	8
S9	1	0	0	1	1001	1001	1001	1001	9

Quadro de instrução

Item	Instrução	$I_{E1,0}$	W_E	$W_{A1,0}$	R_{AE}	$RAA_{1,0}$	R_{BE}	$RAB_{1,0}$	$ULA_{2..0}$	$SH_{1,0}$	OE	PC	Hex
1	Limpa Result.	11	1	10	1	10	1	10	101	00	0	0	1EDA8
2	Leia X_Ador	00	1	00	0	xx	0	xx	xxx	xx	0	1	04000
3	Leia X_Ando	01	1	01	0	xx	0	xx	xxx	xx	0	2	0D000
4	Leia Cont	10	1	11	0	xx	0	xx	xxx	xx	0	3	17000
5	Teste D0	xx	0	xx	1	00	0	xx	000	00	0	4	00800
6	Desloc.Result.	11	1	10	1	10	0	xx	000	10	0	5	1EC04
7	Soma	11	1	10	1	10	1	01	100	00	0	6	1ED60
8	Desloc.X_Ador	11	1	00	1	00	0	xx	000	10	0	7	1C804
9	Atualiza	11	1	11	1	11	0	xx	111	00	0	8	1FE38
10	Apresenta	xx	0	xx	1	10	0	xx	000	00	1	9	00C01

Arquitetura fluxo de dados.



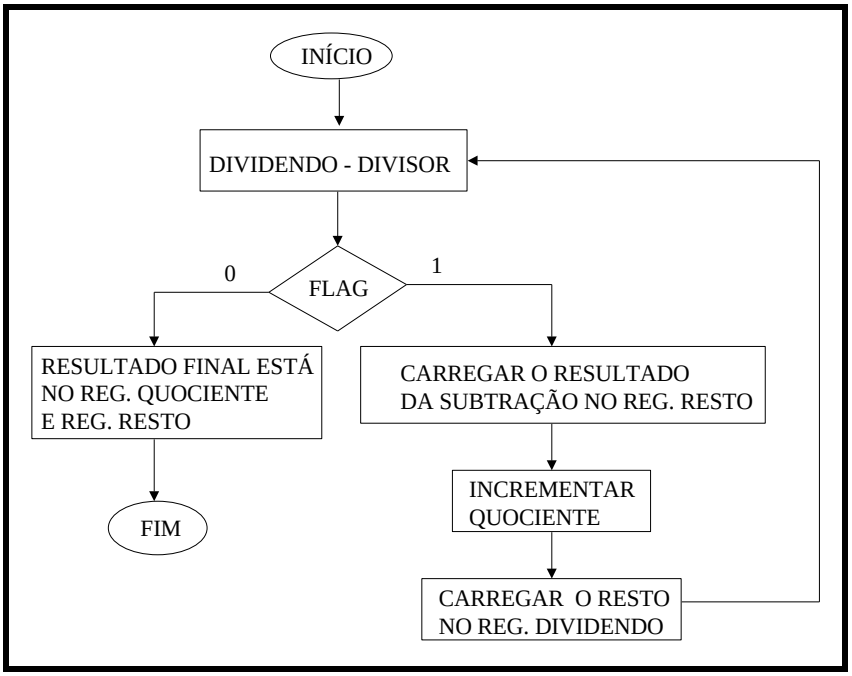
Unidade de Controle.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY uc_multiplica IS
	PORT(
		clkuc		: IN std_logic;
		reset		: IN std_logic;
		d0,cont			: IN std_logic;
		B			: OUT std_logic_vector(16 downto 0);
		q			: OUT std_logic_vector(3 downto 0));
END uc_multiplica;

ARCHITECTURE Behavioral OF uc_multiplica IS
	TYPE maq_estado IS (limpa, Xador, Xando, Contagem, teste, desloca_result, soma,
desloca_Xador, atualiza, apresenta);
	SIGNAL state: maq_estado;
BEGIN
	PROCESS (clkuc)
	BEGIN
		IF reset = '0' THEN
					state <= limpa;
		ELSIF clkuc'EVENT AND clkuc = '1' THEN
			CASE state IS
				WHEN limpa =>
					state <= Xador;

				WHEN Xador =>
					state <= Xando;

				WHEN Xando =>
					state <= Contagem;

				WHEN Contagem =>
					state <= teste;

				WHEN teste =>
					IF D0 = '1' THEN
							state <= soma;
					ELSE
							state <= desloca_result;
					END IF;
```

```
                    WHEN soma =>
                        state <= desloca_result;

                    WHEN desloca_result =>
                        state <= desloca_Xador;

                    WHEN desloca_Xador =>
                        state <= atualiza;

                    WHEN atualiza =>
                        IF (cont = '1') THEN
                            state <= teste;
                            ELSE
                            state <= apresenta;
                            END IF;

                    WHEN apresenta =>
                        state <= apresenta;
            END CASE;
            END IF;
                CASE state IS
WHEN limpa =>
B(16)<='1';B(15)<='1';B(14)<='1';B(13)<='1';B(12)<='0';B(11)<='1';B(10)<='1';B(9)<='0';B(8)<='1';
B(7)<='1';B(6)<='0';B(5)<='1';B(4)<='0';B(3)<='1';B(2)<='0';B(1)<='0'; B(0)<='0';

WHEN Xador =>
B(16)<='0';B(15)<='0';B(14)<='1';B(13)<='0';B(12)<='0';B(11)<='0';B(10)<='0';B(9)<='0';B(8)<='0'
;B(7)<='0';B(6)<='0';B(5)<='0';B(4)<='0';B(3)<='0';B(2)<='0';B(1)<='0'; B(0)<='0';

WHEN Xando =>
B(16)<='0';B(15)<='1';B(14)<='1';B(13)<='0';B(12)<='1';B(11)<='0';B(10)<='0';B(9)<='0';B(8)<='0';
B(7)<='0';B(6)<='0';B(5)<='0';B(4)<='0';B(3)<='0';B(2)<='0';B(1)<='0'; B(0)<='0';

WHEN Contagem  =>
B(16)<='1';B(15)<='0';B(14)<='1';B(13)<='1';B(12)<='1';B(11)<='0';B(10)<='0';B(9)<='0';B(8)<='0';
B(7)<='0';B(6)<='0';B(5)<='0';B(4)<='0';B(3)<='0';B(2)<='0';B(1)<='0';B(0)<='0';

WHEN teste=>
B(16)<='0';B(15)<='0';B(14)<='0';B(13)<='0';B(12)<='0';B(11)<='1';B(10)<='0';B(9)<='0';B(8)<='0'
;B(7)<='0';B(6)<='0';B(5)<='0';B(4)<='0';B(3)<='0';B(2)<='0';B(1)<='0';B(0)<='0';
```

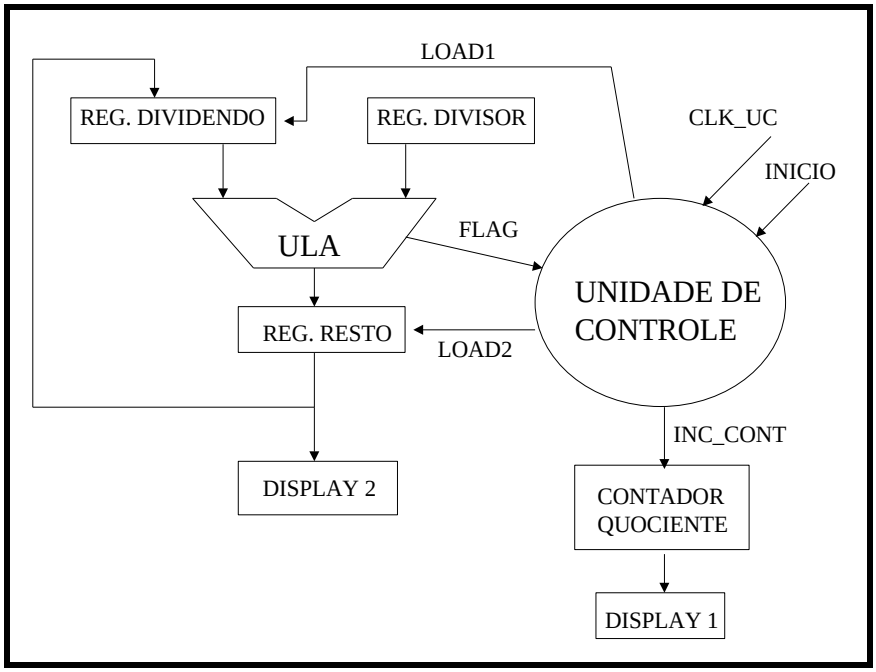
```
WHEN desloca_result =>

B(16)<='1';B(15)<='1';B(14)<='1';B(13)<='1';B(12)<='0';B(11)<='1';B(10)<='1';B(9)<='0';B(8)<='0';
B(7)<='0';B(6)<='0';B(5)<='0';B(4)<='0';B(3)<='0';B(2)<='1';B(1)<='0';B(0)<='0';

WHEN soma  =>
B(16)<='1';B(15)<='1';B(14)<='1';B(13)<='1';B(12)<='0';B(11)<='1';B(10)<='1';B(9)<='0';B(8)<='1';B
(7)<='0';B(6)<='1';B(5)<='1';B(4)<='0';B(3)<='0';B(2)<='0';B(1)<='0';B(0)<='0';

WHEN desloca_Xador =>

B(16)<='1';B(15)<='1';B(14)<='1';B(13)<='0';B(12)<='0';B(11)<='1';B(10)<='0';B(9)<='0';B(8)<='0';
B(7)<='0';B(6)<='0';B(5)<='0';B(4)<='0';B(3)<='0';B(2)<='1';B(1)<='0';B(0)<='0';

WHEN atualiza =>
B(16)<='1';B(15)<='1';B(14)<='1';B(13)<='1';B(12)<='1';B(11)<='1';B(10)<='1';B(9)<='1';B(8)<='0';B
(7)<='0';B(6)<='0';B(5)<='1';B(4)<='1';B(3)<='1';B(2)<='0';B(1)<='0';B(0)<='0';

WHEN apresenta  =>
B(16)<='0';B(15)<='0';B(14)<='0';B(13)<='0';B(12)<='0';B(11)<='1';B(10)<='1';B(9)<='0';B(8)<='0'
;B(7)<='0';B(6)<='0';B(5)<='0';B(4)<='0';B(3)<='0';B(2)<='0';B(1)<='0';B(0)<='1';

END CASE;
        END PROCESS;
                        WITH state SELECT
                        q <="0000" WHEN limpa,
                                "0001" WHEN Xador,
                                "0010" WHEN Xando,
                                "0011" WHEN Contagem,
                                "0100" WHEN teste,
                                "0101" WHEN desloca_result,
                                "0110" WHEN soma,
                                "0111" WHEN desloca_Xador,
                                "1000" WHEN atualiza,
                                "1001" WHEN apresenta;
END Behavioral;
```

EXERCÍCIO4 - O algoritmo descrito no diagrama define as etapas para a DIVISÃO entre dois números de quatro bits. Supondo que o dividendo e o divisor já estão inicialmente carregados nos respectivos registradores, e que em função do resultado da subtração inicial gera-se um FLAG de controle tal que, FLAG=1 se dividendo >= divisor e FLAG=0 se dividendo < divisor, pede-se simular a divisão de 13 por 3, preenchendo a tabela abaixo em binário:



DIVIDENDO	DIVISOR	FLAG	QUOCIENTE	RESTO

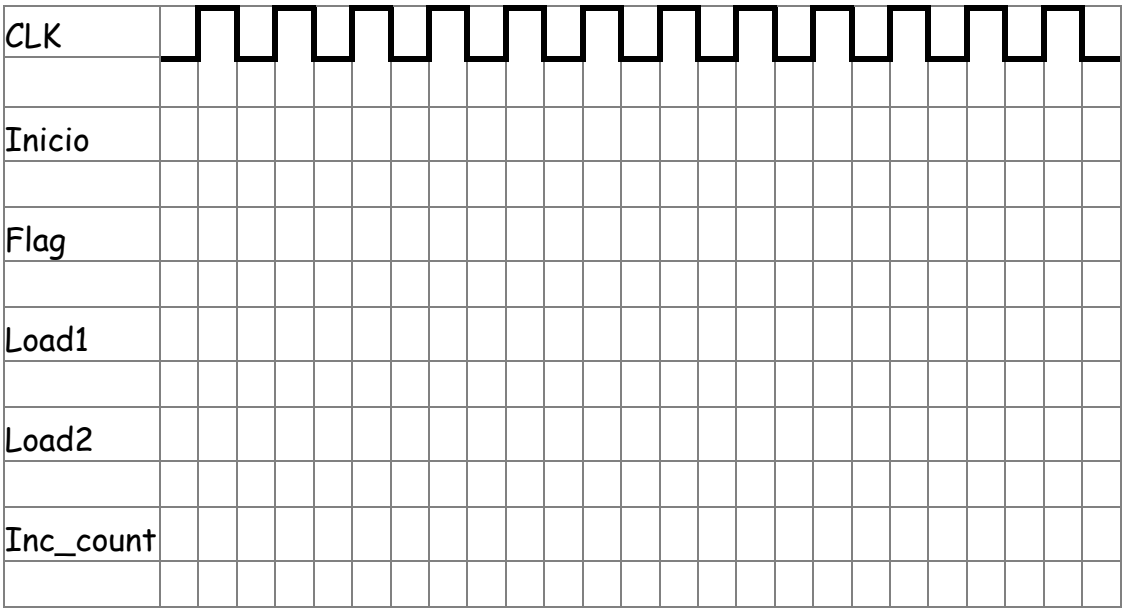
EXERCÍCIO5 - Dado o fluxo de dados descrito no diagrama, defina as palavras de controle necessárias para executar o algoritmo do exercício 4 e determine as formas de onda das variáveis de controle para a divisão de 13 por 3.
Considerar que a unidade de controle evolui na borda de subida do sinal de clock, que o processo de escrita nos registradores é sincronizado com a borda de descida e que os demais componentes são assíncronos.



(a) Palavras de Controle

		Load1	Load2	Inc_count
1				
2				
3				
4				
5				

(b) Carta de Tempos



EXERCÍCIO6 – Implemente o sistema descrito nos exercícios 4 e 5 através de uma rede de Petri.

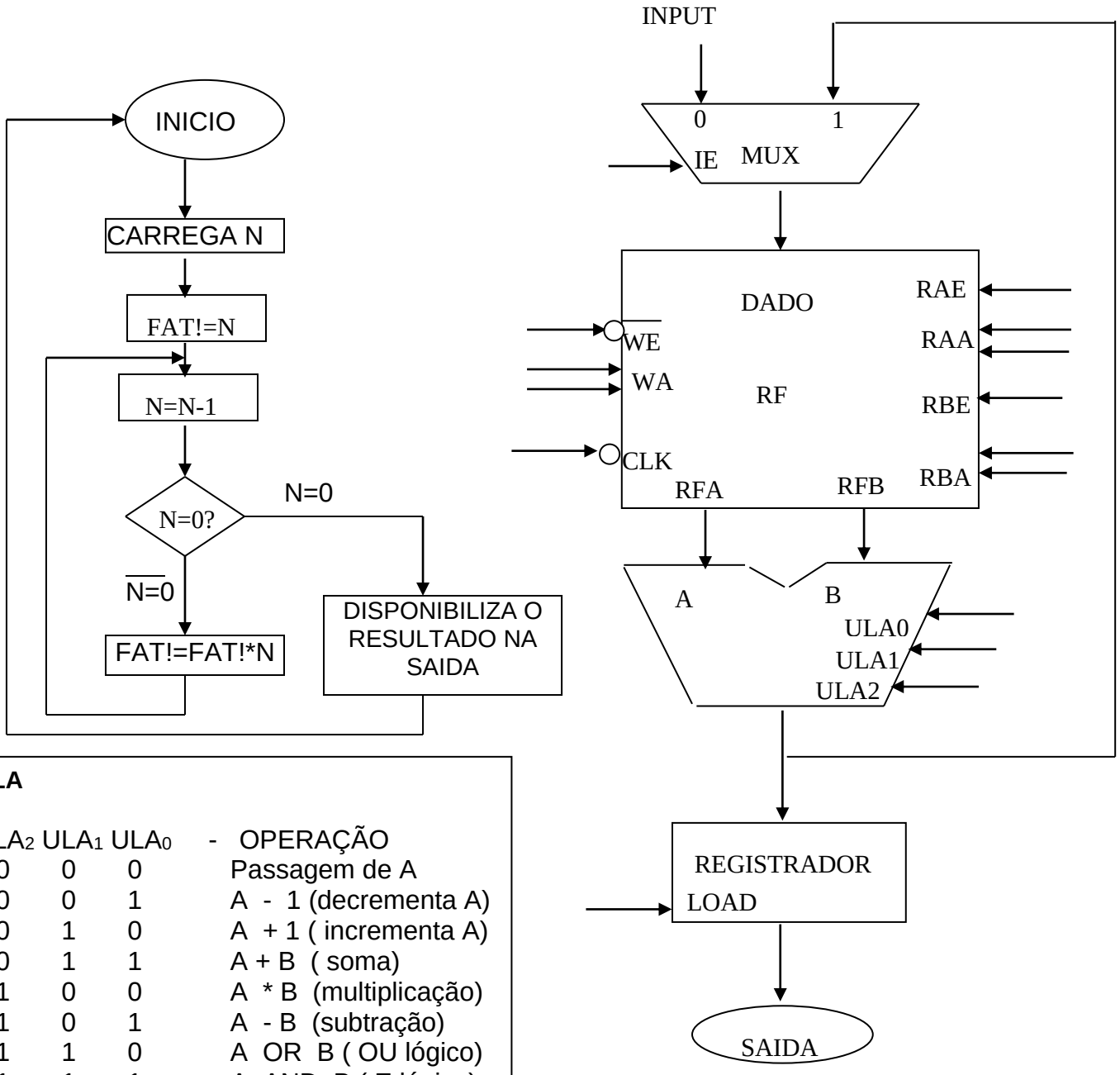
EXERCÍCIO7 : O Algoritmo descrito a seguir pode ser utilizado no cálculo do fatorial de um número N (sendo N>0), lembrando que N!=N*N-1*N-2*N-3.... e assim sucessivamente, até que o valor do multiplicador seja igual a 1. É dado também um fluxo de dados capaz de executar as operações descritas no algoritmo. Analise o procedimento em conjunto com o fluxo de dados e determine:

(a) As palavras de controle necessárias para a execução de cada um dos procedimentos descritos no algoritmo
(b) A carta de tempos do processo, simulando a execução de 3! (fatorial de 3)
(c) A Unidade de Controle do sistema através de uma rede de Petri para gerar essas palavras de controle, formando assim o sistema digital completo

Considerar que a rede de Petri evolui na borda de subida do sinal de clock e que o processo de escrita no registrador de arquivos (RF) é sincronizado com a borda de descida. Os demais componentes e processos são assíncronos. É possível também habilitar simultaneamente a leitura em ambos os ports do registrador de arquivo (RF). Todas as vias de dados são de 8 bits.

LEGENDA

IE – Seleciona entrada do MUX	RAE – Habilita leitura do port A
WE – Habilita escrita no Registrador de Arquivo (RF)	RBE – Habilita leitura do port B
WA –Endereço de escrita no RF	RAA – Endereço de leitura do port A
CLK – clock	RBA – Endereço de leitura do port B
ULA_2, ULA_1, ULA_0 – Seleção da operação da ULA	RFA – saída do port A
LOAD – carrega valor no registrador	RFB – Saída do port B



ULA

ULA_2	ULA_1	ULA_0	- OPERAÇÃO
0	0	0	Passagem de A
0	0	1	A - 1 (decrementa A)
0	1	0	A + 1 (incrementa A)
0	1	1	A + B (soma)
1	0	0	A * B (multiplicação)
1	0	1	A - B (subtração)
1	1	0	A OR B (OU lógico)
1	1	1	A AND B (E lógico)

(a) Palavras de Controle

		IE	$\overline{WE}$	$WA_{1,0}$	RAE	$RAA_{1,0}$	RBE	$RBA_{1,0}$	$ULA_{2,1,0}$	Load
1	Carrega N									
2	FAT!=N									
3	N=N-1									
4	FAT!=FAT!*N									
5	Saída									

(b) Carta de Tempos

c) Rede de Petri

EXERCÍCIO8 – Implemente o sistema descrito no exemplo 1 através de uma rede de Petri.

EXERCÍCIO9 – Implemente o sistema descrito no exemplo 2 através de uma rede de Petri.

MÁQUINAS DE USO GERAL

Introdução: A diferença básica entre uma máquina de uso específico e uma máquina de uso geral está na unidade de controle. Para a máquina de uso específico o fluxo de dados de acordo com a aplicação é projetada a unidade de controle para o uso específico da aplicação e a máquina se torna específica. Para a máquina de uso geral o fluxo de dados e a unidade de controle não alteram com a aplicação e não é preciso projetar a unidade de controle. A diferença básica é que na máquina de uso geral a solução de uma aplicação é feita por instrução a instrução até que a máquina complete todo o programa da aplicação. O usuário transforma o quadro de instrução a qual ele monta para a máquina de uso específico em linhas de programa com instruções que serão executadas passo a passo pela unidade de controle no fluxo de dados.

UNIDADE DE CONTROLE

ARQUITETURA DA UNIDADE DE CONTROLE

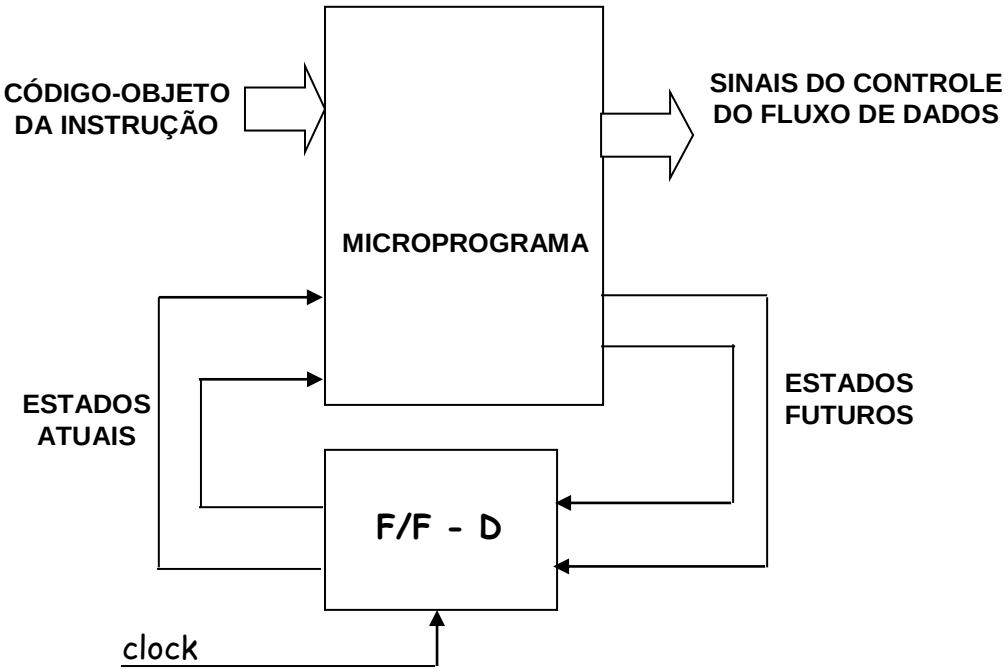
Conforme o diagrama a seguir a U.C. é responsável pela busca da instrução e a geração dos sinais de controle do fluxo de dados. São dois ciclos que a U.C. executa.
1) Ciclo de Busca;
2) Ciclo de Execução.

1.a) CICLO DE BUSCA – O ciclo de busca de uma instrução alocada na memória externa é realizado através do registrador contador de instruções denominado de PC. O conteúdo do PC é o endereço da instrução. O ciclo de busca é dividido em 03 microoperações a saber :

1) O PC endereça a instrução na memória ***Drive do endereço = (PC)***
2) O PC é incrementado para a próxima busca ***PC= PC +1***
3) A instrução é carregada no registrador de instrução ***RI = Instrução***

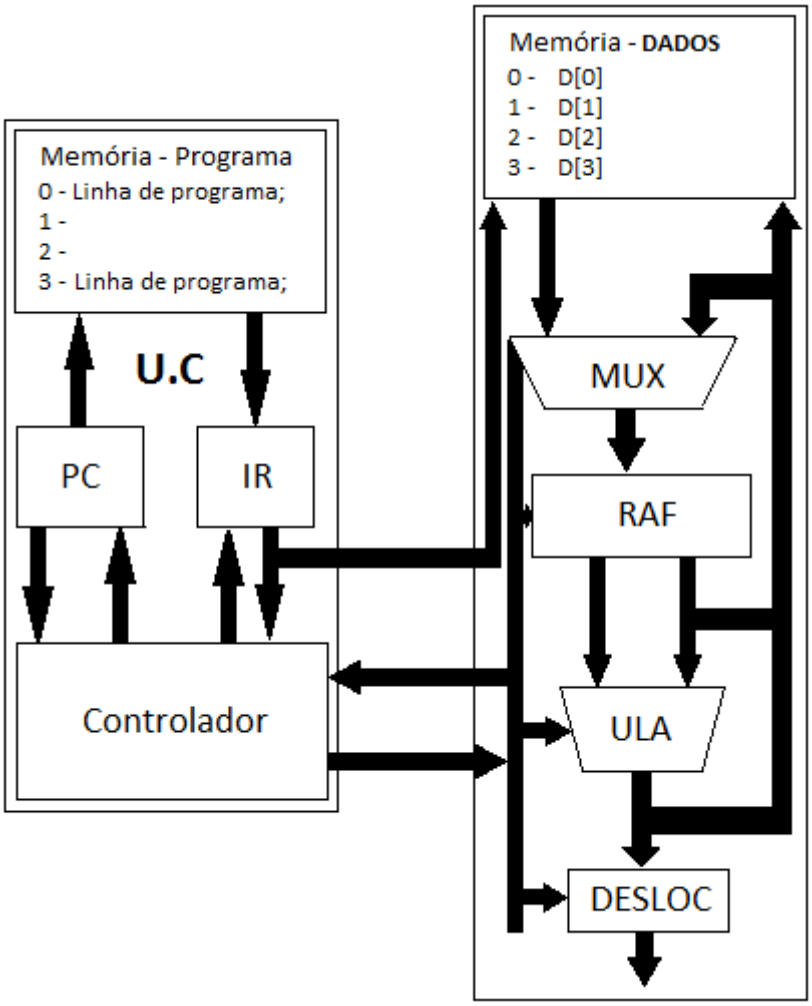
1.b) CICLO DE EXECUÇÃO – O ciclo de execução da instrução, segue arquitetura RISC (Reduzido , onde cada execução é realizada por uma única microoperação. Cada instrução se torna desta forma uma microoperação.
As microoperações das instruções são armazenadas em um tabela de dados e a seqüência de busca e execução da unidade de controle é controlada por um microprograma operando como uma máquina de estados, cuja entrada externa é identificada pelo código objeto da instrução, cuja entrada atual (referida aos estados internos do sistema) são poucos e no máximo 3 estados e cuja saída são os sinais necessários para o fluxo de dados realizar a execução da

instrução em andamento. O diagrama de blocos a seguir mostra um esquema de representação por máquina de estados da unidade de controle.



Arquitetura em bloco da unidade de controle e fluxo de dados.

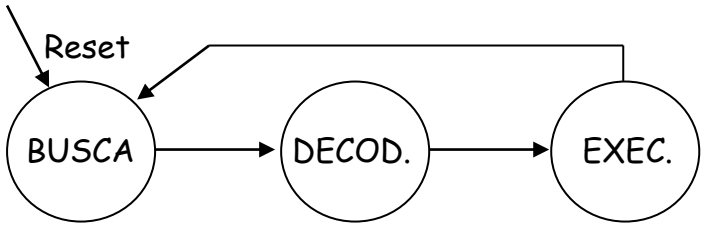
A seguir é apresentada a arquitetura da unidade de controle e o fluxo de dados.



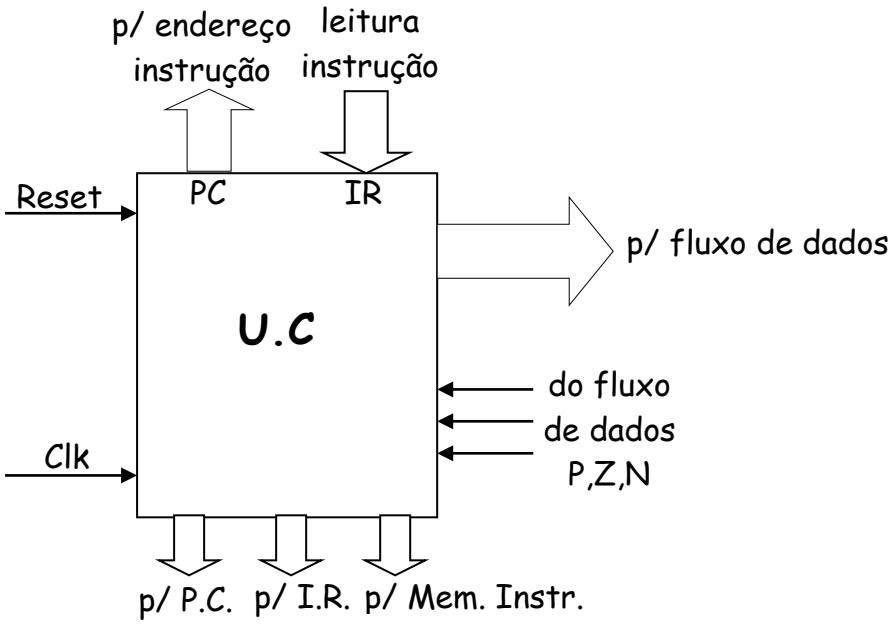
Projeto de uma máquina com 8 instruções utilizando a arquitetura acima da unidade de controle e do fluxo de dados.

Diagrama de estados do controlador.

A seguir é apresentado o diagrama de estados do ciclo de busca da máquina, a qual consiste na busca da instrução, na decodificação da instrução e na execução da instrução.



Projeto da U.C.

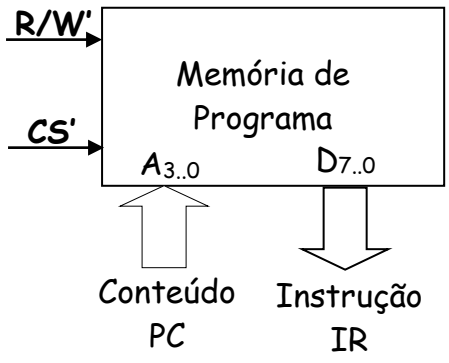


Sinais recebidos e gerados na U.C.

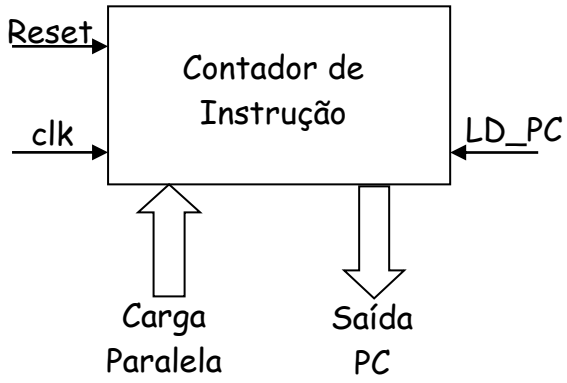
a) Recebidos – P,Z,N, clk, reset, data_Mem (dados da memória de programa).

b) Gerados – $I_{E1.0}$, W_E, $W_{A1..0}$, R_{AE}, $R_{AA1..0}$, $R_{BA1..0}$, $ULA_{2..0}$, $SH_{1..0}$, OE, data PC, data IR, R/W', CS', LD_PC, LD_IR.

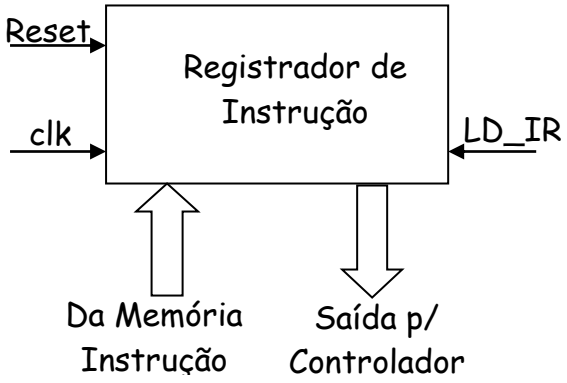
Memória de instruções – Uma memória de 16 x 8 do tipo RAM, sendo 4 bits de endereço por 8 bits de conteúdo. O controle da memória é feito por 2 sinais, sendo um de leitura e escrita e o outro de seleção do dispositivo.



Contador de instrução – É um dispositivo contador síncrono de 4 bits com clock e reset e carga paralela síncrona.



Registrador de instrução – É um dispositivo tipo latch com 8 bits com carga paralela o qual recebe uma instrução da memória de instruções para ser decodificada e executada a seguir.



Conjunto de instruções da máquina.

Instrução	Opcode	Comentário
MOV Reg, (Ad)	000xx	Move o conteúdo da memória de dados para um registrador
ADD Reg, Reg	001xx	Adiciona o conteúdo de um regist. xx para outro registrador
SUB Reg, Reg	010xx	Subtrai o conteúdo de um regist. xx para outro registrador
IN Reg	011xx	Entrada de byte para registrador xx
OUT Reg	100xx	Saída de byte para registrador xx
INC Reg	101xx	Incrementa registrador xx
DEC Reg	110xx	Decrementa registrador xx
JMP cond	111xx	Salta para endereço relativo xx

Codificação de cada instrução

Opcode	Mnemonico	Operação
00000	LD A, d	A ← d
01	LD B, d	B ← d
10	LD C, d	C ← d
11	LD D, d	D ← d
00100	ADD A,A	A ← A + A
01	ADD A, B	A ← A + B
10	ADD A, C	A ← A + C
11	ADD, D	A ← A + D
01000	SUB A, A	A ← A - A
01	SUB A, B	A ← A – B
10	SUB A, C	A ← A - C
11	SUB A, D	A ← A – D
01100	IN A	A ← d
01	IN B	B ← d
10	IN C	C ← d
11	IN D	D ← d
10000	OUT A	Saída ← A
01	OUT B	Saída ← B
10	OUT C	Saída ← C
11	OUT D	Saída ← D
10100	INC A	A ← A + 1
01	INC B	B ← A + 1
10	INC C	C ← A + 1
11	INC D	D ← A + 1
11000	DEC A	A ← A - 1
01	DEC B	B ← A - 1
10	DEC C	C ← A - 1
11	DEC D	D ← A - 1
11100	JZ	Ad ← (dig)
01	JP	Ad ← (dig)
10	JN	Ad ← (DIG)
11	JNZ	Ad ← (DIG)

Proejto da U.C.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY uc_uc_micro IS
	PORT(
		clkuc		: IN std_logic;
		reset		: IN std_logic;
		B			: OUT std_logic_vector(16 downto 0);
		q			: OUT std_logic_vector(1 downto 0));
END uc_micro;
```

```
ARCHITECTURE Behavioral OF uc_micro IS
	TYPE maq_estado IS (busca, decodifica, executa);
	SIGNAL state: maq_estado;
BEGIN
	PROCESS (clkuc)
	BEGIN
		IF reset = '0' THEN
					state <= limpa;
		ELSIF clkuc'EVENT AND clkuc = '1' THEN
			CASE state IS
				WHEN busca =>
					state <= decodifica;

				WHEN decodifica =>
					state <= executa;

				WHEN executa =>
					state <= busca;
END CASE;
		END IF;
			CASE state IS
WHEN busca =>
B(16)<='1';B(15)<='1';B(14)<='1';B(13)<='1';B(12)<='0';B(11)<='1';B(10)<='1';B(9)<='0';B(8)<='1';
B(7)<='1';B(6)<='0';B(5)<='1';B(4)<='0';B(3)<='1';B(2)<='0';B(1)<='0'; B(0)<='0';

WHEN decodifica =>
B(16)<='0';B(15)<='0';B(14)<='1';B(13)<='0';B(12)<='0';B(11)<='0';B(10)<='0';B(9)<='0';B(8)<='0'
;B(7)<='0';B(6)<='0';B(5)<='0';B(4)<='0';B(3)<='0';B(2)<='0';B(1)<='0'; B(0)<='0';

WHEN executa =>
B(16)<='0';B(15)<='1';B(14)<='1';B(13)<='0';B(12)<='1';B(11)<='0';B(10)<='0';B(9)<='0';B(8)<='0';
B(7)<='0';B(6)<='0';B(5)<='0';B(4)<='0';B(3)<='0';B(2)<='0';B(1)<='0'; B(0)<='0';

END CASE;
	END PROCESS;
			WITH state SELECT
			q <="00" WHEN busca,
				"01" WHEN decodifica,
				"10" WHEN executa;
END Behavioral;
```

