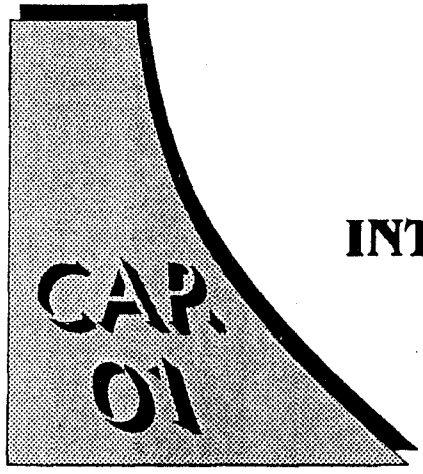


INTRODUÇÃO

HISTÓRICO

Vários fabricantes têm a sua participação na história dos microprocessadores, e disputam o mercado procurando lançar novas tecnologias e arquiteturas para obterem a melhor relação custo/performance, aumentando a freqüência de operação, a capacidade de integração de componentes em um único circuito integrado e o número de bits para transferência de dados e para endereçamento de memória, e implementando novas técnicas de acesso aos dados em memória. Porém, como a abordagem restringe-se à família 486, forneceremos apenas um histórico dos microprocessadores Intel, a pioneira nesta área.

Em 1971, a Intel lançava o seu primeiro microprocessador, o 4004, de 4 bits e capaz de realizar 60.000 instruções por segundo, objetivando atender o pedido de uma empresa japonesa, fabricante de calculadoras, que o utilizaria em seus produtos.

O próximo microprocessador foi o 8008, lançado em 1972, que era de 8 bits e executava até 300.000 instruções por segundo e endereçava 16KB de memória. A partir deste, iniciou-se a aplicação do microprocessador em projetos de microcomputadores para uso genérico.

O 8080 e o 8085, ambos de 8 bits, lançados respectivamente em 1974 e 1976, foram aperfeiçoamentos do 8008, aumentando o conjunto de instruções, a escala de integração de componentes para um único circuito integrado, a freqüência de operação e a capacidade de endereçamento de memória (64 KB - muita memória para aquela época). Eles deram base para o desenvolvimento de vários programas aplicativos.

programadores traduzirem os códigos do 8080 para o 8086. Mas para facilitar ainda mais a migração de 8 para 16 bits, a Intel lançou posteriormente o 8088, um microprocessador com a mesma arquitetura interna de 16 bits do 8086, porém com apenas 8 bits de barramento de dados externo. Quando o 8088 deseja buscar um dado de 16 bits, ele lê automaticamente o primeiro byte e, a seguir, lê o segundo byte.

Naquele mesmo período, a Intel lançava também mais um membro da família: o co-processador aritmético 8087, o qual adicionou um conjunto de instruções de ponto-flutuante ao conjunto do 8088/8086 e aumentou a performance do sistema, pois o 8087 descentraliza o processamento em casos de instruções aritméticas.

Quando a IBM escolheu o 8088 para o seu projeto de um microcomputador pessoal, o IBM PC - "Personal Computer" - no ano de 1981, a arquitetura 8086/8088 deu início a um padrão que passou a ser mundialmente adotado. A Intel fornecia também os componentes de suporte básico para a arquitetura PC: o gerador de clock 8284, o controlador do sistema 8288, o controlador de interrupção 8259, o controlador de DMA 8237, o temporizador/contador 8253 e o controlador de memória 8207.

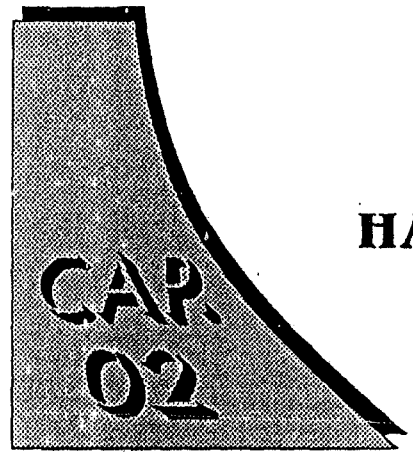
Baseada na arquitetura do 8086, a Intel em 1982, desenvolveu duas novas implementações para servir as necessidades do mercado: o 80186 e o 80286. O 80186 foi desenvolvido para aplicações específicas, principalmente em áreas como automação industrial, telecomunicações, enquanto o 80286 foi desenvolvido para microcomputadores pessoais, pois o mercado necessitava de mais performance, mantendo-se a compatibilidade com os softwares desenvolvidos para os microcomputadores baseados nos microprocessadores anteriores.

O 80286 possui como características adicionais e especiais a capacidade de realizar multitarefamento ("multitasking") e gerenciamento de memória virtual, quebrando a barreira do 1 MB de memória endereçável, podendo chegar até 16 MB. Este microprocessador deu início a um outro padrão de arquitetura de microcomputadores denominada AT, nome derivado do microcomputador IBM PC-AT, ainda hoje mundialmente utilizada. Com os recursos especiais do 80286 e a sua alta performance, surgiram novos aplicativos e sistemas operacionais como o Xenix, para implantação de um sistema multiusuário e multitarefa com vários terminais, o OS/2 também com recursos de multitarefamento, e o aplicativo Windows. Juntamente com o 80286 foram lançados também o co-processador 80287 e novos componentes básicos para a arquitetura.

O primeiro microprocessador da Intel de arquitetura interna e externa de 32 bits foi o 80386; lançado em 1985, capaz de endereçar fisicamente até 4 GB de memória e 64 TB de memória virtual, contém o equivalente a 275.000 transistores utilizando tecnologia CHMOS. Apresenta recursos adicionais de gerenciamento de memória e proteção que permite a execução simultânea de mais de um sistema operacional. Além de propiciar o desenvolvimento de uma série de softwares e sistemas operacionais de 32 bits e que desfrutam dos recursos adicionais deste novo microprocessador, manteve a compatibilidade com aqueles desenvolvidos para o 8086 ao 80286. Devido ao seu alto desempenho, e com as implementações técnicas nas arquiteturas dos microcomputadores, tais como: entrelaçamento e paginação da memória dinâmica, realocação do BIOS (programa básico do microcomputador residente em memórias EPROM) da CPU e do vídeo em memória RAM, onde os acessos são mais rápidos, memórias cache intermediárias para tornar ágil o acesso aos dados da memória principal, todas essas técnicas possibilitaram a unidade de medida de performance MIPS (Milhões de Instruções Por Segundo) passar a fazer parte dos softwares "benchmarks" para microcomputadores, a qual era utilizada apenas em computadores de maior porte. Por se tratar de um microprocessador de 32 bits, foram desenvolvidos novos componentes periféricos como o controlador de DMA 82380, o controlador de cache 82385 e o co-processador 80387.

Objetivando lançar no mercado um microprocessador de menor custo para implementação em um projeto, em relação ao 80386DX, a Intel em 1988 introduziu o 80386SX, também com arquitetura interna de 32 bits, porém com apenas 16 bits no barramento de dados externo e 24 bits no barramento de endereços, o que lhe permite ainda executar todos os softwares desenvolvidos para o 8086 ao 80386DX. Os seus novos componentes periféricos foram: o controlador de interrupção e de memória RAM dinâmica 82335SX, o controlador de DMA 82370, o controlador de cache 82385SX e o co-processador aritmético 80387SX. A ótima relação custo/performance desta arquitetura 386SX deu base ao crescimento do uso de softwares de 32 bits e dos recursos de multiprocessamento e multitarefamento.

1-1 a 1-5
2-1 a 2-7
2-11 a 2-35
3-1 a 3-27
3-52 a 3-55



HARDWARE

Dando continuidade à evolução de seus produtos, a Intel Corporation desenvolveu o microprocessador 80486DX, lançando-o em 1989. Na realidade, ele é mais do que apenas um microprocessador, pois além de possuir internamente um microprocessador compatível com o 80386, possui ainda um co-processador compatível com o 80387 e 8 Kbytes de cache. Porém, a própria Intel, percebendo que nem todos os sistemas necessitavam do co-processador, lançou em 1991 o 80486SX, cuja diferença básica em relação ao 80486DX é a de não possuir um co-processador internamente, oferecendo assim uma opção de custo inferior. Para os usuários que desejam posteriormente acrescentar um co-processador ao 80486SX, a Intel desenvolveu outro circuito integrado, o 80487SX, que é o co-processador dedicado ao 80486SX.

Em 1992, a Intel lançou o 80486DX2; uma versão do 80486DX que possui internamente um dobrador da freqüência de clock, o que resulta num aumento da performance do sistema. Para atualizar um sistema que utiliza um microprocessador DX por um DX2, não é necessário alterar o projeto do circuito onde se encontra instalado, basta substituir os microprocessadores, pois os mesmos são compatíveis em termos de pinagem.

Seguindo essa tendência de facilitar a atualização do microprocessador de um sistema, sem a necessidade de se alterar o projeto do circuito onde se encontra instalado, a Intel lançou em 1992, o OverDrive, que se trata de um 80486DX2 destinado a aumentar a performance de um sistema que utiliza um 80486SX, bastando apenas instalá-lo no soquete destinado ao co-processador 80487SX. Quando o OverDrive é instalado, ele assume o controle do sistema, pois o 80486SX é automaticamente inibido.

Neste capítulo, são apresentadas as características de hardware do 80486DX em detalhes. Devido à sua grande similaridade com os demais processadores da família (80486SX/80487SX, 80486DX2 e OverDrive) e para não tor-

80486DX, 80486DX2, 80486SX e OVERDRIVE

Em 1989, a Intel lançou o microprocessador 80486DX, também de 32 bits e totalmente compatível com a família 386. A alta capacidade de integração é um dos fatores que o destaca, pois conseguiu-se concentrar o equivalente a mais de 1 milhão de transistores em uma única pastilha, na qual agregou-se ao seu antecessor 80386, um co-processador compatível com o 80387, um cache de 8 KB de memória e uma unidade de gerenciamento de memória paginada. Além da alta capacidade de integração, destaca-se também pela sua alta performance, resultante da tecnologia RISC ("Reduced Instruction Set Computer") empregada em sua arquitetura, possibilitando um aumento médio de performance de 3 vezes em relação ao 80386, quando ambos operam no mesmo clock.

A Intel lançou em 1991 o 80486SX, cuja diferença básica em relação ao 80486DX está na não existência do co-processador, o qual não é utilizado por todos os softwares e passou a ser implementado opcionalmente através do 80487SX. Esta estratégia trouxe uma opção de custo e performance mais receptível ao mercado.

Em 1992, a Intel lançou uma nova versão do 80486DX: o 80486DX2. Trata-se de uma versão que possui internamente um dobrador da freqüência de clock, o que permite aumentar a performance de um sistema em cerca de até 70%. Para atualizar um sistema que utiliza um microprocessador DX por um DX2, não é necessário alterar o projeto do circuito onde se encontra instalado, basta substituir os microprocessadores, pois os mesmos são compatíveis em termos de pinagem.

Seguindo essa tendência de facilitar a atualização do microprocessador de um sistema, sem a necessidade de se alterar o projeto do circuito onde se encontra instalado, a Intel lançou em 1992, o OverDrive, que se trata de um 80486DX2 destinado a aumentar a performance de um sistema que utiliza um 80486SX, bastando apenas instalá-lo no soquete destinado ao co-processador 80487SX. Quando o OverDrive é instalado, ele assume o controle do sistema, pois o 80486SX é automaticamente inibido.

A tabela a-seguir, traz um resumo das principais características dos microprocessadores da família Intel.

CARACTERÍSTICAS		MICROPROCESSADORES					
		8086	8088	80286	80386	80386SX	80486SX/DX/DX2/ OverDrive
BARRAMENTO (BITS) - DADOS EXTERNO - ENDEREÇOS		16 20	8 20	16 24	32 32	16 24	32 32
DADOS INTERNOS (BITS)		16	16	16	32	32	32
CO-PROCESSADOR INTERNO		Não	Não	Não	Não	Não	DX-Sim/SX-Não
CACHE INTERNO		Não	Não	Não	Não	Não	Sim
MEMÓRIA ENDEREÇAVEL		1MB	1MB	16MB	4GB	16MB	4GB
MEMÓRIA VIRTUAL		Não	Não	Sim	Sim	Sim	Sim
GERENCIAMENTO DE PROTEÇÃO DE MEMÓRIA		Não	Não	Sim	Sim	Sim	Sim
ENDEREÇAMENTO DE E/S		64 KB	64 KB	64 KB	64 KB	64 KB	64 KB
MODOS DE ENDEREÇAMENTO		24	24	24	28	28	28
Nº DE REGISTRADORES	Aritméticos	8	8	8	8	8	8
	Índice	4	4	4	8	8	7
	Segmento	4	4	4	6	8	6
	Propósito Geral	8	8	8	8	8	8
COMPATIBILIDADE DE CÓDIGOS		8086			8086/80286		8086/80286/ 80386/80386SX
ENCAPSULAMENTO	Pinos	40	40	68	132	100	PGA - 168 PQFP - 196
	Tipo	DIP	DIP	LLC PLCC	PGA	PQFP	PGA(DX/DX2) PGA/PQFP(SX)
ALIMENTAÇÃO		5 V	5 V	5 V	5 V	5 V	5 V
TECNOLOGIA DO PROCESSO DE FABRICAÇÃO		NMOS CHMOS	NMOS CHMO	NMOS	CHMOS	CHMOS	CHMOS

Tabela 1.1 - Resumo das principais características dos microprocessadores c família Intel.

nar esta literatura repetitiva, o descritivo de hardware destes últimos conterá apenas as diferenças deles com o 80486DX.

1. O MICROPROCESSADOR 80486DX

CARACTERÍSTICAS

O 80486DX é um microprocessador de 32 bits de dados fabricado através do processo CHMOS-IV, no qual conseguiu-se concentrar mais de 1 milhão de transistores numa única pastilha no encapsulamento PGA ("Pin-Grid-Array") de 168 pinos, enquanto que o seu antecessor, o 80386, possui aproximadamente 275.000 transistores (1/4 do 80486DX) no mesmo tipo de encapsulamento com 132 pinos. Porém, o número de transistores não é importante por si só. A maneira na qual eles são utilizados no 80486DX, foi baseada na tecnologia RISC ("Reduced Instruction Set Computer"), o que possibilitou aumento médio de performance de 3 vezes em relação ao 80386, quando ambos estão operando no mesmo clock.

Além de possuir internamente um 80386, tornando-o binariamente compatível com todos os programas desenvolvidos para os microprocessadores da linha 8086 ao 80386, o 80486DX possui ainda uma versão avançada de uma unidade de ponto flutuante compatível com o 80387, um cache de 8 Kbytes e uma unidade de gerenciamento de memória com paginação. Tanto o barramento de dados, como os registradores internos e o barramento de endereços do 80486DX são de 32 bits, possuindo assim a capacidade de endereçamento físico de memória de 4 Gbytes e de endereçamento virtual de 64 Tbytes.

Todas as novas características que foram incorporadas no 80486DX têm como objetivos aprimorar os sistemas de multiprocessamento, aumentar a performance e a facilidade de utilização, além de possibilitar um extensivo autoteste.

ARQUITETURA INTERNA

A arquitetura interna do 80486DX consiste de 9 unidades funcionais distintas que operam em paralelo, possibilitando que tanto a busca, decodificação e execução das instruções, assim como o gerenciamento da memória e os acessos ao barramento sejam realizados simultaneamente, resultando num processamento de alta performance. Possui ainda internamente dois barramentos de dados de 32 bits dedicados para agilizar as transferências entre as unidades.

Para o melhor entendimento desta arquitetura, são apresentados alguns conceitos:

Memória virtual:

A memória virtual torna possível a utilização de programas e estruturas de dados maiores do que a memória física endereçável pelo 80486DX que é de 4 Gbytes, definindo assim o espaço de endereçamento lógico que pode exceder o tamanho do espaço de endereçamento físico disponível. Isto é possível porque dispositivos de armazenamento externos ao 80486DX são utilizados no lugar de muitas memórias RAM, sendo que a conversão de endereço lógico para o endereço físco é feita através do hardware interno ao microprocessador. Este largo espaço de endereço virtual de 64 Tbytes do 80486DX é subdividido em unidades de alocação chamadas de segmentos ou páginas, que ora podem endereçar a memória principal do sistema ou dispositivos secundários de armazenamento, como por exemplo as unidades de discos rígidos. Como isto é controlado pelo sistema operacional, esta troca de dados e códigos entre a memória e os dispositivos de armazenamento permance transparente para os programas aplicativos.

Segmentação:

Quando a memória virtual é segmentada, cada módulo de dados ou códigos pode ser associado com o seu segmento de memória lógica. Isto facilita a implementação de esquemas que protegem particularmente cada um destes módulos, compartilhar informações entre eles, ou manipulá-los juntos ou separadamente. O 80486DX permite segmentos de até 4 Gbytes com até 16.000 segmentos por tarefa.

Paginação:

A memória virtual também pode ser subdividida em páginas com a diferença de que enquanto os segmentos variam em tamanho, as páginas possuem um tamanho fixo de 4 Kbytes. Desta maneira, um programa pode tirar o máximo proveito dos esquemas de gerenciamento de memória, permitindo que o espaço de endereçamento lógico seja definido utilizando a segmentação, enquanto o espaço de endereçamento físico é gerenciado através da paginação.

As unidades funcionais do 80486DX são identificadas a seguir:

Interface com o barramento:

Esta unidade provê o interfaceamento entre o microprocessador e o ambiente no qual ele se encontra. Isto ocorre toda vez que a unidade de "prefetch" requisitar a busca de instruções ou de dados que não estão presentes no cache interno, através da geração ou processamento dos sinais de endereços, dados e controle. Desta maneira, o 80486DX acessa as memórias e os dispositivos de I/O externos e realiza os ciclos de cache.

Unidade de "prefetch":

As instruções provenientes do cache ou da unidade de barramento são armazenadas numa pilha de 32 bytes presentes nesta unidade.

Decodificador de instruções:

Esta unidade decodifica as instruções contidas na pilha da unidade de "prefetch" para serem processadas pela unidade de controle.

Unidade de controle:

Esta unidade de controle contém a ROM e o hardware necessário para realizar as operações de cálculo dos endereços a partir das instruções decodificadas.

Unidade de execução:

Esta é a unidade que executa as instruções e que se comunica com as demais unidades através das suas seguintes subunidades: ULA (Unidade Lógica e Aritmética), deslocador para operações de multiplicação, divisão e deslocamento e o arquivo que contém os registradores do microprocessador.

Unidade de segmentação:

Quando é requisitada pela unidade de execução, esta unidade converte o endereço lógico em endereço linear, enviando-o para a unidade de paginação.

Unidade de paginação:

Quando o mecanismo de paginação do 80486DX está habilitado, esta unidade converte o endereço linear gerado pela unidade de segmentação em endereço físico. Quando a paginação não está habilitada, o endereço físico é o mesmo do endereço linear e nenhuma conversão é necessária.

Cache:

Esta unidade possui o controlador e os 8 Kbytes de memória cache do 80486DX. Quando a unidade de "prefetch" solicita um dado, esta solicitação é enviada inicialmente para o cache. Desta maneira, a unidade de interface com o barramento somente será ativada para buscar o dado externo se o cache estiver desabilitado ou se este dado não estiver presente na memória cache ou se o ciclo corrente não for um ciclo com capacidade de cache. (por exemplo, ciclo de I/O).

Unidade de ponto flutuante:

Esta unidade contém o hardware necessário para realizar as operações aritméticas que seriam executadas via software num tempo maior, caso esta unidade não estivesse presente e é uma extensão do resto da arquitetura.

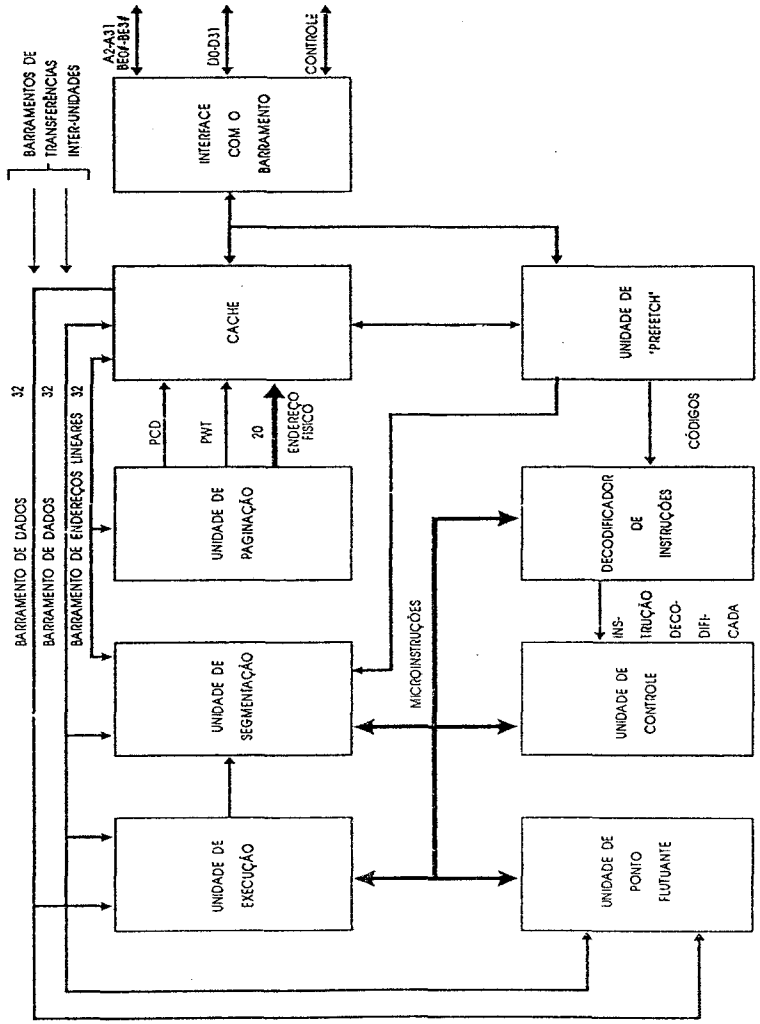


Figura 2.1 Unidades funcionais do 80486DX.

PINAGEM

As figuras e a tabela a seguir, mostram a pinagem do 80486DX:

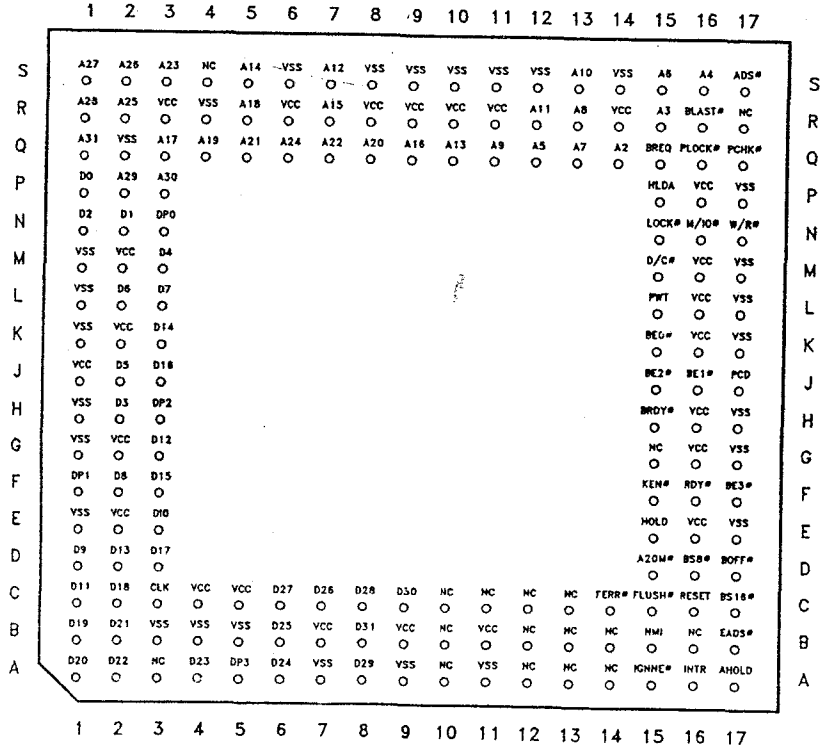


Figura 2.2 Pinagem do 80486DX (lado dos pinos).

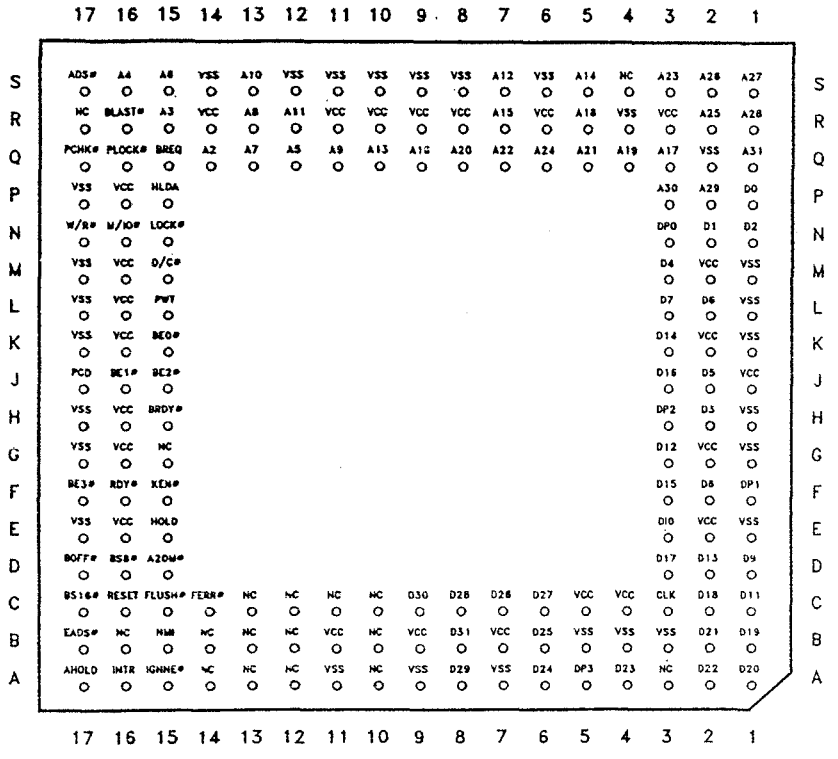


Figura 2.3 Pinagem do 80486DX (lado superior).

