# Conjunto de Instruções Multimídia

## [Uma pequena Introdução] *

Gabriel Dieterich Cavalcante†
RA 079738
Instituto de Computação
Laboratório de Administração e Segurança
Campinas, Brasil
gabriel@las.ic.unicamp.br

## ABSTRACT

Este artigo contém uma breve descrição do funcionamento de instruções multimídia, uma pequena contextualização histórica e alguns exemplos de mercado deste conjunto de instruções. Além de um breve conjunto de publicações que analisaram o ganho de desempenho por meio do uso deste tipo de instrução.

## 1. INTRODUÇÃO

O aumento do uso de computação como suporte para operações multimídia, ou seja, criação e edição de filmes, músicas e imagens; Tanto quanto o uso de estações de trabalho para cálculos avançados, exigiu um novo tipo de instruções especializadas com objetivo de eliminar o contraste entre um *datapath* grande—para aplicações comuns—e o uso de pequenos tipos de dados exigidos por aplicações multimídia.

Muitos sistemas gráficos usam 8 *bits* para representar cada cor primária, além de 8 *bits* para representar a localização de um *pixel*. Para o processamento de áudio são necessários mais que 8 *bits*, porém 16 *bits* bastam. Na década de noventa, programas convencionais usualmente necessitavam de registradores de 32 e 64 *bits* , desta forma utilizá-los para instruções multimídia poderia se interpretado como desperdício.

Esta interpretação motivou os fabricantes de *hardware* a utilizar uma técnica já conhecida para atender outras necessidades: as instruções vetoriais. As instruções vetoriais precisaram ser inseridas na arquitetura de microprocessadores devido a este comportamento atípico dos programas multimídia. Além disso, essas instruções foram responsáveis pelo mercado abraçar a metodologia SIMD—single instruction, multiple data.

A metodologia SIMD aplicada à instruções multimídia é um pouco diferente daquela que consiste no uso de vários processadores para executar a mesma instrução. As instruções multimídia são executadas por um único processador, que empacota pequenos tipos de dados em registradores de maior capacidade—64 e 128 *bits*—para posteriormente executar a instruções sobre todos estes dados em paralelo.

A implantação de instruções vetoriais foi fundamental para obter paralelismo através de estruturas especializadas. Paralelismo que também foi impulsionado pela grande quantidade de operações sequenciais por parte dos programas multimídia.

O restante deste documento é organizado da seguinte maneira: a Seção 2 analisa sumariamente o comportamento geral das aplicações multimídia; a Seção 3 mostra algumas instruções multimídia empregadas no mercado atualmente; a Seção 4 mostra os principais projetos de mercado que incluem um conjunto de instruções multimídia; a Seção 5 cita alguns estudos realizados com o objetivo de validar a eficácia das arquiteturas que possuem instruções multimídia; a Seção 6 contém alguns exemplos de como os programadores podem embutir instruções multimídia em seus códigos e finalmente a Seção 7 faz uma conclusão sobre o material apresentado neste texto.

## 2. COMPORTAMENTO DE APLICAÇÕES MULTIMÍDIA

Existem várias características gerais de aplicações multimídia, grande parte delas é considerada pelos processadores de propósito geral modernos. Algumas delas são[7]:

- Tempo real: videoconferências e comércio eletrônico frequentemente necessitam de resposta em tempo real. Além disso, é necessária alguma qualidade de serviço;
- Processamento de *streaming* de vídeo: rotineiramente aplicações multimídia possuem seu código *on-chip* e processam dados *off-chip*;
- Granularidade considerável: tipicamente instruções multimídia executam a mesma operação sob diferentes da-

*Trabalho apresentado a disciplina de Arquitetura de Computadores I, ministrada Pelo Professor Dr. Paulo Cesar Centoducatte, no ano de 2010.

†Aluno de Doutorado - 2010

dos(pixels). Desde que essas operações sejam independentes, é possível explorar esta característica;

- Reorganização de dados: além do comportamento SIMD das operações multimídia, a maioria das aplicações multimídia precisa reorganizar fontes de dados. Desta forma, não são fáceis de tratar em arquiteturas SIMD tradicionais, onde a reorganização pode ser cara;

- Pequenos Laços: aplicações multimídia gastam 95% da seu tempo de execução em laços aninhados—tipicamente dois, operações sobre matrizes são frequentes. Estes laços possuem um grande número de iterações, tipicamente 10 ou mais. Alguns deles podem ter centenas ou até milhares de iterações;

- Largura de Banda de Memória: aplicações multimídia operam sob grandes conjuntos de dados, o que sobrecarrega o barramento de memória;

- Pequenos Tipos de Dados: normalmente utilizam inteiros de 16 *bits* ou menos. Além disso, executam um número de operações aritméticas significativamente maior do que aplicações de propósito geral.

