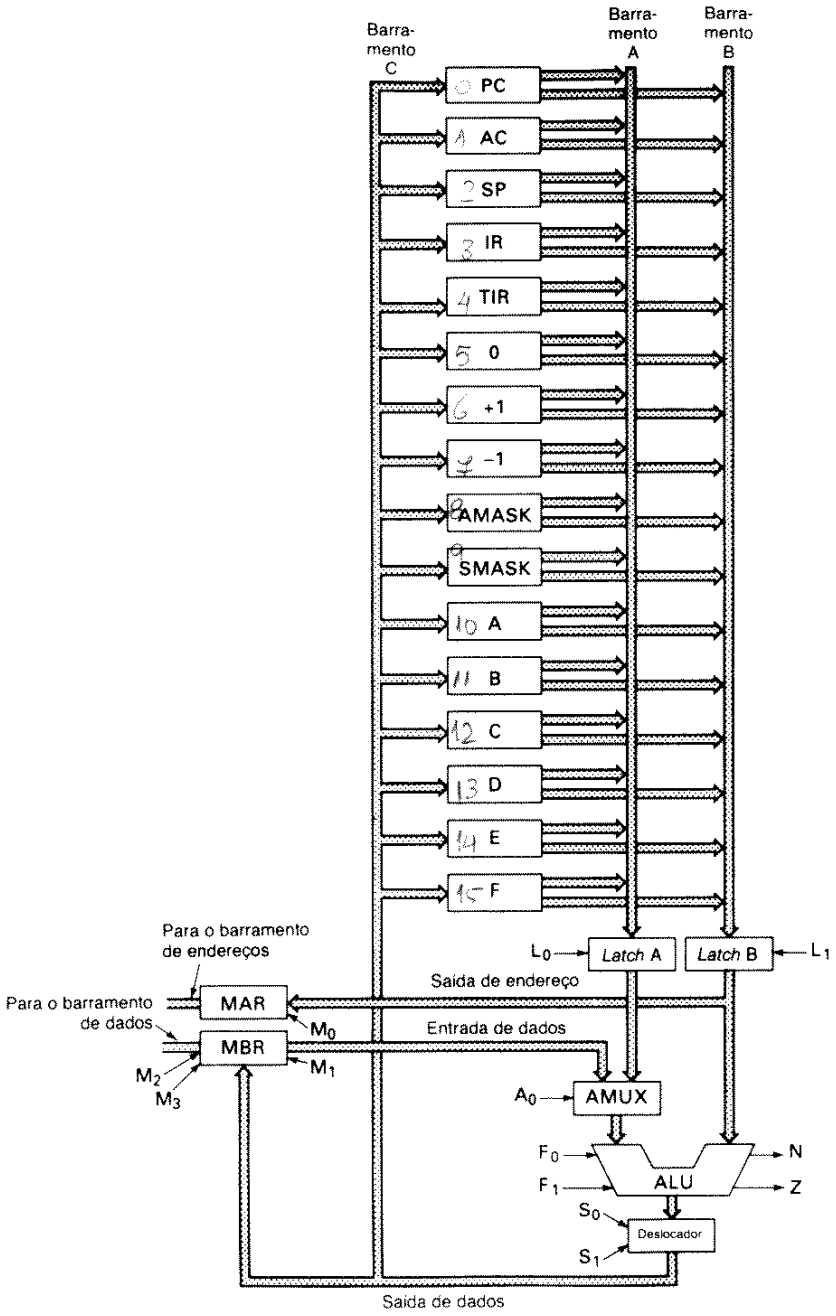

Caminho de Dados

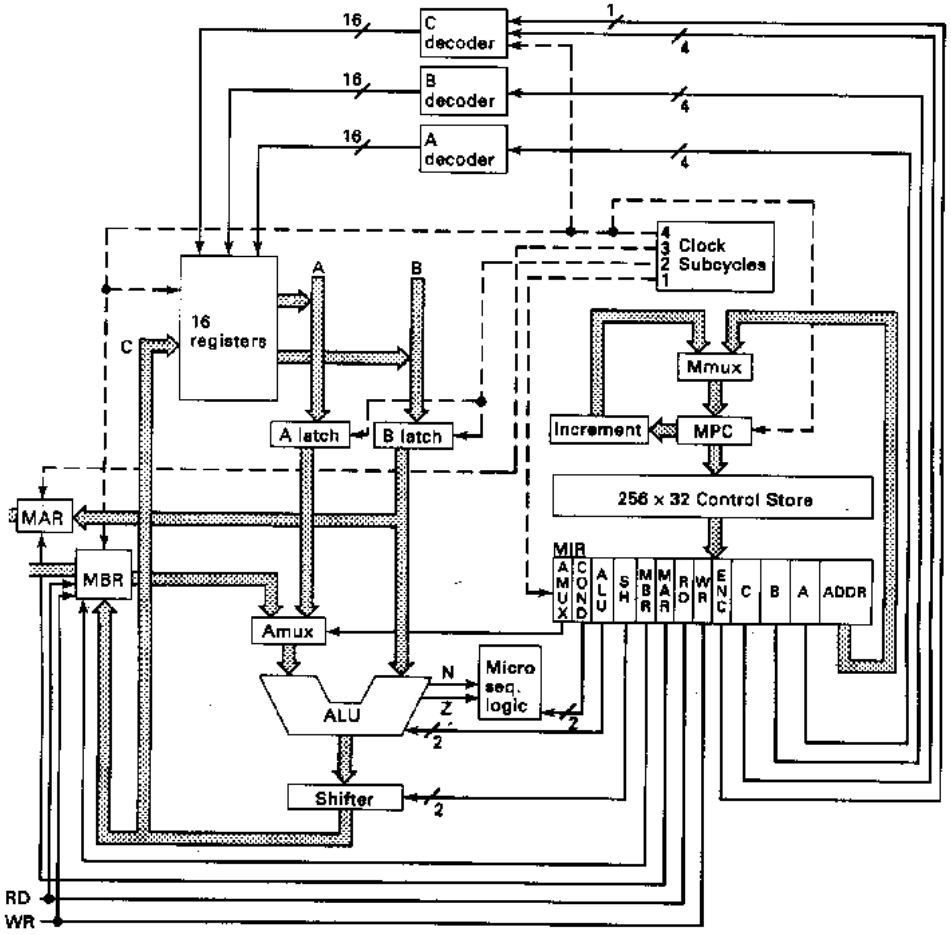

Fig. 4-10.

Micro Arquitetura de um Processador (MIC-1)

AMUX — controla a entrada esquerda da ALU: 0 = *latch* A, 1 = MBR
ALU — função da ALU: 0 = A + B, 1 = A AND B, 2 = A, 3 = $\bar{A}$
SH — função do deslocador: 0 = nenhum deslocamento, 1 = à direita, 2 = à esquerda
MBR — carrega MBR a partir do deslocador: 0 = não carrega, 1 = carrega MBR
MAR — carrega MAR a partir do *latch* B: 0 = não carrega, 1 = carrega MAR
