REPÚBLICA FEDERATIVA DO BRASIL
MINISTÉRIO DA ECONOMIA
INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL

CARTA PATENTE Nº PI 0925284-3

O INSTITUTO NACIONAL DA PROPRIEDADE INDUSTRIAL concede a presente PATENTE DE INVENÇÃO, que outorga ao seu titular a propriedade da invenção caracterizada neste título, em todo o território nacional, garantindo os direitos dela decorrentes, previstos na legislação em vigor.

(21) Número do Depósito: PI 0925284-3

(22) Data do Depósito: 03/04/2009

(43) Data da Publicação Nacional: 06/01/2015

(51) Classificação Internacional: G06F 15/80.

(54) Título: SISTEMA INTEGRADO DE PROCESSAMENTO BASEADO EM REDES EM CHIP QUE NÃO FAZ USO DE PROCESSADORES DO TIPO VON NEUMANN

(73) Titular: UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE. CGC/CPF: 24365710000183. Endereço: Campus Universitario Lagoa, 59072-970, RN, BRASIL(BR)

(72) Inventor: IVAN SARAIVA SILVA; SILVIO ROBERTO FERNANDES DE ARAUJO.

Prazo de Validade: 20 (vinte) anos contados a partir de 03/04/2009, observadas as condições legais

Expedida em: 15/06/2021

Assinado digitalmente por:
Liane Elizabeth Caldeira Lage
Diretora de Patentes, Programas de Computador e Topografias de Circuitos Integrados

Relatório descritivo da patente de invenção para **“SISTEMA INTEGRADO DE PROCESSAMENTO BASEADO EM REDES EM CHIP QUE NÃO FAZ USO DE PROCESSADORES DO TIPO VON NEUMANN”**

[001] A presente invenção refere-se a um sistema de processamento baseado em redes em chip, no qual as aplicações são descritas em um formato de pacote contendo instruções e dados, que são executadas nos elementos de roteamento, por onde o pacote trafega, ou nos núcleos de acesso a memória, e, portanto, não faz uso de processadores do tipo Von Neumann. Para tanto esse sistema utiliza um algoritmo de roteamento próprio que permite re-rotear um pacote quando o mesmo atinge seu destino e ainda possui instruções a serem executadas.

[002] Desde que o computador IAS (*Institute for Advanced Studies*) foi proposto por Von Neumann, estruturalmente as arquiteturas de computadores subsequentes têm sofrido poucas modificações, e por isso são comumente chamadas de processadores do tipo Von Neumann. Esse modelo conceitual de arquitetura pode ser simplificado por uma Unidade Central de Processamento – CPU (1), memória (5) e dispositivos de entrada e saída (6), como ilustra a Figura 1. Na memória (5) são armazenados os dados e os programas (em forma de instruções), os quais são buscados quando solicitados pela unidade de controle (3) da CPU (1). A CPU (1) também é formada por uma unidade lógica e aritmética ou ULA (2) e registradores (4). A ULA(2) realiza as operações lógica ou aritméticas de acordo com instrução decodificada pela unidade de controle (3) que também controla quando e que informações são armazenadas nos registradores (4). Da mesma forma, a unidade de controle (3) define o momento de ler ou escrever nos dispositivos de entrada e saída (6), que são responsáveis pela comunicação do processador com o meio externo. Assim, o trabalho do processador pode ser resumido como a busca,

descodificação e execução de cada instrução que forma o programa, como ilustra a Figura 2.

[003] Do IAS até os dias atuais, a tecnologia na fabricação de chips passou pelo uso de válvulas até os transistores. Do mesmo modo, as técnicas empregadas ao longo dos anos permitiram construir transistores cada vez menores, possibilitando integração de um número muito maior desses elementos em um mesmo chip, chegando atualmente aos bilhões. Consequentemente, processadores cada vez mais complexos e com maior desempenho foram surgindo juntamente com a tecnologia empregada e a metodologia de reuso.

[004] Seguindo esse raciocínio surgiram sistemas completos dentro de um chip, os chamados SoCs (SoC - *System-on-Chip*). Os SoCs podem ser formadas por dezenas a centenas de componentes independentes, os quais podem ser elementos de processamento (como processadores de uso geral), módulos de memória ou outro núcleo de processamento útil para o sistema de computação, simplesmente chamados de núcleos. Muitas vezes, SoCs formados por vários processadores são também chamados de multiprocessadores em chip ou *multiprocessors SoC* (MP-SoC).

[005] Com tantos elementos dentro de um chip, é necessário um mecanismo de comunicação entre os componentes. O mecanismo tradicional é chamado de barramento (7), que é um conjunto de linhas de transmissão compartilhado pelos núcleos (8) que estão ligados a ele, ilustrados na Figura 3. Nesse modelo, apenas um núcleo pode transmitir em um dado instante, controlado possivelmente por um árbitro (9) ou uma lógica compartilhada pelos núcleos. Já foram construídos MP-SoCs usando barramentos como US 1990/4905145, mas a medida que novos núcleos são incluídos, é possível que também seja aumentada a espera de cada núcleo para transmitir algum dado ou diminuída a largura de banda oferecida para cada um. Além disso, o aumento do número de núcleos interligados por barramento aumenta a energia consumida e o

tempo de propagação dos sinais nos fios, devido ao aumento da carga capacitiva desses canais.

[006] Para minimizar o problema da baixa escalabilidade dos barramentos, foi proposto hierarquias de barramentos, Figura 4. Essa solução propõe barramentos mais curtos, possivelmente com diferentes velocidades interligados através de pontes (10). Mesmo assim, esta solução ainda não apresenta a escalabilidade desejada. Outra desvantagem da hierarquia de barramentos é a impossibilidade de reuso, uma vez que é sempre uma solução *ad hoc* para um sistema. Uma variação desse modelo pode ser encontrada em US 2007/0079046. Desse modo, as redes de interconexão em chip ou NoC (*Network-on-Chip*), Figura 5, emergiram como a solução para o problema de baixa escalabilidade e reuso dos mecanismos de interconexão nos projetos de SoC ou Mp-SoC (*Multiprocessor SoC*).

[007] As NoCs são baseadas nos modelos de rede de computadores, com a diferença de estarem dentro do *chip* e, consequentemente, estarem restritas às características próprias desse ambiente. Elas podem ser definidas como um conjunto de roteadores (11) e canais ponto-a-ponto que interconectam os núcleos do sistema (12). A arquitetura de comunicação em uma NoC, geralmente, é baseada em canais bidirecionais ou enlaces (12) chaveados que interligam os núcleos. Esses enlaces permitem comunicações simultâneas entre diferentes núcleos ou entre os mesmos núcleos nos dois sentidos da comunicação. Além disso, as NoCs também podem fazer uso de canais virtuais, que é um o compartilhamento do canal físico. Canais virtuais são implementados a partir de múltiplos *buffers* de entrada e saída, onde vários pacotes são transmitidos entre dois roteadores utilizando canais virtuais diferentes. Nesse processo, o canal físico é alocado de forma alternada (multiplexação no tempo) para a transmissão das palavras de cada pacote. No roteador receptor o processo inverso (demultiplexação) permite recebê-los,armazenando-os em *buffers* diferentes mantendo sua integridade.

[008] Em uma NoC o modelo de comunicação é geralmente baseado na troca de mensagens entre os núcleos do sistema. Essas mensagens são encapsuladas junto com informações adicionais (como o cabeçalho) que são utilizadas pelos roteadores para encaminhá-las da origem até o destino. O conjunto: mensagem (também chamada de carga útil), cabeçalho, e eventualmente um finalizador da mensagem, é comumente chamado de pacote.

[009] Um pacote é normalmente formado por um conjunto de palavras de determinada largura, onde uma ou mais palavras iniciais formam o cabeçalho e as palavras seguintes, a mensagem propriamente dita. O cabeçalho da mensagem contém informações que permitem a um roteador encaminhar o pacote para o próximo roteador a caminho do destino, e este encaminha a um outro, e assim segue até o roteador apto a entregar o pacote para o núcleo destino. No cabeçalho também pode haver outras informações, como comprimento do pacote e informações específicas para o núcleo que receberá a mensagem contida no pacote. Existem diversos modelos de NoC que se diferenciam nas seguintes características: topologia, roteamento, chaveamento, controle de fluxo, arbitragem e armazenamento.

[010] Nos sistemas em *chip* baseados em NoC, os dados e instruções das aplicações circulam sob a forma de pacotes. No trajeto dos pacotes, da origem ao destino, os roteadores interessam-se apenas pelo cabeçalho do pacote sem que nenhum processamento útil, do ponto de vista da execução da aplicação, seja realizado, como em US 2006/6988170. Esse processo de transmissão funciona como um *pipeline,* que poderia ser aproveitado para a execução das instruções e dados da aplicação. Mesmo utilizando meios de comunicação diferentes dos tradicionais, como comunicação ótica empregada na patente US 2006/0036831, ainda assim os MP-SoC utilizam processadores do tipo Von Neumann que podem funcionar em paralelo, mas de forma independente dos demais, deixando o mecanismo de comunicação com o potencial *pipeline* intrínseco destinado unicamente para transmissão dos pacotes entre os núcleos.

[011] Os Computadores Baseados em Fila (*Queue-based computers*) ou Máquinas em Fila (*Queue machines*) propõem um modelo de computação que se aproxima dessa ideia, entretanto tais máquinas são apenas processadores, e não utilizam NoCs. Elas são fundamentadas no uso de referência implícita para uma fila de operandos na execução das instruções, de modo análogo às *stack machine*. Algumas pesquisas apontam que a partir das Máquinas em Fila, é possível construir eficientes máquinas superescalares ou máquinas de fluxos de dados (*data flow machine*). Também podem ser uma nova alternativa para arquiteturas embarcadas devido seus compactos conjuntos de instrução, que promovem códigos densos, alta capacidade de paralelismo em nível de instrução e *hardware* simples, que reduz a área em chip e o consumo de energia.

[012] Arquiteturas de fluxo de dados (*Dataflow Architecture*) também são bem empregadas para processamento de alto desempenho. Em US 2007/0266223 é proposto uma arquitetura de fluxo de dados onde parte da execução das aplicações é realizada em NoC. A arquitetura é formada por um processador e duas redes em chip com topologia em malha (uma rede de dados e uma rede de instruções). Entretanto, a arquitetura não utiliza instruções de desvio (*branch*) como os processadores do tipo Von Neumann, ao invés disso são usadas instruções específicas de *branch* e de união (*merge*) não sendo necessária a utilização da técnica de predição. O processador é responsável pela busca de instruções e operandos e pela execução das instruções fora de laços de repetição. Instruções dentro de laços de repetição devem ser executadas nas NoCs, nesse caso o processador configura os elementos de processamento da rede instruções de acordo com o grafo de fluxo de dados da aplicação que deve ser gerado pelo compilador. Após a configuração da rede de instruções, os dados são distribuídos para a rede de dados através de um barramento localizado na borda superior dessa rede. Essa arquitetura tem escalabilidade comprometida pelo barramento, os grafos de fluxo de dados que determinam a configuração dos elementos de processamento têm tamanho

limitado pelas dimensões da NoC e o processamento feitos pelas NoCs é dependente da busca e configuração das instruções pelo processador. Além disso, o desempenho da arquitetura está relacionado com a forma que as aplicações são programadas, uma vez que a arquitetura explora o paralelismo de instruções dentro de laços de repetição.

[013] Outra proposta que faz uso de redes em chip para o processamento é o chip iWarp. Esse chip é resultado do esforço entre a Universidade Carnegie Mellon e a Intel, que tinha o objetivo de desenvolver um poderoso processador *single-chip* VLSI configurável para diversos sistemas de computação paralela com memória distribuída. iWarp implementa uma variedade de topologia de processadores interconectados. A comunicação pode ser feita através de dois modelos: passagem de mensagem ou sistólica. No modelo de passagem de mensagem os dados são transmitidos através de mensagens onde há um endereço remoto para o receptor da mensagem. No modelo sistólico o iWarp usa um escalonamento estático para assegurar o sincronismo da computação entre os processadores. Entretanto é necessária uma pré-configuração de acordo com as características da aplicação a ser executada.

