APÊNDICE C

SIMULADOR LÓGICO DE ARQUITETURAS DE PROCESSADORES (SLAP)

C.1 - Ambiente

O SLAP é um simulador lógico hierárquico que tem por objetivo testar propostas de arquiteturas de processadores, através da análise do desempenho destas arquiteturas na execução de algoritmos.

A descrição da arquitetura é feita com alto grau de abstração, utilizando para isto um editor gráfico hierárquico (EGHA). Uma arquitetura é descrita em termos de: registradores, *flip-flops*, unidades lógicas e aritméticas, registradores de deslocamento, barramentos, etc. Para isso, foi criada uma biblioteca de elementos (SLAP.BIB), contendo diversos tipos de registradores, decodificadores, multiplexadores, etc.

Para tornar a interface do simulador o mais amigável e familiar possível, foi criada uma estrutura de menus bastante parecida com a dos compiladores da Borland, como o Turbo Pascal e o Turbo C. O menu oferece quatro opções principais e oito opções secundárias:

- **Arquitetura:**
  - **Carrega Arquitetura** - lê um arquivo de descrição de uma arquitetura e monta as estruturas necessárias para tratar a mesma;
  - **Descreve Barramentos** - apresenta a numeração automática dos barramentos, suas entradas e saídas;
  - **Descreve Bits de Controle** - apresenta a descrição de todos os *bits* de controle do microcódigo, gerados automaticamente pelo SLAP;
- **Decodificação:**
  - **Carrega Decodificador** - lê um arquivo de descrição de um decodificador para o microcódigo e monta as estruturas necessárias para tratar o mesmo;
  - **Decodificação Automática** - destrói as estruturas do decodificador atual e seleciona o modo automático de decodificação do microcódigo;
- **Simulação:**
  - **Carrega Programa Fonte** - lê um arquivo contendo um programa de simulação, compila e armazena o mesmo em forma de microcódigo;

- **Escolhe Variáveis Visíveis** - permite a escolha das variáveis da arquitetura que serão vistas durante a simulação;
- **Simula** - executa a simulação do programa, mostrando, passo a passo, o valor das variáveis escolhidas;

- **Fim** - encerra o simulador e volta para o sistema operacional.

C.2 - Biblioteca

A biblioteca de elementos está gravada no arquivo SLAP.BIB e possui, até o momento, 40 tipos de elementos. A biblioteca possui uma descrição dos elementos e de suas funções. Cada tipo de elemento tem um registro com o seguinte formato:

nome	num_ins	num_outs	num_var_ints	num_funcs	in	out	funcao

O campo **nome** contém o apelido dado ao elemento. Os campos **num_ins** e **num_outs** indicam quantas entradas e saídas de dados possui o elemento, respectivamente. O campo **num_var_ints** indica quantas variáveis internas (variáveis diferentes das entradas e saídas) são necessárias para a simulação de um elemento deste tipo. O campo **num_funcs** indica quantas funções possui o elemento. O vetor **in** e o vetor **out** descrevem as diversas entradas e saídas, respectivamente, indicando seus nomes e número de *bits*. O vetor **funcao** descreve as diversas funções, indicando seus nomes, números de *bits* necessários para controlá-las e os tipos de "atuações" das funções. Foram definidos três tipos de "atuações": **S** (sempre), **I** (intervalo) e **F** (fixa). As funções do tipo **S** são aquelas que estão sempre `sendo executadas, independente de qualquer controle (como a soma em um somador). As funções do tipo **I** são aquelas que atuam num intervalo de tempo determinado (como a decodificação em um decodificador com sinal de habilitação). As funções do tipo **F** são aquelas que atuam num tempo determinado (como um *clock* de um *flip-flop*). Em resumo, pode-se dizer que as funções do tipo **S** não exigem controle, as do tipo **I** têm controle por nível e as do tipo **F** têm controle por transição.

A seguir são descritos os elementos existentes na biblioteca atual:

- **registrador de 1 *bit* (REG1)**
- **registrador de 2 *bits* (REG2)**
- **registrador de 4 *bits* (REG4)**
- **registrador de 8 *bits* (REG8)**

  entradas: I
  saídas: O
  funções: se LOAD, então O = I

- ***flip-flop* tipo "D" (FF)**

  entradas: D

  saídas: Q, -Q

  funções: se CK, então Q = D
  se RESET, então Q = 0
  se SET, então Q = 1
  -Q = *not* Q

- **FIFO de 32 *bits* (FIFO)**

  entradas: I, II, IO

  saídas: O

  funções: se LOAD, então F [II] = I
  O = F [IO]

- **memória RAM 256 X 1 (RAM1)**
- **memória RAM 256 X 4 (RAM4)**

  entradas: I

  saídas: O

  funções: se READ (endereço), então O = M [endereço]
  se WRITE (endereço), então M [endereço] = I

- **unidade lógica de 1 *bit* (LOG1)**
- **unidade lógica de 4 *bits* (LOG4)**

  entradas: A, B

  saídas: O

  funções: se LOGIC (i), então O = função lógica (i) (A, B)