RD — requisita leitura de memória: 0 = nenhuma leitura, 1 = carrega MBR a partir da memória
WR — requisita escrita na memória: 0 = nenhuma escrita, 1 = escreve o conteúdo de MBR na memória
ENC — controla armazenamento na memória de rascunho: 0 = não armazena, 1 = armazena
C — seleciona registrador para armazenamento se ENC = 1: 0 = PC, 1 = AC, etc.
B — seleciona fonte do barramento B: 0 = PC, 1 = AC, etc.
A — seleciona fonte do barramento A: 0 = PC, 1 = AC, etc.

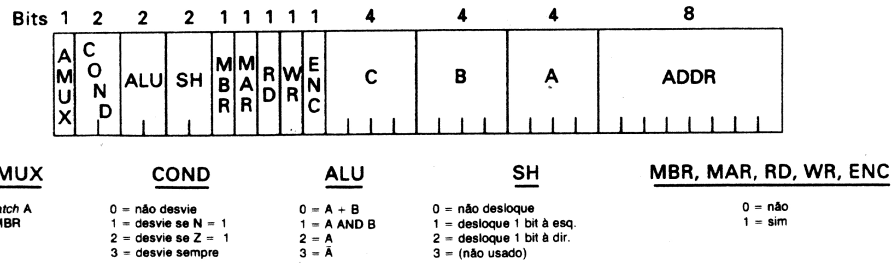

Fig. 4.9 O *layout* da microinstrução para controlar as vias de dados da Fig. 4.8.

