

Capítulo 11

- [Organização do Processador](#)
- [Organização dos registos](#)
 - [Registos visíveis ao utilizador](#)
 - [Registos de Controlo e de Estado](#)
 - [Exemplo de Organização de Registos de Micro-processador](#)
- [O ciclo de Instruções](#)
 - [Ciclo Indirecto](#)
 - [Fluxo de Dados](#)
- [Linha de Encadeamento de Instruções](#)
 - [Estratégia de Encadeamento](#)
 - [Modo de proceder com derivações](#)
 - [Múltiplos fluxos](#)
 - [Pré-extracção do alvo da derivação](#)
 - [Tampão de laço](#)
 - [Prognóstico de Derivação](#)
 - [Linha de encadeamento do Intel 80846](#)
- [O Processador Pentium](#)
 - [Organização de registos](#)
 - [Registo EFLAGS](#)
 - [Registos de Controlo](#)
 - [Processamento de Interrupções](#)
 - [Excepções e Interrupções](#)
 - [Tabela de Vector de Interrupções](#)
 - [Tratamento de Interrupções](#)

Processador Estrutura e Função

Este capítulo discute os aspectos do processador, ainda, não cobertos na parte III e estabelece a plataforma para a discussão das arquitecturas RISC e super-escalar nos capítulos 12 e 13.

O capítulo começa com um resumo da organização do processador. Os registos que constituem a memória interna do processador são então analisados. Ficamos, então, numa posição para regressar à discussão (iniciada na secção 3.2) do ciclo de instrução. Uma descrição do ciclo de instrução e duma técnica comum chamada encadeamento de instruções completa a descrição. O capítulo conclui com o exame de alguns aspectos adicionais da organização do Pentium e do PowerPc.

Organização do Processador

Para compreender a organização do processador, consideremos os requisitos do processador, as coisas que tem de fazer:

- *Extrair Instruções:* O processador tem de ler instruções da memória.
- *Interpretar as Instruções:* As instruções têm de ser decodificadas para determinar as acções necessárias.
- *Extrair Dados:* A execução de uma instrução pode necessitar de ler dados da memória ou de um módulo de E/S.
- *Processar Dados:* A execução de uma instrução pode necessitar de efectuar alguma operação aritmética ou lógica sobre os dados.
- *Escrever Dados:* Os resultados de uma execução podem obrigar à escrita de dados na memória ou num módulo de E/S.

É necessário que fique claro que para poder ser capaz de fazer estas coisas o processador necessita de guardar, temporariamente, alguns dados. Tem de ter presente a posição da última instrução para que possa saber de onde extrair a próxima instrução. Tem de preservar, temporariamente, instruções e dados enquanto uma instrução está a ser executada. Por outras palavras, o processador necessita de uma pequena memória interna.

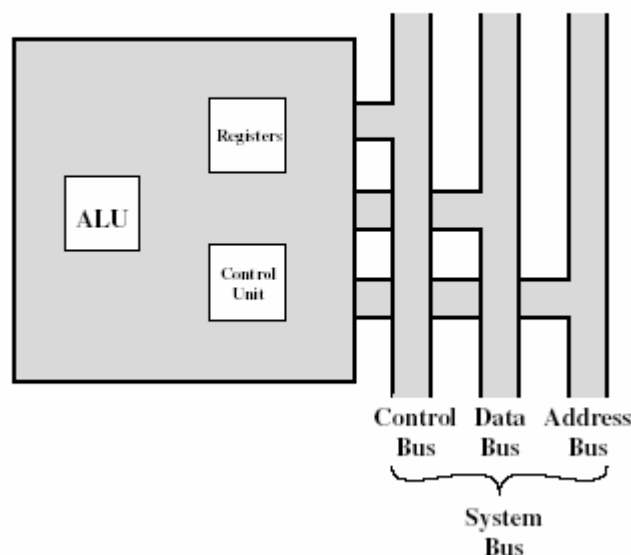


Figura 11.1: O processador com o barramento de sistema

A figura 11-1 é uma vista simplificada do processador, indicando as suas ligações ao resto do sistema através do barramento de sistema.

Uma interface semelhante seria necessário para cada uma das estruturas de interconexão apresentadas no capítulo 3. O leitor recordará que os componentes mais importantes do processador são a *unidade lógica e aritmética* (ALU) e a *unidade de controlo* (UC). A ALU é quem efectivamente

efectua os cálculos e o processamento dos dados. A unidade de controlo dirige o movimento de dados e de instruções de e para o processador e controla a operação da ALU. A figura mostra, além do mais, uma memória interna mínima constituída por um jogo de posições de armazenamento chamados *registos*.

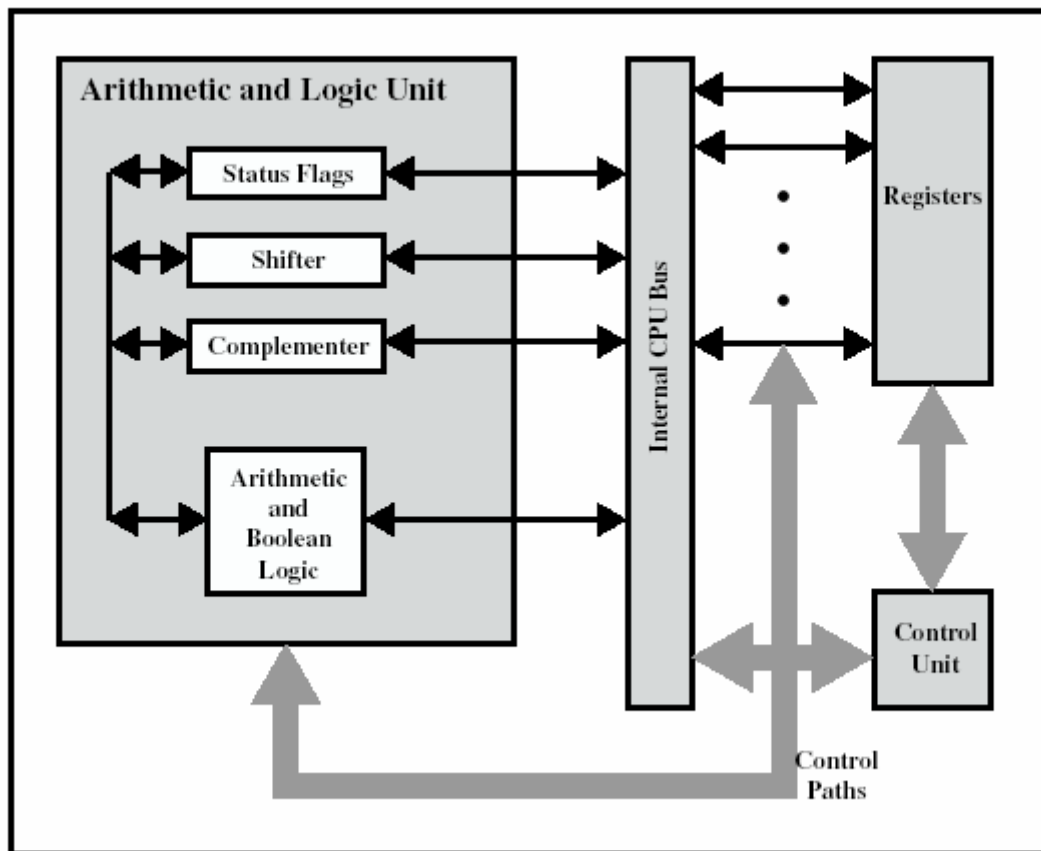


Figura 11.2: Estrutura interna do processador.

A figura 11-2 é uma vista ligeiramente mais detalhada do processador. Estão indicados os caminhos para a transferência de dados e para a lógica de controlo, incluindo um elemento designado por *barramento interno do processador*. Este elemento é necessário para mover os dados entre os vários registos e a ALU, uma vez que a ALU, de facto, opera exclusivamente sobre dados na memória interna do processador. A figura também mostra os elementos típicos básicos da ALU. É de notar a similaridade entre a estrutura interna do computador como um todo e a estrutura interna do processador. Em ambos os casos, há uma pequena colecção de elementos mais importantes (computador: processador, E/S, memória; processador: unidade de controlo, ALU, registos) ligados através de caminhos de dados.

Organização dos registos

Tal como foi discutido no capítulo 4, um sistema de computação usa uma hierarquia de memória. Nos níveis mais elevados da hierarquia, a memória é mais rápida, mais pequena e mais dispendiosa (por bit). Dentro do processador, há um conjunto de registos que funcionam a um nível de memória superior, na hierarquia, à memória principal e à *cache*. Os registos no processador servem para duas funções:

- *Registos Visíveis ao Utilizador*: Estes permitem ao programador de linguagens máquina e de montagem minimizar as referências à memória principal através da optimização do uso dos registos.
- *Registos de Controlo e de Estado*: Estes são usados pela unidade de controlo para dirigir as operações do processador e por programas privilegiadas do sistema de operação para controlar a execução de programas.

Não há uma separação nítida dos registos pertencentes a estas duas categorias. Por exemplo em algumas máquinas o contador de programa é visível ao programador (e.g. VAX) mas em muitos outros não é. Contudo, para efeito da discussão que segue iremos usar estas categorias.

Registos visíveis ao utilizador

Um registo visível ao utilizador é aquele que pode ser referenciado através da linguagem máquina que o processador executa. Virtualmente todos os projectistas de processadores actuais oferecem um certo número de registos visíveis ao utilizador, por oposição a um único acumulador. Podemos categorizá-los nas seguintes categorias:

- Uso Geral
- Dados
- Endereço
- Códigos de Condição

Os *registos de uso-geral* podem ser dedicados pelo programador a uma variedade de funções. Algumas vezes a sua utilização no jogo de instruções é ortogonal à operação. Isto corresponde a uma verdadeira utilização de uso geral. Muitas vezes, porém, há restrições. Por exemplo, podem existir registos dedicados para operações de vírgula flutuante.

Em alguns casos, os registos de uso geral podem ser usados para funções de endereçamento (e.g. indirecto por registo, deslocamento). Em outros casos, há uma nítida, ou parcial, separação entre registos de dados e de endereçamento. Os *registos de dados* devem ser usados, apenas, para guardar dados e não deverão ser usados para o cálculo do endereço de um operando. Os *registos de endereços* podem eles próprios ser, de alguma forma, de uso geral, ou podem ser dedicados a um modo de endereçamento específico. Os exemplos incluem:

- *Apontadores de Segmento*: Numa máquina com endereços segmentados (ver secção 7.3), um registo de segmento mantém o endereço base do segmento. Podem existir múltiplos registos: por exemplo, um para o sistema de operação e um para o processo corrente.
- *Registos de Índice*: Estes são usados para endereçamento por indexação e podem ser auto-indexados.
- *Apontador de Pilha*: Se existir endereçamento da pilha visível ao utilizador, então, tipicamente a pilha está na memória e há um registo dedicado que aponta para o topo da pilha. Isto permite endereçamento implícito; isto é, pôr, tirar, e outras instruções de pilha não necessitam de conter um operando explícito de pilha.