Existe uma grande variedade de algoritmos para processamento, captura, transmissão, edição e visualização de dados multimídia. Objetos deste tipo podem consistir em texto, desenho à mão, gráficos 2D/3D e objetos de áudio. Além disso, existem vários padrões de compressão de vídeo, áudio e figuras. Exemplos são: MPEG-1/2; MPEG-4; MPEG-7; JPEG; JPEG2000; H.263; H.264; X.264; MP3; OGG; FLAC etc. Todos estes formatos desafiam projetistas de *hardware* e *software*[1].

# 3. INSTRUÇÕES MULTIMÍDIA

Há muito tempo é reconhecido que instruções SIMD podem ser um bom caminho para adquirir melhor desempenho. As arquiteturas SIMD surgiram na década de 60, porém a teoria permaneceu inalterada ao longo de cinco décadas; uma simples instrução operando sobre vários elementos de dados.

Atualmente vários microprocessadores apresentam um consolidado conjunto de instruções que aplica as ideias de SIMD especialmente para aplicações multimídia. Estes são chamados "extensões multimídia"[11]. Mais detalhes sobre implementações de mercado serão vistas na Seção 4.

Um microprocessador com extensão multimídia típico normalmente inclui instruções SIMD que são executadas sobre 2-16 operandos simultaneamente, eles também podem ser processados aos pares.

A Figura 1 esboça a diferença entre uma instrução normal de 32 *bits* e uma instrução SIMD sobre 8 operandos de 4 *bits*. No caso normal, toda a operação considera dois registradores fonte e seu resultado é escrito em um registrador destino de 32 *bits*. Na instrução SIMD cada registrador fonte de 32 *bits* é interpretado como quatro sub-palavras de 8 *bits* cada. A operação é realizada em cada par de palavras entre os dois registradores e o resultado é escrito posicionado nas sub-palavras resultantes do registrador destino de 32 *bits*.

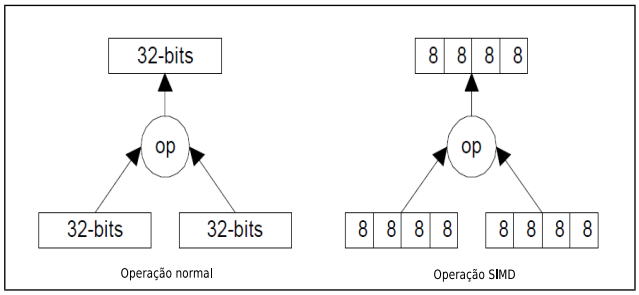


**Figure 1: Diferença entre instruções multimídia e de propósito geral.**

O principal objetivo de instruções multimídia do tipo SIMD em um conjunto de instruções envolve basicamente a exploração do paralelismo de sub-palavras, principalmente pelo baixo custo envolvido nesta adaptação[19]. Por exemplo, um processador com unidades aritméticas de 64 *bits* particionada pode suportar simultaneamente 8 operandos de 16 *bits* com apenas 3 portas para os registradores (2 para leitura e 1 para escrita). Enquanto que um processador com quatro unidades diferentes necessitaria de 12 portas.

## 3.1 Tipos de dados

Os tipos de dados suportados pelos diferentes conjuntos de instrução multimídia incluem valores {*signed* | *unsigned*} de {8, 16, 32, 64} *bits*, além de pontos flutuantes com pouca precisão. A seguir serão mostradas algumas instruções tipicamente implementadas para operação com valores inteiros e pontos flutuantes.

### *3.1.1 Inteiros*

Imagem e vídeo normalmente está armazenado através de inteiros empacotados de 8 *bits*, a porém maioria do processamento intermediário necessita de mais. Dados *signed* de 16 *bits* servem para áudio e fala. No final do espectro, dados com 32 *bits* ou mais são usados como acumuladores.

**Conversão de Tipo e Largura de Dados**

As instruções multimídia frequentemente fazem uso de mudança na largura dos dados. A operação chamada empacotamento reduz a largura de um dado, enquanto que a operação de desempacotamento aumenta a largura de um dado. Isso é possível com uso de outro registrador. Por exemplo, um registrador de 64 *bits* contendo 4 palavras de 16 *bits* cada, pode ser convertido colocando os 8 *bits* menos ou mais significativos em outro registrador de 32 *bits*.