Breve descrição da invenção

[014] A presente invenção propõe um modelo arquitetural de processador, diferente do tradicional modelo de Von Neumann, e permite o processamento paralelo e distribuído dos programas, implementados em forma de pacotes, em diversas unidades de processamento e roteamento. A invenção possibilita comunicação (transmissão) paralela de pacotes através dos diversos canais ponto-a-ponto entre os elementos de processamento e roteamento e os núcleos de acesso a memória. Permite o armazenamento temporário de pacotes ou partes de pacotes antes da transmissão, que se dá como em um *pipeline* nos elementos de processamento e roteamento que estão entre a origem e o destino dos pacotes, além de possibilitar que pacotes sejam

roteados novamente sempre que chegam aos seus destinos, mas ainda carreguem instruções a serem executadas. Essa invenção é capaz de manter um fluxo de pacotes sem *deadlock*, *livelock* e *starvation*, problemas comumente atrelados a sistemas baseados em NoC. E, sobretudo, oferece recursos e lógica para a execução das instruções que implementam aplicações/programas nos elementos de processamento e roteamento, bem como nos núcleos de acesso a memória sem a necessidade de processadores do tipo Von Neumann.

Breve descrição das figuras

[015] A Figura 1 apresenta um modelo simplificado de um computador do tipo Von Neumann, formado por CPU, memória, dispositivo de entrada e saída, todos ligados por um barramento.

[016] A Figura 2 ilustra de maneira simplificada o ciclo de execução das instruções em processadores do tipo Von Neumann.

[017] A Figura 3 representa um sistema formado por vários núcleos que utilizam um barramento como mecanismo de comunicação.

[018] A Figura 4 ilustra um sistema formado por vários núcleos que utilizam uma hierarquia de barramento como mecanismo de comunicação, onde barramentos de velocidades diferentes se comunicam através de pontes.

[019] A Figura 5 representa um sistema formado por vários núcleos que utilizam uma rede em chip como mecanismo de comunicação.

[020] A Figura 6 ilustra uma instância da invenção, formada por uma rede 4x4, com 16 RPUs e 4 MAUs e 4 módulos de memória.

[021] A Figura 7 apresenta o formato dos pacotes regulares utilizados para descrever aplicações/programas da invenção.

[022] A Figura 8 ilustra o método de execução das instruções e inserção dos resultados no mesmo pacote.

[023] A Figura 9 ilustra um simples exemplo e sua dependência de dados.

[024] A Figura 10 apresenta o pacote regular da invenção correspondente ao exemplo anterior.

[025] A Figura 11 ilustra o caminho da primeira espiral que um pacote segue utilizando o algoritmo *spiral complement*.

[026] A Figura 12 ilustra o caminho da segunda espiral que um pacote segue utilizando o algoritmo *spiral complement*.

[027] A Figura 13 ilustra o caminho da terceira espiral que um pacote segue utilizando o algoritmo *spiral complement*.

[028] A Figura 14 ilustra o caminho da quarta espiral que um pacote segue utilizando o algoritmo *spiral complement*.

[029] A Figura 15 representa a arquitetura interna da RPU.

[030] A Figura 16 apresenta o fluxograma do funcionamento da RPU.

[031] A Figura 17 representa a arquitetura interna da MAU.

[032] A Figura 18 apresenta o formato dos pacotes de controle utilizados para descrever aplicações/programas da invenção.

Descrição detalhada da invenção

[033] A invenção ilustrada através da Figura 6, é formado por uma rede de elementos de processamento e roteamento (13) ou RPU (*Routing and Processing Unit*), ligando as unidades de acesso a memória (14) ou MAU

(*Memory Access Unit*) nos cantos da rede juntamente com os módulos de memória (15) também nos cantos. Essa rede possui topologia em grelha 2D de dimensão quadrada, ou seja, o número de linhas igual ao número de colunas; utiliza, no mínimo, dois canais virtuais; roteamento baseado no padrão XY; chaveamento que combina VCT e *wormhole*; controle de fluxo baseado em crédito; arbitragem distribuída; armazenamento na entrada em *buffers* do tipo FIFO (*First In First Out*); e enlaces de comunicação entre as RPUs e MAUs com 36 bits de largura bits *full-duplex*. Essas características são comuns em redes em chip, entretanto algumas delas sofreram alterações, em relação ao que é tradicionalmente usado, de acordo com as particularidades do sistema. A Figura 6 ilustra uma instância da invenção formada por uma rede 4x4, com 16 RPUs (13) em forma de grelha 2D e as 4 MAUs (14) nos cantos da rede, cada uma ligada a um módulo de memória (15). Cada um dos componentes (RPU e MAU) é endereçável através da sua posição no plano cartesiano.

[034] Os pacotes utilizados na invenção são estruturas contendo instruções e operandos a serem processados durante o roteamento. Esse sistema aproveita o comportamento *pipeline* da transmissão, de modo que os pacotes passam a ser uma sequência de instruções que constituem uma aplicação. Na estrutura dos pacotes a instrução corrente é aquela que está no início do pacote. Os resultados das instruções podem ser inseridos em qualquer parte do pacote, eventualmente servindo de operandos para as instruções seguintes, nas posições indicadas pela própria instrução. Esse modelo de processamento permite execução paralela baseada nos múltiplos canais ponto-a-ponto da rede em chip. Para tanto, cada RPU executa a primeira instrução do pacote e encaminha as demais para a próxima RPU, desde que o canal de transmissão esteja disponível. Quando o canal de transmissão está indisponível a RPU entra em modo de execução local, que será apresentada mais adiante.

[035] O pacote utilizado na invenção é um conjunto variável de palavras de trinta e dois *bits* de largura, onde as três primeiras palavras formam o cabeçalho (16), seguidas pelas palavras com instruções (17) e operandos (18),

terminado por uma palavra que indica o fim do pacote (19). Além dos trinta e dois *bits* de largura do pacote, são usados mais quadro *bits* sinalizadores que informam o tipo de palavra transmitida: cabeçalho (20) ou *header*; instrução (21) ou *instruction*; operando (22) ou *operand*; e fim de pacote (23) ou *terminator*. A Figura 7 apresenta o formato do pacote junto aos *bits* de controle ou *control bits*. Como as palavras dos pacotes possuem 36 *bits* (4 para *bits* de controle e 32 para informações) os enlaces entre as RPUs e as MAUs também possuem 36 bits de largura.

[036] Em cada palavra apenas um dos quatro *bits* do controle pode estar ativo (nível lógico um), todos os demais devem estar desativados (nível lógico zero). A seguir é feita a descrição dos campos de cada palavra, com os respectivos tamanhos em *bits*.

- Cabeçalho (16) – apenas o 1º. *bit* do controle (20) pode estar ativo.
    - Destino (24) ou *destination*: 8 *bits* para indicar o endereço da RPU final do roteamento atual.
    - Origem (25) ou *source*: 8 *bits* para indicar o endereço da RPU que iniciou o roteamento atual.
    - Número de instruções (26) ou *number of instructions*: 8 *bits* para indicar a quantidade de instruções corrente contidas no pacote.
    - Re-rotear (27) ou *re-route*: 6 *bits* que indicam a forma de calcular o novo endereço de destino quando necessário. No algoritmo proposto, *spiral complement,* que será detalhado mais adiante, apenas 3 *bits* são utilizados, nomeados de Prox, CX e CY, nessa ordem.
        - Prox: 1 *bit* que indica qual a próxima coordenada a ser alterada (‘0’ para X; ‘1’ para Y).

- CX: 1 *bit* que indicar como alterar a coord. X (‘0’= incremento; ‘1’= decremento).
- CY: 1 *bit* para indicar como alterar a coord. Y (‘0’= incremento; ‘1’= decremento).

- Tipo de pacote (28) ou *type*: 2 *bits* usados para indicar o tipo do pacote. É possível identificar até quatro tipos de pacotes, entretanto, atualmente são usados apenas dois tipos de pacotes (“00_b” = pacotes regulares; “01_b” = pacotes de controle).
- Número do pacote (29) ou *number of packet*: 32 *bits* usados como identificador único do pacote para uma dada aplicação.
- Apontador (30) ou *pointer*: 32 *bits* usados para indicar a próxima instrução a ser executada no pacote.

- Instrução (17) – apenas o 3º. *bit* do controle (21) pode estar ativo
  - Instrução (31) ou *instruction id*: 8 *bits* que identificam a instrução a ser executada
  - NO (32) ou *number of operands*: 2 *bits* que indicam o número de operandos necessário para executar a instrução (“00_b” = nenhum operando; “01_b” = 1 operando; “10_b” = 2 operandos; “11_b” = vários operandos, a quantidade deve ser definida no campo Resultado_2 (34))
  - Resultado_1 (33) ou *Result_1*: 11 *bits* que indicam a palavra ou posição relativa no pacote onde o resultado da instrução deve ser inserido; a contagem das palavras é feita a partir de zero (primeira palavra do cabeçalho) e deve considerar o número de operandos necessários para executar cada instrução, mesmo que os operandos ainda não tenham sido inseridos; em instruções de

controle (instruções executadas por uma MAU), esse campo é usado para indicar o endereço da MAU que deve executá-la; possui o valor zero quando esse campo não está sendo usado.

- Resultado_2 (34) ou *Result_2*: 11 bits que tem duas funções; quando o campo “NO” (32) possui valor diferente de “11_b” indica uma segunda palavra ou posição relativa no pacote onde o resultado da instrução deve ser inserido; a contagem das palavras é feita a partir de zero (primeira palavra do cabeçalho) e deve considerar o número de operandos necessários para executar cada instrução, mesmo que os operandos ainda não tenham sido inseridos; quando o campo “NO” (32) possui valor “11_b” indica a quantidade de operandos necessário para executar a instrução; possui valor zero quando a instrução não retorna resultado ou quando o resultado retornado não deve ser inserido em mais que uma posição do pacote

- Operando (18) ou *operand* – apenas o 4º. *bit* do controle (22) pode estar ativo

  - Operando (35) ou *operand*: 32 *bits* usados como dado ou operando da instrução

- Fim de pacote (19) ou *terminator*– apenas Só o 2º. *bit* do controle (23) pode estar ativo

  - Fim (36) ou *terminator*: 32 *bits* usados como padrão de fim de terminador do pacote, todos os *bits* no nível lógico zero.

[037] Com o formato de pacote apresentado na Figura 7 é possível escrever programas para serem executados no sistema. No modelo de computação desse sistema cada RPU é responsável por remover a primeira instrução do

pacote, executá-la ou enviá-la à MAU que deve executá-la, armazenar os resultados (se houverem) juntamente com a posição onde devem inseri-los no pacote, transmitir o restante do pacote para a próxima RPU de acordo com o algoritmo de roteamento e inserir os resultados (se houverem) no momento da transmissão do pacote onde devem ser incluídos.

[038] Na Figura 8 é apresentado um pacote hipotético (37) contendo no mínimo duas instruções, cada uma delas necessitando de dois operandos (38) e (39). Nesse exemplo, a primeira instrução do pacote (40) indica no campo “Resultado_1” (41) que o resultado dessa instrução deve ser inserido na sétima palavra do pacote (contagem global, incluindo as palavras removidas) para servir de operando para a instrução seguinte (42). Logo após a execução da instrução (40) a RPU que a executou transmite o restante do pacote (sem a instrução executada e seus operandos) para RPU seguinte, de acordo com o algoritmo de roteamento. O pacote resultante (43) tem como primeira instrução (42) aquela que era vista como segunda instrução na RPU anterior, incluindo o dado (44) gerado e inserido na RPU anterior depois de executada a instrução (40).

[039] Dessa forma, o pacote diminui à medida que é executado. Nas tradicionais implementações de sistemas integrados baseados em redes em chip a carga de dados inseridos na rede pode ser um problema, dependendo da comunicação requerida pelas aplicações. Nesse sistema, em média, a carga de dados diminui à medida que o pacote flui pela rede, o que diminui as chances de congestionamentos e aumenta a capacidade da rede receber novos pacotes mais rapidamente.

[040] Aplicações/programas executados na invenção devem estar descritos no formato do pacote apresentado na Figura 7, que além do cabeçalho e terminador, possui uma série de instruções e dados, esses pacotes são chamados de pacotes regulares. Aplicações/programas são descritas através de um ou mais pacotes, os quais são injetados e executados no sistema, na

ordem da dependência dos dados entre as computações que cada pacote representa. Do mesmo modo, um pacote é um conjunto de instruções com seus respectivos dados, empilhados de acordo com a dependência de dados. Assim uma instrução é executada apenas quando estiver no topo dessa pilha, ou seja, no início do pacote. Com isso, o resultado de uma instrução executada pode fornecer dados que servirão de operandos para outras instruções a serem executadas posteriormente. Desse modo a arquitetura oferece um conjunto mínimo de instruções, que ainda pode ser estendido. A tabela a seguir sumariza as instruções e informações sobre o tipo, quantidade de operandos necessário e uma breve descrição de cada uma.

