Processamento de subpalavras em arquiteturas orientadas a palavras

Geraldo Lino de Campos

Departamento de Engenharia de Computação e Sistemas Digitais

Escola Politécnica da Universidade de São Paulo

Caixa Postal 8174 São Paulo 01065-970

e-mail: RTC@FPSP.FAPESP.BR

Resumo:

Apresenta-se uma proposta para tratamento de subpalavras (bytes e campos de 16 e 32 bits) em arquiteturas orientadas a palavras. A proposta é baseada na utilização de endereços expressos em bytes, agregando-se os bits de ordem mais baixa do endereço à palavra vinda da memória. Estes bits são utilizados por instruções especiais de extração e inserção. A proposta pode ser estendida para acessos desalinhados e palavras parciais de tamanho arbitrário, mesmo quando a arquitetura suportar apenas acessos à palavras individuais. Estuda-se as implicações de desempenho para várias arquiteturas e várias implementações de cache.

Abstract

This paper presents a proposal for handling sub-words (bytes, 16 and 32 bit-fields) in word-oriented architectures. The proposal is based on using addresses expressed in units of bytes, and on keeping the low-order bits of the address attached to words read from memory. These bits are used by especial instructions to do the extract/insert operations. The solution holds for any unaligned accesses to sub-words even when the architecture supports only aligned accesses to words. Performance implications for machines with or without caches are examined, and extensions to arbitrary bit fields are proposed.

1 – Introdução e notação

O projeto de novas arquiteturas de computadores apresenta alguns pontos em comum. A característica mais marcante é que os projetos são todos voltados para o desenvolvimento de microcomputadores, isto é, toda a lógica correspondente às antigas CPUs é integrada em um único componente. Esta característica é facilmente explicada pelo fato do tempo de propagação entre componentes ser muito significativo, e portanto comprometer substancialmente o desempenho final.

Outra característica comum aos novos projetos é possuir uma arquitetura orientada a palavras de 64 [DEC92] bits, e não orientada a byte, como ocorreu nos projetos dos 10 ou 20 anos anteriores.

O objetivo do presente trabalho é apresentar e discutir uma implementação de manipulação de caracteres que seja geral, eficiente, e realizada através de instruções do processador.

Neste trabalho a notação palavra será utilizada para indicar a palavra física; na maior parte dos casos, supõe-se que esta palavra tenha 64 bits. Apesar da presunção de que o acesso se realiza por palavras, convenciona-se que os endereços são expressos em bytes, como ocorre na arquitetura citada acima. Palavra dupla indicará duas palavras físicas consecutivas.

Existe uma razão simples para as arquiteturas serem orientadas a palavras, que é a dificuldade de construção de um subsistema de memória rápido e capaz de operar, ao mesmo tempo, com palavras completas e parciais.

Para que se possa aquilatar a importância das dificuldades do projeto de memórias para computadores de alto desempenho, é necessário examinar preliminarmente a diferença de velocidade de evolução do desempenho dos circuitos de memória e dos circuitos lógicos.

Os circuitos lógicos têm evoluído a uma taxa aproximada de 25% ao ano, dobrando em densidade e velocidade a cada dois anos [Hen90]; quanto às memórias, sua evolução tem sido diferente: a capacidade tem aumentado muito rapidamente, quadruplicando a cada três anos; por outro lado, a velocidade tem evoluído muito pouco, dobrando apenas a cada dez anos!

Assim, pode-se concluir que o desempenho de um computador tenderá cada vez mais a ser limitado mais pelas restrições de acesso à memória do que pela velocidade de processamento que evoluem em sua velocidade de comutação muito mais rapidamente que a velocidade de acesso das memórias. Como agravante, as memórias são fisicamente grandes, e se encontram distribuídas em vários chips, de modo que o tempo de propagação nas placas e o tempo de conversão dos sinais de níveis internos para níveis externos contribui ainda mais para o desequilíbrio entre a velocidade

do processador e a capacidade da memória fornecer os dados requeridos. Assim, é importante que o sistema como um todo coloque a máxima complexidade no processador, e não no subsistema de memória.

