1. **Arquitetura de Processadores: SIMD (Conjunto Único de Instruções, Múltiplos Conjuntos de Dados)**

Os computadores SIMD operam sobre vetores de dados. Por exemplo, quando uma única instrução SIMD soma 64 números, o hardware da máquina SIMD envia 64 conjuntos de dados para 64 ULAs, obtendo as 64 somas dentro de um único ciclo de clock.

A grande vantagem das máquinas SIMD é o fato de as unidades de execução paralela estarem sincronizadas, todas elas respondendo ao comando de uma única instrução, originada de um único *program conter* (PC). Apesar de nas máquinas SIMD cada uma das unidades executar a mesma instrução, cada uma dessas unidades tem seu próprio registrador de endereços, podendo, portanto, cada uma delas opera sobre dados armazenados em endereços diferentes.

A motivação original por trás das máquinas SIMD era amortizar o custo do desenvolvimento da unidade de controle por dezenas de unidade de execução. Outra vantagem é o tamanho reduzido da memória para programas. A técnica da memória virtual e o custo reduzido das memórias DRAM reduziram substancialmente a importância dessa vantagem.

As arquiteturas realmente SIMD têm em seu conjunto de instruções tanto instruções SISD (Conjunto único de instruções, Conjunto único de dados) quanto SIMD. Nessas máquinas é como se houvesse um hospedeiro SISD para realizar as operações sequenciais como desvios e cálculos de endereços. As instruções SIMD são distribuídas em broadcast para todas as unidades de execução, cada uma com seu conjunto particular de registradores. As unidades de execução se apoiam em redes de conexão para realizar a troca de dados com os demais componentes do sistema.

As arquiteturas SIMD funcionam melhor quando tratam com arrays e loops *for*. Portanto, para que possamos aproveitar ao máximo as perspectivas de paralelismo em máquinas SIMD, deve haver muitos conjuntos de dados a serem operados, ou paralelismo no nível dos dados. As máquinas SIMD experimentam seu pior momento quando da execução de comandos *case* ou *switch*, em que cada unidade de execução precisa executar uma operação diferente sobre seus dados, operação essa que depende do valor de um parâmetro da instrução. Algumas das unidades de execução devem ser desabilitadas, e somente aquelas com os dados a serem processados devem ser autorizadas a continuar a execução.

1. **Arquitetura de Processadores: SUPERESCALAR**

Um processador superescalar é aquele no qual são usadas várias *pipelines* de instrução independentes. Cada pipeline tem diversos estágios, podendo manipular várias instruções a cada instante. O uso de várias *pipelines* introduz um novo nível de paralelismo, possibilitando processar diversos fluxos de instrução de cada vez. Um processador superescalar explora o que é conhecido como paralelismo no nível de instruções, que diz respeito ao nível em que instruções de um programa podem ser executadas em paralelo.

Um processador superescalar busca tipicamente várias instruções de cada vez, e tenta encontrar instruções próximas que sejam independentes umas das outras e que podem, portanto, ser executadas em paralelo. Se os dados de entrada de uma instrução dependem da saída produzida por uma instrução precedente, a execução dessa instrução não pode ser completada antes ou ao mesmo tempo que a execução da instrução da qual ela depende. Uma vez que essas dependências de dados sejam identificadas, o processador pode executar e completar instruções em uma ordem diferente da que ocorre no código de máquina original.

O processador pode eliminar dependências desnecessárias por meio do uso de registradores adicionais e de renomeação de referências a registradores no código original.

Uma implementação superescalar de uma arquitetura de processador é uma implementação na qual instruções usuais (aritmética de números inteiros e de números de ponto flutuante, instruções de carga e armazenamento e instruções de desvios) podem ser iniciadas simultaneamente e executadas independentemente. Tais implementações suscitam diversas questões de projeto bastante complexas, relacionadas à *pipeline* de instruções.

O projeto superescalar apareceu logo depois dos projetos de arquitetura RISC. Embora as técnicas de arquitetura superescalar possam ser aplicadas mais diretamente a uma arquitetura RISC com um conjunto simplificado de instruções, a abordagem superescalar também pode ser usada em arquiteturas CISC.

A essência da abordagem superescalar é a habilidade de executar instruções independentemente, em diferentes *pipelines*. Esse conceito pode ser ainda mais explorado, permitindo que instruções sejam executadas em uma ordem diferente da ordem em que aparecem no programa.

1. **Arquitetura de Processadores: VLIW**

A arquitetura *Very Long Instruction Word* (VLIW) tenta alcançar maiores níveis de paralelismo de instrução pela execução de instruções longas compostas por múltiplas operações. As palavras de instrução longas consistem de várias operações aritméticas, lógicas e de controle cada uma das quais poderia ser uma operação individual em um processador RISC.

O processador VLIW executa o conjunto de operações concorrentemente, alcançando assim um alto grau de paralelismo no nível de instrução. É responsabilidade do compilador escalonar as operações de modo a utilizar o melhor possível as unidades funcionais disponíveis no processador.

Cada instrução longa é formada por um conjunto de operações que podem ser executas em paralelo. As instruções longas são feitas através de modos de escalonamento por “software” aplicadas em tempo de compilação ou através da compactação do código gerado por um compilador convencional sendo responsável por entregar as instruções de certa forma que a arquitetura seja aproveitada ao máximo. A arquitetura é organizada com múltiplas unidades funcionais e banco de registradores com múltiplas portas de leitura.

Assim *Very Long Instruction Word* ou VLIW, é uma arquitetura de CPU que executa um grupo de instruções ao mesmo tempo. Isso tudo pode ser controlado por um compilador no qual garante que as instruções a serem processadas não sejam dependentes entre si para que possam para que ser executadas ao mesmo tempo sem perda de lógica do processamento