Formato da Microinstrução (MIC-1)

Binary	Mnemonic	Instruction	Meaning
0000xxxxxxxxxxx	LODD	Load direct	$ac := m[x]$
0001xxxxxxxxxxx	STOD	Store direct	$m[x] := ac$
0010xxxxxxxxxxx	ADDD	Add direct	$ac := ac + m[x]$
0011xxxxxxxxxxx	SUBD	Subtract direct	$ac := ac - m[x]$
0100xxxxxxxxxxx	JPOS	Jump positive	**If** $ac \geq 0$ **then** $pc := x$
0101xxxxxxxxxxx	JZER	Jump zero	**If** $ac = 0$ **then** $pc := x$
0110xxxxxxxxxxx	JUMP	Jump	$pc := x$
0111xxxxxxxxxxx	LOCO	Load constant	$ac := x\ (0 \leq x \leq 4095)$
1000xxxxxxxxxxx	LODL	Load local	$ac := m[sp + x]$
1001xxxxxxxxxxx	STOL	Store local	$m[x + sp] := ac$
1010xxxxxxxxxxx	ADDL	Add local	$ac := ac + m[sp + x]$
1011xxxxxxxxxxx	SUBL	Subtract local	$ac := ac - m[sp + x]$
1100xxxxxxxxxxx	JNEG	Jump negative	**If** $ac < 0$ **then** $pc := x$
1101xxxxxxxxxxx	JNZE	Jump nonzero	**If** $ac \neq 0$ **then** $pc := x$
1110xxxxxxxxxxx	CALL	Call procedure	$sp := sp - 1;\ m[sp] := pc;\ pc := x$
1111000000000000	PSHI	Push indirect	$sp := sp - 1;\ m[sp] := m[ac]$
1111001000000000	POPI	Pop indirect	$m[ac] := m[sp];\ sp := sp + 1$
1111010000000000	PUSH	Push onto stack	$sp := sp - 1;\ m[sp] := ac$
1111011000000000	POP	Pop from stack	$ac := m[sp];\ sp := sp + 1$
1111100000000000	RETN	Return	$pc := m[sp];\ sp := sp + 1$
1111101000000000	SWAP	Swap ac, sp	$tmp := ac;\ ac := sp;\ sp := tmp$
11111100yyyyyyyy	INSP	Increment sp	$sp := sp + y\ (0 \leq y \leq 255)$
11111110yyyyyyyy	DESP	Decrement sp	$sp := sp - y\ (0 \leq y \leq 255)$

xxxxxxxxxxx is a 12-bit machine address; in column 4 it is called x.
yyyyyyy is an 8-bit constant; in column 4 it is called y.

Fig 4-14 - Conjunto de Instruções de Máquina (Macroinstruções MAC-1)

```
program ProdutoEscalar (output);

{Este programa inicializa dois vetores, x e y, de 20 elementos cada,
 então calcula o seu produto escalar:
    x[1] * y[1] + x[2] * y[2] + ... + x[20] * y[20]}

const max = 20;                                  {tamanho dos vetores}

type PequenoInt = 0..100;
     vet = array[1..max] of PequenoInt;

var k : integer;
    x, y: vec;

function pmul (a, b: PequenoInt): integer;
{Esta função multiplica seus dois parâmetros e retorna o produto.
 Ela realiza a multiplicação por adições sucessivas.}
var p, j: integer;
begin                                            {0: reserve espaço na pilha para p e j}
  if (a = 0) or (b = 0) then                     {1: se um dos dois for 0, resultado é o 0}
    pmul:= 0                                     {2: função retorna 0}
  else
    begin
    p := 0;                                      {3: inicializa p}
    for j := 1 to a do                           {4: some b a p a vezes}
      p := p + b;                                {5: faça a adição}
    pmul := p                                    {6: atribua resultado à função}
    end
end; {pmul}                                      {7: remova locais e retorne valor}

procedure escalar (var v: vet; var resp: integer);
{Calcula o produto escalar de v e x e o retorna em resp.}
var soma, i: integer;
begin                                            {8: reserve espaço na pilha para soma e i}
  soma := 0;                                     {9: soma vai acumular o produto escalar}
  for i:= 1 to max do                            {10: loop para todos os elementos}
    soma := soma + pmul (x[i], v[i]);            {11: acumule um termo}
  resp := soma                                   {12: copie o resultado para resp}
end; {escalar}                                   {13: remova soma e i e retorne}

begin                                            {14: reserve espaço para k, x e y}
  for k := 1 to max do                           {15: inicialização do loop}
    begin
      x[k] := k;                                 {16: inicialize x}
      y[k] := pmul (2, k) +1                     {17: inicialize y}
    end;
  escalar (y, k);                                {18: chame escalar}
  writeln (k)                                    {19: imprima os resultados}
end.
```