Várias arquiteturas propõem soluções parciais para atender aos acessos a palavras parciais. A arquitetura ALPHA, por exemplo, oferece soluções, aliás muito eficientes, para o caso de cadeias de caracteres formadas por bytes e alinhadas em palavras, isto é, começando no primeiro byte de um registrador. Não existem estudos mostrando que seja possível trabalhar sempre com cadeias alinhadas dessa forma, e uma abordagem intuitiva indica que não é sequer um caso freqüente. É verdade que a arquitetura ALPHA permite acessos externos não alinhados, o que pode aumentar o número de casos em que esse alinhamento possa ser conseguido. De qualquer forma, não é uma solução geral, e como exposto acima, dificilmente outras novas arquiteturas poderão arcar com o ônus de acessos desalinhados.

Um outro aspecto importante é prever a possibilidade do caracter possuir um número maior de bytes. A codificação usual em 8 bits é muito limitada, e funciona adequadamente somente para o inglês. Para outras línguas ocidentais tem-se que usar 10 códigos diferentes, os ISO 8859-1 a 8859-10, o que é uma péssima solução, pelos problemas de compatibilidade que causa. O sentido natural da evolução é a utilização de caracteres de 16 bits, provavelmente no padrão UNICODE. Assim, torna-se possível utilizar uma única codificação para cada caracter, não só das línguas ocidentais, como também das línguas orientais, que tem em conjunto aproximadamente 45.000 símbolos, e ainda dispor de um bom número de combinações disponíveis para futuras expansões. Modems com compactação automática incorporada podem permitir a transferência desses caracteres em redes de longa distância com praticamente a mesma eficiência da situação atual; outras formas de transmissão e armazenamento podem ser compactadas ou não, sem causar maiores problemas.

Em resumo, é importante que a solução adotada para o processamento de caracteres funcione igualmente bem com caracteres de 8 bits ou com outras subpalavras.

A implementação de palavras parciais desta proposta é baseada na preservação dos bits de ordem mais baixa do endereço especificado em anexo a todas as palavras vindas da memória. Pretende-se utilizá-la na implementação da arquitetura definida em [Cam92]. Esta implementação não usa nenhum tipo de rede de alinhamento (alignment network) e aproveita a lógica de deslocamento indispensável em qualquer unidade aritmética e lógica. A mesma idéia pode ser estendida para o processamento de palavras parciais de tamanho e alinhamento arbitrários, se a arquitetura o requerer.

Para dar um tratamento mais uniforme não só aos bytes mas também a outras subdivisões de palavras, será utilizada a notação palavra parcial para designar qualquer tipo de subdivisão de palavra. Assim, palavras parciais poderão designar um subconjunto arbitrário de bits de uma palavra, ou de uma palavra dupla. As palavras parciais formadas por 8, 16 ou 32 bits receberão a designação genérica de subpalavras, e receberão os nomes específicos de byte, dibyte e tetrabyte, conforme tenham 8, 16 ou 32 bits, respectivamente. Um acesso a uma subpalavra **s** num endereço de memória **x** será chamado de alinhado se **x mod s** = 0, e de desalinhado no complemento.

Campos em uma palavra serão descritos por [**a** : **b**], onde **a** designa o primeiro bit à esquerda afetado, e **b** o número de bits. Considera-se o bit 0 como o menos significativo.

A notação A << B significa A deslocado a esquerda por B bits. UNS designa uma palavra com todos os seus bits ligados.

O restante deste texto está organizado em 5 seções. A seção 2 apresenta o princípio de funcionamento, utilizando como exemplo uma máquina com um cache "write-back" e acessos alinhados. A seção 3 estuda o caso, mais complexo, de uma máquina com cache "write-through" ou com acesso desacoplado. A seção 4 discute alguns aspectos de custo, e a seção 5 apresenta extensões para o caso de acessos desalinhados e para palavras parciais arbitrárias. Finalmente, a seção 6 apresenta conclusões.

2 – Princípio de funcionamento

Nesta seção supõe-se que todos os campos sejam alinhados, e que a implementação disponha de um cache duplamente associativo e do tipo "write-back", de forma que leituras ou escritas repetidas em um mesmo endereço demorem apenas um ciclo. Estas restrições serão levantadas em seções posteriores.

