Especificação da Arquitetura

Processador BLA

Efficiency Eletronics S.A.

Build 2.0

History Review

Date	Description	Author(s)
05/12/2015	Definição do propósito do documento e conteúdo das seções (introdução)	Joacy Mesquita
05/12/2015	Definição das características operacionais	Thiago Sampaio Lima
06/12/2015	Definição do conjunto de instruções	Matheus Moura Batista
08/12/2015	Definição dos requisitos não funcionais e flags do processador	Thiago Sampaio Lima
09/12/2015	Definição do formato das instruções	Matheus Moura Batista
09/12/2015	Definição dos acrônimos e abreviações	Todos
10/12/2015	Definição dos componentes do processador e criação do datapath	Joacy Mesquita
14/12/2015	Adição de requisitos não funcionais e descrição do limite de endereços para saltos	Thiago Sampaio Lima

SUMÁRIO

1. Introdução

1.1. Propósito

O propósito deste documento é mostrar uma visão geral sobre o processador BLA, incluindo suas principais características e a descrição dos seus componentes. Este documento também apresenta a arquitetura do conjunto de instruções do processador, incluindo o código da operação e uma breve descrição da mesma.

1.2. Organização do documento

As seções deste documento são organizadas conforme mostrado a seguir.

- Seção 2: Apresenta uma visão geral das principais características do processador, incluindo os principais componentes do mesmo e os requisitos não funcionais.
- Seção 3: Apresenta a arquitetura do conjunto de instruções do processador e suas características.

1.3. Acrônimos e Abreviações

Acrônimo	Descrição
RISC	Computador com Conjunto Reduzido de Instruções
MIPS	Microprocessador sem Estágios Interligados de Pipieline
GPR	Registradores de Propósito Geral
CPU	Unidade de Processamento Central
ISA	Arquitetura do Conjunto de Instruções
ULA	Unidade Lógica e Aritmética
PC	Contador de Programa
IR	Registrador de Instruções
RA, RB	Registradores Fonte
RC	Registrador Destino
NOP	Instrução Sem Operação

2. Visão Geral

2.1. Principais características

O processador em questão possui um conjunto de 16 registradores de propósito geral (organizados em um banco de registradores) e apresenta uma arquitetura com 32 bits de largura. A ISA deste processador foi desenvolvida utilizando-se algumas características da arquitetura µRISC e outras da arquitetura MIPS. As principais características operacionais e seus respectivos significados são apresentados a seguir.

2.1.1. Arquitetura com 32 bits de largura

Neste modelo de arquitetura, todos registradores do processador possuem uma largura de 32 bits (4 bytes). Com isso é possível utilizar valores inteiros sem sinal que variam de 0 a 4294967296 ou com sinal que variam de -2147483648 a +2147483647.

2.1.2. Dezesseis registradores de propósito geral

O processador fornece 16 registradores de propósito geral (R_0 a R_{15}) de 32 bits cada um (4 bytes). O registrador R_0 possui a característica de manter-se sempre em nível lógico baixo, ou seja, o "0"binário. Esta característica tem por finalidade aumentar o desempenho de algumas operações comuns aos processadores como, por exemplo, a transferência de dados entre registradores. O valor do registrador é fixo e não pode ser alterado por nenhum programa rodando neste processador. O registrador R_7 é utilizado como padrão para armazenar o endereço de retorno quando se utiliza a instrução jump and link (jal).

2.1.3. Palavras com tamanho de 4 bytes

O tamanho das palavras de dados são de 4 bytes cada. Esta característica está relacionada principalmente com a largura da arquitetura do processador, permitindo armazenar uma palavra por registrador.

2.1.4. Endereçamento a nível de palavra

Cada endereço utilizado pelo processador para acessar a memória é feito a nível de palavra, ou seja, cada endereço aponta para um conjunto de 4 bytes. Ao realizar um incremento no registrador PC, o mesmo apontará para uma nova palavra na memória (pulará 4 bytes ao no lugar de apenas 1).

2.1.5. Operações aritméticas inteiras

Para a realização de cálculos aritméticos, os operandos fornecidos através das instruções terão que estar apresentados na forma inteira, ou seja, o processador em questão não fornece suporte ao cálculo envolvendo ponto flutuante.