Programa exemplo em Pascal

```
            K = 4020        /DEFINA ALGUNS SÍMBOLOS
            X = 4000
            Y = 3980
            A = 4
            B = 3
            P = 1
            J = 0
            V = 5
            RESP = 4
            SOMA = 1
            I = 0
                                                    PRINCIPAL
            JUMP    MAIN    /COMECE NO PROGRAMA PRINCIPAL
PMUL:       DESP    2       /0
            LODL    A       /1
            JNZE    ANOTZ   /DESVIE SE A <> 0
            LOCO    0       /2
            JUMP    DONE    /RETORNE 0
ANOTZ:      LODL    B       /AC := B
            JNZE    BNOTZ   /DESVIE SE B <> 0
            LOCO    0       /2
            JUMP    DONE    /RETORNE 0
BNOTZ:      LOCO    0       /3
            STOL    P       /P := 0
            LOCO    1       /4
            STOL    J       /J := 1
            LODL    A       /0 LOOP PODE SER EXECUTADO?
            JNEG    L2      /1 < 0, NÃO FAÇA O LOOP
            JZER    L2      /A = 0, NÃO FAÇA O LOOP
L1:         LODL    P       /5
            ADDL    B       /AC := P + B
            STOL    P       /P := P + B
            LOCO    1       /TESTE NO FINAL DO LOOP
            ADDL    J       /AC := J + 1
            STOL    J       /J := J + 1
            SUBL    A       /AC := J - A
            JNEG    L1      /DESVIE SE J < A
            JZER    L1      /DESVIE SE J = A
L2:         LODL    P       /6
DONE:       INSP    2       /7
            RETN            /RETORNE

ESCALAR:    DESP    2       /8
            LOCO    0       /9
            STOL    SOMA    /SOMA := 0
            LOCO    1       /10
            STOL    I       /I := 1
L3:         LOCO    X-1     /11
            ADDL    I       /AC := X + I - 1
            PSHI            /EMPILHE X[I]
            LODL    V       /AC := ENDEREÇO DO VETOR
            ADDL    I       /AC := V + I
            SUBD    C1      /V COMEÇA EM 1, NÃO 0
            PSHI            /EMPILHE V[I]
            CALL    PMUL    /PMUL(X[I], V[I])

            INSP    2       /REMOVA PARÂMETROS
            ADDL    SOMA    /AC := SOMA + PMUL(...)
            STOL    SOMA    /SOMA := SOMA + PMUL(...)
            LOCO    1       /TESTE NO FINAL DO LOOP
            ADDL    I       /AC := I + 1
            STOL    I       /I := I + 1
            SUBD    C20     /AC := I - MAX
            JNEG    L3      /DESVIE SE I < MAX
            JZER    L3      /DESVIE SE I = MAX
            LODL    SOMA    /12
            PUSH            /EMPILHE SOMA
            LODL    RESP    /AC := ENDEREÇO DE RESP
            POPI            /RESP := SOMA
            INSP    2       /13
            RETN            /RETORNE

PRINCIPAL:  DESP    41      /14
            LOCO    1       /15
            STOD    K       /K NÃO É LOCAL
L4:         LODD    K       /16
            PUSH            /EMPILHE K
            LOCO    X - 1   /AC := (ENDEREÇO DE X[1])-1
            ADDD    K       /AC := X + K - 1
            POPI            /X[K] := K
            LOCO    2       /17
            PUSH            /PREPARE PMUL (2,...)
            LODD    K       /PREPARE PMUL (2, K)
            PUSH            /AMBOS OS PARÂMETROS EMPILHADOS
            CALL    PMUL    /PMUL(2,K)
            INSP    2       /REMOVA PARÂMETROS
            ADDD    C1      /AC := 2 * K + 1
            PUSH            /PREPARE Y[K] := 2 * K + 1
            LOCO    Y - 1   /AC := (ENDEREÇO DE Y[1])-1
            ADDD    K       /AC := Y + K - 1
            POPI            /Y[K] := 2 * K + 1
            LOCO    1       /TESTE NO FINAL DO LOOP
            ADDD    K       /AC := K + 1
            STOD    K       /K := K + 1
            SUBD    C20     /AC := K - MAX
            JNEG    L4      /DESVIE SE K < 0
            JZER    L4      /DESVIE SE K = MAX
            LOCO    Y       /18
            PUSH            /EMPILHE O ENDEREÇO DE Y
            LOCO    K       /AC := ENDEREÇO DE K
            PUSH            /EMPILHE ELE TAMBÉM
            CALL    ESCALAR /CHAMADA DE PROCEDIMENTO
            INSP    2       /REMOVA PARÂMETROS
            LODD    K       /19
            PUSH            /PREPARE WRITELN(K)
            CALL    OUTNUM1 /ROTINA DE BIBLIOTECA
            INSP    1       /REMOVA PARAM
            CALL    STOP    /FIM DA TAREFA

C1:         1               /CONSTANTE 1
C20:        20              /CONSTANTE 20
```