ENDEREÇOS	
Pino	Número
A2	Q14
A3	R15
A4	S16
A5	Q12
A6	S15
A7	Q13
A8	R13
A9	Q11
A10	S13
A11	R12
A12	S7
A13	Q10
A14	S5
A15	R7
A16	Q9
A17	Q3
A18	R5
A19	Q4
A20	Q8
A21	Q5
A22	Q7
A23	S3
A24	Q6
A25	R2
A26	S2
A27	S1
A28	R1
A29	P2
A30	P3
A31	Q1

DADOS	
Pinos	Número
D0	P1
D1	N2
D2	N1
D3	H2
D4	M3
D5	J2
D6	L2
D7	L3
D8	F2
D9	D1
D10	E3
D11	C1
D12	G3
D13	D2
D14	K3
D15	F3
D16	J3
D17	D3
D18	C2
D19	B1
D20	A1
D21	B2
D22	A2
D23	A4
D24	A6
D25	B6
D26	C7
D27	C6
D28	C8
D29	A8
D30	C9
D31	B8

Tabela 2.1 - Numeração dos pinos do 80486DX (PARTE)

CONTROLE	
Pino	Número
A20M#	D15
ADS#	S17
AHOLD	A17
BE0#	K15
BE1#	J16
BE2#	J15
BE3#	F17
BLAST#	R16
BOFF#	D17
BRDY#	H15
BREQ#	Q15
BS8#	D16
BS16#	C17
CLK	C3
D/C#	M15
DP0	N3
DP1	F1
DP2	H3
DP3	A5
EADS#	B17
FERR#	C14
FLUSH#	C15
HLDA	P15
HOLD	E15
IGNNE#	A15
INTR	A16
KEN#	F15
LOCK#	N15
M1O#	N16
NMI	B15
PCD	J17
PCHK#	Q17
PWT	L15
PLOCK#	Q16
RDY#	F16
RESET	C16
W/R#	N17

N/C	VCC	VSS
Número	Número	Número
A3	B7	A7
A10	B9	A9
A12	B11	A11
A13	C4	B3
A14	C5	B4
B10	E2	B5
B12	E16	E1
B13	G2	E17
B14	G16	G1
B16	H16	G17
C10	J1	H1
C11	K2	H17
C12	K16	K1
C13	L16	K17
G15	M2	L1
R17	M16	L17
S4	P16	M1
	R3	M17
	R6	P17
	R8	Q2
	R9	R4
	R10	S6
	R11	S8
	R14	S9
		S10
		S11
		S12
		S14

Tabela 2.1 - Numeração dos pinos do 80486DX (PARTE)

INTERFACE DE HARDWARE

O barramento do microprocessador 80486DX foi projetado para ser o mais similar possível ao do 80386, porém, novas características foram adicionadas, resultando num aumento de performance e funcionalidade, tais como: "1X clock", mecanismo de barramento "burst", mecanismo de invalidação da linha do cache, capacidade aprimorada de arbitragem do barramento, mecanismo de tratamento do barramento em 8 bits e detecção e verificação da paridade dos dados.

A característica de "1X clock" difere do 80386 que opera com "2X clock". Isto significa que, por exemplo, se a freqüência de operação do 80486DX for de 33 MHz, ele necessitará um clock nesta mesma freqüência, enquanto o 80386 necessitará um clock de 66 MHz. Esta característica traz o benefício de simplificar os projetos de hardware dos sistemas, pois assim, os componentes associados ao 80486DX poderão ser mais lentos, reduzindo a sensibilidade e a suceptibilidade do sistema aos ruídos e às interferências de alta freqüência.

Assim como o 80386DX, o 80486DX possui barramentos separados para dados e endereços. O barramento de dados bidirecional é de 32 bits, enquanto o barramento de endereços possui duas componentes: 30 linhas de endereços (A2 a A31) e 4 linhas de habilitação de bytes (BE0# a BE3# - sinais de "byte enable"). As linhas de endereços formam os 30 bits superiores do endereço e os sinais de "byte enable" selecionam bytes individuais dentre uma localização de 4 bytes. As linhas de endereços são bidirecionais e são utilizadas como entradas nas invalidações das linhas do cache.

Para realizar operações de preenchimento do cache interno ("cache fill") em alta velocidade, a partir dos dados provenientes da memória externa, o 80486DX possui um mecanismo de barramento chamado "burst". Os ciclos "burst" podem transferir os dados para dentro do microprocessador a uma taxa de um dado por clock, enquanto os ciclos não "burst" necessitam de dois como pode ser visualizado posteriormente na carta de tempos do ciclo "burst". Estes ciclos não se limitam apenas a operações do cache, mas sim para todos os ciclos que requerem mais do que um ciclo de dados. Isto acontece nos ciclos de escrita e leitura de memória cujos operandos são maiores do que 32 bits e que correspondem a longas leituras e escritas de ponto flutuante (64 bits), leituras do descritor da tabela de segmento (64 bits) e "cache line fills" (128 bits).

A característica de "hold" do 80486DX também é similar ao do 80386. Durante este ciclo, o microprocessador cede o controle dos barramentos locais, desativando o seu barramento de dados, endereços e controle. Esta técnica é muito utilizada em sistemas multiprocessados, onde existem vários microprocessadores dominadores ("masters") presentes no barramento local, ou em operações de DMA ("direct memory access") e até mesmo em ciclos de "refresh" das

memórias DRAM do sistema. Adicionalmente ao "hold", o 80486DX possui a capacidade de "address hold", ou seja, "hold" apenas do barramento de endereços, enquanto os demais continuam ativos. Esta característica é utilizada nas invalidações das linhas do cache.

Apesar do 80486DX ser um microprocessador de 32 bits de dados, ele também pode realizar transferências de dados em barramentos de 16 e de 8 bits, através dos pinos de entrada de controle BS16# e BS8#, permitindo assim que o tamanho do barramento seja especificado para cada ciclo de barramento. Esta determinação dinâmica do tamanho do barramento de dados fornece aos sistemas baseados no 80486DX, a flexibilidade de utilizar componentes e barramentos de 16 e 8 bits.

Para manter a confiabilidade nos dados provenientes do sistema, o 80486DX possui internamente o circuito de detecção e verificação da paridade par dos dados. Caso esta paridade não corresponda à do dado lido, o microprocessador informa ao sistema, via hardware, que um erro de dados ocorreu.

A figura a seguir, mostra os pinos de entrada e saída do 80486DX arranjados em grupos funcionais.

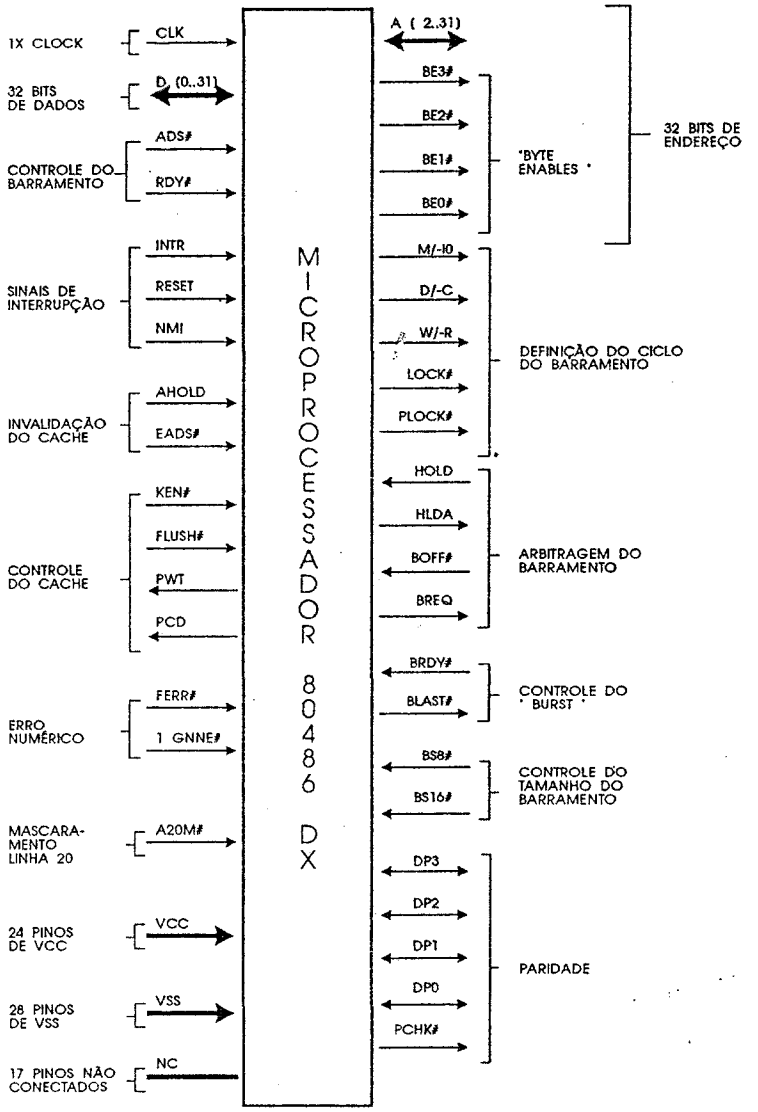


Figura 2.4 Pinos do 80486DX em arranjos funcionais.

DESCRIÇÃO DOS PINOS

A seguir, é apresentada uma breve descrição dos pinos do 80486DX, mantendo a convenção da Intel, onde o símbolo "#" no final do nome do pino indica-o como ativo em nível lógico baixo ("0") e quando não está presente significa que o mesmo é ativo em nível lógico alto ("1").

Clock (CLK):

A entrada CLK ("CLOCK") fornece a temporização básica e define a freqüência interna de operação do 80486DX. Todos os parâmetros externos de temporização são especificados em relação à sua borda de subida.

Barramento de endereços (A2 a A31 e BE0# a BE3#):

Os bits A2 a A31 ("ADDRESS") e BE0# a BE3# ("BYTE ENABLE") formam o barramento de endereços e fornecem os endereços físicos de memória e de I/O. O 80486DX é capaz de endereçar 4 Gbytes de memória física e 64 Kbytes de I/O.

Enquanto A2 a A31 identificam uma localização de endereço de 4 bytes, BE0# a BE3# identificam quais bytes nesta localização estão envolvidos no acesso corrente, sendo que BE0# está associado com o byte de dados menos significativo e assim por diante com os demais sinais de "byte enable" e os bytes de dados. As saídas BE0# a BE3# podem ser decodificadas para gerarem os sinais A0, A1 e BHE# que podem ser utilizados em sistemas de 8 e 16 bits.

Os bits de A4 a A31 tornam-se sinais de entrada para o 80486DX durante as invalidações das linhas do cache.

Barramento de dados (D0 a D31):

As linhas bidirecionais de D0 a D31 ("DATA") formam o barramento de dados de 32 bits do 80486DX. onde D0 é o bit menos significativo. Transferências de 8 e 16 bits também são possíveis através da ativação dos pinos de entrada BS8# ou BS16#.

Paridade (DP0 a DP3 e PCHK#):

Os pinos DP0 a DP3 ("DATA PARITY") correspondem às paridades pares do microprocessador, cada qual associado com um byte de dados (DP0 com o menos significativo e assim por diante). Cada paridade ativa significa que existe um número par de entradas em nível lógico alto no byte.

Os dados de paridade são gerados pelo 80486DX em todos os ciclos de escrita de dados e as informações de paridade par devem chegar ao 80486DX durante os ciclos de leitura, junto com as linhas de dados para garantir o correto estado das paridades a serem verificadas pelo microprocessador. Os valores lidos nestes pinos não afetam a execução do programa. O resultado desta verificação é sinalizado através do pino de saída PCHK# ("PARITY CHECK") após a operação de leitura dos códigos de instrução, de uma posição de memória ou de I/O, não ocorrendo durante os ciclos de reconhecimento de interrupção ("interrupt acknowledge").

A ativação do sinal PCHK# é a única consequência no caso do 80486DX detectar um erro de paridade. Caso este tipo de erro ocorra, a responsabilidade pelas devidas ações é do sistema.

Definição de ciclo do barramento (M/IO#, D/C#, W/R#, LOCK#, PLOCLK#):

Os pinos de saída M/IO# ("MEMORY/IO"), D/C# ("DATA/CONTROL") e W/R# ("WRITE/READ") definem o ciclo do barramento: M/IO# distingue entre ciclos de memória e de I/O, D/C# distingue entre ciclos de dados e de controle e W/R# distingue entre ciclos de escrita e leitura, conforme a tabela a seguir:

M/IO#	D/C#	W/R#	CICLO DE BARRAMENTO INICIALIZADO
0	0	0	"Interrupt acknowledge"
0	0	1	Ciclo de "halt"
0	1	0	Leitura de I/O
0	1	1	Escrita de I/O
1	0	0	Leitura do código de instrução
1	0	1	Reservado
1	1	0	Leitura de memória
1	1	1	Escrita de memória

Tabela 2.2 - Definição do ciclo de barramento.

A saída LOCK# ("LOCK") é ativada através da instrução LOCK ou quando o microprocessador está executando um outro tipo de instrução que resulte num ciclo de escrita logo após a leitura ("read-modify-write") para sinalizar ao sistema que os barramentos não poderão ser cedidos para outro dispositivo durante estes ciclos. O 80486DX reconhecerá a requisição de cessão do barramento pelo pino HOLD somente quando esta saída estiver inativa.

A saída PLOCK# ("PSEUDO-LOCK") é similar à saída LOCK#, porém somente é ativada para os ciclos de escrita e leitura de memória de operandos maiores do que 32 bits.

Controle do barramento (ADS#, RDY#):

A saída ADS# ("ADDRESS STATUS") indica que o endereço e os sinais de definição do ciclo do barramento são válidos e é utilizada pelo circuito externo ao 80486DX para identificar que o mesmo iniciou um novo ciclo de barramento.

A entrada RDY# ("READY") sinaliza ao microprocessador que o corrente ciclo de barramento não "burst" está completo. Durante uma operação de escrita, esta entrada informa que o sistema aceitou os dados do 80486DX, e numa operação de leitura, informa que o sistema forneceu os dados válidos para o microprocessador.

Controle do "burst" (BRDY#, BLAST#):

A entrada BRDY# ("BURST READY") possui a mesma função da entrada RDY# porém, somente para ciclos "burst". Se ambas retornarem simultaneamente, BRDY# é ignorada e o ciclo "burst" é abortado.

A saída BLAST# ("BURST LAST") indica que na próxima vez que BRDY# retornar, ela será tratada como se fosse o sinal RDY#, concluindo o ciclo de múltipla transferência de dados.

Sinais de interrupção (RESET, INTR, NMI):

A entrada RESET ("MICROPROCESSOR RESET") força o microprocessador a iniciar sua execução num estado conhecido.

A ativação da entrada INTR ("MASKABLE INTERRUPT") indica que uma requisição de interrupção externa foi requisitada. Esta interrupção será processada se o sinalizador ("flag") IF do 80486DX estiver ativo. Em resposta a esta requisição, o 80486DX gerará dois ciclos de barramento de reconhecimento da

mesma e no fim do segundo, o controlador externo de interrupção deverá gerar um número de 8 bits que será capturado pelo 80486DX. Este número é conhecido como vetor de interrupção e será utilizado pelo microprocessador para identificar qual interrupção externa foi requisitada. Existe ainda o pino de entrada NMI ("NON MASKABLE INTERRUPT") no qual um dispositivo externo causa uma interrupção não mascarada, ou seja, esta interrupção sempre será atendida e o vetor de interrupção é gerado internamente ao 80486DX.

Sinais de arbitração do barramento (BREQ, HOLD, HLDA, BOFF#):

O 80486DX sempre ativa o sinal de saída BREQ ("BUS REQUEST") toda vez que um ciclo de barramento está pendente internamente, ou seja, em todo início de ciclo de barramento junto com a saída ADS#, ou quando ciclos adicionais são necessários para completar a transferência de dados via BS8# ou BS16#.

A entrada HOLD ("BUS HOLD REQUEST") permite que outro dispositivo controle o barramento do 80486DX. Em resposta a esta entrada, o 80486DX gera a saída HLDA ("HOLD ACKNOWLEDGE") e coloca a maioria de seus pinos de saída e bidirecionais em alta impedância após completar o ciclo corrente, o ciclo "burst" ou o ciclo "locked", com exceção dos pinos BREQ, HLDA, PCHK# e FERR#. O microprocessador irá se manter neste estado até a desativação do sinal HOLD. A entrada BOFF# ("BACKOFF") é similar à entrada HOLD, porém a diferença entre elas é que o 80486DX cede seu barramento imediatamente após a requisição pela entrada BOFF#, sem esperar a conclusão do corrente ciclo de barramento.

Invalidação do cache (AHOLD, EADS#):

As entradas AHOLD ("ADDRESS HOLD") e EADS# ("VALID EXTERNAL ADDRESS") são utilizadas durante os ciclos de invalidação do cache. AHOLD é a requisição somente do barramento de endereços do 80486DX, permitindo que outro dispositivo acesse-o, realizando deste maneira um ciclo de invalidação do cache interno. Enquanto esta entrada estiver ativa, somente o barramento de endereços do 80486DX permanece em alta impedância e os demais permanecem ativos.

A entrada EADS# indica ao 80486DX que um endereço externo nos pinos de endereços é válido e este endereço será checado com o conteúdo corrente do cache para sinalizar sua invalidação.

Controle do cache (KEN#, FLUSH#):

A entrada KEN# ("CACHE ENABLE") é responsável pela habilitação do cache e é utilizada para determinar se o dado retornado pelo ciclo corrente pode fazer parte do cache ou não, enquanto a entrada FLUSH# ("CACHE FLUSH") anula todo o conteúdo do cache.

Cache de página (PWT, PCD):

As saídas PCD ("PAGE CACHE DISABLE") e PWT ("PAGE WRITE-THROUGH") correspondem aos dois bits de atributo da entrada da tabela de paginação do 80486DX. Quando a paginação está habilitada, estes dois bits correspondem aos bits 3 e 4 da entrada da tabela, respectivamente, e quando está desabilitada, ou para ciclos que não são paginados, mesmo quando a paginação está habilitada (por exemplo: ciclos de I/O), correspondem aos bits 3 e 4 do registrador de controle 3 do 80486DX, respectivamente.

A saída PCD é determinada pelo bit CD ("cache disable") do registrador de controle 0 (CR0) do 80486DX: quando igual a "1", força esta saída a permanecer em nível lógico alto, caso contrário, em nível lógico baixo. O propósito desta saída é o de indicar se a paginação ocorre com capacidade de cache ou não. O 80486DX não realizará nenhum "cache fill" para nenhuma página quando este bit estiver em "1".

A saída PWT corresponde ao bit de "write-back" e pode ser utilizada por um cache externo ao 80486DX.

Erro numérico (FERR#, IGNNE#):

O 80486DX ativa a saída FERR# ("FLOATING POINT ERROR") toda vez que um erro não mascarável de ponto flutuante for encontrado. Este pino é similar ao pino ERROR# do co-processador aritmético 80387 e pode ser utilizado pela lógica externa para informar que houve um erro de ponto flutuante em sistemas compatíveis AT que utilizam o microprocessador 80486DX. Porém, se a entrada IGNNE# ("IGNORE NUMERIC ERROR") estiver ativa, o 80486DX irá ignorar o eventual erro numérico por ele detectado e continuará a sua execução.

Controle do tamanho do barramento (BS16# e BS8#):

Estas entradas BS16# ("BUS SIZE 16") e BS8# ("BUS SIZE 8") permitem que barramentos de dados externos de 16 ou 8 bits sejam suportados pelo 80486DX com um mínimo de hardware necessário. O 80486DX sempre amostra

estas entradas, determinando o tamanho do barramento no qual ele operará. Quando BS16# ou BS8# é ativada, somente 16 ou 8 bits de dados, respectivamente, devem ser válidos. Se ambas são ativadas simultaneamente, o barramento selecionado é o de 8 bits. Se nenhuma for, o microprocessador poderá realizar as transferências em 32 bits. Assim, o 80486DX converterá o dado requisitado num número apropriado de transferências de dados de tamanho menor em bits. Os sinais BE0# a BE3# também serão alterados apropriadamente.

Mascaramento da linha de endereço 20 (A20M#):

Quando a entrada A20M# ("ADDRESS BIT 20 MASK") está ativa, o microprocessador 80486DX mascara fisicamente a linha de endereço 20 antes de procurar um dado no cache ou na memória externa. Desta maneira, o 80486DX emula o endereçamento físico de 1 Mbyte do 8086. Esta função foi implementada originalmente nos microcomputadores compatíveis AT.

Alimentação (VCC e VSS):

O 80486DX é alimentado por uma única tensão de alimentação de 5 Volts com tolerância de 5%, aplicados nos seus pinos VCC (+5V) e VSS (terra).

OPERAÇÕES DO 80486DX

Reset

O 80486DX possui internamente um autoteste (BIST - "built in self test") que pode ser executado durante o reset se na borda de descida do pino RESET, o pino AHOLD estiver ativado. O resultado deste teste encontra-se no registrado EAX, informando se houve sucesso quando o conteúdo deste registrador for igual a 0, ou falha se for diferente de 0. O registrador EDX sempre reflete o identificador do componente logo após o reset, independentemente do BIST: o byte mais significativo do registrador DX conterá 04 e o menos significativo conterá o identificador da revisão do microprocessador.

Após o reset, o 80486DX irá executar as instruções contidas na localização de memória FFFFFFF0H. Quando a primeira instrução de "jump" ou "call" for executada, as linhas de endereços de A20 a A31 irão para 0 e o microprocessador somente executará instruções no primeiro Mbyte de memória física. Isto

permite que nos projetos de sistemas, utilize-se memória ROM no topo da memória física para inicializar o sistema.

Quando ocorre o reset, o 80486DX interrompe a sua execução e a sua atividade no barramento local.

Barramento

Os ciclos de barramentos do 80486DX podem acessar espaços de memória física na faixa de endereço 00000000H até FFFFFFFFH (4 Gbytes) ou de I/O do endereço 00000000H até 0000FFFFH (64 Kbytes), onde os dispositivos periféricos podem ser mapeados em memória, ou em I/O, ou em ambos.

Todas as transferências de dados ocorrem como resultado de um ou mais ciclos de barramentos, onde bytes, "words" (16 bits) ou "double-words" (32 bits) de dados podem ser transferidos sem restrições de alinhamento no endereço físico, ou seja, não é obrigatório que as transferências de "double-words" ocorram sempre em endereços múltiplos de 4 e também não é obrigatório que as transferências de "words" (16 bits) ocorram em endereços pares. Como consequência disso, até três ciclos de barramentos podem ser requisitados para a transferência de dados.

Os sinais de endereços do 80486DX são divididos em duas componentes: A2 a A31, que são as linhas de endereços mais significativas e BE0# a BE0#, que são os sinais que selecionam os quatro bytes de dados no barramento de 32 bits, conforme tabela a seguir:

SINAL	BARRAMENTO DE DADOS ASSOCIADO
BE0#	D0 a D7 (byte 0 - menos significativo)
BE1#	D8 a D15 (byte 1)
BE2#	D16 a D23 (byte 2)
BE3#	D24 a D31 (byte 3 - mais significativo)

Tabela 2.3 - Seleção dos bytes de dados

Os bits de endereços A0 e A1 ou os sinais BLE# e BHE#, utilizados para endereçar sistemas de 16 ou 8 bits podem ser gerados, se necessários, a partir dos sinais BE0# a BE3#, conforme tabela a seguir, onde "0" significa nível lógico baixo, "1" significa nível lógico alto e "x" significa irrelevante:

SINAIS DO 80486DX				SINAIS P/ O SISTEMA			OBSERVAÇÕES
BE3#	BE2#	BE1#	BE0#	A1	BHE#	BLE# (A0)	
1	1	1	1	x	x	x	bytes inativos
1	1	1	0	0	1	0	
1	1	0	1	0	0	1	
1	1	0	0	0	0	0	
1	0	1	1	1	1	0	
1	0	1	0	x	x	x	situação inexistente
1	0	0	1	0	0	1	
1	0	0	0	0	0	0	
0	1	1	1	1	0	1	
0	1	1	0	x	x	x	situação inexistente
0	1	0	1	x	x	x	situação inexistente
0	1	0	0	x	x	x	situação inexistente
0	0	1	1	1	0	0	
0	0	1	0	x	x	x	situação inexistente
0	0	0	1	0	0	1	
0	0	0	0	0	0	0	

Tabela 2.4 - Bits de endereços para sistemas de 16 ou 8 bits.