2.1.6. Dois modos de endereçamento

Para que as instruções possam ser executadas é necessário obter os operandos da mesma e, para isso, é utilizada algum dos dois modos de endereçamento possíveis: (1) via registrador, (2) imediato. O modo de endereçamento via registrador é o que utiliza apenas

os registradores do processador para realizar as operações, visto que os operandos estarão previamente armazenados neles. O modo imediato deste processador diz respeito a somente as operações de carregamento de constantes, visto que as demais operações precisam que estas constantes já estejam em registradores. É válido notar também que é possível utilizar o modo indexado de forma indireta, realizando incremento ou adição em um endereço base disponível e, com isso, obter um novo endereço para o que se deseja acessar na memória.

2.1.7. Quatro tipos de instrução

As instruções da ISA deste processador são divididas em 4 principais grupos: (1) instruções de acesso à memória, (2) instruções logico-aritméticas, (3) instruções de desvios e saltos e (4) instruções de carregamento de constantes. No primeiro grupo estão as instruções utilizadas para ler (load) da memória e gravar em um registrador e escrever (store) na memória apartir de um valor presente em um registrador. No segundo grupo estão as principais instruções de cálculos aritméticos (add, sub, mult, etc) e lógicas (and, or, etc). No terceiro grupo estão presentes as instruções de salto (j) com e sem condições. Já no quarto grupo estão presentes as instruções responsáveis por gravar as constantes em um registrador para posterior uso (lcl, lch, etc).

2.1.8. Instruções com três operandos

Instruções lógico-aritméticas possuem três operandos, sendo um deles o registrador de destino e os outros dois os registradores fonte.

2.1.9. Flags para determinados estados do processador

O processador possui a capacidade de indicar diversas flags para determinar algum estado decorrente de uma execução. A tabela a seguir mostra as flags suportadas pelo processador e uma breve descrição das situações em que são ativadas.

Código	Flag	Descrição
0001	.overflow	Operação executada extrapolou o número máximo de bits para representar o resultado e aconteceu um "vai-um". Overflow é alerado em operações com sinal.
0010	.zero	Operação executada apresentou como resultado o valor zero
0011	.neg	Operação executada apresentou como resultado um valor negativo
0100	.negzero	Operação executada apresentou como resultado um valor igual ou menor que zero
0101	.true	Operação executada apresentou como resultado um valor diferente de zero
0110	.carry	Operação realizou um "vai-um". Carry é alterado em operações sem sinal.

2.1.10. *Dados organizados na forma Big Endian*

Ao serem armazenados na memória, os dados são organizados na forma Big Endian, ou seja, os bytes mais significativos ficam nas menores posições de memória.

2.2. Requisitos não funcionais

O projeto do processador em questão engloba algumas características que não dizem respeito diretamente a uma funcionalidade do mesmo e sim a questões de escalabilidade, eficiência ou, até mesmo, plataforma de funcionamento. A seguir são apresentadas todas estas características, conhecidas também como requisitos não funcionais.

2.2.1. *Assembler*

Para que seja possível utilizar o processador em diversas atividades e operações é necessário a criação de um montador para esta arquitetura. O montador é o responsável por traduzir códigos-fonte em linguagem assembly para o código de máquina correto que o processador necessita. O montado faz parte do pacote de programas necessários para validar o funcionamento lógico do processador (confiabilidade).

2.2.2. *Simulador*

Devido a complexidade do processo de implementação de um processador, é necessário a criação de um simulador em software para validar a especificação e a arquitetura do conjunto de instruções. Essa validação é necessária para que todos os erros graves no projeto sejam descobertos com antecedência e, com isso, não cheguem a prejudicar o processo de implementação. O simulador também faz parte do pacote de programas para validar o funcionamento do processador (confiabilidade).

2.2.3. *Programas de Teste*

A fim de atestar o funcionamento do montador e do processador nas diversas situações que poderão ocorrer, é necessário a entrega de três programas em assembly para de serem traduzidos e executados durante a etapa de testes. Esses programas serão, cada um, um dos algoritmos especificados no problema: (1) Busca binária, (2) Ordenação Bubble Sort e (3) Sequência Fibonacci. Estes programas de teste também fazem parte do pacote de programas necessários para validar o funcionamento do processador (confiabilidade).