- **porta lógica AND com 2 entradas (AND)**

  entradas: I1, I2

  saídas: O

  funções: O = I1 *and* I2

- **porta lógica NAND com 2 entradas (NAND)**

  entradas: I1, I2

  saídas: O

  funções: O = I1 *nand* I2

- **porta lógica XOR com 2 entradas (XOR)**

  entradas: I1, I2

  saídas: O

  funções: O = I1 *xor* I2

- **porta lógica inversora (NOT)**

  entradas: I

  saídas: O

  funções: O = *not* I

- **porta lógica OR com 4 entradas (OR4)**

  entradas: I

  saídas: O

  funções: $O = I_3$ *or* I_2 *or* I_1 *or* I_0

- **somador de 1 *bit* (ADD1)**
- **somador de 4 *bits* (ADD4)**
- **somador de 8 *bits* (ADD8)**

  Seja *n* o número de *bits* do somador

  entradas: A, B, C0

  saídas: S, C*n*

  funções: (C*n*, S) = A + B + C0

- **somador de 4 *bits* com *carry* duplo (ADD4C)**

  entradas: A, B, C0/C4

  saídas: S, C4

  funções: se SUMC4, então (C4, S) = A + B + (C0/C4,0000)

  senão (C4, S) = A + B + C0/C4

- **contador de 2 *bits* (CONT2)**
- **contador de 4 *bits* (CONT4)**
- **contador de 5 *bits* (CONT5)**
- **contador de 8 *bits* (CONT8)**

  entradas: D

  saídas: Q

  funções: se LOAD, então Q = D

  se UP, então Q = Q + 1

  se DOWN, então Q = Q -1

- **contador decrescente de 8 *bits* (DOWN8)**

  entradas: D

  saídas: Q

  funções: se LOAD, então Q = D

  se DOWN, então Q = Q -1

- **registrador de deslocamento de 4 *bits* (SH4)**
- **registrador de deslocamento de 8 *bits* (SH8)**
- **registrador de deslocamento de 16 *bits* (SH16)**

  Seja *n* a largura do registrador de deslocamento

  entradas: SL, SR, PI

  saídas: Q0, Q*n*-1, PO

  funções: se LOAD, então PO = PI

  se SHR, então PO = PO *shift right* 1 + SL *shift left* (*n*-1)

  se SHL, então PO = PO *shift left* 1 + SR

  Q0 = PO [0], Q*n*-1 = PO [*n*-1]

- ***barrel shifter* de 4 *bits* (BARSH)**

  entradas: I, S

  saídas: O

  funções: se LEFT, então O = I *shift left* (4 - S)

  senão O = I *shift right* S

- **multiplexador 2 X 1 (MUX2)**
- **multiplexador 8 X 4 (MUX8)**

  entradas: I0, I1

  saídas: O

  funções:

HAB0	O
0	I0
1	I1

- **multiplexador 4 X 1 (MUX4)**
- **multiplexador 16 X 4 (MUX16)**

  entradas: I0, I1, I2, I3

  saídas: O

  funções:

HAB1 HAB0	O
0	I0
1	I1
2	I2
3	I3

- **transforma um barramento de 4 *bits* em 4 fios (B1X4)**

  entradas: B4

  saídas: b0, b1, b2, b3

  funções: b3 = B4 *shift right* 3

  b2 = (B4 *shift right* 2) *and* 0001

  b1 = (B4 *shift right* 1) *and* 0001

  b0 = (B4 *shift right* 0) *and* 0001

- **transforma 4 fios num barramento de 4 *bits* (B4X1)**

  entradas: b0, b1, b2, b3

  saídas: B4

  funções: B4 = (b3, b2, b1, b0)

- **decodificador 1 X 2 (sem habilitador) (D1_2)**

  entradas: I

  saídas: Q0, Q1

  funções:

I	Q0	Q1
0	1	0
1	0	1

- **decodificador 2 X 4 (sem habilitador) (D2_4)**

  entradas: I1, I0

  saídas: Q0, Q1, Q2, Q3

  funções:

I1 I0	Q0	Q1	Q2	Q3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

- **decodificador 3 X 8 (sem habilitador) (D3_8)**

  entradas: I2, I1, I0

  saídas: Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7

  funções:

I2 I1 I0	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0	1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0
3	0	0	0	1	0	0	0	0
4	0	0	0	0	1	0	0	0
5	0	0	0	0	0	1	0	0
6	0	0	0	0	0	0	1	0
7	0	0	0	0	0	0	0	1

- **decodificador 1 X 2 (com habilitador) (DH1_2)**

  entradas: H, I

  saídas: Q0, Q1

  funções:

H	I	Q0	Q1
0	X	0	0
1	0	1	0
1	1	0	1

- **decodificador 2 X 4 (com habilitador) (DH2_4)**

  entradas: H, I1, I0

  saídas: Q0, Q1, Q2, Q3

  funções

H	I1 I0	Q0	Q1	Q2	Q3
0	X	0	0	0	0
1	0	1	0	0	0
1	1	0	1	0	0
1	2	0	0	1	0
1	3	0	0	0	1

- **decodificador 3 X 8 (com habilitador) (DH3_8)**

  entradas: H, I2, I1, I0

  saídas: Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7

  funções:

H	I2 I1 I0	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0	X	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0
1	2	0	0	1	0	0	0	0	0
1	3	0	0	0	1	0	0	0	0
1	4	0	0	0	0	1	0	0	0
1	5	0	0	0	0	0	1	0	0
1	6	0	0	0	0	0	0	1	0
1	7	0	0	0	0	0	0	0	1

No módulo SLAPBIBL.C, encontram-se as rotinas de simulação de todos os elementos da biblioteca. Ao acrescentar algum elemento à biblioteca, faz-se necessário alterar a rotina **calc_funcao** no módulo SLAPBIBL.C.