Instrução	**Tipo**	**# Operandos**	**Descrição**
ADD	Aritmética	2	Soma 2 inteiros
SUB	Aritmética	2	Subtrai 2 inteiros
MUL	Aritmética	2	Multiplica 2 inteiros
DIV	Aritmética	2	Divide 2 inteiros
NOT	Lógica	1	Negação de 1 valor
AND	Lógica	2	Conjunção de 2 valores
OR	Lógica	2	Disjunção de 2 valores
XOR	Lógica	2	Ou-exclusivo de 2 valores

RSHIFT	Deslocamento	2	Desloca *n* bits de um valor à direita
LSHIFT	Deslocamento	2	Desloca *n* bits de um valor à esquerda
LOAD	Acesso à Memória	1	Solicita um valor da memória
RELOAD	Acesso à Memória	1	Retorna um valor carregado da memória
STORE	Acesso à Memória	Vários	Armazena um valor na memória
EXEC	Sincronização	1	Ordena a injeção imediata de um pacote
SYNEXEC	Sincronização	Vários	Ordena a injeção de um pacote após sincronização
SYNC	Sincronização	1	Sinal de sincronização de um pacote
SEND	Sincronização	2	Envia um valor para um ser inserido em um pac.
BE	Condicional	2	Desvia se igual
BNE	Condicional	2	Desvia se diferente
BL	Condicional	2	Desvia se menor

BG	Condicional	2	Desvia se maior
BLE	Condicional	2	Desvia se menor ou igual
BGE	Condicional	2	Desvia se maior ou igual
JUMP	Incondicional	0	Desvia incondicionalmente
COPY	Auxiliar	1	Copia 1 valor para outra instrução no mesmo pac.
NOP	Auxiliar	0	Sem operação

[041] As instruções lógicas e aritméticas utilizam os campos dos pacotes assim como foram propostos inicialmente, ou seja, a palavra de instrução indica a operação a ser realizada na ULA e a posições relativas no pacote (até 2 posições) onde o resultado da operação será inserido. O(s) operando(s) usado(s) na operação são encontrados na(s) palavra(s) logo após a palavra de instrução.

[042] As instruções lógicas, aritméticas, deslocamento, condicionais e auxiliares chegam as RPUs em pacotes regulares e por isso são chamadas de instruções regulares. Tais instruções são executadas nas RPUs que as identificam, enquanto que as instruções de acesso à memória e sincronização, são identificadas pelas RPU e enviadas para as MAUs através de pacotes de controle, onde são executadas. Por esse motivo, tais instruções são chamadas de instruções de controle. A única instrução de controle executada na RPU é a instrução RELOAD, a qual traz o valor carregado da memória, solicitado anteriormente por uma instrução de LOAD na mesma RPU. Na verdade, a execução do RELOAD é apenas a inserção do valor carregado da memória no

pacote. Quando a RPU identifica uma instrução de LOAD, ele cria um pacote de controle com esta instrução e envia-o à MAU que deve executá-la. Nesse momento, a execução do pacote que originou o pacote de controle é interrompida até que o resultado do LOAD retorne para RPU através de um RELOAD.

[043] As instruções de controle devem ser executadas em uma MAU específica, exceto RELOAD, desse modo, os campos “Resultado_1” (33) e “Resultado_2” (34) não indicam as posições onde o resultado será inserido. Todas as instruções (exceto RELOAD) utilizam o campo “Resultado_1” (33) para indicar o endereço da MAU que deve executar a instrução. Apenas as instruções LOAD e SEND utilizam o campo “Resultado_2” (34) como instruções regulares. LOAD utiliza “Resultado_2” (34) para indicar a posição no mesmo pacote onde será inserido o valor carregado da memória trazido por um RELOAD. A instrução SEND utiliza “Resultado_2” (34) para indicar a posição em outro pacote (identificado pelo número do pacote no campo do primeiro “Operando” (35) dessa instrução), para inserir o valor presente no campo do segundo “Operando” (35) dessa instrução. O valor enviado através de SEND é inserido por uma MAU em um pacote que ainda esteja na memória, antes de ser injetado no sistema.

[044] Na invenção, um valor pode ser armazenado em várias posições de memória através de uma única instrução de STORE. Desse modo, o valor a ser armazenado aparece no primeiro campo de “Operando” (35), seguido por vários endereços de memória nos campos de “Operandos” (35) seguintes. Como o número de operandos é variável, o campo “NO” (32) da instrução recebe o valor “11_b” e o número exato de operandos (quantidade de endereços) é informado no campo “Resultado_2” (34).

[045] Quando uma MAU recebe uma instrução EXEC, ela injeta assim que possível o pacote armazenado na memória, identificado pelo número no único

campo de “Operando” (35) dessa instrução. A instrução EXEC não utiliza o campo “Resultado_2” (34).

[046] Entretanto, a injeção do pacote pode estar condicionada a sincronização da execução de outros pacotes. Esse tipo de injeção é indicado através da instrução SYNEXEC. Essa instrução identifica o número do pacote a ser injetado, no primeiro campo de “Operando” (35), e os números de todos os outros pacotes que devem enviar sinais de sincronização, nos campos de “Operandos” (35) seguintes. A injeção do pacote, solicitada, acontecerá apenas depois que todos os sinais de sincronismos dos outros pacotes chegarem à MAU que o injetará. Contudo, a MAU não é bloqueada para outras atividades, ela continua executando outras instruções que chegam até ele. Assim como STORE, a instrução SYNEXEC tem o seu campo “NO” (32) preenchido como o valor “11_b” e o número de operandos (quantidade de sinais de sincronização) indicados no campo “Resultado_2” (34).

[047] Quando a RPU encontra uma instrução SYNC em um pacote, ela envia para a MAU específica (identificado no campo “Resultado_1” (33)) com o número do pacote que espera o sinal de SYNC como primeiro “Operando” (35) e o número do pacote que trouxe o SYNC como o segundo “Operando” (35)**.** O campo “Resultado_2” (34) não é usado. No pacote regular essa instrução apresenta só um operando (o número do pacote que espera a sincronização) o outro operando é inserido apenas no pacote de controle, pelo SU (51), quando está sendo encaminhado para a MAU (14) que executará a instrução.

[048] As instruções de desvio condicional usam os campos com o propósito original, com algumas peculiaridades. Como instruções de desvio condicional não produzem resultados, o campo “Resultado_1” (33) é usado para informar a posição no pacote onde a execução deve ser desviada quando a condição é satisfeita. As instruções de desvio condicional, utilizando dois operandos. Os operandos são comparados e se a condição for satisfeita, todas as palavras a partir da instrução até a posição de desvio (indicada por “Resultado_1” (33))

são descartadas. Caso a condição seja falsa apenas a instrução de desvio é descartada e a próxima instrução no pacote é executada normalmente. A única instrução de desvio que possui uma pequena modificação nesse formato é a instrução JUMP, uma vez que um desvio incondicional não utiliza operandos para comparação, sendo formada apenas pela palavra da instrução.

[049] Finalmente, o conjunto de instruções possui duas instruções auxiliares: COPY e NOP. A primeira é usada para replicar um valor para ser usado como operando de outras instruções, em até duas posições no pacote. A instrução NOP não produz resultados nem modifica o pacote, é uma instrução que não realiza nenhuma operação.

[050] O conjunto de instruções são usados para implementações de programas a serem executados nessa arquitetura, contudo algumas considerações devem ser feitas: (i) um pacote injetado (pacote em execução) é uma cópia do pacote que permanece na memória e que tem uma identificação (número do pacote). Dessa forma é possível traduzir o identificador do pacote para o endereço de memória onde ele está armazenado, e também é possível modificar o valor de um operando no pacote da memória, tendo uma nova instância do pacote quando ele for injetado novamente; (ii) a invenção não utiliza registradores, então variáveis de programas são sempre posições de memória que podem ser lidas ou escritas. Entretanto, durante a execução das instruções nos pacotes, o valor das variáveis e os resultados intermediários podem ser inseridos no pacote como operando de uma instrução no mesmo ou em outro pacote; (iii) as instruções são executadas na ordem que aparecem nos pacotes, e nunca pode acontecer um desvio do fluxo da execução para instruções anteriores, entretanto repetições podem ser executadas usando a propriedade (i), (iv) e (v); (iv) um pacote pode ser injetado tantas vezes quanto for necessário, dependendo de quantas instruções EXEC ou SYNEXEC foram enviadas para uma MAU injetá-lo; (v) um pacote pode se auto-executar, desde que haja pelo menos uma instrução EXEC ou SYNEXEC, dentro do pacote, para injetá-lo da memória para o sistema novamente.

[051] A Figura 9 ilustra a dependência de dado da simples expressão (y+x)*(x-z). O pacote que representa essa expressão para execução no sistema é ilustrado na Figura 10.

Algoritmo de roteamento

[052] O número de instruções, e operandos, é variável em cada pacote. O tamanho do pacote, as instruções e dados que ele possui, bem como as posições onde os dados calculados devem ser inseridos, são definidos pelo grafo de fluxo de dados da aplicação. Em geral, cada RPU executa apenas uma instrução por vez, de um pacote. Dessa forma, o destino de cada pacote deve ser suficientemente distante da origem para que todas as instruções contidas no pacote possam ser executadas, considera-se a distância entre destino e origem como o número de RPUs intermediárias entre origem e destino. Entretanto, o mais provável é que a distância entre origem e destino, de uma rede com dimensões limitadas, não seja suficiente para a execução de todas as instruções de qualquer pacote que represente uma aplicação real. Em outras palavras, sempre existirá uma aplicação que possua mais instruções que o número de RPUs do maior caminho que o pacote possa trafegar. Neste caso o pacote deve, ao chegar ao destino, ser novamente roteado, isto é, um novo destino deve ser definido, a fim de que a execução continue.

[053] Portanto, a invenção utiliza um algoritmo de roteamento próprio capaz de encontrar um novo destino, quando o pacote atinge o destino atual e ainda possui instruções a serem executadas, de modo que estas instruções contidas no pacote possam também ser executadas. O algoritmo desenvolvido foi denominado *spiral complement*, e é capaz de rotear novamente um pacote, tantas vezes quanto for necessário, até que seja garantida a execução de todas as instruções, independentemente do número de instruções e das dimensões da rede (número de RPUs). Esse algoritmo é inspirado roteamento XY e no padrão de tráfego *complement*.

[054] No roteamento XY, é considerado que cada nodo da grelha 2D é endereçado pelas coordenadas XY do plano cartesiano onde estão situados. Dessa forma o pacote é encaminhado no sentido do eixo X até a coluna da coordenada Y do destino e em seguida segue no sentido do eixo Y até a linha da coordenada X do destino, quando finalmente o pacote é entregue ao seu receptor.

[055] O *Complement* é um padrão de tráfego, independente do algoritmo de roteamento, usado para calcular os destinos dos pacotes em simulações, normalmente usadas para avaliação de desempenho de NoCs. Nesse padrão o destino é calculado através da inversão dos *bits* da origem, isto é, o complemento da origem. No padrão *complement* o nodo origem que possui coordenadas no formato binário $a_{n-1}, a_{n-2}, \dots, a_1, a_0$ tem como destino o nodo com coordenadas $\neg a_{n-1}, \neg a_{n-2}, \dots, \neg a_1, \neg a_0$.

[056] No algoritmo *spiral complement*, o destino inicial de um pacote é o complemento da origem. Então, quando um pacote atinge seu destino e o número de instruções contidas no pacote é maior que zero, um novo destino deve ser calculado, ou seja, é feito um novo roteamento. Neste novo roteamento, a origem passa a ser o antigo destino, e o nodo corrente passa a ser a origem atual, o qual calcula o novo destino, levando-se em consideração a necessidade de se obter a melhor distribuição de carga possível entre os canais da NoC.

[057] Na invenção os pacotes são inicialmente injetados em um dos quatro cantos da grelha. Isso acontece devido às maiores distâncias entre origem e destino em uma rede nessa topologia, serem os cantos opostos (por exemplo, os nodos 0,0 e 3,3 em uma grelha 4x4), esse também é o motivo dos módulos de memória estarem nos cantos da rede. Quando um pacote precisa ser novamente roteado, o novo destino é calculado como sendo o nodo mais distante diferente da última origem, caso contrário o pacote circularia apenas entre um par de nodos. Dessa forma, o algoritmo reduz virtualmente a rede em

uma linha ou uma coluna e o novo destino é o nodo mais distante dessa rede reduzida.