Além dos pinos BE0# a BE3#, o 80486DX possui os pinos BS8# e BS16# que controlam o barramento, permitindo conectar diretamente a ele memórias e dispositivos de I/O de 8 e 16 bits. As transferências de dados com dispositivos de 32, 16 ou 8 bits são suportadas através da determinação dinâmica do tamanho do barramento a cada ciclo. Um circuito externo de decodificação de endereços pode ativar o pino BS16# para dispositivos de 16 bits ou o pino BS8# para dispositivos de 8 bits. Se ambos não são ativados, a transferência ocorre em 32 bits. Como consequência da ativação destes pinos, o 80486DX realizará ciclos adicionais de barramentos para completar as requisições internas maiores do que 8 ou 16 bits. Por exemplo, uma transferência de 32 bits será convertida em duas de 16 (ou três, se o dado está desalinhado) quando BS16# é ativado ou em quatro de 8 bits quando BS8# é ativado.

O 80486DX gerará apropriadamente os sinais BE0# a BE3# quando BS8# ou BS16# forem ativados e A2 a A31 não mudará se o acesso for numa área alinhada de 32 bits.

Esta característica do 80486DX de determinação dinâmica do tamanho do barramento é significativamente diferente do 80386, pois o 80486DX requer

que os bytes de dados sejam gerados nos pinos de dados endereçados. Por exemplo, quando o 80486DX deseja fazer uma operação de leitura de 32 bits de dados alinhados e o pino BS16# é ativado, ele lerá os dois bytes mais significativos nos pinos D16 a D31 e os menos significativos nos pinos D0 a D15. O 80386 sempre escreve ou lê os dados nos 16 bits de dados menos significativos quando BS16# é ativado. A tabela a seguir, mostra as linhas do barramento de dados válidos pelo 80486DX para cada combinação válida dos sinais BE0# a BE3#, BS8# e BS16#.

BE3#	BE2#	BE1#	BE0#	PINOS DE DADOS VÁLIDOS		
				SEM BS8#/BS16#	COM BS8#	COM BS16#
1	1	1	0	D7 a D0	D7 a D0	D7 a D0
1	1	0	0	D15 a D0	D7 a D0	D15 a D0
1	0	0	0	D23 a D0	D7 a D0	D15 a D0
0	0	0	0	D31 a D0	D7 a D0	D15 a D0
1	1	0	1	D15 a D8	D15 a D8	D15 a D8
1	0	0	1	D23 a D8	D15 a D8	D15 a D8
0	0	0	1	D31 a D8	D15 a D8	D15 a D8
1	0	1	1	D23 a D16	D23 a D16	D23 a D16
0	0	1	1	D31 a D16	D23 a D16	D31 a D16
0	1	1	1	D31 a D24	D31 a D24	D31 a D24

Tabela 2.5 - Bytes válidos na ativação do BS16# ou BS8#.

As figuras a seguir, mostram, respectivamente, como devem ser feitos o endereçamento e a interface do barramento de dados do 80486DX com dispositivos de 16 e 8 bits:

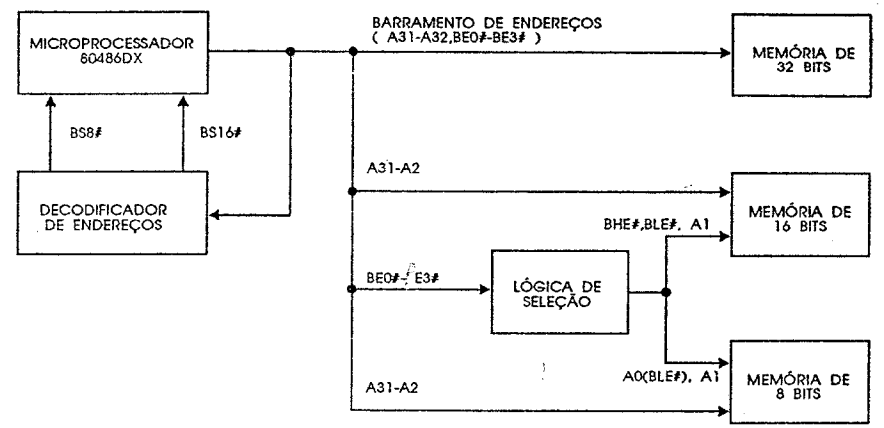


Figura 2.5 - Endereçamento com dispositivos de 16 e 8 bits.

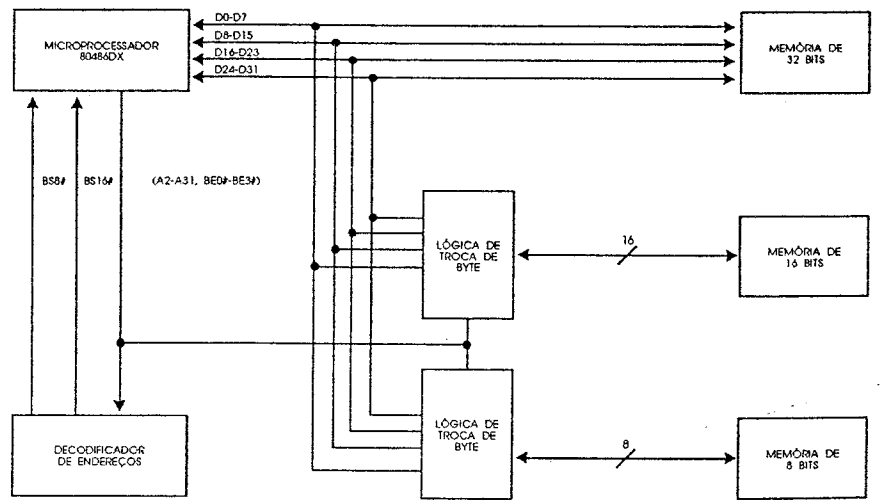


Figura 2.6 - Interface com dispositivos de 16 e 8 bits.

Interrupção

O 80486DX possui duas entradas de interrupção assíncronas: INTR que é uma entrada mascarável, ou seja, pode ser habilitada ou desabilitada via software e NMI que é uma entrada não mascarável.

A entrada INTR é sensível a nível e assim deve ser mantida até o microprocessador reconhecê-la, gerando um ciclo de reconhecimento de interrupção ("interrupt acknowledge").

A entrada NMI, em oposição à entrada INTR, é sensível à borda de subida para gerar a requisição de interrupção e não necessita permanecer ativa até ser atendida, bastando assim permanecer por um único período de clock para atender as exigências de tempos do microprocessador. Esta entrada somente é mascarada internamente ao 80486DX, toda vez que a rotina de NMI é inicializada e assim permanecerá até que a instrução IRET de retorno da interrupção for executada. Desta maneira, evita-se que outra NMI seja inicializada enquanto a corrente não estiver concluída.

Cache interno

Com o propósito de obter alta performance, o 80486DX possui internamente um cache de 8 Kbytes transparente para os programas, mantendo compatibilidade binária com as arquiteturas das gerações anteriores ao 80486DX. Este cache possui vários modos de operação, oferecendo grande flexibilidade, onde as áreas de memória podem ser definidas com ou sem capacidade de cache via software ou através de hardware externo. Os protocolos para atualizações e invalidações da linha de cache são implementadas por hardware.

O cache do 80486DX é utilizado tanto para acessos de instruções, como para dados e atua nos endereços físicos. É organizado em "4-way set associative" e cada linha do cache possui 16 bytes (maiores detalhes sobre cache podem ser encontrados no apêndice "A"). Os 8 Kbytes de memória cache são fisicamente divididos em 4 blocos de 2 Kbytes, cada um contendo 128 linhas, conforme figura a seguir. Associados com cada bloco, existem 128 "tags" de 21 bits e para cada linha no cache existe um bit de validade.

A estratégia de escrita do cache é a "write-through". Todas as escritas gerarão um ciclo de escrita no barramento externo do 80486DX, adicionalmente à escrita da informação no cache interno, se a escrita foi do tipo "cache hit". Uma escrita num endereço não contido no cache será somente efetivada na memória externa.

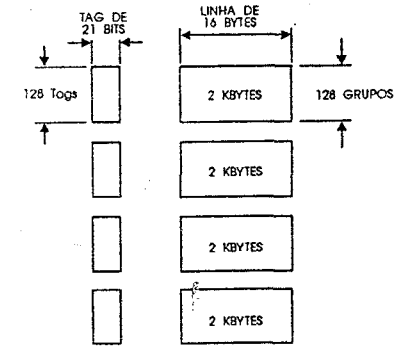


Figura 2.7 - Cache do 80486DX.

Diagrama de estados do barramento

O diagrama de estados do barramento e a descrição destes estados são mostrados a seguir:

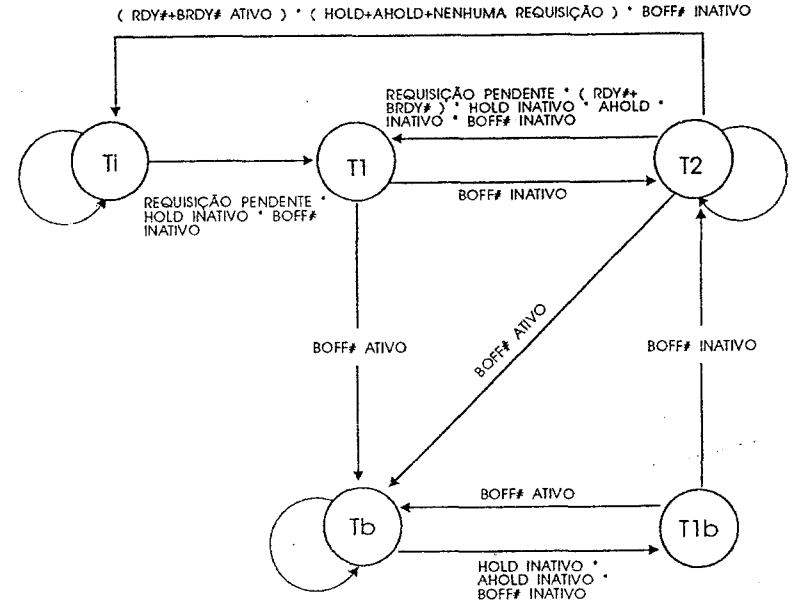


Figura 2.8 - Diagrama de estados do barramento.

Significado dos Estados:

Ti — O barramento se encontra inativo ("idle"): os pinos de endereços e de estado do barramento contêm valores indefinidos ou estão em alta impedância.

T1 — Primeiro clock de um ciclo de barramento: o endereço válido e o estado do barramento são gerados e o pino ADS# é ativado.

T2 — Segundo e subsequentes clocks de um ciclo de barramento: os dados são gerados se o ciclo for de escrita, ou são aguardados pelo 80486DX se for um ciclo de leitura e os pinos RDY# e BRDY# são amostrados.

T1b — Primeiro clock de um ciclo de barramento reinicializado e que havia sido abortado: o endereço válido e o estado do barramento são gerados e o pino ADS# é ativado. Este estado não é distinguível do T1 externamente.

Tb — Segundo e subsequentes clocks de um ciclo de barramento abortado.

CARTAS DE TEMPOS

Um ciclo de barramento do 80486DX dura no mínimo dois períodos de clock e inicia com a saída ADS# ativa no primeiro. Os dados são transferidos do/para o microprocessador durante o ciclo de dado e um ciclo de barramento contém um ou mais ciclos de dados.

A seguir, são apresentados os principais ciclos de transferências de dados entre o microprocessador e o sistema.

Ciclo básico

O mais rápido ciclo de barramento não "burst" que o 80486DX suporta dura dois clocks. Este ciclo é chamado de 2-2 porque tanto a leitura como a escrita duram dois clocks, onde o primeiro 2 refere-se à leitura e o segundo à escrita.

O 80486DX inicia o seu ciclo ativando o sinal ADS# na borda de subida do primeiro clock, indicando que um novo endereço e um novo ciclo são válidos.

A entrada RDY# é retornada pelo sistema no segundo clock, indicando que o sistema está fornecendo dados válidos em resposta a uma leitura, ou o

sistema os aceitou em resposta a uma escrita do 80486DX. No fim do segundo clock, o ciclo se completa quando este pino é amostrado em nível lógico baixo.

A saída BLAST# é ativada pelo 80486DX para indicar que a transferência de dados está completa. Numa transferência simples de um único ciclo, é ativada durante o segundo clock e numa transferência múltipla, o 80486DX ativa esta saída no último ciclo.

A saída PCHK# de paridade é gerada pelo microprocessador um clock depois de terminar o ciclo de leitura, indicando o estado da paridade do dado amostrado no fim do clock anterior, podendo ser utilizada pelo sistema.

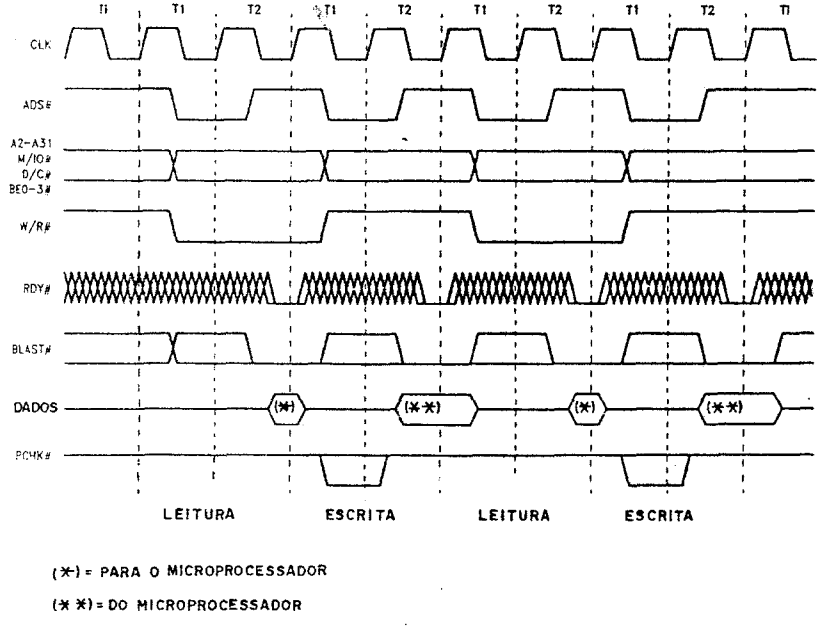


Figura 2.9 - Ciclo básico.

Ciclo básico com "wait-state"

O sistema pode inserir "wait-states" no ciclo básico 2-2 através do RDY# inativo no fim do segundo clock. Mantendo-o inativo, qualquer número de "wait-states" pode ser adicionado ao ciclo de barramento do 80486DX.

A entrada BRDY# deve permanecer inativa em todas as bordas do clock, enquanto RDY# estiver inativo, para a correta operação deste ciclo básico não "burst".

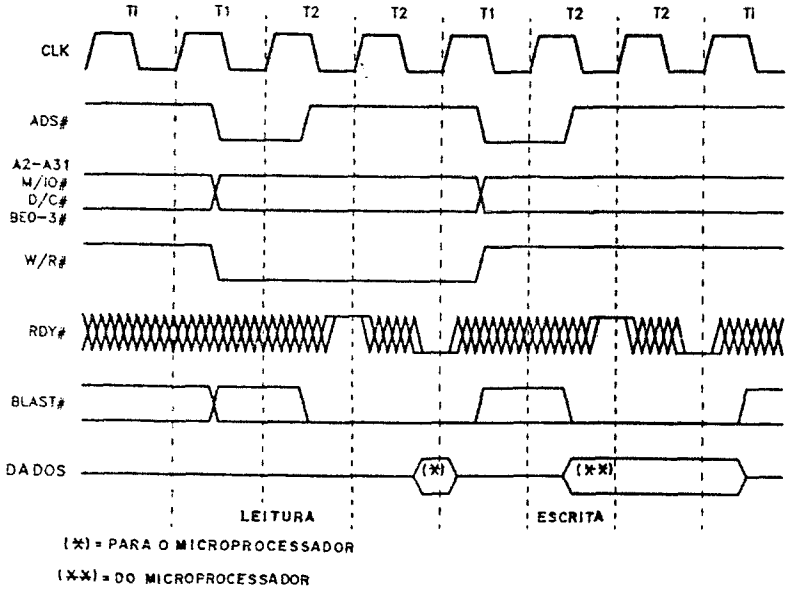


Figura 2.10 - Ciclo básico com "wait state".

Ciclo "burst"

O 80486DX pode gerar ciclos "burst" para qualquer requisição de barramento que requeira mais do que um ciclo de dados. Durante estes ciclos, um novo dado é armazenado no 80486DX a cada clock ao invés de capturá-los a cada dois, como acontece no ciclo básico. O ciclo "burst" mais rápido requer dois clocks para o primeiro dado e os subsequentes dados retornam a cada clock.

Durante uma escrita, o 80486DX é capaz de realizar um ciclo "burst" a uma taxa máxima de 32 bits por ciclo. Estas escritas somente ocorrerão se o pino BS8# ou o BS16# forem ativados. Por exemplo, o microprocessador pode realizar uma escrita "burst" de quatro operandos de 8 bits ou de dois operandos de 16 bits num único ciclo "burst", porém ele não é capaz de realizar escritas múltiplas de 32 bits num único ciclo "burst".

Este ciclo "burst" inicia com o microprocessador ativando a saída ADS#, da mesma maneira que no ciclo básico. O 80486DX informa que o ciclo corrente é "burst" através da saída BLAST# inativa no segundo clock do ciclo. O sistema deve ativar a entrada BRDY# para informar que o dado presente nos pinos do microprocessador é válido na leitura, ou que o sistema aceitou o dado em resposta à escrita do 80486DX durante o ciclo "burst".

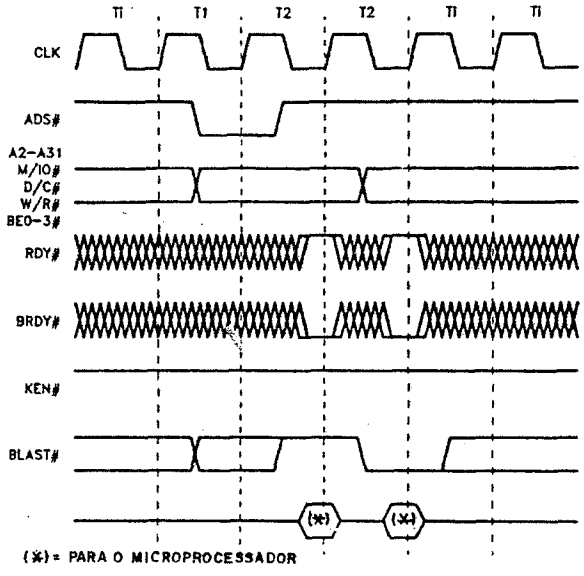


Figura 2.11 - Ciclo "Burst".

Ciclo "cache"

Uma leitura de memória pode corresponder a uma operação de "cache fill". Para tanto, o sistema deve ativar a entrada KEN# um clock antes do RDY# ou BRDY# durante o primeiro ciclo de transferência de dados do barramento externo para o microprocessador. O 80486DX converterá somente as leituras de memória ou buscas dos códigos de operação em "cache fill". A entrada KEN# é ignorada durante ciclos de escrita ou de I/O. As escritas em memória somente serão armazenadas no cache se a escrita for do tipo "cache hit".

Para que uma leitura de memória resulte num "cache line fill" é necessário que as seguintes condições sejam obedecidas:

- O pino KEN# deve ser ativado um clock antes do RDY# ou BRD# retornarem no primeiro ciclo de dados;
- O ciclo deve ser do tipo que pode internamente ocorrer cache (leituras de memória "iocked", ciclos de I/O e ciclos de reconhecimento de interrupção não são);
- O entrada da tabela de página deve conter o bit PCD em 0;

- O bit CD do registrador CR0 deve ser 0.

O sistema pode determinar quando o 80486DX está transformando uma leitura de memória num "cache fill", examinando os pinos KEN#, M/IO#, D/C#, W/R#, LOCK# e PCD.

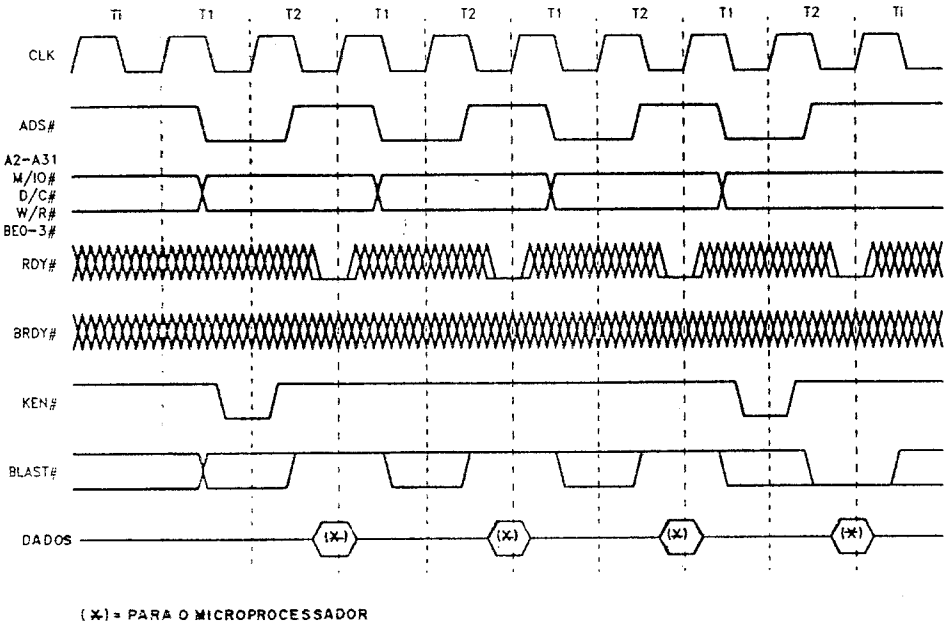


Figura 2.12 - Ciclo "CACHE".

Ciclo de barramento de 16 e 8 bits

O 80486DX suporta tanto barramentos externos de 32 bits quanto de 16 e 8 bits através das entradas BS16# e BS8#. A temporização destas entradas é a mesma da entrada KEN# e devem ser ativadas antes do primeiro RDY# ou BRDY# ativos.

Ativando BS16# ou BS8#, o 80486DX é forçado a realizar ciclos adicionais para completar o que teria sido realizado num único ciclo de 32 bits.

A figura a seguir, mostra um exemplo no qual a entrada BS8# força o 80486DX a realizar dois ciclos extras para completar a transferência de 24 bits de dados. O sistema ativa esta entrada, indicando que somente 8 bits podem ser suportados por ciclo.

Os ciclos extras resultantes da ativação das entradas BS16# e BS8# devem ser encarados como ciclos de barramentos independentes. O 80486DX manterá a saída BLAST# inativa até o último ciclo antes que a transferência esteja completa.

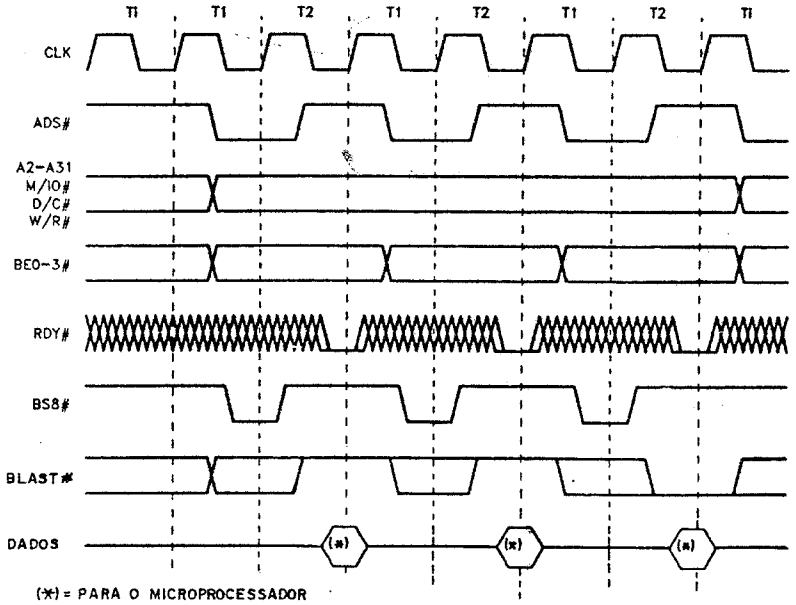


Figura 2.13 - Ciclo de barramento de 8 bits.

Ciclo "locked"

Estes ciclos são gerados através da instrução LOCK ou por qualquer instrução que resulte numa operação do tipo "read-modify-write" na qual o microprocessador lê uma variável da memória externa, trata-a e a altera como por exemplo, as instruções de teste e alteração de bits, tais como BTS, BTR e BTC. O sistema é informado sobre este ciclo e que o microprocessador não irá ceder o seu barramento para outro dispositivo através da saída LOCK# ativa.

Os ciclos "locked" são gerados automaticamente durante certas transferências de dados, tais como quando se executa a instrução XCHG, quando um

dos operandos é referenciado a uma posição de memória, ou durante um ciclo de reconhecimento de interrupção.

A saída LOCK# torna-se ativa junto com a ativação da saída ADS# no primeiro ciclo de leitura e assim permanece até RDY# retornar para o último ciclo de escrita.

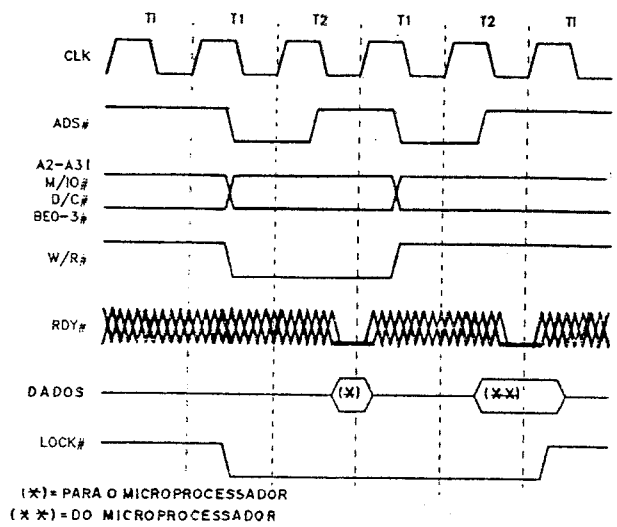


Figura 2.14 - Ciclo "LOCKED".