2.2.4. *Suporte a grande quantidade memória*

Devido as características de endereçamento a nível de palavra e arquitetura com 32 bits de largura, o processador em questão dá suporte a grande quantidade de memória de dados, possibilitanto a ampla utilização do sistema para diversas atividades (escalabilidade). A quantidade máxima de memória suportada pelo processador é calculada através da quantidade de endereços que o o mesmo suporta (2^{32}) multiplicado pela quantidade de memória armazenada em cada endereço (4 bytes). Com isso, chegamos ao valor de 16 Gigabytes de memória.

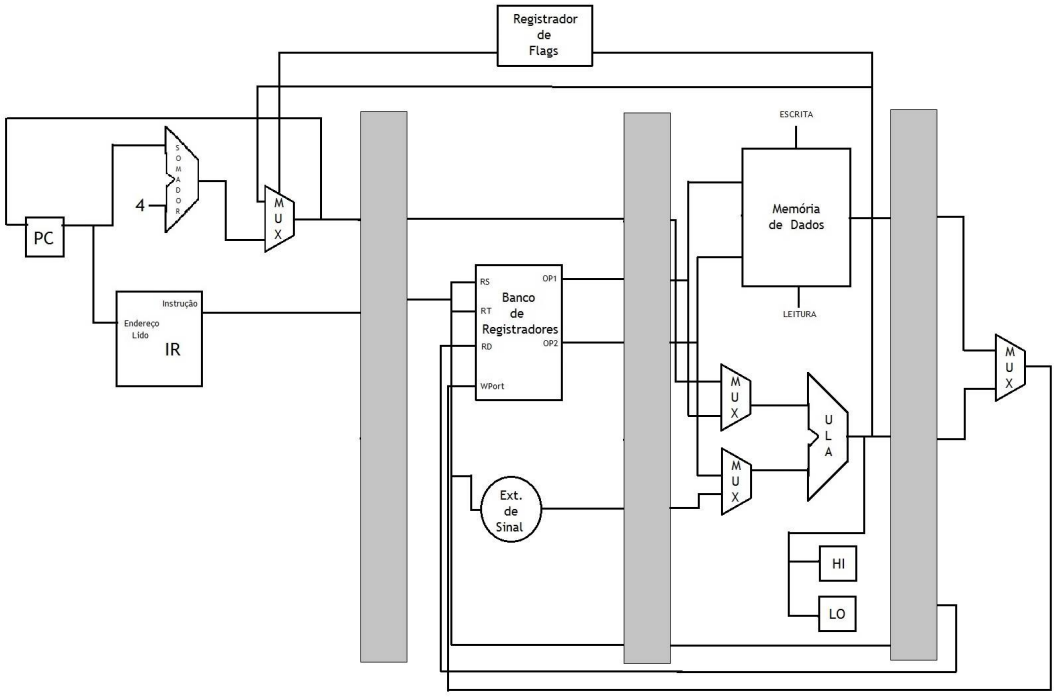

Figura 1: Visão geral dos componentes e datapath do processador

2.2.5. *Suporte à ampliação do ISA*

Como pode ser visto na seção que trata do ISA do processador, cada instrução possue um código. Esse código foi feito em um conjunto de bits com capacidade acima do necessário, ou seja, conforme surgir a necessidade de mais instruções, será consideravelmente fácil adicionar estas funcionalidades ao processador (escalabilidade).

2.2.6. *Plataforma Linux*

Tanto o assembler (montador) quanto o simulador desenvolvidos devem ser capazes de executar corretamente na plataforma Linux (plataforma).

2.2.7. *Velocidade de clock*

O processador possui uma velocidade de clock de 50 Mhz. Esta velocidade foi definida com base na plataforma que será utilizada para implementação física do mesmo, a FPGA Cyclone III (EP3C25F324).

2.3. Componentes internos e datapath

Esta seção tem por finalidade apresentar uma visão geral da arquitetura do processador, abrangendo os compentes principais e o caminho dos dados na execução de uma instrução, além do ciclo de execução de uma instrução.

2.3.1. *Program Counter (PC)*

O *Program Counter*, ou PC é o registrador que armazena o endereço da próxima instrução a ser buscada na memória.

2.3.2. *Instruction Register (IR)*

O *Instruction Register*, ou IR é o registrador que armazena a instrução buscada na memória.