[058] Com isso, para o cálculo de um novo destino, o algoritmo *spiral complement* leva em conta duas informações: (i) o antigo destino do pacote (nó corrente que calcula o novo destino); e (ii) a antiga origem (nó que enviou o pacote ao nó corrente). Ao considerar o antigo destino o algoritmo verifica se este está em um dos quatro cantos da rede. Se não estiver, o novo destino é calculado como sendo a antiga origem com o incremento ou decremento da coordenada X ou coordenada Y em uma unidade. A operação de incremento ou decremento de uma das coordenadas, geralmente, é alternada e a coordenada afetada também alterna em cada novo cálculo. Se o antigo destino (nó que calcula o novo destino) estiver em um dos cantos da rede o algoritmo considera ainda a antiga origem (nó que o enviou o pacote). Se a antiga origem for um dos quatro cantos da rede o novo destino é calculado em dois passos: no primeiro é calculado o complemento do nó corrente e no segundo acontece o incremento ou decremento, de uma unidade, de umas das duas coordenadas, X ou Y. Entretanto, se a antiga origem não for um dos quatro cantos da rede o novo destino será calculado em um único passo, o complemento do nó corrente. As Figuras 11, 12, 13 e 14 mostram o roteamento utilizando o algoritmo *spiral complement* para uma matriz de 4x4 RPUs, para pacotes originados em cada um dos quatro cantos da rede. A transmissão propriamente dita dos pacotes, entre origem e destino, é realizada utilizando o roteamento XY. O algoritmo de roteamento *spiral complement* pode ser resumido como:

- Se destino inicial
    - Complemento do nó corrente
- Senão
    - Se o nó corrente estiver em um dos 4 cantos

- Se o nó que enviou o pacote for o complemento do nó corrente
    - Calcula o complemento do nó corrente
    - Incremento/decremento de X ou Y do complemento (alternado)
- Senão
    - Complemento do nó corrente
- Senão
    - Endereço do nó que enviou o pacote com incremento/decremento de X ou Y (alternado)

[059] Na Figura 11 o pacote é injetado inicialmente no canto superior esquerdo (45), então o primeiro destino é calculado como sendo o complemento, ou seja, o canto inferior direito (46). Nos demais cálculos dessa espiral, os novos destinos são calculados alternando entre incremento da coordenada X e decremento da coordenada Y da origem atual do pacote. Quando o pacote transmitido atinge seu destino o canto inferior direito (46), o segundo destino é calculado com os dois passos descritos anteriormente: complemento da origem atual e o incremento de uma unidade da coordenada X do complemento calculado. No exemplo o complemento é o endereço 0,0 que após o incremento da coordenada X determinará o novo destino, o endereço 1,0. O pacote é transmitido para o novo destino (47) com a nova origem (endereço 3,3). Em seguida tem-se que a nova origem não está em um dos quatro cantos da rede. O próximo destino é então calculado subtraindo-se uma unidade à coordenada Y da origem (endereço 3,3), resultando no endereço 3,2 (48) como destino. Ao chegar ao endereço 3,2, que também não está em um dos quatro

cantos da rede, o processo se repete, e o próximo destino é calculado como sendo a coordenada do endereço origem (endereço 1,0) com o incremento da coordenada X de uma unidade, resultando no endereço 2,0. E assim segue a execução do algoritmo *spiral complement*, alternando entre incremento da coordenada X e decremento da coordenada Y da origem atual do pacote. O algoritmo provoca o movimento do pacote como em uma espiral levando-o até o canto inferior esquerdo (49).

[060] Quando o pacote tem origem no canto inferior esquerdo (49), após o complemento, os destinos são calculados alternando-se apenas a coordenada afetada por operação de incremento. Neste caso o algoritmo inicia com o incremento da coordenada Y e no cálculo do próximo destino com o incremento da coordenada X da antiga origem, realizando a segunda espiral, isto é, os novos destinos vão levando o pacote para o canto inferior direito da rede (46), como mostra a Figura 12. Para pacotes originados no canto inferior direito (46), os destinos são calculados alternando em decremento da coordenada X e incremento da coordenada Y da antiga origem, realizando a terceira espiral até o canto superior direito (50), como ilustra a Figura 13. E, quando são originados no canto superior direito (50) a alternância acontece entre o decremento da coordenada Y e o decremento da coordenada X da origem, realizando a quarta espiral, como ilustra a Figura 14.

[061] O destino final de cada espiral é igual à primeira origem da espiral seguinte, e que o último destino (da quarta espiral) coincide com a origem da primeira espiral (45), dessa forma, se o pacote percorre toda uma espiral, os próximos destinos a serem calculados serão aqueles da espiral seguinte, como se o pacote tivesse sido injetado pelo nodo inicial dessa nova espiral, fechando o ciclo e permitindo o roteamento recomeçar.

Routing and Processing Unit (RPU)

[062] As RPUs (*Routing and Processing Unit*) (13) são os componentes com capacidade de processamento e roteamento. Desse modo, a arquitetura da RPU apresenta uma Unidade Lógica e Aritmética ou ALU (2) e uma Unidade de Sincronismo ou SU (51), um par de *buffers* (52) do tipo FIFO (*First In First Out*) em cada porta de entrada, um árbitro (53) para cada porta de saída e um *crossbar* (54), como representado na Figura 15. A ALU (2) executa operações lógicas e aritméticas, desse modo, a maioria das instruções executadas na RPU faz uso da ALU. Enquanto a SU (51) é responsável pela criação de pacotes de controle a partir de instruções que devem ser executadas em uma MAU específica.

[063] Dependendo da localização, a RPU pode ter três porta de entrada/saída (RPU das bordas da rede) ou quatro portas (RPU do interior da rede). As portas são identificadas como norte, sul, leste e oeste. Cada porta de entrada possui dois *buffers* (52), um para cada canal virtual, onde são armazenados os pacotes que chegam à RPU. Cada canal virtual é destinado a um tipo de pacote (pacote regular ou pacote de controle). O *crossbar* é parcialmente conectado, onde um pacote pode ser chaveado de uma porta de entrada para qualquer saída, exceto a saída correspondente a porta por onde o pacote entrou. Os árbitros (53) que além de controlar as disputas pelo canal de transmissão (usando *round-robin*), como em um roteador tradicional, também é responsável por remover a instrução no início do pacote para solicitar execução e posteriormente inserir o resultado no mesmo pacote durante a transmissão.

[064] As funções primordiais da RPU, ilustrada na Figura 16, são: rotear pacotes, executar instruções contidas nos pacotes, inserir os resultados ou gerar pacotes de controle e transmiti-los.

[065] Quando um pacote chega a um dos *buffers* de entrada da RPU, primeiramente é verificado o tipo de pacote no campo **“**tipo” (28) do cabeçalho. Se esse campo tem o valor igual a “01_b”, trata-se de um pacote de controle, então o pacote deve apenas ser roteado e transmitido para a próxima RPU,

sem alterar o destino nem executar as instruções da sua carga útil, através do canal virtual exclusivo para pacotes de controle. Se esse campo possui o valor “00_b”, trata-se de um pacote regular, o qual possui instruções que devem ser executadas uma a uma, em cada RPU por onde o pacote trafega até que todas as instruções tenham sido executadas.

[066] Para pacotes regulares, a RPU corrente deve verificar se o destino atual no cabeçalho do pacote é igual ao seu endereço. Em caso afirmativo, o pacote deve ser novamente roteado caso haja mais instruções a serem executadas aplicando-se o algoritmo *spiral complement*. Nas RPUs localizadas em um dos quatro cantos da rede, o novo destino será o complemento, caso a origem também não seja uma RPU que se encontra nos cantos da rede (neste caso o cálculo do complemento levaria o pacote de volta à antiga origem). Nos outros casos, o destino será o valor da antiga origem do pacote com alguma modificação, de acordo com o algoritmo *spiral complement*. Isso é feito com o auxílio dos campos “Origem” (25) e “Rerotear” (27) do cabeçalho. No algoritmo *spiral complement* apenas 3 *bits* do campo “Rerotear” (27) são usados e nomeados respectivamente de “Prox”¸ “CX” e “CY”. A modificação do valor da origem se dá em apenas uma das coordenadas (X ou Y), indicada no *bit* “Prox”. Caso seja alterada a coordenada X o *bit* “CX” indica se será feito um incremento ou decremento dessa coordenada. Caso seja a coordenada Y, a modificação a ser realizada (incremento ou decremento) será indicada pelo *bit* “CY”. Em seguida, o *bit* do campo “Prox” é invertido para que no próximo cálculo de novo roteamento seja realizado na outra coordenada. Assim, o valor de apenas uma das coordenadas da origem é modificado, alternadamente, a cada novo roteamento, e esse valor passa a ser o novo destino do pacote. A nova origem do pacote é o endereço da RPU que calculou o novo destino. Com um novo endereço ou o endereço corrente, o pacote é roteado para uma das portas de saída usando o roteamento XY tradicional.

[067] Na porta de saída, o árbitro remove a primeira instrução e os respectivos operandos do pacote e identifica seu tipo. Caso seja uma instrução de controle,

o árbitro solicita a Unidade de Sincronização (51) que gere um pacote de controle com tal instrução, transmitindo-o, através do canal virtual exclusivo para este tipo de pacote, para a MAU que deve executá-la. Caso a instrução seja regular, o árbitro solicita a operação da ALU (2) sobre os operandos e os resultados retornados por ela são armazenados pelo árbitro em *buffers* de resultados juntamente com a informação do(s) número(s) da(s) palavra(s), no pacote, onde este(s) resultado(s) deve(m) ser inserido(s). Essas informações são indicadas na própria instrução através dos campos “Resultado_1” (33) e “Resultado_2” (34)**.** Independente do tipo de instrução, os campos “Apontador” (30) e “Num_Instruções” (26) do cabeçalho são atualizados e o restante do pacote (sem a instrução e os dados recém-executados) deve ser encaminhado para a próxima RPU.

[068] No momento da transmissão o árbitro verifica a disponibilidade dos canais de transmissão e dos *buffers* de destino na próxima RPU (protocolo baseado em crédito). Caso não seja possível transmitir, o árbitro solicita a execução da próxima instrução do pacote, entrando no estado de execução localizada, como prevenção a um eventual *deadlock*. Sendo possível a transmissão, o árbitro mantém um canal exclusivo com a próxima RPU até que seja transmitida a última palavra do pacote desse *buffer*. Durante a transmissão, um contador, iniciado com o valor do campo “Apontador” (30), é incrementado a cada palavra transmitida. Quando o contador torna-se igual a “Resultado_1” (33) ou “Resultado_2” (34) da instrução executada, o árbitro interrompe a transmissão do *buffer* e transmite o resultado retornado pela ALU (armazenado no *buffer* de resultados), em seguida retoma a transmissão do *buffer* de entrada novamente. A cada novo pacote o contador é iniciado com o valor do campo “Apontador” (30), permitindo assim uma contagem global das palavras do pacote mesmo em RPUs diferentes. Além do “Apontador” (30), outro campo que auxilia a contagem durante a transmissão de instruções é o campo “NO” (32) (localizado na palavra de instrução), que indica o número de operandos necessário para executar a instrução. Quando a instrução é seguida por todos

os seus operandos, o contador de palavras é simplesmente incrementado de um a cada palavra transmitida. Contudo, se os operandos de uma instrução serão inseridos posteriormente, através da execução de outras instruções, o contador de palavras é somado à diferença entre o valor de “NO” (32) e a quantidade de operandos da instrução presentes no pacote. Enquanto um pacote de um dos *buffers* de entrada está sendo transmitido, os demais *buffers* que solicitam o árbitro têm suas instruções executadas, através da requisição da ALU (2) ou da SU (51), até que sejam selecionados pelo árbitro para serem transmitidos. Dessa forma, os pacotes que disputam o canal de saída terão suas instruções executadas mesmo que não possam ser transmitidos. Também nos casos em que um *buffer* é selecionado para a transmissão, mas o espaço disponível no receptor é insuficiente, a ALU ou a SU são novamente requisitados até que a transmissão possa ser realizada. Essa estratégia, chamada de modo de execução local, é usada para prevenir contra uma possível situação de *deadlock*.

Modo de Execução Local

[069] Como a invenção é baseada em uma rede em *chip*, então tal sistema deve prever e/ou tratar os problemas decorrentes desse subsistema de comunicação. Um deles é a ocorrência de *deadlock* provocado pela dependência cíclica entre pacotes, impossibilitando o tráfego nos canais de transmissão. Como nesse sistema os pacotes possuem comprimentos variados, e o algoritmo de roteamento utilizado (*spiral complement*) permite um mesmo pacote passar inúmeras vezes pelos mesmos canais, uma situação de *deadlock* seria inevitável, dependendo do comprimento do pacote e da dimensão da rede.