Ciclo "hold/holda"

O 80486DX possui um protocolo para cessão do seu barramento através da entrada HOLD de requisição desta cessão e da saída HLDA do reconhecimento da mesma. A ativação do pino HOLD indica que outro dispositivo deseja controlar o barramento do 80486DX. O microprocessador responderá colocando o barramento em alta impedância e ativará a saída HLDA quando o ciclo corrente ou a sequência de ciclos "locked" estiver concluída, como pode ser visto na figura a seguir.

Os pinos que permanecem em alta impedância durante o "bus hold" são: BE0# a BE3#, PCD, PWT, W/R#, D/C#, M/IO#, LOCK#, PLOCK#, ADS#, BLAST#, D0 a D31, A2 a A31 e DP0 a DP3.

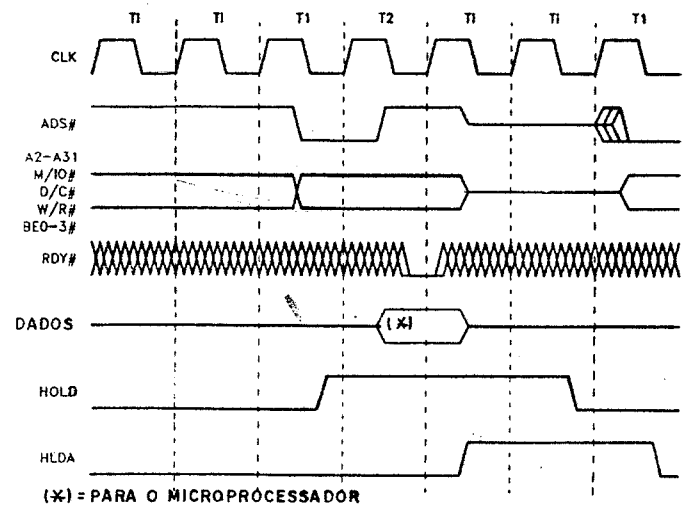


Figura 2.15 - Ciclo "HOLD/HOLDA".

Ciclo de reconhecimento da requisição de interrupção

O 80486DX gera ciclos de reconhecimento de requisições de interrupções mascaráveis em resposta à ativação da entrada INTR. Como pode ser visto na figura a seguir, em cada ciclo de reconhecimento são gerados dois ciclos "locked", onde os dados retornados durante o primeiro são ignorados e o vetor de interrupção é retornado no segundo ciclo nos 8 bits menos significativos do barramento de dados. O 80486DX possui 256 possíveis vetores de interrupção distintos. O estado da linha de endereço A2 distingüe entre o primeiro e o segundo ciclos "locked" do reconhecimento da requisição de interrupção: em "1" corresponde ao primeiro ciclo e em "0" corresponde ao segundo. O ciclo é terminado quando o sistema retorna RDY# ou BRDY# ativo. O 80486DX gera automaticamente 4 clocks entre o primeiro e o segundo ciclos para dar tempo suficiente para o controlador de interrupções colocar no barramento de dados, o vetor de interrupção válido.

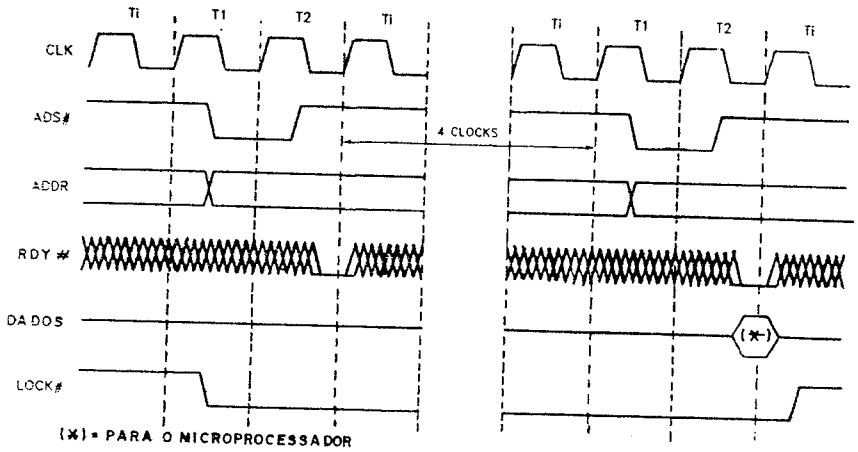


Figura 2.16 - Ciclo de reconhecimento de interrupção.

Ciclo de barramento reinicializado (BOFF# ativo)

Num sistema multiprocessado, outro dispositivo pode querer utilizar imediatamente o barramento do 80486DX. Isto pode ser obtido através da ativação da entrada BOFF# e que resultará na cessão do barramento pelo 80486DX no próximo clock, diferentemente da entrada HOLD pela qual o microprocessador cederá seu barramento apenas quando concluir o ciclo corrente. Quando o sistema desativar a entrada BOFF#, o 80486DX reinicializará o seu ciclo de barramento após este outro dispositivo ter completado sua utilização do barramento.

Este tipo de ciclo inicia com o sistema ativando a entrada BOFF#. O 80486DX amostra esta entrada a cada clock. No próximo ciclo da ativação desta entrada, o 80486DX colocará imediatamente em alta impedância os seus pinos de endereços, dados e estados, como pode ser visto na figura a seguir. Assim, o corrente ciclo de barramento é abortado e qualquer dado retornado para o microprocessador é ignorado. Nesta situação, o dispositivo que ativou BOFF# pode realizar qualquer ciclo no barramento do 80486DX.

Quando a entrada BOFF# é desativada, o 80486DX reinicializa seu barramento, gerando seus barramentos e ativando a saída ADS# e o ciclo de barramento continua como um ciclo usual.

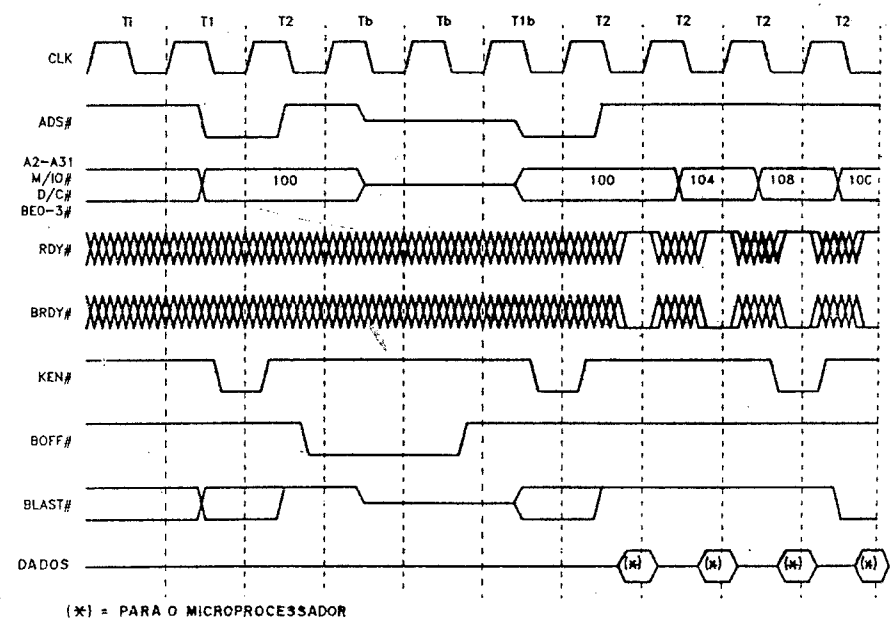


Figura 2.17 - Ciclo de barramento reinicializado.

DIFERENÇAS DE PINAGEM ENTRE O 80486DX E O 80486DX2

Como pode ser observado, a pinagem do 80486DX2 PGA é muito similar à do 80486DX PGA. As únicas diferenças são alguns pinos que no 80486DX são não conectados e no 80486DX2 são utilizados para prover aspectos de testabilidade dos componentes e da placa de circuito impresso, onde se encontra o 80486DX2 e também para o mesmo entrar no modo de "power down". Caso não se utilize destas características de testabilidade, os mesmos podem permanecer não conectados, pois as entradas possuem resistores de "pull-up" e a saída ficará sem função.

NUMERAÇÃO DO PINO	FUNÇÃO DO PINO	
	80486DX	80486DX2
A3	NC	TCK
A14	NC	TDI
B14	NC	TMS
B16	NC	TDO
C11	NC	UP#

Tabela 2.10 - Diferenças de pinagem entre o 80486DX e o 80486DX2.

DESCRIÇÃO DOS NOVOS PINOS

Os pinos de entrada TCK, TDI e TMS e o pino de saída TDO do 80486DX2 são pinos que provêem aspectos de testabilidade compatível com o padrão IEEE Std. 1149.1 e que cujos correspondentes no 80486DX são pinos não conectados. A lógica de teste associada a estes pinos garantem o correto funcionamento e as interconexões dos diversos componentes interligados na placa de circuito impresso. Esta lógica de teste chama-se "Boundary Scan".

O pino de entrada UP# ("UPGRADE PRESENT") é ativo em nível lógico baixo e o seu correspondente pino no 80486DX é um pino não conectado. Quando este pino é ativado, força as saídas do 80486DX2 a irem para o estado "3-state".

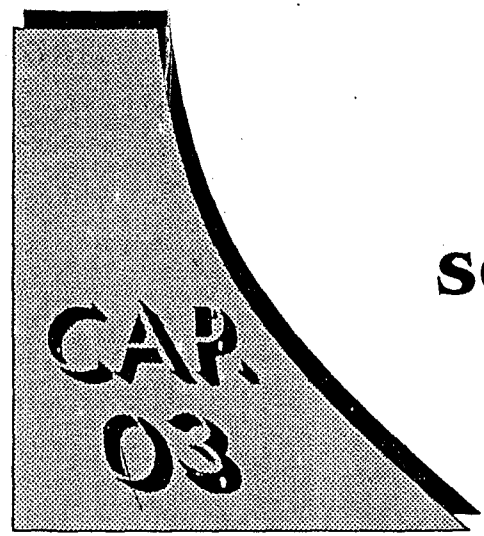
O pino de entrada CLK do 80486DX2 é similar ao mesmo do 80486DX, provendo a temporização fundamental para a unidade de interface com o barramento e portanto, do ponto de vista do mundo exterior, ao mesmo nada muda, incluindo os acessos a dispositivos de I/O e à memória, porém diferentemente do 80486DX, é multiplicado por dois internamente para gerar a freqüência interna de operação (daí a razão do aumento de performance do 80486DX2 em relação ao 80486DX).

SUBSTITUIÇÃO DO 80486DX PELO 80486DX2

A substituição do 80486DX pelo 80486DX2 nos sistemas resulta num aumento médio de performance de 70%, porém alguns detalhes devem ser inicialmente verificados para se certificar de que o projeto da placa CPU está adequado para receber o 80486DX2. Estes detalhes são:

- O BIOS, que inicialmente foi projetado para o 80486DX, pode ter alguns "loops" de temporização. Como o 80486DX2 executa as instruções duas vezes mais rápido que o 80486DX, estes "loops" podem não ser longos o suficiente para os resultados esperados. A maioria destes "loops" já foram removidos dos BIOS padrões, porém existem algumas versões que necessitam ser atualizadas.

- Como a pastilha do 80486DX2 opera duas vezes mais rápido que a do 80486DX quando ambos operam no mesmo clock, o 80486DX2 consome mais energia e gera mais calor do que o 80486DX. Portanto, é necessário verificar se a refrigeração e a fonte de alimentação são adequadas. É recomendado o uso de dissipador térmico com o 80486DX2.



SOFTWARE

1. CONCEITOS BÁSICOS

NOTAÇÕES CONVENCIONAIS

Este capítulo se utiliza de notações especiais para referenciar-se aos formatos das estruturas de dados, às representações das instruções e aos números hexadecimais. A leitura destes conceitos será útil para a interpretação do presente capítulo.

Ordenação de Bit e de Byte

Nas ilustrações de estruturas de dados em memória, o endereço menor aparece na parte de baixo da figura e o incremento dos endereços é feito em direção à parte de cima. As posições dos bits são numeradas da direita para a esquerda e o valor numérico do mesmo é dado por dois elevado à potência da posição do bit. O processador 80486 é uma máquina do tipo "little endian". Isto significa que os bytes de uma word são numerados a partir do byte menos significativo. A figura a seguir, ilustra essas convenções.

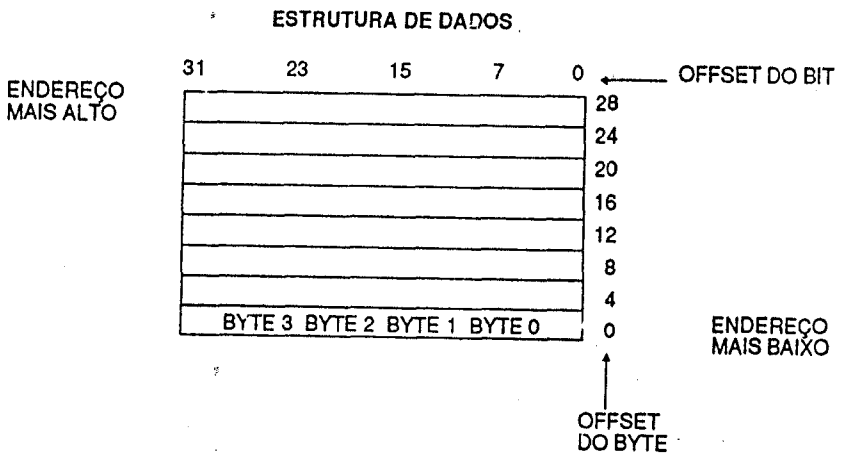


Figura 3.1 - Ordenação de bits e de bytes.

Bits Indefinidos e Compatibilidade com Softwares

Em muitos desenhos de registradores e de memória, alguns bits aparecem como "reservados". Quando bits são indicados como indefinidos ou reservados, é fundamental para a compatibilidade com futuras gerações de processadores que o software trate estes bits como tendo um efeito futuro que, por ora, é desconhecido. Para lidar com esses bits, as recomendações a seguir devem ser observadas.

- Não dependa de nenhum estado específico de bits reservados ao testar os registradores ou as posições de memória que os contenham. Mascare estes bits antes de testá-los.
- Não dependa dos estados destes bits quando armazenar informações nos registradores ou na memória.
- Não dependa da preservação dos dados escritos nestes bits, pois eles podem vir a se alterar independente do seu acesso.
- Quando carregar um registrador, observe na documentação, qual o valor que deve ser escrito nos bits reservados, se for indiferente, opte por fazer leitura prévia e restaurar este valor na operação de escrita.

Operandos de Instrução

Quando as instruções são representadas simbolicamente, um subconjunto da linguagem assembly para o processador 80486 é utilizado. Neste subconjunto, uma instrução tem o seguinte formato:

label:mneumônico argumento1,argumento2,argumento3

onde:

- label é um identificador que é seguido por dois pontos.
- mneumônico é um nome reservado para uma classe de códigos de operação (opcodes) que possuam a mesma função.
- Os operandos argumento1, argumento2 e argumento3 são opcionais. Dependendo do código da operação, pode-se ter de zero a três operandos. Quando presentes, eles podem assumir ou a forma literal ou identificadora para os itens de dados. As identificadoras de operandos são nomes reservados de registradores, ou assumidos como itens de dados declarados (definidos) em outra parte do programa (que pode não ser mostrada no exemplo).

Quando dois operandos aparecem numa instrução aritmética ou lógica o operando da direita é a fonte (origem) e o operando da esquerda é o destino. Algumas linguagens assembly colocam a fonte e o destino em ordem inversa a essa.

Por exemplo:

ARMAZENA: MOV EAX,VALOR1

Neste exemplo, ARMAZENA é label, MOV é o mneumônico identificador do código de operação, EAX é o operando de destino e VALOR1 é o operando fonte.

Números Hexadecimais

Os números com base 16 são representados por uma string de dígitos hexadecimais seguidos pelo caractere "H". Dígito hexadecimal é um caracter proveniente do conjunto (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). O caracter zero é mostrado antes dos símbolos alfabéticos para indicar que se trata de número; por exemplo: 0FH é o equivalente ao número 15 decimal.

Endereçamento Segmentado

O processador 80486 usa o endereçamento de bytes. Isto significa que a memória está organizada como uma seqüência de bytes. Quando um ou mais bytes estão sendo acessados, um número é usado para endereço de memória. A memória que pode ser endereçada com este número é chamada de espaço de endereço.

O 80486 também aceita o endereçamento segmentado. Esta é a forma de endereçamento na qual um programa pode ter muitos espaços de endereços independentes, chamados de segmentos. Por exemplo, um programa pode manter seus códigos (instruções) e stack em segmentos separados. Endereços de códigos sempre se refeririam ao espaço de códigos e endereços de stacks ao espaço de stack. Um exemplo de notação usada para mostrar endereços segmentados é mostrado abaixo:

CS:EIP

Este exemplo refere-se a um byte dentro do segmento de códigos. O número do byte está armazenado no registrados EIP.

Exceções

Exceção é um evento que ocorre quando uma instrução causa erro. Por exemplo, tentativa de divisão por zero gera exceção. Existem muitos tipos de exceções, e alguns deles podem fornecer códigos de erros. Um código de erro reporta informações adicionais sobre o problema ocorrido e somente são produzidos por exceções. Um exemplo de notação usada para mostrar exceção e código de erro é apresentado abaixo:

#PF(código da falha).

Este exemplo refere-se à exceção de Page fault (falha de página) sob condições onde é reportado código de erro que identifica a falha. Em algumas condições, exceções podem não estar aptas a reportar um código de erro com precisão. Neste caso, o código será zero, como mostrado abaixo:

#PF(0).

TIPOS DE DADOS

Os principais tipos de dados são os bytes, as words e as doublewords (figura 3.2). Um byte são oito bits. E os bits são numerados de 0 a 7, o bit 0 é o menos significativo (LSB - least significant bit).

Word é uma seqüência de dois bytes que ocupam dois endereços consecutivos quaisquer. Uma word tem, portanto, 16 bits. Os bits de uma word são numerados de 0 a 15. Novamente, o bit 0 é o menos significativo (LSB). O byte que contém o bit zero da word é chamado de low byte (byte baixo) e o que contém o bit 15 é chamado de high byte (byte alto). No 80486, o low byte é armazenado no endereço mais baixo e seu endereço é também o endereço da word. O endereço do high byte é utilizado somente quando a metade superior da word estiver sendo endereçada separadamente da parte de baixo.

Doubleword é uma seqüência de quatro bytes que ocupam quatro endereços consecutivos quaisquer. Uma doubleword contém 32 bits. Os bits de uma doubleword são numerados de 0 a 31, o bit 0 é o menos significativo. A word que contém o bit 0 da doubleword é chamada de low word e a que contém o bit 31 é chamada de high word. A low word está armazenada nos dois bytes cujos endereços são os menores. O endereço do byte mais baixo é considerado o endereço da doubleword. Os endereços mais altos são utilizados somente quando a parte de cima da doubleword é endereçada separadamente da parte de baixo, ou quando se deseja acessar seus bytes isoladamente. A figura 3.3 ilustra os arranjos dos bytes dentro das words e das doublewords.

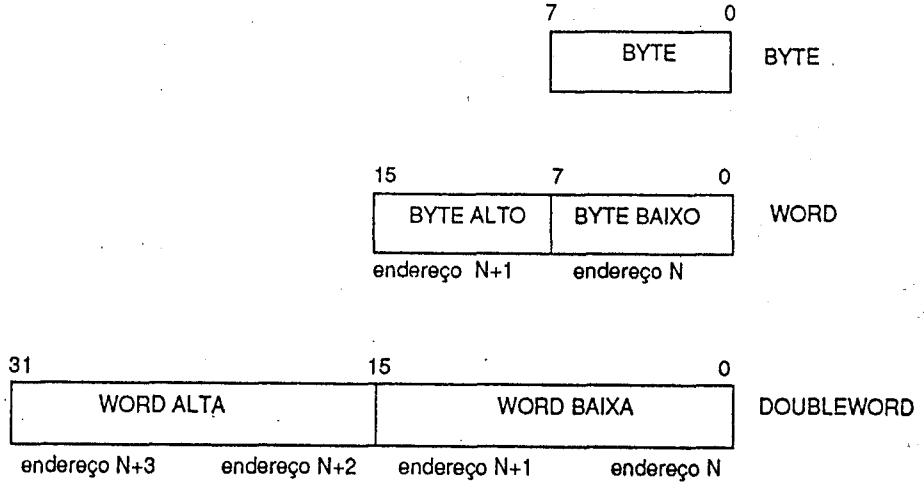


Figura 3.2 - Tipos fundamentais de dados

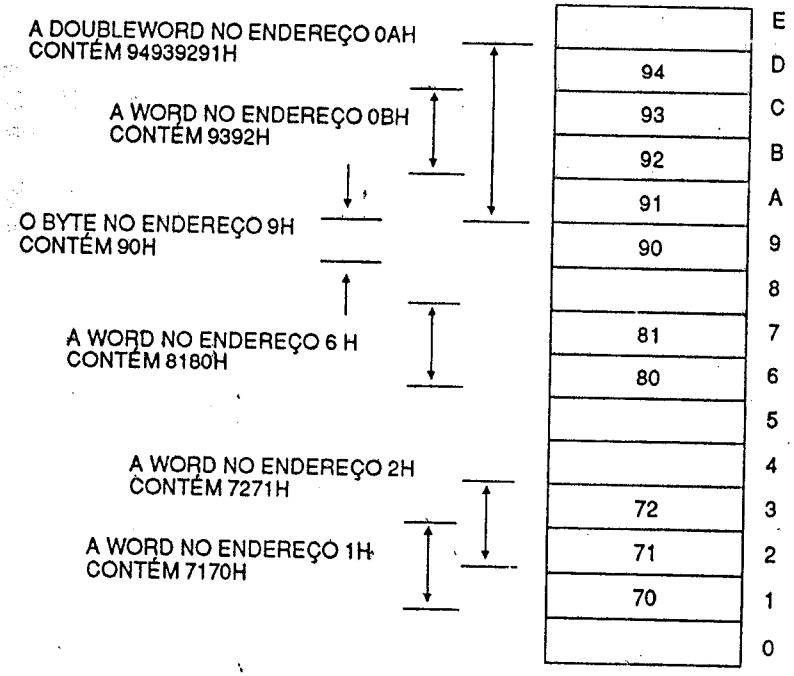


Figura 3.3 - Bytes, words e doublewords em memória.

Observe que as words não precisam estar alinhadas a endereço par e que as doublewords não precisam estar alinhadas a endereço múltiplo de quatro. Isto permite máxima flexibilidade nas estruturas de dados (por exemplo, registros que contenham bytes, words e doublewords misturadas.). Além disso, temos eficiência maior na utilização da memória. Como o 80486 tem um barramento de 32 bits, as comunicações entre o processador e a memória ocorrem como transferências de doublewords alinhadas a endereços sempre múltiplos de 4; o processador converte transferências entre endereços não alinhados a doublewords, em múltiplas transferências alinhadas. As operações desalinhadas reduzem a velocidade do processamento em função dos ciclos extras de barramento. Para se obter a melhor performance, as estruturas de dados, sempre que possível, devem estar alinhadas em endereços pares quando formadas por words e em endereços múltiplos de quatro quando formadas por doublewords.

Embora os bytes, words e doublewords sejam os principais tipos de dados, o processador tem a possibilidade de interpretar estes operandos de maneiras diversas. Algumas instruções especializadas reconhecem os seguintes tipos de dados:

- Inteiro: Número binário sinalizado armazenado em uma doubleword, word ou byte. Todas as operações assumem a representação em complemento de dois. O bit de sinal é o 7 se for um byte, o 15 se for uma word e o 31 se for uma doubleword. Se for número inteiro negativo, o bit de sinal será 1; se for positivo ou zero, o bit de sinal será 0. Em um byte poderemos ter os valores compreendidos entre -128 a +127; na word poderemos representar de -32768 a +32767 e na doubleword teremos de -2^{31} a $(+2^{31}) - 1$.
- Ordinal: Número binário sem sinal armazenado em um byte, uma word ou doubleword. Se estiver contido em um byte, estará na faixa entre 0 a 255; numa word, entre 0 a 65536 e numa doubleword de 0 a $(2^{32})-1$.
- Near pointer: Endereço lógico de 32 bits. Near pointer (ponteiro próximo) é um offset dentro de um segmento. Os near pointers são utilizados para todos os endereçamentos no modelo flat de memória, ou para referências dentro do segmento no modelo segmentado.
- Far pointer: Endereço lógico de 48 bits sendo 16 de seletor de segmento e 32 de offset. Os far pointers são utilizados no modelo segmentado para o acesso a outros segmentos.
- String: Seqüência contínua de bytes, words ou doublewords. Uma string pode conter de 0 a $(2^{32})-1$ bytes (4 gigabytes).
- Campo de bit (bit field): Seqüência contínua de bits. Um campo de bit pode começar em qualquer posição de bit de qualquer byte e pode conter até 32 bits.
- String de bit: Seqüência contínua de bits. Uma string pode começar em qualquer posição de bit de qualquer byte e conter até $(2^{32})-1$ bits.
- BCD: Representação de um dígito decimal codificado em binário (binary coded decimal) na faixa de 0 até 9. Um decimal não compactado é expresso por um byte sem bit de sinal. Cada dígito é armazenado em um byte. A magnitude do número é o valor binário do nibble de ordem mais baixa (4 bits menos significativos do byte); os dígitos somente podem assumir valores entre 0 e 9. O nibble de ordem mais alta (4 bits mais significativos do byte), deve estar zerado para as operações de multiplicação e de divisão, e podem conter qualquer valor nas somas e subtrações.
- BCD compactado (packed BCD): Representação de dígitos decimais codificados em binário, cada um na faixa de 0 a 9. Um dígito é armazenado nos 4 bits menos significativos do byte e outro nos 4 bits mais significativos; portanto cada byte comporta dois dígitos.