2.3.3. *Somador*

Utilizado para calcular o endereço da próxima instrução. Para tal, o endereço armazenado em PC é adicionado a 4. Levando em consideração a arquitetura de 32 bits do processador e o modo de endereçamento a nível de palavra, o número 4 representa a palavra (4 bytes), fazendo com que o PC seja atualizado com um endereço 4 bytes após o que já possui.

2.3.4. *Banco de Registradores*

O banco de registradores é o conjunto onde os Registradores de Propósito Geral ficam agrupados. Neste caso, o banco possui 64 bytes de capacidade (16 GPR de 4 bytes cada). É neste banco que são armazenadas as variáveis temporárias de cada execução de instrução.

2.3.5. *Unidade de Extensão de Sinal*

] O extensor de sinal é o responsável por extender operandos com menos de 32 bits. Essa extensão é necessária pois todas as operações deste processador necessitam de valores com 32 bits de largura. Essa extensão é feita replicando o bit mais significativo do operando a todos os bits que vêm a sua esquerda, até o bit de número 31.

2.3.6. *Unidade Lógico-Aritmética*

A Unidade Lógico-Aritmética é a responsável por realizar todas as operações aritméticas (soma, subtração, etc) e lógicas (and, or, etc).

2.3.7. *Registradores HI e LO*

Numa operação de multiplicação, a quantidade de bits do produto pode facilmente ultrapassar a largura de bits utilizada nos operandos. Levando este fato em consideração, são utilizados dois registradores específicos (HI e LO) para armazenar o resultado destas operações. O registrador HI fica responsável por armazenar os 32 bits mais significativos do resultado enquanto que o registrador LO armazena os 32 bits menos significativos.

2.3.8. *Registrador de Flags*

Registrador utilizado para armazenar todas as flags geradas durante as operações do processador. Estas flags são utilizadas, principalmente, nas operações de jump condicionais.

2.3.9. *Memória de Dados*

É a memória sistema, necessária para o armazenamento de dados a longo prazo. Acessada através das intruções de acesso à memória, load e store.

2.3.10. Ciclo de Execução da Instrução

O ciclo de execução de uma instrução será dividido em cinco passos.

- 1º: Busca da Instrução (*Fetch Instruction*).
- 2º: Decodificação da Instrução.
- 3º: Busca dos Operandos (*Fetch Operands*).
- 4º: Execução da Instrução.
- 5º: Armazenamento dos Resultados.

3. Instruction Set Architecture

BLA is built based on RISC processor architecture. The ISA has 42 CPU instructions each one with 32-bits long word and organized into the following functional groups:

- Arithmetic and Logical
- Constant load
- Data transfer
- Jump and branch

In BLA load/store architecture, operations are performed through operands held into the register file (GPR Bank) and main memory is accessed only through load and store instructions. Signed and unsigned constants of 16-bits are supported by constant load instructions that load the sign-extend number into the register.

3.1. Arithmetic and logical instructions

The arithmetic and logical instructions have the following format:

31 30	29 18	17 12	11 8	7 4	3 0
0 0	00...0	Code	RC	RA	RB

Code	Mnemonic	Operation	Description
000000	add c, a, b	c = a + b	Add two words.
000001	addu c, a, b	c = a + b	Add two unsigned words.
000010	addinc c, a, b	c = a + b + 1	Add two words and increase 1.
000011	inca c, a	c = a + 1	Increase 1.
000100	sub c, a, b	c = a - b	Substract two words.
000101	subdec c, a, b	c = a – b - 1	Substract two words and decrease 1.
000110	deca c, a	c = a - 1	Decrease 1.
000111	mult a, b	Hi,Lo = a * b	Multiplies two words and keeps the 64-bits result on 'Hi' and 'Lo'.
001000	multu a, b	Hi,Lo = a * b	Multiplies two unsigned words and keeps the 64-bits result on 'Hi' and 'Lo'.
001001	mfh c	c = Hi	Copy the value from Hi to 'c'.
001010	mfl c	c = Lo	Copy the value from Lo to 'c'.