As características básicas requeridas para a implementação desta proposta são:

- uso de endereços expressos em bytes, apesar da arquitetura ser orientada para palavras;
- os bits de ordem mais baixa, que expressam o número do byte dentro da palavra, são enviados para o subsistema de memória, lá preservados e posteriormente anexados aos dados dela recebidos, quando de sua transferência para o processador;
- estes bits adicionais são utilizados por instruções especiais para a extração ou inserção de subpalavras, e ignorados pelas outras instruções, com a ressalva que devem ser copiados pelas instruções de movimentação de registradores.

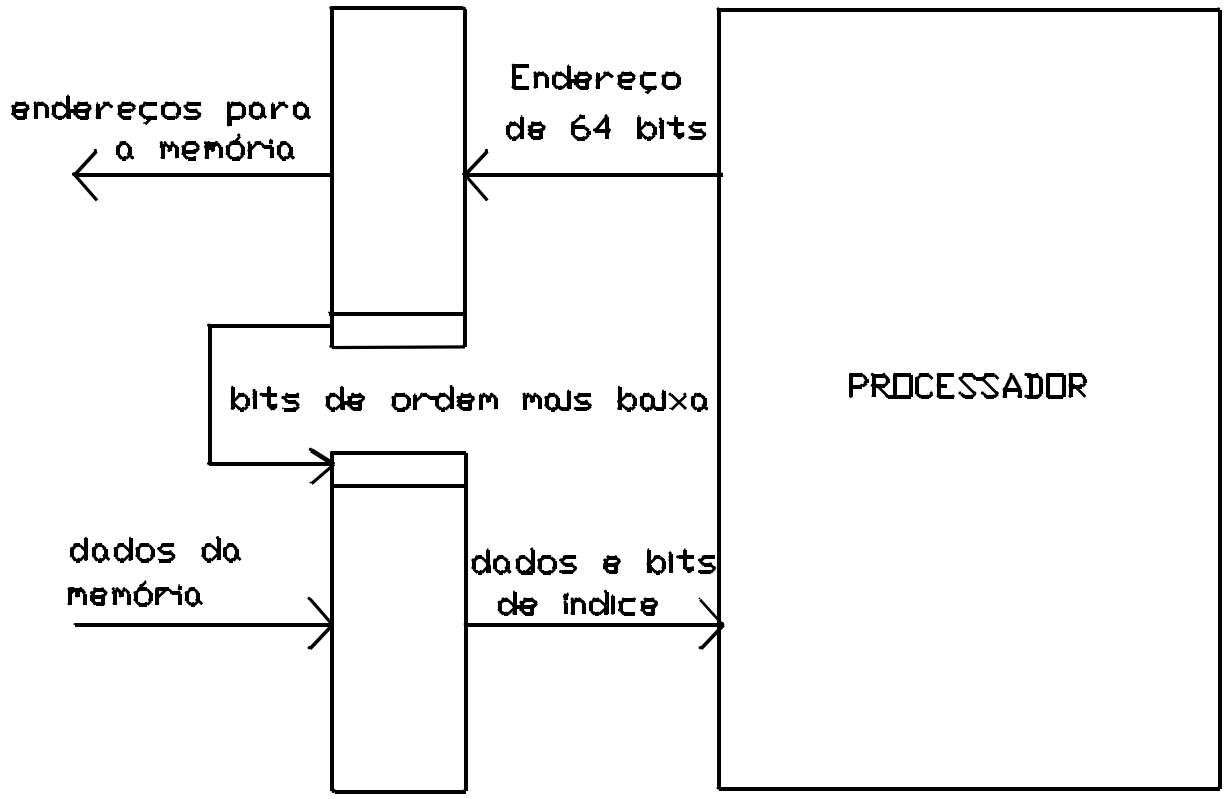

Figura 1 - Arquitetura proposta

No caso de palavras de 64 bits, que será usado sempre como exemplo, os 3 bits de ordem mais baixa do endereço serão os envolvidos neste processo.