- Ponto flutuante: dados do tipo ponto flutuante serão discutidos no capítulo referente ao co-processador matemático.

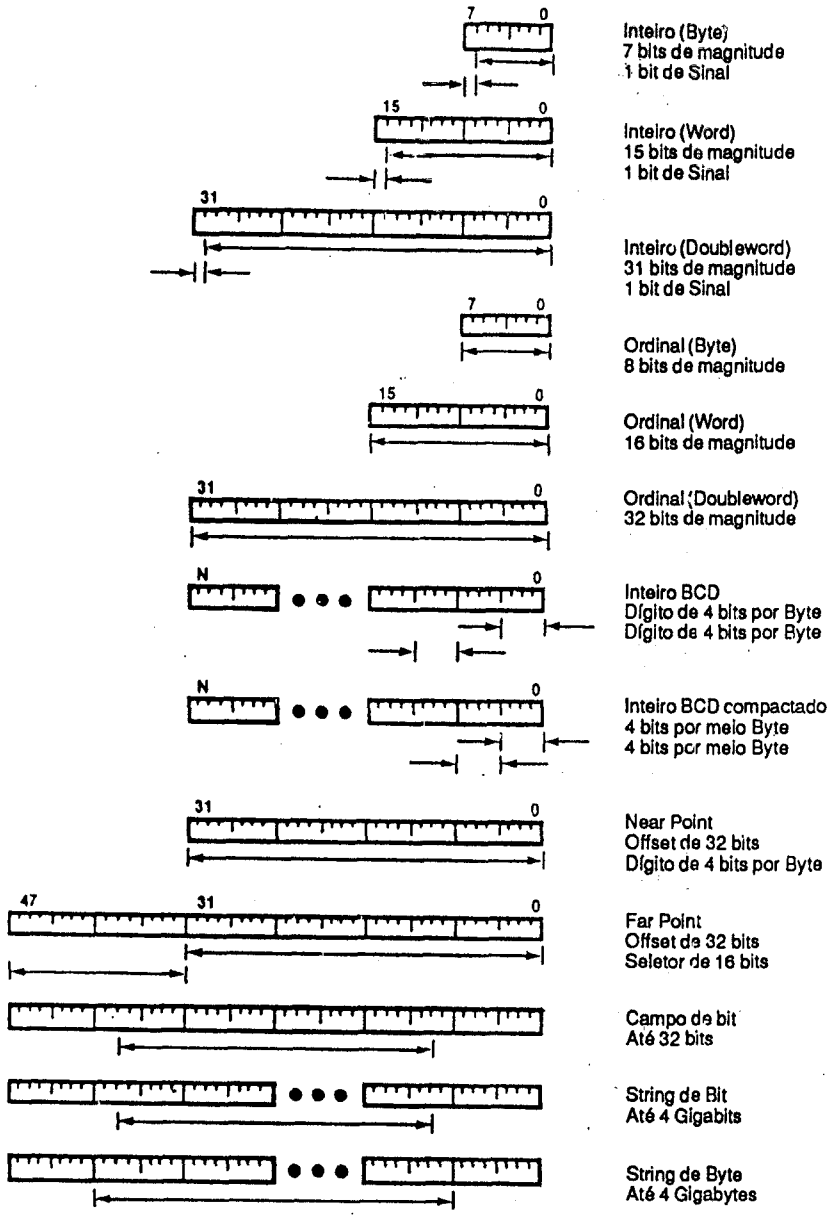


Figura 3.4- Tipos de Dados

ORGANIZAÇÃO DA MEMÓRIA

A memória no barramento do 80486 é chamada de memória física. Ela está organizada como uma seqüência de bytes de 8 bits. Cada byte está associado a um único endereço, chamado endereço físico e cuja faixa vai de 0 ao máximo de $(2^{32})-1$ (4 gigabytes). O gerenciamento de memória é um mecanismo de hardware para se utilizar a memória de maneira segura e eficiente. Quando o gerenciamento de memória é usado, os programas não acessam diretamente a memória física. Os programas endereçam um modelo de memória chamado memória virtual.

O gerenciamento de memória consiste em paginação e segmentação. A segmentação é um mecanismo que permite a existência de múltiplos espaços de endereços independentes. A paginação é um mecanismo que permite o endereçamento de grande espaço de RAM, a partir de pequena quantidade de memória conjugada com o armazenamento em disco. Ambos ou apenas um destes mecanismos pode ser utilizado. Um endereço enviado por um programa é chamado de endereço lógico. A segmentação de hardware traduz o endereço lógico em outro correspondente a um espaço contínuo de endereços não segmentados, conhecido por endereço linear. O paginamento de hardware traduz o endereço linear em endereço físico.

A memória pode parecer um único espaço físico de endereços; como a memória física. Ou então com um ou mais espaços independentes de memória, chamados segmentos. Os segmentos podem ser designados para armazenar especificamente códigos de programas, dados ou stack. De fato, um simples programa pode ter até 16383 segmentos de diferentes tamanhos e tipos. Os segmentos podem ser utilizados para incrementar a confiabilidade dos sistemas. Por exemplo, o stack de um programa pode ser colocado em segmento diferente das instruções a fim de se prevenir o seu crescimento sobre área de códigos.

O uso de segmentos estabelece que os endereços lógicos serão traduzidos em endereços lineares pelo tratamento do endereço como sendo um offset dentro do segmento. Cada segmento possui um descritor, que armazena seu endereço base e o seu limite. Se o offset não excede o limite, e nenhuma outra condição existe que possa impedir a leitura do segmento, o offset e o endereço base são somados para formar o endereço linear.

O endereço linear produzido pela segmentação é utilizado diretamente como endereço físico se o bit 31 do registrador CR0 está em zero. Este bit controla se a paginação está sendo utilizada ou não. Se este bit está em um, o paginamento de hardware é utilizado para converter o endereço linear em endereço físico.

O paginamento de hardware fornece outro nível de organização da memória. Ele quebra o espaço de endereço linear em blocos fixos de 4 KBytes, cha-

mados páginas. O espaço de endereço lógico é mapeado no espaço de endereço linear, que por sua vez é mapeado em alguns números de páginas. Uma página pode estar na memória ou no disco. Quando um endereço lógico é enviado, ele é traduzido em um endereço para uma página em memória, ou então uma exceção é gerada. A exceção dá a chance para o sistema operacional buscar a página em disco e atualizar o conteúdo da memória, mapeando a nova página. O programa que gerou a exceção pode então ser reinicializado sem gerar nova exceção.

Se múltiplos segmentos são utilizados, eles são partes de um ambiente de programação que é visto pelo programador de aplicativos. Se o paginamento é utilizado, ele é normalmente invisível para o programador de aplicação. Ele somente se torna visível quando há interação entre o programa aplicativo e o algoritmo de paginamento utilizado pelo sistema operacional. Quando todas as páginas em memória estão ocupadas, o sistema operacional, a partir de algoritmo, decide qual página deve ser enviada ao disco a fim de liberar espaço para nova página recém acessada.

A arquitetura do 80486 permite aos projetistas certo grau de liberdade para escolher diferentes modelos de memória para cada programa, mesmo que mais de um programa esteja sendo executado ao mesmo tempo. O modelo de organização de memória pode variar entre os seguintes extremos:

- Um espaço de endereço flat onde os códigos, stacks e dados são mapeados todos no mesmo endereço linear. Isto elimina a segmentação e permite qualquer tipo de referência à memória para acessar qualquer tipo de dado.
- Um espaço de endereço segmentado onde códigos, stacks e dados estão em regiões de memória separadas. Um máximo de 16383 espaços de endereço linear com 4 gigabytes cada pode ser utilizado.

Ambos os modelos possuem mecanismos de proteção de memória. Modelos intermediários entre estes dois extremos podem ser escolhidos.

2. ARQUITETURA

REGISTRADORES

O 80486 possui 16 registradores básicos à disposição dos programadores de aplicativos. Estes registradores são agrupados, conforme a sua utilização, em:

1. Registradores de uso geral: são 8 registradores de 32 bits cuja utilização é livre para o programador.
2. Registradores de segmento: em acordo com o tipo de acesso à memória, estes registradores contêm os seletores de segmentos. Por exemplo, existe um registrador separado para o acesso de códigos e outro para informações da área de stack. No total são 6 registradores que determinam, em um dado instante, quais segmentos de memória estão à disposição.
3. Registradores de controle e de status: informam e modificam o estado de operação do 80486.

Registradores de Uso Geral

Os registradores de uso geral são de 32 bits e são chamados de: EAX, EBX, ECX, EDX, EBP, ESP, ESI e EDI. Os nomes destes registradores são derivados dos seus 16 bits mais baixos que, agrupados, correspondem aos equivalentes encontrados previamente no processador 8086: AX, BX, CX, DX, BP, SP, SI e DI.

Os 16 bits dos registradores AX,BX,CX e DX são subdivididas em dois grupos referenciados como: AH,BH,CH,DH (8 bits mais altos) e AL,BL,CL,DL (8 bits mais baixos).

A tabela a seguir, apresenta os nomes possíveis para referências aos registradores do 80486.

8-bits	16-bits	32-bits
AL	AX	EAX
AH		
BL	BX	EBX
BH		
CL	CX	ECX
CH		
DL	DX	EDX
	SI	ESI
	DI	EDI
	BP	EBP
	SP	ESP

Tabela 3.1 - Nome dos registradores de uso geral.

no, parâmetros passados pela rotina que acessou outra e variáveis temporárias alocadas pela sub-rotina. Todos os acessos ao stack são feitos a partir do registrador de segmento SS que, ao contrário do CS, pode ser carregado diretamente. Deste modo, um programa aplicativo pode especificar sua área de stack.

Os registradores DS,ES,FS e GS permitem a utilização de quatro segmentos de dados simultaneamente. Podem-se criar diferentes tipos de estruturas de dados e ter-se meio seguro e eficiente de acesso pela associação de um desses registradores a um segmento específico. Por exemplo, podemos definir um segmento de dados para o módulo que está em execução, outro para dados exportados para módulos de linguagem de alto nível, um terceiro para uma estrutura dinâmica e finalmente um para dados comuns com outros módulos. Se um problema provocar o descontrole de algum programa, a segmentação limitará os danos provocados somente aos segmentos associados ao mesmo. Um operando dentro de segmento de dados é endereçado, especificando-se o seu offset a partir de registrador de uso geral ou na própria instrução.

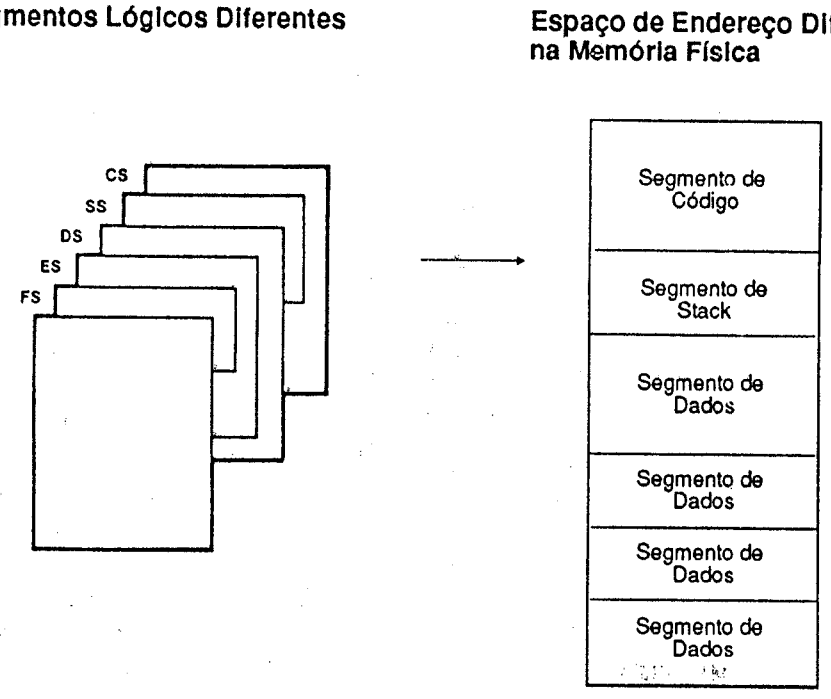


Figura 3.7 - Modelo segmentado de memória

Dependendo da estrutura dos dados; isto é, da forma como eles estão particionados em segmentos, um programa pode requerer mais do que quatro segmentos para dados. Para acessar segmentos adicionais, os registradores DS, ES, FS e GS podem ser carregados durante a execução de um aplicativo pelo próprio programa. Observando-se, é claro, que o registrador de segmento deve ser carregado antes do acesso ao dados.

Um endereço base é mantido para cada segmento. Para endereçar os dados dentro de um segmento, um offset de 32 bits é somado ao endereço base. A partir do momento em que algum segmento está selecionado (pelo armazenamento do seu seletor em registrador de segmento), uma instrução somente precisa especificar o offset da informação desejada. Algumas regras estipulam qual registrador de segmento é usado para formar o endereço quando só o offset é especificado no acesso a um dado em memória.

Implementação do Stack

As operações com stack estão suportadas por três registradores:

1. Registrador de segmento de stack (SS): o stack reside em memória. A quantidade máxima de stacks em um sistema está limitada somente pelo número máximo de segmentos. Um stack pode possuir até 4 gigabytes de comprimento, que é o tamanho máximo que um segmento pode ter para o 80486. Em dado instante, somente um stack pode ser acessado pelo sistema, ou seja, aquele cujo seletor encontra-se no registrador SS. Este será o stack atual, freqüentemente referenciado apenas como "o stack". O registrador SS é utilizado automaticamente pelo processador para todas as operações envolvendo o stack.
2. Registrador de ponteiro de stack (ESP - stack pointer). O registrador ESP armazena o offset do topo do stack (TOS - top-of-stack), no segmento atual do stack. Ele é utilizado pelas instruções de PUSH e POP, chamadas e retornos de sub-rotinas, interrupções e exceções. Ao colocar um item no stack a partir de uma instrução de PUSH, o processador automaticamente decrementa o registrador ESP, e então escreve o item na posição do novo TOS. Para retirá-lo do stack, através de uma instrução POP, o processador copia o conteúdo do TOS e incrementa o registrador ESP. O stack portanto se desenvolve do endereço mais alto para o endereço mais baixo do segmento a ele reservado.

3. Registrador de base para estrutura de stack (EBP). O registrador EBP (Base Pointer) é usado com freqüência para o acesso a estruturas de dados passadas de uma rotina a outra pelo stack. Por exemplo, quando uma rotina é chamada, o stack contém o endereço de retorno e pode vir a conter um número de bytes previamente estruturados pelo uso de instruções PUSH. Para acesso mais cômodo a estes valores, a rotina chamada pode posicionar o EBP no mesmo endereço do registrador ESP e utilizá- lo como base para a localização desses dados.

Quando o EBP é utilizado no acesso à memória, o registrador de segmento associado é o de stack (SS); não sendo necessário especificá-lo na instrução. Outros registradores de segmento podem ser utilizados junto com o EBP mas, neste caso, deverão ser especificados na operação e, como conseqüência, o código gerado é maior do que quando utilizado o SS.

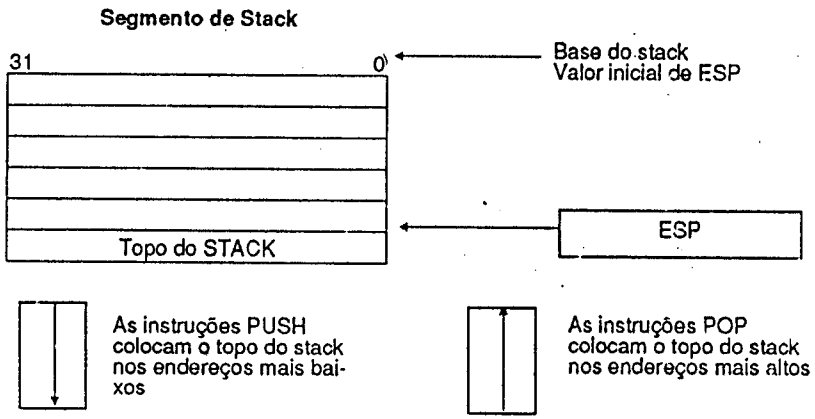


Figura 3.8 - Stacks

Registrador de Flags

Códigos de condição (por exemplo, carry, sign e overflow) e bits de modo de operação do 80486, são armazenados em um registrador de 32 Bits chamado EFLAGS. A figura a seguir, define a posição dos bits no registrador. Cada flag ou controla a operação do processador ou reporta o seu status.

Os flags podem ser agrupados em: indicadores de status, de controle e de sistema. Estes últimos serão discutidos posteriormente.

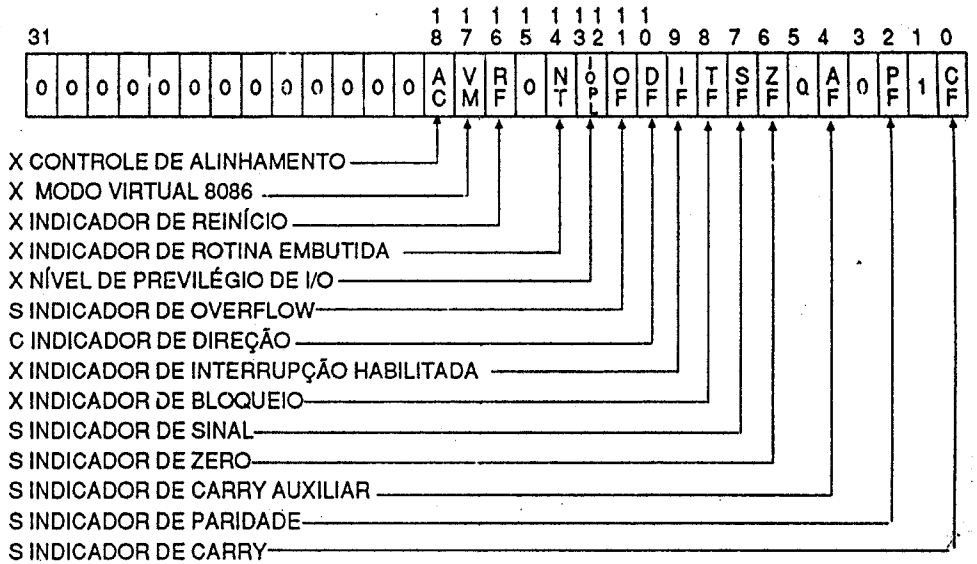


Figura 3.9 - Registrador EFLAGS.

FLAGS DE STATUS

Os flags de status (indicadores de status) reportam que tipo de resultado foi produzido por uma operação aritmética. Desvios condicionais e chamadas a sub-rotinas verificam o estado destes indicadores e agem de acordo com o mesmo. Por exemplo, quando um contador de controle de um loop é decrementado por zero, o estado do flag ZF (Zero Flag) muda, e esta alteração pode ser utilizada por um desvio condicional para o reinício da contagem.

Os flags de status são mostrados na tabela 3.2.

Nome	Finalidade	Condição Reportada
OF	Estouro (overflow)	O resultado excede o limite positivo ou negativo da faixa de números
SF	Sinal	O resultado é negativo (menor do que zero)
ZF	Zero	O resultado é igual a zero
AF	Carry Auxiliar	Carry proveniente da posição do bit 3 (utilizado para BCD)
PF	Paridade	O byte baixo do resultado tem paridade par (quantidade par de bits iguais a 1)
CF	Indicador de Carry	Carry proveniente do bit mais significativo do resultado

Tabela 3.2 - Flags de status.

FLAG DE CONTROLE

O flag DF controla a execução das operações com strings.

Quando em 1, o DF (Direction flag), determina que as instruções com strings devem decrementar seus ponteiros uma vez executadas; isto é, devem processar as strings do endereço mais alto para o mais baixo.

Quando em 0, o DF determina que as instruções devem incrementar seus ponteiros após suas execução, ou seja, as strings serão processadas do endereço mais baixo para o mais alto.

PONTEIRO DE INSTRUÇÕES

O ponteiro de instruções (EIP - instruction pointer), contém o offset do endereço da próxima instrução a ser executada pelo processador dentro do segmento de códigos (CS). O instruction pointer não está diretamente à disposição do programador. Ele é alterado como conseqüência da execução das instruções de transferência de controle (desvios, retornos, etc.), interrupções e exceções.

O EIP avança sempre de instrução em instrução, isto é, uma por vez. Devido ao prefetching de instruções, (busca antecipada de instruções), somente uma pequena parcela da atividade do barramento é dedicada ao armazenamento do conteúdo do EIP.

O 80486 não realiza a busca (fetch) de uma instrução a cada acesso ao bus. O processador busca grupos de códigos de 128 bits alinhados, isto é, blocos de 128 bits cujo endereço tem os quatro bits menos significativos em 0. Estes blocos são lidos independente da instrução ocupá-los totalmente, em parte ou precisar de mais bits para a sua operação. Mas, com certeza, ao iniciar sua execução, ela encontra-se pronta e decodificada dentro do processador. Esta característica maximiza a performance do processamento, pois permite que uma instrução seja executada ao mesmo tempo que outra está sendo obtida e decodificada.

Quando uma instrução de desvio é realizada, o processador busca todo o bloco alinhado que contém o endereço de destino do desvio. As instruções que haviam sido previamente lidas ou decodificadas são descartadas. Se uma busca for provocar exceção, por exemplo a busca por instruções após o limite do segmento, a exceção não é reportada até que a instrução que a geraria seja efetivamente executada. Se for descartada (em função de uma instrução de desvio), a exceção não é gerada.

Em modo real, o prefetching pode realizar o acesso a posições de memória que não foram previstas pelo programador. Em modo protegido, a execução dessas posições causará uma exceção que reportará este procedimento indevido. Todavia, não existem mecanismos de hardware para controlar o prefetching em modo real. Por exemplo, se um sistema não retorna o sinal de RDY# (ready: sinal que determina o fim de um ciclo de bus), quando é feito um acesso a endereços não implementados, o prefetching deve ser impedido para que não avance nestes endereços. Do mesmo modo para um sistema de geração e verificação de paridade em memória, o prefetching deve precaver-se ao endereçar posições que estão fora da faixa coberta pelo circuito de paridade. Podemos, como alternativa, gerar o sinal de RDY# para todos os acessos fora dos endereços onde a paridade é gerenciada, ou então ignorar os erros de paridade decorrentes dessa situação.

Para prevenir o acesso do prefetching a endereços indesejados, deve-se colocar o último byte executável longe o bastante destes endereços. Por exemplo: para que o prefetching não afete os endereços do block entre 10000H a 1000FH, o último byte executável deve estar no endereço 0FFEEH ou anteriores. Isto coloca ao menos um byte livre seguido de um bloco de 128 bits alinhados, também livre, entre a última instrução e o último endereço que não pode ser referenciado. É importante observar que a Intel não se compromete a manter este comportamento do prefetching nas futuras gerações de seus processadores, referindo-se ao mesmo como particularidade do 80486.

3.MODO REAL DE OPERAÇÃO

OPERADORES EM MEMÓRIA

As instruções que se utilizam de operadores localizados na memória, devem especificar o segmento e o offset do endereço do operador. Os segmentos são definidos pelo prefixo de segment- override, que nada mais é do que um byte colocado no início da instrução com este objetivo. Caso o segmento não seja especificado, algumas regras serão utilizadas para se determinar o segmento default (assumido) para a instrução. O offset é definido em uma das seguintes formas:

1. Muitas instruções de acesso à memória, contêm um byte para especificar o método de endereçamento do operando. Este byte é chamado de modR/M e está posicionado após o opcode, (código de operação), indicando se o operador encontra-se em registrador ou na memória. Se o operador encontra-se na memória, seu endereço é calculado a partir de um registrador de segmento e um dos seguintes valores: registrador de base, de índice, fator de escala ou deslocamento. Quando um registrador de índice é utilizado, um segundo byte segue ao modR/M, especificando o mesmo e o fator de escala. Esta forma de endereçamento é a mais flexível.
2. Algumas poucas instruções, usam modos de endereçamento implícito:

Uma instrução de MOV com o registrador AL ou EAX como origem ou destino, pode endereçar a memória com uma doubleword codificada na própria instrução. Esta forma especial da instrução MOV descarta o uso de registradores de base, de índice ou fatores de escala. Esta forma apresenta a vantagem de ser um byte menor do que a de uso geral.

Operações com strings endereçam a memória a partir do registrador de segmento DS, usando o registrador ESI (são as instruções MOVS, CMPS, OUTS, LODS e SCAS) ou usam o registrador ES juntamente com o EDI (instruções MOVS, CMPS, INS e STOS).