[070] Considere um pacote injetado pela RPU 0,0 (45), em uma rede 4x4, como mostra a Figura 11. Considerando uma operação de roteamento como sendo a transmissão de um pacote entre duas RPUs adjacentes a caminho do destino

desse pacote, após quinze operações de roteamento, segundo o algoritmo *spiral complement*, o cabeçalho do pacote chega à RPU 3,2 (48) pela porta norte dessa RPU, enquanto palavras do mesmo pacote continuam a chegar e serem roteadas da porta leste para a saída oeste da mesma RPU.

[071] Neste caso, o *buffer* da entrada norte está livre para receber o cabeçalho do pacote. Entretanto, de acordo com o *spiral complement*, o pacote deveria ser novamente roteado e transmitido, da porta norte para a saída oeste, chegando novamente a RPU 3,1 (55). Como o canal de transmissão entre as RPUs 3,2 (48) e 3,1 (55) está ocupado, transmitindo outra parte do mesmo pacote, o cabeçalho do pacote ficará a espera da liberação desse canal, provocando a ocupação total dos *buffers* de entrada em toda a rota do pacote. Esta situação caracteriza um *deadlock*, pois o cabeçalho não libera espaço no *buffer* que ocupa, por estar esperando liberação de espaço no *buffer* de entrada da porta leste da RPU 3,1 (55), entretanto, para que isto aconteça é necessário que, o cabeçalho libere espaço, e assim continua a espera mútua.

[072] Para evitar que aconteça algum *deadlock*, a invenção utiliza a solução chamada de execução local por se tratar da execução de inúmeras instruções em uma mesma RPU. Como apresentado, os árbitros de uma RPU são os responsáveis pela solicitação da ALU ou da SU, para execução de uma instrução, e pela transmissão dos pacotes pela porta de saída. Desse modo, quando o cabeçalho de um pacote não pode ser transmitido (devido ao canal estar ocupado ou por não haver espaço suficiente no *buffer* destino), o árbitro solicita novamente a ALU ou a SU para executar a próxima instrução do pacote. Assim, a cada instrução executada o árbitro tenta transmitir o pacote, e no caso de insucesso, volta a executar a próxima instrução do pacote. Ressalva-se que a execução acontece somente na RPU que está tentando transmitir o cabeçalho, uma vez que apenas as instruções do início do pacote podem ser executadas, nos demais nodos o pacote só pode ser transmitido.

Memory Access Unit (MAU)

[073] A arquitetura da invenção propõe a descrição das aplicações através de um ou mais pacotes no formato apresentado na Figura 7. Tais pacotes devem ser armazenados na memória e os números dos pacotes (29) devem ser associados aos endereços de memória onde estão armazenados. A memória é formada por quatro módulos (15), fisicamente distribuídos nos cantos da rede, como apresentado anteriormente na Figura 6. Apesar de estarem separados, os módulos formam um espaço de endereçamento único, igualmente compartilhado para toda a arquitetura. Com essa organização, foram necessários componentes responsáveis pelo acesso aos módulos da memória, chamados de Unidades de Acesso a Memória (MAU – *Memory Access Unit*) (14).

[074] Para cada módulo de memória há uma MAU associada que se encarrega da injeção dos pacotes para execução na invenção, pela comunicação ou sincronização dos resultados entre pacotes, e pela execução de instruções que lêem ou escrevem dados na memória. Tais tarefas são executadas por uma MAU através de instruções destinadas a ela contidas nas aplicações em execução no sistema.

[075] As MAUs são núcleos muito simples, em comparação a processadores em NoCs diretas tradicionais. A arquitetura da MAU é formada por um gerenciador de memória (56), um gerenciador de pacotes (57), uma unidade de controle (58) e uma fila de requisição (59), como ilustrado na Figura 17. O gerenciador de memória permite localizar dados nos endereços de memória especificados (para leitura ou escrita) ou pacotes através do endereço de memória onde estão armazenados. Enquanto o gerenciador de pacotes é responsável por executar as instruções destinadas à MAU, as quais podem solicitar a injeção de um novo pacote, inserção de um valor em um pacote ainda armazenado na memória ou fazer leitura/escrita de valores na memória. O gerenciador de pacotes também é responsável pela associação do número

do pacote ao endereço de memória onde está armazenado. Já a unidade de controle estabelece o sincronismo das tarefas entre os dois gerenciadores dentro da MAU, bem como o controle da entrada e saída dos pacotes na MAU. As solicitações para envio de pacote por uma MAU são armazenadas na fila de requisições (59) e são atendidas na ordem que chegam.

[076] Instruções executadas em uma MAU são chamadas de instruções de controle, as quais só podem ser executadas pela MAU a qual foi destinada. Tais instruções fazem parte dos pacotes que descrevem as aplicações, como as demais instruções executadas nas RPUs. Contudo, quando estas instruções estão prontas para serem executadas elas são removidas do pacote original e incluídas em um novo pacote e encaminhadas para as MAUs que devem executá-las. Esse novo pacote é chamado de pacote de controle, carregando apenas a instrução para a MAU indicado na própria instrução e seus operandos, um cabeçalho com apenas uma palavra e o terminador do pacote, como representado na Figura 18. Os pacotes de controle são identificados no campo “Tipo” (28) do cabeçalho com o valor “01_b” e encaminhados através de um canal virtual exclusivo para esse tipo de pacote até a MAU que executará a instrução.

REIVINDICAÇÕES

1. SISTEMA INTEGRADO DE PROCESSAMENTO BASEADO EM REDES EM CHIP QUE NÃO FAZ USO DE PROCESSADORES DO TIPO VON NEUMANN caracterizado por uma rede de RPUs (*Routing and Processing Unit*) com capacidade de rotear e executar instruções, contidas em pacotes de formato específico, ligados entre si por canais *full duplex* formando uma topologia em grelha 2D e também ligados a simples núcleos, localizados nos cantos da rede, com capacidade de acessar a memória e injetar pacotes no sistema, chamados de MAU (*Memory Access Unit*), e por módulos de memória, fisicamente distribuídos mas que formam um único espaço de endereçamento compartilhado, ligados aos MAUs.

2. SISTEMA, de acordo com a reivindicação 1, caracterizado por cada unidade de acesso a memória (MAU – *Memory Access Cores*) compreender um gerenciador de memória, um gerenciador de pacotes, uma unidade de controle e uma fila de requisição.

3. SISTEMA, de acordo com a reivindicação 1, caracterizado por cada unidade de processamento e roteamento (RPU – *Routing and Processing Unit*) compreender uma unidade de lógica e aritmética (ALU – *Arithmetic Logic Unit*) tradicional, uma unidade de sincronização (SU – *synchronization unit*), um par de *buffers* do tipo FIFO (*First In First Out*) em cada porta de entrada, um árbitro em cada porta de saída e um *crossbar* parcialmente conectado.

4. SISTEMA, de acordo com a reivindicação 3, caracterizado por compreender RPUs localizadas nas bordas da rede que possuem três portas de entrada e três portas de saída para comunicação com as RPUs ou MAUs adjacentes.

5. SISTEMA, de acordo com a reivindicação 3, caracterizado por compreender RPUs localizadas no interior da rede, isto é, não estão nas

bordas, possuem quatro portas de entrada e quatro portas de saída para comunicação com outras RPUs.

6. SISTEMA, de acordo com a reivindicação 3, caracterizado por compreender unidade sincronização capaz de criar pacotes de controle a partir de instruções de controle da invenção, endereçando tais pacotes para a respectiva MAU que deve executar a instrução.

7. SISTEMA, de acordo com a reivindicação 3, caracterizado pelas portas de entrada de cada RPU com um par de *buffer* do tipo FIFO, onde cada um representa um canal virtual, destinados cada um a um tipo de pacote.

8. SISTEMA, de acordo com a reivindicação 3, caracterizado por compreender um árbitro que provê a lógica da resolução de disputas pelo canal de saída da RPU quando pelo menos dois pacotes são roteados para a mesma saída; identificação do tipo de instrução para solicitar a computação da mesma pela ALU ou pela SU; armazenamento temporário dos resultados das instruções executadas pela ALU; inserção dos resultados da ALU no pacote corrente nas palavras indicadas na própria instrução executada; detectar a indisponibilidade do canal de saída para transmissão de um pacote, habilitando o modo de execução local até que o canal de transmissão torne-se disponível; transmissão do pacote para a RPU seguinte pela porta de saída.

9. FORMATO DO PACOTE DE REDE QUE TRANSPORTA A CARGA ÚTIL PARA TODO PROCESSAMENTO NA INVENÇÃO, de acordo com a reivindicação 1, caracterizado por compreender quatro tipos de palavras (cabeçalho, instrução, operando, fim do pacote) indicadas pelos bits de controle, tendo cada tipo de palavra campos específicos para representação das informações correspondentes ao tipo de palavra, de modo que possam ser descritas aplicações/programas executáveis na invenção; para determinar o tipo de palavra apenas 1 bit do 4 bits de controle pode estar ativo (nível lógico igual a 1) por vez; sendo o padrão “1000_b” usado para indicar palavras de cabeçalho; o padrão “0100_b” usado para indicar palavras de fim de pacote; o padrão “0010_b” usado para indicar palavras de instrução; o padrão “0001_b”

usado para indicar palavras de operando; cada palavra do pacote possui 36 bit, onde os 4 bits mais significativos representam os bits de controle e os outros 32 os campos de cada palavra.

10. PACOTE, de acordo com a reivindicação 9, caracterizado por compreender cabeçalho, caracterizado por três palavras em pacotes regulares e por uma palavra em pacotes de controle; onde a primeira palavra é formada por 8 bits que indicam o destino atual, 8 bits que indicam a origem da atual, 8 bits que indicam a quantidade de instruções contidas no pacote, 6 bits usados para indicar como calcular novos destinos quando necessário e 2 bits que indicam o tipo do pacote; a segunda palavra do cabeçalho é formada por 32 bits que representa um número exclusivo para identificação do pacote para uma dada aplicação; a terceira palavra do cabeçalho formada por 32 bits usados como apontador ou identificador do número da palavra no pacote que representa a próxima instrução que será executada.

11. PACOTE, de acordo com a reivindicação 9, caracterizado por compreender palavra de instrução com 32 bits, nos quais 8 bits identificam a instrução, 2 bits que indicam a quantidade de operandos necessário para execução da instrução, 11 bits que podem ser usados como posição relativa no pacote onde o resultado da instrução deve ser inserido ou endereço da MAU que executará a instrução (depende do tipo de instrução) e 11 bits que podem ser usados como segunda posição no pacote onde o resultado da instrução executada deve ser inserido ou a quantidade de operandos necessários a instrução (depende do tipo de instrução).

12. PACOTE, de acordo com a reivindicação 9, caracterizado pela palavra de operando, compreende 32 bits usados como dado pela instrução corrente.

13. PACOTE, de acordo com a reivindicação 9, caracterizado pela palavra de fim de pacote, compreende 32 bits todos em nível lógico zero.

14. PACOTE, de acordo com a reivindicação 9, caracterizado por pacote regular, compreende aqueles cujo “tipo” do cabeçalho possuir o valor “00_b” e transportarem instruções e operandos que podem ser executados em qualquer RPU durante o roteamento do pacote até o destino; por poderem ser novamente roteados (o endereço de destino do pacote pode ser alterado) quando o pacote atinge o destino atual e o número de instruções contidas no pacote é maior que zero.

15. PACOTE, de acordo com a reivindicação 9, caracterizado por pacote de controle, compreende aqueles cujo “tipo” do cabeçalho ter o valor “01_b”; por possuir apenas a primeira palavra de cabeçalho; por transportar instruções e operandos para execução em uma MAU; por serem transmitidos por canais virtuais exclusivos livres de *deadlock*.

16. ALGORITMO DE ROTEAMENTO, de acordo com a reivindicação 3, caracterizado por conduzir os pacotes até seus destinos atuais primeiramente no eixo horizontal em seguida no eixo vertical da rede, com adaptação de um novo destino quando o pacote atinge o destino atual e sua carga útil ainda possui pelo menos uma instrução a ser executada; tal algoritmo cria caminhos em forma de uma espiral partindo de um canto para outro canto da rede, de forma cíclica pelos quatro cantos da rede.