Há diversos tópicos de projecto que deverão ser aqui considerados. Um tópico importante é se os registos deverão ser totalmente de uso geral ou deverão ser de uso especializado. Já, antes, em capítulos precedentes, tocámos neste assunto, uma vez que este afecta o projecto de jogo de instruções. Com o uso de registos especializados pode, em geral, estar implícito no código de operação o tipo de registo a que uma certa especificação de operando se refere. O especificação de operando tem apenas de identificar um registo de entre um jogo de registos especializados em vez de um registo de entre a totalidade dos registos, desta forma, poupando bits. Noutra perspectiva, esta especialização limita a flexibilidade do programador. Para este tópico de projecto não existe solução nem final nem melhor, mas, como foi mencionado, a tendência parece ser favorável ao uso de registos especializados.

Um tópico de projecto é o número de registos, a oferecer tanto de uso geral, como de dados mais endereços. De novo, isto afecta o projecto de jogo de instruções, uma vez que mais registos obriga a mais bits de especificação de operandos. Tal como discutimos anteriormente, qualquer coisa entre 8 e 32 registos pode ser óptimo. Menos registos resulta em mais referências à memória; mais registos não reduz significativamente as referências à memória (e.g. ver). Contudo, uma nova abordagem que encontra vantagens no uso de centenas de registos manifesta-se em alguns sistemas RISC e é discutida no capítulo 12.

Finalmente, há o tópico de comprimento de registo. Os registos que têm de conter endereços têm, obviamente, de ser pelo menos suficientemente longos para conter o maior dos endereços. Os registos de dados devem ser capazes de guardar valores do maior número de tipos de dados. Algumas máquinas permitem que dois registos contíguos possam ser usados como um para guardar valores de duplo-tamanho.

Uma categoria final de registos, que são, pelo menos, parcialmente visíveis ao utilizador guarda *códigos de condição* (também referidos como *bandeiras*). Os códigos de condição são bits ajustados pelo *hardware* do processador como resultado das operações. Por exemplo, uma operação aritmética pode produzir um resultado positivo, negativo, zero ou excesso. Além do resultado, ele próprio, ser guardado num registo ou em memória, é, também, feito um ajuste

do código de condição. O código pode ser subsequentemente testado como parte da operação de uma derivação condicional.

Os bits do código de condição são colecionados em um ou mais registos. Habitualmente, fazem parte de um registo de controlo. Geralmente, as instruções máquina permitem que estes bits sejam lidos através de referências implícitas, mas não podem ser alterados pelo programador.

Em algumas máquinas, uma chamada a uma sub-rotina resulta na salvaguarda automática de todos os registos visíveis que irão ser restaurados no retorno. A salvaguarda e o restauro é efectuado pelo processador como fazendo parte da execução das instruções de chamada e de retorno. Isto permite a cada sub-rotina usar de forma independente os registos visíveis do utilizador. Em outras máquinas, é da responsabilidade do programador salvarguardar o conteúdo dos registos visíveis ao utilizador relevantes imediatamente antes de uma chamada a sub-rotina, através da inclusão de instruções para esse fim.

Registos de Controlo e de Estado

Há uma variedade de registos de processador que são empregues para controlar a operação do processador. A maior parte destes, na maioria das máquinas, não são visíveis ao utilizador. Alguns deles podem ser visíveis para as instruções máquina executadas em modo de controlo ou de sistema de exploração.

Claros, máquinas diferentes terão diferentes organizações de registos e irão usar diferentes terminologias. Listamos aqui uma lista, razoavelmente, completa de tipos de registos, juntamente com uma descrição breve.

Quatro registos são essenciais para execução de instruções:

- *Contador de Programa (PC)*: Contém o endereço de uma instrução a ser extraída
- *Registo de Instrução (IR)*: Contém a instrução extraída mais recentemente.
- *Registo de Endereçamento de Memória (MAR)*: Contém o endereço de uma localização na memória.
- *Registo Tampão de Memória (MBR)*: Contém uma palavra de dados para ser escrita para memória ou a palavra mais recentemente lida.

O contador de programa contém o endereço de uma instrução. Tipicamente, o contador de programa é actualizado pelo processador depois de cada instrução de forma a apontar sempre para a próxima instrução a ser executada. Uma derivação ou instrução de omissão também irá modificar o conteúdo do PC. A instrução extraída é carregada num registo de instrução, onde o código de operação e a especificação de operando é analisada. As trocas de dados com a memória são através do MAR e MBR. Num sistema organizado em barramento, o MAR liga-se directamente ao barramento de endereços e o MBR liga directamente ao barramento de dados. Os registos visíveis do utilizador, por sua vez, trocam dados com o MBR.

Os quatro registos que acabamos de referir são usados para a movimentação de dados entre o processador e a memória. Dentro do processador, os dados tem de ser apresentados ao processador para serem processados. A ALU pode ter acesso directo ao MBR e aos registos visíveis do utilizador. Em alternativa, podem existir registos tampão adicionais próximos da ALU; estes registos servem como registos de entrada de saída para a ALU e trocam dados com o MBR e os registos visíveis do utilizador.

Todos os projectos de processadores incluem um registo ou jogo de registos, muitas vezes conhecidos como *palavra de estado do programa* (PSW) que contém informação de *status*. O PSW contém, tipicamente, códigos de condição mais informação de *status*. Os campos comuns ou bandeiras incluem o seguinte:

- *Sinal*: Contém o bit de sinal resultante da última operação aritmética
- *Zero*: Ajustada a 1 quando o resultado é 0
- *Transporte*: Ajustado a 1 se o resultado da operação inclui transporte (adição e subtração) do bit mais significativo. Usado em operações aritméticas de multi-palavras.
- *Igualdade*: Ajustado a 1 se o resultado de uma comparação lógica é a igualdade.
- *Excesso*: Usado para indicar transbordo aritmético.
- *Habilitação/Inibição de Interrupções*: Usado para autorizar ou impedir interrupções.
- *Supervisor*: Indica se o processador está a executar em modo supervisor ou modo utilizador. Certas instruções privilegiadas só podem ser executadas em modo supervisor e o acesso a certas áreas de memória só pode ser feito em modo supervisor.

Há um certo número de outros registos relacionados com o *status* e o controlo que podem ser encontrados em projectos particulares de processadores. A juntar ao PSW, pode existir um apontador para um bloco de memória contendo informação de *status* adicional (e.g. blocos de controlo de processos). Em máquinas que utilizam interrupções vectorizadas, pode existir um registo de vector de interrupção. Se uma pilha é usada para implementar certas funções (e.g. chamada a sub-rotina) então é necessário um apontador de pilha. Um apontador para tabela de páginas é usado com um sistema de memória virtual. Finalmente, podem ser usados registos para controlo de operações de E/S.

Um certo número de factores influencia o projecto de organização dos registos de controlo e de *status*. Um tópico chave é o suporte ao sistema de operação. Certos tipos de informação de controlo são de utilidade específica do sistema de operação. Se o projectista do processador tiver um conhecimento da funcionalidade do sistema de exploração a ser usado, então a organização de registos pode, em certa medida, ser ajustado ao sistema de exploração.

Um outro tópico de projecto é a alocação de informação de controlo entre os registos e a memória. É habitual dedicar as primeiras (mais baixos) centenas ou milhares de palavras de memória para efeitos de controlo. O projectista tem de decidir quanta informação de controlo deve residir em registos e quanta

deve residir em memória. Surge então o habitual compromisso custo versus velocidade.

Exemplo de Organização de Registos de Micro-processador

É instrutivo examinar e comparar a organização de registos de sistemas comparáveis. Nesta secção, olhamos para três micro-processadores de 16-bits que foram projectados mais ou menos ao mesmo tempo: O Zilog Z8000 o Intel 8086 e o Motorola MC68000. A figura 11-3 mostra a organização de registos de cada um; registos puramente internos, tais como registos de endereçamento de memória não são apresentados.

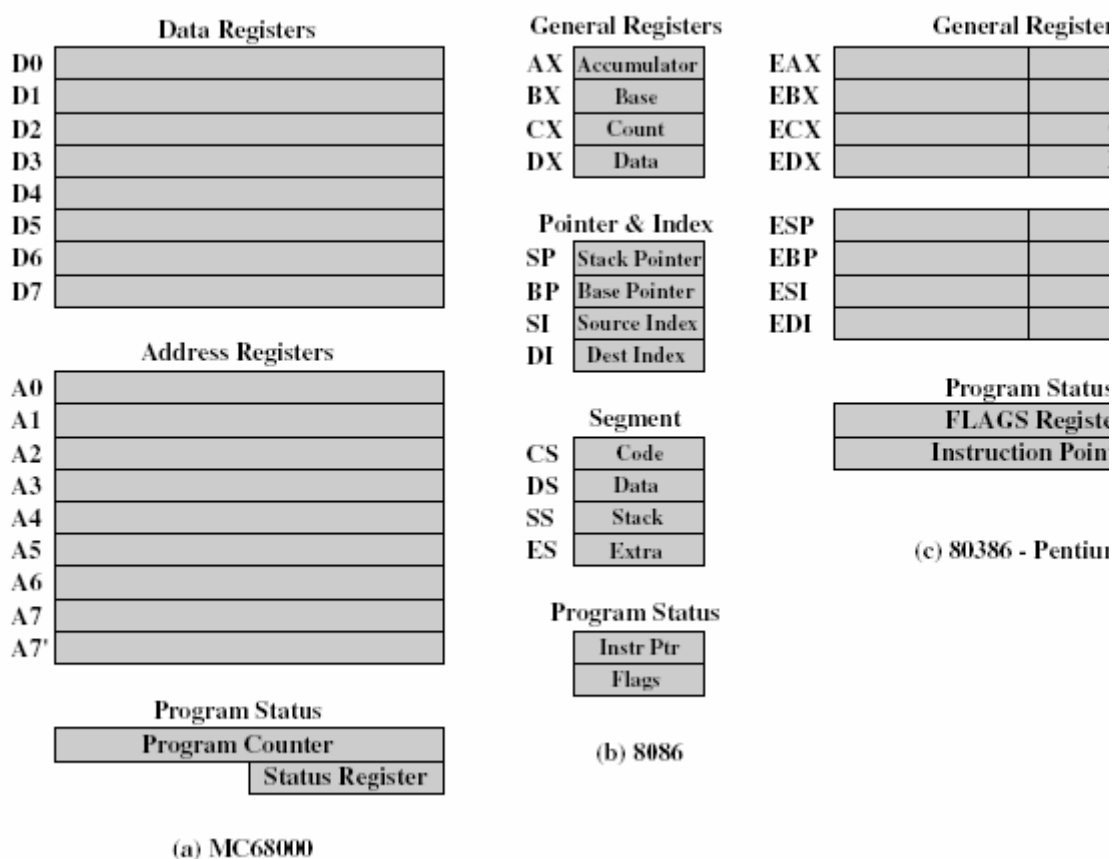


Figura 11.3: Organização dos registos de um microprocessador.