As operações com stack endereçam a memória a partir dos registradores SS (segmento) e ESP (offset) e são as instruções PUSH, POP, PUSHA, POPA, PUSHAD, POPAD, PUSHF, POPF, PUSHFD, POPFD, CALL, RET, IRET, e IRETD, além das interrupções e das exceções.

Seleção de Segmento

A especificação explícita de segmento é opcional. Quando um segmento não é definido por um prefixo de segment-override, o processador automaticamente seleciona um dos registradores de segmento a partir das regras mostradas na tabela 3.3.

Cada tipo de acesso à memória tem um registrador de segmento default associado. Operandos de dados em geral se utilizam do registrador de segmento DS que é considerado o principal para este tipo de acesso. Já os operandos em stack são referenciados pelos registradores SS e ESP.

Os prefixos de segment-override permitem a alteração do segmento default da instrução para aquele da conveniência do programador: Todavia, algumas instruções não alteram o registrador de segmento a ser utilizado, mesmo que um prefixo de segment-override as anteceda:

- O operando destino das instruções de strings é sempre definido pelo registrador ES.
- A origem de um POP ou destino de um PUSH é sempre referenciado pelo registrador SS.
- A busca de instruções é feita a partir do CS exclusivamente.

Tipo da referência	Segmento usado e Registrador usado	Regra de seleção Default
Instruções	Segmento de códigos - Registrador CS	Automático com a busca por instruções
Stack	Segmento de stack - Registrador SS	Todas as operações PUSH e POP. Qualquer referência à memória que utilize ESP ou EBP como registrador de base
Dados locais	Segmento de dados - Registrador DS	Todas as referências a dados
Strings de Destino	Segmento extra de dados - Registrador ES	Destino das instruções com strings

Tabela 3.3 - Regras para seleção do segmento default

Computação do Endereço Efetivo

O byte modR/M fornece a mais flexível forma de endereçamento. A maioria das instruções têm o byte modR/M após o seu opcode. Para operandos em memória especificados pelo byte modR/M, o offset a ser utilizado com o registrador de segmento será o resultado da soma de três componentes:

- Um deslocamento.
- Um registrador de base.
- Um registrador de índice. Que pode ainda vir a ser multiplicado por um fator de 2, 4 ou 8. (Fator de escala).

O offset que resulta da soma destes componentes é chamado de endereço efetivo. Cada um destes elementos pode ser um valor positivo ou negativo. A figura 3.4 ilustra todas as possibilidades para o endereçamento via modR/M.

O deslocamento, por ser codificado na instrução, é útil para o endereçamento relativo de quantidades fixas como:

- Localização de operandos de escala simples.
- Início de arranjo alocado estaticamente.
- Offset para um campo dentro de um registro.

A base e o índice têm funções similares. Ambos se utilizam dos mesmos registradores e podem ser utilizados para o endereçamento de informações cuja posição se altera durante a execução do programa. Exemplo:

- Localização de parâmetros da rotina e variáveis locais no stack.
- O início de um registro entre muitos dentro de um arranjo.
- O começo de uma dimensão dentro de um arranjo multidimensionado.
- O início de um array alocado dinamicamente.

A utilização de registradores de uso geral como base ou índice apresenta as seguintes considerações:

- O registrador ESP não pode ser utilizado como índice.
- Quando o registrador ESP ou EBP é utilizado como base, o segmento, por default, estará no SS. Nos demais casos, o registrador de segmento será o DS.

O fator de escala permite indexação eficiente em um arranjo quando seus elementos são de 2, 4 ou 8 bytes. A escala de um registrador de índice é feita no hardware no momento em que o endereço é determinado. Como resultado, economiza-se uma instrução de deslocamento (shift) ou uma multiplicação.

O deslocamento, a base e o índice podem ser utilizados em qualquer combinação e qualquer um destes elementos pode inexistir. O fator de escala somente é utilizado a partir da existência de um índice. Cada combinação desses elementos pode ser extremamente útil para estruturas de dados mais comuns em linguagem de alto nível. A seguir, temos alguns exemplos:

DESLOCAMENTO

O deslocamento sozinho indica o offset de um operando. Esta forma de endereçamento é utilizada no acesso de operadores alocados estaticamente na memória. Um deslocamento pode ser um byte, uma word ou uma doubleword.

BASE

A base é o endereçamento de operando realizado indiretamente através de registrador de uso geral.

BASE+DESLOCAMENTO

Um registrador somado a um deslocamento pode ser utilizado com dois objetivos:

1. Como índice em um array (arranjo estático) quando o tamanho dos elementos não é de 2, 4 ou 8 bytes. O deslocamento codifica o offset do início do arranjo. O registrador armazena o resultado dos cálculos realizados para a determinação de um elemento específico dentro do arranjo.
2. Para o acesso a campo em um registro. Neste caso, o registrador de base informa o início do registro e o deslocamento do offset dentro do mesmo.

Um caso importante para esta combinação é o acesso a parâmetros em registro de ativação de um processo. Registro de ativação é uma estrutura de stack (stack frame) criada quando uma sub-rotina é chamada. Neste caso, o registrador EBP é a melhor escolha para base pois, automaticamente, seleciona o segmento de stack (via SS), o que resulta em compactação dos códigos do programa.

(ESCALA X ÍNDICE)+DESLOCAMENTO

Esta combinação é eficiente para o acesso a arranjos cujos elementos são de tamanho igual a 2, 4 ou 8 bytes. O deslocamento informa o início do array, o registrador de índice, multiplicado pelo fator de escala, qual o elemento procurado.

BASE+ÍNDICE+DESLOCAMENTO

Dois registradores são utilizados em conjunto para endereçar ou arranjo bidimensionado (o deslocamento informa o início do array) ou os múltiplos re-

gistros dentro de um array (neste caso, o deslocamento é o offset para um campo dentro de um registro).

BASE+(ÍNDICE X ESCALA)+DESLOCAMENTO

Esta combinação é um modo de indexação eficiente para arranjo bidimensional quando os elementos do mesmo são de 2, 4 ou 8 bytes.

INSTRUÇÕES DE MOVIMENTAÇÃO DE DADOS

Estas instruções providenciam um método conveniente para a movimentação de bytes, words ou doublewords entre memória e os registradores internos ao processador. Elas são de três tipos:

1. Instruções de movimentação de uso geral.
2. Instruções para manipulação de stack.
3. Instruções de conversão de tipo.

Instruções de Movimentação Para Uso Geral

MOV (Move) transfere byte, word ou doubleword do operando fonte para o operando destino. A instrução MOV é útil para a transferência de dados através de um desses meios:

- De memória para registrador
- De registrador para memória
- Entre registradores de uso geral
- Valores imediatos para registrador
- Valores imediatos para memória

A instrução MOV não pode transferir diretamente informação de memória a memória, também não é permitida a transferência direta de registrador de segmento para registrador de segmento. Entretanto, existe uma instrução de movimentação de strings, cujo mneumônico é MOVS, que possibilita a transferência de dados de memória a memória. Existe também uma forma especial da instrução MOV para ser utilizada a partir do registrador EAX ou AL, que movimenta o seu conteúdo para a posição de memória especificada por um offset de 32 bits codificado na instrução. Esta forma citada não admite o uso de registrador de segmento alternativo, registrador de índice ou fator de escala. O código gerado é um byte menor do que o da instrução MOV de uso geral. Uma codificação similar

(reduzida) é providenciada para a movimentação de dados de 8, 16 ou 32 bits imediatos para qualquer registrador de uso geral.

XCHG (Exchange) permuta o conteúdo de dois operandos. Esta instrução substitui o uso de três MOV e não requer alocação temporária para preservar o conteúdo de um operando enquanto o outro é carregado. A instrução XCHG é especialmente útil na implementação de semáforos ou estruturas similares para sincronização de processos.

A instrução XCHG permuta dois operandos que podem ser bytes, words ou doublewords. Os operadores podem ser ambos registradores ou um registrador e uma posição de memória. Quando utilizada com memória, XCHG automaticamente ativa o sinal de LOCK.

Instruções de Manipulação de Stack

PUSH (Push) decrementa o stack pointer (registrador ESP), e logo em seguida copia o conteúdo do operando para o topo do stack (top of stack) (ver figura 3.10). A instrução PUSH é utilizada com freqüência para a colocação de parâmetros no stack antes da chamada a uma sub-rotina. Dentro de uma rotina, pode ser utilizada para reservar espaço no stack para variáveis temporárias. A instrução PUSH opera com memória, operandos imediatos, e registradores (inclusive os de segmento). Uma forma especial da instrução PUSH está disponível para o armazenamento dos registradores de 32 bits no stack. Esta forma é um byte menor do que a de uso geral.

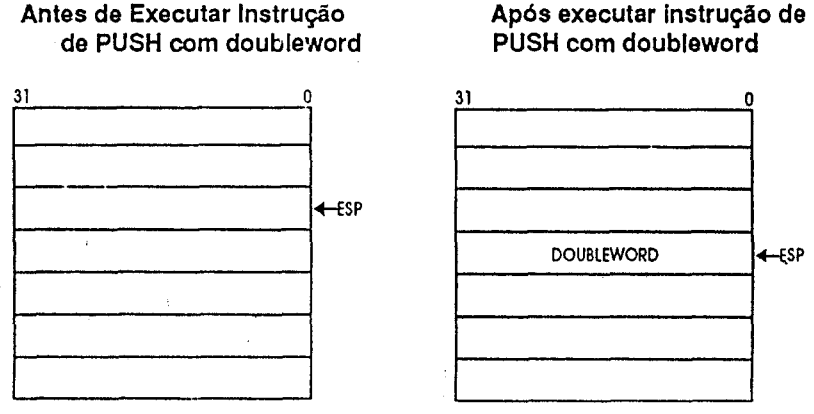


Figura 3.10 - Instrução PUSH

PUSHA (Push All Registers) armazena o conteúdo de oito registradores de uso geral no stack (ver FIGURA 3.11). Esta instrução simplifica as chamadas a rotinas, reduzindo o número de instruções necessárias para preservar o conteúdo dos registradores de uso geral. A ordem na qual são inseridos os registradores no stack é a seguinte: EAX, ECX, EDX, EBX, o valor inicial do ESP antes do EAX ser colocado no stack, EBP, ESI e EDI. O efeito da instrução PUSHA é revertido pela instrução POPA.

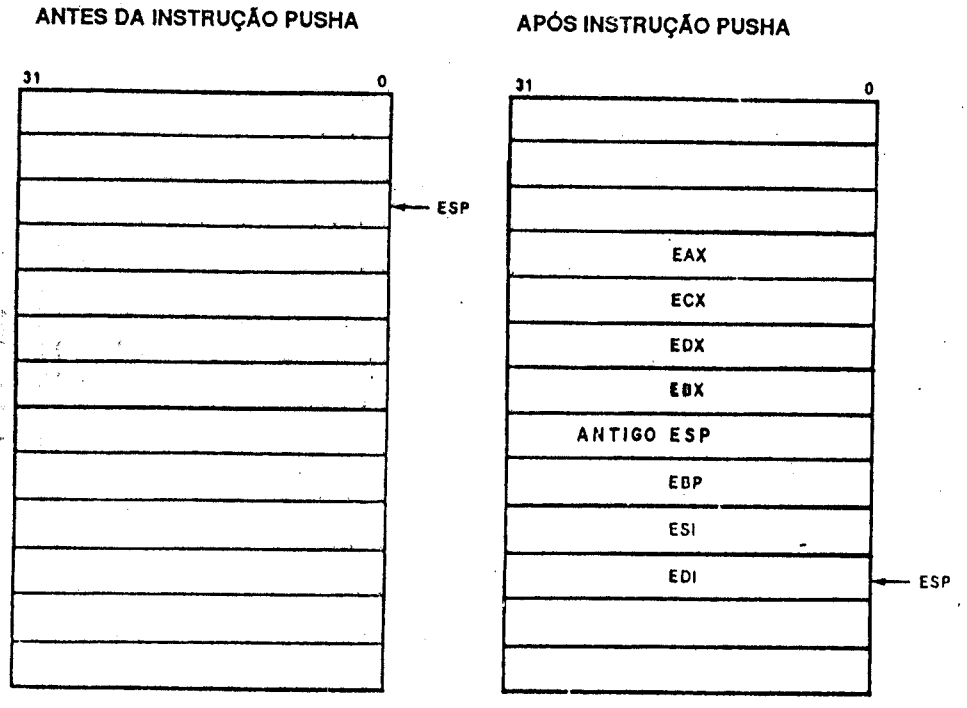


Figura 3.11 - Instrução PUSHA

POP (Pop) transfere a word ou doubleword do atual topo do stack (indicado pelo registrador ESP), para o operando destino, em seguida, incrementa o registrador ESP para apontar para o novo topo de stack. Ver FIGURA 3.12. A instrução POP move informação do stack para registradores de uso geral, de segmento, ou memória. Uma forma especial da instrução POP está disponível para o armazenamento nos registradores de 32 bits a partir do stack. Esta forma é um byte menor do que a de uso geral.

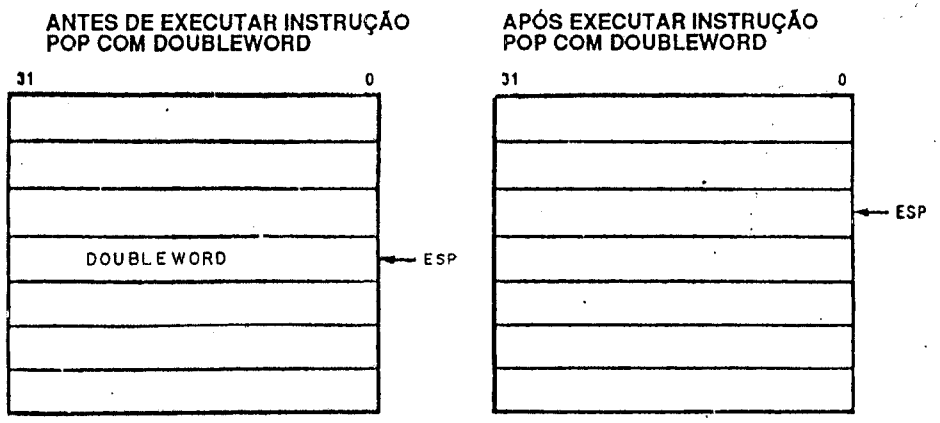


Figura 3.12 - Instrução POP

POPA (Pop All Registers) recupera todos os dados armazenados previamente no stack por uma instrução PUSHA. Excetuando-se o ESP cujo valor é derivado da execução da instrução.

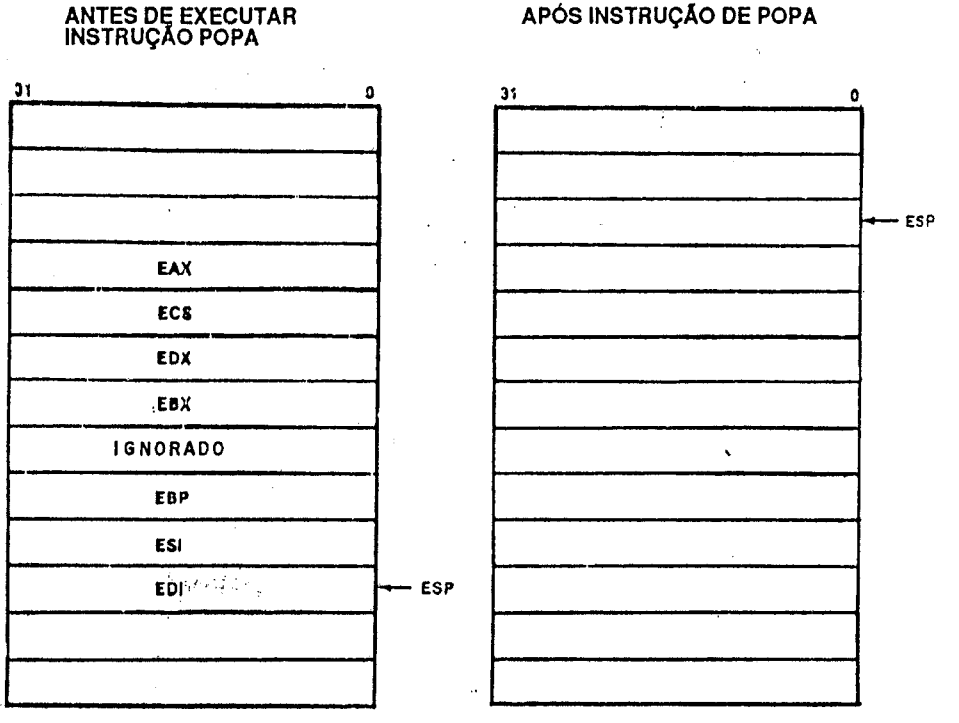


Figura 3.13 - Instrução POPA

STOS (Store String) armazena no elemento da string endereçado pelo registrador EDI (destino), o conteúdo do registrador AL, AX ou EAX, caso o elemento seja um byte, uma word ou uma doubleword respectivamente.

INSTRUÇÕES PARA LINGUAGENS BLOCOESTRUTURADAS.

Estas instruções fornecem suporte em linguagem de máquina para a implementação de linguagens blocoestruturadas, como "C" e "Pascal". São as instruções ENTER e LEAVE que simplificam a entrada e a saída de rotinas geradas a partir de um processo de compilação. Elas suportam uma estrutura de ponteiros e variáveis locais armazenada no stack e conhecida como stack frame.

ENTER (Enter Procedure) cria um stack frame compatível com as regras das linguagens blocoestruturadas. Nestas linguagens, uma rotina tem acesso às suas próprias variáveis e a outras definidas em alguma parte do programa. O âmbito de uma rotina é o grupo de variáveis a que ela tem acesso. As regras que definem este âmbito variam de linguagem a linguagem; e podem se basear no encadeamento de rotinas, na divisão do programa em arquivos compilados separadamente ou em algum outro esquema de modularização.

A instrução ENTER possui dois operandos. O primeiro especifica o número de bytes a ser reservado no stack para o armazenamento dinâmico na rotina em que se está ingressando. Armazenamento dinâmico é a memória alocada para variáveis criadas quando a rotina é chamada, também conhecidas como variáveis automáticas. O segundo parâmetro informa o nível da ordem de inserção da rotina, que pode ser de 0 a 31. A ordem de inserção é a profundidade da rotina na hierarquia de um programa bloco estruturado. Este nível da ordem não possui nenhuma relação com o nível de privilégio de proteção em modo virtual, ou com o privilégio de acesso aos dispositivos de I/O.

O nível da ordem de inserção determina o número de ponteiros de *stack frame a serem copiados do frame precedente para o novo stack frame. Um ponteiro de stack frame é uma doubleword utilizada para acessar as variáveis de* uma rotina. O grupo de ponteiros de stack frame utilizado por uma rotina para acessar as variáveis de outras é conhecido como display. A primeira doubleword em um display é o ponteiro do stack frame precedente. Este ponteiro é utilizado pela instrução LEAVE para desfazer o efeito de uma instrução ENTER. Isto é realizado descartando-se o stack frame em curso.

Após a instrução ENTER criar o display para a rotina, ela aloca as variáveis dinâmicas locais para o processo, decrementando o registrador ESP do número de bytes especificados no primeiro parâmetro. Este novo valor para o registrador ESP serve como o topo de stack inicial para todas as instruções de PUSH e POP existentes na rotina.

Para permitir à rotina o endereçamento do seu display, a instrução ENTER aponta o registrador EBP para a primeira doubleword no display. Como o stack evolui para baixo, esta doubleword é a de mais alto endereço no display. As instruções de manipulação de dados via registrador EBP, automaticamente adotam o registrador SS como seletor de segmento ao invés do registrador DS.

A instrução ENTER pode ser utilizada de duas formas: embutida ou não embutida; podemos nos expressar também como inserida ou não inserida. Se o nível da ordem é zero, a forma não embutida é a utilizada. A forma não inserida armazena o conteúdo do registrador EBP no stack e copia o conteúdo do registrador ESP no EBP. Em seguida, o primeiro operando é subtraído do ESP para o armazenamento dinâmico. A diferença em relação à forma embutida é que nenhum ponteiro de stack frame é copiado. A forma embutida ocorre quando o segundo parâmetro é diferente de zero.

A rotina principal (main procedure), na qual todas as demais estão inseridas opera no mais alto nível de ordem, o nível 1. A primeira rotina que ela chama, opera um nível mais profundo; nível dois. Uma rotina em nível dois pode acessar as variáveis do programa principal que estão em posições fixas especificadas através do compilador. No caso do nível 1, a instrução ENTER aloca no stack somente o armazenamento dinâmico solicitado pois não há nenhum display prévio a copiar.

Uma rotina que chama a outra em um nível mais baixo, concede a esta o acesso a variáveis da rotina chamadora. A instrução ENTER possibilita este acesso posicionando um ponteiro para o stack frame da rotina chamadora no display.

Uma rotina que chama a outra no mesmo nível de ordem não concede o acesso a suas variáveis. Neste caso, a instrução ENTER copia somente a parte do display da rotina chamadora que referencia as rotinas embutidas previamente e que operam em um nível de ordem mais elevado. O novo stack frame não inclui o ponteiro para o stack frame da rotina chamadora.

A instrução ENTER considera uma rotina reentrante como uma chamada a programa de mesmo nível de ordem. Neste caso, cada iteração da rotina reentrante pode endereçar somente suas próprias variáveis e as da rotina na qual encontra-se embutida.

Uma rotina reentrante sempre pode endereçar suas próprias variáveis, e não requer ponteiros para o stack frame das iterações prévias.

Copiando somente os ponteiros de stack das rotinas de níveis de ordem mais elevadas, a instrução ENTER assegura que uma certa rotina acessará

somente aquelas variáveis de mais alto nível e não aquelas das rotinas com o mesmo nível de ordem.

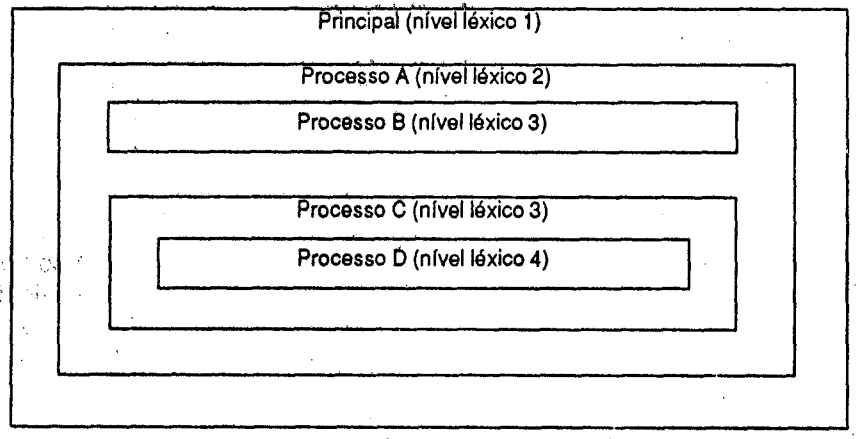


Figura 3.24 - Rotinas embutidas

Linguagens blocoestruturadas podem usar os níveis de ordem definidos pela instrução ENTER para controlar o acesso a variáveis de uma rotina inserida em outra.

LEAVE (Leave Procedure) reverte a ação de uma instrução ENTER executada anteriormente. A instrução LEAVE não possui qualquer operando. A instrução LEAVE copia o conteúdo do registrador EBP no ESP e libera todo o espaço do stack alocado para a rotina. Em seguida, a instrução LEAVE restaura o valor do registrador EBP preservado anteriormente no stack. Simultaneamente, esta operação recupera o valor original do ESP. Uma instrução RET subseqüente pode remover algum argumento e o endereço de retorno introduzidos no stack pelo programa chamador para uso da rotina.

INSTRUÇÕES DE CONTROLE DE FLAG

As instruções de controle de flag alteram o estado dos bits do registrador EFLAG, conforme apresentado na tabela a seguir.

Instrução	Efeito
STC (Estabeleça CF=1)	CF=1
CLC (Zerar CF)	CF=0
CMC (Complementar CF)	CF=-(CF)
CLD (Zerar indicador de direção)	DF=0

Tabela 3.8 - Instruções de controle de flag.

Instruções de Controle do Flag de Direção e de Carry

As instruções de controle de carry são úteis em conjunto com as instruções de rotações com carry como RCL e RCR. Podemos, a partir delas, inicializar o conteúdo do flag de carry que será carregado em um operando.

As instruções de controle do flag de direção (DF), permitem definirmos a evolução de uma operação de string. Quando DF estiver em zero, a instrução incrementa, para cada iteração, os registradores de índice para string: ESI (fonte) e EDI (destino). Quando estiver em um , os registradores são decrementados.

Instruções de Transferência de Flags

Somente os flags CF e DF possuem instruções específicas que permitem a alteração de seus estados diretamente. Para acessar os demais indicadores as instruções de transferência de flags devem ser utilizadas para movimentar o conteúdo do EFLAGS para o registrador AH ou para o stack. Onde então, eles poderão ser alterados e em seguida retornados para o EFLAGS.

LAHF (Load AH from Flags) copia os flags SF, ZF, AF, PF e CF para os Bits 7, 6, 4, 2 e 0, respectivamente, do registrador AH (ver FIGURA 3.25). Os bits 5, 3 e 1 são indefinidos. O conteúdo do registrador EFLAGS permanece inalterado.

SAHF (Store AH into Flags) copia para os flags SF, ZF, AF, PF e CF os Bits 7, 6, 4, 2 e 0, respectivamente, do registrador AH.

Instrução de Não Operação

NOP (No-operation) ocupa um byte do espaço de códigos. Quando executada, ela incrementa o registrador EIP para a próxima instrução e não afeta mais nada.