**Saturação e *Overflow***

Operações tradicionais de inteiros lidam com *overflow* usando o mesmo comportamento de uma operação de módulo, retornando o resto da divisão do resultado pelo valor máximo ou mínimo que a arquitetura suporta. Este comportamento é indesejável em várias aplicações multimídia, onde a melhor forma é saturar um *overflow* positivo ou negativo(*underflow*), ou seja, deixar o maior ou o menor valor que arquitetura pode representar. A Figura 2 mostra a diferença entre os dois tratamentos de *over/underflow*, imagine uma operação de soma de brilhos em uma imagem, caso o

valor total exceda o máximo é desejado que o valor máximo seja retornado, não o resto da divisão total como nas instruções aritméticas comuns.

Além disso, existem mecanismos de saturação quem lidam eficientemente quando há possibilidade de haver múltiplos *overflows* em valores empacotados.

**Adição e Subtração**

Tratam de operações simples sobre números particionados, conforme visto na Figura 1. As arquiteturas do mercado possuem diferentes instruções para cada tamanho de dado empacotado, ou seja, um *opcode* para cada tipo de particionamento— 8, 16 ou 32 *bits*— e além disso para decidir qual será o tratamento em caso de *overflow*.

**Soma das diferenças Absolutas**

SAD—*Sum of Absolute Differences*—opera em um par de registradores empacotados com dados de 8 *bits*, somando as diferenças absolutas entre os respectivos elementos de cada registrador. O resultado é acumulado em um terceiro registrador.

Esta operação consiste em uma simples maneira de procurar objetos dentro de uma imagem, porém pode não ser muito confiável devido aos efeitos de fatores contextuais como: cor, ponto de vista, tamanho ou forma. SAD deve ser usado em conjunto com outra técnica de reconhecimento para aumentar a confiabilidade de seus resultados.

**Multiplicações**

A multiplicação comporta-se de maneira semelhante a adição quando se trata de valores empacotados. Porém entre as diversas arquiteturas comercias há bastante diferença nas implementações. Isso devido ao fato de que operações de multiplicação serem mais demoradas. Além disso, são de difícil manejo, já que o espaço requerido para armazenar seu resultado pode ser muitas vezes maior do que o requerido para os operandos. Assim os fabricantes adotaram diferentes abordagens para tratar do resultado de uma multiplicação deste tipo:

- Redução: Consiste em somar os resultados das multiplicações entre os dados particionados—*multiply-add*;
- Par/Ímpar: Multiplicação seletiva, multiplicar os fontes par ou ímpar (posição no registrador com dados empacotados), assim pode-se escrever o resultado inteiro no registrador destino;
- Truncagem: Predefinir um conjunto de *bits* resultantes que serão ignorados no resultado, geralmente os menos significativos;
- Maior Precisão no Resultado: Adicionar um acumulador maior—192 *bits*—para armazenar o resultado;
- Primitivas de Multiplicação: Criar primitivas ou subinstruções que geram o resultado da multiplicação.

***Shifts*(Deslocamento)**

Amplamente conhecida otimização para realizar operações aritméticas binárias. Por exemplo, multiplicar um valor $N$ múltiplo de 2 consiste em deslocar *bits* à esquerda $\log_2 N$ vezes. A divisão ocorre de maneira análoga, porém os *bits* são deslocados para a direita. Algumas arquiteturas implementaram o chamado *packed-shift* que considera todos os elementos empacotados em um registrador.

**Operações de comunicação de dados**

Este tipo de operação visa a reorganização das palavras empacotadas em um registrador, ou ainda entre registradores. Esta operação é independente do tipo de dados, ou seja, não precisa saber como um campo com *bits* precisa ser interpretado. Para este tipo de instrução temos os seguintes comportamentos:

- *Merge*: Alternar dados da parte superior e inferior de dois registradores;
- *Align/Rotate*: Reajuste de palavras particionadas em dois registradores. Semelhante a rotacionar *bits* em um registrador, porém o número de *bits* rotacionados deve ser múltiplo do tamanho das palavras empacotadas;
- Inserção/Remoção: Extrair um elemento empacotado como escalar ou vice-versa;
- *Shuffle/Permute*: Permite escolher uma ordem diferente para as palavras empacotadas em um registrador.

O comportamento dessas instruções é mostrado na Figura 3, onde estão representados 3 registradores empacotados como fonte além de um registrador com dado comum.

**Mínimo/Máximo**

Tem como saída o maior ou o menor valor entre as subpalavras correspondentes de cada registrador, o resultado contém as respectivas maiores palavras entre os dois registradores. Usualmente estas operações utilizam os mesmos comparadores usados para o cálculo de saturação, porém não mais comparando dados de registrador com uma constante.