Estes bits serão chamados de índice, e representados por I (Rx), onde RX é o número do registrador referido, e serão utilizados apenas pelas instruções ExtraiaX e InsiraX, onde X representa o número de bits afetados. Nesta seção, X estará restrito a algumas potências de 2 (8, 16, e 32).

A figura 1 apresenta uma descrição geral deste mecanismo.

A instrução ExtraiaX utiliza os bits de índice para extrair a palavra parcial especificada por eles; o comprimento especificado no nome da instrução; esta é a razão para se dispor de instruções diferentes para tamanhos diferentes. Esta operação ocorre usualmente imediatamente após a leitura de uma palavra, e é realizada através de lógica já presente na Unidade Aritmética e Lógica, eventualmente complementada por lógica específica para a realização destas operações.

Dependendo das especificações da arquitetura, poderão existir variantes destas instruções para operação com números com e sem sinal.

A funcionalidade da instrução ExtraiaX Rs, Rd é a seguinte:

Rd = Rs [I (Rs) * 8 + (X-1) : X]

Em palavras, a instrução ExtraiaX Rs, Rd extrai X bits da palavra Rs, na posição indicada pelo índice, e os coloca na posição de ordem mais baixa da palavra Rd.

Uma malha típica para a comparação de duas cadeias terminadas por nulls, com endereços iniciais arbitrários contidos nos registradores R1 e R4 seria:

```
Teste:      Carregue(R1),R2
            Extraia8  R2,R3
            Carregue(R4),R5
            Extraia8  R5,R6
            Compare   R3,R6
            DesvieNI  Diferentes
            Compare   R3,0
            DesvieI   Iguais
            Incremente R1,1
            Incremente R4,1
            Desvie    Teste
            ...
Iguais:     ...
            ...
Diferentes: ...
```

Alterando-se Extraia8 e o valor do incremento nas instruções correspondentes, o código acima compara cadeias com subpalavras de qualquer comprimento.

É importante lembrar que está-se supondo a presença de um cache, de forma que após a primeira leitura de uma palavra não haverá nenhuma penalidade para repetir-se esta leitura.

As operações de escrita seguem o mesmo padrão geral. A palavra de destino é lida em cada execução da malha, desde que isto é mais econômico que testar o limite da palavra, pelo fato da palavra de destino também estar no cache.

A funcionalidade da instrução InsiraX Rb, Rs, Rd é

Rd = Rb AND ((UNS AND 0 [X - 1 : X]<<I(Rb)*8) OR (Rs [X-1:X] << I(Rb)*8))

Em palavras, a instrução InsiraX Rb, Rs, copia a palavra Rs para o registrador de destino Rd, inserindo os X bits de ordem mais baixa da palavra Rb na posição da palavrad Rd indicada pelo índice.

No exemplo abaixo, a cadeia de subpalavras de tamanho X, terminada por um null, localizada a partir do endereço arbitrário apontado por R1 é copiada para o endereço contido em R4:

```
Copia:      Carregue     (R1),R2
            ExtraiaX     R2, R3
            Carregue     (R4),R5
            InsiraX          R5, R3, R5
            Guarde           (R4),R5
            DesvieZ          Copiado
            Incremente       R1,tamanho(X)
            Incremente       R4,tamanho(X)
            Desvie           Copia
Copiado:    ...
```

3 – Extensão para outras formas de implementação

Sob o ponto de vista de software, as propostas acima funcionam com qualquer implementação da arquitetura. A eficiência, entretanto, pode ser muito ruim para outras formas de implementar o subsistema de memória. Na seção 2 esta consideração foi evitada, uma vez que se supôs a existência de um cache duplamente associativo e "write-back", e portanto acessos repetidos a até duas palavras não apresenta problemas de desempenho. Nesta seção serão estudados outros casos importantes, envolvendo outras execuções físicas para o subsistema de memória.

3.1– Arquiteturas convencionais sem cache

Este caso é de menor importância, uma vez que é muito pouco provável o projeto de uma nova arquitetura que não se baseie no uso intensivo de caches. Neste caso, cada acesso a uma subpalavra exige um acesso à memória; entretanto, isto não afeta o desempenho, uma vez que é o mesmo que ocorreria se existissem acessos a palavras parciais realizados diretamente pelo subsistema de memória.