O Z8000 faz uso de 16 registos de 16-bits de uso-geral, que podem ser usados para dados, endereçamento e indexação. Os projectistas sentiram que era mais importante oferecer um jogo de registos regularizados em vez de poupar em bits de instrução através do uso de registos de uso-específico. Além disso, eles preferiram deixar ao programador a atribuição de funções aos registos, assumindo que poderiam existir diferentes arranjos para diferentes aplicações. Os registos podem ser usados para operações de 8-bits e de 32-bits. É usado um espaço de endereçamento segmentado (7-bits para número de segmento, 16 bits para deslocamento) sendo necessário dois registos para conter um

endereço simples. Dois dos registos são também usados como apontadores de pilha implícitos para modo de sistema e modo normal.

O Z8000 também inclui cinco registos relacionados com o *status* do programa. Dois registos contêm o contador de programa e dois o endereço da Área de *Status* do Programa em memória. Um registo de condições de 16-bits mantém vários bits de *status* e de controlo.

O Intel 8086 segue uma abordagem diferente à organização de registos. Todo o registo é de uso específico, embora alguns registos possam também ser usados como registos de uso-geral. O 8086 contém quatro registos de dados de 16-bits que são acessíveis numa base de 8 ou 16-bits e quatro apontadores de 16-bits e registos e índice. Os registos de dados podem ser utilizados como de uso-geral em algumas instruções. Em outras, os registos são usados implicitamente. Por exemplo uma instrução de multiplicação usa sempre o acumulador. Os quatro registos apontadores são também usados implicitamente num certo número de instruções; cada um contém um deslocamento de segmento. Há também quatro registos de segmentos de 16-bits. Três dos quatro registos de segmento são usados, de forma dedicada e implícita, para apontar para o segmento da instrução corrente (útil em instruções de derivação), um segmento que contém dados e um segmento que contém a pilha, respectivamente. Estes usos dedicados e implícitos suportam codificação compacta pagando o custo da redução em flexibilidade. O 8086 também inclui um apontador de instruções e um jogo de condições de 1-bit para *status* e controlo.

O Motorola MC68000 cai, de alguma maneira, entre as filosofias de projecto dos micro-processadores Zilog e Intel. O MC68000 parte os seus registos de 32-bits em oito registos de dados e nove registos de endereços. Os oito registos de dados são usados em primeiro lugar para o manuseamento de dados e são usados em endereçamento apenas como registos de índice. A largura dos registos permite operações de dados sobre 8-, 16- e 32-bits, determinadas pelo código de operação. Os registos de endereçamento contêm endereços de 32-bits (sem segmentação); dois destes registos são também usados como apontadores para a pilha, um para utilizadores e outro para o sistema de exploração, dependendo do modo de execução corrente. Ambos os registos têm o número 7, uma vez que só um pode ser usado de cada vez. O MC68000 também inclui um contador de programa de 32-bits e um registo de *status* de 16-bits.

Tal como os projectistas do Zilog, A equipa da Motorola pretendia uma jogo de instruções regular, sem registos de uso-especial. Uma preocupação com a eficiência do código levou-os a dividir os registos em dois componentes funcionais, poupando um bit em cada especificador de registo. Isto parece um compromisso razoável entre a generalização completa e a compactação do código.

Os termos desta comparação devem ser claros. Não há, ainda, nenhuma filosofia universalmente aceite acerca da melhor maneira de organizar os registos do processador. Tal como com o projecto de jogo de instruções e

muitos outros tópicos do projecto de processadores, é ainda uma questão de gosto e de opção.

Um segundo ponto instrutivo respeitante à organização de registos é ilustrado pela figura 11-4. Esta figura mostra a organização dos registos visíveis aos utilizadores para o Zilog 80000 e o Intel 80386 que são micro-processadores de 32-bits projectados como extensões ao Z8000 e ao 8086 respectivamente [11.1](#). Ambos os dois novos processadores usam, registos de 32-bits. Contudo, para manter a compatibilidade dos programas escritos para máquinas mais antigas, ambos os novos processadores retêm a organização original dos registos embebida na nova organização. Dadas estas restrições os arquitectos de processadores de 32-bits tiveram uma flexibilidade limitada no projecto de organização de registos.

Figura 11.4: Extensão da organização de registos em microprocessador de 32-bits.

O ciclo de Instruções

Na secção 3.2 descrevemos o ciclo de instruções do processador. A figura [11.5](#) repete uma das figuras usadas na descrição (fig. 3.9)..Para recordar, um ciclo de instruções inclui os seguintes sub-ciclos:

- *Extracção:* Ler a próxima instrução da memória para o processador.
- *Execução:* Interpretar o código de operação e efectuar a operação indicada.
- *Interrupção:* Se as interrupções estiverem autorizadas e tiver ocorrido uma interrupção, guardar o estado actual do processo e servir a interrupção

Estamos agora em posição para aumentar a elaboração do ciclo de instruções. Em primeiro lugar, temos de introduzir um sub-ciclo adicional, conhecido como ciclo indirecto.

Ciclo Indirecto

Vimos, no capítulo 10, que a execução de uma instrução podia envolver um ou mais operandos na memória, cada um dos quais necessitava de um acesso à memória. Além disso, se fosse usado endereçamento indirecto, então, eram necessários acessos adicionais à memória.

Figura 11.5: Ciclo de instruções com interrupções.

Podemos pensar na extracção de endereços indirectos como mais um sub-ciclo de instrução. O resultado é mostrado na figura 11-6. A linha principal de actividade consiste em actividades alternadas de extracção e execução de

instruções. Depois de ser extraída uma instrução é examinada para determinar se está envolvido endereçamento indirecto. Se assim for, os operandos requeridos são extraídos usando endereçamento indirecto.

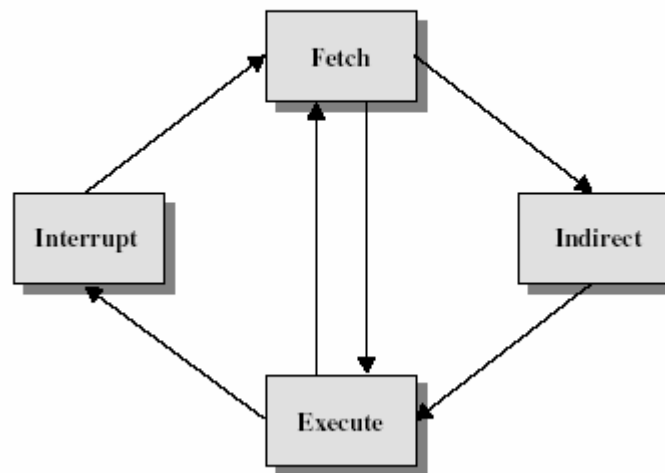


Figura 11.6: O ciclo de instruções.

A seguir a execução e a interrupção podem ser processadas antes da extracção da próxima instrução.

Uma outra forma de ver este processo é mostrado na figura 11-7, que é uma versão revista da figura 3.12. Isto ilustra mais correctamente a natureza do ciclo de instruções. Uma vez extraída uma instrução, os seus especificadores de operandos têm de ser identificados.

Cada operando de entrada em memória é então extraído e este processo pode necessitar de endereçamento indirecto. Operandos baseados em registos não necessitam de ser extraídos. Uma vez executado o código de operação um processo semelhante pode ser necessário para guardar o resultado na memória principal.

Figura 11.7: Diagrama de estados do ciclo de instruções.

Fluxo de Dados

Durante um ciclo de instruções, a sequência exacta de eventos depende do projecto do processador. Podemos contudo indicar, em termos gerais, o que tem de acontecer. Assumamos que o processador emprega um registo de endereços de memória (MAR), um registo de tampão de memória, um contador de programa (PC) e um registo de instruções (IR).

que usa endereçamento indirecto. Se assim for, é efectuado um *ciclo indirecto*. Tal como se mostra na figura 11-9 este é um ciclo simples.

Os n -bits mais à direita do MBR, que contêm a referência à memória, são transferidos para o MAR. Daí, a unidade de controlo faz um pedido leitura à memória, para colocar o endereço pretendido do operando no MBR.

Os ciclos de caminho indirecto e de extracção são simples e previsíveis. O *ciclo de instrução* assume várias formas uma vez que a forma depende de qual das várias instruções máquina está no IR. Este ciclo pode envolver a transferência de dados entre registos, a leitura ou escrita da memória ou da E/S e ou a chamada à ALU.

Tal como os ciclos de caminho indirectos e de extracção, o *ciclo de interrupção* é simples e previsível (figura 11-10).

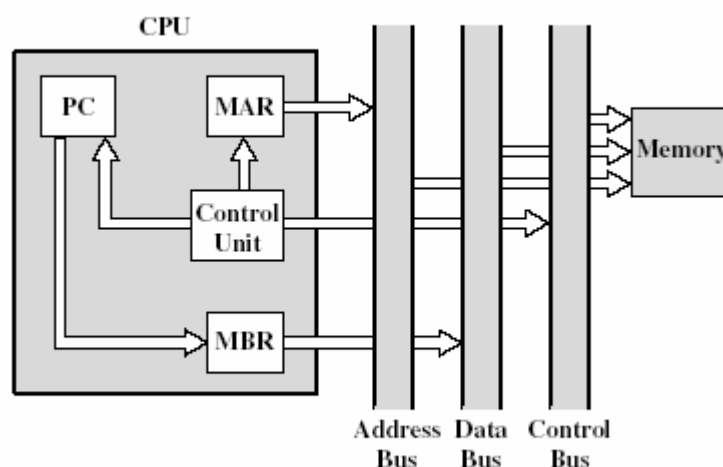


Figura 11.10: Fluxo de dados, ciclo de interrupção.

O conteúdo corrente do PC deve ser salvaguardado para que o processador possa voltar à actividade normal após a interrupção. Assim, o conteúdo do PC é transferido para o MBR para ser escrito na memória. A posição especial da memória reservada para este efeito é carregada no MAR da unidade de controlo. Pode ser, por exemplo, um apontador de pilha. O PC é carregado com o endereço da rotina de interrupção. Como resultado, o próximo ciclo de instrução irá começar com a extracção da instrução apropriada.

Linha de Encadeamento de Instruções

À medida que os sistemas de computação evoluem, maiores rendimentos podem ser alcançados aproveitando os melhoramentos da tecnologia, tais como, circuitos mais rápidos. Adicionalmente, os melhoramentos na organização do processador podem melhorar o rendimento. Já vimos alguns exemplos disto, tais como o uso de múltiplos registos, em vez de um simples acumulador e o uso da memória *cache*. Uma outra abordagem organizacional, que é muito comum, é o encadeamento de instruções.