**Média**

Em adição ao cálculo da soma das diferenças absolutas, a média é necessária para várias funções de Interpolação. A interpolação é feita através do cálculo da média de valores de um conjunto de pixels com os seus vizinhos. Esta operação é muito usada ao ampliar-se imagens, tentando suavizar a perda de qualidade.

### 3.1.2 Instruções de Ponto Flutuante

Grande parte das arquiteturas multimídia possuem dados de ponto flutuante com precisão simples e dupla—32 e 64 *bits* respectivamente. A precisão simples geralmente é suficiente, porém aplicações científicas requerem maior precisão. A abordagem SIMD para pontos flutuantes é útil,

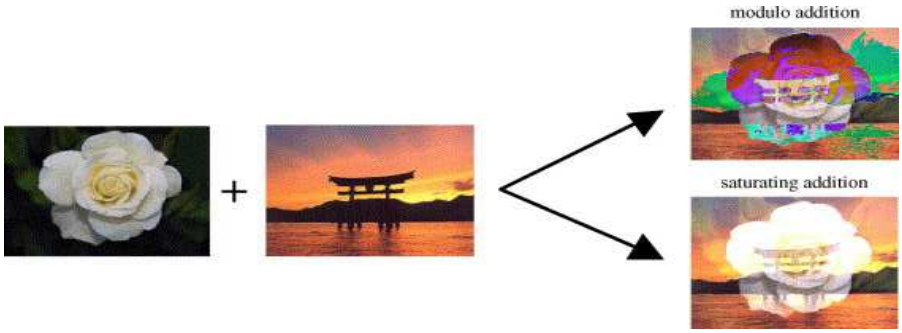


**Figure 2: Exemplificação dos dois modos de *overflow*[18].**

porém apresenta menor ganho de desempenho comparada a de inteiros. Instruções de ponto flutuante incluem, além das aritméticas básicas, operações de aproximação, inversão e raiz quadrada. Estes três tipos de instrução são utilizadas por aplicações de geometria 3D e são implementadas a partir de tabelas preestabelecidas. Dentre algumas operações multimídia genéricas de ponto flutuante temos:

- Raiz Quadrada: Normalmente baseada em dados preestabelecidos, como tabelas de mantissa;
- Aproximação: Algumas aproximações também utilizam este tipo de tabela;
- Arredondamento: Algumas arquiteturas possuem diferentes métodos de arredondamento, porém as mais usadas são o modelo *flush-to-zero*, principalmente por aplicações multimídia não aceitarem erros de arredondamento[20];
- Operações Escalares: Algumas arquiteturas possuem suporte a operações escalares de ponto flutuante;

**Exceções**

Aparentemente é o mesmo problema que ocorre quando há *overflow* de valores empacotados. Verificar os resultados e gerar uma exceção a partir de dados empacotados é um desafio, pois requer muito tempo para descobrir qual dado levou a geração da exceção. Felizmente, na maioria dos casos onde há exceção é possível devolver algum valor plausível como resultado da operação e prosseguir com a execução. Isso aumenta a velocidade de execução pois não há erro explícito e nenhuma verificação formal precisa ser realizada. Este método funciona de maneira semelhante à operação de saturação, onde o máximo/mínimo valor é retornado em substituição ao tratamento do *overflow/underflow*.

### 3.1.3 Comparações e Controle de Fluxo

Operar sobre vários elementos em paralelo pode ser problemático se, alguma operação precisa ser executada apenas se uma verificação condicional de um operando for verdadeira , pois é complicado gerar um flag ou exceção diferente para cada execução em paralelo. Existem duas soluções para este problema.

A primeira consiste em esquemas de máscara, um elemento de máscara é um vetor onde cada elemento empacotado contém uns ou zeros. Operações de comparação resultam em um elemento de máscara que corresponde ao tamanho dos elementos empacotados. Por exemplo, o resultado de uma comparação entre elementos empacotados de 16 *bits* é uma máscara de 64 *bits* contendo 4 elementos de 16 *bits*, cada um deles contendo zeros ou uns—`0x0000 para condição verdadeira, caso contrário 0xffff`. Essas máscaras são usadas em conjunção com operações lógicas—`AND, NAND e OR`—para obter o resultado desejado.

Um outra opção são os vetores de *bits*, onde um único *bit* representa o resultado de uma comparação lógica para cada elemento particionado. Vetores de *bits* são tipicamente usados em instruções de armazenamento parcial para gerar o resultado desejado em memória.