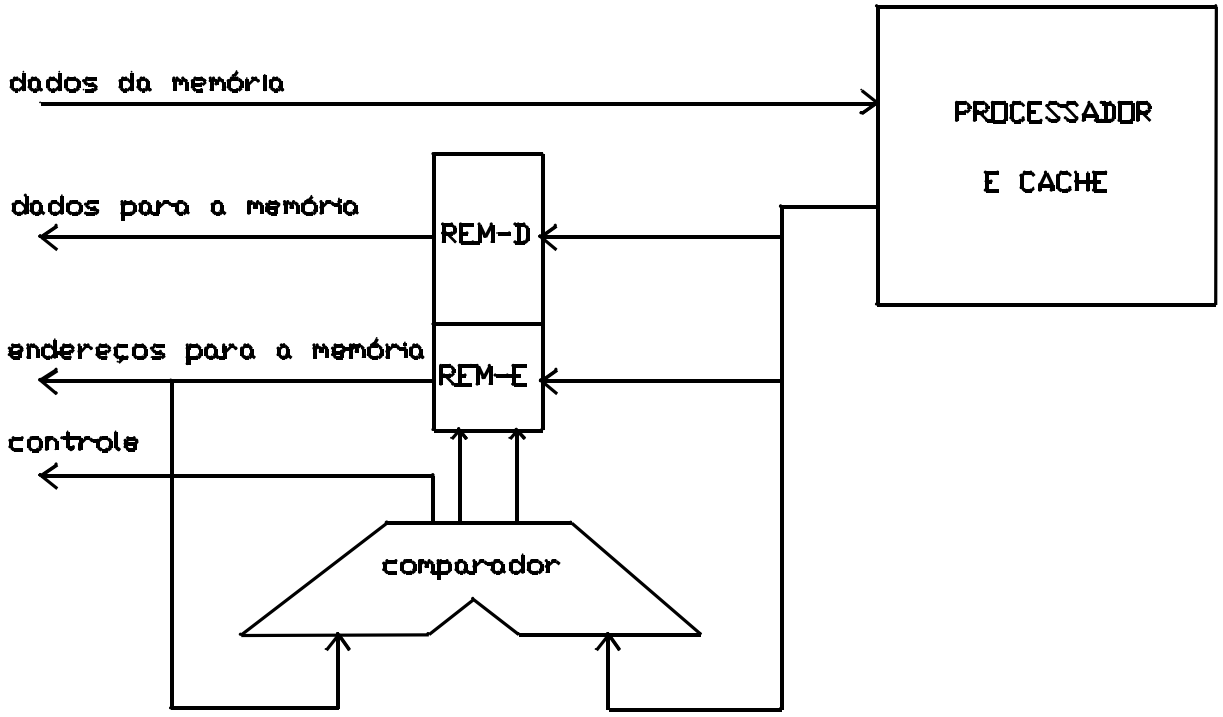

Figura 2 - Estrutura para evitar escritas repetidas

3.2– Máquinas com cache "write-back"

Este é o caso já estudado na seção 2. Como os caches "write-back" devolvem seu conteúdo para a memória somente sob demanda, o processo de leituras e escritas repetidas não causa qualquer perda de desempenho.

3.3– Máquinas com cache "write-through"

Neste caso, não há problemas com leituras repetidas de um mesmo endereço, uma vez que a palavra em questão já estará no cache a partir da segunda leitura. Entretanto, há um problema grave com a escrita, uma vez que neste tipo de cache uma palavra é escrita na memória toda vez que for alterada no cache; assim, escritas repetidas num mesmo endereço causarão escritas repetidas à memória, provocando a paralisação do processador e uma severa redução do desempenho.