DESENHOS

Figura 1

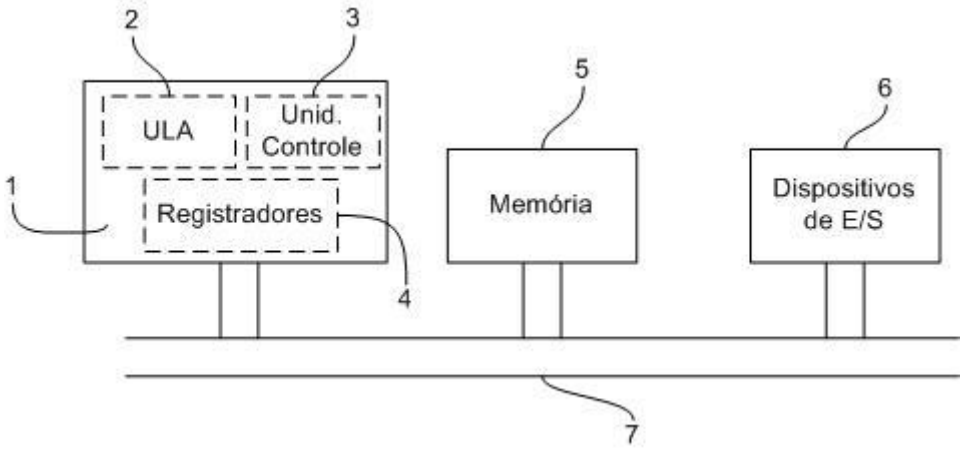


Figura 2

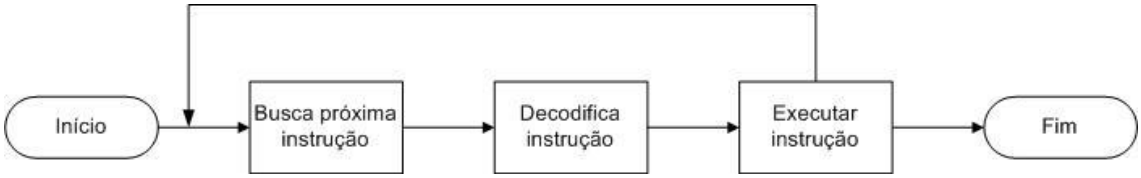

Início
Busca próxima instrução
Decodifica instrução
Executar instrução
Fim

Figura 3

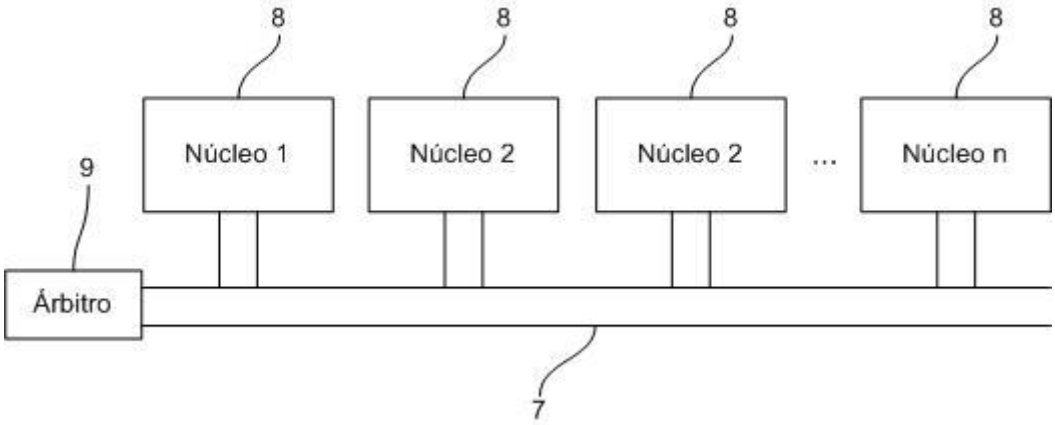


Figura 4

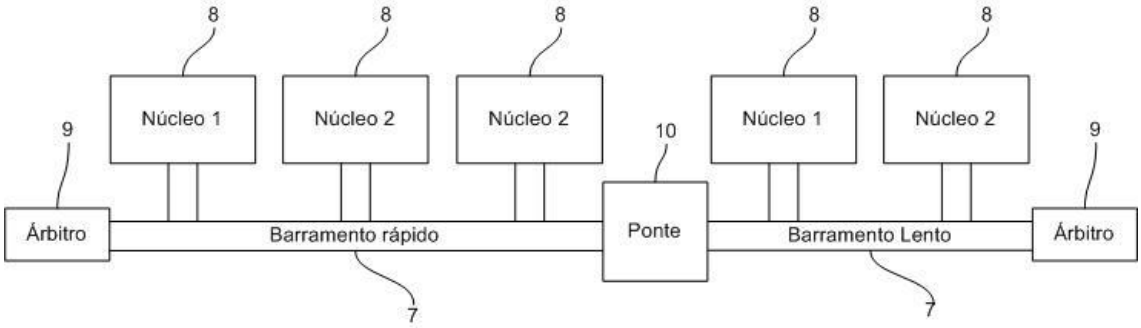

8
8
8
8
8
Núcleo 1
Núcleo 2
Núcleo 2
Núcleo 1
Núcleo 2
9
10
9
Árbitro
Barramento rápido
Ponte
Barramento Lento
Árbitro
7
7

Figura 5

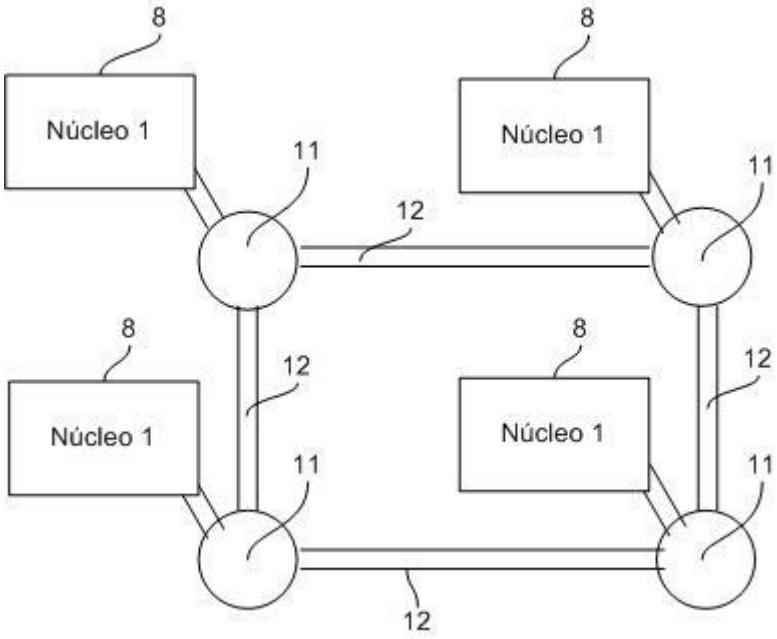


Figura 6

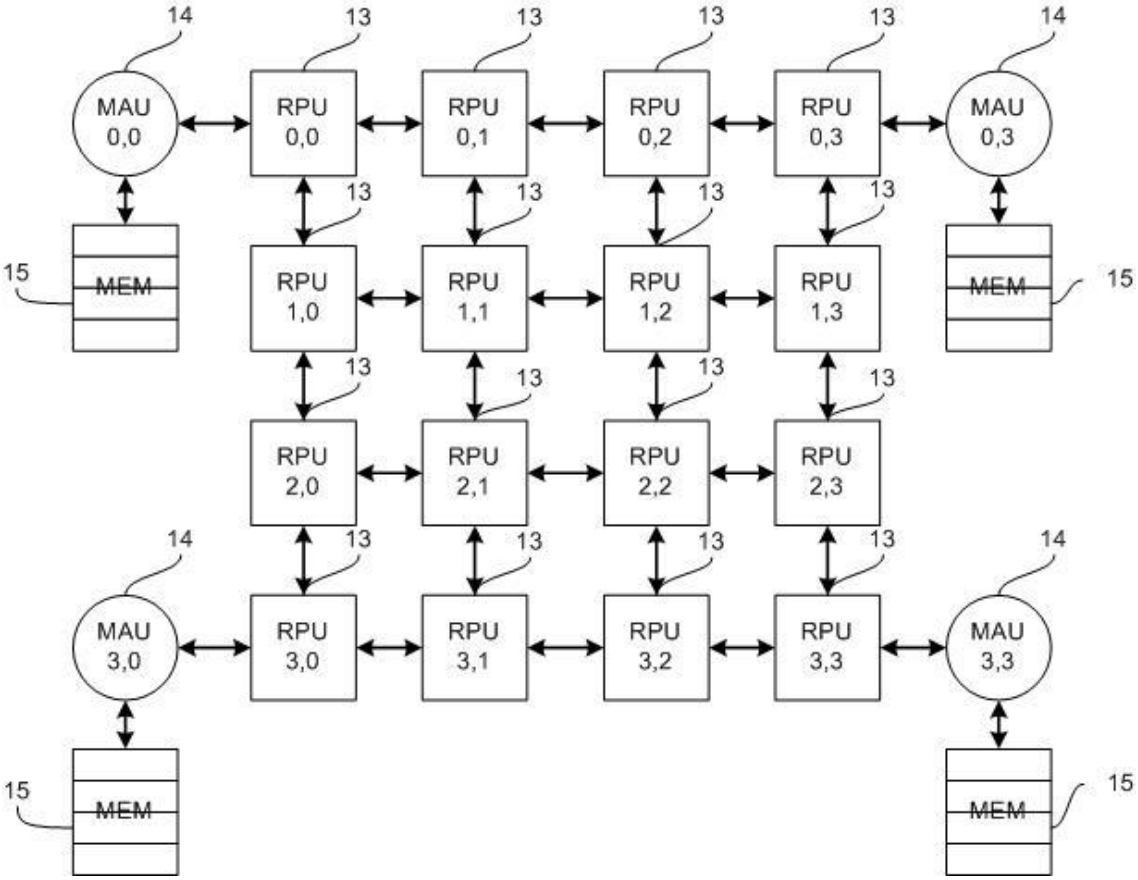

14
13
MAU 0,0
RPU 0,0
RPU 0,1
RPU 0,2
RPU 0,3
MAU 0,3
15
MEM
RPU 1,0
RPU 1,1
RPU 1,2
RPU 1,3
RPU 2,0
RPU 2,1
RPU 2,2
RPU 2,3
MAU 3,0
RPU 3,0
RPU 3,1
RPU 3,2
RPU 3,3
MAU 3,3

Figura 7

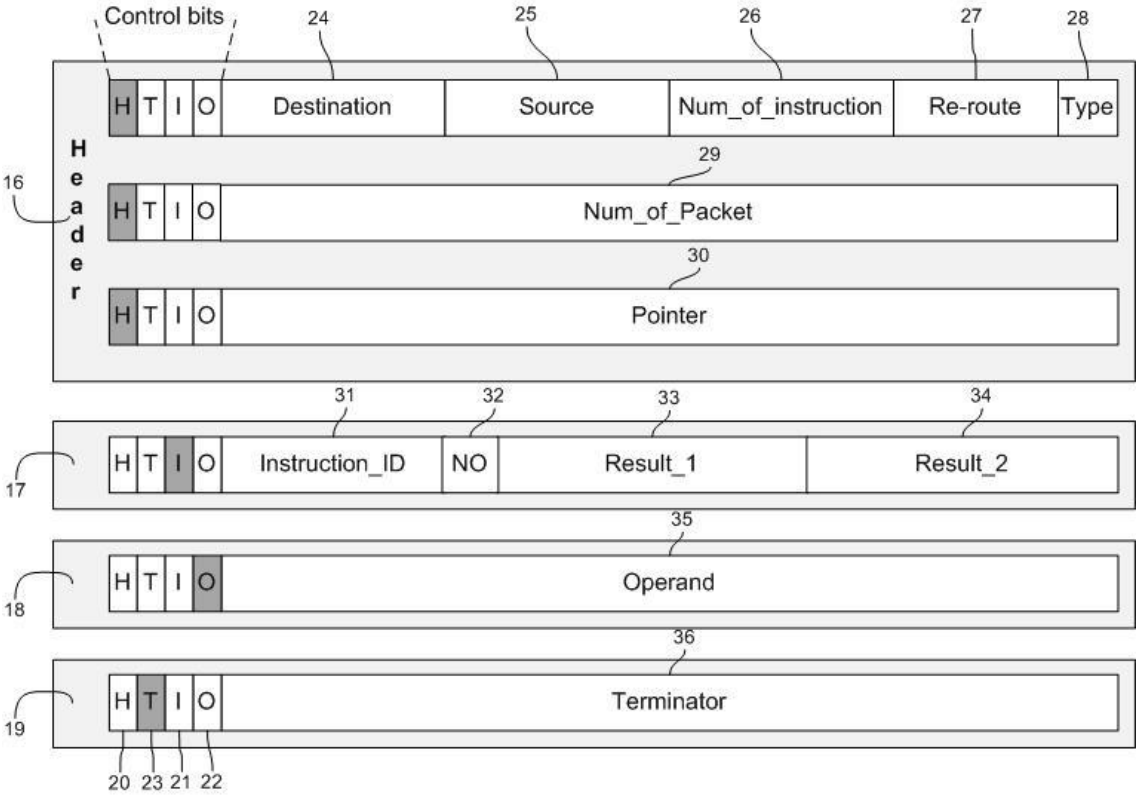

Control bits
24
25
26
27
28
H T I O
Destination
Source
Num_of_instruction
Re-route
Type
Header
16
29
Num_of_Packet
30
Pointer
31
32
33
34
17
Instruction_ID
NO
Result_1
Result_2
35
18
Operand
36
19
Terminator
20 23 21 22