### 3.1.4 Operações Polimorfas

São aquelas onde a mesma instrução pode ser usada independentemente do tamanho da partição dentro do registrador—operações diretamente sobre cada *bit*. Algumas arquiteturas provêm instruções para as mais simples operações lógicas—`AND e OR`—serem feitas "*bit*-a-*bit*".

### 3.1.5 Operações com memória

As únicas instruções multimídia existentes que fazem uso da memória são as de *load* e *store*, sendo que os tipos de acesso mais comuns são os sequenciais, indicados apenas pelo endereço do dado a ser lido. A natureza dos dados empacotados

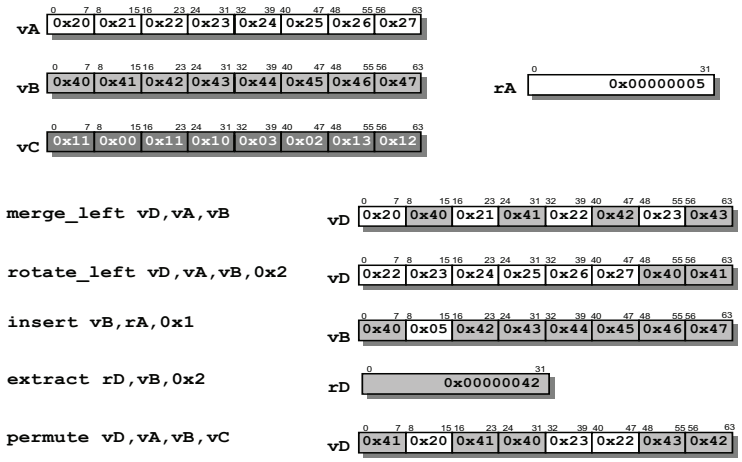


**Figure 3: Instruções de comunicação de dados.**

pode causar algum problema com o acesso sequencial da memória, algumas arquiteturas tentaram criar um novo tipo de acesso "não sequencial", porém estas ideias foram abandonados devido ao *overhead* adicionado no barramento.

Frequentemente um vetor deve representar um pixel inteiro, contendo sua posição e três cores RGB. A maioria dos aplicativos não faz armazenamento sequencial *per*-pixel, ou seja armazena sequencialmente as cores e posições. Para este cenário seria desejável fazer *loads* e *stores* não sequenciais.

# 4. PRINCIPAIS REPRESENTANTES

Atualmente a maioria dos produtores de microprocessadores definiram extensões multimídia para suas arquiteturas. Existem grandes diferenças no tipo de suporte que cada extensão multimídia oferece, nesta seção serão mostrados—quando possível—detalhes e objetivos do projeto de diferentes arquiteturas com suporte a instruções multimídia.

## 4.1 Hewlett Packard MAX-1 — 1994

Foi o primeiro microprocessador a implantar um conjunto de instruções especializadas que operavam apenas sobre números inteiros. Consistiu em uma implementação focada em decodificações do formato MPEG-1, onde o objetivo geral era não afetar a complexidade, tempo de ciclo ou área de chip.

Ao invés de adicionar circuitos especializados para a decodificação de MPEG, os projetistas primeiramente analisaram quais eram as instruções mais utilizadas para este propósito na arquitetura anterior, sem extensão multimídia. A quebra destas instruções em primitivas mais simples, possibilitou um ganho significativo no desempenho[13].

## 4.2 Sun VIS – Visual Instruction Set — 1995

Implementado no *hardware* do processador UltraSPARC I e posteriores. O foco era criar uma plataforma padrão para aplicações multimídia—aplicações de visualização e modelagem 3D; renderização de MPEG-1/MPEG-2. Diminuiu o custo de máquinas Sun por não necessitar mais de dispositivos gráficos especializados[22].

## 4.3 Hewlett Packard MAX-2 — 1995

Sucessor do HP-MAX1, apresenta como principal característica, duas unidades de multiplicação com diferentes abordagens. Uma delas especializada para aplicações 3D e de áudio, utilizando um acumulador para pontos flutuantes. A outra unidade era utilizada para operações de multiplicações mais simples, como multiplacões envolvendo constantes [14].

## 4.4 MIPS MDMX — 1996

Tem sua arquitetura centrada em um registrador acumulador de 192 *bits*, o qual operações aritméticas como soma sequencial utilizam para obter maior precisão. Além disso possui um conjunto de 32 registradores de 64 *bits*, dedicados a instruções multimídia[9].

## 4.5 MIPS MIPS-V - MIPS-64 — 1996

Pioneiro ao implementar instruções multimídia para registradores de ponto flutuante dentro do microprocessador. Complementou o MDMX para aplicações de OpenGL como geometria 3D e VRML-*Virtual Reality Modeling Language*[9].