Projeto do microprograma controlador - As microoperações a serem realizadas na execução de cada instrução são mostradas a seguir.

Instrução MOV Reg,(Ad)	Operações	$muxsel_{1,0}$ 13,12	W_E	$W_{A1..0}$	R_{AE} 9	$R_{AA1,0}$ 8 – 6	R_{BE}	$R_{AB1..0}$	$ALU_{2,1,0}$ 5 – 3	$SH_{1,0}$ 2 – 1	O_E 0	Total Microop.
MOV A,(Ad)	A ← Mem [xxx]											
MOV B,(Ad)	B ← Mem [xxx]											
MOV C,(Ad)	C ← Mem [xxx]											
MOV D,(Ad)	D ← Mem [xxx]											
Instrução ADD A,Reg	Operações	$muxsel_{1,0}$ 13,12	W_E	$W_{A1..0}$	R_{AE} 9	$R_{AA1,0}$ 8 – 6	R_{BE}	$R_{AB1..0}$	$ALU_{2,1,0}$ 5 – 3	$SH_{1,0}$ 2 – 1	O_E 0	Total Microop.
ADD A,A	A ← A + A											
ADD A,B	A ← A + B											
ADD A,C	A ← A + C											
ADD A,D	A ← A + D											
Instrução SUB A,Reg	Operações	$muxsel_{1,0}$ 13,12	W_E	$W_{A1..0}$	R_{AE} 9	$R_{AA1,0}$ 8 – 6	R_{BE}	$R_{AB1..0}$	$ALU_{2,1,0}$ 5 – 3	$SH_{1,0}$ 2 – 1	O_E 0	Total Microop.
SUB A,A	A ← A – A											
SUB A,B	A ← A – B											
SUB A,C	A ← A – C											
SUB A,D	A ← A – D											
Instrução IN Reg	Operações	$muxsel_{1,0}$ 13,12	W_E	$W_{A1..0}$	R_{AE} 9	$R_{AA1,0}$ 8 – 6	R_{BE}	$R_{AB1..0}$	$ALU_{2,1,0}$ 5 – 3	$SH_{1,0}$ 2 – 1	O_E 0	Total Microop.
IN A	A ← d											
IN B	B ← d											
IN C	C ← d											
IN D	D ← d											
Instrução OUT Reg	Operações	$muxsel_{1,0}$ 13,12	W_E	$W_{A1..0}$	R_{AE} 9	$R_{AA1,0}$ 8 – 6	R_{BE}	$R_{AB1..0}$	$ALU_{2,1,0}$ 5 – 3	$SH_{1,0}$ 2 – 1	O_E 0	Total Microop.
OUT A	Saída ← A											
OUT B	Saída ← B											
OUT C	Saída ← C											
OUT D	Saída ← D											

Instrução INC Reg	Operações	$Muxsel_{1,0}$ 13,12	W_E	$W_{A1..0}$	R_{AE} 9	$R_{AA1,0}$ 8 – 6	R_{BE}	$R_{AB1..0}$	$ALU_{2,1,0}$ 5 – 3	$SH_{1,0}$ 2 – 1	O_E 0	Total Microop.
INC A	A ← A + 1											
INC B	B ← B + 1											
INC C	C ← C + 1											
INC D	D ← D + 1											
DEC A	A ← A + 1											
INC B	B ← B + 1											
INC C	C ← C + 1											
INC D	D ← D + 1											
Instrução Jump cond	Operações	$muxsel_{1,0}$ 13,12	W_E	$W_{A1..0}$	R_{AE} 9	$R_{AA1,0}$ 8 – 6	R_{BE}	$R_{AB1..0}$	$ALU_{2,1,0}$ 5 – 3	$SH_{1,0}$ 2 – 1	O_E 0	Total Microop.

JMP	**PC ← Ad**											
JP	**PC ← Ad**											
JN	**PC ← Ad**											
JNZ	**PC ← Ad**											

Salto Condicional P, N e Z.

2 - Projeto de uma máquina com 16 instruções utilizando a arquitetura acima da unidade de controle e do fluxo de dados.

2.1 Diagrama de bloco da máquina completa.

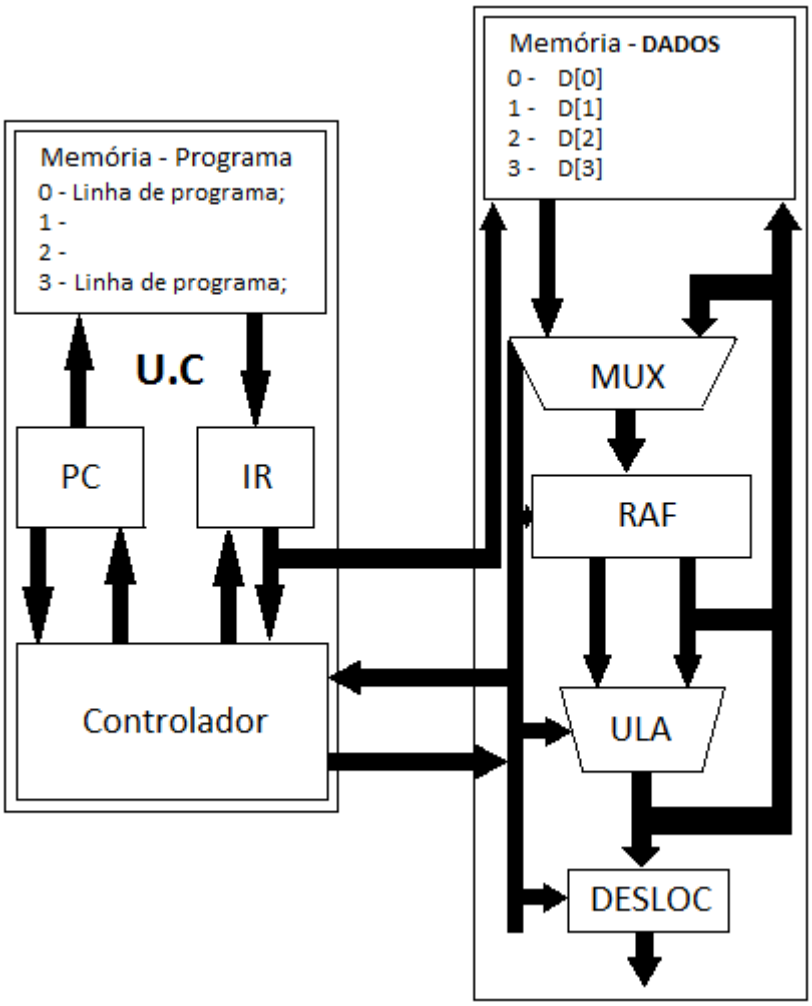
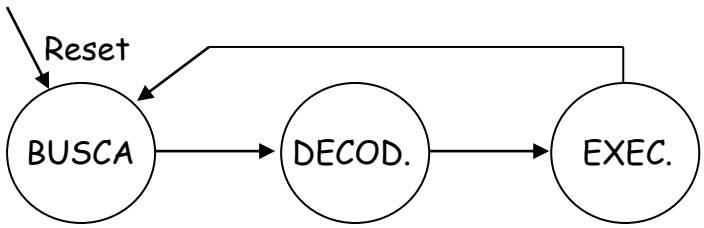
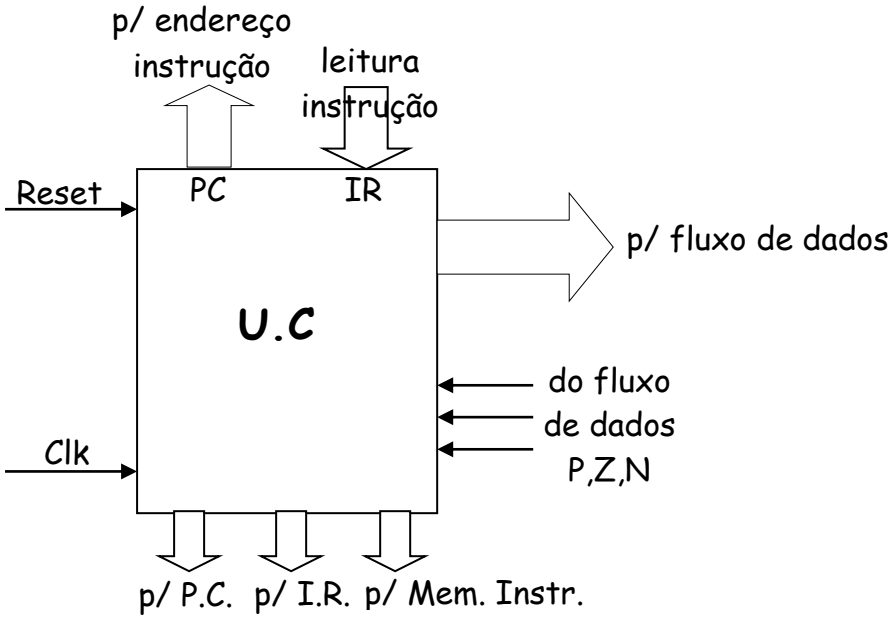


Diagrama de estados do controlador.

A seguir é apresentado o diagrama de estados do ciclo de busca da máquina, a qual consiste na busca da instrução, na decodificação da instrução e na execução da instrução.



Projeto da U.C.

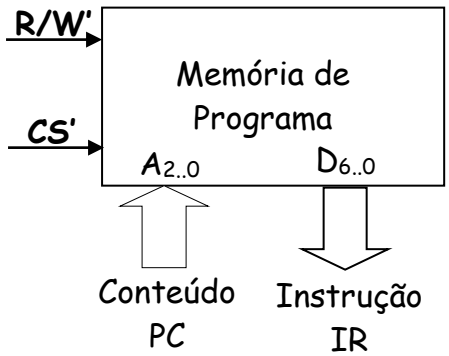


Sinais recebidos e gerados na U.C.

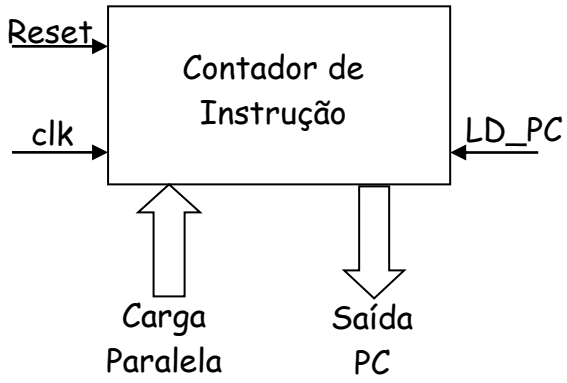
a) Recebidos – P,Z,N, clk, reset, data_Mem (dados da memória de programa).

b) Gerados – $I_{E1.0}$, W_E, $W_{A1..0}$, R_{AE}, $R_{AA1..0}$, $R_{BA1..0}$, $ULA_{2..0}$, $SH_{1..0}$, OE, data PC, data IR, R/W', CS', LD_PC, LD_IR.