Instrução de Tradução

XLATB (Translate) armazena no registrador AL, o conteúdo de um byte proveniente de uma tabela de conversão. O conteúdo do registrador AL a princípio, é interpretado como um índice nesta tabela. O registrador EBX é considerado como o endereço base da mesma. A instrução XLAT carrega o conteúdo do byte indexado no registrador AL.

Instrução de Byte Swap

BSWAP (Byte swap) inverte a ordem dos bits em um registrador de 32 bits. Os bits 7 a 0 são trocados com os bits 30 a 24 e os bits 15 a 8 são trocados com os bits de 23 a 16. Esta instrução é útil para converter dados no formato "big-endian" para o "little- endian".

Instrução de Troca e Adição

XADD (Exchange and Add) necessita de dois operandos: o operando fonte no registrador e o destino em memória ou registrador. O operando fonte é armazenado com o valor do operando destino. O operando destino é armazenado com o valor da soma entre o operando fonte e o operando destino. Os flags refletem o resultado da soma.

Instrução de Comparação e Troca

CMPXCHG (Compare and Exchange) necessita de três operandos: um operando fonte em registrador, um operando destino em registrador ou memória e o acumulador (AL, AX ou EAX dependendo do tamanho do operando). Se o operando destino for igual ao acumulador então o valor do operando fonte é armazenado no operando destino. Caso contrário, o valor do operando destino é armazenado no acumulador. Os flags reportam o resultado da subtração do acumulador do operando destino. O flag ZF será 1 sempre que o operando destino e o acumulador forem iguais, caso contrário será zero.

INTERRUPÇÕES E EXCEÇÕES

O processador 80486 possui dois mecanismos para interromper a execução de um programa:

1. Exceções são eventos síncronos provocados pelo próprio processador em resposta a certas condições detectadas quando da execução de uma instrução.
2. Interrupções são eventos assíncronos normalmente desencadeados por dispositivos externos que estão solicitando a atenção do processador.

As interrupções e as exceções são semelhantes por suspenderem temporariamente a execução de um programa a favor de uma rotina de maior prioridade. A principal distinção entre ambas está em suas origens. Uma exceção sempre pode ser reproduzida reexecutando-se o programa que a causou, enquanto que uma interrupção pode ter um relacionamento extremamente complexo e dependente de temporizações com o programa. O que muitas vezes dificulta a sua reprodução.

Em condições normais, um programa aplicativo não deve se preocupar com o tratamento de interrupções e exceções. O sistema operacional, o programa monitor ou o device driver se encarregarão das mesmas. Todavia, alguns tipos de exceções são relevantes para um programa aplicativo e muitos sistemas operacionais deixam a cargo do mesmo a decisão final sobre determinadas situações de programação. O sistema operacional define, entretanto, a interface entre o programa aplicativo e o mecanismo de exceção do processador 80486. A tabela a seguir, lista as interrupções e as exceções.

Número do vetor	Descrição
0	Erro de divisão
1	Chamada ao depurador
2	Interrupção sem máscara (NMI)
3	Breakpoint
4	INTO - Detectado estouro (Overflow)
5	Excedidos limites da faixa - (BOUND)
6	Código de operação inválido
7	Periférico não disponível
8	Dupla falha
9	(Reservada a Intel. Não utilizar. O processador 80486 não faz uso.)
10	Segmento de estado de tarefa (TSS) inválido
11	Segmento ausente
12	Exceção de stack
13	Proteção geral
14	Falha de página
15	(Reservada a Intel. Não utilizar.)

Tabela 3.9 - Interrupções e Exceções

A exceção de erro de divisão (divide error) é conseqüência da tentativa de execução da instrução DIV ou IDIV com o denominador zero, ou quando o quociente é muito grande para ser armazenado no operando destino.

Uma exceção de debug (depuração) pode ser proveniente de um programa aplicativo em função do indicador TF (Trap flag).

Uma exceção de breakpoint ocorre como resultado da instrução INT3. Em alguns programas depuradores, esta instrução é utilizada para interceptar a execução de um programa em análise.

Uma exceção de overflow é decorrência da execução da instrução INTO com o indicador OF (Overflow flag) igual a 1.

A exceção de verificação de limites (Bound-check) decorre da instrução BOUND executada com índice fora dos limites de um arranjo em memória.

Uma exceção de dispositivo não disponível ocorre sempre que o processador encontrar uma instrução de escape e o indicador TS (Task switched) ou EM (emulate coprocessor) no registrador de controle CR0 é igual a 1.

Uma exceção de verificação de alinhamento ocorre em operações com memória não alinhada no modo usuário (nível de privilégio 3), desde que os bits AC (aligment check) e AM (Aligment mask) no CR0 estejam ambos iguais a 1.

A instrução INT gera uma interrupção sempre que for executada; o processador trata esta interrupção como uma exceção. Seus efeitos, assim como os de todas as demais exceções, são determinados pela rotina responsável pela sua manipulação inserida no programa aplicativo ou no sistema operacional.

As exceções causadas pela segmentação ou pelo paginamento são tratadas de modo diferente das interrupções. Normalmente, o conteúdo do registrador EIP é preservado no stack quando uma interrupção ou exceção é gerada. Entretanto, as exceções decorrentes da segmentação ou do paginamento preservam o conteúdo de alguns registradores do processador anterior ao início da interpretação da instrução. O conteúdo preservado do contador de programa endereça a instrução que causou a exceção, ao invés de apontar a instrução seguinte. Isto permite ao sistema operacional corrigir a condição de exceção e reiniciar o programa que a gerou. Este mecanismo é totalmente transparente para o programa.

INSTRUÇÕES COM REGISTRADOR DE SEGMENTO

Existem muitos tipos distintos de instruções que manipulam registradores de segmento. Elas encontram-se agrupadas uma vez que se o projetista de sistema escolher um modelo não segmentado, nenhuma delas será utilizada.

Instruções de Transferência Para Registradores de Segmento

As instruções MOV, POP e PUSH podem ser utilizadas para armazenar informações nos registradores de segmento. Existe um formato especial para cada uma destas instruções quando elas operam com registradores de segmento. A instrução MOV não pode copiar o conteúdo de um registrador de segmento diretamente para outro registrador de segmento.

As instruções POP e MOV não podem ser utilizadas para armazenar um valor no registrador CS (de códigos); somente as instruções de desvios longes (far) podem fazê-lo. Quando o operando destino é o registrador SS (stack segment), as interrupções são desabilitadas até a próxima instrução.

O prefixo de comprimento de operando de 16 bits não é necessário para as instruções que transferem dados entre os registradores de 32 bits e os registradores de segmento.

Instruções de Transferência de Controle Distante (far)

As instruções de transferência de controle para posições distantes, desviam a execução para um destino localizado em outro segmento. Isto é possível armazenando-se novo valor no registrador CS. O destino é especificado por um ponteiro de 16 bits para o seletor de segmento e 32 bits para o offset dentro do segmento. Este ponteiro (ponteiro distante) pode ser um operando imediato ou valor em memória.

Far CALL é uma instrução de chamada intersegmentos que armazena os valores de EIP e CS no stack.

Far RET é uma instrução de retorno intersegmentos que recupera os valores de EIP e CS do stack.

Instruções de Ponteiro de Dados

As instruções de ponteiro de dados armazenam um ponteiro distante (far) nos registradores do 80486. Um ponteiro distante (ou longe) é formado por 16 bits de seletor de segmento e 32 bits de offset dentro deste segmento. O primeiro é armazenado em um registrador de segmento e o segundo em um registrador geral.

LDS (Load pointer Using DS) copia o ponteiro longe do operando fonte no registrador DS e em um registrador geral. O operando fonte tem que ser uma posição de memória e o destino um registrador geral.

LES (Load pointer using ES) realiza a mesma operação que LDS mas com o registrador ES.

LFS (Load Pointer using FS) idem para o registrador FS.

LGS (Load Pointer using GS) idem para o registrador GS.

LSS (Load Pointer Using SS) idem para o registrador SS. É importante observar que as interrupções não são desabilitadas automaticamente durante esta operação.

4. MODO VIRTUAL DE OPERAÇÃO

GERENCIAMENTO DE MEMÓRIA

O gerenciamento de memória é um mecanismo de hardware que permite ao sistema operacional criar ambientes simplificados para a execução de programas. Por exemplo; quando vários programas estão rodando ao mesmo tempo, cada um deve estar em seu próprio espaço de endereçamento. Caso eles compartilhassem o mesmo espaço, estariam sujeitos a complexos procedimentos de verificação para evitar a interferência de um no outro, o que acarretaria entre outros inconvenientes, um elevado consumo de tempo.

O gerenciamento de memória consiste na paginação (também chamado paginamento) e na segmentação. A segmentação é utilizada para fornecer a cada programa um espaço de endereçamento independente e protegido. A paginação simula um grande espaço de endereçamento em pequena quantidade de RAM auxiliada pelo armazenamento em disco. Quando vários programas estão rodando ao mesmo tempo, qualquer um dos mecanismos pode ser utilizado para evitar a interferência de um programa em outro.

A segmentação permite à memória ser completamente desestruturada e simples, semelhante aos antigos modelos de 8 bits. Entretanto, pode ser utilizada de forma altamente estruturada com traduções de endereços e proteções. O gerenciamento de memória aplica-se a unidades chamadas de segmento. Cada segmento é um espaço de endereços protegidos e independentes. O acesso aos segmentos é controlado por dados que descrevem seu tamanho, o nível de privilégio requerido para o seu acesso, os tipos de referência à memória que podem ser feitos (busca por instruções, stacks, etc.) e se o segmento está ou não presente na memória.

Em uma arquitetura simples de memória, todos os endereços referem-se ao mesmo espaço de endereçamento. Este é o modelo utilizado nos microcomputadores de 8 bits. O endereço lógico é o próprio endereço físico. O 80486 pode trabalhar desta forma mapeando-se todos os segmentos no mesmo endereço físico e desabilitando-se o paginamento. Isto pode ser feito quando da atualização de um projeto antigo para a arquitetura de 32 bit sem utilizar-se de novas características desta última.

É possível uma aplicação utilizar-se parcialmente da arquitetura segmetada. Por exemplo, um problema comum que ocasiona erros de software é o crescimento descontrolado do stack sobre áreas de dados ou de programas. A segmentação é um modo eficiente de se previnir este problema. O stack pode ser posicionado em um espaço de endereços independente das áreas de códigos e de dados. Deste modo teríamos um segmento independente para cada tipo

de dado. Se por ventura a manipulação do stack invadir uma área não estipulada para ele, um mecanismo de hardware irá provocar uma interrupção do processamento que permitirá ao programa recuperar-se desta situação.

A segmentação de hardware traduz endereço segmentado lógico em endereço contíguo não segmentado, chamado endereço linear. Caso a paginação estiver desabilitada, o endereço físico será o próprio segmento linear. O endereço físico aparecerá no barramento do processador e será encaminhado para fora do mesmo.

O mecanismo de paginação é utilizado para simular um largo espaço de endereços não-segmentados em uma pequena região de memória em conjunto com armazenamento em disco. Podemos deste modo acessar estruturas de dados maiores do que a memória existente, em virtude de utilizarmos o armazenamento em disco como auxiliar.

A paginação é aplicada em unidades de 4KBytes chamadas de páginas. Quando um programa tenta acessar uma página que encontra-se em disco, o mesmo é interrompido de uma forma especial. O conteúdo de todos os registradores é armazenado de forma a permitir o reinício da instrução que fez o acesso indevido. O sistema operacional lê a página do disco, atualiza o mapeamento para o endereço linear e físico correspondente à nova página e reinicia o programa. Este processo é totalmente transparente para o software.

O mecanismo de paginação somente é habilitado colocando-se o bit 31 do registrador CR0 em 1. Esta operação normalmente é realizada pelo Sistema Operacional. O endereço linear será utilizado como físico. Isto é feito quando um projeto originalmente desenvolvido para 16 bits é atualizado para uma arquitetura de 32 bits. Um sistema operacional escrito para 16 bits não se utiliza de paginação uma vez que seu espaço de endereço é muito pequeno (64 KBytes), o que torna mais simples a transferência de segmentos inteiros entre a RAM e o disco ao invés de páginas individuais.

Todo este mecanismo é habilitado por sistemas operacionais que suportam a demanda por páginas virtuais em memória, como o UNIX. A paginação é transparente para softwares de aplicação, isto significa que um sistema operacional que suporta aplicações escritas para processadores de 16 bits, pode rodar estes programas com a paginação habilitada. Ao contrário da paginação, a segmentação não é transparente para os programas aplicativos. Eles devem rodar somente nos segmentos para os quais foram designados.

Modelo Segmentado

Um modelo segmentado é escolhido tendo como parâmetros a confiabilidade e o desempenho. Por exemplo: um sistema que possui muitos programas dividindo informações em tempo real, obteria a máxima performance a partir de um modelo que controla as referências à memória via hardware. Este sistema poderia ser multissegmentado.

Em oposição a este sistema, poderíamos considerar outro onde existe somente um único programa em busca de máximo desempenho a partir de um modelo sem segmentos (flat model). A eliminação de ponteiros para acesso a outros segmentos (far pointers) e prefixos alternativos para registradores de segmento (segment override) permite a redução da quantidade de códigos e o incremento da velocidade de execução.

MODELO FLAT

É o modelo mais simples. Os segmentos são mapeados para todo o espaço físico. Um offset pode se referenciar à área de códigos ou de dados pois este modelo elimina todo o mecanismo de segmentação. Isto deve ser feito para ambientes de programação como o UNIX que suportam paginamento mas não suportam segmentação.

Um segmento é definido por um descritor de segmento. Ao menos dois descritores são criados no modelo flat: um para referências a códigos e outro para referências a dados. Ambos descritores possuem o mesmo endereço de base. Sempre que a memória é acessada, o conteúdo de um dos registradores de segmento é utilizado para selecionar um descritor de segmento. O descritor de segmento fornece o endereço base do segmento e seu limite, bem como as informações de controle de acesso.

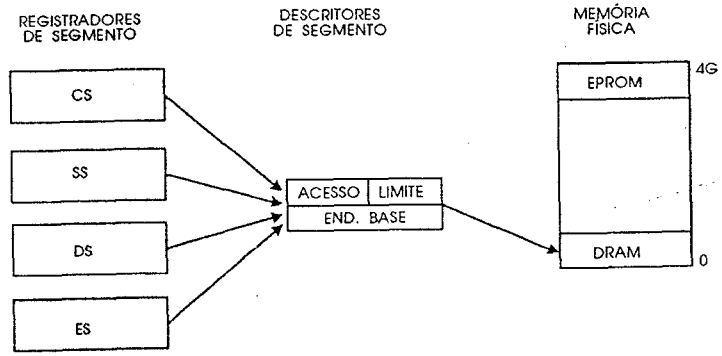


Figura 3.27 - Modelo Flat

Usualmente, a memória ROM é colocada no topo do endereço físico, pois o processador inicia a execução a partir de 0FFFFFFF0H. E a RAM é colocada no endereço mais baixo pois o registrador DS após o reset tem o conteúdo igual a zero.

No modelo flat, cada descritor tem como endereço base 0 e como limite de segmento 4 gigabytes.

MODELO FLAT PROTEGIDO

Este modelo é similar ao anterior exceto pelo fato de que os limites de segmento incluem somente a faixa de endereços na qual efetivamente existe memória. Uma exceção de proteção geral ocorrerá sempre que se tentar algum acesso à memória não implementada.

Ainda que um programa não tente acessar diretamente uma região de memória inexistente, isto pode ocorrer devido a erros de programação (bugs). Sem o mecanismo de proteção, este problema pode interromper a execução repentinamente. A partir de um sistema protegido, a ocorrência de um acesso indevido permite, por exemplo, que seja apresentado em vídeo o endereço onde se encontra a instrução cuja execução apresenta problemas.

Um exemplo do modelo flat protegido é apresentado na figura 3.28

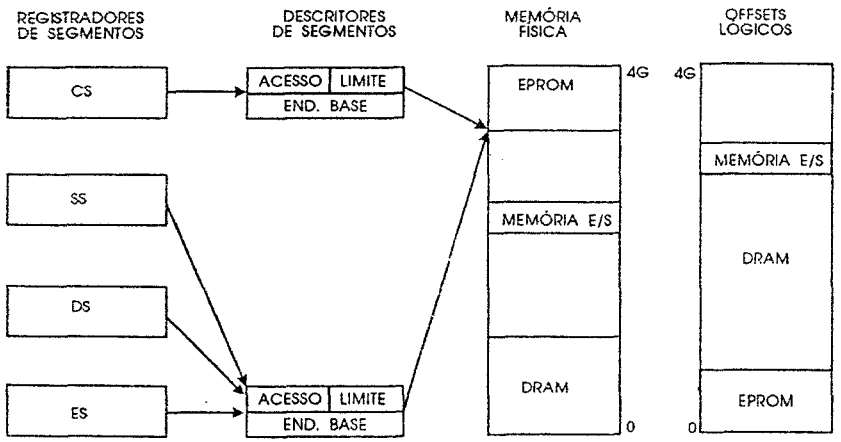


Figura 3.28 - Modelo Flat Protegiido

MODELO MULTISSEGMENTADO

É o modelo mais complexo, onde são utilizados todos os recursos da segmentação. Cada programa possui uma tabela de descritores de segmento própria e seus respectivos segmentos. É possível que mais de um programa acesse um mesmo segmento, mas cada acesso pode ser individualmente controlado.

Até seis segmentos podem estar prontos para uso imediato. São aqueles cujo seletor encontra-se armazenado em um dos registradores de segmento.

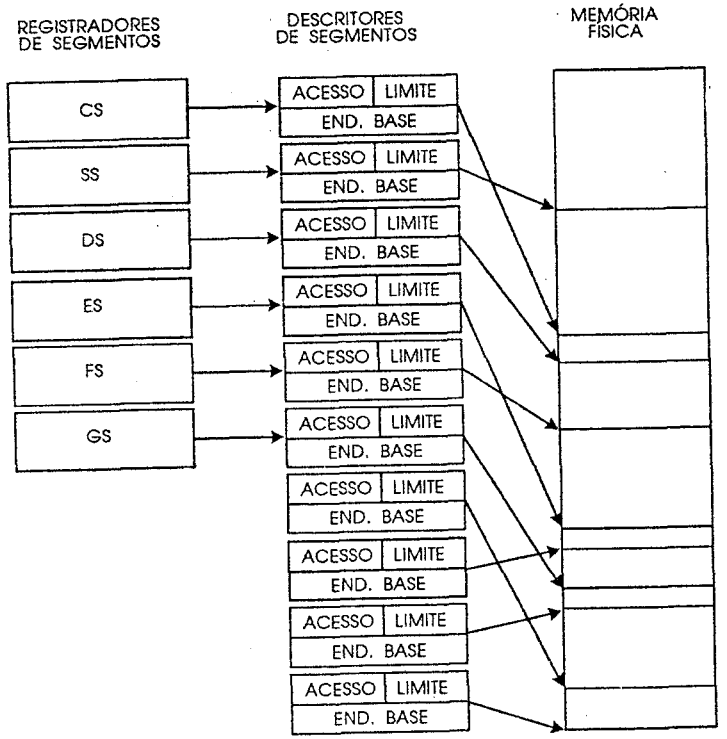


Figura 3.29 - Modelo MultiSSegmentado.

Cada segmento tem o seu espaço de endereço em separado. Mesmo quando dispostos em blocos de endereços adjacentes, o mecanismo de segmentação evita o acesso ao conteúdo de um segmento quando da leitura dos endereços finais de outro. Este procedimento de leitura após o final do segmento acarreta exceção de proteção geral.

O mecanismo de segmentação permite a execução somente da faixa de endereços especificada pelo descritor. É responsabilidade do sistema operacional alocar faixas de endereços em separado para cada segmento. Em algumas situações, pode ser interessante que mais de um segmento divida uma mesma porção de endereços.

Tradução do Segmento

Um endereço lógico é formado por um seletor de 16 bits para o seu segmento e um offset de 32 bits dentro do segmento. Um endereço lógico é traduzido em linear pela soma do offset com o endereço base do segmento. O endereço base é proveniente do descritor de segmento, uma estrutura de dados em memória que fornece o tamanho e a localização do segmento, bem como as informações de controle de acesso. O descritor de segmento pode estar em duas tabelas: tabela de descritor global (GDT, Global Descriptor Table) ou na tabela de descritor local (LDT, Local Descriptor table). Existe somente uma tabela global para todos os programas no sistema e uma tabela local para cada programa. O sistema operacional pode permitir que mais de um programa se utilize da mesma tabela local. Em alguns casos, o sistema não se utiliza de LTD e somente a GDT é usada pelos programas.

Cada endereço lógico é associado a um segmento (mesmo que o sistema defina todos os segmentos para o mesmo espaço de endereço linear). Um programa pode ter milhares de segmentos, todavia apenas seis estão disponíveis ao mesmo tempo para uso imediato. São aqueles cujo seletor encontra-se carregado no processador. O seletor armazena a informação necessária para a tradução do endereço lógico no endereço linear correspondente.

Existem registradores de segmentos específicos para cada tipo de acesso à memória: dados, códigos e stacks. Eles contêm os seletores para os segmentos em uso. O acesso a outro segmento, requer o armazenamento de um registrador de segmento a partir de uma forma especial da instrução MOV. Um máximo de 4 espaços para dados estarão disponíveis ao mesmo tempo, quando do uso de todos os seis registradores de segmento.

Quando um seletor de segmento é carregado, o endereço base, o limite de segmento e a informação de controle de acesso são carregados ao mesmo tempo no registrador de segmento. O processador não faz novas referências à tabela de descritores até um novo seletor de segmento ser carregado. A informação armazenada no processador permite a tradução de endereço sem ciclos extras de barramento. Em sistemas com múltiplos processadores acessando as mesmas tabelas de descritores, é responsabilidade do programa recarregar os registradores de segmento quando a tabela é alterada. Caso isto não seja feito, um descritor de segmento antigo armazenado em um registrador poderá ser usado após sua versão em memória ter sido atualizada.

O seletor de segmento possui 13 bits para ser utilizado como índice dentro de uma tabela de descritor. O índice é múltiplo de 8 (número de bytes que

formam um descritor) e é somado ao endereço de 32 bits que é base da tabela de descritores. O endereço base é providenciado ou pelo registrador GDTR, no caso de acesso à tabela global; ou pelo registrador LDTR no caso de acesso à tabela local. São eles que armazenam o endereço linear de início das tabelas de descritores. Um bit no seletor de segmento especifica qual tabela deve ser utilizada, conforme a figura a seguir.

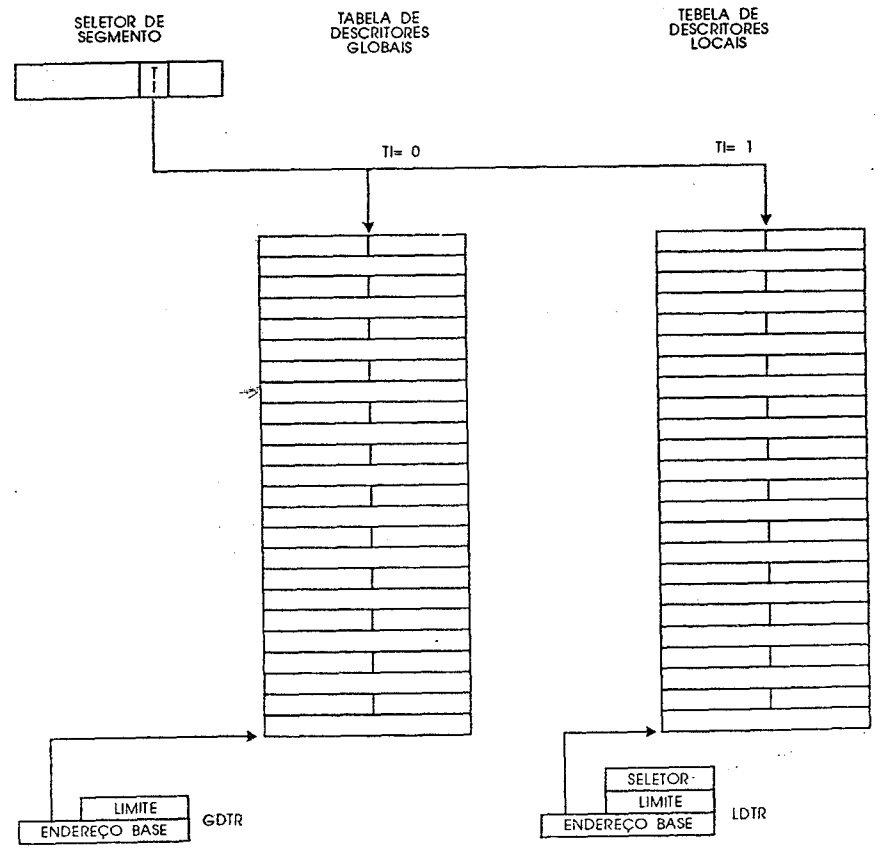


Figura 3.30 - Bit TI seleciona a tabela de descritores.

O endereço traduzido é chamado de endereço linear, conforme mostra a figura a seguir. Se a paginação estiver desabilitada, ele é também o endereço físico. Caso a paginação seja usada, um segundo nível de tradução de endereço produzirá o endereço físico.