Para que o algoritmo proposto possa operar neste tipo de máquina é necessário incluir um hardware adicional, formado por um registrador (chamado REM - registrador de escrita à memória, que é dividido em duas partes: D, que contém os dados a escrever, e E, que contém o endereço onde escrever) e um comparador colocados entre o cache e a memória principal, como mostra a figura 2. Sempre que uma palavra é escrita no cache, e portanto também encaminhada ao

subsistema de memória para escrita, seu endereço é comparado com o contido em REM-E; se forem diferentes, o valor contido em REM-D é enviado à memória, para ser escrito no endereço REM-E, e os novos valores colocados no registrador. Caso os endereços coincidam, a operação de escrita não é efetuada, e o valor vindo do processador é escrito em REM-D. Esta operação implica numa latência adicional para as operações de escrita à memória, mas não altera a duração do ciclo do processador, e portanto não compromete diretamente o desempenho. Evidentemente, a operação acima terá de ser eliminada em alguns casos, como durante a realização de operações de sincronização.

Caso a máquina já disponha de uma fila para escrita à memória, o hardware acima pode ser consideravelmente simplificado, restringindo-se ao comparador; os detalhes dependerão de como essa fila estiver implementada.

3.4– Arquiteturas com acesso e execução desacoplados

Arquiteturas com acesso e execução desacoplados, originalmente propostas em [Smi84], procuram antecipar tanto quanto possível os pedidos de dados à memória como uma forma de ocultar o tempo de acesso à memória. Se a antecipação for suficiente, quando o processador precisar do dado este já estará disponível em uma fila de entrada. Este mecanismo pode ser utilizado em lugar do cache de dados, o que normalmente ocorre neste tipo de arquitetura. Se isto não ocorrer e a arquitetura prever um cache, o problema já estará resolvido por um dos casos anteriores.

Não existindo o cache, é necessário dispor de mecanismos que tratem eficientemente o caso de leituras e escrita repetidas em um mesmo endereço.

Um dos problemas fundamentais a ser resolvido por uma arquitetura com acesso e execução desacoplados é a sincronização de leituras e escritas em um mesmo endereço. Quando o processador envia uma palavra para ser escrita na memória é necessário verificar se existe um pedido de leitura desse endereço ainda pendente. Se existir, o novo valor gerado deve ser escrito também na posição correspondente da fila. Se o valor anterior já tinha chegado da memória, é descartado por essa operação; se a leitura ainda estiver pendente, o pedido deve ser marcado de tal forma que o valor vindo da memória seja desprezado quando chegar. Se a arquitetura já dispuser desse mecanismo, o algoritmo proposto funcionará correta e eficientemente, mesmo com cadeias que se sobreponham.

Para atingir o mesmo nível de desempenho do caso com cache "write-back" é necessário alterar a administração da fila de dados vindos da memória, não se removendo imediatamente o elemento do topo quando for usado, e preservando-o enquanto vierem pedidos de acesso ao mesmo endereço. O mecanismo descrito na figura 2 deve também ser incorporado no caminho de escrita.

Estes mecanismos, combinados com a lógica de acesso acima, têm o mesmo efeito de um cache "write-back" de uma posição.

4 – Considerações de custo

As instruções propostas são uma extensão simples direta das instruções de deslocamento já presentes em qualquer arquitetura. Em particular, arquiteturas que visem alto desempenho têm sempre um "barrel shifter" ou solução equivalente, e isto é perfeitamente factível mesmo em microprocessadores, com o nível atual de integração. Assim o custo de sua implementação em hardware é apenas um modesto aumento da lógica de controle necessária, e é certamente muito mais econômica que a implementação de um mecanismo de acesso desalinhado à memória, que exige a duplicação deste tipo de circuito em outro ponto do fluxo de dados.

O verdadeiro problema de custo em hardware tem a ver com os bits de índice, e é de sua solução que depende a conveniência ou não de sua adoção.

Esta proposta foi desenvolvida para o Projeto ***** [*], que é uma arquitetura com tags. Estes tags são utilizados para permitir execução fora de ordem e interrupções retardadas. Quando uma palavra contém dados válidos, há bits de tags disponíveis, de forma que neste caso seu custo é zero.