Estratégia de Encadeamento

O encadeamento de instruções é muito semelhante ao uso de uma linha de montagem de uma fábrica. Uma linha de montagem aproveita o facto de que um produto passa por diversos estágios de produção. Ao optar por um processo de produção em linha de montagem os produtos em vários estágios podem ser trabalhados simultaneamente. Este processo é também referido como *linha de encadeamento*, porque numa linha de encadeamento, novas entradas são aceites numa extremidade antes que entradas aceites previamente possam aparecer à saída na outra extremidade.

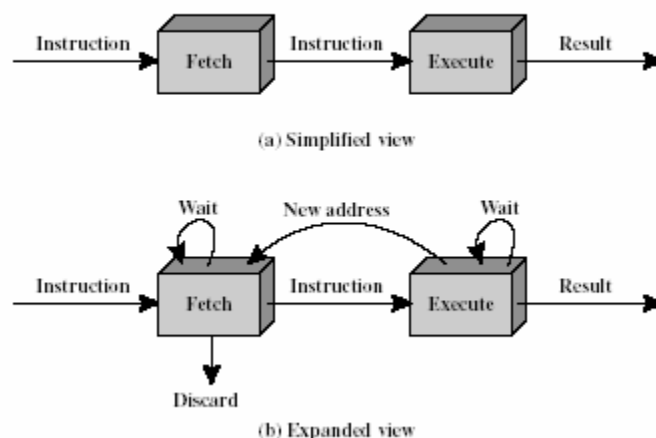


Figura 11.11: Linha de Encadeamento de instruções de dois estágios.

Para aplicar este conceito a execução de instruções, devemos de reconhecer que, de facto, uma instrução dispõe de um certo número de estágios. A figura 11-7, por exemplo, secciona o ciclo de instruções em 10 tarefas, que ocorrem em sequência. Claramente, deverá existir alguma oportunidade para encadeamento.

Numa abordagem simples consideremos a subdivisão do processamento de instruções em dois estágios; extracção da instrução e execução da instrução. Haverá momentos durante a execução de uma instrução em que não está a ser feito um acesso à memória. Estes momentos podem ser usados para extrair a próxima instrução em paralelo com a execução da instrução corrente. A figura 11-11(a) ilustra esta abordagem. O encadeamento tem dois estágios independentes. O primeiro estágio extrai a instrução e coloca-a num tampão. Quando o segundo estágio está livre, o primeiro estágio passa-lhe a instrução no tampão. Enquanto o segundo estágio está a executar a instrução, o primeiro estágio aproveita não importa qual ciclo de memória não usado para extrair e colocar no tampão a próxima instrução. Isto é chamado pré-extracção ou sobreposição da extracção.

Deverá ficar claro que este processo irá acelerar a execução de instruções. Se os estágios de extracção e de execução forem de igual duração, o ciclo de instrução passará a metade. Contudo, se olharmos mais de perto para esta linha de encadeamento (figura 11-11(b), veremos também que esta duplicação da taxa de execução é improvável por duas razões:

1. O tempo de execução é em geral mais longo que o tempo de extracção. A execução envolve a leitura e armazenamento de operandos e a efectivação de algumas operações. Assim, o estágio de extracção deverá de esperar algum tempo antes de esvaziar o seu tampão.
2. Uma instrução condicional de derivação torna desconhecido o endereço da próxima instrução a ser extraída. Assim, o estágio de extracção deverá esperar até que receba o endereço da próxima instrução do estágio de execução. O estágio de execução poderá ter de esperar enquanto é extraída a próxima instrução.

O tempo perdido pela segunda razão pode ser reduzido por previsão. Uma regra simples é a seguinte: Quando uma instrução de derivação condicional passa do estágio de extracção para o estágio de execução, o estágio de extracção extrai a próxima instrução da memória. Então se não houver derivação, não há perda de tempo. Se houver derivação a instrução extraída tem de ser descartada e extraída uma nova instrução.

Enquanto estes factores reduzem a potencial eficácia do encadeamento de dois estágios, produz-se algum ganho. Para obter maiores ganhos, o encadeamento deverá ter mais estágios. Consideremos a seguinte decomposição do processamento de instruções.

- *Extrair Instrução (FI)*: Ler a próxima instrução esperada para um tampão.
- *Descodificar Instrução (DI)* : determinar o código de operação e os especificadores de operandos.
- *Calcular Operandos (CO)* : Calcular o endereço efectivo de cada operando. Isto pode envolver deslocamentos, caminhos directos e indirectos de registo, ou outras formas de cálculo de endereços.
- *Extrair Operandos (FO)*: Extrair cada operando da memória. Os operandos em registos não necessitam de extracção.

- *Executar Instrução (EI)*: Efectuar a operação indicada e salvar o resultado, se existir, na posição de destino especificada pelo operando.
- *Escrever Operando (WO)*: Salvar o resultado na memória.

Com esta decomposição, os vários estágios serão aproximadamente da mesma duração. Para efeitos de exemplificação, assumamos que possuem igual duração. Então, a figura 11-2 mostra que um encadeamento de seis-estágios de encadeamento pode reduzir o tempo de execução de 9 instruções de 54 unidades de tempo para 14 unidades de tempo.

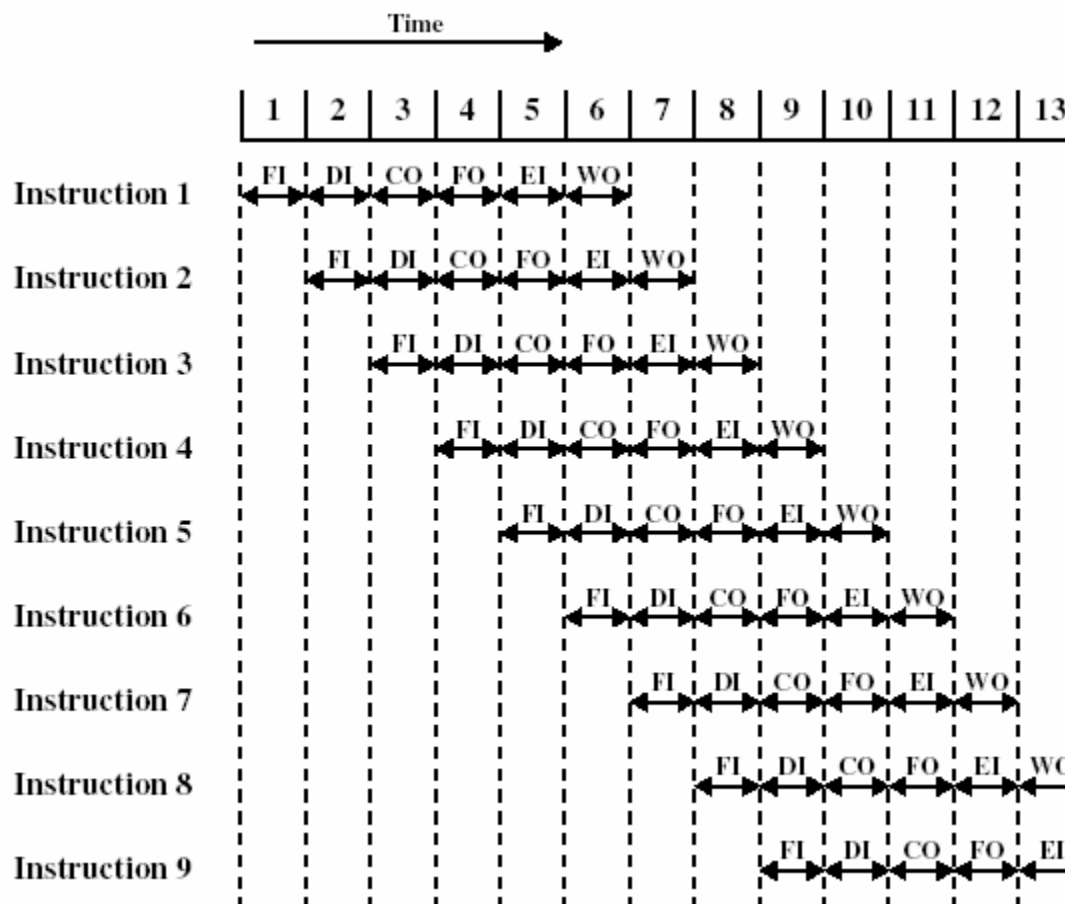


Figura 11.12: Diagrama temporal para uma operação de encadeamento de instruções.

Vários comentários: O diagrama assume que cada instrução atravessa todos os seis estágios da linha de encadeamento. Isto não será sempre o caso. Por exemplo, uma instrução de carregar não necessita do estágio WO. Contudo para simplificar o *hardware* do encadeamento, a temporização é feita assumindo que cada instrução necessita dos seis estágios. O diagrama assume, também, que todos os estágios podem ser efectuados em paralelo. Em particular, é assumido que não existem conflitos de memória. Por exemplo, os estágios FI, FO e WO envolvem o acesso à memória. O diagrama implica que todos estes acessos podem ocorrer simultaneamente. A maioria dos sistemas de memória não o permitiria. Assim, o valor pretendido pode estar

na *cache*, ou os estágios FO ou WO podem ser nulos. Pelo que, a maior parte do tempo, os conflitos de memória não irão atrasar o encadeamento.

Vários outros factores impõem limitações ao melhoramento do rendimento. Se os seis estágios não tiverem igual duração, haverá alguma espera em vários estágios de encadeamento, tal como antes foi discutido para o encadeamento de dois estágios. Uma outra dificuldade é a instrução de derivação condicional que pode invalidar a extracção de várias instruções. Um evento semelhante, não previsível, é uma interrupção. A figura 11-13 ilustra os efeitos de uma derivação condicional, usando o mesmo programa da figura 11-12. Assuma que a instrução 3 é uma derivação condicional para a instrução 15.