Programa exemplo em MAC-1

```
0: mar := pc; rd;                                   {main loop}
1: pc := pc + 1; rd;                                {increment pc}
2: ir := mbr; if n then goto 28;                    {save, decode mbr}
3: tir := lshift(ir + ir); if n then goto 19;
4: tir := lshift(tir); if n then goto 11;           {000x or 001x?}
5: alu := tir; if n then goto 9;                    {0000 or 0001?}

6: mar := ir; rd;                                   {0000 = LODD}
7: rd;
8: ac := mbr; goto 0;

9: mar := ir; mbr := ac; wr;                        {0001 = STOD}
10: wr; goto 0;

11: alu := tir; if n then goto 15;                  {0010 or 0011?}

12: mar := ir; rd;                                  {0010 = ADDD}
13: rd;
14: ac := mbr + ac; goto 0;

15: mar := ir; rd;                                  {0011 = SUBD}
16: ac := ac + 1; rd;                               {Note: x - y = x + 1 + not y}
17: a := inv(mbr);
18: ac := ac + a; goto 0;

19: tir := lshift(tir); if n then goto 25;          {010x or 011x?}
20: alu := tir; if n then goto 23;                  {0100 or 0101?}

21: alu := ac; if n then goto 0;                    {0100 = JPOS}
22: pc := band(ir, amask); goto 0;                  {perform the jump}

23: alu := ac; if z then goto 22;                   {0101 = JZER}
24: goto 0;                                         {jump failed}

25: alu := tir; if n then goto 27;                  {0110 or 0111?}
26: pc := band(ir, amask); goto 0;                  {0110 = JUMP}

27: ac := band(ir, amask); goto 0;                  {0111 = LOCO}

28: tir := lshift(ir + ir); if n then goto 40;      {10xx or 11xx?}
29: tir := lshift(tir); if n then goto 35;          {100x or 101x?}
30: alu := tir; if n then goto 33;                  {1000 or 1001?}

31: a := ir + sp;                                   {1000 = LODL}
32: mar := a; rd; goto 7;

33: a := ir + sp;                                   {1001 = STOL}
34: mar := a; mbr := ac; wr; goto 10;

35: alu := tir; if n then goto 38;                  {1010 or 1011?}

36: a := ir + sp;                                   {1010 = ADDL}
37: mar := a; rd; goto 13;

38: a := ir + sp;                                   {1011 = SUBL}
39: mar := a; rd; goto 16;
```