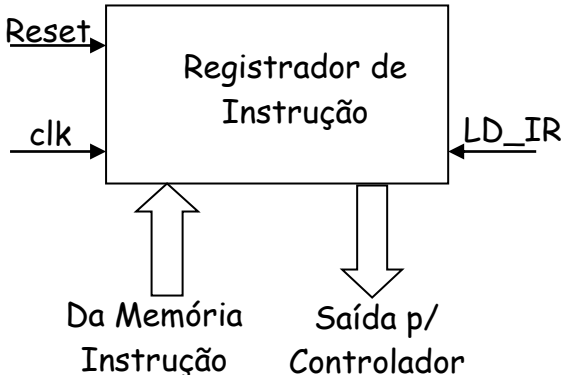
Memória de instruções – Uma memória de 8 x 7 do tipo RAM, sendo 3 bits de endereço por 7 bits de conteúdo. O controle da memória é feito por 2 sinais, sendo um de leitura e escrita e o outro de seleção do dispositivo.



Contador de instrução – É um dispositivo contador síncrono de 3 bits com clock e reset e carga paralela síncrona.



Registrador de instrução – É um dispositivo tipo latch com 7 bits com carga paralela o qual recebe uma instrução da memória de instruções para ser decodificada e executada a seguir.



Conjunto de instruções da máquina.

Instrução	Opcode	Comentário
LDA,(xxx)	0000xxx	Carrega o conteúdo da memória de dados para um registrador
STA,(xxx)	0001xxx	Armazena o conteúdo do registrador para a memória de dados
ADDA,Reg	0010xxx	Adiciona o conteúdo de um regist. xx para outro registrador
SUBA,Reg	0011xxx	Subtrai o conteúdo de um regist. xx para outro registrador
INC Reg	0100xxx	Incrementa registrador
DEC Reg	0101xxx	Decrementa registrador
IN A	0110xxx	Entrada de byte para registrador
OUT A	0111xxx	Saída de byte do registrador
ROT A	1000xxx	Gira o conteúdo do reg .A
SHL A	1001xxx	Desloca a esquerda o registrador
SHR A	1010xxx	Desloca a direita o registrador
JP	1011xxx	Salta se positivo para endereço dado por xxx
JNZ	1100xxx	Salta se não zero para endereço dado por xxx
JZ	1101xxx	Salta se zero para endereço dado por xxx
JN	1110xxx	Salta se negativo para endereço dado por xxx
JMP	1111xxx	Salta incondicional para endereço dado por xxx

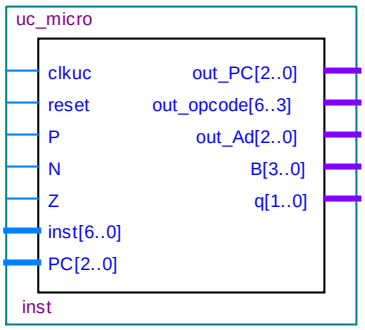
Codificação de cada instrução

Opcode	Mnemonico	Operação	Opcode	Mnemonico	Operação
0000000	LDA, (Ad)	A ← (d)	**0011**000	SUB A, A	A ← A - A
001	LDB, (Ad)	B ← (d)	001	SUB A, B	A ← A – B
010	LDC, (Ad)	C ← (d)	010	SUB A, C	A ← A - C
011	LDD, (Ad)	D ← (d)	011	SUB A, D	A ← A – D
100	LDE, (Ad)	E ← (d)	100	SUB A, A	A ← A – E
101	LDF, (Ad)	F ← (d)	101	SUB A, A	A ← A – F
110	LDG, (Ad)	G ← (d)	110	SUB A, A	A ← A – G
111	LDH, (Ad)	H ← (d)	111	SUB A, A	A ← A – H
0001000	STA,(Ad)	(Ad) ← A	**0100**000	IN A	A ← A + 1
001	STB,(Ad)	(Ad) ← B	01	IN B	B ← B + 1
010	STC,(Ad)	(Ad) ← C	10	IN C	C ← C + 1
011	STD,(Ad)	(Ad) ← D	011	IN D	D ← D + 1
100	STE,(Ad)	(Ad) ← E	100	IN E	E ← E + 1
101	STF,(Ad)	(Ad) ← F	101	IN F	F ← F + 1
110	STAG,(Ad)	(Ad) ← G	110	IN G	G ← G + 1
111	STAH,(Ad)	(Ad) ← H	111	IN H	H ← H + 1
0010000	ADDA,A	A ← A + A	**0101**000	DEC A	A ← A - 1
001	ADDA,B	A ← A + B	001	DEC B	B ← A - 1
010	ADDA,C	A ← A + C	010	DEC C	C ← A - 1
011	ADDA,D	A ← A + D	011	DEC D	D ← A - 1
100	ADDA,E	A ← A + E	100	DEC E	E ← E - 1
101	ADDA,F	A ← A + F	101	DEC F	F ← F - 1
110	ADDA,G	A ← A + G	110	DEC G	G ← G - 1
111	ADDA,H	A ← A + H	111	DEC H	H ← H - 1
Opcode	Mnemonico	Operação	Opcode	Mnemonico	Operação
0110000	IN A	A ← inp	**1001**000	SHLA	A ← Bit 0
001	IN B	B ← inp	001	SHLB	B ← Bit 0
010	IN C	C ← inp	010	SHLC	C ← Bit 0
011	IN D	D ← inp	011	SHLD	D ← Bit 0
100	IN E	E ← inp	100	SHLE	E ← Bit 0
101	IN F	F ← inp	101	SHLF	F ← Bit 0
110	IN G	G ← inp	110	SHLG	G ← Bit 0
111	IN H	H ← inp	111	SHLH	H ← Bit 0
0111000	OUT A	Saída ← A	**1010**000	SHRA	A ← Bit 7
001	OUT B	Saída ← B	001	SHRB	B ← Bit 7
010	OUT C	Saída ← C	010	SHRC	C ← Bit 7
011	OUT D	Saída ← D	011	SHRD	D ← Bit 7
100	OUT E	Saída ← E	100	SHRE	E ← Bit 7
101	OUT F	Saída ← F	101	SHRF	F ← Bit 7
110	OUT G	Saída ← G	110	SHRG	G ← Bit 7
111	OUT H	Saída ← H	111	SHRH	H ← Bit 7
1000000	ROTA	Bit 7 ← bit 0	**1011**xxx	JP	PC ← xxx
001	ROTB	Bit 7 ← bit 0	**1100**xxx	JNZ	PC ← xxx
010	ROTC	Bit 7 ← bit 0	**1101**xxx	JZ	PC ← xxx
011	ROTD	Bit 7 ← bit 0	**1110**xxx	JN	PC ← xxx
100	ROTE	Bit 7 ← bit 0	**1111**xxx	JMP	PC ← xxx
101	ROTF	Bit 7 ← bit 0	-	-	-
110	ROTG	Bit 7 ← bit 0	-	-	-
111	ROTH	Bit 7 ← bit 0	-	-	-

Carta de microoperações da U.C.

Instrução Busca	Operações	LD_{PC}	R_{UC}	CS_{UC}'	LD_{IR}	Total Microop.
inst,(PC)	inst ← Mem [xxx]	1	1	0	0	1
Instrução Incrementa						
PC = PC + 1	inst ← Mem [xxx]	0	1	0	0	1
Instrução Execução						
IR ← inst	inst ← Mem [xxx]	0	0	1	1	1

Bloco U.C com entrada e saídas.



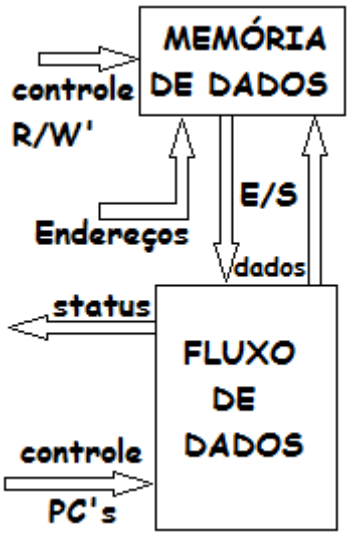
SINAIS DE ENTRADA E SAÍDA DA U.C.

Entradas:
clkuc, reset, P, N, Z, inst[6..0].

Saídas:
PC[2..0], out_opcode[3..0], out_Ad[2..0], B[3..0]

Obs.: out_PC[2..0]. q[1..0] são sinais monitoração.

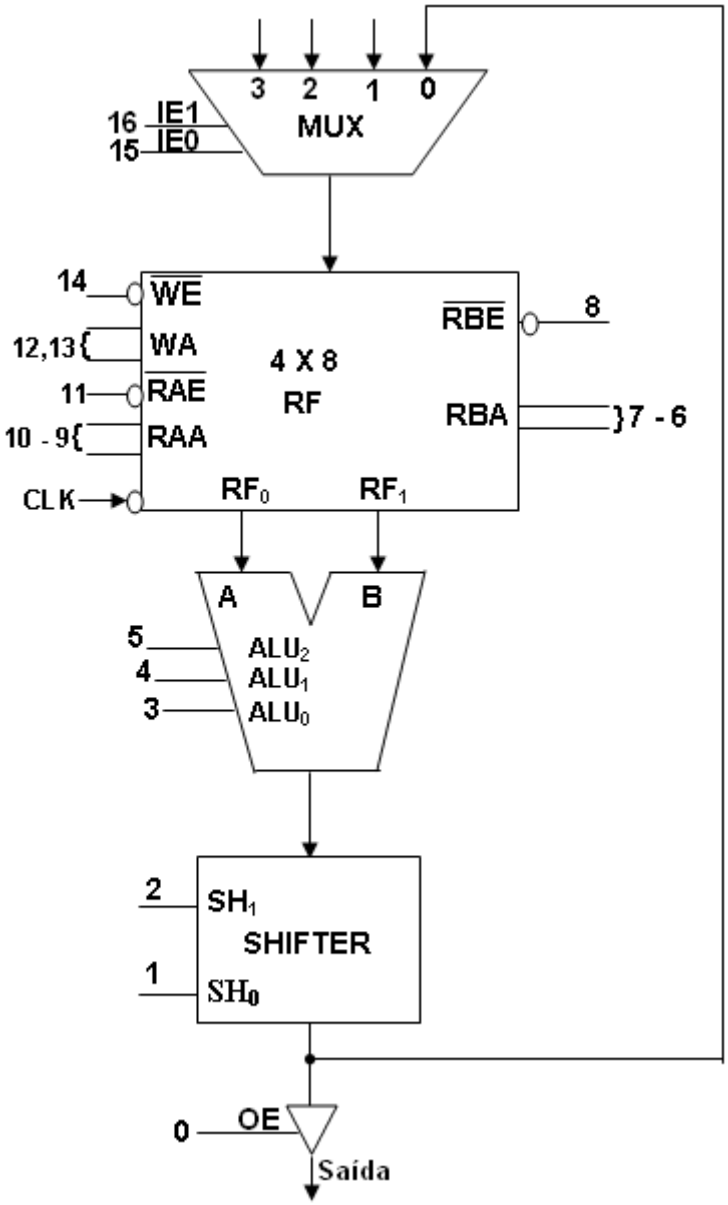
Fluxo de dados e memória de dados.



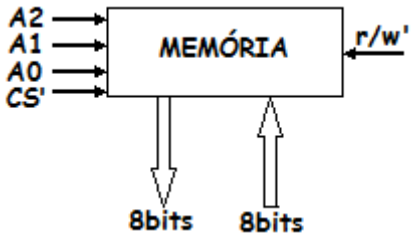
Sinais entre U.C., memória de dados e fluxo de dados.

1 - Entrada/Saída – Entrada e saída de dados linha de dados controlada por sinal R/W ';
2 – Linha de endereços 3 bits $A_2A_1A_0$, vindo da U.C.;
3 – Sinal de controle leitura e escrita, vindo da U.C.;
4 – Linha de controle PC's vindo da U.C.;
5 – Linha de "status" para U.C.

O fluxo de dados a seguir.



b) Memória de dados.



Projeto da U.C.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
```

```
ENTITY uc_micro IS
	PORT(
		clkuc		: IN std_logic;
		reset		: IN std_logic;
		P,N,Z		: IN std_logic;
		inst		: IN std_logic_vector(6 downto 0);
		--out_PC		: OUT INTEGER RANGE 0 to 7;
		out_PC		: OUT std_logic_vector(2 downto 0);
		out_opcode	: OUT std_logic_vector(6 downto 3);
		out_Ad		: OUT std_logic_vector(2 downto 0);
		B		: OUT std_logic_vector(3 downto 0);
		--salto		: INTEGER RANGE 0 to 15;
		--PC		: OUT INTEGER RANGE 0 to 7;
		PC		: IN std_logic_vector(2 downto 0);
		q		: OUT std_logic_vector(1 downto 0));
END uc_micro;

ARCHITECTURE Behavioral OF uc_micro IS
	TYPE maq_estado IS (inicial, busca, decodifica, executa);
	SIGNAL state: maq_estado;
BEGIN
	PROCESS (reset,clkuc)

	-- VARIABLE PM:		PM_BLOCK;
	VARIABLE IR:		std_logic_vector (6 downto 0);
	VARIABLE PC:		std_logic_vector(2 downto 0);
	VARIABLE OPCODE:	std_logic_vector (6 downto 3);
	--VARIABLE PC:		INTEGER RANGE 0 to 7;
	VARIABLE Ad:		std_logic_vector(2 downto 0);
	VARIABLE positive_flag,negative_flag,zero_flag  : std_logic;

	BEGIN
		IF reset = '0' THEN PC := "000";out_PC <="000";
					B <="0000";
					state <= inicial;
		ELSIF clkuc'EVENT AND clkuc = '1' THEN

			CASE state IS
				WHEN inicial =>
					state <= busca;

				WHEN busca =>
					state <= decodifica;
```

```
                        WHEN decodifica =>
                                PC := PC + 1;
                                state <= executa;

                        WHEN executa =>
                                IR := inst;
                                OPCODE:= IR(6 DOWNTO 3);
                                Ad:=IR(2 DOWNTO 0);
                                IF OPCODE ="1111" THEN PC:=Ad;
                                        ELSIF OPCODE ="1110" AND N = '1' THEN PC:=Ad;
                                                ELSIF OPCODE ="1101" AND Z = '1' THEN
PC:=Ad;
                                                        ELSIF OPCODE ="1100" AND Z = '0'
THEN PC:=Ad;
                                                                ELSIF OPCODE ="1011" AND P = '1'
THEN PC:=Ad;
                                                                        ELSE PC:=PC;
                                        END IF;
                                state <= busca;
                        END CASE;
                END IF;
                        CASE state IS
WHEN inicial => B <="0000";

WHEN busca =>
out_PC <= PC;B(3)<='1';B(2)<='1';B(1)<='0'; B(0)<='0';

WHEN decodifica =>
out_PC <= PC;B(3)<='0';B(2)<='0';B(1)<='1'; B(0)<='0';

WHEN executa =>
out_opcode<=OPCODE;B(3)<='0';B(2)<='0';B(1)<='1';B(0)<='1';
out_Ad<=Ad;
END CASE;


        END PROCESS;
                        WITH state SELECT
                        q <="00" WHEN inicial,
                                "01" WHEN busca,
                                "10" WHEN decodifica,
                                "11" WHEN executa;
END Behavioral;
```

Projeto do microprograma da U.C.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY microprograma_micro IS
	PORT(
		opcode			: IN std_logic_vector(3 downto 0); -- código da
instrução
		inst_ender		: IN std_logic_vector(2 downto 0); -- endereço da instrução
		B			: OUT std_logic_vector(20 downto 0)); -- palavra de
controle

END microprograma_micro;

ARCHITECTURE gera OF microprograma_micro IS
	BEGIN
		PROCESS (opcode,inst_ender)
			BEGIN

	IF OPCODE ="xxxx" THEN B(20) <='x';B(19) <='1'; B(18) <='0'; B(17) <='1';B(16) <='0';
B(15) <='1';B(14) <='0';B(13) <='1';B(12) <='0';B(11) <='1';B(10) <='0';B(9) <='1';B(8) <='0';
B(7) <='1';B(6) <='0';B(5) <='1';B(4) <='0';B(3) <='1'; B(2) <='0';B(1) <='1';B(0) <='1';
	END IF;
		END PROCESS;
END gera;
```

Carta de microoperações do fluxo de dados.

Instrução MOV Reg,(Ad)	Operações	$muxsel_{1,0}$ 20,19	W_E 18	$W_{A2..0}$ 17-15	R_{AE} 14	$R_{AA2,0}$ 13–11	R_{BE} 10	$R_{AB2..0}$ 9..7	R/W' 6	$ALU_{2,1,0}$ 5 – 3	$SH_{1,0}$ 2 – 1	O_E 0	Total Microop.
LDA,(Ad)	A ← Mem [xxx]	xx	0	xxx	0	xxx	1	000	1	xxx	xx	0	1
LDB,(Ad)	B ← Mem [xxx]	xx	0	xxx	0	xxx	1	001	1	xxx	xx	0	1
LDC,(Ad)	C ← Mem [xxx]	xx	0	xxx	0	xxx	1	010	1	xxx	xx	0	1
LDD,(Ad)	D ← Mem [xxx]	xx	0	xxx	0	xxx	1	011	1	xxx	xx	0	1
LDE,(Ad)	E ← Mem [xxx]	xx	0	xxx	0	xxx	1	100	1	xxx	xx	0	1
LDF,(Ad)	F ← Mem [xxx]	xx	0	xxx	0	xxx	1	101	1	xxx	xx	0	1
LDG,(Ad)	G ← Mem [xxx]	xx	0	xxx	0	xxx	1	110	1	xxx	xx	0	1
LDH,(Ad)	H ← Mem [xxx]	xx	0	xxx	0	xxx	1	111	1	xxx	xx	0	1
STA,(Ad)	Mem [xxx] ← A	xx	0	xxx	0	xxx	1	000	0	xxx	xx	0	1
STB,(Ad)	Mem [xxx] ← B	xx	0	xxx	0	xxx	1	001	0	xxx	xx	0	1
STC,(Ad)	Mem [xxx] ← C	xx	0	xxx	0	xxx	1	010	0	xxx	xx	0	1
STD,(Ad)	Mem [xxx] ← D	xx	0	xxx	0	xxx	1	011	0	xxx	xx	0	1
STE,(Ad)	Mem [xxx] ← E	xx	0	xxx	0	xxx	1	100	0	xxx	xx	0	1
STF,(Ad)	Mem [xxx] ← F	xx	0	xxx	0	xxx	1	101	0	xxx	xx	0	1
STG,(Ad)	Mem [xxx] ← G	xx	0	xxx	0	xxx	1	110	0	xxx	xx	0	1
STH,(Ad)	Mem [xxx] ← H	xx	0	xxx	0	xxx	1	111	0	xxx	xx	0	1

Instrução ADD A,Reg	Operações	$muxsel_{1,0}$ 20,19	W_E 18	$W_{A1..0}$ 17-15	R_{AE} 14	$R_{AA1,0}$ 13 – 11	R_{BE} 10	$R_{AB1..0}$ 9..7	R/W' 6	$ALU_{2,1,0}$ 5 – 3	$SH_{1,0}$ 2 – 1	O_E 0	Total Microop.
ADD A,A	A ← A + A												
ADD A,B	A ← A + B												
ADD A,C	A ← A + C												
ADD A,D	A ← A + D												

Instrução SUB A,Reg	Operações	$muxsel_{1,0}$ 20,19	W_E 18	$W_{A1..0}$ 17-15	R_{AE} 14	$R_{AA1,0}$ 13 – 11	R_{BE} 10	$R_{AB1..0}$ 9..7	R/W' 6	$ALU_{2,1,0}$ 5 – 3	$SH_{1,0}$ 2 – 1	O_E 0	Total Microop.
SUB A,A	A ← A - A												
SUB A,B	A ← A - B												
SUB A,C	A ← A - C												
SUB A,D	A ← A - D												

Instrução IN Reg	Operações	$muxsel_{1,0}$ 20,19	W_E 18	$W_{A1..0}$ 17-15	R_{AE} 14	$R_{AA1,0}$ 13 – 11	R_{BE} 10	$R_{AB1..0}$ 9..7	R/W' 6	$ALU_{2,1,0}$ 5 – 3	$SH_{1,0}$ 2 – 1	O_E 0	Total Microop.
IN A	A ← d												
IN B	B ← d												
IN C	C ← d												
IN D	D ← d												

Instrução OUT Reg	Operações	$muxsel_{1,0}$ 20,19	W_E 18	$W_{A1..0}$ 17-15	R_{AE} 14	$R_{AA1,0}$ 13 – 11	R_{BE} 10	$R_{AB1..0}$ 9..7	R/W' 6	$ALU_{2,1,0}$ 5 – 3	$SH_{1,0}$ 2 – 1	O_E 0	Total Microop.
OUT A	Saída ← A												
OUT B	Saída ← B												
OUT C	Saída ← C												
OUT D	Saída ← D												

Projeto da U.C.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY uc_uc_micro IS
	PORT(
		clkuc		: IN std_logic;
		reset		: IN std_logic;
		B			: OUT std_logic_vector(16 downto 0);
		q			: OUT std_logic_vector(1 downto 0));
END uc_micro;

ARCHITECTURE Behavioral OF uc_micro IS
	TYPE maq_estado IS (busca, decodifica, executa);
	SIGNAL state: maq_estado;
BEGIN
	PROCESS (clkuc)
	BEGIN
		IF reset = '0' THEN
					state <= limpa;
		ELSIF clkuc'EVENT AND clkuc = '1' THEN
			CASE state IS
				WHEN busca =>
					state <= decodifica;

				WHEN decodifica =>
					state <= executa;

				WHEN executa =>
					state <= busca;
END CASE;
		END IF;
			CASE state IS
WHEN busca =>
B(16)<='1';B(15)<='1';B(14)<='1';B(13)<='1';B(12)<='0';B(11)<='1';B(10)<='1';B(9)<='0';B(8)<='1';
B(7)<='1';B(6)<='0';B(5)<='1';B(4)<='0';B(3)<='1';B(2)<='0';B(1)<='0'; B(0)<='0';

WHEN decodifica =>
B(16)<='0';B(15)<='0';B(14)<='1';B(13)<='0';B(12)<='0';B(11)<='0';B(10)<='0';B(9)<='0';B(8)<='0'
;B(7)<='0';B(6)<='0';B(5)<='0';B(4)<='0';B(3)<='0';B(2)<='0';B(1)<='0'; B(0)<='0';

WHEN executa =>
```

```
B(16)<='0';B(15)<='1';B(14)<='1';B(13)<='0';B(12)<='1';B(11)<='0';B(10)<='0';B(9)<='0';B(8)<='0';
B(7)<='0';B(6)<='0';B(5)<='0';B(4)<='0';B(3)<='0';B(2)<='0';B(1)<='0'; B(0)<='0';