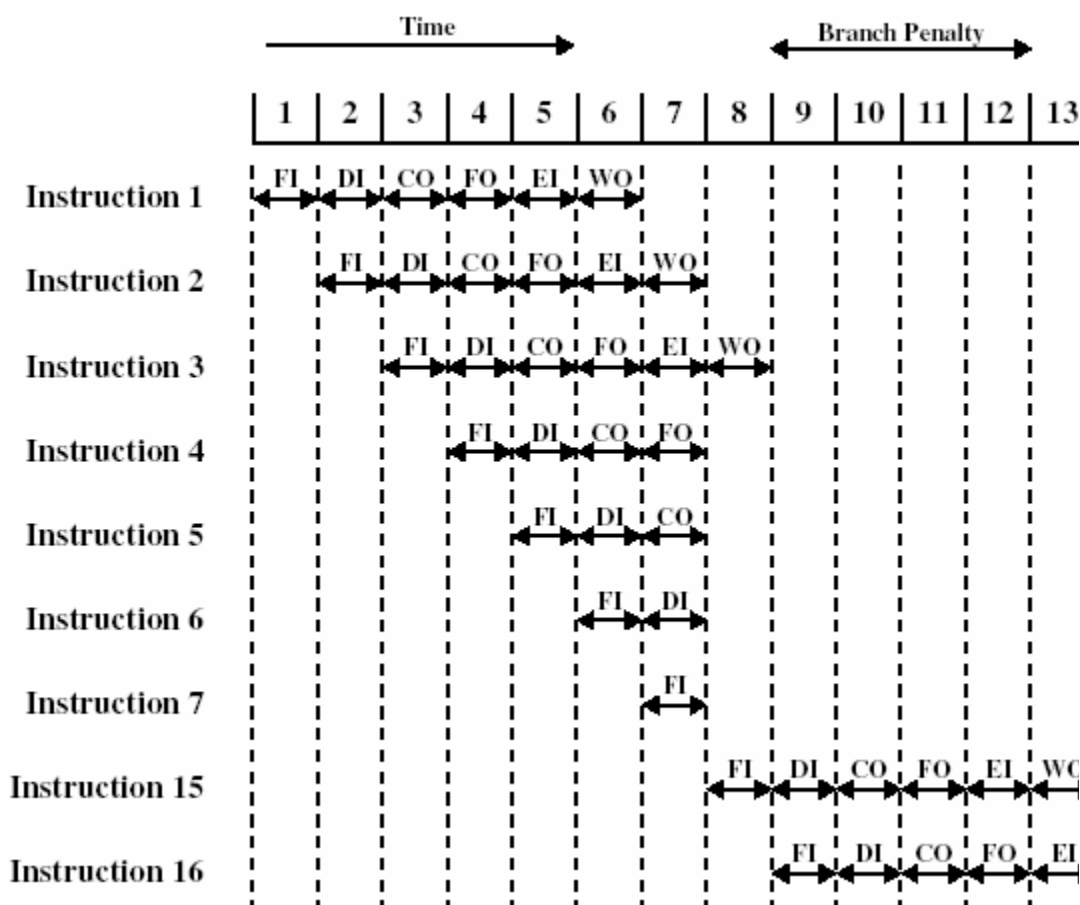


Figura 11.13: O efeito de uma derivação condicional para uma operação de encadeamento de instruções.

Até que a instrução seja executada, não há forma de saber qual a instrução que vem a seguir. O encadeamento, neste exemplo, carrega simplesmente a próxima instrução na sequência (instrução 4) e prossegue. Na figura 11-12, não há derivação e atingimos o rendimento máximo que advém do melhoramento. Na figura 11-13 há derivação. Isto não é determinado antes do fim da unidade de tempo 7. Neste ponto, a linha de encadeamento deve ser esvaziada das instruções que não são úteis. Durante a unidade de tempo 8, a instrução 15 entra na linha de encadeamento. Nenhuma instrução se completa durante as

unidades de tempo de 9, 10, 11 e 12; isto é a penalização do rendimento em que se incorre porque não podemos antecipar a derivação. A figura [11.14](#) indica a lógica necessária para que o encadeamento tome em consideração derivações e interrupções.

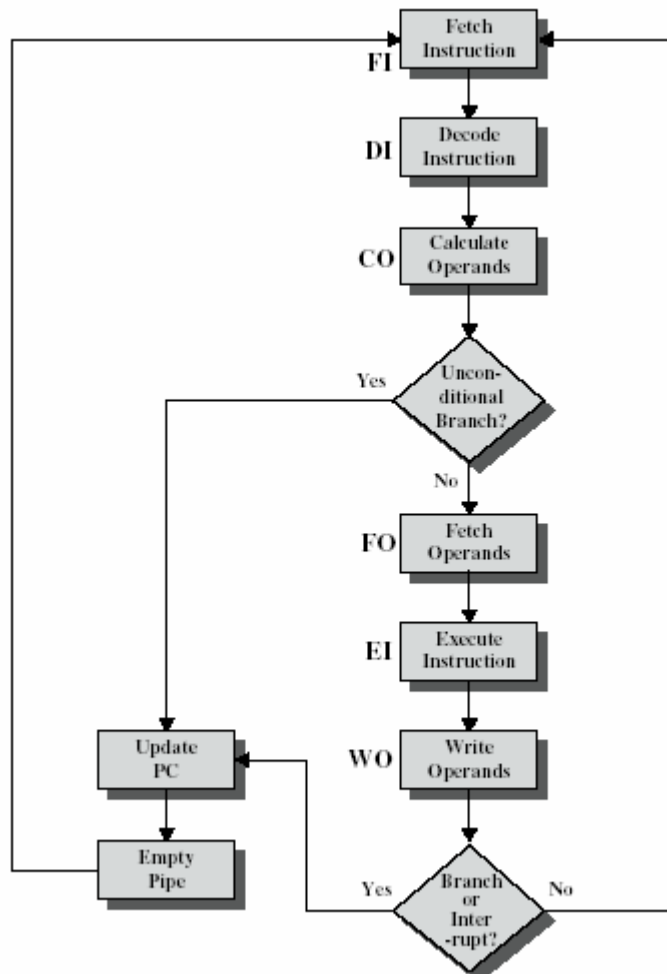


Figura 11.14: Encadeamento de instruções num processador de seis-estágios.

Surge outro problema que não apareceu na organização simples de dois-estágios. O estágio CO pode depender do conteúdo de um registo que podia ter sido alterado por uma instrução anterior que ainda está na linha de encadeamento. Outros conflitos de memória e de registos podem ocorrer. O sistema tem de conter lógica para entrar em consideração com este tipo de conflitos.

Da discussão precedente, pode parecer que quanto maior o número de estágios na linha de encadeamento, maior é a taxa de execução. Alguns dos projectistas do IBM/360 realçaram dois factores que frustam este aparente padrão simples para o projecto de alto-rendimento e que, hoje, se mantém verdadeiro:

1. Em cada estágio da linha de encadeamento, há alguma sobrecarga envolvida na movimentação de dados de tampão para tampão e na efectivação de várias funções de preparação e de entrega. Esta

sobrecarga pode alongar de forma apreciável o tempo total de execução de uma instrução simples. Isto é significativo quando as instruções sequenciais são logicamente dependentes, quer pelo uso intensivo de derivações quer através das dependências nos acessos à memória.

2. A quantidade de lógica de controlo necessária para manipular as dependências da memória e de registos e para otimizar o uso de linha de encadeamento, aumenta muitíssimo com o número de estágios. Isto pode conduzir a uma situação onde a lógica de entradas entre estágios é muito mais complexa do que os estágios que estão a ser controlados.

O encadeamento de instruções é uma técnica poderosa para melhorar o rendimento mas necessita de um projecto cuidadoso, com razoável complexidade, para alcançar resultados óptimos.

Modo de proceder com derivações

Um dos maiores problemas no projecto de linha de encadeamento de instruções é assegurar um fluxo constante de instruções nos estágios iniciais da linha de encadeamento. O primeiro impedimento, tal como vimos, é a instrução de derivação condicional. Até que a instrução seja efectivamente executada, é impossível determinar se se segue ou não a derivação.

Uma variedade de abordagens tem vindo a ser tomadas para lidar com derivações condicionais:

- Múltiplos fluxos (Multiple Stream)
- Pré-extracção do alvo do endereço (Prefetch Branch Target)
- Tampão para laços (Loop Buffer)
- Prognóstico de derivação (Branch Prediction)
- Derivação atrasada (Delayed Branch)

Múltiplos fluxos

Uma linha de encadeamento simples é penalizado por uma instrução de derivação porque tem de escolher de entre duas instruções a próxima a extrair e pode fazer a escolha errada. Uma abordagem de força bruta é replicar as porções iniciais da linha de encadeamento e permitir extrair ambas as instruções, usando duas linhas de encadeamento. Com esta abordagem há vários problemas:

- Com múltiplas linhas de encadeamento há atrasos de contenção nos acesso aos registos e à memória.
- Instruções adicionais de derivação podem entrar na linha de encadeamento (qualquer uma) antes da decisão de derivação inicial ter sido resolvida. Cada uma daquelas instruções requer uma linha de encadeamento adicional.

Apesar destes inconvenientes, esta estratégia pode melhorar o rendimento. Exemplos de máquinas com duas ou mais linhas de encadeamento são o IBM 370/168 e o IBM 3033.

Pré-extracção do alvo da derivação

Quando é reconhecida uma derivação condicional, o alvo da derivação é pré-extraído, conjuntamente com a instrução a seguir à derivação. O alvo é, então, salvaguardado até que seja executada a instrução de derivação. Se se segue a derivação, o alvo já tinha sido pré-extraído.

O IBM 360/91 usa esta abordagem.

Tampão de laço

Um tampão de laço é uma pequena memória, de muito elevada velocidade, mantida pelo estágio de pré-extracção da linha de encadeamento, contendo, uma sequência, das → mais recentes instruções extraídas. Se for para seguir a derivação, o *hardware* começa por verificar se o alvo da derivação está no tampão. Se assim for, a próxima instrução é extraída do tampão. O tampão de laço tem três vantagens:

1. Quando se usa pré-extracção, o tampão de laço conterá algumas instruções sequencialmente à frente do endereço de extracção da instrução corrente. Assim, as instruções extraídas em sequência estarão disponíveis sem os tempos habituais de acesso à memória.
2. Quando ocorre uma derivação para um endereço apenas algumas posições adiante do endereço da instrução de derivação, o alvo já irá estar no tampão. Isto é útil no caso da muito habitual ocorrência de sequências SE-ENTÃO e SE-ENTÃO-SENÃO.
3. Esta estratégia é particularmente bem indicada para lidar com laços, ou iterações, daí o nome de tampão de laço. Se o tampão de laço for suficientemente grande para conter todas as instruções de um laço, então, essas instruções só necessitam de ser extraídas da memória, apenas uma vez, durante a primeira iteração. Nas iterações subsequentes todas as instruções necessárias já estão no tampão.

O tampão de laço é semelhante, no princípio, a uma *cache* dedicada a instruções. As diferenças estão em que o tampão de laço apenas retém instruções contíguas e é muito mais pequeno em tamanho e por isso de custo mais reduzido.

A figura 11-15 dá um exemplo de tampão de laço. Se o tampão contiver 256 octetos e for usado endereçamento ao octeto, então os 8 bits menos significativos são usados para indexar o tampão. Os bits mais significativos, restantes, são verificados para determinar se o alvo da derivação está no ambiente capturado pelo tampão.

De entre as máquinas que usam um tampão de laço estão algumas máquinas CDC (Star 100, 600, 7600) e CRAY-1. Uma forma especializada de tampão de laço está disponível no Motorola 68010, para executar um laço de três instruções envolvendo uma instrução DBcc (decremento derivação condicional) (ver Problema 11.8). É mantido um tampão de três-palavras e o processador executa estas instruções repetidamente até a condição do laço ser satisfeita.