Microprograma – MIC-1

```
40: tir := lshift(tir); if n then goto 46;          {110x or 111x?}
41: alu := tir; if n then goto 44;                  {1100 or 1101?}

42: alu := ac; if n then goto 22;                   {1100 = JNEG}
43: goto 0;

44: alu := ac; if z then goto 0;                    {1101 = JNZE}
45: pc := band(ir, amask); goto 0;

46: tir := lshift(tir); if n then goto 50;

47: sp := sp + (-1);                                {1110 = CALL}
48: mar := sp; mbr := pc; wr;
49: pc := band(ir, amask); wr; goto 0;

50: tir := lshift(tir); if n then goto 65;          {1111, examine addr}
51: tir := lshift(tir); if n then goto 59;
52: alu := tir; if n then goto 56;

53: mar := ac; rd;                                  {1111000 = PSHI}
54: sp := sp + (-1); rd;
55: mar := sp; wr; goto 10;

56: mar := sp; sp := sp + 1; rd;                    {1111001 = POPI}
57: rd;
58: mar := ac; wr; goto 10;

59: alu := tir; if n then goto 62;

60: sp := sp + (-1);                                {1111010 = PUSH}
61: mar := sp; mbr := ac; wr; goto 10;

62: mar := sp; sp := sp + 1; rd;                    {1111011 = POP}
63: rd;
64: ac := mbr; goto 0;

65: tir := lshift(tir); if n then goto 73;
66: alu := tir; if n then goto 70;

67: mar := sp; sp := sp + 1; rd;                    {1111100 = RETN}
68: rd;
69: pc := mbr; goto 0;

70: a := ac;                                        {1111101 = SWAP}
71: ac := sp;
72: sp := a; goto 0;

73: alu := tir; if n then goto 76;

74: a := band(ir, smask);                           {1111110 = INSP}
75: sp := sp + a; goto 0;

76: a := band(ir, smask);                           {1111111 = DESP}
77: a := inv(a);
78: a := a + 1; goto 75;
```

Fig. 4-16. (cont.)

Microprograma (MIC-1)

Binário	Mnemônico	Instrução	Significado
0000	ADD	Adição	*r1* := *r1* + *r2*
0001	AND	AND booleano	*r1* := *r1* AND *r2*
0010	MOVE	Move registrador	*r1* := *r2*
0011	COMPL	Complemento	*r1* := *inv (r2)*
0100	LSHIFT	Deslocamento à esquerda	*r1* := *lshift(r2)*
0101	RSHIFT	Deslocamento à direita	*r1* := *rshift (r2)*
0110	GETMBR	Armazena MBR em registrador	*r1* := *mbr*
0111	TEST	Testa registrador	**If** *r2* < 0 **then** *n* := *true*; **If** *r2* = 0 **then** *z* := *true*
1000	BEGRD	Início de leitura	*mar* := *r1*; *rd*
1001	BEGWR	Início de escrita	*mar* := *r1*; *mbr* := *r2*; *wr*
1010	CONRD	Continua leitura	*rd*
1011	CONWR	Continua escrita	*wr*
1100		(Não utilizado)	
1101	NJUMP	Desvie se N = 1	**If** *n* **then goto** *r*
1110	ZJUMP	Desvie se Z = 1	**If** *z* **then goto** *r*
1111	UJUMP	Desvio incondicional	**goto** *r*

r = 16 * *r1* + *r2*

Fig. 4.17 Os *opcodes* do Mic-2.