END CASE;
    END PROCESS;
        WITH state SELECT
        q <="00" WHEN busca,
            "01" WHEN decodifica,
            "10" WHEN executa;
END Behavioral;
```

Projeto do controlador.

As microoperações a serem realizadas na execução de cada instrução são mostradas a seguir.

Instrução MOV Reg,(Ad)	**Operações**	**$muxsel_{1,0}$ 13,12**	**W_E**	**$W_{A1..0}$**	**R_{AE} 9**	**$R_{AA1,0}$ 8 – 6**	**R_{BE}**	**$R_{AB1..0}$**	**$ALU_{2,1,0}$ 5 – 3**	**$SH_{1,0}$ 2 – 1**	**O_E 0**	**Total Microop.**
MOV A,(Ad)	**A ← Mem [xxxx]**											
MOV B,(Ad)	**B ← Mem [xxxx]**											
MOV C,(Ad)	**C ← Mem [xxxx]**											
MOV D,(Ad)	**D ← Mem [xxxx]**											
Instrução ADD A,Reg	**Operações**	**$muxsel_{1,0}$ 13,12**	**W_E**	**$W_{A1..0}$**	**R_{AE} 9**	**$R_{AA1,0}$ 8 – 6**	**R_{BE}**	**$R_{AB1..0}$**	**$ALU_{2,1,0}$ 5 – 3**	**$SH_{1,0}$ 2 – 1**	**O_E 0**	**Total Microop.**
ADD A,A	**A ← A + A**											
ADD A,B	**A ← A + B**											
ADD A,C	**A ← A + C**											
ADD A,D	**A ← A + D**											
Instrução SUB A,Reg	**Operações**	**$muxsel_{1,0}$ 13,12**	**W_E**	**$W_{A1..0}$**	**R_{AE} 9**	**$R_{AA1,0}$ 8 – 6**	**R_{BE}**	**$R_{AB1..0}$**	**$ALU_{2,1,0}$ 5 – 3**	**$SH_{1,0}$ 2 – 1**	**O_E 0**	**Total Microop.**
SUB A,A	**A ← A - A**											
SUB A,B	**A ← A - B**											
SUB A,C	**A ← A - C**											
SUB A,D	**A ← A - D**											
Instrução IN Reg	**Operações**	**$muxsel_{1,0}$ 13,12**	**W_E**	**$W_{A1..0}$**	**R_{AE} 9**	**$R_{AA1,0}$ 8 – 6**	**R_{BE}**	**$R_{AB1..0}$**	**$ALU_{2,1,0}$ 5 – 3**	**$SH_{1,0}$ 2 – 1**	**O_E 0**	**Total Microop.**
IN A	**A ← d**											
IN B	**B ← d**											
IN C	**C ← d**											
IN D	**D ← d**											
Instrução OUT Reg	**Operações**	**$muxsel_{1,0}$ 13,12**	**W_E**	**$W_{A1..0}$**	**R_{AE} 9**	**$R_{AA1,0}$ 8 – 6**	**R_{BE}**	**$R_{AB1..0}$**	**$ALU_{2,1,0}$ 5 – 3**	**$SH_{1,0}$ 2 – 1**	**O_E 0**	**Total Microop.**
OUT A	**Saída ← A**											
OUT B	**Saída ← B**											
OUT C	**Saída ← C**											
OUT D	**Saída ← D**											

Instrução OUT Reg	Operações	$muxsel_{1,0}$ 13,12	W_E	$W_{A1..0}$	R_{AE} 9	$R_{AA1,0}$ 8 – 6	R_{BE}	$R_{AB1..0}$	$ALU_{2,1,0}$ 5 – 3	$SH_{1,0}$ 2 – 1	O_E 0	Total Microop.
INC A	A ← A + 1											
INC B	B ← B + 1											
INC C	C ← C + 1											
INC D	D ← D + 1											
DEC A	A ← A + 1											
INC B	B ← B + 1											
INC C	C ← C + 1											
INC D	D ← D + 1											
Instrução Jump cond	Operações	$muxsel_{1,0}$ 13,12	W_E	$W_{A1..0}$	R_{AE} 9	$R_{AA1,0}$ 8 – 6	R_{BE}	$R_{AB1..0}$	$ALU_{2,1,0}$ 5 – 3	$SH_{1,0}$ 2 – 1	O_E 0	Total Microop.
JMP	PC ← Ad											
JP	PC ← Ad											
JN	PC ← Ad											
JNZ	PC ← Ad											

Salto Condicional P, N e Z.

Processador Programável para 32 instruções de 16bits.

Um processador capaz de processar até seis instruções é implementado a seguir. Mais potente que o processador de três instruções, a extensão permitirá realizar operações de transferência e de movimento de dados. Uma instrução de teste foi acrescentada no conjunto de instruções para operação com programas com laços.
Esse conjunto de instruções permitirá programar exemplos como o da máquina de refrigerantes, de aproximação sucessivas onde o algoritmo será desenvolvido na programação. Ao conjunto de três instruções a saber: carregar, armazenar e somar serão acrescentadas as instruções descritas a seguir.

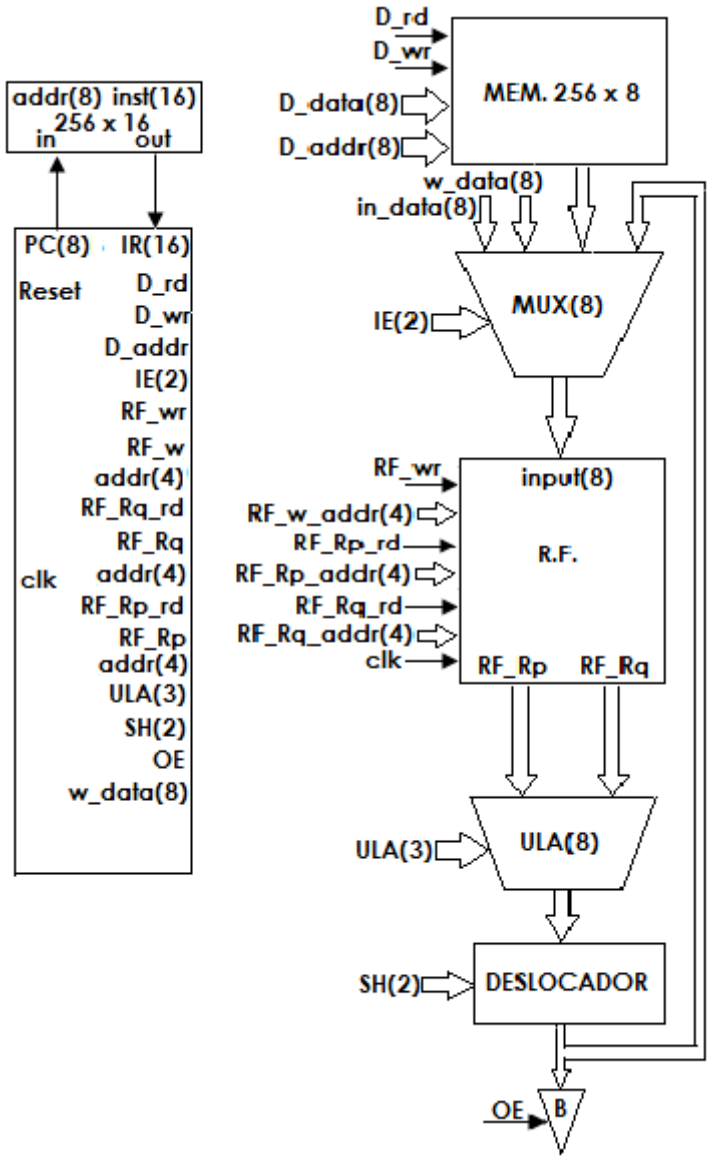
Podemos estender o conjunto de instruções do processador de 03 instruções para 16 instruções. Iniciando pela introdução da instrução que é capaz de:

- Instrução MOV Ra, #c, carregar uma constante – cujo opcode é 0011 em um registrador do banco de registradores. Uma instrução que movesse o valor da constante para dentro do registrador. O mnemônico da instrução será MOVRa,#c, uma instrução de endereçamento imediato. Opcode: 0011 $r_3r_2r_1r_0$ $c_7c_6c_5c_4c_3c_2c_1c_0$, onde $r_3r_2r_1r_0$ especifica qual registrador deve ser carregado com o valor dado por $c_7c_6c_5c_4c_3c_2c_1c_0$, RF[0...15] = c;
- Instrução SUB Ra, Rb,rc, subtrair o conteúdo dos registradores dados pelos endereços dos registradores rb e rc e armazenar o conteúdo da subtração no registrador Ra. O opcode dessa instrução é 0100 – $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$, e especifica uma subtração entre 2 conteúdos de 2 registradores de RF fontes endereçados por rb

e rc e armazenados no registrador destino $ra_3ra_2ra_1ra_0$. A instrução 0100 0000 0001 0010, especifica a: RF[0] = RF[1] – RF[2]. O mnemônico da instrução é: SUBra,Rb,rc.

- Instrução JZ d - salto se zero – cujo opcode é 0101 especifica o endereço do salto resultado da comparação com o valor zero, dado por 0000 $o_7o_6o_5o_4o_3o_2o_1o_0$, onde o endereço do salto condicional é dado por $o_7o_6o_5o_4o_3o_2o_1o_0$.
- Instrução Out (Ra) – saída de conteúdo – cujo opcode é 0110 especifica o endereço do conteúdo em RF dado por $ra_3ra_2ra_1ra_0$ 0000 0000 e envia para a saída esse conteúdo.
- Instrução SHR Ra – desloca a direita 1 bit MSB – cujo opcode é 0111 especifica um deslocamento a direita do bit LSB para MSB e bit zero em D_0 no conteúdo endereçado pelo registrador $ra_3ra_2ra_1ra_0$.
- Instrução SHL Ra – desloca a esquerda 1 bit LSB – cujo opcode é 1000 especifica um deslocamento a esquerda do bit MSB para LSB e bit zero em D_7 no conteúdo endereçado pelo registrador $ra_3ra_2ra_1ra_0$.
- Instrução JMP d, – cujo o opcode 1001 especifica desviar o programa corrente incondicionalmente para o valor do offset. O PC é carregado imediatamente pelo valor do offset (d) e o programa passará a continuar na instrução nesse endereço.
- Instrução JP d, – cujo opcode é 1010 especifica desviar o programa corrente se o resultado na ULA apresentar um valor positivo. O PC será carregado com o valor do offset se P = 1 e permanece corrente no caso contrário.
- Instrução JN d, – cujo opcode é 1011 especifica desviar o programa corrente se o resultado na ULA apresentar um valor negativo. O PC será carregado com o valor do offset se N = 1 e permanecerá corrente no caso contrário.
- Instrução INC Ra, - cujo opcode é 1100 especifica incrementar uma unidade no conteúdo do registrador endereçado por ra.
- Instrução DEC Ra, - cujo opcode é 1101 especifica decrementar uma unidade no conteúdo do registrador endereçado por ra.
- Instrução IN (Ra), - cujo opcode é 1110 especifica entrar com o dado externo e armazenar no endereço dado por ra.
- Instrução HLT – cujo opcode é 1111 especifica a parada do programa.

Arquitetura do processador de 8 instruções.



Códigos de Operações e mnemônicos das instruções.

Tabela dos códigos de operações

Projeto da Unidade de Controle.

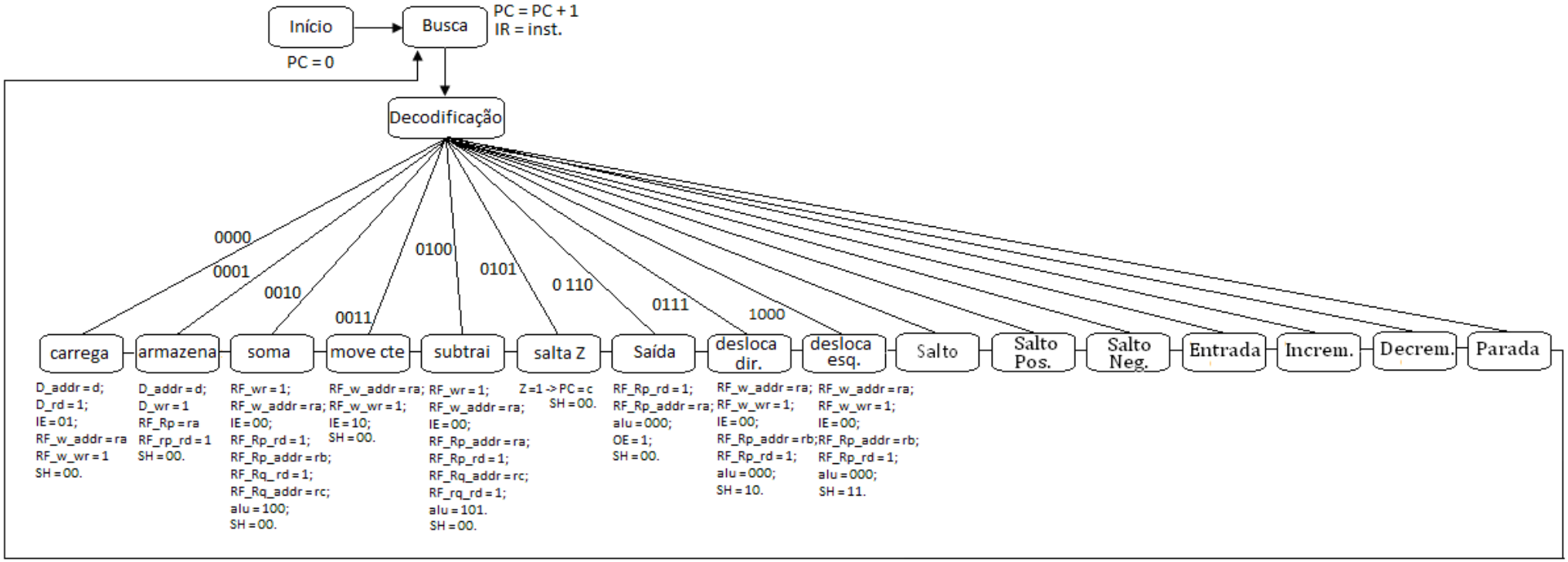
A F.S.M. da unidade de controle têm 2 ciclos de instrução, sendo ciclo de busca e ciclo de execução da instrução. O total serão 4 estados sendo: Início, busca, decodificação e execução. Para cada estado a U.C. consome um ciclo de relógio. Para ciclo o ciclo de busca as seguintes microoperações são realizadas pela U.C..

1. PC = 0 (estado inicial);
2. PC = PC + 1;
3. IR ← Instrução;

Para o ciclo de execução, teremos:

4. Execução da instrução.

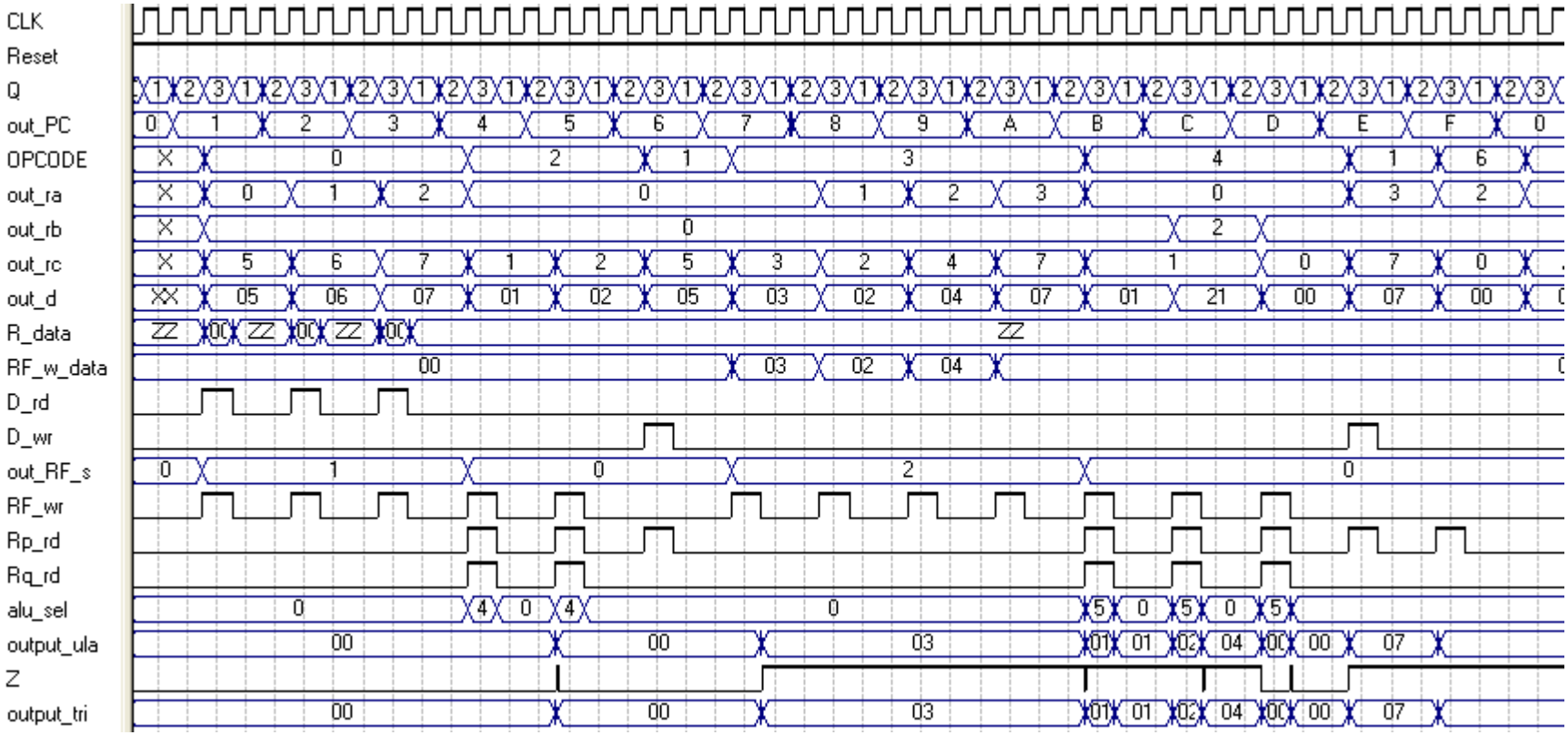
A seguir apresentamos a F.S.M. com o ciclo de busca e de execução das 16 instruções.



Programas em linguagem de montagem.

1. Realizar a multiplicação entre dois números. Conforme o algoritmo utilizado no capítulo de máquinas de uso específico fazer a programação e simulação da multiplicação.

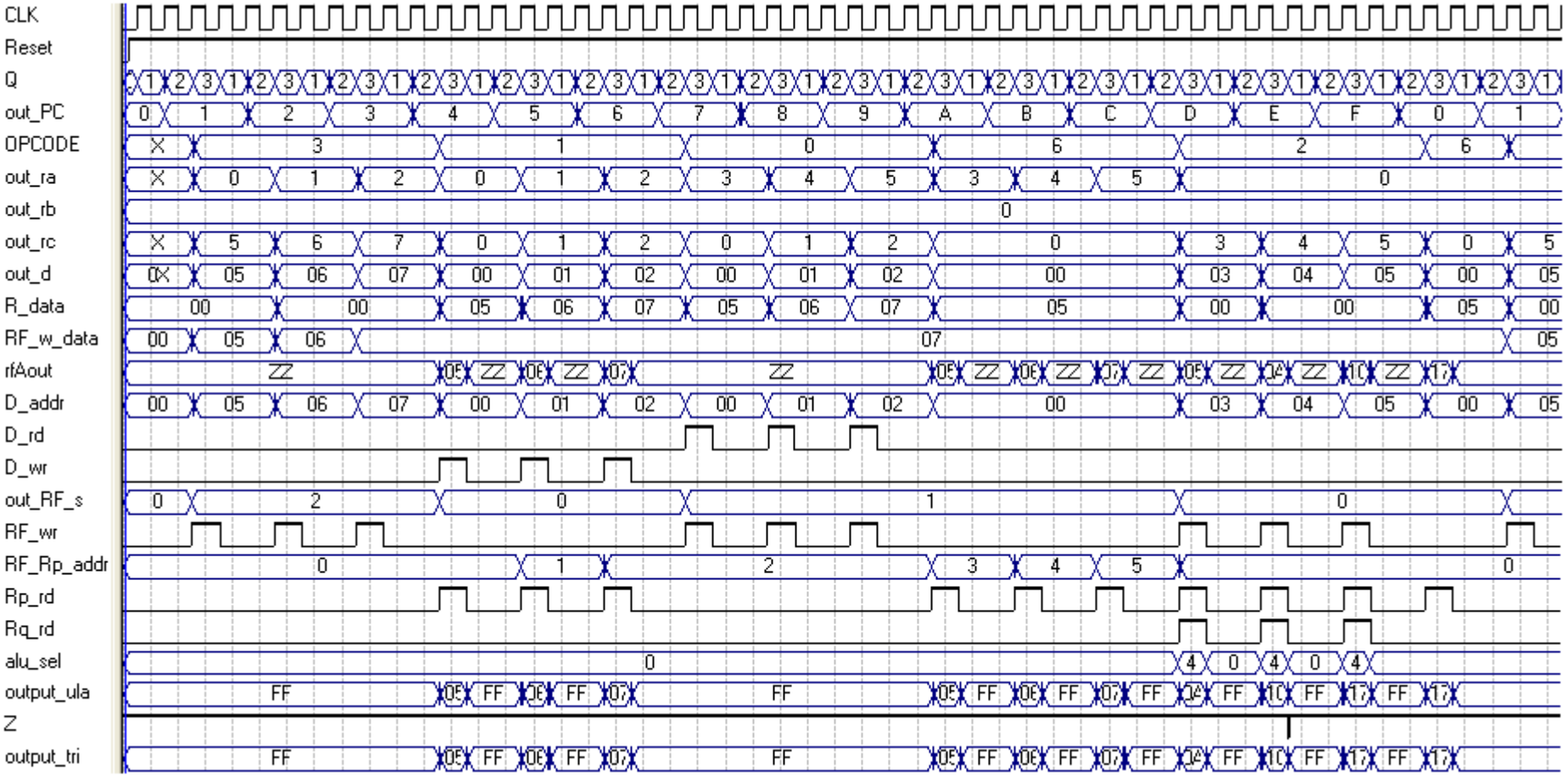
2. Realizar o algoritmo da aproximação sucessiva. Realizar a programação conforme o capítulo de máquinas de uso específico e simulação.


CLK
Reset
Q
out_PC
OPCODE
out_ra
out_rb
out_rc
out_d
R_data
RF_w_data
D_rd
D_wr
out_RF_s
RF_wr
Rp_rd
Rq_rd
alu_sel
output_ula
Z
output_tri

Exercício: Um programa colocado na memória produziu a forma de onda apresentada abaixo, pede-se:

a) O programa escrito na memória de instruções.

b) O resultado da soma realizada.



Referências

1) Hwang, O Enoch – Principles of Digital Logic Design – Brookes/Cole 2002
2) Caldas L – Trabalhos e Notas de aulas em circuitos digitais.
3) Vahid, F. –

Agradecimentos

1) Caldas R. Mayra – Pelos serviços prestados na edição deste trabalho.

PROJETO DE UM PROCESSADOR PROGRAMÁVEL DE 32 INSTRUÇÕES.

O projeto de 32 instruções é uma extensão máxima do projeto do processador programável de 03 instruções. A arquitetura básica do processador de 03 instruções é modificada para que programas que contenham blocos de decisões fossem introduzidas e com isso a implementação de algorítmos mais complexos. A introdução do bloco "detetor de zero" vai permitir a resposta da ULA com operações cujo resultado produz um valor zero na saída, bem como o bloco comparador de amplitude que gera 03 saídas "maior, igual e menor" e um segundo bloco comparador o qual produz 02 saídas maior ou igual e menor ou igual. Todos esses indicadores servirão para produzirem um "status" da operação realizada na ULA ou nos testes de comparações. A tabela a seguir define todos os indicadores "flags" e a sua lógica ativa.

Indicador	Flag	Comentário
Zero	Z	Z = "1" qdo a operação realizada na ULA produz saída igual a zero.
Maior	M	M = "1" qdo a operação de comparação realizada entre 2 operandos do registrador de arquivos produzir a saída maior a "1".
Menor	N	N = "1" qdo a operação de comparação realizada entre 2 operandos do registrador de arquivos produzir a saída menor a "1".
Igual	I	I = "1" qdo a operação de comparação realizada entre 2 operandos do registrador de arquivos produzir a saída igual a "1".
Maior ou igual	MEQ	MEQ = "1" qdo a operação de comparação realizada entre 2 operandos do registrador de arquivos produzir a saída maior ou igual a "1".
Menor ou igual	NEQ	NEQ = "1" qdo a operação de comparação realizada entre 2 operandos do registrador de arquivos produzir a saída menor ou igual a "1".
Bit_DO	B0	B0 = "1" qdo o bit LSB do deslocador é igual a "1".
Bit_D7	B7	B7 = "1" qdo o bit MSB do deslocador é igual a "1".
Carry	C	C = "1" qdo a operação realizada na ULA estoura a sua capacidade máxima.
Overflow	OV	OV = "1" qdo a operação entre 2 operandos assinalados realizada na ULA produz um transbordamento na ULA.

Os indicadores informam a unidade de controle sobre o "status" da operação na execução da instrução. O programa feito pelo usuário deve utilizar essa informação para a tomada de decisão. A instrução de Salto foi expandida para 12 novas instruções e 4 bits seguidos dos 4 bits fixos do "opcode" servirão para a criação dessas instruções de saltos e como a seguir.

Expansão	Mnem.	Operação	Comentário
0000	JZ	Salto se zero	Se a operação realizada na ULA produz zero;
0001	JNZ	Salto se não zero	Se a operação realizada na ULA produz um resultado diferente de zero;
0010	-	-	-
0011	JNEQ	Salto se menor ou igual	Salto se a operação de comparação entre 2 números resulta em menor ou igual;
0100	-	-	-
0101	JMEQ	Salto se maior ou igual	Salto se a operação de comparação entre 2 números resulta em maior ou igual;
0110	-	-	-
0111	-	-	-

1000	JB0	Salto de bit_D0	Salto se bit D0 do deslocador é igual a "1";
1001	JNB0	Salto se bit_NDO	Salto se bit D0 do deslocador é igual a "0";
1010	JB7	Salto se bit_D7	Salto se bit D7 do deslocador é igual a "1";
1011	JNB7	Salto se bit_ND7	Salto se bit D7 do deslocador é igual a "0";
1100	JM	Salto se maior	Salto se a operação de comparação entre 2 números resulta em maior;
1101	JI	Salto se igual	Salto se a operação de comparação entre 2 números resulta em igual;
1110	JN	Salto se menor	Salto se a operação de comparação entre 2 números resulta em menor;
1111	JMP	Salto incondicional	Salto incondicionalmente

O conjunto de instruções com 16 instruções é apresentado a seguir, bem como a tabela com os "opcode" e comentários.

Opcode	Mnem	Operação	Comentário
0000	LD(ra),(d)	RF[ra] ← (d)	Carrega o conteúdo endereçado em d para o RF[ra];
0001	ST(d),(ra)	(d) ← RF[ra]	Armazena o conteúdo de RF[ra] no endereço (d);
0010	ADD(ra),(rb),(rc)	RF[ra] = RF[rb] + RF[rc]	Soma os conteúdos dados por RF[rb,rc] e armazena em RF[ra];
0011	MVI(ra) #c	RF[ra] ← w_data	Transfere o conteúdo imediato em RF[ra];
0100	SUB(ra),(Rb),(rc)	RF[ra] = RF[rb] - RF[rc]	Subtrai os conteúdos dados por RF[rb,rc] e armazena em RF[ra];
0101	JX, offset	X = 0000 a 1111	Salta para o endereço dado em offset;
0110	OUT(ra)	Saída ← RF[ra]	Transfere para a saída o conteúdo de RF[ra];
0111	IN(ra)	RF[ra] ← input_ext	Transfere da entrada o conteúdo para RF[ra];
1000	CMP(ra),(rb)	RF[ra] : RF[Rb]	Compara os conteúdos de RF[ra] com RF[rb];
1001	XOR(ra),(Rb),(rc)	RF[ra] = RF[rb] ⊕ RF[rc]	XOR os conteúdos dados por RF[rb,rc] e armazena em RF[ra];
1010	AND(ra),(Rb),(rc)	RF[ra] = RF[rb].RF[rc]	AND os conteúdos dados por RF[rb,rc] e armazena em RF[ra];
1011	OR(ra),(Rb),(rc)	RF[ra] = RF[rb] + RF[rc]	OR os conteúdos dados por RF[rb,rc] e armazena em RF[ra];
1100	INC(ra)	RF[ra] = RF[ra] - 1	Incrementa o conteúdo de RF[ra];
1101	DEC(ra)	RF[ra] = RF[ra] – 1	Decrementa o conteúdo de RF[ra];
1110	SHX	RF[ra] ← bit	Desloca o conteúdo RF[ra] bit;
1111	HLT	(PC) = (PC)	Pára o processamento.

Assim como foi projetado para a instrução SALTO é adotado o mesmo procedimento para as instruções de deslocamentos, as quais podem ser do tipo: deslocamento para esquerda, para a direita, circular a esquerda e circular a direita. A tabela a seguir apresenta a expansão da instrução SHX para 04 instruções a seguir.

Expansão	Mnem.	Lógica	Comentário
0000	SHL	RF[ra] ← 0	Deslocamento a esquerda do conteúdo do deslocador um bit e introdução do bit D0 igual a zero.
0001	SHR	RF[ra] ← 0	Deslocamento a direita do conteúdo do deslocador um bit e introdução do bit D7 igual a zero.
0010	ROL	RF[ra] ← D7	Deslocamento circular a esquerda do conteúdo do deslocador um bit e

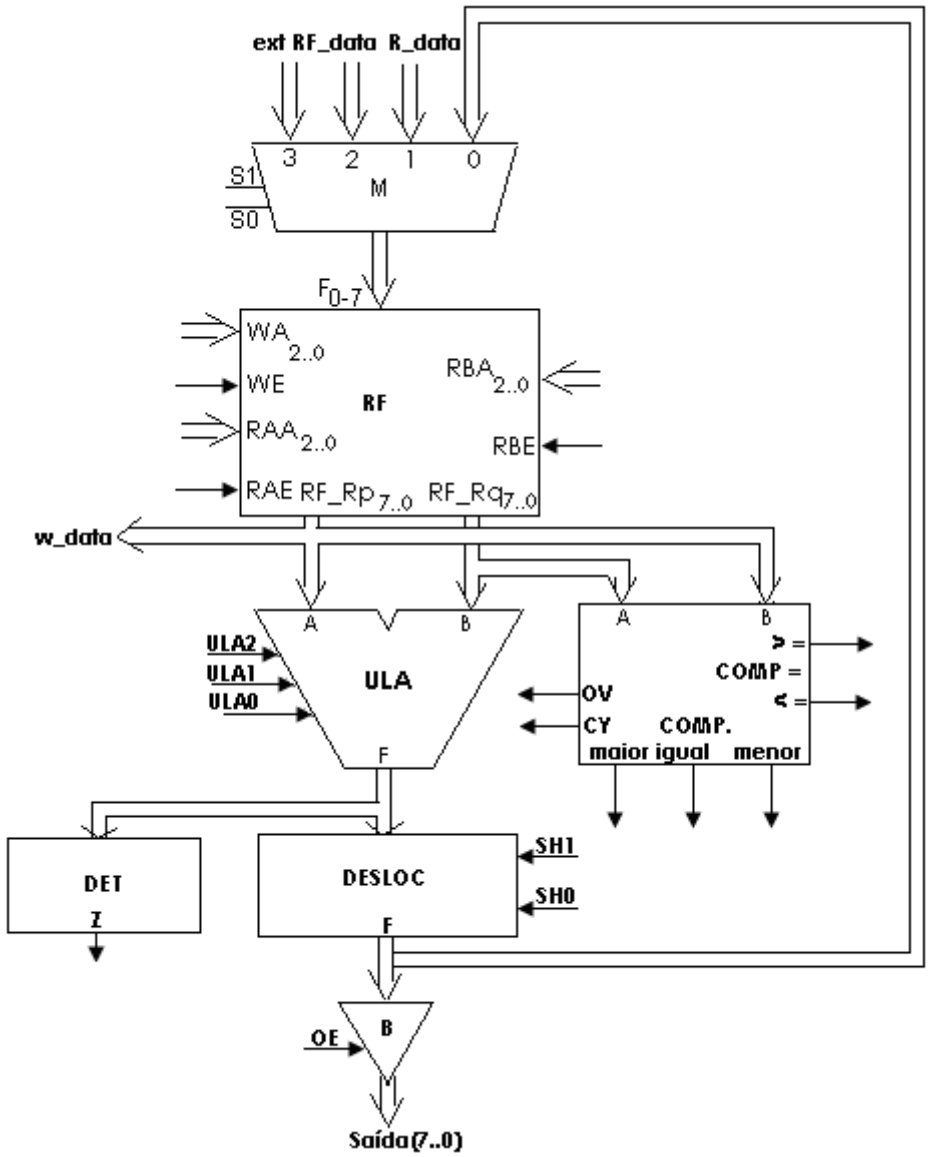
			introdução do bit D7.
0011	ROR	RF[ra] ← D0	Deslocamento circular a direita do conteúdo do deslocador um bit e introdução do bit D0.

Projeto do fluxo de dados para o processador de 16 instruções.

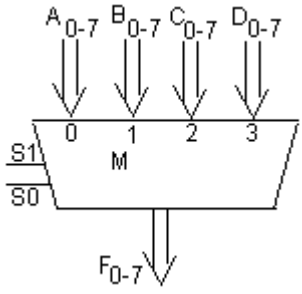
O projeto do fluxo de dados é uma melhoria na arquitetura do processador de 03 instruções apresentado nesse capítulo. Para que fossem expandido o conjunto de instruções, a introdução dos indicadores para a U.C. é fundamental para a tomada de decisão. Os blocos a seguir fazem parte do fluxo de dados para 16 instruções. A saber:

1. Bloco MUX;
2. Bloco RF;
3. Bloco comparador;
4. Bloco detetor de zero;
5. Bloco comparador com igualdade;
6. Bloco ULA;
7. Bloco deslocador;
8. Bloco saída.

Arquitetura do fluxo de dados.



a. Bloco Multiplex – Um bloco multiplexador de 8 bits com 2 variáveis de seleções das 04 entradas do multiplexador a saber:



As 04 entradas:
3: input_ext(d7..d0);
2: w_data(d7..d0);
1: read_data(d7..d0);
0: deslocador(d7..d0).

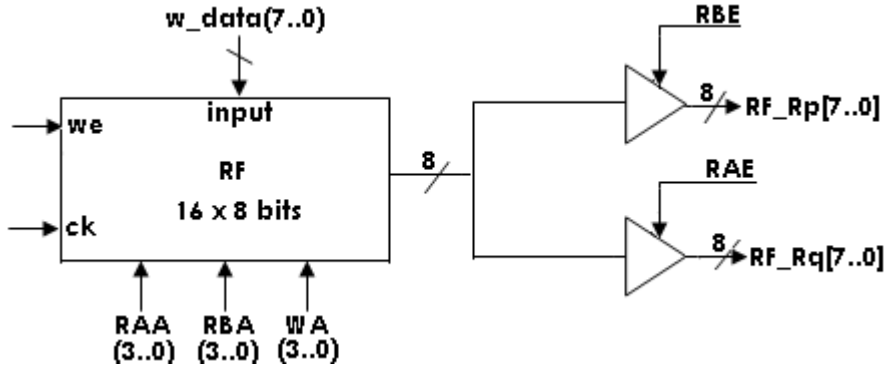
Em VHDL, fica:

```
-- 4-to-1 MUX
LIBRARY ieee;
USE ieee.std_logic_1164.all;
        ENTITY mux2 IS PORT (
        S1,S0     : IN std_logic; -- select line
        D3,D2,D1,D0: IN std_logic_vector(7 downto 0); -- data bus input
        Y: OUT std_logic_vector(7 downto 0)); -- data bus output
```

```
END mux2;

ARCHITECTURE Behavioral OF mux2 IS
        BEGIN
                PROCESS(S1,S0,D3,D2,D1,D0)
        BEGIN
        IF (S1 ='0')AND(S0='0')THEN
        Y <= D0;
        END IF;
        IF (S1 ='0') AND (S0 ='1') THEN
        Y <= D1;
        END IF;
        IF (S1 ='1') AND (S0='0') THEN Y <= D2;
        END IF;
        IF (S1 ='1') AND (S0 ='1') THEN Y <= D3;
        END IF;
                END PROCESS;
END Behavioral;
```

b. Bloco RF – Um bloco registrador de arquivo de 16 x 8bits, uma entrada input de 8bits para a escrita dos dados no RF[15..0], endereçada pela variável $WA_{3..0}$ controlada por we . A saída dos dados são por 02 canais RF_R_P e RF_R_P, onde canal de saída tem variável de endereçamento $RF\_R_P\_addr_{3..0}$, controlada por RF_R_P_rd e para o segundo canal a variável de endereçamento $RF\_Rq\_addr_{3..0}$, controlada pela variável RF_Rq_rd.

Bloco do Registrador de Arquivos - RF



RF_Rp = Porto (saída);
RF_Rq = Porto (saída);
we = Habilita escrita;
input= Entrada de dados;
WA = Endereço escrita.

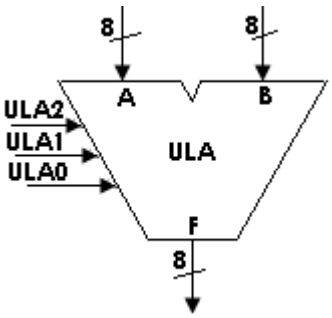
RAA = Endereço porto R_P;
RAB = Endereço porto Rq;
RAE = Habilita porto Rp;
RBE = Habilita porto Rq;

Em VHDL, fica:

```
-- Register File
	LIBRARY ieee;
	USE ieee.std_logic_1164.all;
	USE ieee.std_logic_unsigned.all;
--USE ieee.std_logic_arith.all;
	ENTITY regfile IS PORT (
	clk: IN std_logic; --clock
	WE: IN std_logic; --write enable
	WA: IN std_logic_vector(3 DOWNTO 0); --write address
	input: IN std_logic_vector(7 DOWNTO 0); --input
	RAE: IN std_logic; --read enable ports A & B
	RAA: IN std_logic_vector(3 DOWNTO 0); --read address port A & B
	RBE: IN std_logic; --read enable ports A & B
	RBA: IN std_logic_vector(3 DOWNTO 0); --read address port A & B
	Aout, Bout: OUT std_logic_vector(7 DOWNTO 0)); --output port A & B
	END regfile;


ARCHITECTURE Behavioral OF regfile IS
	SUBTYPE reg IS std_logic_vector(7 DOWNTO 0);
	TYPE regArray IS array(0 TO 7) OF reg;
	SIGNAL RF: regArray; --register file contents
		BEGIN
			WritePort: PROCESS (clk)
				BEGIN
					IF (clk'EVENT AND clk = '1') THEN
						IF (WE = '1') THEN
					RF(CONV_INTEGER(WA)) <= input;
						END IF;
					END IF;
						END PROCESS;

			ReadPortA: PROCESS (RAE, RAA)
				BEGIN
					IF (RAE = '1') then
					Aout <= RF(CONV_INTEGER(RAA)); -- convert bit VECTOR to integer
					ELSE
					Aout <= (others => 'Z');
					END IF;
					END PROCESS;

			ReadPortB: PROCESS (RBE, RBA)
				BEGIN
					IF (RBE = '1') then
					Bout <= RF(CONV_INTEGER(RBA)); -- convert bit VECTOR to integer
					ELSE
					Bout <= (others => 'Z');
					END IF;
					END PROCESS;
	END Behavioral;
```

c. Bloco ULA – É um bloco de capacidade de 8bits onde se realizam as operações aritméticas e lógicas. A saída F com 8 bits é definida conforme a tabela funcional a seguir. Como o número de funções da ULA é igual a 8, são necessárias 03 variáveis de seleções $ULA_{2..0}$. A tabela a seguir apresenta o funcional da ULA.

ULA2	ULA1	ULA0	Operação
0	0	0	PASSA A
0	0	1	A AND B
0	1	0	A OR B
0	1	1	COMPL. A
1	0	0	A PLUS B
1	0	1	A - B
1	1	0	A + 1
1	1	1	A - 1

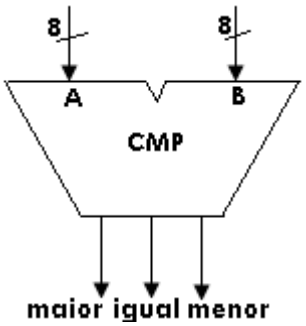


Em VHDL, fica:

```
-- ALU
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- need the following to perform arithmetics on std_logic_vectors
USE ieee.std_logic_unsigned.all;
        ENTITY alu IS PORT (
        ALUSel: IN std_logic_vector(2 DOWNTO 0); -- select for operations
        A, B: IN std_logic_vector(7 DOWNTO 0); -- input operands
        C       : OUT std_logic;--vai um
        F: OUT std_logic_vector(7 DOWNTO 0)); -- output
END alu;

        ARCHITECTURE Behavior OF alu IS
                BEGIN
                        PROCESS(ALUSel, A, B)
                                BEGIN
                                        CASE ALUSel IS
                                                WHEN "000" => -- pass A through
                                                F <= A;
                                                WHEN "001" => -- AND
                                                F <= A AND B;
                                                WHEN "010" => -- OR
                                                F <= A OR B;
                                                WHEN "011" => -- NOT
                                                F <= A XOR B;
                                                WHEN "100" => -- add
                                                F <= A + B;
                                                WHEN "101" => -- subtract
                                                F <= A - B;
                                                WHEN "110" => -- increment
                                                F <= A + 1;
                                                WHEN others => -- decrement
                                                F <= A - 1;
                                        END CASE;
                        END PROCESS;
```

```
END Behavior;
```

d. Bloco comparador – É um bloco com a capacidade de 8 bits o qual opera com 02 operandos A e B e realiza a função de comparação entre eles. O comparador produz 03 saídas a saber: maior, menor e igual.

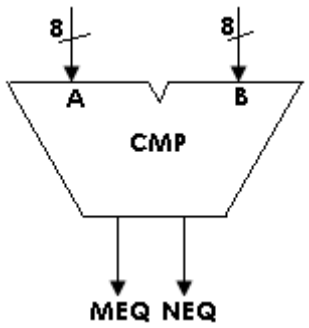


Maior $\rightarrow$ (A > B);
Menor $\rightarrow$ (A < B);
Igual $\rightarrow$ (A = B).

Em VHDL, fica:

```
-- Comparador 8 bits
 library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity comparador_8bits is
    port(
       A : in STD_LOGIC_VECTOR(7 downto 0);
       B : in STD_LOGIC_VECTOR(7 downto 0);
       igual : out STD_LOGIC;
       maior : out STD_LOGIC;
       menor : out STD_LOGIC);
end comparador_8bits;
architecture comp of comparador_8bits is
begin
   comparador : process (a,b) is
   begin
      if (a=b) then
         igual <= '1';
         maior <= '0';
         menor <= '0';
      elsif (a<b) then
         igual <= '0';
         maior <= '0';
         menor <= '1';
      else
         igual <= '0';
         maior <= '1';
         menor <= '0';
      end if;
   end process comparador;

end comp;
```

e. Bloco comparador de igualdade

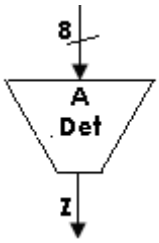


Maior ou Igual→ (A ≥ B);
Menor ou Igual → (A ≤ B).

Em VHDL, fica:

```
-- Comparador_2 8 bits
 library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity comparador_2 is
    port(
        A : in STD_LOGIC_VECTOR(7 downto 0);
        B : in STD_LOGIC_VECTOR(7 downto 0);
        MEQ : out STD_LOGIC;
        NEQ : out STD_LOGIC);
end comparador_2;

architecture comp_2 of comparador_2 is
begin
   comparador : process (a,b) is
   begin
      if (a=b) OR (a>b)then
         MEQ <= '1';ELSE MEQ <='0';
      END IF;
      if (a=b) OR (a<b) then
         NEQ <= '1';ELSE NEQ <='0';
      end if;
   end process comparador;
end comp_2;
```

f. Detetor de zero



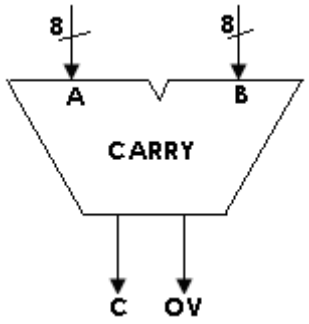
A = 0 → Z = 1

Em VHDL, fica:

```
--Detetor de zero
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

```
entity detetor is
    port(
       a : in STD_LOGIC_VECTOR(7 downto 0);
       Z : out STD_LOGIC);
end detetor;

architecture det of detetor is
begin
   detetor : process (A) is
   begin
      if A <="00000000" then
         Z <= '1';
         ELSE
         Z <= '0';
        end if;
   end process detetor;
end det;
```

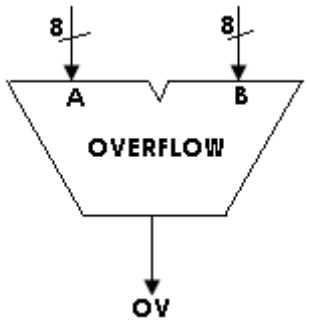
g. Bit de vai um "carry"




```
--vai um
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY vai_um IS
   PORT
      (
       A_in     : IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
       B_in     : IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
       carry_in : IN   STD_LOGIC;
       --sum      : OUT  STD_LOGIC_VECTOR(7 DOWNTO 0);
       carry_out : OUT  STD_LOGIC);
END vai_um;

ARCHITECTURE behavioral OF vai_um IS

SIGNAL   carry_generate    :   STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL   carry_propagate   :    STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL   carry_in_internal :   STD_LOGIC_VECTOR(7 DOWNTO 1);

BEGIN
```

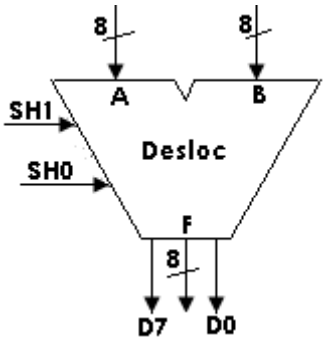
```
   carry_generate <= A_in AND B_in;
   carry_propagate <= A_in OR B_in;
   PROCESS (carry_generate,carry_propagate,carry_in_internal)
   BEGIN
   carry_in_internal(1) <= carry_generate(0) OR (carry_propagate(0) AND carry_in);
      inst: FOR i IN 1 TO 6 LOOP
           carry_in_internal(i+1) <= carry_generate(i) OR (carry_propagate(i) AND carry_in_internal(i));
           END LOOP;
   carry_out <= carry_generate(7) OR (carry_propagate(7) AND carry_in_internal(7));
   END PROCESS;
END behavioral;
```

h. Bit de transbordamento "OV"




```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;
ENTITY adder8 IS
PORT ( Cin : IN STD_LOGIC ;
A, B : IN STD_LOGIC_VECTOR(7 DOWNTO 0) ;
S : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) ;
Cout,Overflow : OUT STD_LOGIC ) ;
END adder8 ;
ARCHITECTURE Behavior OF adder8 IS
SIGNAL Sum : STD_LOGIC_VECTOR(8 DOWNTO 0) ;
BEGIN
Sum <= ('0' & A) + B + Cin ;
S <= Sum(7 DOWNTO 0) ;
Cout <= Sum(8) ;
Overflow <= Sum(8) XOR A(7) XOR B(7) XOR Sum(7) ;
END Behavior ;
```

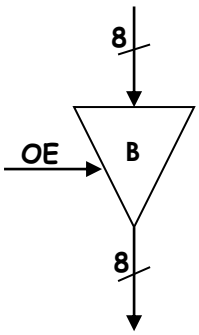
i. Deslocador 8 bits



Em VHDL, fica:

```
-- deslocador 8 bits
LIBRARY ieee;
USE ieee.std_logic_1164.all;
	ENTITY deslocador IS PORT (
		SHSel: IN std_logic_vector(1 DOWNTO 0); -- seleção das operações
		input: IN std_logic_vector(7 DOWNTO 0); -- operandos de entrada
		bit\-carry: IN std_logic;--bit carry de entrada
		output: OUT std_logic_vector(7 DOWNTO 0); -- saída
		bit_D0: OUT std_logic;
		bit_D7:		OUT std_logic);
	END deslocador;
	ARCHITECTURE Behavior OF deslocador IS
		BEGIN
			PROCESS(SHSel, input)
				BEGIN
					CASE SHSel IS

						WHEN "00" => -- passagem através
						bit_D0 <= input(0);bit_D7 <= input(7);
						output <= input;
						WHEN "01" => -- desloca direita
						output <= input(6 DOWNTO 0) & '0';
						WHEN "10" => -- desloca esquerda
						output <= bit_carry & input(7 DOWNTO 1);
						WHEN OTHERS => -- giro a direita
						output <= input(0) & input(7 DOWNTO 1);
					END CASE;
				END PROCESS;
	END Behavior;
```

j. Saída 8 bits

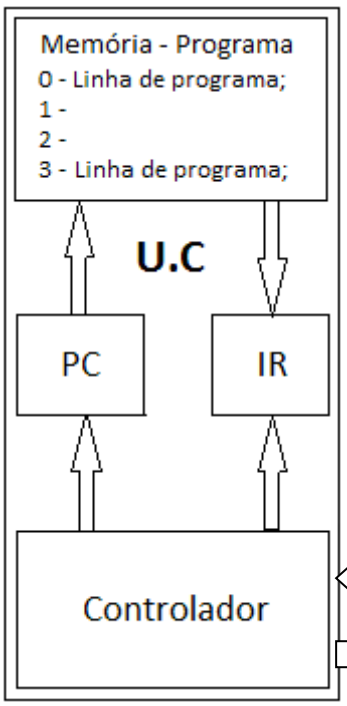


Em VHDL, fica:

```
-- Tri-state buffer
LIBRARY ieee;
USE ieee.std_logic_1164.all;
        ENTITY TriStateBuffer IS PORT (
        OE: IN std_logic;
        D: IN std_logic_vector(7 DOWNTO 0);
        saida: OUT std_logic_vector(7 DOWNTO 0));
        END TriStateBuffer;

ARCHITECTURE Behavioral OF TriStateBuffer IS
                BEGIN
                        PROCESS (OE, D) -- get error message if no d
                                BEGIN
                                IF (OE = '1') THEN
                                saida <= D;
                                ELSE
                                saida <= (OTHERS => 'Z'); -- to get 8 Z values
                                END IF;
                        END PROCESS;
        END Behavioral;
```

Projeto da unidade de controle.