C.3 - Descrição da Arquitetura

Como foi dito anteriormente, a descrição da arquitetura é feita através de um editor gráfico. Este editor gera um arquivo de saída com extensão **.ARQ**, composto por registros. O primeiro campo do registro indica o seu tipo. Existem 9 tipos de registros, abaixo descritos:

- **registro de elemento** - descreve um elemento da arquitetura. O elemento tem que pertencer a biblioteca de elementos SLAP.BIB. O campo **indice** contém o índice do elemento dentro de arquivo de descrição. O campo **nome** contém o nome dado ao elemento. O campo **tipo** contém o índice do elemento na biblioteca. Os campos seguintes contém os intervalos de atuação das diversas funções do elemento.

				função [0]		função [1]		função [2]	
ELEMENTO	indice	nome	tipo	t_min	t_max	t_min	t_max	t_min	t_max

• **registro de bloco** - descreve um bloco, que representa outro arquivo de descrição de arquitetura. São os blocos que permitem a estrutura hierárquica do simulador. O campo **indice** contém o índice do bloco dentro do arquivo de descrição. O campo **nome** contém o nome dado ao bloco. O campo **nome_arq** contém o nome do arquivo de descrição da arquitetura do bloco.

BLOCO	indice	nome	nome_arq

• **registro de entrada** - descreve um ponto de entrada da arquitetura. O campo **indice** contém o índice da entrada dentro do arquivo de descrição. O campo **nome** contém o nome dado a entrada. O campo **tipo** indica o tipo da entrada. Existem três tipos de entrada: FIXA, VARIAVEL, CTRL_POR_MICROCÓDIGO. A entrada FIXA, como o próprio nome já diz, tem valor fixo, dado pelo campo **valor**. A entrada VARIAVEL age como um rótulo. Ela serve para a ligação entre blocos. A entrada CTRL_POR_MICROCÓDIGO tem seu valor controlado pelo microcódigo. Os campos **t_min** e **t_max** indicam o intervalo de tempo em que o conteúdo da entrada é jogado no barramento ao qual ela está ligada.

ENTRADA	indice	nome	tipo	valor	t_min	t_max

• **registro de saída** - descreve um ponto de saída da arquitetura. O campo **indice** contém o índice da saída dentro do arquivo de descrição. O campo **nome** contém o nome dado a saída. A saída age como um rótulo. Ela serve para a ligação entre blocos.

SAIDA	indice	nome

• **registro de entrada de elemento** - descreve um ponto de entrada de um elemento. Este registro é necessário, pois vários barramentos podem concorrer à mesma entrada, ou seja, para uma única entrada declarada para o elemento na biblioteca de elementos, podem existir vários registros de entrada de elemento. Neste caso, o simulador interpreta que, implicitamente, existe um multiplexador selecionando que barramento será responsável por enviar dados a entrada. Além disso, o simulador cria *bits* no microcódigo responsáveis por controlar esse multiplexador implícito. O campo **indice** contém o índice da entrada de elemento dentro do arquivo de descrição. O campo **i_el** contém o índice do elemento ao qual pertence a entrada e o campo **i_in** contém o índice da entrada na descrição do elemento na biblioteca.

ENTRADA_DE_ELEMENTO	indice	i_el	i_in

• **registro de saída de elemento** - descreve um ponto de saída de um elemento. Este registro é necessário, pois uma saída pode enviar dados para vários barramentos, ou seja, para uma única saída declarada para o elemento na biblioteca de elementos, podem existir vários registros de saída de elemento. Quando necessário, o simulador interpreta que, implicitamente, existem *buffers* 3-states entre a saída e os diversos barramentos. Neste caso, o simulador cria

bits no microcódigo responsáveis por controlar as saídas dos *buffers*. O campo **indice** contém o índice da saída de elemento dentro do arquivo de descrição. O campo **i_el** contém o índice do elemento ao qual pertence a saída e o campo **i_out** contém o índice da saída na descrição do elemento na biblioteca. Os campos **t_min** e **t_max** definem o intervalo em que a saída envia dados para o barramento.

SAIDA_DE_ELEMENTO	indice	i_el	i_out	t_min	t_max

• **registro de controle de elemento** - descreve um controle de uma função de um elemento. Para cada controle não declarado, o simulador cria os *bits* de controle necessários no microcódigo, automaticamente. Desta forma, só é necessário declarar os controles que são dependentes de variáveis internas do processador, como por exemplo, operações mascaradas. O campo **indice** contém o índice do controle do elemento dentro do arquivo de descrição. O campo **i_el** contém o índice do elemento ao qual pertence o controle e o campo **i_func** contém o índice da função na descrição do elemento na biblioteca. Cabe aqui um pequeno parênteses. Para funções que necessitam de mais de um *bit* para controla-las, foi criada uma subdivisão no controle: o primeiro *bit* é tratado de forma diferente dos seguintes. Sendo assim, foi necessário criar uma distinção no campo **i_func**. Quando não se trata do primeiro *bit*, é somado quatro ao valor deste campo.

CONTROLE_DE_ELEMENTO	indice	i_el	i_func

• **registro de fio** - descreve um fio através de seus extremos. O campo **indice** contém o índice do fio dentro do arquivo de descrição. Os campos **tipo1**, **índice1** e **compl1** descrevem um dos extremos e os campos **tipo2**, **índice2** e **compl2** descrevem o outro extremo. **Tipo** indica o que existe ligado ao extremo do fio: uma entrada de elemento, uma saída de elemento, um controle de elemento, uma entrada, uma saída, uma entrada de um bloco, uma saída de um bloco, um outro fio ou um nó. Para especificar exatamente quem está ligado ao extremo do fio, na maioria dos casos só é necessário mais um índice, a não ser nos casos do extremo estar ligado a uma entrada ou saída de um bloco. Nestes casos, são necessários dois índices (o **indice** e o **compl**), o primeiro para especificar o bloco e o segundo para especificar a entrada (ou saída).