Opcodes do MIC-2

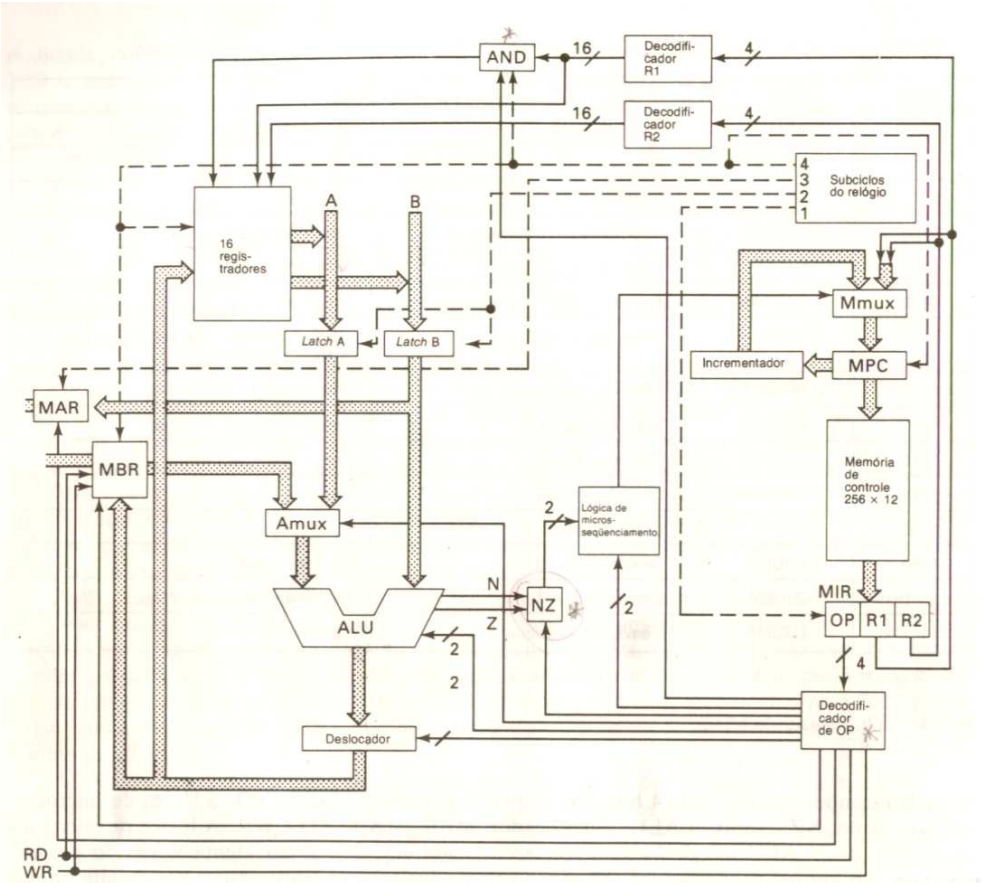

Fig. 4.18 Uma microarquitetura com microinstruções verticais.

Micro arquitetura vertical (MIC-2)

		Linhas de controle												
			ALUL		SHL		AMUX		MAR		RD		MSLH	
Opcode da microinstrução		ALUH	↓	SHH	↓	NZ	↓	AND	↓	MBR	↓	WR	↓	MSLL
0	ADD					+		+						
1	AND		+			+		+						
2	MOVE	+				+		+						
3	COMPL	+	+			+		+						
4	LSHIFT	+		+		+		+						
5	RSHIFT	+			+	+		+						
6	GETMBR	+				+	+	+						
7	TEST	+				+								
8	BEGRD	+							+		+			
9	BEGWR	+							+	+		+		
10	CONRD	+									+			
11	CONWR	+										+		
12														
13	NJUMP	+												+
14	ZJUMP	+											+	
15	UJUMP	+											+	+

Fig. 4.19 Os sinais de controle para cada *opcode* de microinstrução. Um mais significa que o sinal está ativo; um espaço significa que ele está desativado.

MIC-2 OPCODE Decoder

```
0:  mar := pc ; rd ;
1:  rd ;
    pc := pc + 1;
2:  ir := mbr ;
    tir := lshift (ir );
    if n then goto 28;
3:  tir := lshift (tir );
    if n then goto 19;
4:  tir := lshift (tir );
    if n then goto 11;
5:  alu := tir ;
    if n then goto 09;

6:  mar := ir ; rd ; {LODD}
7:  rd ;
8:  ac := mbr ;
    goto 0;

9:  mar := ir ; mbr := ac ; wr ; {STOD}
10: wr ;
    goto 0;

11: alu := tir ;
    if n then goto 15;

12: mar := ir ; rd ; {ADDD}
13: rd ;
14: a := mbr ;
    ac := ac + a ;
    goto 0;

15: mar := ir ; rd ; {SUBD}
16: rd ;
99: ac := ac + 1;
17: a := mbr ;
    a := inv (a );
18: ac := ac + a ;
    goto 0;

19: tir := lshift (tir );
    if n then goto 25;
20: alu := tir ;
    if n then goto 23;

21: alu := ac ; {JPOS}
    if n then goto 0;
22: pc := ir ;
    pc := band (pc , amask );
    goto 0;

23: alu := ac ; {JZER}
    if z then goto 22;
24: goto 0;

25: alu := tir ;
    if n then goto 27;
26: pc := ir ; {JUMP}
    pc := band (pc , amask );
    goto 0;

27: ac := ir ; {LOCO}
    ac := band (ac , amask );
    goto 0;

28: tir := lshift (tir );
    if n then goto 40;
29: tir := lshift (tir );
    if n then goto 35;
30: alu := tir ;
    if n then goto 33;

31: a := ir ; {LODL}
    a := a + sp ;
32: mar := a ; rd ;
    rd ;
    ac := mbr ;
    goto 0;

33: a := ir ; {STOL}
    a := a + sp ;
34: mar := a ; mbr := ac ; wr ;
    wr ;
    goto 0;

35: alu := tir ;
    if n then goto 38;

36: a := ir ; {ADDL}
    a := a + sp ;
37: mar := a ; rd ;
    rd ;
    a := mbr ;
    ac := ac + a ;
    goto 0;

38: a := ir ; {SUBL}
    a := a + sp ;
39: mar := a ; rd ;
    rd ;
    goto 99;

40: tir := lshift (tir );
    if n then goto 46;
41: alu := tir ;
    if n then goto 44;

42: alu := ac ; {JNEG}
    if n then goto 22;
43: goto 0;

44: alu := ac ; {JNZE}
    if z then goto 0;
45: pc := ir ;
    pc := band (pc , amask );
    goto 0;

46: tir := lshift (tir );
    if n then goto 50;

47: sp := sp + (-1); {CALL}
48: mar := sp ; mbr := pc ; wr ;
    wr ;
49: pc := ir ;
    pc := band (pc , amask );
    goto 0;
```