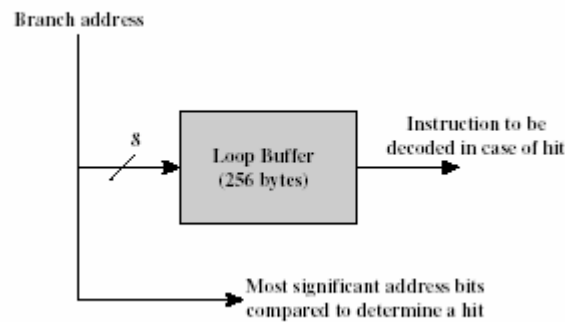


Figura 11.15: Tampão de laço.

Prognóstico de Derivação

Várias técnicas podem ser usadas para prever se uma derivação vai ocorrer. Entre as mais habituais estão as seguintes:

- Prognóstico nunca seguir
- Prognóstico seguir sempre
- Alternar entre seguir/não-seguir
- Tabela de história de derivação

As três primeiras abordagens são estáticas: não dependem da história da execução até chegar à instrução de derivação condicional. As duas últimas abordagens são dinâmicas: dependem da história da execução.

As duas primeiras abordagens são as mais simples. Ou assumem, sempre, não seguir a derivação e continuam a extrair instruções em sequência, ou assumem, sempre, seguir a derivação e extraem sempre do endereço de derivação. O 68020 e o VAX 11/780 usam o prognóstico nunca seguir. O VAX 11/780 também inclui uma característica para minimizar os efeitos de uma decisão errada. Se a extração da instrução a seguir à derivação provocar uma falha de página ou violação de protecção, o processador suspende a pré-extracção até estar seguro que a instrução deve ser extraída.

Estudos que analisam o comportamento dos programas têm mostrado que as derivações condicionais ocorrem mais do que 50% do tempo, daí que, se o custo de extração de ambos o caminhos for o mesmo, então, pré-extrair do endereço de derivação deverá proporcionar melhorar rendimento do que pré-

extrair do caminho na sequência. Contudo, numa máquina com paginação, a pré-extracção do alvo da derivação pode mais provavelmente provocar uma falha de página do que extrair a próxima instrução na sequência pelo que esta penalização deverá ser tomada em consideração. Pode ser usado um mecanismo de precaução para reduzir esta penalização.

A abordagem estática final toma a decisão baseada no código da instrução de derivação. O processador assume que segue a derivação para algumas códigos de derivação e não segue para outros apresentam relatórios com taxas de sucesso superiores a 75% usando esta estratégia.

As estratégias dinâmicas de derivação melhoram a precisão do prognóstico registando a história das instruções de derivação condicional de um programa. Por exemplo, um ou mais bits podem ser associados com cada instrução de derivação condicional que reflecte a história recente das instruções. Estes bits são referido como comutadores segue/não-segue que conduzem o processador a tomar decisões particulares na próxima vez que encontra a instrução. Tipicamente, a história destes bits não está associada com a instrução na memória principal. Em vez, disso, são mantidos numa memória local de alta-velocidade. Uma possibilidade é associar estes bits com toda a instrução de derivação condicional que está na cache. Quando a instrução é substituída na cache, perde-se a sua história. Uma outra possibilidade é manter uma pequena tabela para as instruções de derivação mais recentemente executadas com um ou mais bits por cada entrada. O processador pode fazer o acesso à tabela de modo associativo, como uma cache, ou usando os bits menos significativos do endereço da instrução de derivação.

Com um único bit, tudo o que pode ser registado é se a última execução desta instrução resultou, ou não, numa derivação. Um defeito da utilização de um único bit ocorre no caso de uma instrução de derivação condicional que se segue quase sempre, tal como uma instrução de laço. Com apenas um bit para história, um erro no prognóstico ocorre duas vezes para cada laço: uma vez à entrada e uma vez à saída.

Se se usarem dois bits, estes podem registar o resultado das duas últimas instâncias da execução das instruções associadas, ou para registar um estado de uma outra forma qualquer. A figura [11.16](#) mostra uma abordagem típica (ver Problema 11-6 para outras possibilidades). O processo de decisão pode ser representado por uma máquina de estados finitos com quatro estágios. Se as últimas duas derivações de uma dada instrução tiverem seguido o mesmo caminho, o prognóstico é tomar aquele caminho de novo. Se o prognóstico estiver errado, mantém-se o mesmo a próxima vez que a instrução for encontrada. Se o prognóstico for errado de novo, contudo, o próximo prognóstico será seguir o caminho oposto. Assim, o algoritmo requer dois prognósticos errados consecutivos para mudar a decisão sobre o prognóstico. Se uma derivação toma uma direcção não-usual uma vez, tal como para um laço, o prognóstico será errado só uma vez.

Um exemplo de um sistema que usa a abordagem comutadores segue/não-segue é o IBM 3090/400.

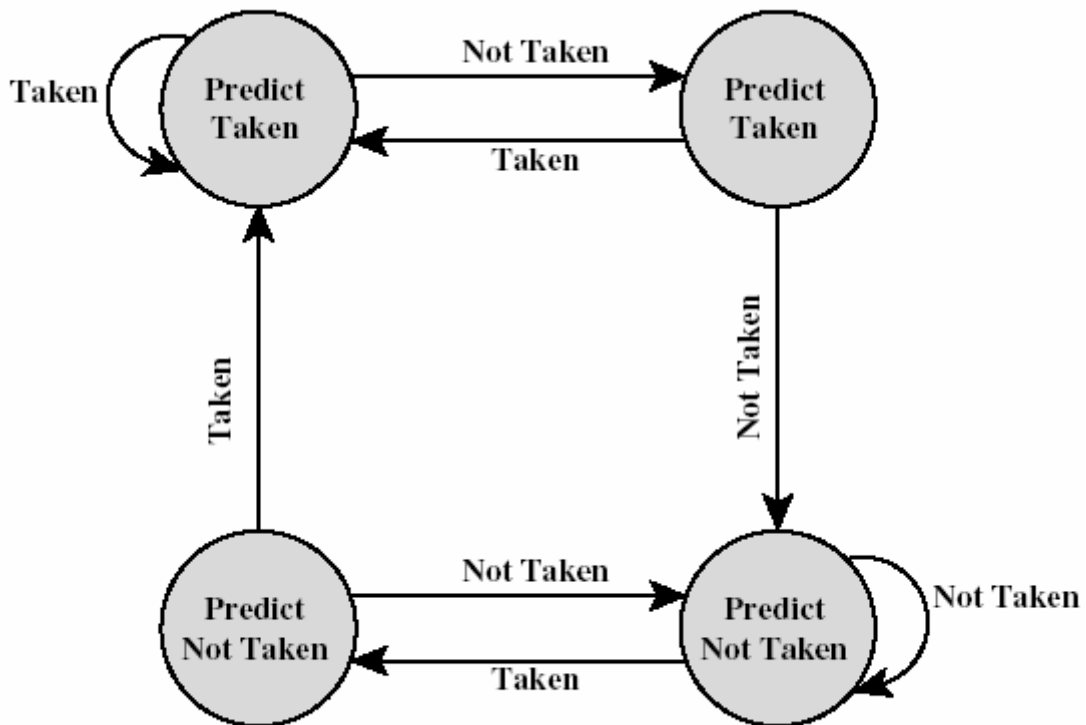


Figura 11.16: Diagrama de estado de prognósticos de derivação.

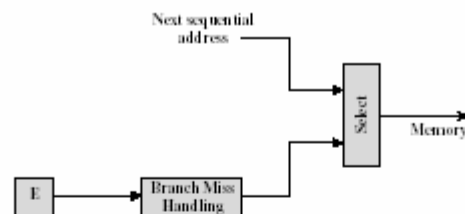
A utilização de bits de história, tal como foram apresentados, tem um inconveniente. Se a decisão tomada for seguir a derivação, a instrução alvo não pode ser extraída senão quando o endereço alvo, que é um operando na instrução de derivação condicional, for decodificado. Pode alcançar-se um melhor rendimento se a extracção da instrução poder ser iniciada mal for tomada a decisão de derivação. Para este propósito, mais informação deverá ser salvaguardada, no que é conhecido como tampão de alvo da derivação, ou tabela de história da derivação.

A tabela de história da derivação é uma pequena memória *cache* associada com o estágio de extracção da linha de encadeamento. Cada entrada na tabela consiste de três elementos: o endereço da instrução de derivação condicional, um certo número de bits de história que registam o estado de utilização daquela instrução e informação sobre a instrução alvo. O compromisso é claro: Guardar o endereço alvo produz uma tabela mais pequena, mas um tempo de extracção de instrução superior comparado com guardar a instrução alvo.

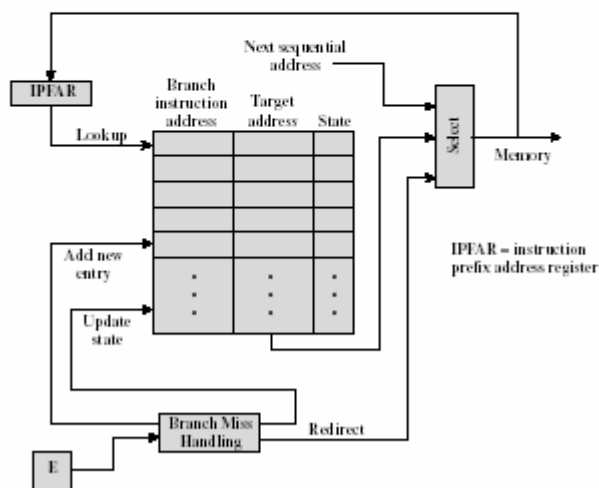
A figura 11-17 faz um contraste deste esquema com o prognóstico nunca seguir. Com a primeira estratégia, o estágio de extracção de instrução extrai sempre o próximo endereço sequencial. Se a derivação for seguida, lógica dentro do processador detecta isto e indica que a próxima instrução deve ser extraída do endereço alvo (para além de descartar a linha de encadeamento).

A tabela de história da derivação é usada como uma *cache*. Cada pré-extracção acciona uma pesquisa na tabela de história da derivação. Se não houver nenhuma correspondência, o próximo endereço sequencial é usado para a extracção. Se for encontrada uma correspondência, é feito o prognóstico baseado no estado da instrução: o próximo endereço sequencial ou o endereço alvo da derivação vai alimentar a lógica de selecção.

Quando uma instrução de derivação é executada, o estágio de execução é actualizado para reflectir um prognóstico correcto ou incorrecto. Se o prognóstico for incorrecto, a lógica de selecção é redireccionada para o endereço correcto para a próxima extracção. Quando uma instrução de derivação condicional que não está na tabela, é encontrada é adicionada à tabela e uma das entradas existentes é descartada, usando um dos algoritmos de substituição discutidos no capítulo 4.