No caso mais geral, não há bits disponíveis para o armazenamento do índice. Adicionar 3 bits a cada um dos registradores do processador não compromete o desempenho da máquina, e é pouco se considerarmos que é um acréscimo de 3 a uma palavra de 64 bits. O problema se torna importante, entretanto, quando se sabe que é necessário salvar e restaurar registradores, e que esta operação deve envolver necessariamente os bits de índice. Pode-se tornar a memória 3 bits mais larga, mas certamente é um desperdício aumentar a memória somente para esta finalidade.

Outra alternativa consiste em utilizar duas palavras de memória para armazenar cada registrador a salvar, mas a influência sobre o desempenho pode ser bastante grave.

Registradores podem ser salvos individualmente ou em blocos. O primeiro caso ocorre quando se torna necessário salvar alguns registradores pela razão que em um certo instante o número de variáveis ativas ultrapassou o número de registradores disponíveis. Nas arquiteturas de projeto mais recente, com pelo menos 32 registradores, este é um caso bastante raro. Registradores são salvos em bloco no caso de mudança de contexto por interrupção, mas este também é um evento relativamente raro. O caso mais freqüente é o de chamada e retorno de procedimentos, onde registradores do contexto antigo devem ser copiados para a memória para que variáveis do novo contexto possam ser alocadas. Esta operação pode ser realizada em bloco, em máquinas que usam janelas de registradores, como as que seguem a arquitetura SPARC, ou individualmente.

Nas arquiteturas que utilizam janelas, é bastante fácil resolver o problema dos bits de índice: a cada bloco de registradores (no caso de palavras de 64 bits, tem-se blocos de 16 registradores) faz-se corresponder uma palavra reunindo todos os índices correspondentes a essas palavras. Salvar essa palavra a mais é um overhead muito pequeno, de modo que existe uma solução simples e natural para o problema de salvamento e restauração dos índices.

Nas arquiteturas que exigem salvamento e restauração de registradores individuais, a aplicabilidade da presente proposta depende da existência de características particulares, como os tags na arquitetura apresentada acima, para sua viabilização.

5 – Operações com subpalavras não alinhadas

Quando o único tipo de subpalavra importante é o byte, não há problema de falta de alinhamento, pelo menos enquanto o endereço for expresso em bytes. As operações com subpalavras não alinhadas são importantes quando subpalavras maiores forem requeridas.

Caso a arquitetura suporte operações com pares de registradores operando como registradores de precisão dupla, as operações com palavras desalinhadas implicam apenas numa extensão trivial da funcionalidade das operações ExtraiaX e InsiraX, para que possam trabalhar com operandos de precisão dupla.

Entretanto, não é usual dispor deste tipo de operandos, quando o tamanho da palavra é de 64 bits. A razão fundamental é que acessos com palavras duplas podem causar problemas de violação de presença quando a primeira palavra do par for a última de uma página; por esta razão, é de se esperar que futuras arquiteturas também evitem este tipo de acesso. Outra razão é que os registradores de deslocamento teriam de possuir 128 bits, ocupando muito espaço para um uso bastante infreqüente.

Assim, torna-se necessário dispor de instruções independentes para tratar este caso. Utilizando-se instruções independentes, os mecanismos usuais de proteção de acesso serão aplicados a cada um dos acessos, sem os problemas de limite de página, e cada operação individual será realizada sobre os registradores de 64 bits.

A instrução ExtraiaX deve ter sua funcionalidade estendida para extrair apenas a parte inferior de uma subpalavra quando for o caso, isto é, deve-se considerar que os bits que se encontrem fora da palavra de onde se faz a extração sejam todos extraídos como zeros.

Além disto, torna-se necessário complementá-la por uma instrução que realize a extração somente da parte superior da subpalavra. A funcionalidade da nova instrução ExtraiSuperX Rb, Rs, Rd é

if (Z = 8 * I(Rs) + X - 64) > 0 then Rd = Rb AND Rs [Z-1 : Z] << (X-Z)
else Rd = Rb