FIO	indice	tipo1	índice1	compl1	tipo2	índice2	compl2

• **registro de nó** - descreve um nó através dos fios ligados ao mesmo. O campo **indice** contém o índice do nó dentro do arquivo de descrição. O vetor **i_fio** contém os índices dos diversos fios (máximo de quatro) ligados ao nó. Quando o nó possui menos que quatro fios ligados a ele, o resto do vetor é preenchido pela constante SEM_LIGACAO.

NO	indice	i_fio

Internamente, O SLAP armazena a arquitetura como uma árvore de blocos, cada bloco sendo originado pela leitura de um arquivo com extensão **.ARQ**. O bloco raiz (bloco principal) é originado pelo arquivo de descrição da arquitetura e é armazenado na variável **bl_p**. Esta variável é uma estrutura do tipo **tipo_bloco**, composta pelos campos descritos abaixo.

- **nome** - nome dado ao bloco.
- **nome_arq** - nome do arquivo que originou o bloco.
- **num_els** - número de elementos.
- **num_bls** - número de blocos "filhos" pertencentes ao bloco.
- **num_ins_arq** - número de entradas do próprio bloco.
- **num_outs_arq** - número de saídas do próprio bloco.
- **num_ins_els** - número de pontos de entrada de elementos.
- **num_outs_els** - número de pontos de saída de elementos.
- **num_ctrls_els** - número de controles declarados de elementos.
- **num_ins_bl** - número de entradas de blocos filhos.
- **num_outs_bl** - número de saídas de blocos filhos.
- **num_fios** - número de fios.
- **num_nos** - número de nós.
- **prox_bl_equiv** - apontador para um bloco equivalente a este, ou seja, um bloco com a mesma descrição e que vai funcionar em paralelo com ele, com os mesmos controles. Esse apontador forma uma lista circular encadeada, contendo todos os blocos que vão receber controles idênticos. Caso o bloco não possua nenhum bloco equivalente, este campo aponta para o próprio bloco. Quem indica quais blocos são equivalentes é o programa de simulação.
- **el** - vetor de descrição dos elementos pertencentes ao bloco. Este campo é um apontador para uma estrutura do tipo **tipo_elemento**, composta por:
  - **nome** - nome do elemento.
  - **tipo** - índice do tipo do elemento na biblioteca SLAP.BIB.
  - **i_in** - índice da primeira entrada do elemento no vetor de entradas **in**. Este vetor, usado na simulação, contém o valor de todas as entradas de todos os elementos, num determinado tempo.
  - **i_out** - índice da primeira saída do elemento no vetor de saídas **out**. Este vetor, usado na simulação, contém o valor de todas as saídas de todos os elementos e de todas as saídas de todos os blocos, num determinado tempo.
  - **i_var_int** - índice da primeira variável interna do elemento no vetor de variáveis internas **var_int**. Este vetor, usado na simulação, contém o valor de todas as variáveis internas de todos os elementos, num determinado tempo.
  - **funcao** - vetor contendo a descrição de todas as funções do elemento. Este campo é um vetor de estruturas do tipo **descr_func**, compostas por:

- **t_min** - tempo mínimo de atuação da função.
- **t_max** - tempo máximo de atuação da função.
- **tipo** - tipo do primeiro *bit* de controle. São definidos três tipos de controle: MICROCÓDIGO (quando o controle advém do microcódigo), INTERNO (quando o controle advém de um sinal da arquitetura) e INEXISTENTE (quando a função não necessita de controle).
- **tipo2** - tipo dos demais *bits* de controle (válido somente para funções com mais de um *bit* de controle).
- **i_bit** - índice do primeiro *bit* de controle no microcódigo (se tipo = MICROCÓDIGO) ou no vetor **ctrls_el** (se tipo = INTERNO).
- **i_bit2** - índice dos demais *bits* de controle da função (válido somente para funções com mais de um *bit* de controle).
- **bit_func** - índice do primeiro *bit* de controle no vetor **bits_ctrl_funcoes**, que é o vetor que contém o valor atual dos controles de todos os elementos.

- **bl** - vetor de descrição dos blocos "filhos" pertencentes ao bloco. É a partir deste vetor que a árvore de blocos cresce. Este campo é um apontador para uma estrutura do tipo **tipo_bloco**.
- **in_arq** - vetor de descrição das entradas do próprio bloco. Este campo é um apontador para uma estrutura do tipo **tipo_in_arq**, composta por:
  - **nome** - nome da entrada.
  - **tipo** - tipo da entrada (FIXA, VARIAVEL, CTRL_POR_MICROCOD).
  - **i_fio_int** - se **tipo** = VARIAVEL, contém o índice do fio do bloco ligado a esta entrada.
  - **i_fio_ext** - se **tipo** = VARIAVEL, contém o índice do fio de outro bloco ligado a esta entrada deste bloco.
  - **bit_ctrl** - índice do *bit* do microcódigo que controla a liberação desta entrada para o barramento ao qual ela está ligada.
  - **bit_valor** - se **tipo** = CTRL_POR_MICROCOD, indica o índice do *bit* do microcódigo que contém o valor da entrada.
  - **i_out** - índice da entrada no vetor **out** utilizado para a visualização de resultados.
  - **t_min** e **t_max** - indicam o intervalo de liberação desta entrada para o barramento ao qual ela está ligada.
  - **bloco_ext** - se tipo = VARIAVEL, contém um apontador para o outro bloco ao qual esta entrada está ligada.
  - **valor** - valor atual da entrada.
- **out_arq** - vetor de descrição das saídas do próprio bloco. Este campo é um apontador para uma estrutura do tipo **tipo_out_arq**, composta por:

  - **nome** - nome da saída.
  - **i_fio_int** - contém índice do fio do bloco ligado a esta saída.
  - **i_fio_ext** - contém o índice do fio de outro bloco ligado a esta saída deste bloco.
  - **i_out** - índice da saída no vetor **out**.
  - **bloco_ext** - contém um apontador para o outro bloco ao qual esta saída está ligada.
- **in_el** - vetor de descrição dos pontos de entrada dos elementos do bloco. Este campo é um apontador para uma estrutura do tipo **tipo_in_el**, composta por:
  - **i_el** - índice do elemento ao qual pertence a entrada.
  - **i_in** - índice da entrada na descrição do elemento na biblioteca.
  - **bit_ctrl** - índice do *bit* do microcódigo que controla se este ponto de entrada vai ou não ser responsável por fornecer o valor da entrada do elemento.
- **out_el** - vetor de descrição dos pontos de saída dos elementos do bloco. Este campo é um apontador para uma estrutura do tipo **tipo_out_el**, composta por:
  - **i_el** - índice do elemento ao qual pertence a saída.
  - **i_out** - índice da saída na descrição do elemento na biblioteca.
  - **t_min** e **t_max** - intervalo de tempo em que este ponto de saída pode enviar dados para o barramento ao qual ele está ligado.
  - **bit_ctrl** - índice do *bit* do microcódigo que controla se este ponto de saída vai ou não enviar dados para o barramento.
- **ctrl_el** - vetor de descrição dos controles dos elementos do bloco. Este campo é um apontador para uma estrutura do tipo **tipo_ctrl_el**, composta por:
  - **i_el** - índice do elemento ao qual pertence o controle.
  - **i_func** - índice do controle na descrição do elemento na biblioteca.
  - **bit_func** - índice do controle no vetor **bits_ctrl_funcoes**.
- **in_bl** - vetor de descrição das entradas dos blocos filhos ligadas a este bloco. Este campo é um apontador para uma estrutura do tipo **tipo_in_bl**, composta por:
  - **i_bl** - índice do bloco ao qual pertence a entrada.
  - **i_in** - índice da entrada no bloco filho.
- **out_bl** - vetor de descrição das saídas dos blocos filhos ligadas a este bloco. Este campo é um apontador para uma estrutura do tipo **tipo_out_bl**, composta por:
  - **i_bl** - índice do bloco ao qual pertence a saída.
  - **i_out** - índice da saída no bloco filho.
- **.fio** - vetor de descrição dos fios do bloco. Este campo é um apontador para uma estrutura do tipo **tipo_fio**, composta por:
  - **tipo1, índice1 e compl1** - indica o que está ligado a um extremo do fio.
  - **tipo2, índice2 e compl2** - indica o que está ligado ao outro extremo do fio.

- **ja_visitado** - indica se o fio já foi percorrido pela rotina que monta a estrutura dos barramentos ou não.

- **no** - vetor de descrição dos nós do bloco. Este campo é um apontador para uma estrutura do tipo **tipo_no**, composta por:
  - **i_fio** - vetor que contém os índice dos fios ligados pelo nó.
  - **ja_visitado** - indica se o nó já foi percorrido pela rotina que monta a estrutura dos barramentos ou não.

C.4 - Decodificador

Uma arquitetura de um processador pode necessitar de muitos *bits* de controle, o que geraria um microcódigo muito grande. Na prática porém, muitos controles são iguais ou mutuamente exclusivos. Sendo assim, pode-se criar um circuito decodificador para o microcódigo, tornando-o mais enxuto.

O SLAP permite que se use um esquema de decodificação que transforme o microcódigo comprimido (armazenado como programa) no microcódigo expandido (contendo cada controle em separado). Esse decodificador também é gerado pelo editor gráfico, mas com algumas diferenças em relação a um arquivo de descrição comum. A primeira se refere ao nome do arquivo de descrição, que tem extensão **.DEC**. A segunda se refere ao arquivo gerado: é inserido um registro a mais, o **registro de decodificação**, contendo o nome do arquivo de descrição cujo microcódigo está sendo decodificado. Este registro tem o seguinte formato:

NOME_ARQ_DESCR	nome_arq

Quanto ao conteúdo do circuito, só são aceitos elementos decodificadores. Além disso, as entradas do circuito, que exprimem o microcódigo comprimido, devem ter o nome com o formato **BIT [n]**, onde n é o índice do *bit* de controle no microcódigo comprimido. As saídas do circuito, que exprimem o microcódigo expandido, devem ter o nome do sinal que elas controlam, conforme o formato mostrado na opção **Descreve Bits de Controle**.