A unidade de controle é responsável pelo ciclo de busca e de execução da instrução. Cada instrução deve ser decodificada e a execução é feita no fluxo de dados. A u.c. gera para cada instrução uma palavra de controle, a qual controla a operação em cada bloco do fluxo de dados. A microoperação realizada é gerada pela u.c. sincronizada com o sinal de relógio. Por exemplo, a instrução LD(ra),(d), para ser executada a u.c. deve gerar sinais de controle para a memória externa (leitura de dados), deverá também encaminhar o dado lido para o endereço de um registrador no registrador de arquivo. Outras instruções seguem também sinais específicos para a operação indicada na instrução. O diagrama de bloco a seguir da arquitetura da unidade de controle é apresentada.



Bloco U.C. é composto por:

1. contador de instrução PC(7..0);
2. Registrador de instrução(15..0);
3. Controlador (decodificador)

Bloco memória de programa – Contém linhas de programa a serem executadas na u.c.

Os sinais relativos a microoperação são gerados na u.c.;
A u.c. recebe o "status" do fluxo de dados.

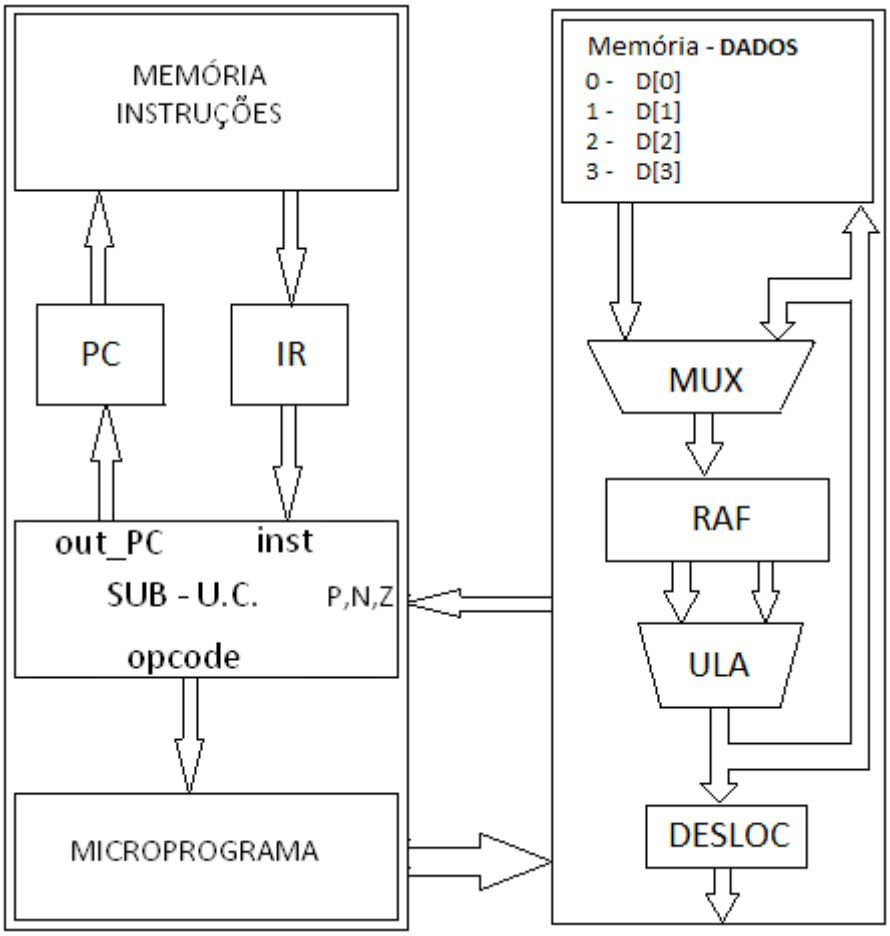
Implementação da U.C. como uma máquina de estados finitos F.S.M.

A seqüência de operações realizadas por um processador programável se resume em dois cilcos básicos de operação chamado de ciclo de instrução. O ciclo de instrução controlado por um relógio é composto por:

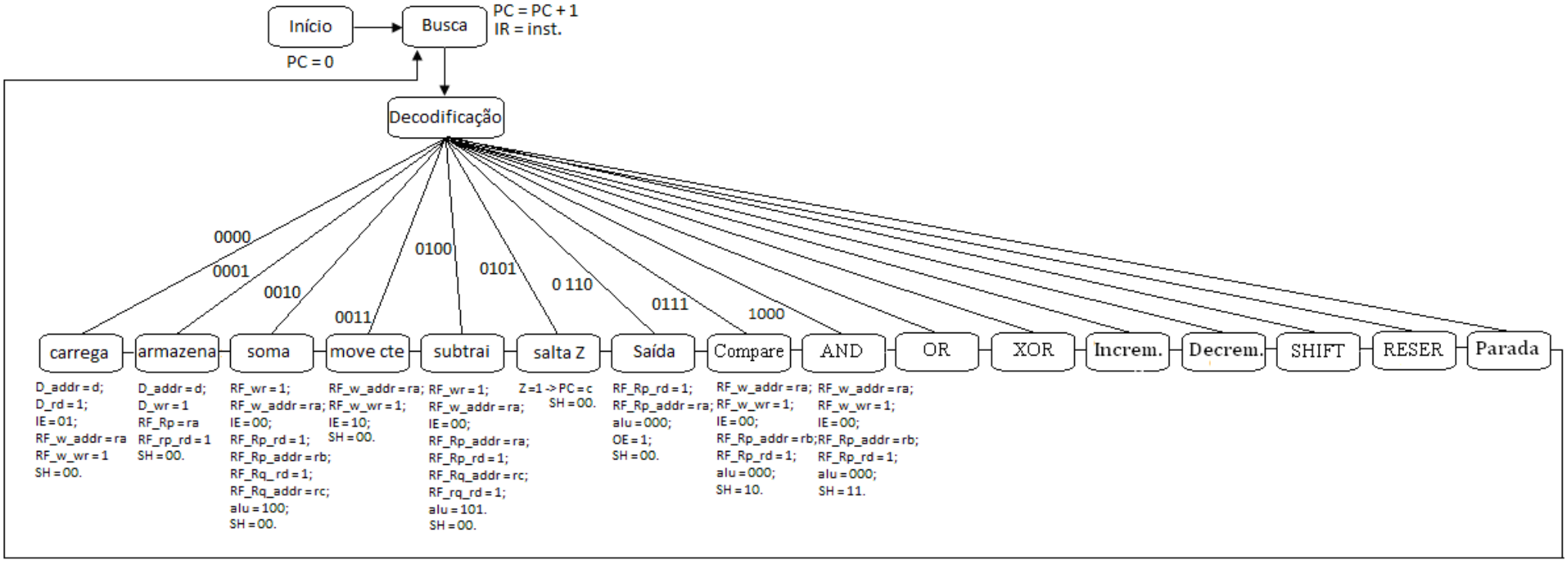
a) ciclo de busca da instrução – O ciclo de busca requer 2 estados a saber:
O primeiro estado é a busca da instrução na memória de programa, onde o PC endereça a memória e coloca a instrução na entrada do registrador de instrução;
O segundo estado a instrução é decodificada no decodificador de instrução e os sinais de controle formam a palavra de controle para o fluxo de dados.

b) ciclo de execução – O ciclo de execução é a realização da operação decodificada no decodificador de instrução.

O arquitetura do processador programável a seguir mostra a u.c. com o fluxo de dados.


MEMÓRIA
INSTRUÇÕES
PC
IR
out_PC
inst
SUB - U.C.
P,N,Z
opcode
MICROPROGRAMA
Memória - DADOS
0 - D[0]
1 - D[1]
2 - D[2]
3 - D[3]
MUX
RAF
ULA
DESLOC

O diagrama de estados a seguir mostra a busca da instrução, decodificação da instrução e execução.

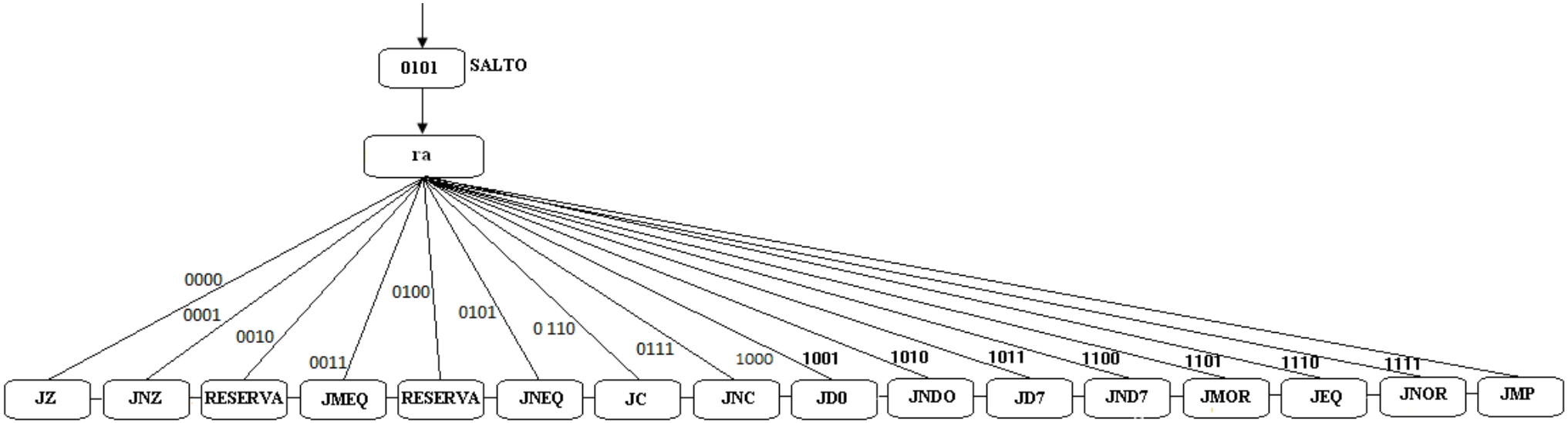


O quadro das palavras de controle geradas pela u.c. para a execução da instrução.

Instrução	Opcode	D_addr	D_rd	D_wr	RF_s	RF_w_addr	RF_w_wr	RF_Rp_rd	RF_Rp_addr	RF_Rq_rd	RF_Rq_addr	ALU	SH	OE
Carga	0	d	1	0	1	ra	1	0	xxxx	0	xxxx	xxx	xx	0
Armazena	1	d	0	1	x	xxxx	0	1	ra	0	xxxx	xxx	xx	0
Soma	2	x..x	0	0	0	ra	1	1	rb	1	rc	100	00	0
Transfer	3	x..x	0	0	2	ra	1	0	xxxx	0	xxxx	Xxx	xx	0
Subtrai	4	x..x	0	0	0	ra	1	1	rb	1	rc	101	00	0
Salto	5	x..x	0	0	0	xxxx	0	0	xxxx	0	xxxx	xxx	xx	x
Saída	6	x..x	0	0	0	xxxx	0	1	ra	0	xxxx	000	00	1
Entrada	7	x..x	0	0	3	ra	1	0	xxxx	0	xxxx	xxx	xx	0
Compare	8	x..x	0	0	0	xxxx	0	1	ra	1	Rb	xxx	xx	0

Instrução	Opcode	D_addr	D_rd	D_wr	RF_s	RF_w_addr	RF_w_wr	RF_Rp_rd	RF_Rp_addr	RF_Rq_rd	RF_Rq_addr	ALU	SH	OE
XOR	9	0	0	0	0	ra	1	1	rb	1	rc	011	00	0
AND	A	0	0	0	0	ra	1	1	rb	1	rc	001	00	0
OR	B	0	0	0	0	ra	1	1	rb	1	rc	010	00	0
INC	C	0	0	0	0	ra	1	1	ra	0	xxxx	111	00	0
DEC	D	0	0	0	0	ra	1	1	ra	0	xxxx	110	00	0
SHIFT	E	-	-	-	-	-	-	-	-	-	-	-	-	-
STOP	F	0	0	0	0	xxxx	0	0	xxxx	0	xxxx	xxx	xx	x

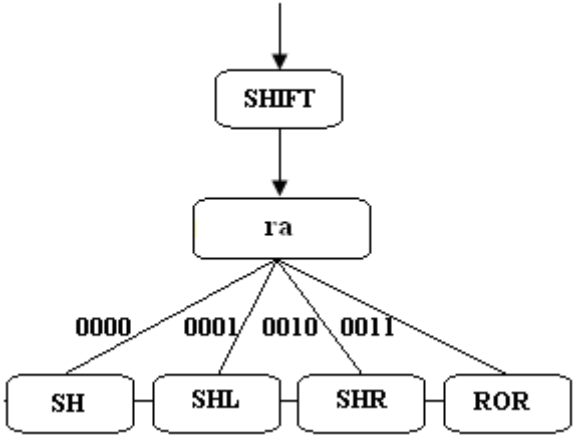
As instruções "salto" e "shift" vão ser tratadas separadamente, pois fazem parte da expansão das instruções. A instrução salto se expande em 12 instruções a saber. O diagrama de estados a seguir mostra as instruções expandidas.



Quadro das palavras de controle geradas pela u.c. nas instruções de saltos.
Na execução da instrução de salto tanto condicional como incondicional os bits da instrução $d_7...d_0$ são carregados no PC se satisfeita a condição ou se a instrução for incondicional.

Instrução	JZ	JNZ	JMEQ	JNEQ	JC	JNC	JD0	JND0	JD7	JND7	JMOR	JEQ	JNOR	JMP
ra	0	1	3	5	6	7	8	9	A	B	C	D	E	F
PC	d	d	d	d	D	d	d	d	d	d	d	d	d	d

A instrução de deslocamento "shift" também será expandida como a seguir:



Quadro de palavras de controle gerados pela u.c. na execução da instrução de deslocamento "shift".

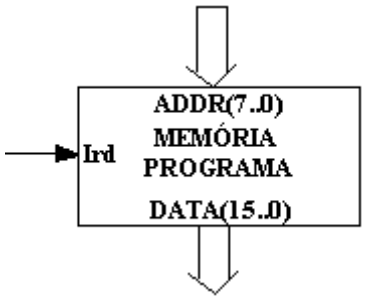
Instrução	ra	D_addr	D_rd	D_wr	RF_s	RF_w_addr	RF_w_wr	RF_Rp_rd	RF_Rp_addr	RF_Rq_rd	RF_Rq_addr	ALU	SH	OE
SH	0	0	0	0	0	rb	1	1	rc	0	xxxx	000	00	0
SHL	1	0	0	0	0	rb	1	1	rc	0	xxxx	000	01	0
SHR	2	0	0	0	0	rb	1	1	rc	0	xxxx	001	10	0
ROR	3	0	0	0	0	rb	1	1	rc	0	xxxx	111	11	0

Implementação das memórias de instruções e de dados.

Memória de programa ou de instruções – É uma memória apenas de leitura o qual contém o código completo de instruções numa seqüência de endereços consecutivos. O programa é criado pelo usuário e introduzido na memória de programa da forma:

Endereço	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0
2	0	0	1	0	0	0	0	0	0	1	1	1	1	0	1	1
3	0	1	0	1	1	1	0	1	1	0	0	0	1	0	1	1
4	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	0
5	1	0	1	1	1	0	1	0	0	0	1	1	0	1	1	0
6	0	0	0	0	0	1	0	1	1	1	1	0	1	1	1	1
7	1	1	0	0	1	0	0	1	0	1	0	0	1	1	0	0

Bloco memória de programa ou de instruções.

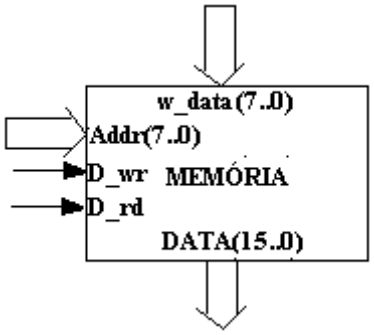


Em VHDL, fica:

```
-- Memória ROM DE 256 x 16 bits

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mem_inst_256x16 is

        port
        (
                endereco            : in unsigned (7 downto 0);
                saida               : out unsigned (15 downto 0);
                rd              : in std_logic);

end mem_inst_256x16;

architecture ROM of mem_inst_256x16 is
        type arranjo_memoria is array (NATURAL RANGE <>) of unsigned (15 downto 0);
```

```
        CONSTANT dados : arranjo_memoria(0 to 255):=
        ("0100000000000000","0011000100000011","0011001000000111","1000001000010000","0101110100001000","0010000000000001","1100000100000000","0101111100000011",
….., "0110000000000000");
begin

        saida <= dados (to_integer(endereco)) When rd = '1' else (others => 'Z');
end ROM;
```

Bloco memória de dados – Responsável por conter parâmetros do programa e permite ler e escrever dados. A sua capacidade de armazenamento é de 256 x 8bits e possui 8 linhas de endereçamento e de entrada e saída de dados, controladas por 2 sinais de leitura e escrita.



Em VHDL, fica:

```
-- Memoria 256 x 8

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

--USE ieee.std_logic_arith.all;

ENTITY memoria_dados IS PORT (
clk: IN std_logic; --clock
D_wr: IN std_logic; --write enable
D_addr: IN std_logic_vector(7 DOWNTO 0); --write address
W_data: IN std_logic_vector(7 DOWNTO 0); --input
D_rd: IN std_logic; --read enable ports A & B
R_data: OUT std_logic_vector(7 DOWNTO 0)); --output port A & B
END memoria_dados;
```

```
ARCHITECTURE Behavioral OF memoria_dados IS

SUBTYPE reg IS std_logic_vector(7 DOWNTO 0);
TYPE regArray IS array(0 TO 7) OF reg;
SIGNAL MEM: regArray; --register file contents

BEGIN
WritePort: PROCESS (clk)

BEGIN
IF (clk'EVENT AND clk = '1') THEN
IF (D_wr = '1') THEN
MEM(CONV_INTEGER(D_addr)) <= W_data;
END IF;
END IF;
END PROCESS;

ReadPortA: PROCESS (D_rd)
BEGIN
IF (D_rd = '1') then
R_data <= MEM(CONV_INTEGER(D_addr)); -- convert bit VECTOR to integer
ELSE
R_data <= (others => 'Z');
END IF;

END PROCESS;
END Behavioral;
```

Formato da instrução – A instrução têm 16bits subdivididos de 4 em 4 para informação de fonte e destino. Quando se utiliza a memória de dados são necessários 8bits para o endereçamento e a união das fontes rb e rc forma o d.

Opcode Instrução	Destino ra	Fonte rb	Fonte rc
$d_{15}..d_0$	$d_{11}...d_8$	$d_7,,d_4$	$d_3..d_0$

d(8bits) = rb(4bits) e rc(4bits).

Descrição de cada instrução.

1. LD(ra),(d) - Carrega da memória de dados o conteúdo endereçado pelo parâmetro d e transfere para o registrador de arquivo endereçado pelo endereço dado por ra.

Instrução	**Mnem**	**opcode**	**Operação**

Carrega	LD(ra) ← D(d)	0000	RF[ra] ← D(d)

2. ST(d),(ra) - Carrega na memória de dados o conteúdo endereçado pelo parâmetro d e transfere do registrador de arquivo endereçado pelo endereço dado por ra.

Instrução	Mnem	opcode	Operação
Armazena	ST(d) ← (ra)	0001	D(d) ← RF[ra]

3. ADD(ra),(rb),(rc) – Os conteúdos no registrador de arquivo dado pelos registradores b e c são somados na ULA e o resultado deve ser armazenado no próprio registrador de arquivo no endereço dado pelo registrador ra.

Instrução	Mnem	opcode	Operação
Carrega	ADD(ra),(rb),(rc)	0010	RF[ra] ← RF[rb] + RF[rc]

4. MVI(ra), #C - Carrega imediatamente o conteúdo c no registrador de arquivo no endereço fornecido pelo registrador ra.

Instrução	Mnem	opcode	Operação
Mover dado	MVI(ra) ← c	0011	RF[ra] ← c

5. SUB(ra),(rb),(rc) - Os conteúdos no registrador de arquivo dado pelos registradores b e c são subtraídos na ULA e o resultado deve ser armazenado no próprio registrador de arquivo no endereço dado pelo registrador ra.

Instrução	Mnem	opcode	Operação
Subtrai	SUB(ra),(rb),(rc)	0100	RF[ra] ← RF[Rb] – RF[rc]

6. JZ,(d) – Salta para o endereço dado em d se o resultado na saída da ULA for igual a zero.

Instrução	Mnem	opcode	ra	Operação
Salto se zero	JZ,(d)	0101	0000	PC ← (d)

7. JNZ,(d) - Salta para o endereço dado em d se o resultado na saída da ULA for diferente de zero.

Instrução	Mnem	opcode	ra	Operação
Salto se não zero	JNZ,(d)	0101	0001	PC ← (d)

8. JNEQ,(d) - Salta para o endereço dado em d se a comparação entre conteúdos A e B for menor ou igual.

Instrução	Mnem	opcode	ra	Operação

Salto menor ou igual	JNEQ,(d)	0101	0011	PC ← (d)

9. JNEQ,(d) - Salta para o endereço dado em d se a comparação entre conteúdos A e B for maior ou igual.

Instrução	Mnem	opcode	ra	Operação
Salto maior ou igual	JMEQ,(d)	0101	0101	PC ← (d)

10. JC,(d) - Salta para o endereço dado em d se a a operação realizada na ULA gerou um bit vai um "carry" .

Instrução	Mnem	opcode	ra	Operação
Salto se carry	JC,(d)	0101	0110	PC ← (d)

11. JNC,(d) - Salta para o endereço dado em d se a operação realizada na ULA não gerou um bit vai um "carry".

Instrução	Mnem	opcode	ra	Operação
Salto se não carry	JNC,(d)	0101	0111	PC ← (d)

12. JD0,(d) - Salta para o endereço dado em d se o bit D0 do conteúdo no registrador de arquivo dado pelo registrador ra e deslocado para o deslocador é igual a "1'.