Figura 8

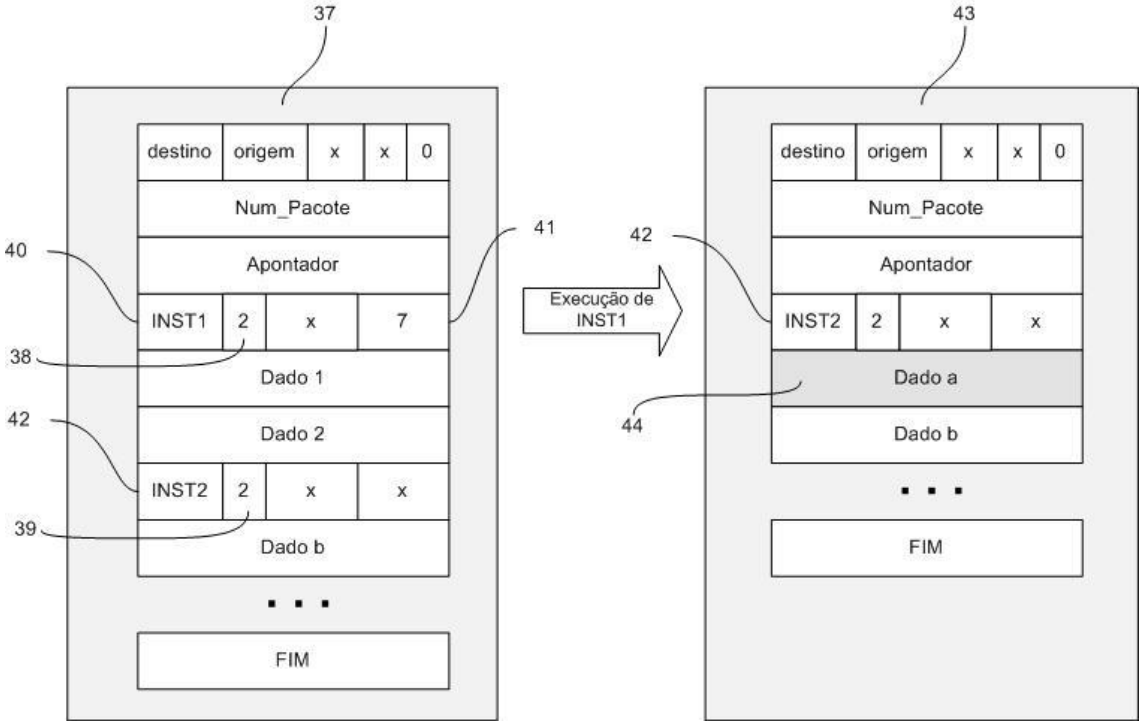

37
43
destino
origem
x
x
0
Num_Pacote
Apontador
INST1
2
x
7
Dado 1
Dado 2
INST2
2
x
x
Dado b
FIM
40
41
42
38
42
39
Execução de INST1
destino
origem
x
x
0
Num_Pacote
Apontador
INST2
2
x
x
Dado a
Dado b
FIM
44

Figura 9

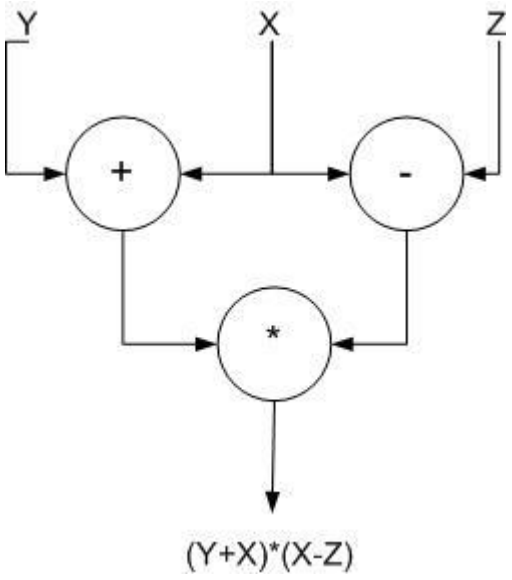


Figura 10

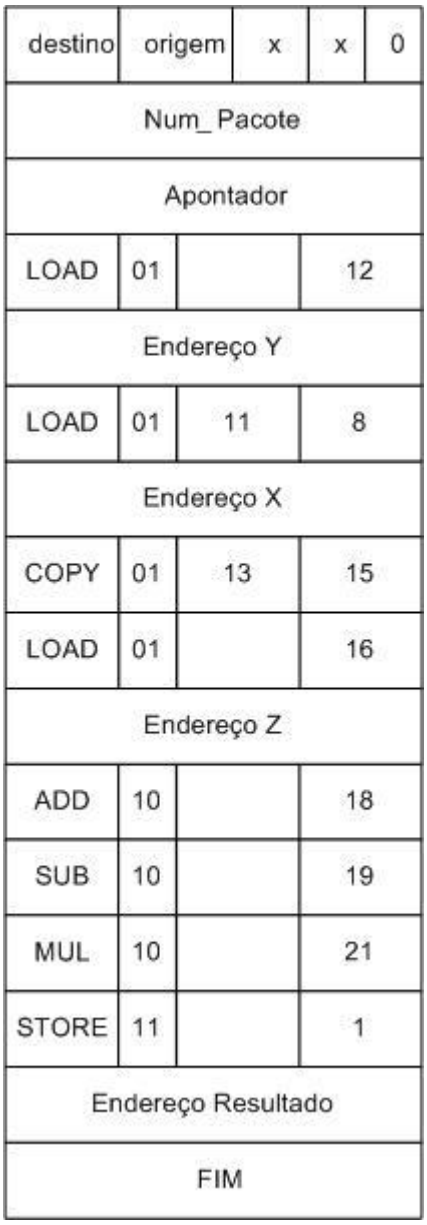

destino
origem
x
x
0
Num_ Pacote
Apontador
LOAD
01
12
Endereço Y
LOAD
01
11
8
Endereço X
COPY
01
13
15
LOAD
01
16
Endereço Z
ADD
10
18
SUB
10
19
MUL
10
21
STORE
11
1
Endereço Resultado
FIM

