Trabalho 3 - VLIW e Vetorial

SERGIO SIVONEI DE SANT'ANA FILHO GRR20242337
EDUARDO KALUF GRR20241770

May 28, 2025

1 Apresentação

Ao decorrer da disciplina aprendemos inúmeras maneiras de realizar a implementação de um processador, o modo como operam, seus prós e contras e seus contextos históricos. Sendo assim, neste trabalho iremos nos aprofundar em duas dessas maneiras sendo elas o VLIW e o processador Vetorial, o objetivo é a partir de uma ISA já predefinida (SAGUI) criar o projeto de ambos os processadores utilizando **Logisim**, criar duas instruções adicionais para cada ISA e então realizar a execução de um programa assembly em cada um deles.

Para cada um dos processadores iremos utilizar uma ISA diferente que deriva do SAGUI, no caso do VLIW será o Sagui de rabo longo, já para o vetorial o Sagui em bando, além disso, o programa assembly teste consiste em fazer a soma de dois vetores com 12 posições e guardar os resultados na memória, com todos os vetores alinhados. Devido a maneira de como os processadores operam, cada um deles terá um programa Assembly diferente, porém que deverá apresentar o mesmo resultado.

2 VLIW

2.1 Arquitetura

VLIW significa "Very long instruction Word" que se traduz para "palavra de instrução muito grande" e a ideia do processador gira exatamente em torno disso, basicamente, iremos ter uma palavra grande que carrega mais de uma instrução dentro dela, no nosso caso, 4 instruções, dessa forma o processador irá executa-las em paralelo aumentando o IPC total. No VLIW a responsabilidade de evitar dependencias e escritas simultaneas ficam completamente na mão do compilador, o qual utiliza diversas técnicas como loop unroling e predicação a fim de diminuir a quantidade de NOPS necessários, fazendo com que o código seja executado de maneira mais otimizada.

2.2 ISA

A ISA que iremos utilizar para o nosso VLIW é a Sagui de Rabo Longo (SRB) com algumas modificações para operar com predicação. A SRB consiste em uma ISA parecida com a REDUX-V, sendo tão compacta e amigável quanto para quem está iniciando no mundo da arquitetura de computadores. Ela possui 8 bits com 4 registradores de propósito geral e será endereçada de palavra a palavra ou seja de 4 em 4 bytes, além de definir 4 "lanes" diferentes, esses "lanes" são os tipos de operação (Branch, aritmética, lógica e etc...) que cada posição da palavra do VLIW poderá realizar. O opcode possui 4 bits, fazendo com que possamos operar 16 instruçoes diferentes, devido as modificações para predicação as instruções serão apresentadas mais a frente já em suas lanes específicas.

Por padrão, os formatos das instruções e as lanes são definidas a seguir:

Formato Padrão 1: Tipo I

Tipo I								
Bits	7	6	5	4	3	2	1	0
	Opcode				**Imm.**			

Formato Padrão 2: Tipo R

Tipo R								
Bits	7	6	5	4	3	2	1	0
	Opcode				**Ra**		**Rb**	

Lanes:

Dados	Controle	ULA	ULA
1º	2º	3º	4º

2.3 Motivações e Instruções

Para este processador gostariamos de dar a maior quantidade de operações lógicas possíveis ao programador e uma facilidade maior para mexer com loops, adicionando outra instrução branch de controle. Além disso, para que a utilização de predicados seja viável, adaptamos a ISA de modo que ela possua um conjunto de instruções específicas para cada uma das lanes do VLIW e instruções para settar o predicado na parte de controle

2.3.1 Instrução 1

2.3.2 Instrução 2

2.3.3 Predicação

Lane 1: Dados

Opcode	Tipo	Mnemonic	Nome	Operação
Dados				
0100	R	ld	Load	R[ra] = M[R[rb]]
0101	R	st	Store	M[R[rb]] = R[ra]
0110	I	movh	Move High	R[0] = Imm, R[0](3:0)
0111	I	movl	Move Low	R[0] = R[0](7:4), Imm
Free Slot				
1111	R	nop	No Operation	

Lane 2: Controle Tipo R

Opcode	Tipo	Mnemonic	Nome	Operação
Controle				
0000	R	brzr	Branch On Zero Register	if (R[ra] == 0) PC = R[rb]
0001	I	brzi	Branch On Zero Immediate	if (R[0] == 0) PC = PC + Imm
0010	R	jr	Jump Register	PC = R[rb]
0011	NULL	NULL	NULL	NULL
1111	R	nop	No Operation	

Lanes 3 e 4: Aritmética e Lógica

Opcode	Tipo	Mnemonic	Nome	Operação
Aritmética e Lógica				
1000	R	add	Add	R[ra] = R[ra] + R[rb]
1001	R	sub	Sub	R[ra] = R[ra] - R[rb]
1010	NULL	NULL	NULL	NULL
1011	R	or	Or	R[ra] = R[ra] — R[rb]
1100	R	not	Not	R[ra] = ! R[rb]
1101	R	slr	Shift Left Register	R[ra] = R[ra] ¡¡ R[rb]
1110	R	srr	Shift Right Register	R[ra] = R[ra] ¿¿ R[rb]
Free Slot				
1111	R	nop	No Operation	

3 Implementação e Organização

3.0.1 Instrução 1

3.0.2 Instrução 2

3.0.3 Predicação

4 Resultado