Instrução	Mnem	opcode	ra	Operação
Salto se bit D0	JD0(ra) ← (d)	0101	1000	PC ← (d)

13. JND0,(d) - Salta para o endereço dado em d se o bit D0 do conteúdo no registrador de arquivo dado pelo registrador ra e deslocado para o deslocador é igual a "0'.

Instrução	Mnem	opcode	ra	Operação
Salto se não bit D0	JND0(ra) ← (d)	0101	1001	PC ← (d)

14. JD7,(d) - Salta para o endereço dado em d se o bit D7 do conteúdo no registrador de arquivo dado pelo registrador ra e deslocado para o deslocador é igual a "1'.

Instrução	Mnem	opcode	ra	Operação
Salto se bit D7	JD7(ra) ← (d)	0101	1010	PC ← (d)

15. JND7,(d) - Salta para o endereço dado em d se o bit D0 do conteúdo no registrador de arquivo dado pelo registrador ra e deslocado para o deslocador é igual a "0'.

Instrução	Mnem	opcode	ra	Operação
Salto se não bit D7	JND7(ra) ← (d)	0101	1011	PC ← (d)

16. JMOR,(d) - Salta para o endereço dado em d se a comparação entre os conteúdos nos registradores de arquivo dados pelos endereços dos registradores ra e rb, for maiorl.

Instrução	Mnem	Opcode	ra	Operação
Salta se maior	JMOR(ra) ← (d)	0101	1100	PC ← (d)

17. JEQ,(d) - Salta para o endereço dado em d se a comparação entre os conteúdos nos registradores de arquivo dados pelos endereços dos registradores ra e rb, for igual.

Instrução	Mnem	Opcode	ra	Operação
Salto se igual	JEQ(ra) ← (d)	0101	1101	PC ← (d)

18. JNOR,(d) - Salta para o endereço dado em d se a comparação entre os conteúdos nos registradores de arquivo dados pelos endereços dos registradores ra e rb, for menor.

Instrução	Mnem	opcode	ra	Operação
Salto se menor	JNOR(ra) ← (d)	0101	1110	PC ← (d)

19. JMP,(d) - Salta para o endereço dado em d.

Instrução	Mnem	opcode	ra	Operação
Salto incondicional	JMP(ra) ← (d)	0101	1111	PC ← (d)

20. OUT(ra) – Transfere para a saída de dados o conteúdo do registrador de arquivo endereçado pelo registrador ra.

Instrução	Mnem	opcode	Operação
Saída	OUT(ra)	0110	OUT ← [ra]

21. IN(ra),ext - Carrega no registrador de arquivo no endereço dado pelo registrador ra o conteúdo externo.

Instrução	Mnem	opcode	Operação
Entrada	IN(ra) ← ext	0111	RF[ra] ← ext

22. CMP(ra),(rb) – Compara os conteúdos endereçados pelos registradores ra e rb no registrador de arquivo.

Instrução	Mnem	Opcode	Operação
Compare	CMP(ra),(rb)	1000	[ra] : [rb]

23. XOR(ra),(rb),(rc) - Os conteúdos no registrador de arquivo dado pelos registradores b e c são realizadas a operação exclusivo-ou na ULA e o resultado deve ser armazenado no próprio registrador de arquivo no endereço dado pelo registrador ra.

Instrução	Mnem	opcode	Operação
XOR	XOR(ra),(rb),(rc)	1001	RF[ra] $\leftarrow$ RF[rb] $\oplus$ RF[rc]

24. AND(ra),(rb),(rc) - Os conteúdos no registrador de arquivo dado pelos registradores b e c são realizadas a operação "E" na ULA e o resultado deve ser armazenado no próprio registrador de arquivo no endereço dado pelo registrador ra.

Instrução	Mnem	opcode	Operação
E	AND(ra),(rb),(rc)	1010	RF[ra] $\leftarrow$ RF[rb] $\wedge$ RF[rc]

25. OR(ra),(rb),(rc) - Os conteúdos no registrador de arquivo dado pelos registradores b e c são realizadas a operação "OU" na ULA e o resultado deve ser armazenado no próprio registrador de arquivo no endereço dado pelo registrador ra.

Instrução	Mnem	opcode	Operação
OU	OU(ra),(rb),(rc)	1011	RF[ra] $\leftarrow$ RF[rb] $\vee$ RF[rc]

26. INC(ra) – Incrementa o conteúdo endereçado do registrador de arquivo endereçado pelo endereço dado por ra.

Instrução	Mnem	opcode	Operação
Incrementa	INC(ra)	1100	RF[ra] $\leftarrow$ RF[ra] + 1

27. DEC(ra) - Decrementa o conteúdo endereçado do registrador de arquivo endereçado pelo endereço dado por ra.

Instrução	Mnem	opcode	Operação
Decrementa	DEC(ra)	1101	RF[ra] $\leftarrow$ RF[ra] - 1

28. SH(rb),(rc) - Transfere o conteúdo do registrador de arquivo fornecido pelo registrador Rb e escreve o conteúdo no endereço fornecido pelo registrador rc no registrador de arquivo.

Instrução	Mnem	opcode	ra	Operação
Transfere	SH(ra),(rb)	1110	0000	RF[ra] $\leftarrow$ (rc)

29. SHL(ra) – Desloca um bit a esquerda o conteúdo endereçado pelo registrador ra e preenche o bit com zero e armazena o conteúdo no endereço dado por ra.

Instrução	Mnem	opcode	ra	Operação
Desloca esquerda	SHL(ra) ← (d)	1110	0001	RF[ra] ← (d)

30. SHR(ra) - Desloca um bit a direita o conteúdo endereçado pelo registrador ra e preenche o bit com zero e armazena o conteúdo no endereço dado por ra.

Instrução	Mnem	opcode	ra	Operação
Desloca direita	SHR(ra) ← (d)	1110	0010	RF[ra] ← (d)

31. ROR(ra) – Gira um bit a direita o conteúdo endereçado pelo registrador ra e preenche o bit com bit deslocado e armazena o conteúdo no endereço dado por ra.

Instrução	Mnem	opcode	ra	Operação
Gira a direita	ROR(ra) ← (d)	1110	0011	RF[ra] ← (d7)

32. HLT - Carrega da memória de dados o conteúdo endereçado pelo parâmetro d e transfere para o registrador de arquivo endereçado pelo endereço dado por ra.

Instrução	Mnem	opcode	Operação
Parada	HLT	1111	PC ← PC

O próximo passo após concluído o projeto é a simulação e para isso precisamos criar vários programas envolvendo todo o conjunto de instruções e situações de decisões. Usaremos o quartus II para essa finalidade.

Simulação – Vamos iniciar criando um programa com instruções.

Programa 1

Opcode	Mnemonica	Operação
0: 3005	MVI[0],05	RF[0] = #05;
1: 3106	MVI[1],06	RF[1] = #06;
2: 3207	MVI[2],07	RF[2] = #07;
3: 2001	ADD[0],[0],[1]	RF[0] = RF[0] + RF[1];
4: 2002	ADD[0],[0],[2]	RF[0] = RF[0] + RF[2]
5: 6000	OUT[0]	OUT = RF[0].

Programa 2

Opcode	Mnemônica	Operação
6: 1000	ST[0],[00]	D[0] = RF[0];
7: 0300	LD[3],[00]	RF[3] = D[0];
8: 4421	SUB[4],[2],[1]	RF[4] = RF[2] – RF[1];
9: 9504	XOR[5],[0],[4]	RF[5] = RF[0] ⊕ RF[4];

A: 6500	OUT = RF[5]	Saída = RF[5].

Programa 3

Opcode	Mnemônica	Operação
1: 3600	MVI[6],00	RF[6] = #00;
2: A665	AND[6],[6],[5]	RF[6] = RF[6] $\wedge$ RF[5];
3: 37F0	MVI[7],F0	RF[7] = #F0;
4: 6700	OUT = RF[7]	Saída = RF[7].

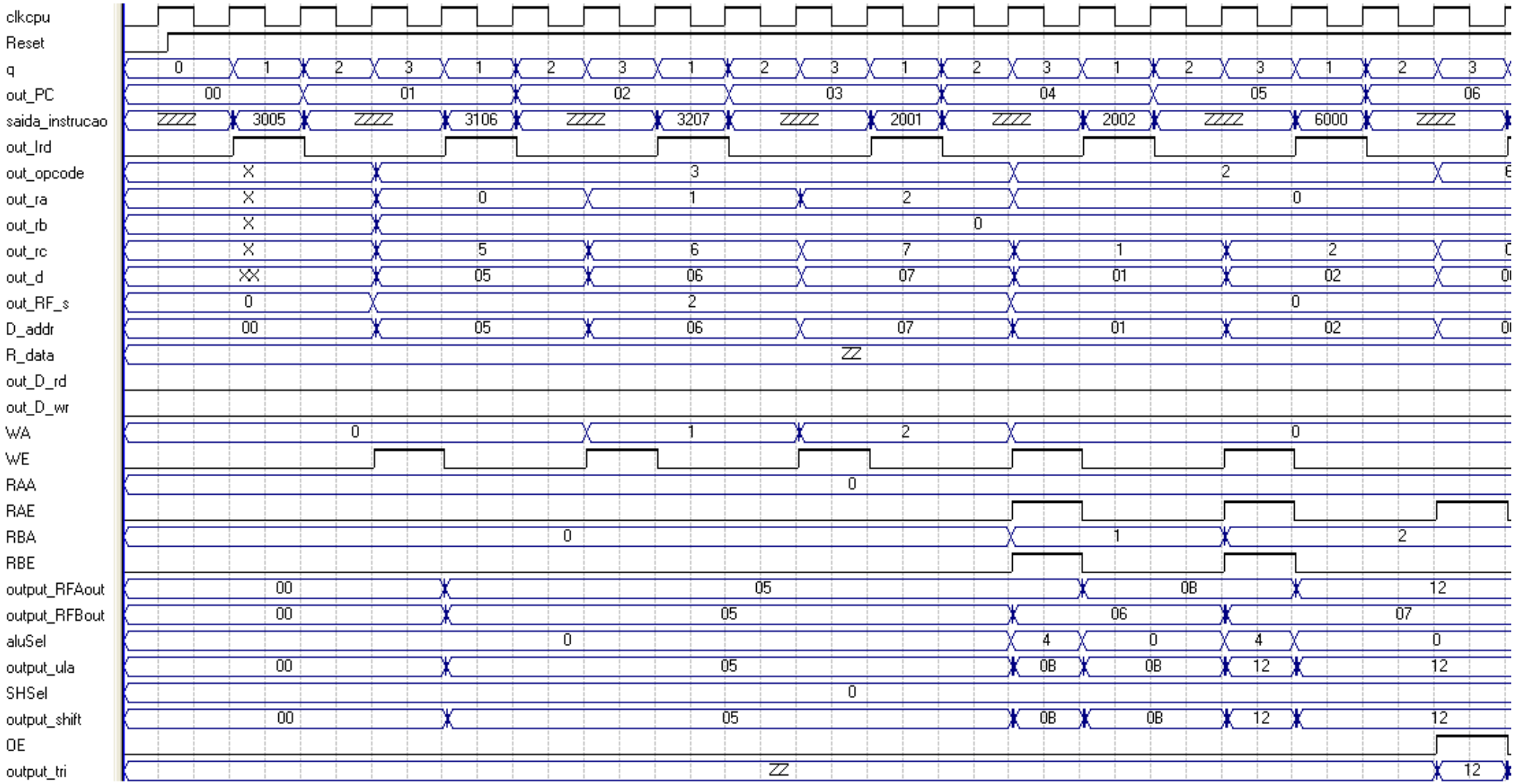
Programa 4 - Multiplicação

Opcode	Mnemônica	Operação
1: 4222	SUB[2],[2],[2]	RF[2] = RF[2] –RF[2];-- Zera Resultado
2: 300A	MVI[0],0A	RF[0] = #0A;-- Multiplicador
3: 3170	MVI[1],70	RF[1] = #70;--Multiplicando
4: 3304	MVI[3],04	RF[3] = #04;--Contador
5: E000	SH[0],[0]	RF[0] = RF[0];--Teste bit d0
6: 5908	JND0,[08]	PC = #d;--Salta DO = 1 para d;
7: 2221	ADD[2],[2],[1]	RF[2] = RF[2] + RF[1];--Resultado + multiplicando
8: E220	SHR[2]	C $\rightarrow$ RF[2];--Desloca bit a direita
9: E200	SHR[0]	0 $\rightarrow$ RF[0],--Desloca multiplicador
A: D300	DEC[3]	RF[3] = RF[3] – 1;Atualiza contador
B: 5105	JNZ,[05]	PC = #d;--Salta Z = 0 para d
C: 6200	OUT[2]	Saída = RF[2];--Apresenta resultado
D: F000	HLT	PC = PC;--Parada

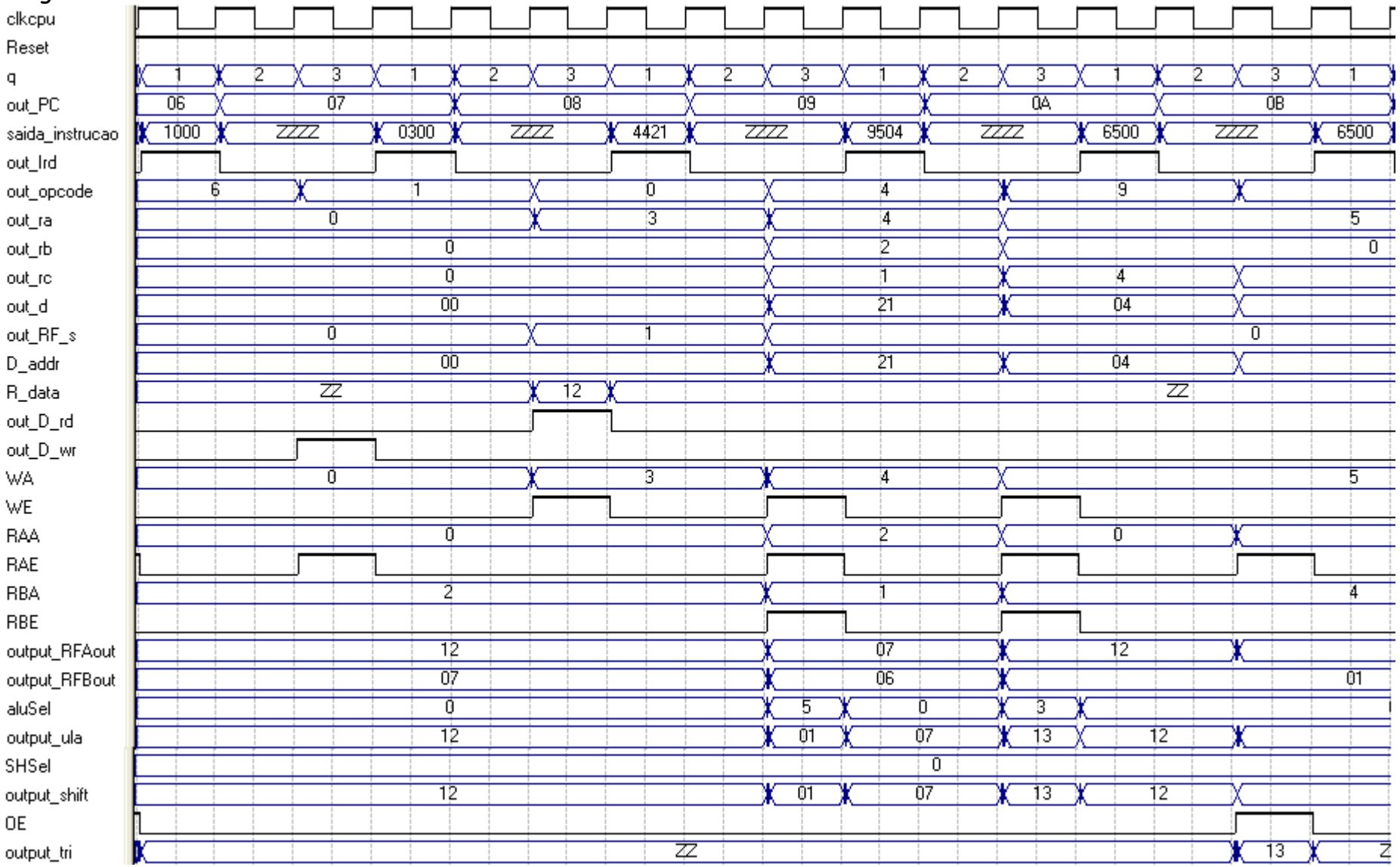
Programa 5 – Determinação de número por aproximação sucessivas

Opcode	Mnemônica	Operação
1: 301D	MVI[0],1D	RF[0] = #1D;--Número a ser determinado
2. 3180	MVI[1],80	RF[1] = #80;--Valor da primeira tentativa
3: 3208	MVI[2],08	RF[2] = #08;--Contador de tentativas igual a 8
4: 4333	SUB[3],[3],[3]	RF[3] = RF[3] – RF[3];--Zera resultado
5: 2313	ADD[3],[1],[3]	RF[3] = RF[1] + RF[3];--Primeira tentativa
6: 8300	CMP[3],[0]	RF[3] $\leq$ RF[0];--Comparação
7: 5309	JNEQ,[09]	True PC = 09;--Salta para endereço 09
8: 4331	SUB[3],[3],[1]	RF[3] = RF[3] – RF[1];--Ajusta valor
9: E210	SHR[1]	RF[1] $\div$ 2;--Desloca a direita
A: D200	DEC[2]	RF[2] = RF[2] – 1;--Decrementa contador
B: 5105	JNZ,[05]	True PC = 05;--Salta para endereço soma
C: 6300	OUT[3]	Saída = RF[3];--Apresenta
D; F000	HLT	Pára;--Parada do programa

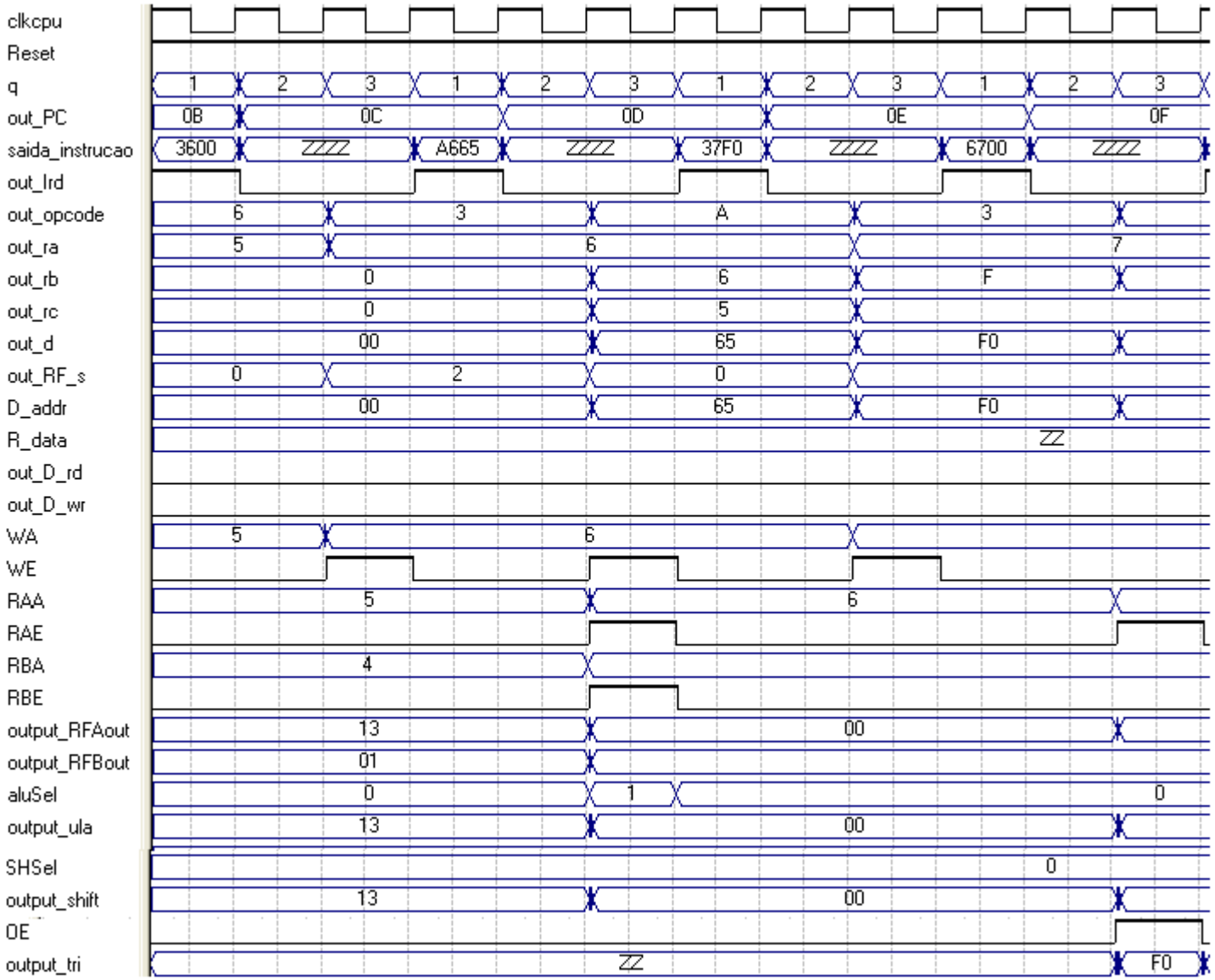
As formas de ondas a seguir mostram a evolução dos sinais gerados e a solução é apresentada na saída igual a 12;



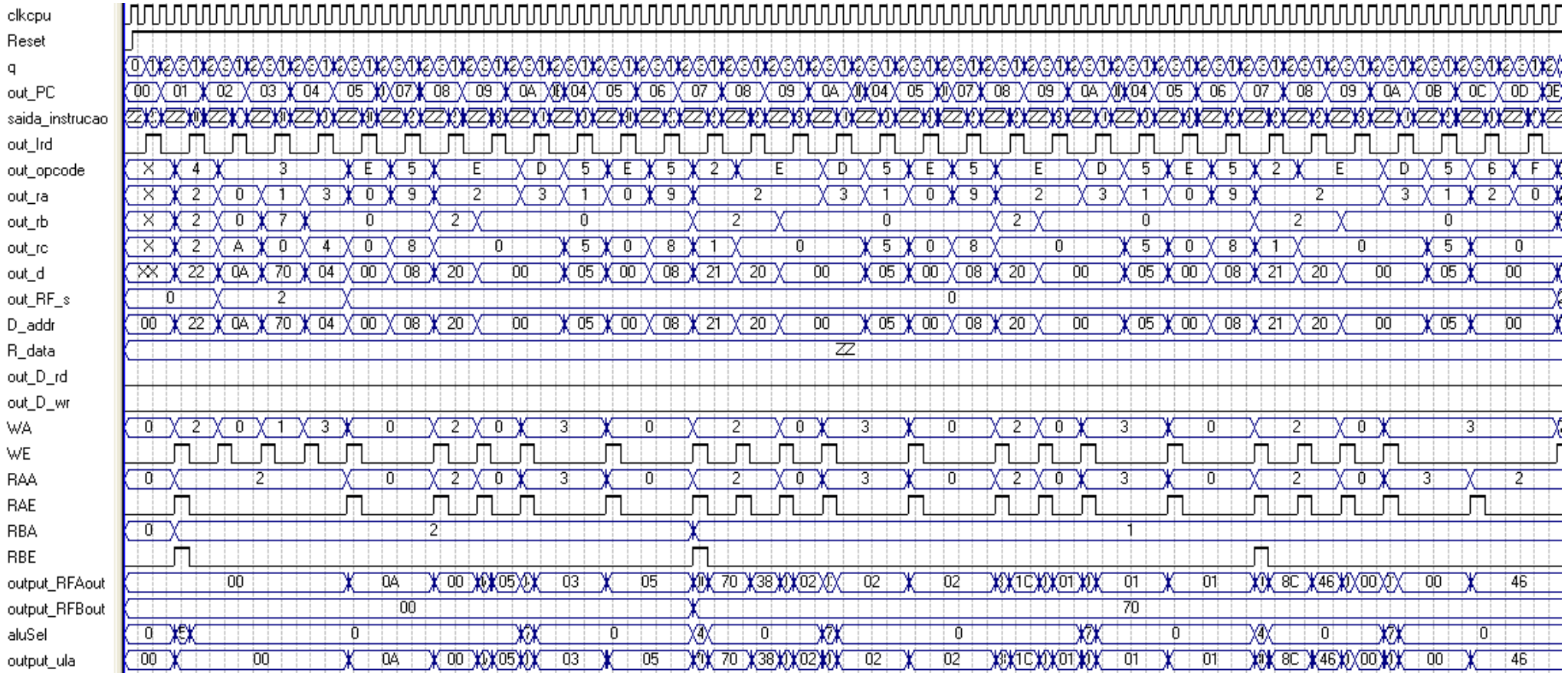
Programa 2:


clkcpu
Reset
q
out_PC
saida_instrucao
out_Ird
out_opcode
out_ra
out_rb
out_rc
out_d
out_RF_s
D_addr
R_data
out_D_rd
out_D_wr
WA
WE
RAA
RAE
RBA
RBE
output_RFAout
output_RFBout
aluSel
output_ula
SHSel
output_shift
OE
output_tri