## 4.6 Intel MMX — 1997

O projeto MMX teve como prioridade o melhoramento de aplicações multimídia, comunicações e as emergentes ferramentas do mundo Web. Vários formatos foram estudados pelos projetistas—MPEG-1/2; música em geral; compressão de dados de voz; reconhecimento de voz; processamento de imagens etc. Estas fontes de dados foram submetidas a ferramentas de *profiling*, para determinar padrões dentro de seus conjuntos de instruções.

MMX foi capaz de suportar os sistemas operacionais da época, o que significa que não precisaram ser considerados novos estados do processador (novos registradores ou exceções). Este conjunto de instruções foi projetado para utilizar a arquitetura de pontos flutuantes já existentes em seu sucessor. Desta forma, instruções multimídia e de ponto flutuante não podiam ser utilizadas concorrentemente[15]. Além disso, para retornar a pilha de pontos flutuantes após uma instrução de MMX, é necessário o uso de uma instrução EMMS—*Empty Multimedia State*.

## 4.7 DEC MVI — 1997

As instruções MVI—*Motion Video Instructions*—foram incorporadas primeiramente pelo Alpha 21164PC[2]. A DEC idealizadora do MVI foi comprada pela HP. O foco do MVI era realizar teleconferências a 30 *frames* por segundo com áudio estéreo, semelhante aos *hardwares* dedicados da época[3]. A DEC implementou apenas 13 instruções multimídia alegando que largura de banda de memória insuficiente para alimentar o processador com tanto paralelismo. A DEC alegou também que o conjunto MMX era complexo devido à deficiências das arquiteturas anteriores.

## 4.8 AMD 3DNow! — 1998

Após comprar a licença para uso do MMX da Intel, a AMD decidiu expandi-lo, criando o conjunto "3DNow!" para o seu processador K6. 3DNow! utiliza os mesmos registradores e o mesmo conjunto de instruções para formatos básicos que o MMX, porém adiciona um tipo particionado de pontos flutuantes. Isso permite que duas operações simples de ponto flutuante sejam executadas em paralelo. Durante as primeiras análises de instruções de ponto flutuante em códigos multimídia, os projetistas do 3DNow! encontraram duas possibilidades. A primeira era criar uma extensão de pontos flutuantes para o MMX, esta alternativa foi tomada. A segunda era criar um conjunto completo de instruções independentes como no projeto Motorola Altivec (um grande elevado número de registradores grandes, instruções com quatro operandos)[16].

## 4.9 Intel SSE — 1999

*Internet Streaming SIMD Extension*, é sumariamente uma extensão de ponto flutuante projetado para o cálculo de operação geométricas 3D, *encoding/decoding* de vídeo e reconhecimento de voz. Incorporou sugestões de produtores de software do MMX para a criação de uma nova forma de agrupar inteiros(paralelismo de sub-palavras). Além disso, como a AMD criou novas instruções para operação multimídia com ponto flutuante.

Este conjunto de instruções adicionou um grupo totalmente novo de registradores, dedicado à operações multimídia. Desta forma, não há mais dados multimídia sobrecarregando registradores de ponto flutuante, possibilitando o uso de instruções de ponto flutuante e multimídia paralelamente.

Durante o projeto constatou-se que não era viável construir um *datapath* maior do que 128 *bits*, pois os benefícios não justificavam o alto custo. Assim, o Pentium III com suporte a SSE continha instruções de 128 *bits* e unidades de execução de 64 *bits*. Portanto cada instrução multimídia possui no mínimo duas instruções empacotadas, A unidade de execução é responsável por traduzir esta instrução longa para duas micro-instruções[20].

## 4.10 Motorola AltiVec — 1999

Embutido nos processadores Motorola 7400—PowerPC G4. Incluído nas máquinas Apple com objetivo de otimizar a decodificação de vídeo. Foram adicionados na arquitetura 32 registradores de 128 *bits*, que representam diversos tipos de vetores: inteiros com ou sem sinal de 8, 16 e 32 *bits*; pontos flutuante de 32 *bits*; pixels RGB de 16 *bits*e inteiros de 128 *bits*. Não são suportados valores de 64 *bits*[10].

O conjunto total de instruções é bastante extenso, contendo mais de 150 tipos de instruções, que podem conter até três registradores de origem e um de destino. As operações podem executar para até dois vetores. Tendo em vista os outros fabricantes, Altivec tem como objetivo um conjunto geral de aplicações do que apenas aplicações multimídia[8].