Continued on next page

Continued from previous page

Code	Mnemonic	Operation	Description
001011	div c, a, b	c = a / b	Divides two words
001100	divu c, a, b	c = a / b	Divides two unsigned words
001101	asl c, a	c = a ASL 1	Arithmetic left shift.
001110	asr c, a	c = a ASR 1	Arithmetic right shift.
001111	zeros c	c = 0	Reset the register.
010000	ones c	c = 1	Sets the register as max-value.
010001	passa c, a	c = a	Tranfer the word in register 'a' to 'c'.
010010	passnota c, a	c = !a	Tranfer the logical NOT of the word in register 'a' to 'c'.
010011	and c, a, b	c = a & b	Bit-by-bit logical AND.
010100	andnota c, a, b	c = !a & b	Logical AND between the logical NOT of 'a' and 'b'.
010101	nand c, a, b	c = !(a & b)	Bit-by-bit logical NAND.
010110	or c, a, b	c = a | b	Bit-by-bit logical OR.
010111	ornotb c, a, b	c = a | !b	Logical OR between 'a' and the logical NOT of 'b'.
011000	nor c, a, b	c = !(a | b)	Bit-by-bit logical NOR.
011001	xor c, a, b	c = a $\oplus$ b	Bit-by-bit logical XOR.
011010	xornota c, a, b	c = !a $\oplus$ b	Logical XOR between the logical NOT of 'a' and 'b'.
011011	xnor c, a, b	c = !(a $\oplus$ b)	Bit-by-bit logical XNOR.
011100	lsl c, a	c = a LSL 1	Logical left shift.
011101	lsr c, a	c = a LSR 1	Logical right shift.
011110	slt c, a, b	if (a < b): c = 1; else: c = 0	Compares two words.
011111	sltu c, a, b	if (a < b): c = 1; else: c = 0	Compares two unsigned words.

The arithmetic and logical instructions need only a 5-bits code, because have only 32 instructions. But on the BLA architecture, the arithmetic and logical instructions code have 6-bits length, because this way, 32 more instructions are supported, making the architecture be much more scalable. The logical instructions reset the *flags overflow* and *carry*.

3.2. Constant load instructions

The constant load instructions have the following format:

31 30	29 22	21 20	19 16	15 0
0 1	00...0	Code	RC	Constant

Code	Mnemonic	Operation	Description
00	loadlit c, const	c = const	Load constant.
01	lcl c, const	c = const | (c & 0xff00)	Load constant to the two less significative bytes of 'c'
10	lch c, const	c = (const « 8) | (c & 0xff00)	Load constant to the two more significative bytes of 'c'

Before the constant be load shoud extend its signal bits to the more significative bits to complete a 32-bits word. For example: if the constant is 1001011101100101 it will become 11111111111111111001011101100101. The constants in the BLA architecture have 2 bytes lenth.

3.3. Data transfer instructions

The data transfer instructions have the following format:

31 30	29 13	12	11 8	7 4	3 0
1 0	00...0	Code	RC	RA	RB

Code	Mnemonic	Operation	Description
0	load c, a	c = Mem[a]	Read a word from the memory.
1	store c, a	Mem[c] = a	Write a word on the memory.

3.4. Jump and branch instructions

The jump and branch instructions have the following format:

31 30	29 24	23 20	19 16	15 12	11 0
1 1	00...0	Code	Condition	RA	Destination

When the 'condition', 'RA' and 'destination' areas are not used, they must be filled with logical zero.

Code	Mnemonic	Operation	Description
0000	j Destination	unconditional jump	Unconditional jump to the target address.
0001	jt.Condition Destination	if (condition) jump	Jump if condition is true.
0010	jf.Condition Destination	if (!condition) jump	Jump if condition is false.
0011	jal a	jump and link	Unconditional jump to the address specified in 'a' and save the address of the next instruction in '$r7'.
0100	jr a	jump register	unconditional jump to the address specified in 'a'.

Nas operações de salto j, jt e jf o endereço do destino é limitado a 12 bits, fazendo com que só seja possível realizar saltos até um certo limite de endereços. Para solucionar este problema basta utilizar a instrução jr (jump register), onde o endereço para o qual se deseja saltar estará pré armazenado em um registrador de 32 bits e, com isso, será possível saltar para qualquer endereço de memória que se deseje.

3.5. No Operation Instruction

The *No Operation* instruction (NOP) is used to control the instruction flow or to insert delays (stalls) into the datapath, such as when computing the result of a jump/branch instruction.

3.6. End of Opperations

The HALT instruction (system stop) must be implemented as a `L: j L` (a unconditional branch to the current address).