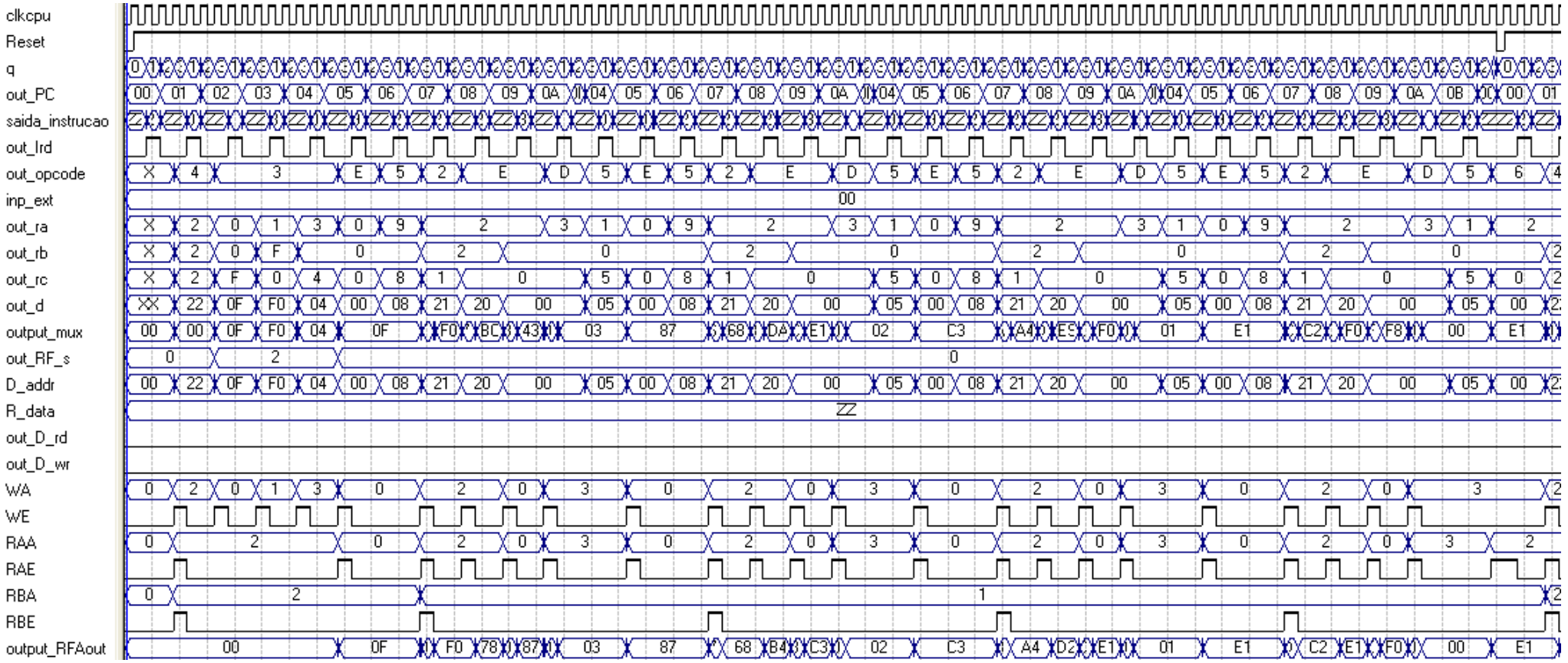
Programa 3:


clkcpu
Reset
q
out_PC
saida_instrucao
out_Ird
out_opcode
out_ra
out_rb
out_rc
out_d
out_RF_s
D_addr
R_data
out_D_rd
out_D_wr
WA
WE
RAA
RAE
RBA
RBE
output_RFAout
output_RFBout
aluSel
output_ula
SHSel
output_shift
OE
output_tri

Programa 4: Multiplicação entre 2 números 10 e 7 = 70 ou 46_H.



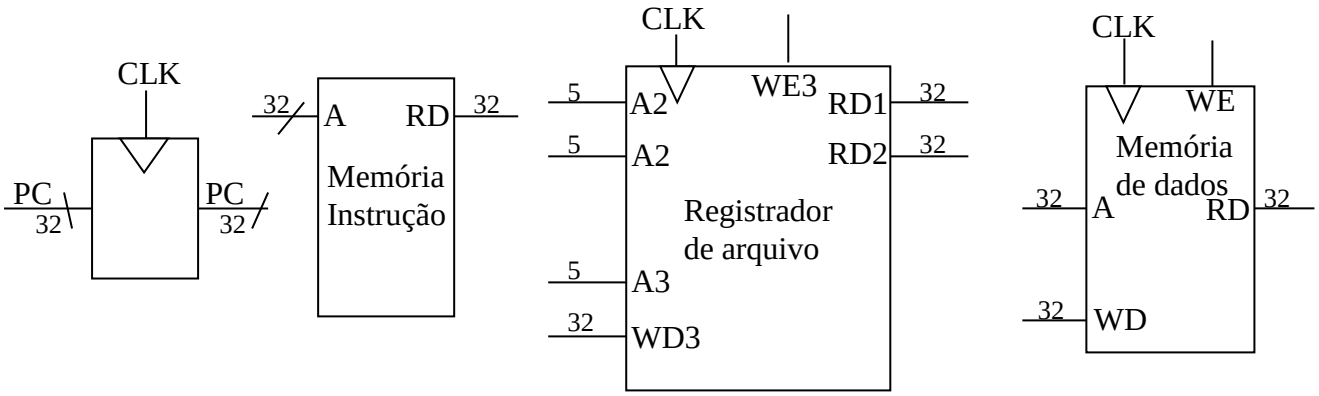
Multiplicação de 8 bits 15 x 15 = 225 ou E1.



MICROARQUITETURA

A microarquitetura consiste uma conexão entre lógica e arquitetura e é um específico arranjo de registradores. ULAs, máquina de estados finitos, memórias, e lógicas de seleções, enfim um arranjo de subsistemas digitais necessários na implementação de uma arquitetura. Algumas arquiteturas particulares serão estudados e especialmente a MIPS, criada por David Patterson, em 1984 (Stanford University). Diferentes microarquiteturas podem ter desempenhos diferentes.
Como estudado anteriormente o MIPS pode ser dividido em duas partes sendo controle e fluxo de dados (datapath). O fluxo de dados opera sobre palavras de dados e a sua estrutura contém, registrador de arquivo, memórias, ULAs, multiplexadores. No MIPS a arquitetura é de 32bits, tal que o fluxo de dados é de 32bits. A unidade de controle busca a instrução na memória de instrução e produz o controle de seleção para o multiplexador, habilita registrador, escreve sinais para controlar o fluxo de dados.
O estado arquitetural para o processador MIPS consiste do PC e de 33 registradores e para iniciar o estudo vamos introduzir elementos de estado para o estado da arquitetura.

ELEMENTOS DE ESTADO ARQUITETURAL – MIPS



Cada elemento de estado da microarquitetura MIPS tem:

1. O PC tem largura de 32bits,
2. O RF tem 32 registradores de 32 bits endereçados para escrita através das linhas WD3 e controle WE3 e para leitura dois portos de saída RD1 e RD2 endereçados respectivamente por 5 linhas de endereços A1 e A2.
3. Memória de instruções ou de programa somente de leitura de 2^{32} x 32 bits, endereçadas por A e saída por RD.
4. Memória de dados de 2^{32} x 32 bits, de leitura e escrita através da linha de controle WE e WD, acessadas pelas linhas de endereços A e saída RD.

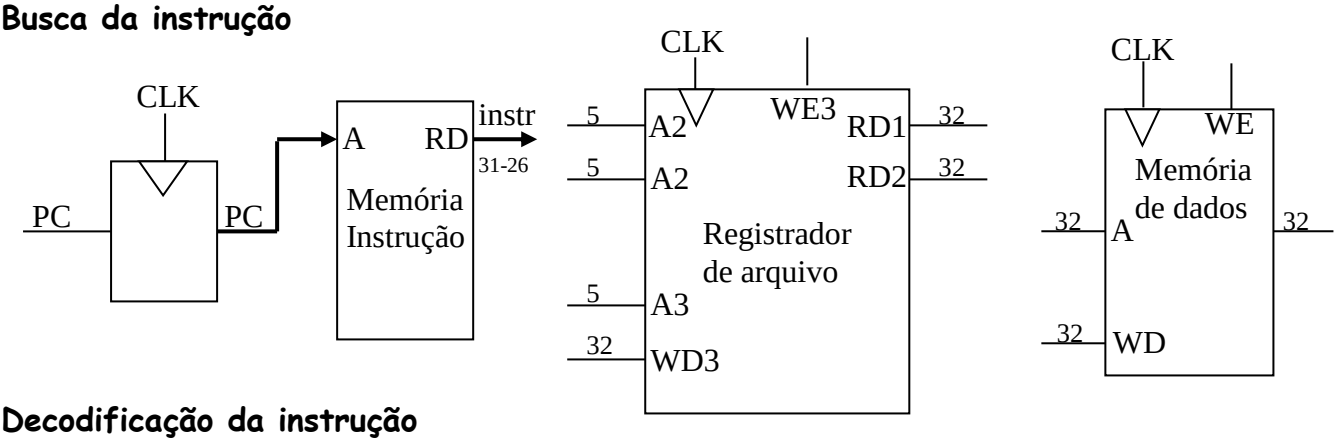
Para exemplificar a microarquitetura vamos a busca e execução da instrução lw. A arquitetura apresentada a seguir apresenta somente os blocos envolvidos na busca, decodificação e execução dessa instrução. O ciclo de instrução inicia com:

1. Ciclo de busca – A instrução tem 32bits de largura e é formatada conforme a seguir:

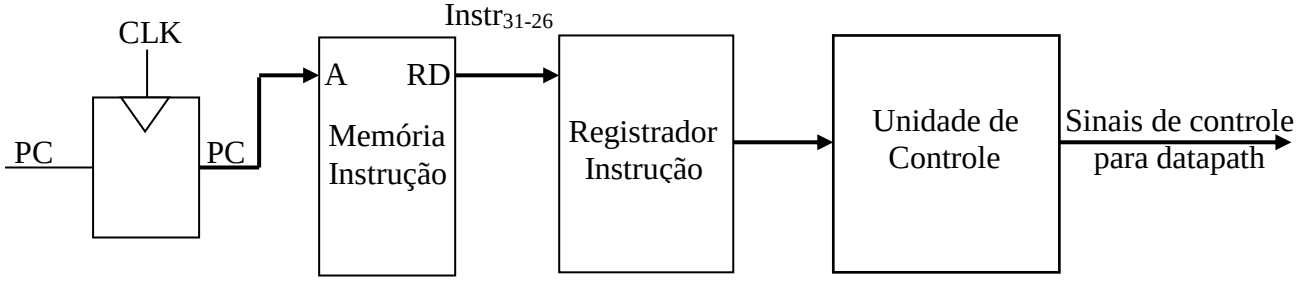
Opcode - 6 bits	rs	rt	imediato

$Inst_{31-26}$ – Opcode
rs_{25-21} - Fonte
rt_{20-16} - Destino
$imediato_{15-0}$ - Offset

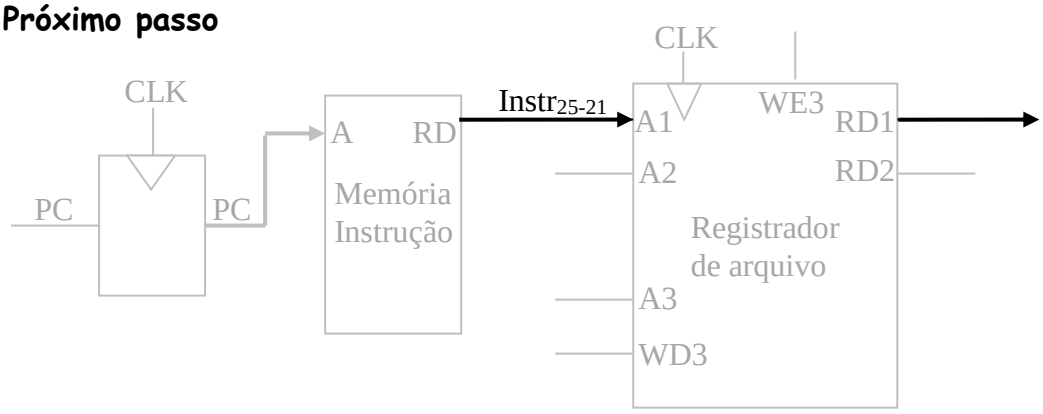
1.Busca da instrução



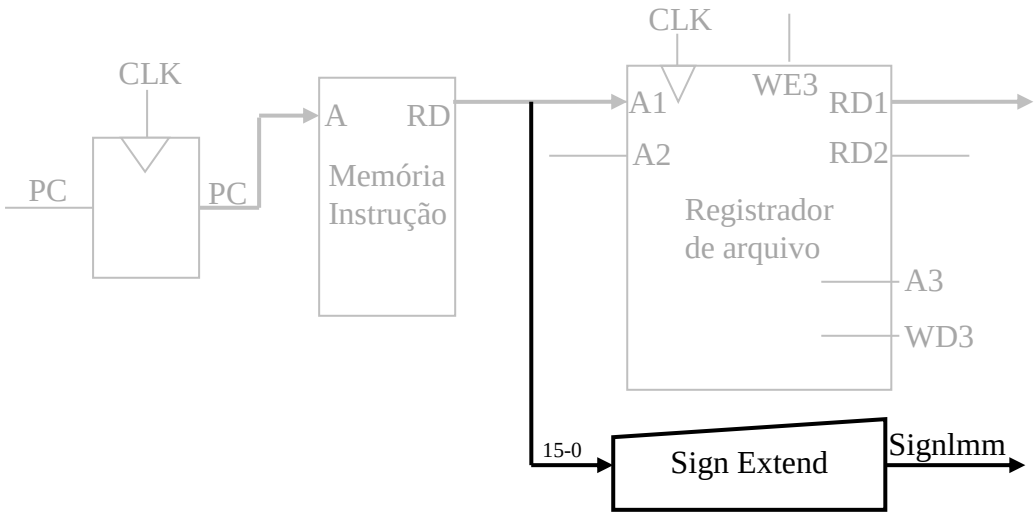
2.Decodificação da instrução



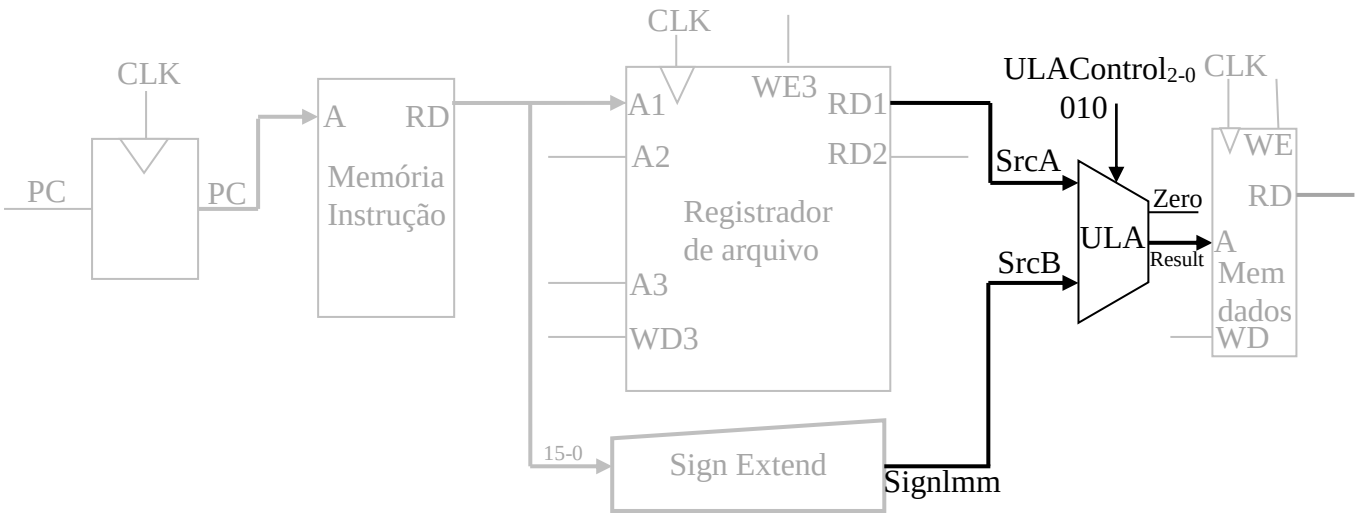
3. Próximo passo



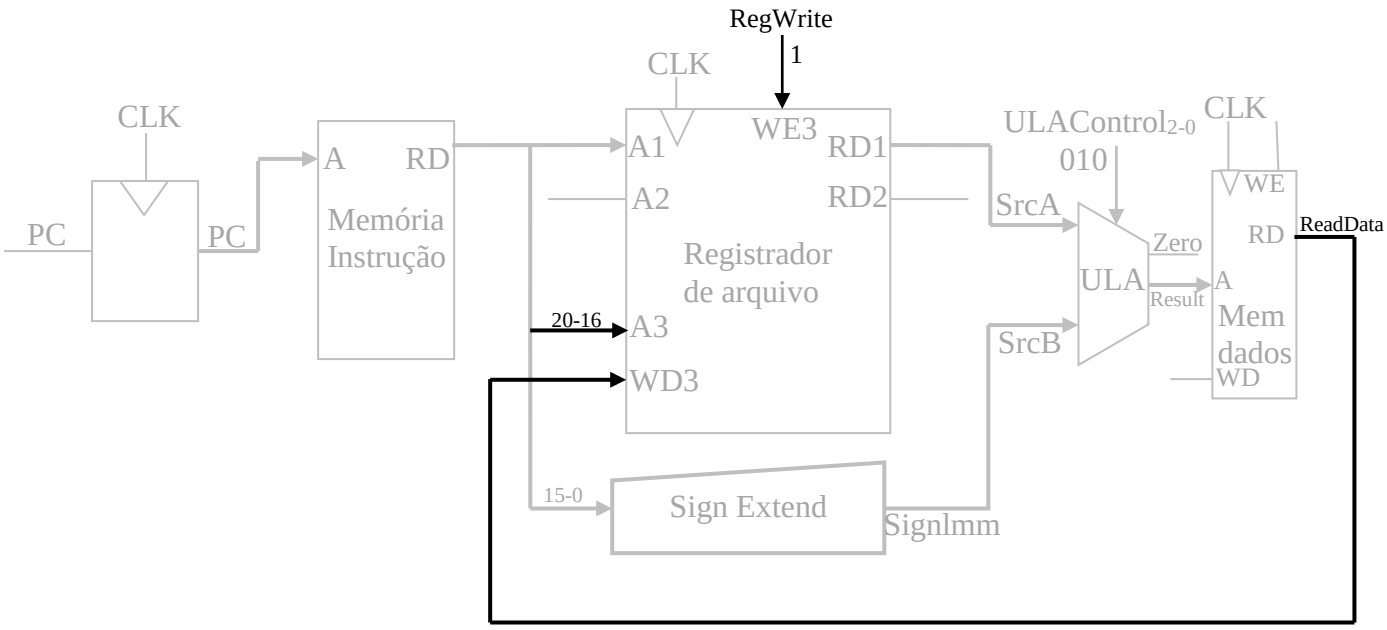
4. Próximo passo



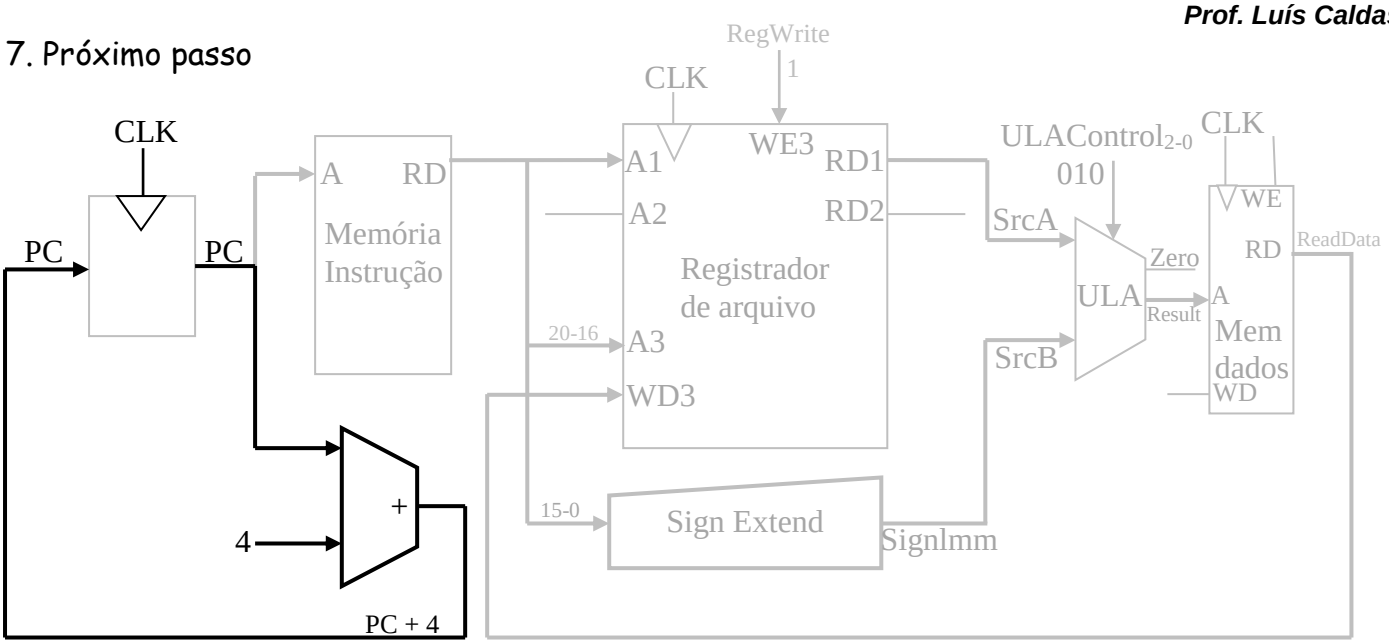
5. Próximo passo



6. Próximo passo



7. Próximo passo



Exemplo: Instrução Sw