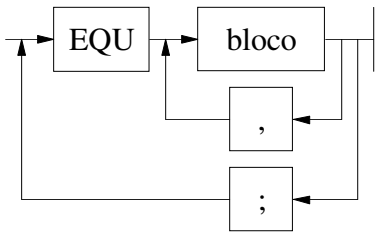
Em tempo de montagem do programa de simulação, uma rotina é encarregada de codificar as instruções, para que elas sejam armazenadas na forma compactada. Essa rotina age como um simulador ao contrário, procurando que entradas resultam numa determinada saída. Em tempo de simulação, uma outra rotina é encarregada de decodificar as instruções, a medida em que elas vão ser executadas. Essa rotina age como um simulador.

C.5 - Simulador

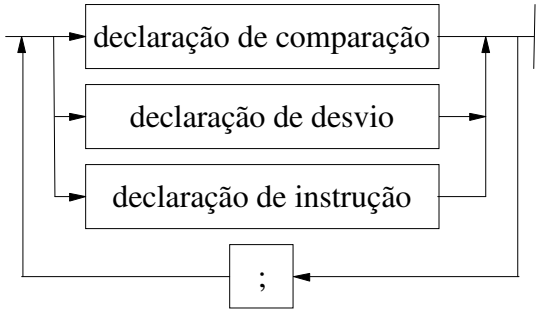
A simulação executa um programa de simulação previamente lido e compilado. O SLAP possui um pequeno montador capaz de transformar um programa fonte (num formato padrão) em um programa objeto (vetor de instruções codificadas). O formato padrão do programa fonte , apresentado a seguir:

programa - estrutura geral do programa fonte.

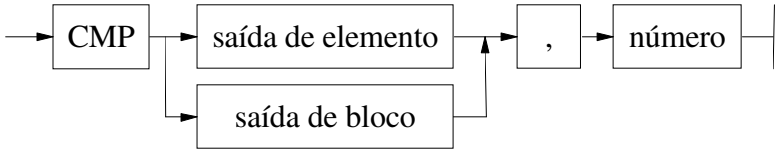
equivalências - declara quais blocos são equivalentes, isto é, quais blocos iguais têm os mesmos controles. Os blocos equivalentes são separados por vírgulas. As declarações de equivalência são separadas por ponto e vírgula.



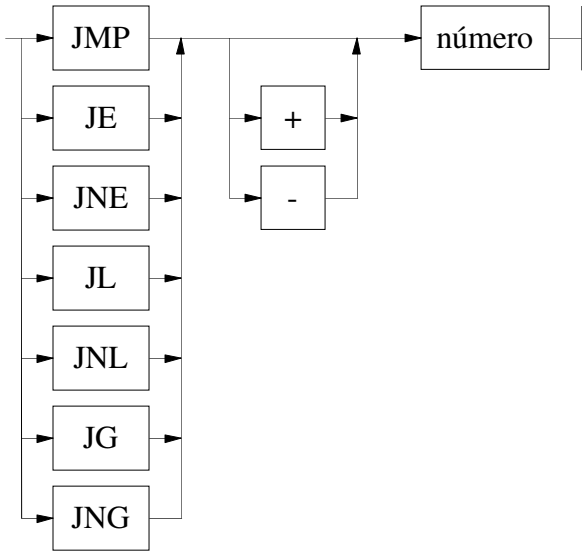
códigos - possíveis tipos de instruções. As instruções são separadas por ponto e vírgula.



declaração de comparação - este tipo de instrução compara uma saída a um valor. O resultado da comparação é armazenado para posterior uso em uma instrução de desvio.



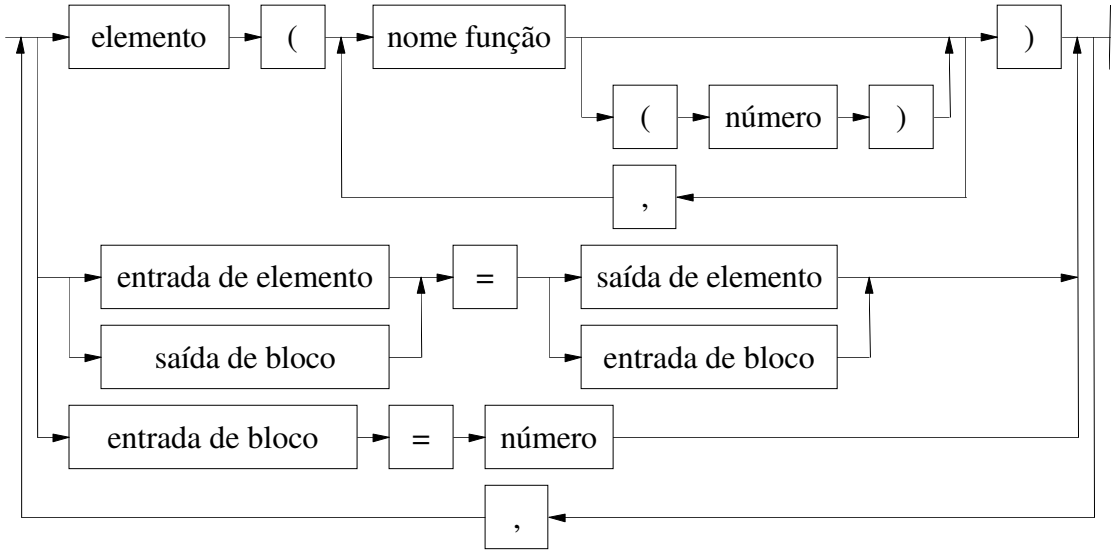
declaração de desvio - este tipo de instrução executa um desvio relativo (mais ou menos um determinado número de instruções), condicional ou não. Os desvios condicionais dependem do resultado da última instrução de comparação.