## 4.11 AMD Enhanced 3DNow! — 1999

Embarcado nos processadores Athlon—anteriormente conhecido como K7. Adicionou operações particionadas de inteiros e de pontos flutuantes, representando certa equivalência ao SSE. As operações inclusive possuem os mesmos *opcodes* incluídos no Intel SSE.

## 4.12 Intel SSE2 — 2000

Incluída no Pentium IV essa extensão ampliou as operações de inteiros particionadas do MMX. Foram adicionadas 68 instruções que utilizam os mesmos registradores que foram adicionados no SSE original. Nenhum estado arquitetural foi adicionado com SSE2.

## 4.13 Intel SSE3 — 2004

As instruções SSE3 foram introduzidas com o Pentium 4 "Prescott". A principal alteração em relação à SSE2 vem com a possibilidade de operar horizontalmente sobre dados de 1 único registrador em contraste com a operação vertical exigida com 2 registradores. Com a introdução do Hyper-Threading algumas instruções que auxiliam o tratamento de threads também foram acrescentadas além de operações de conversão de tipo de dados.

## 4.14 Intel SSE4 — 2007

A Intel adicionou 47 novas instruções aos processadores `Penryn`. Estas instruções foram adicionadas tendo em mente um aumento de performance para aplicações de processamento de vídeo, imagem e processamento de ambientes 3D[6].

Já nos `Nehalem` a Intel adicionou mais 7 instruções(total de 54 instruções) que visam melhorar a performance em processamento de strings, em processamentos especiais(a Intel

chama de Application-Targeted Accelerator) e processamentos de inteiros de 128 *bits*[6].

A organização no que toca aos registradores se manteve a mesma da extensão SSE3(8 registradores em modo 32 *bits* e 16 em modo 64 *bits*).

# 5. ANÁLISE DE DESEMPENHO

Existem diversos estudos que comprovam a eficiência das instruções multimídia em processadores de propósito geral. A metodologia mais usada consiste em medir o desempenho de ferramentas usando ou não instruções multimídia em diferentes arquiteturas. O estudo de Erickson *et al*[5] conclui que com pequenas alterações no código fonte é possível obter um ganho significativo de desempenho nas plataformas MIPS V MDMX.

Por sua vez, Tran & Cho[21] mostraram que o uso de instruções da extensão SSE2 para algoritmos de estimativa de movimento acelera de 3 a 5 vezes a sua execução. Existem vários outros estudos que comprovam a eficácia no uso de instruções multimídia[18][12][17].

# 6. PROGRAMANDO COM INSTRUÇÕES MULTIMÍDIA

Apenas incluir extensões multimídia em um processador de propósito geral não basta, um poderoso conjunto de instruções SIMD é inútil sem mecanismos que o utilizem.

## 6.1 Bibliotecas

Uma das mais simples maneiras de aumentar a eficácia através de instruções SIMD é reescrever o conjunto de bibliotecas compartilhadas usando-as. Todas as aplicações que fazer uso dessa biblioteca imediatamente farão uso do novo conjunto de instruções sem necessidade de recompilação. Entretanto, não haverá ganho se a aplicação não realizar chamadas para as funções da biblioteca que foram reescritas.

## 6.2 Macros

A vantagem de se utilizar macros para este propósito é que o compilador irá realizar otimizações específicas para a máquina como agendamento de instruções e alocação de registradores.

## 6.3 Compiladores

Idealmente, compiladores de alto nível devem ser capazes de identificar sistematicamente trechos de código e gerar instruções SIMD para eles. Instruções SIMD não precisam estar limitadas ao conjunto de aplicações multimídia, podendo ser aplicadas de forma genérica por qualquer aplicação. Atualmente nenhum compilador comercial age desta forma em qualquer plataforma. O leque de linguagens que permite que o programador especifique tipos de dados e semânticas de *overflow* é restrito, sendo exclusivamente usado para o desenvolvimento de aplicações multimídia.

## 6.4 Linguagem Assembly

O mais efetivo método para programação com instruções multimídia é através de manipulação de código *assembly*. Entretanto esta é uma metodologia complicada, sendo realizada apenas por programados especializados, além disso consiste em um método manual e tedioso[4].

# 7. CONCLUSÃO

Instruções especializadas foram introduzidas em microprocessadores para suportar a demanda de aplicações multimídia. O contraste entre grandes *datapaths* e relativamente pequenos tipos de dados fizeram com que a indústria abraçasse a teria SIMD—*single instruction, multiple data*. A metodologia SIMD teve apoio significativo da simplificação de circuitos de controle, tipicamente um dos mais complicados aspectos dos superescalares.