(a) Predict never taken strategy



(b) Branch history table strategy

Figura 11.17: Lidando com derivações.

Um exemplo da implementação de uma tabela de história da derivação é o processador Advanced Micro Device AMD29000.

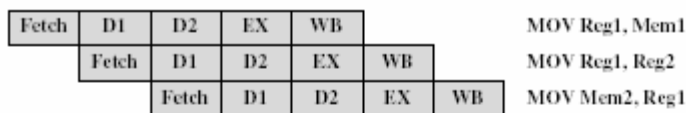
Linha de encadeamento do Intel 80846

O 80846 implementa uma linha de encadeamento com os seguintes estágios:

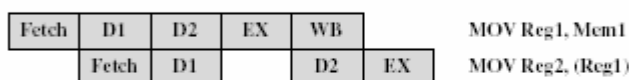
- *Extracção*: As instruções são extraídas da cache ou da memória externa e colocadas em um dos dois tampões de extracção de 16-octetos. O objectivo do estágio de extracção é preencher os tampões de pré-extracção com novos dados, logo que os dados anteriores tenham sido consumidos pelo decodificador de instruções. Uma vez que as instruções são de tamanho variável (de 1 a 11 octetos não contando os prefixos), o estado do mecanismo de pré-extracção relativamente aos outros estágios da linha de encadeamento variam de instrução para instrução. Em média cerca de cinco instruções são extraídas com cada carregamento de 16-octetos.[#!CRAW90!#]. O estágio de extracção opera independente dos outros estágios para manter cheios os tampões de pré-extracção.
- *Estágio de Descodificação 1*: Todo o código de operação e toda a informação sobre os modos de endereçamento é, no máximo, incluída nos três primeiros octetos da instrução. Assim, são passados três octetos para o estágio D1 dos tampões de pré-extracção. O decodificador D1 pode então dirigir o estágio D2 para capturar o resto da instrução (deslocamento e dados imediatos), que não estiverem envolvidos na descodificação D1.
- *Estágio de Descodificação 2*: O estágio D2 expande cada código em sinais de controlo para a ALU. Também controla o cálculo dos modos de endereçamento mais complexos.
- *Execução* : Estes estágio inclui as operações da ALU, acesso à *cache* e actualização de registos.
- *Escrita à Retaguarda*: Este estágio, se for necessário, actualiza os registos e as condições de estado modificadas durante o estágio precedente de execução. Se a instrução corrente actualizar a memória, o valor calculado é remetido para a *cache* e, ao mesmo tempo, para o interface do barramento com os tampões de escrita.

Com a utilização de dois estágios de descodificação, a linha de encadeamento pode sustentar um desempenho de perto de uma instrução por ciclo de relógio. As instruções complexas e as derivações condicionais podem fazer baixar esta taxa.

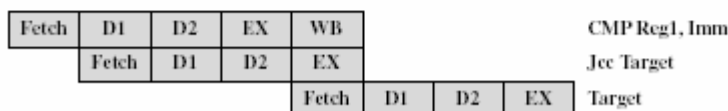
A figura [11.18](#) mostra exemplos da operação da linha de encadeamento. A parte a) mostra que não é introduzido nenhum atraso quando é necessário um acesso à memória.



(a) No Data Load Delay in the Pipeline



(b) Pointer Load Delay



(c) Branch Instruction Timing

Figura 11.18: Exemplos da linha de encadeamento de instruções do 80846.

Contudo, tal como mostra a parte b), pode existir um atraso para valores usados para calcular endereços da memória. Isto é, se um valor for carregado da memória para um registo e aquele registo for depois usado como um registo de base na próxima instrução, o processador empata durante um ciclo. Neste exemplo, o processador faz o acesso à *cache* no estágio EX da primeira instrução e, no seu estágio D2, guarda no registo o valor adquirido. Quando o estágio D2 depara com o estágio WB da instrução precedente, sinais de passagem por cima permitem ao estágio D2 ter acesso aos mesmos dados que estão a ser usados pelo estágio WB para escrita, poupando um estágio da linha de encadeamento.

A figura [11.18](#) c) ilustra a temporização de um instrução de derivação, assumindo que se segue a derivação. A instrução de comparação actualiza os códigos de condição no estágio WB e, ao mesmo tempo, as passagens por cima, tornam-nos disponíveis para o estágio EX da instrução de salto. Paralelamente, o processador executa um ciclo de extracção especulativo no endereço alvo do salto, durante o estágio EX da instrução de salto. Se o processador identifica uma condição de derivação falsa, é descartada a pré-extracção e a execução continua com a próxima instrução sequencial (já extraída e decodificada).

O Processador Pentium

Um vista geral da organização do processador Pentium é apresentada na figura 4-24. Nesta secção, examinamos alguns pormenores.

Tabela 11.1: Registos do Processador Pentium.

(a) Integer Unit			
Type	Number	Length (bits)	Purpose
General	8	32	General-purpose user registers
Segment	6	16	Contain segment selectors
Flags	1	32	Status and control bits
Instruction Pointer	1	32	Instruction pointer

(b) Floating-Point Unit			
Type	Number	Length (bits)	Purpose
Numeric	8	80	Hold floating-point numbers
Control	1	16	Control bits
Status	1	16	Status bits
Tag Word	1	16	Specifies contents of numeric registers
Instruction Pointer	1	48	Points to instruction interrupted by exception
Data Pointer	1	48	Points to operand interrupted by exception

Organização de registos

A organização de registos inclui os seguintes tipos de registos (tabela [11.1](#)):

- **Geral:** Há oito registos de 32-bits de propósito geral (ver figura [11.4b](#)). Podem ser usados em todos os tipos de instruções Pentium; podem também conter operandos para cálculo de endereços. Adicionalmente, alguns destes registos também servem para uso especial. Por exemplo, instruções sobre sequências de caracteres usam o conteúdo dos registos ECX, ESI e EDI como operandos sem terem de explicitamente referir aqueles registos na instrução. Como resultado, um certo número de instruções pode ser codificada de modo mais compacto.
- **Segmento:** Os seis registos de segmento de 16-bits contêm selectores de segmento que indexam tabelas de segmentos, tal como foi discutido no capítulo 6. O registo de segmento de código (CS) aponta para o segmento que contém a instrução que está a ser executada. O registo de segmento de pilha (SS) aponta para o segmento que contém a pilha visível ao utilizador. Os restantes registos de segmento (DS, ES, FS, GS) habilitam o utilizador a referir até quatro segmentos de dados separados, de cada vez.


- *Condições*: O registo EFLAGS contém códigos de condição e vários bits de modo.
- *Apontador de Instrução*: Contém o endereço da instrução corrente.

Há também registos especialmente dedicados à unidade de vírgula flutuante:

- *Número*: Cada registo mantém um número em vírgula flutuante com precisão estendida a 80-bits. Há oito registos que funcionam como uma pilha, com operações de empurrar e retirar disponível no jogo de instruções.
- *Controlo*: O registo de controlo de 16-bits contém bits que controlam a operação da unidade de vírgula flutuante, incluindo o tipo e controlo de arredondamento; precisão simples, dupla ou estendida; e bits para habilitar ou inibir várias condições de excepção
- *Status*: O registo status de 16-bits contém bits que reflectem o estado presente da unidade de vírgula flutuante, incluindo um apontador, de 3-bits, para o topo da pilha; códigos de condição relatam o efeito da última instrução; e condições de excepção.
- *Etiqueta de Palavra*: Este registo de 16-bits contém uma etiqueta de 2-bits para cada registo numérico de vírgula flutuante que indica a natureza do conteúdo do registo correspondente. As quatro valores possíveis, zero, especial, (NaN, infinito, des-normalizado) e vazio. Estas etiquetas habilitam o programador a testar o conteúdo dum registo numérico sem efectuar descodificação completa dos dados presentes no registo.

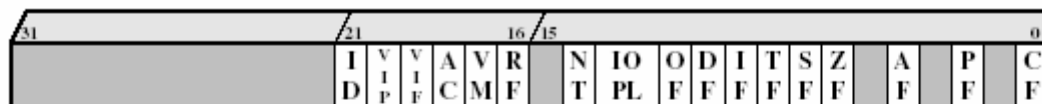
A utilização da maior parte dos registos, acima, é facilmente compreensível. Deixem-nos discorrer brevemente sobre vários dos registos.

Registo EFLAGS

O registo EFLAGS (Figura [11.19](#)) indica o estado do processador e ajuda a controlar a sua operação. Inclui os seis códigos de condição definidas na tabela  (transpor, paridade, auxiliar, zero, sinal e excesso) que relatam os resultados de uma operação inteira. Adicionalmente, há bits no registo que podem ser chamados de bits de controlo, estes são

- *Armadilha (TF)*: Quando ajustado, causa uma interrupção após a execução de cada instrução. É usado para depuração.
- *Interrupção ITF*: Quando ajustado, o processador reconhece interrupções externas.
- *Direcção (EF)*: Determina se o processamento de uma sequência de caracteres incrementa ou decrementa os meios registos de 16-bits SI e DI (para operações de 16-bits) ou os registos de 32-bits ESI e EDI (para operações de 32-bits)
- *Privilégio E/S (IOPL)*: Quando ajustado, causa o processador gerar uma interrupção em todos os acessos aos dispositivos de E/S durante a operação em modo protegido.

- *Retoma (RF)* : Permite ao programador desactivar as excepções de depuração de tal forma que a instrução possa ser retomada, depois de uma excepção de depuração, sem causar outra excepção de depuração.
- *Teste de Alinhamento (AC)*: Activada se uma palavra ou dupla palavra for endereçada fora dos limites da palavra ou dupla palavra.
- *Identificação (ID)*: Se este pode ser ajustado e limpo isso indica que este processador suporta a instrução CPUID. Esta instrução fornece informação sobre o vendedor, família e modelo.



ID	=	Identification flag	DF	=	Direction flag
VIP	=	Virtual interrupt pending	IF	=	Interrupt enable flag
VIF	=	Virtual interrupt flag	TF	=	Trap flag
AC	=	Alignment check	SF	=	Sign flag
VM	=	Virtual 8086 mode	ZF	=	Zero flag
RF	=	Resume flag	AF	=	Auxiliary carry flag
NT	=	Nested task flag	PF	=	Parity flag
IOPL	=	I/O privilege level	CF	=	Carry flag
OF	=	Overflow flag			

Figura 11.19: Registos EFLAGS do Processador Pentium.