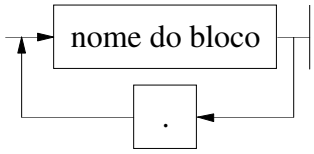
declaração de instrução - este tipo de instrução executa realmente alguma ação no processador. Podem existir declarações de função e declarações de entrada e saída. As declarações de função permitem "ligar" os *bits* de controles das funções dos elementos. O primeiro *bit* de controle de cada função é "ligado" através de uma referência ao nome da função (declarado na biblioteca). Os demais, quando existirem, têm seu valor igualado ao conteúdo entre parênteses, declarado logo após o nome da função. As declarações de entrada e saída podem ter três finalidades:

- selecionar quem envia dados a um determinado barramento,
- selecionar de que barramento uma determinada entrada de elemento recebe dados,
- definir qual o valor de uma determinada entrada de bloco controlada por microcódigo.

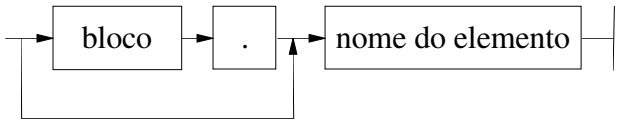
Uma declaração de entrada e saída associando um fornecedor de dados de um barramento a um consumidor de dados do mesmo barramento pode ter duas finalidades simultâneas: selecionar quem envia dados ao barramento e selecionar que consumidor vai receber dados deste barramento.



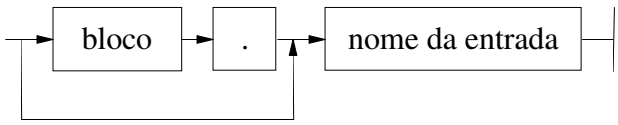
bloco - um bloco é sempre referenciado a partir do bloco raiz. A partir daí são feitas referências aos nomes dos blocos filhos, separados por pontos, até chegar ao nome do bloco em questão, seguindo a estrutura hierárquica.



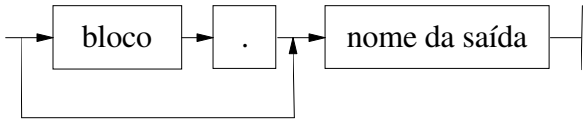
elemento - um elemento é referenciado não só pelo seu nome, mas também pelo nome do bloco ao qual ele pertence, a não ser que ele pertença ao bloco principal.



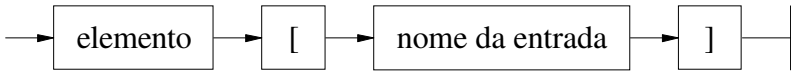
entrada de bloco - uma entrada é referenciada não só pelo seu nome, mas também pelo nome do bloco ao qual ela pertence, a não ser que ela pertença ao bloco principal.



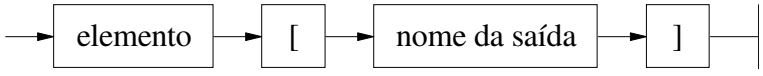
saída de bloco - uma saída é referenciada não só pelo seu nome, mas também pelo nome do bloco ao qual ela pertence, a não ser que ela pertença ao bloco principal.



entrada de elemento



saída de elemento



As variáveis que serão rastreadas durante a simulação são escolhidas através de menus de inclusão e exclusão. A simulação só é executada se houver pelo menos uma variável "visível", ou seja, uma variável cujo conteúdo será mostrado em tempo de simulação. Através da opção **Escolhe Variáveis Visíveis**, é permitida não só a escolha de quais variáveis serão apresentadas durante a simulação, como também a ordem em que elas serão apresentadas.

A opção **Simula** monta na tela o nome e o valor de todas as variáveis "visíveis", além do valor do ip (*instruction pointer*) atual. Para cada instrução, o simulador executa o seguinte algoritmo básico:

- transforma o microcódigo comprimido em microcódigo expandido
- calcula o valor de todas as entradas controladas por microcódigo
- calcula o valor de todos os controles de funções
- de tempo = 0 até tempo = período, com acréscimos de delta_t:
  - calcula o valor de todas as entradas de todos os elementos
  - simula o funcionamento de todos os elementos, calculando as suas saídas
  - calcula o conteúdo de todos os barramentos, verificando possíveis conflitos e alterando os controles de funções provenientes de sinais internos da arquitetura
- mostra o valor atual de todas as variáveis "visíveis"

C.6 - Exemplo

A seguir é apresentado um exemplo de utilização do SLAP. A arquitetura de teste é bastante simples, contendo apenas dois registradores de um *bit* (A e B), um *flip-flop* (C), uma unidade lógica (P), um somador completo de um *bit* (S) e uma memória RAM 256 X 1 (M). O esquema da arquitetura é apresentado na Figura C.1.