Fig. 4.20 O microprograma para Mic-2.

MIC-2 Microprograma

```
50: tir := lshift(tir);
    if n then goto 65;
51: tir := lshift(tir);
    if n then goto 59;
52: alu := tir;
    if n then goto 56;

53: mar := ac; rd; {PSHI}
    rd;
54: sp := sp + (-1);
55: a := mbr;
    mar := sp; mbr := a; wr;
    wr;
    goto 0;

56: mar := sp; rd; {POPI}
57: rd;
    sp := sp + 1;

58: a := mbr;
    mar := ac; mbr := a; wr;
    wr;
    goto 0;
59: alu := tir;
    if n then goto 62;

60: sp := sp + (-1); {PUSH}
61: mar := sp; mbr := ac; wr;
    wr;
    goto 0;

62: mar := sp; rd; {POP}
63: rd;
    sp := sp + 1;

64: ac := mbr;
    goto 0;

65: tir := lshift(tir);
    if n then goto 73;
66: alu := tir;
    if n then goto 70;

67: mar := sp; rd; {RETN}
68: rd;
    sp := sp + 1;
69: pc := mbr;
    goto 0;

70: a := ac; {SWAP}
71: ac := sp;
72: sp := a;
    goto 0;

73: alu := tir;
    if n then goto 76;

74: a := ir; {INSP}
    a := band(a, smask);
75: sp := sp + a;
    goto 0;

76: a := ir; {DESP}
    a := band(a, smask);
77: a := inv(a);
78: a := a + 1;
    sp := sp + a;
    goto 0;
```

Fig. 4.20 (cont.)

MIC-2 Microprograma (cont)

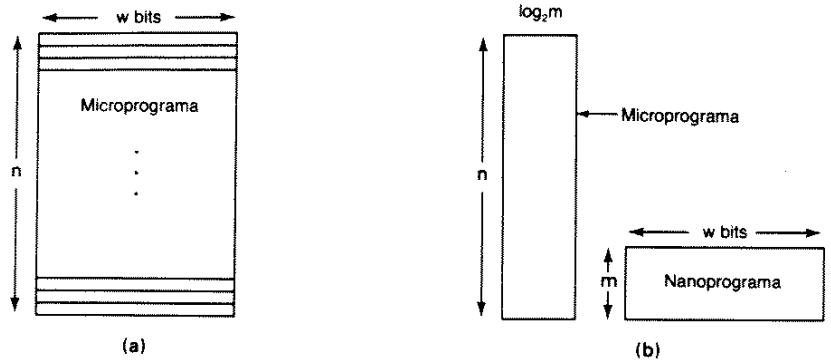

Fig. 4.21 (a) Um microprograma convencional. (b) O nanoprograma correspondente se apenas *m* microinstruções diferentes ocorrem no microprograma.

Nanoprograma