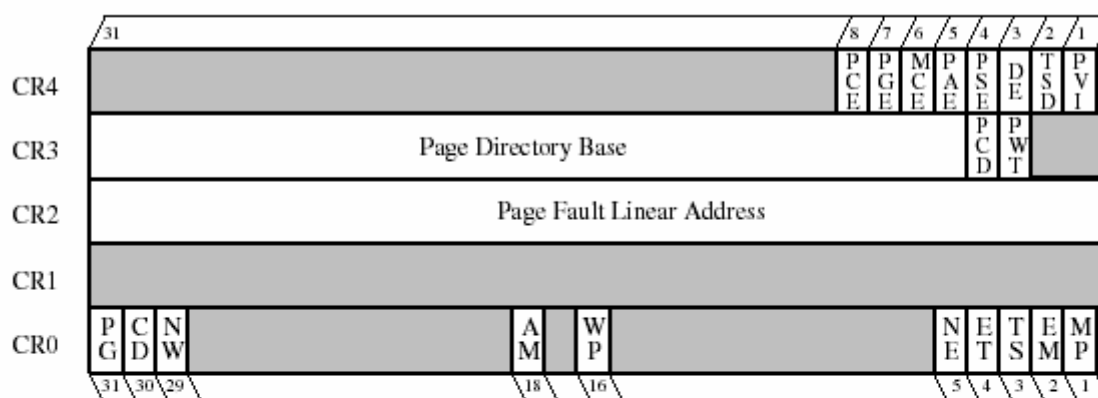
Adicionalmente, há quatro bits relacionados com o sistema de operação. A bandeira de tarefa aninhada (NT) indica que a tarefa corrente está aninhada dentro de outra tarefa que opera em modoprotegido. O bit de modo virtual (VM) permite ao programador autorizar ou inibir o modo virtual 8086 que determina se o processador corre como uma máquina 8086. A bandeira de interrupção virtual e a bandeira de interrupção virtual pendente (VIP) são usadas em ambiente multi-tarefa.

Registos de Controlo

O Pentium emprega quatro registos de controlo de 32-bits (o registo CR1 não está a uso) para controlar vários aspectos da operação do processador (Figura 11.20.) O registo CR0 contém bandeiras de controlo do sistema que controlam os modos ou indicam os estados aplicáveis, em geral, ao processador em vez da aplicação a tarefas individuais. As bandeiras são

- *Habilita Protecção (PE)*: Habilita/Impede o modo de protecção protegido.
- *Monitorização do Co-processador (MP)*: De interesse, apenas, quando se executam no Pentium programas para máquinas mais antigas. Está relacionada com a presença de um co-processador aritmético
- *Emulação (EM)*: Levantada quando o processador não tem unidade de vírgula flutuante, provoca uma interrupção quando é feita uma tentativa de executar instruções de vírgula flutuante.

- *Comutação de Tarefa (TS)*: Indica que o processador trocou de tarefa.
- *Extensões de Tipo (ET)*: Não é usada no Pentium, usada para indicar nas máquinas mais antigas o suporte a instruções matemáticas no co-processador.
- *Erro numérico (NE)*: Habilita o mecanismo convencional de relatar erros de vírgula flutuante nas linhas externas do barramento.
- *Máscara de Alinhamento (AM)*: Habilita/Impede o teste de alinhamento
- *Escrita não Imediata (NW)*: Selecciona o modo de operação da cache de dados. Quando este bit é ajustado, a cache de dados é impedida de fazer operações de escrita-imediata.
- *Desactiva Cache (CD)*: Activa/Desactiva o mecanismo interno de encher a cache.
- *Paginação (PG)*: Activa/Desactiva a paginação.



PCE = Performance Counter Enable	PG = Paging
PGE = Page Global Enable	CD = Cache Disable
MCE = Machine Check Enable	NW = Not Write Through
PAE = Physical Address Extension	AM = Alignment Mask
PSE = Page Size Extensions	WP = Write Protect
DE = Debug Extensions	NE = Numeric Error
TSD = Time Stamp Disable	ET = Extension Type
PVI = Protected Mode Virtual Interrupt	TS = Task Switched
VME = Virtual 8086 Mode Extensions	EM = Emulation
PCD = Page-level Cache Disable	MP = Monitor Coprocessor
PWT = Page-level Writes Transparent	PE = Protection Enable

Figura 11.20: Registos de Controlo do Processador Pentium.

Quando a paginação está habilitada, os registos CR2 e CR3 são válidos. O registo CR2 contém o endereço linear de 32-bits da última página a que foi feito o acesso antes da interrupção da falta de página. Os 20-bits mais à esquerda de CR3 contêm os 20-bits mais significativos do endereço base do directório de páginas. O resto do endereço contém zeros. Dois bits de CR3 são usados para alimentar os pinos que controlam a operação da *cache* externa. O bit (PCD) activa ou desactiva a *cache* externa e o bit de escrita transparente (PWT) controla a escrita imediata na *cache* externa.

Seis bits de controlo adicionais são definidos em CR4:

- *Extensão de Modo Virtual-8086 (VME)*: Activa o suporte à bandeira de interrupção virtual no modo virtual-8086.
- *Interrupções no Modo-Protegido Virtual (PVI)*: Activa o suporte à bandeira de interrupção virtual no modo-protegido
- *Desactiva Registo de Tempo (TSD)*: Desactiva a leitura da instrução do contador de registo de tempo (RDTSC) que é usada para efeitos de depuração.
- *Extensões de Depuração (DE)*: Activa os pontos de paragens de E/S, isto permite que o processador seja interrompido em leituras e escritas de E/S.
- *Extensões ao Tamanho da Página (PSE)*: Autoriza o uso de páginas de 4-Moctetos.
- *Activa Teste da Máquina (MCE)*: Autoriza a interrupção de teste de máquina que ocorre quando há um erro de paridade dos dados durante um ciclo de leitura de barramento ou quando um ciclo de barramento não é completado com sucesso.

Processamento de Interrupções

O processamento de interrupções num processador é uma facilidade fornecida para suportar o sistema de operação. Permite um programa de aplicação ser suspenso, de forma a que uma variedade de condições de interrupção possam ser servidas, e mais tarde retornado.

Excepções e Interrupções

Duas classes de eventos causam o Pentium suspender a execução da cadeia corrente de instruções e responder ao evento: interrupções e excepções. Em ambos os casos, o processador salvaguarda o contexto do processo corrente e transfere-se para uma rotina pré-definida para servir a condição. Uma *interrupção* é gerada por um sinal do *hardware* o que pode ocorrer em instantes imprevisto durante a execução de um programa. Uma *excepção* é gerada pelo software e é provocada pela execução de uma instrução. Há duas fontes de interrupção e duas fontes de excepção:

1. Interrupções

- *Interrupções Mascaráveis*: Recebidos no pino INTR do processador. O processador não reconhece uma interrupção mascarável a menos que a bandeira de habilitação de interrupções esteja levantada
- *Interrupções Não-Mascaráveis*: Recebidos no pino NMI do processador. O reconhecimento de tais interrupções não pode ser impedido.

2. Excepções

- *Excepções Detectados pelo Processador*: Ocorrem quando o processador encontra um erro enquanto tenta executar uma instrução.

- *exceções Programadas*: Estas são instruções que geram instruções (INT0, INT3, INT, BOUND).

Tabela de Vector de Interrupções

O processamento de interrupções no Pentium usa a tabela de vector de interrupções. A todo o tipo de interrupção é atribuído um número e este número é usado para indexação da tabela de vectores de interrupção. Esta tabela contém 256 vectores de interrupção de 32-bits, que é o endereço (segmento e deslocamento) da rotina de serviço de interrupções para esse número de interrupção.

A tabela [11.2](#) mostra a atribuição dos números na tabela de vector de interrupções; as entradas a sombreado representam interrupções, enquanto as entradas não sombreadas são exceções. A interrupção de hardware NMI é tipo 2. Às interrupções de hardware INTR são atribuídos números na gama 32-255; quando é gerada uma interrupção, isso deve ser acompanhado no barramento com o número de vector de interrupção para esta interrupção. Os restantes números de vector são usados para exceções.

Tabela 11.2: Tabela de vectores de excepção e de interrupção do Pentium.

Vector Number	Description
0	Divide error; division overflow or division by zero
1	Debug exception; includes various faults and traps related to debugging
2	NMI pin interrupt; signal on NMI pin
3	Breakpoint; caused by INT 3 instruction, which is a 1-byte instruction useful for debugging
4	INT0-detected overflow; occurs when the processor executes INTO with the OF flag set
5	BOUND range exceeded; the BOUND instruction compares a register with boundaries stored in memory and generates an interrupt if the contents of the register is out of bounds.
6	Undefined opcode
7	Device not available; attempt to use ESC or WAIT instruction fails due to lack of external device
8	Double fault; two interrupts occur during the same instruction, and cannot be handled serially
9	Reserved
10	Invalid task state segment; segment describing a requested task is not initialized or not valid
11	Segment not present; required segment not present
12	Stack fault; limit of stack segment exceeded or stack segment not present
13	General protection; protection violation that does not cause another exception (e.g., writing to a read-only segment)
14	Page fault
15	Reserved
16	Floating-point error; generated by a floating-point arithmetic instruction
17	Alignment check; access to a word stored at an odd byte address or a doubleword stored at an address not a multiple of 4
18	Machine check; model specific
19-31	Reserved
32-255	User interrupt vectors; provided when INTR signal is activated

Unshaded: exceptions

Shaded: interrupts

Se mais do que uma excepção ou interrupção estiver pendente, o processador serve-as numa ordem prevista. A localização dos números de vector não reflecte a prioridade. Em vez disso, a prioridade entre exceções e interrupções é organizada em cinco classes. Em ordem descendente de prioridade, estas são

- *Classe 1:* Armadilhas na instrução precedente (número de vector 1).
- *Classe 2:* Interrupções externas (2, 32-255).
- *Classe 3:* Falhas de extracção da próxima instrução (3,14).
- *Classe 4:* Falhas da descodificação da próxima instrução (6,7).
- *Classe 5:* Falhas durante a execução de uma instrução (0,4,5,8,10-14,16,17).

Tratamento de Interrupções

Tal como com a transferência da execução através da instrução CALL, a transferência para a rotina de tratamento de interrupções usa a pilha de sistema para salvar o estado do processador. Quando uma interrupção ocorre, e é reconhecida pelo processador, toma lugar a seguinte sequência de eventos.

1. Se a transferência envolve uma alteração do nível de privilégio, então o corrente registo de segmento de pilha ou o corrente registo estendido de segmento de pilha (ESP) é empurrado para a pilha.
2. O valor corrente do registo EFLAGS é empurrado para a pilha.
3. Tanto a bandeira de interrupção (IR) como a de armadilha (TF) são limpas. Isto desactiva as interrupções INTR e a armadilha ou a propriedade passo a passo.
4. Se a interrupção for acompanhada por um código de erro, então o erro de código empurrado para a pilha.
5. O conteúdo da tabela de vectores de interrupções é extraído e carregado para os registos CS e IP ou EIP. A execução continua da rotina de serviço de interrupção.

This document was created with Win2PDF available at <http://www.daneprairie.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.