Figura 11

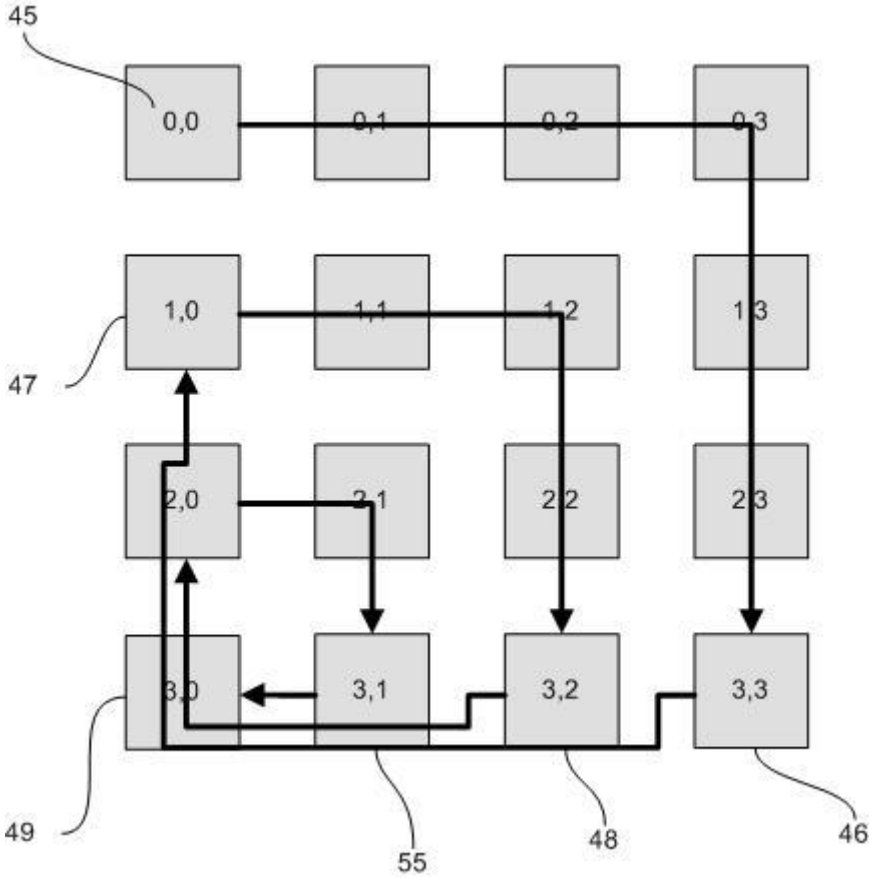


Figura 12

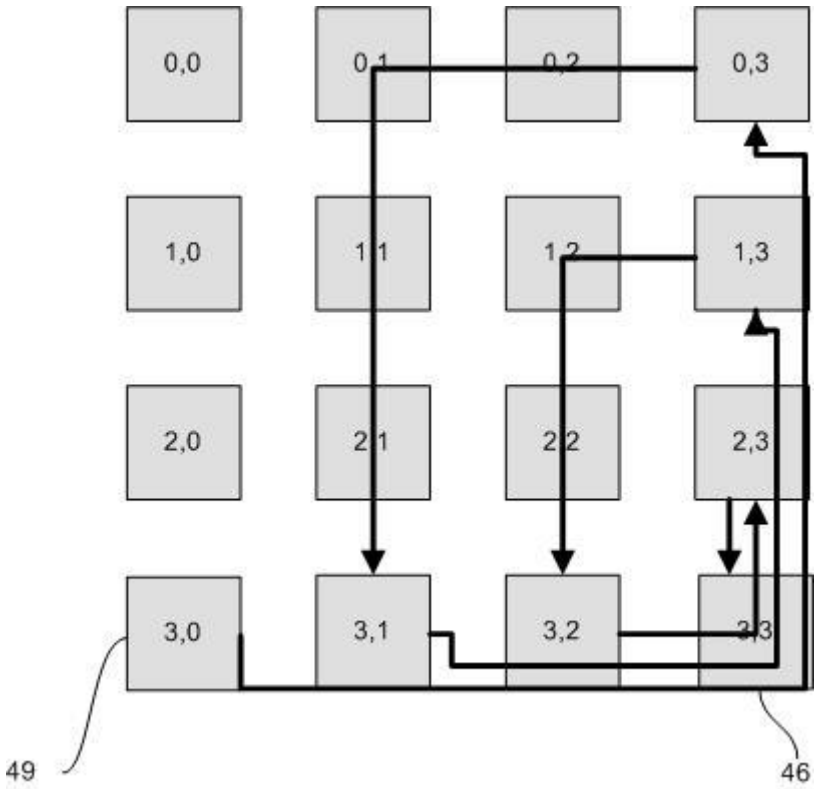


Figura 13

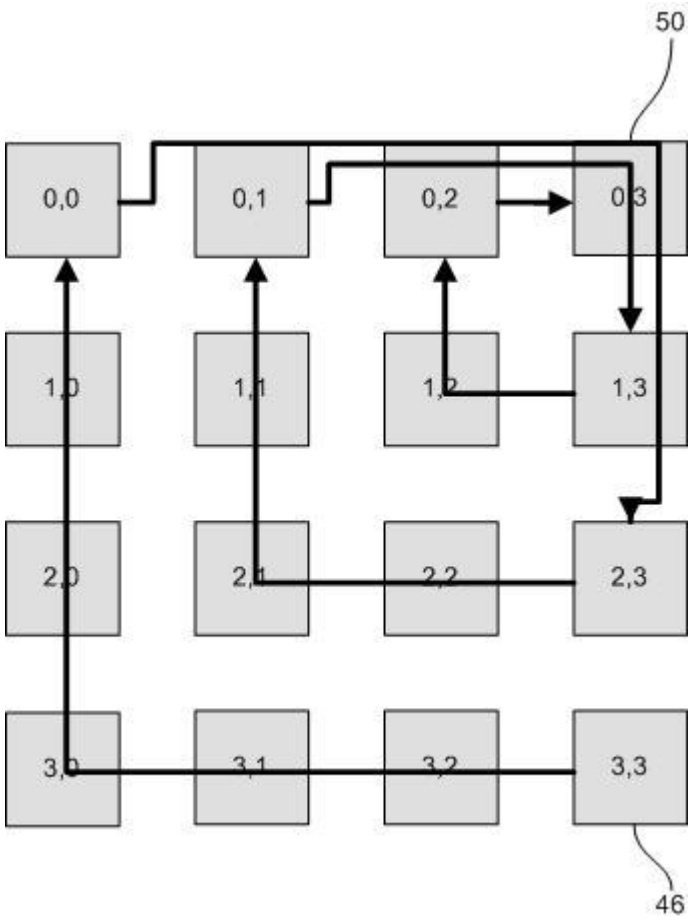


Figura 14

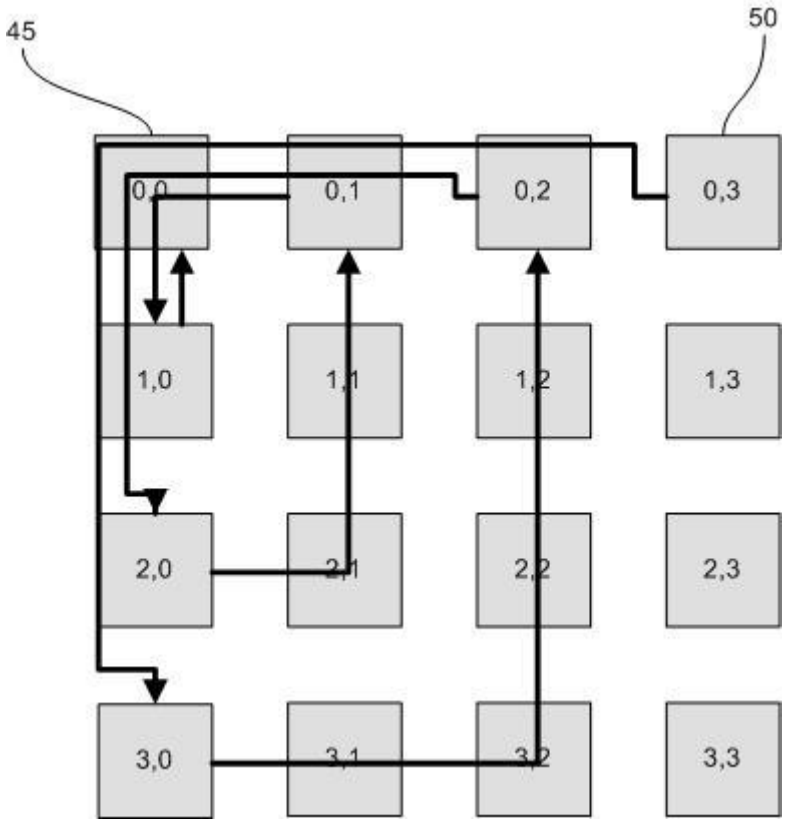


Figura 15

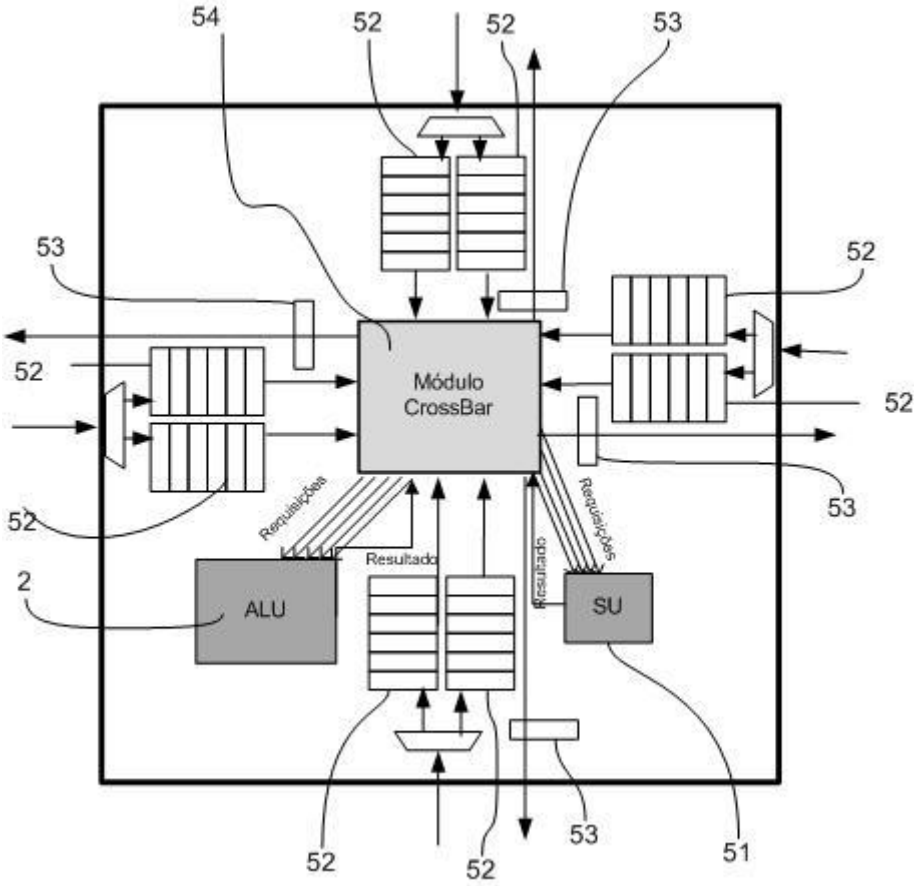

54
52
52
53
53
52
52
Módulo
CrossBar
52
52
53
Requisições
Resultado
Resultado
Requisições
2
ALU
SU
52
52
53
51

Figura 16

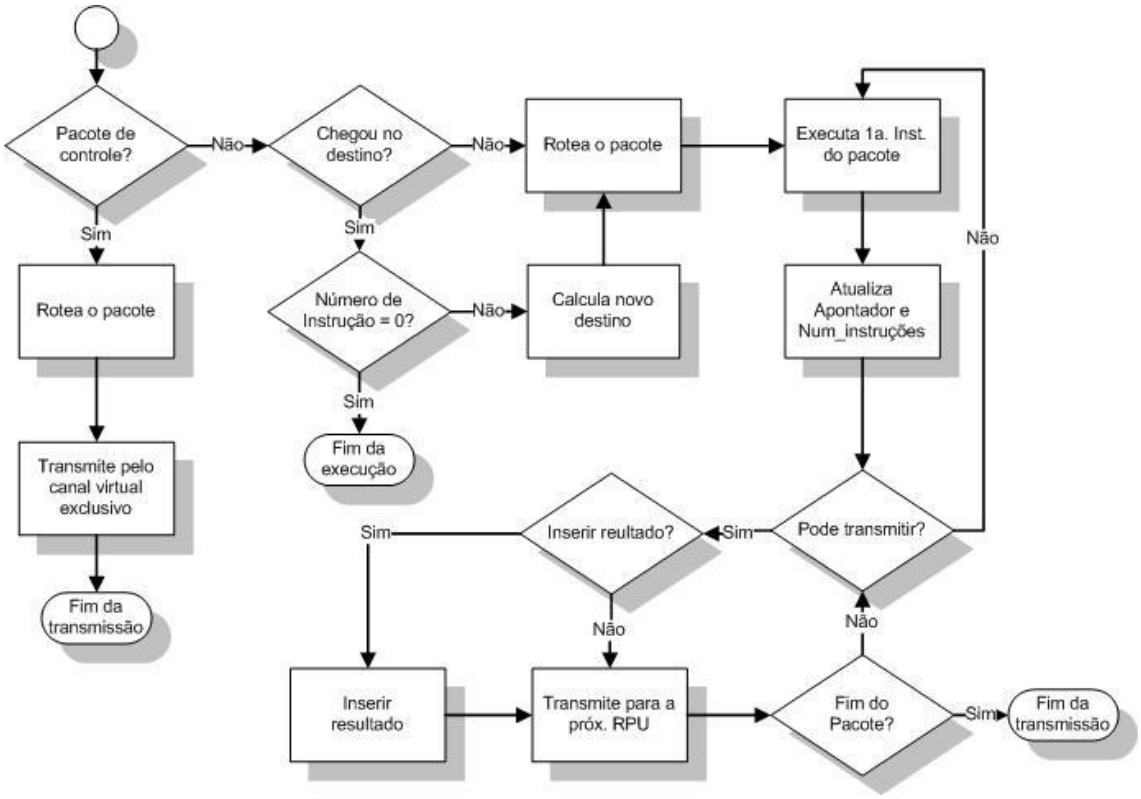

Pacote de controle?
Não
Chegou no destino?
Não
Rotea o pacote
Executa 1a. Inst. do pacote
Não
Sim
Sim
Rotea o pacote
Número de Instrução = 0?
Não
Calcula novo destino
Atualiza Apontador e Num_instruções
Sim
Fim da execução
Transmite pelo canal virtual exclusivo
Sim
Inserir reultado?
Sim
Pode transmitir?
Fim da transmissão
Não
Não
Inserir resultado
Transmite para a próx. RPU
Fim do Pacote?
Sim
Fim da transmissão

Figura 17

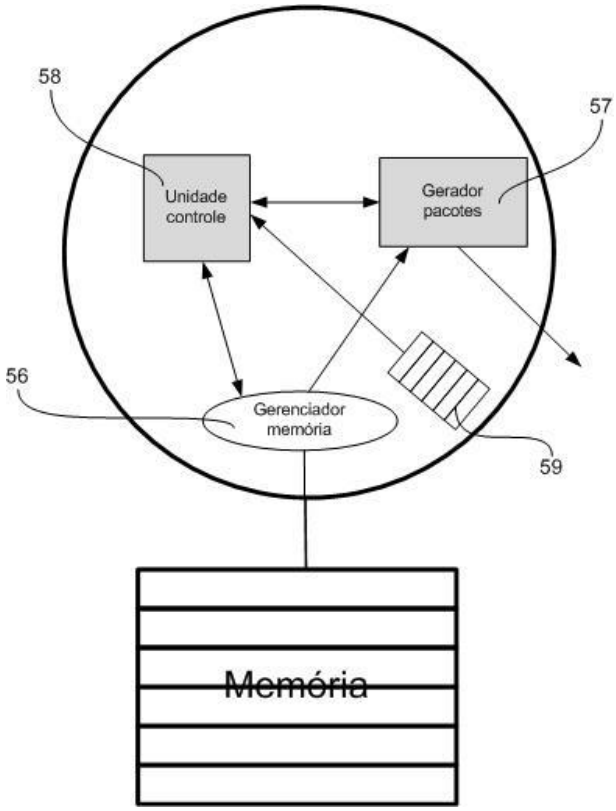
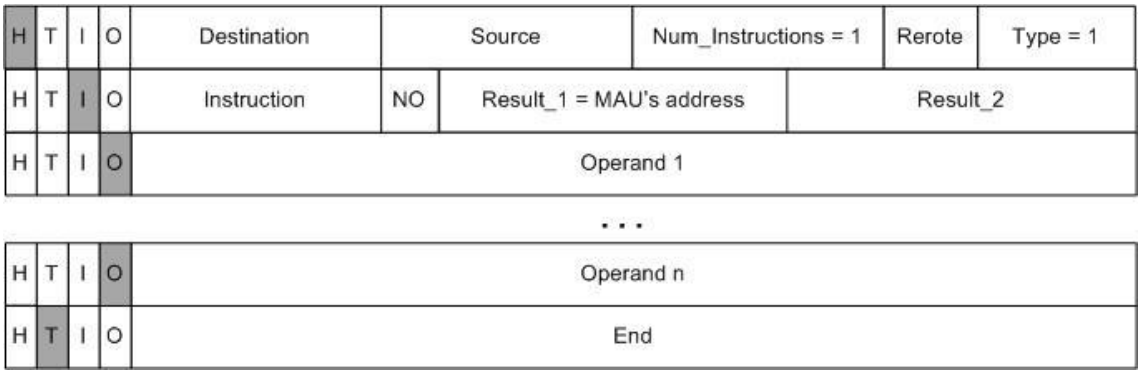

58
57
Unidade controle
Gerador pacotes
56
Gerenciador memória
59
Memória

Figura 18


H T I O Destination Source Num_Instructions = 1 Rerote Type = 1
H T I O Instruction NO Result_1 = MAU's address Result_2
H T I O Operand 1
. . .
H T I O Operand n
H T I O End