Os conjuntos de instrução multimídia foram inicialmente introduzidos visando melhoria de desempenho em aplicações de vídeo, hoje são usados em diversos ambientes. Deste modo, tornaram-se requisito obrigatório em todo processador de propósito geral. Além disso, um padrão foi estabelecido pela plataforma SSE, o que motivou a utilização de instruções multimídia por parte dos fabricantes de software.

Estudos revelam que instrução multimídia aceleram de 2 a 5 vezes a execução, porém deve-se levar em consideração que nem todos os compiladores estão aptos a extrair todo o paralelismo introduzido pelas instruções SIMD.

# 8. REFERENCES


[1] M. Assaf, T. Arima, W. Tobago, and A. RAJESH. GENERAL ARCHITECTURE AND INSTRUCTION SET ENHANCEMENTS FOR MULTIMEDIA APPLICATIONS.

[2] P. Bannon and Y. Saito. The alpha 21164PC microprocessor. In *IEEE Compcon'97. Proceedings*, pages 20–27, 1997.

[3] D. Carlson, R. Castelino, and R. Mueller. Multimedia extensions for a 550-MHz RISC microprocessor. *IEEE Journal of Solid-State Circuits*, 32(11):1618–1624, 1997.

[4] W. Chen, H. Reekie, S. Bhave, and E. Lee. Native signal processing on the Ultrasparc in the Ptolemyenvironment. In *Signals, Systems and Computers, 1996. 1996 Conference Record of the Thirtieth Asilomar Conference on*, pages 1368–1372, 1996.

[5] G. Erickson, F. Quarter, and D. Yew. RISC for graphics: A survey and analysis of multimedia extended instruction set architectures. *Electrical Engineering*, 8362, 1996.

[6] S. Fischer. Technical Overview of the 45nm Next Generation Intel CoreTM Microarchitecture (Penryn). In *Intel Developer Forum*, 2007.

[7] J. Fritts, W. Wolf, and B. Liu. Understanding multimedia application characteristics for designing programmable media processors. In *Proceedings of SPIE- The International Society for Optical Engineering*, volume 3655, pages 2–13. Citeseer, 1999.

[8] S. Fuller. Motorola's AltiVec$^{TM}$ Technology. *Motorola, Inc. white paper ALTIVECWP/D*, 1998.

[9] L. Gwennap. Digital, MIPS add multimedia extensions. *Microprocessor Report*, 10(15):24–28, 1996.

[10] L. Gwennap. AltiVec Vectorizes PowerPC. *Microprocessor Report*, 12(6):1–6, 1998.

[11] J. Hiser, S. Carr, and P. Sweany. Global Register Partitioning. In *Proceedings of the 2000 International Conference on Parallel Architectures and Compilation*

*Techniques*, page 13. IEEE Computer Society, 2000.

[12] W. Jiang, C. Mei, B. Huang, J. Li, J. Zhu, B. Zang, and C. Zhu. Boosting the performance of multimedia applications using SIMD instructions. In *Compiler Construction*, pages 59–75. Springer, 2005.

[13] R. Lee. Subword parallelism with MAX-2. *IEEE micro*, 16(4):51–59, 1996.

[14] R. Lee and L. Mcmahan. Mapping of application software to the multimedia instructions of generalpurpose microprocessors. 1997.

[15] M. Mittal, A. Peleg, and U. Weiser. Mmx technology architecture overview. Intel Technology Journal, 1997.

[16] S. Oberman, G. Favor, and F. Weber. AMD 3DNow! Technology: Architecture and Implementations. *IEEE MICRO*, pages 37–48, 1999.

[17] N. Slingerland and A. Smith. Cache performance for multimedia applications. In *Proceedings of the 15th international conference on Supercomputing*, page 217. ACM, 2001.

[18] N. Slingerland and A. Smith. Multimedia extensions for general purpose microprocessors: A survey. *Microprocessors and Microsystems*, 29(5):225–246, 2005.

[19] D. Talla, L. John, and D. Burger. Bottlenecks in multimedia processing with SIMD style extensions and architectural enhancements. *IEEE Transactions on Computers*, 52(8):1015–1031, 2003.

[20] S. Thakkur and T. Huff. Internet streaming SIMD extensions. *Computer*, 32(12):26–34, 1999.

[21] T. Tran, H. Cho, and S. Cho. Performance Enhancement of Motion Estimation Using SSE2 Technology. *World Academy of Science Engineering and Technology*, 30, July 2008.

[22] M. Tremblay, J. O'Connor, V. Narayanan, L. He, S. Inc, and C. Mountain View. VIS speeds new media processing. *IEEE micro*, 16(4):10–20, 1996.