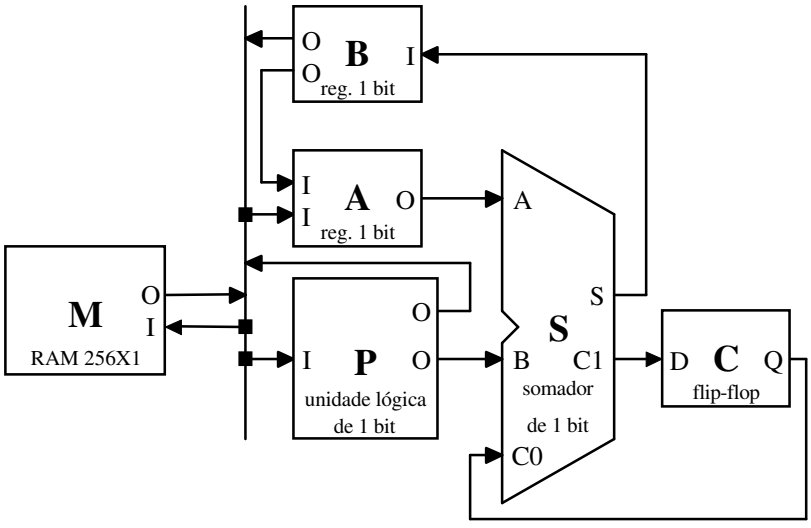


Figura C.1 - Arquitetura para exemplo de simulação.

Considere que, através do editor gráfico, esta arquitetura foi desenhada e seu arquivo de descrição (FPP.ARQ) foi gerado. O primeiro passo ao entrar no SLAP é ativar a opção **Carrega Arquitetura**. O nome do arquivo de descrição é então, pedido. Digite FPP.ARQ e tecle <ENTER>. A arquitetura será carregada na memória e diversas estruturas serão criadas. Para ver a numeração automática dada aos barramentos, suas entradas e saídas, ativa-se a opção **Descreve Barramentos**. O resultado desta operação é apresentado a seguir:

```
barramento [0]  entradas: S [S],
                saídas: B [I],
barramento [1]  entradas: S [C1],
                saídas: C [D],
barramento [2]  entradas: C [Q],
                saídas: S [C0],
barramento [3]  entradas: P [O],
                saídas: S [B],
barramento [4]  entradas: A [O],
                saídas: S [A],
barramento [5]  entradas: B [O],
                saídas: A [I],
barramento [6]  entradas: B [O], M [O], P [O],
                saídas: A [I], M [I], P [I],
```

Pode-se ainda ver a numeração automática dada aos *bits* do microcódigo. Para isto, ativa-se opção **Descreve *Bits* de Controle**. O resultado desta operação é apresentado a seguir:

```
bit 0 -       A(LOAD)
bit 1 -       B(LOAD)
bit 2 -       C(CK)
bit 3 -       C(RESET)
bit 4 -       C(SET)
bit 5 -       P(LOGIC)
bit 6.. 9 -   P(LOGIC(D3..D0))
bit 10 -      M(READ)
bit 11..18 -  M(READ(D7..D0))
bit 19 -      M(WRITE)
bit 20..27 -  M(WRITE(D7..D0))
bit 28 -      b[6] = B[O]
bit 29 -      b[6] = M[O]
bit 30 -      b[6] = P[O]
bit 31 -      A[I] = b[5]
bit 32 -      A[I] = b[6]
```

Para este exemplo existem quatro tipos de controle:

- Primeiro *bit* de controle de uma função, como A(LOAD);
- Demais *bits* de controle de uma função, como M(READ(D7..D0));
- Controle de envio de dados para um barramento (simulação de um *buffer* 3-state), como b[6] = P[O];
- Controle de seleção de dados (simulação de um multiplexador), como A[I] = b [6].

O próximo passo é ler um programa de simulação. A seguir é apresentado um programa exemplo, contido no arquivo FPP.PRG

```
P (LOGIC (15)), C (RESET);
A [I] = P [O], A (LOAD);
B (LOAD), C (CK);
M (WRITE (50)), M [I] = B [O];
A (LOAD), M (READ (50)), A [I] = M [O];
END;
```

Através da opção **Carrega Programa Fonte**, o programa FPP.PRG é carregado e montado, gerando um código armazenado na memória. Antes de iniciar a simulação, tem-se ainda que escolher as variáveis a serem apresentadas, através da opção **Escolhe Variaveis Visiveis**. Para este exemplo escolheremos as saídas de elementos mais significativas: A[O], B[O], C[Q], P[O] e M[O]. Enfim, através da opção **Simula**, obtemos os seguintes resultados:

```
A[O]   X   1   1   1   0
B[O]   X   X   0   0   0
C[Q]   0   0   1   1   1
P[O]   1   1   1   1   1
M[O]   X   X   X   X   0
```

Cada coluna apresenta o valor das variáveis escolhidas após a simulação de uma instrução. Todas as variáveis são iniciadas com valor indefinido (X).

Após a primeira instrução, o *flip-flop* C executa a função RESET, fazendo com que a saída Q receba valor 0 e a unidade lógica P executa a função lógica 15, que leva a saída O para 1, independente da entrada e do valor anterior da saída.

Após a segunda instrução, o valor da saída O da unidade lógica P é carregado no registrador A, através do barramento [6] e da função LOAD do registrador. Desta forma, a saída O do registrador A vai para 1.

Após a terceira instrução, o resultado da soma da saída O do registrador A, da saída O da unidade lógica P e da saída O do *flip-flop* C é carregado no registrador B (através da função LOAD) e no *flip-flop* C (carry) (através da função CK). Desta forma, como 1 + 1 + 0 = 1 0, a saída O do registrador B vai para 0 e a saída O do *flip-flop* C vai para 1.

A quarta instrução grava o valor da saída O do registrador B na posição 50 da memória M, através da função WRITE. Desta forma, a posição 50 da memória M passa a conter o valor 0.

Após a quinta e última instrução, a posição 50 da memória M é lida (através da função READ) e seu valor é armazenado no registrador A. Sendo assim, a saída O do registrador A vai para 0.

Por último, a opção **Fim** permite o encerramento do programa.