Analogamente, define-se a instrução InsereSuperX Rb, Rs, Rd é

if (Z = 8 * I(Rb) + X - 64) > 0

then Rd = Rb AND ((ONES AND 0 [Z - 1: Z]) OR (Rs [X - 1 : Z]))
else Rd = Rb

Com estas instruções, a movimentação de uma cadeia de subpalavras apontada pelo endereço contido em R1 para o endereço contido em R4, sem restrições de alinhamento, seria realizada pelo seguinte código:

```
Copia:   Carregue      (R1),R2 % contém a 1a palavra da origem
         Some          R1, tamanho(X), R3
         Carregue      (R3),R3 % contém a 2a palavra da origem
         ExtraiX       R2, R8
         ExtraiSuperX  R8, R3, R8 % R8 contém a subpalavra
         Carregue      (R4),R5 % contém a 1a palavra do destino
         Some          R4, tamanho(X), R7
         Carregue      (R7),R3 % contém a 2a palavra do destino
         InsereX       R8, R5
         InsereSuperX  R8, R3, R3
         Guarde        (R4), R5
         Guarde        (R7), R3
         DesvieZ       Copiado
         Incremente    R1, tamanho(X)
         Incremente    R4, tamanho(X)
         Desvie        Copia
Copiado:...
```

Embora a aplicação de palavras parciais arbitrárias seja muito mais restrita, uma vez que estas operações tenham sido implementadas, é mais ou menos imediato extendê-las para atender a este caso. Evidentemente, a menos que o projeto da Unidade aritmética e lógica já tenha previsto estas operações desde o início de seu projeto, esta adaptação poderá ser um tanto complexa. Um aspecto arquitetônico que deve ser previsto é que a unidade de endereçamento deve ser o bit, e não o byte. Este tipo de endereçamento já foi usado, por exemplo, na linha Burroughs B1000 [Rei72]. Com 64 bits para conter o endereço, esta especificação de endereçamento a bit não deve apresentar problemas.

As instruções ExtraiaX e InsereX devem ter sua funcionalidade estendida para trabalhar com bits e não com bytes, o que é trivial. A instrução de extração pode ainda ser estendida para obter o valor de X de um registrador adicional, permitindo a apenas uma instrução abranger todos os casos. Infelizmente, o mesmo não ocorre com a instrução de inserção, que exigirá 63 variantes, uma vez que já utiliza dados fornecidos por dois registradores, e a maior parte das arquiteturas não tem condições de fornecer mais de dois operandos por instrução.

6 – Conclusões

A implementação proposta para operações com subpalavras é bastante geral, não requer hardware adicional para máquinas com caches "write-back" e pode ser implementada com um mínimo de hardware adicional em máquinas de outros tipos. O conceito pode ser facilmente estendido para acessos desalinhados ou com palavras parciais de tamanho arbitrário, caso isto seja necessário.

Suas principais vantagens são:

- É um mecanismo uniforme, capaz de operar com qualquer tipo de subpalavras - bytes, dibytes e tetrabytes - de uma maneira uniforme e consistente. A operação com subpalavras maiores que o byte é importante tanto para a futura utilização de textos codificados em UNICODE como para a utilização aritmética de valores codificados com menos de 64 bits;
- Não requer que cadeias estejam alinhadas, mas não impede que compiladores utilizem alternativas otimizadas se existir a informação de alinhamento;
- Dependendo das outras características já possuídas pela arquitetura, pode ser implementada com um custo de hardware muito reduzido.

7 – Bibliografia

[Cam92] Campos, Geraldo L. "Asynchronous Polycyclic Architecture". Parallel Processing: CONPAR 92-VAPP V (Lecture Notes in Computer Science, vol 634, 387-398, Springer-Verlag, setembro de 1992

[DEC92] Digital Equipment Corporation, "Alpha architecture handbook", 1992

[Hen90] Hennessy, J. L. e Patterson, D. A. "Computer Architecture: A Quantitative Approach", Morgan Kaufmann Publishers, San Mateo, CA, 1990

[Rei72] Reigel, E. W.,Faber , U. e Fisher, D. A.,"The Interpreter - a microprogrammable building block system", Proc. AFIPS Spring Joint Computer Conf, 40, 705-723

[Smi84] Smith, J. E.,"Decoupled Access/Execute Architecture Computer Architectures", ACM Trans. Computer Systems 2(4):298-308, Nov 1984