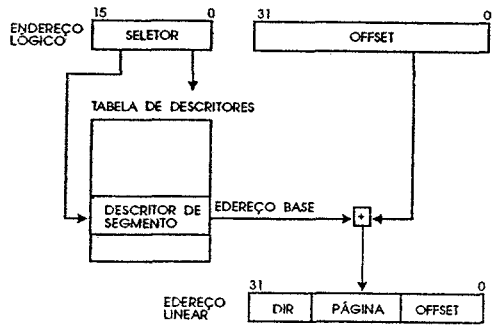


Figura 3.31 - Tradução de segmento

REGISTRADORES DE SEGMENTO

Cada tipo de referência à memória está associado a um registrador de segmento. Ou seja, o acesso a códigos, dados e stack é definido pelo registrador de segmento respectivo. Mais segmentos podem estar disponíveis pelo armazenamento de seus seletores em um destes registradores durante a execução do programa.

Todos os registradores de segmento possuem uma parte visível e outra invisível. Existem formatos para a instrução MOV que permitem o armazenamento na parte visível. A região invisível é sempre carregada pelo processador.

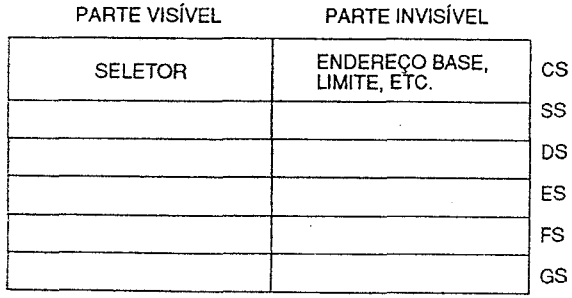


Figura 3.32 - Registradores de Segmento

As operações que carregam estes registradores são instruções para programas aplicativos e podem ser de dois tipos:

1. Instruções de armazenamento direto; como as de MOV, POP, LDS, LSS, LGS e LFS. Estas instruções explicitamente referenciam-se aos registradores de segmento.
2. Instruções de armazenamento implícito como as versões intersegmentos (far) das instruções CALL e JMP. Elas alteram o conteúdo do registrador CS como conseqüência da instrução.

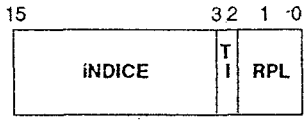
Quando uma destas instruções é executada, a parte visível do registrador é carregada com um seletor de segmento. Automaticamente, o processador busca pelo endereço de base, o limite, o tipo e outras informações provenientes da tabela de descritores e carrega a parte invisível do registrador.

Como a maior parte das instruções acessam a segmentos cujos seletores já foram previamente carregados nos registradores de segmento, o processador pode somar endereços dos offsets aos endereços de base dos segmentos sem prejuízo à performance.

SELETORES DE SEGMENTO

Um seletor de segmento aponta para a informaçãc que define um segmento, chamado de descritor de segmento. Um programa pode ter mais do que os seis segmentos cujos seletores ocupam os registradores em dado instante. Para acessar novo segmento, o programa deve alterar o conteúdo do registrador de segmento a partir de um formato especial da instrução MOV.

Um seletor identifica um descritor de segmento pela especificação da tabela de descritores e pelo próprio descritor dentro da tabela. Os seletores de segmento são para os programas aplicativos partes de um ponteiro, mas seus valores são usualmente designados ou modificados por link editores ou link carregadores, não por programas aplicativos. A figura a seguir, apresenta o formato de um seletor de segmento.



Indicador de tabela
nível de privilégio solicitado
(oo = mais privilégio, 11 = menos privilégio)

Figura 3.33 - Seletor de segmento.

Índice: Seleciona um dos 8192 descritores em uma tabela de descritores. O processador multiplica o valor do índice por 8 (número de bytes que formam um descritor) e soma o resultado ao endereço de base da tabela (proveniente do registrador GDTR ou do LDTR).

Bit indicador de tabela (Table Indicator bit): especifica qual tabela de descritores deve ser utilizada. Em zero seleciona a GDT; em um a LDT.

Nível de privilégio solicitado (Requested Privilege Level): quando este campo possui nível de privilégio cujo valor é maior (ou seja menos privilegiado), do que o programa, ele ignora o nível de privilégio do programa. Quando um programa usa um seletor de segmento menos privilegiado, o acesso à memória ocorre no menor nível de privilégio. Isto é feito como prevenção contra violações à segurança onde um programa menos privilegiado se utilizaria de outro mais privilegiado para acessar dados protegidos.

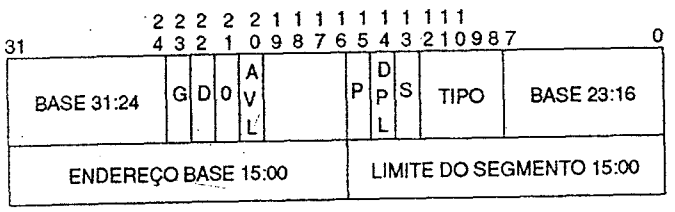
Por exemplo, utilitários de sistema ou device drivers devem rodar com alto nível de privilégio a fim de acessar instalações protegidas como registradores de controle de periféricos para interfaces. Mas eles não devem interferir em outras instalações protegidas mesmo se solicitado por programa de menor privilégio.

Como o primeiro descritor da GDT não é utilizado pelo processador, um seletor que contém índice e indicador de tabela igual a zero, ou seja, que aponta para a primeira entrada da GDT; é usado como "seletor nulo". O processador não gera uma exceção quando um registrador de segmento é carregado com um seletor nulo (excetuando-se os registradores CS e SS). A exceção somente é gerada quando um acesso à memória é tentado a partir de um seletor nulo. Esta característica pode ser utilizada para inicializar registradores de segmentos disponíveis.

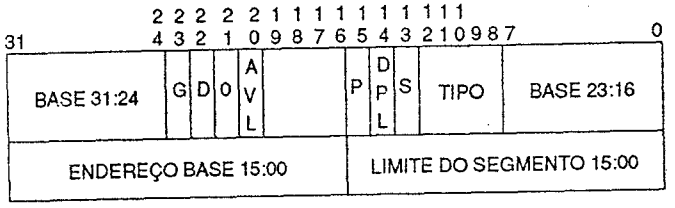
DESCRITORES DE SEGMENTOS

Um descritor de segmento é uma estrutura de dados em memória que fornece ao processador o tamanho, a localização, o controle e o status de um segmento. Os descritores são criados normalmente a partir de compiladores, linkers, carregadores ou sistemas operacionais, mas não por programas aplicativos. A figura a seguir, ilustra os formatos de dois descritores genéricos. Todos os tipos de descritores de segmentos possuem um destes formatos.

DESCRITORES UTILIZADOS PARA SEGMENTOS DE DADOS E DE CÓDIGOS DE APLICAÇÕES



DESCRITORES UTILIZADOS PARA SEGMENTOS ESPECIAIS DO SISTEMA



AVL	Disponível para uso do programa de sistema
BASE	Endereço base do segmento
DPL	Nível de privilégio do descritor
S	Tipo do descritor (0 = sistema, 1 = aplicação)
G	Granularidade
LIMITE	Limite do segmento
P	Presença do segmento
TIPO	Tipo do segmento
D	Comprimento default da operação reconhecido somente nos descritores de segmento de código (0 = segmento de 16 bits, 1 = segmento de 32 bits)

Figura 3.34 - Descritores de segmento.

Base *(Base)*: Define a localização do segmento dentro do espaço físico de 4 gigabytes de endereços. As três áreas de endereço base são dispostas juntas para formar um valor de 32 bits. A base do segmento deve ser um valor múltiplo de 16 a fim de permitir a maior performance possível na execução dos programas.

Bit de granularidade *(Granularity bit)*: Habilita a multiplicação do campo de limite por um fator de 4096 (2^{12}). Quando este bit é igual a zero, o limite de segmento é interpretado em unidades de um byte; quando igual a 1, o limite de segmento é interpretado em unidades de 4K bytes (uma página). Observe que os 12 bits menos significativos do endereço não são testados quando a multiplicação é utilizada. Por exemplo, um limite de 0 com o bit de granularidade em 1, resulta em offsets válidos de 0 até 4095. Observe também que somente o campo de limite é afetado. O endereço base permanece com granularidade de byte.

Limite *(Limit)*: Define o tamanho do segmento. As duas áreas de limite são dispostas juntas pelo processador para formar um campo de 20 bits. Este campo será interpretado de duas formas possíveis em função do bit de granularidade.

- 1. Se o bit de granularidade for igual a zero, o limite tem um valor de um byte até 1 megabyte, em incrementos de 1 byte.
- 2. Se o bit de granularidade for igual a 1, o limite tem valor de 4 kilobytes até 4 gigabytes, em incrementos de 4 kilobytes.

O endereço lógico deverá ser um offset dentro da faixa compreendida entre 0 e o limite. Fora disso, o offset provocará uma exceção. Segmentos cuja expansão é do endereço maior para o menor, reverte o sentido do campo de limite.

Bit S: Determina se um segmento é de sistema ou se é um segmento de dados ou códigos. Quando em 1, trata-se de um segmento de dados ou códigos. Quando em zero, é um segmento de sistema.

Bit D: Indica o comprimento default para operandos e endereços efetivos. Se o bit D for igual a 1, é adotado 32 bits para operandos e endereços efetivos. Se for zero, assume-se 16 bits para ambos.

Tipo *(Type)*: A interpretação deste campo depende do descritor se referenciar a um segmento de sistema ou não. Em um descritor de memória, o campo especifica os tipos de acessos que podem ser feitos a um segmento, bem como a direção do seu crescimento.

Número	E	W	A	Tipo do descritor	Descrição
0	0	0	0	Dados	Somente para leitura
1	0	0	1	Dados	Somente para leitura - acessado
2	0	1	0	Dados	Leitura e escrita
3	0	1	1	Dados	Leitura e escrita - acessado
4	1	0	0	Dados	Somente para leitura - expansão para baixo
5	1	0	1	Dados	Somente para leitura - expansão para baixo - acessado
6	1	1	0	Dados	Leitura e escrita - expansão para baixo
7	1	1	1	Dados	Leitura e escrita - expansão para baixo - acessado
Número	**C**	**R**	**A**	**Tipo do descritor**	**Descrição**
8	0	0	0	Códigos	Somente para execução
9	0	0	1	Códigos	Somente para execução - acessado
10	0	1	0	Códigos	Execução e leitura
11	0	1	1	Códigos	Execução e leitura - acessado
12	1	0	0	Códigos	Somente para execução - conforme
13	1	0	1	Códigos	Somente para execução - conforme - acessado
14	1	1	0	Códigos	Execução e leitura - conforme
15	1	1	1	Códigos	Execução e leitura - conforme - acessado

Tabela 3.10 - Tipos de segmentos de aplicações

Para segmentos de dados, os três bits mais baixos são interpretados como expansão para baixo (E, expand-down); escrita habilitada (W, write-enable); e acessado (A, accessed). No caso de segmento de códigos, a interpretação a estes bits é: conforme (C, conforming); leitura habilitada (R, read enable); e acessado (A, accessed).

Os segmentos de dados podem ser apenas para leitura ou para leitura e escrita. Segmentos de stack são exemplos de segmentos que devem ser de

escrita e leitura. Se por ventura o registrador SS for carregado com um seletor para outro tipo de segmento, será gerada uma exceção de proteção geral. Caso o segmento de stack necessitar de tamanho variável, poderá ser definido como um segmento de dados com expansão para baixo. O significado do limite de segmento é revertido neste caso. Enquanto um offset na faixa entre 0 e o limite do segmento é válido para outros tipos de segmentos (e fora desta faixa uma exceção de proteção geral é gerada), em um segmento com expansão para baixo são estes os offsets que geram uma exceção. Ou seja, os offsets válidos para este tipo de segmento são aqueles que ocasionam exceções nos demais. Segmentos com expansão para cima devem ser endereçados por offsets que são iguais ou menores do que o limite de segmento. Offsets em segmentos com expansão para baixo devem ser sempre maiores do que o limite do segmento. Esta interpretação do limite do segmento faz com que o espaço de memória seja alocado na base do segmento quando o seu limite é incrementado; o que é correto para segmentos de stacks uma vez que eles crescem em direção aos endereços menores.

Segmentos de códigos podem ser apenas de execução ou de execução e leitura. Este último é utilizado por sistemas que armazenam valores constantes em conjunto com códigos de instruções em uma memória ROM. As constantes poderão ser lidas a partir ou do uso do registrador CS como prefixo de uma instrução para dados, ou colocando-se o seletor do segmento de códigos em um registrador de segmento de dados.

Os segmentos de códigos podem ser conforme ou não-conforme. A transferência da execução de um programa para outro mais privilegiado e conforme, mantém o atual nível de privilégio. Enquanto que uma transferência entre programas de diferentes níveis e não-conforme acarreta uma exceção de proteção geral, a menos que seja utilizado um task gate.

O campo de tipo também reporta se o segmento foi acessado. Inicialmente, os descritores de segmento informam um seguimento como acessado. Se o campo de tipo é alterado para segmento não acessado, o processador retorna o valor original quando do acesso ao mesmo. Zerando-se e testando-se o bit menos significativo do campo de tipo, o software pode monitorar o uso do segmento. Este bit também é chamado de bit de acesso.

Por exemplo, um sistema de desenvolvimento de programas pode apagar todos os bits de acesso para os segmentos de uma aplicação. Caso a aplicação apresente problemas, os estados destes bits podem ser utilizados para gerar um mapa de todos os segmentos que forma, acessados pela aplicação. Ao contrário dos breakpoints fornecidos pelo mecanismo de depuração interno ao processador, a informação de uso aplica-se a segmentos ao invés de endereços físicos.

O processador atualiza o campo de tipo mesmo quando o acesso ao segmento é para leitura

DPL (Descriptor Privilege Level): define o nível de privilégio do segmento. É utilizado para o controle do acesso ao segmento pelo mecanismo de proteção.

Bit de presença do segmento (*Segment Present bit*): se este bit for igual a zero, uma exceção de "segmento não está presente" é gerada quando um seletor para o descritor é carregado em um registrador de segmento. Isto é utilizado para detectar-se um acesso a segmento que não se encontra disponível. Isto pode ter ocorrido em função de um procedimento executado pelo sistema para liberar áreas de memória. Itens em memória que não estavam em uso na ocasião foram portanto removidos. Quando este tipo de gerenciamento de memória é fornecido de maneira transparente aos programas aplicativos, é chamado de memória virtual. Um sistema pode manter uma quantidade total de memória virtual bem maior do que a memória física, mantendo poucos segmentos na mesma em um mesmo instante.

A figura 3.35, apresenta o formato de um descritor quando o bit de segmento presente é igual a zero. Neste caso, o sistema está livre para utilizar a área remanescente para armazenar informações independentes do segmento que ocupou a área liberada.

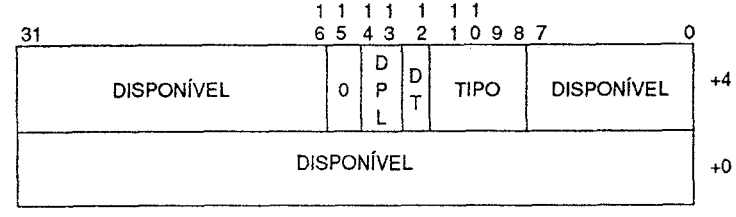


Figura 3.35 - Descritor de segmento (com segmento ausente).

TABELAS DE DESCRITORES DE SEGMENTOS

Uma tabela de descritores de segmento é um arranjo de descritores. Elas podem ser de dois tipos:

- ❑ A tabela de descritores globais (GDT).
- ❑ A tabela de descritores locais (LDT).

Existe uma única GDT para todas as tarefas, e uma LDT para cada uma delas. A figura a seguir, apresenta o arranjo das tabelas.

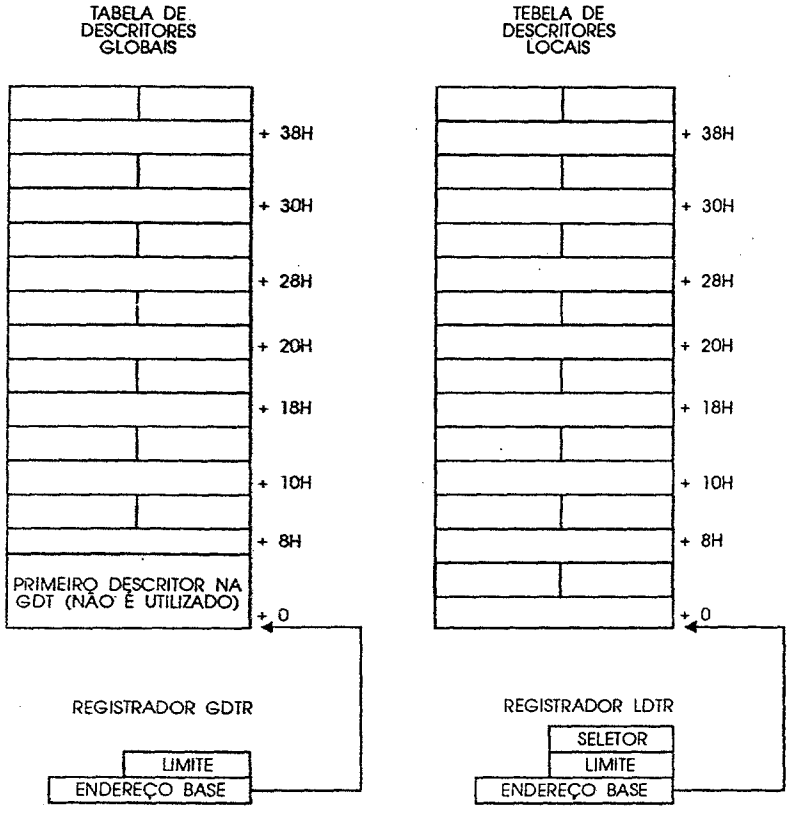


Figura 3.36 - Tabela de descritores.

Uma tabela de descritores possui comprimento variável e pode ter um máximo de 8192 descritores (2^{13}). O primeiro descritor da GDT não é utilizado pelo processador. Um seletor de segmento que aponte para este descritor é considerado nulo e não gera exceção quando carregado em um registrador (a menos que se tente acessá- lo).

REGISTRADORES DE BASE DAS TABELAS DE DESCRITORES

O processador localiza as tabelas de descritores globais e de interrupções respectivamente pelos registradores GDTR e IDTR. Ambos armazenam endereços lineares de base de 32 bits; além de 16 bits de limite que corresponde ao tamanho destas tabelas. Quando estes registradores são carregados, uma estrutura de 48 bits conhecida como pseudodescritor é acessada em memória. A figura a seguir, mostra como ela é. Tanto a GDT quanto a IDT devem ser alinhadas em múltiplos de 16 bytes para maximização da performance e devido ao preenchimento das linhas do cache.

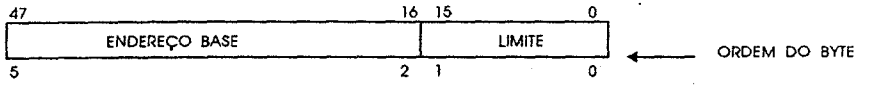


Figura 3.37 - Formato do pseudodescritor

O limite é expresso em bytes. Assim como nos segmentos, o valor do limite é somado à base para descobrir-se o endereço do último byte válido. Limite igual a zero resulta em exatamente um byte válido. Como os descritores de segmento possuem 8 bytes cada, o limite deverá ser um número múltiplo de oito subtraído de um (isto é; 8N-1). O registrador GDTR é lido pela instrução LGDT e escrito pela instrução SGDT. Similarmente, o registrador IDTR é lido pela LIDT e escrito pela SGDT.

A terceira tabela de descritores é a Local (LDT), identificada pelo registrador LDTR de 16 bits que armazena o seletor correspondente à tabela. As instruções LLDT e SLDT acessam para leitura e escrita respectivamente o registrador LDTR. O endereço de base e o limite da LDT são armazenados também pelo LDTR, todavia estes valores são obtidos automaticamente pelo processador a partir do descritor correspondente à LDT.

Tradução de Página

Um endereço linear é formado por 32 bits em um espaço de endereçamento não-segmentado e uniforme. Este espaço pode ocupar todos os endereços físicos, atingindo portanto 4 Gigabytes; ou a partir da paginação, podemos simular este espaço em uma pequena quantidade de RAM acrescida de armazenamento em disco. Quando este mecanismo é utilizado, o endereço linear precisa ser traduzido em físico ou, deve ser gerada uma exceção caso este não esteja presente em memória. A partir desta exceção, o sistema operacional tem a oportunidade de trazer a página do disco para ser processada.

A paginação difere da segmentação pelo uso de páginas de tamanho fixo e pequeno. No 80486, as páginas têm sempre 4Kbytes. Se em dado instante somente a segmentação é utilizada, uma estrutura de dados, por exemplo, deve estar toda em memória para ser acessada. No caso da paginação, esta mesma estrutura pode estar parte em memória e parte em disco.

A informação que traduz o endereçamento linear em físico e em exceções é conhecida como tabela de páginas e fica armazenada em memória em uma estrutura de dados. Assim como na segmentação, uma vez acessada, a tabela permanece interna ao processador para minimizar os ciclos necessários à tradução do endereço. Ao contrário da segmentação, estes registradores são totalmente invisíveis aos programas aplicativos.

O mecanismo de paginação visualiza os 32 bits de endereço linear como tendo três partes: dois índices de 10 bits para as tabelas de páginas e um offset de 12 bits dentro da página endereçada. Como tanto os endereços lineares quanto os físicos em memória encontram-se alinhados em múltiplos de 4 Kbytes, não existe a necessidade de modificarmos os 12 bits menos significativos do endereço. Estes bits não sofrem interferência do mecanismo de paginação, quer ele esteja ou não habilitado. Encontramos aqui uma outra diferença em relação à segmentação pois, esta última pode ter seus segmentos com endereço inicial em qualquer locação.

Os 20 bits mais significativos são utilizados para indexar as tabelas de páginas. Se cada página no endereço linear fosse mapeada em uma única tabela de páginas em RAM, 4 Megabytes seriam necessários. Isto não é feito, o mapeamento é realizado a partir de dois níveis de tabelas. O primeiro conhecido por diretório de páginas. Ele mapeia os 10 bits mais significativos do endereço linear para o segundo nível de tabelas de páginas. O segundo nível de tabelas de páginas mapeia os 10 bits intermediários do endereço linear para a base do endereço da página em memória física (conhecida como endereço da estrutura da página).

Uma exceção pode ser gerada em função do conteúdo da tabela de páginas ou do diretório de páginas. Uma exceção dá ao sistema operacional a chance de trazer uma página do disco para a memória. Deste modo, como o segundo nível de páginas pode ser enviado ao disco, o mecanismo de paginação pode suportar o mapeamento de todo o espaço de endereçamento linear, usando poucas páginas em memória.

O registrador CR3 armazena o endereço do diretório de páginas. Por este motivo também é chamado de registrador de base do diretório de páginas (PDBR). Os 10 bits mais significativos do endereço linear são multiplicados por quatro (que é o número de bytes de cada entrada na tabela de páginas) e somados ao valor do PDBR a fim de obter-se o endereço físico de uma entrada na tabela de diretório de páginas.

Quando uma entrada no diretório de páginas é acessada, uma série de verificações é realizadas. Podem-se gerar exceções caso a página seja protegida ou caso não se encontre em memória. Se isto não ocorrer, os 20 bits superiores da entrada na tabela de páginas são utilizados como endereço da estrutura da página no segundo nível de tabelas. Os 10 bits intermediários do endereço linear são multiplicados por 4 (novamente o tamanho de cada entrada na tabela de páginas) e concatenados com o endereço da estrutura da página para se obter o endereço físico no segundo nível de tabela de páginas.

Novas verificações são feitas e exceções podem ocorrer. Caso contrário, os 20 bits superiores da entrada do segundo nível de tabela de páginas são concatenados com os 12 bits inferiores do endereço linear para formar o endereço físico do operando em memória.

Embora este processo pareça complicado, todo ele é realizado com um mínimo de tempo. O processador possui um cache para as entradas da tabela de páginas chamado Translation Lookaside buffer (TLB). O TLB satisfaz a maior parte das solicitações por páginas para leitura. Um ciclo de bus extra ocorre somente quando nova página é acessada. O tamanho de página de 4 Kbytes é grande o bastante para assegurar poucos acessos às tabelas de páginas, comparando-se com o número de ciclos gastos com dados e códigos. Ao mesmo tempo em que pode ser considerado pequeno o bastante para fazer um eficiente uso da memória.

A HABILITAÇÃO DA PAGINAÇÃO VIA BIT PG

Se a paginação está habilitada, um segundo estágio da tradução de endereço é usado para gerar o endereço físico a partir do endereço linear. Quando desabilitada, o endereço linear é utilizado como endereço físico.

A paginação é habilitada pelo bit 31 (o bit PG) do CR0 em valor igual a 1. Normalmente, este bit é igualado a um pelo sistema operacional quando da inicialização do software. Este bit deve ter valor igual a 1 sempre que o sistema estiver rodando programa no modo virtual-8086.

ENDEREÇO LINEAR

A figura 3.38, apresenta o formato de um endereço linear:

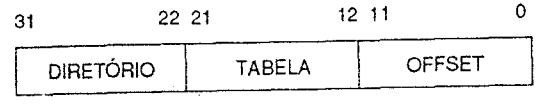


Figura 3.38 - Formato de um endereço linear.

A figura 39 mostra como o processador traduz o endereço linear em campos de diretório, tabela e offset para gerar o endereço físico, utilizando-se de dois níveis de tabelas de páginas. O mecanismo de paginação usa o campo de diretório como um índice para o diretório de páginas, o campo tabela como um outro índice na tabela de páginas determinada pelo diretório de páginas, e o campo de offset para endereçar um operando dentro da página especificada pela tabela de páginas.

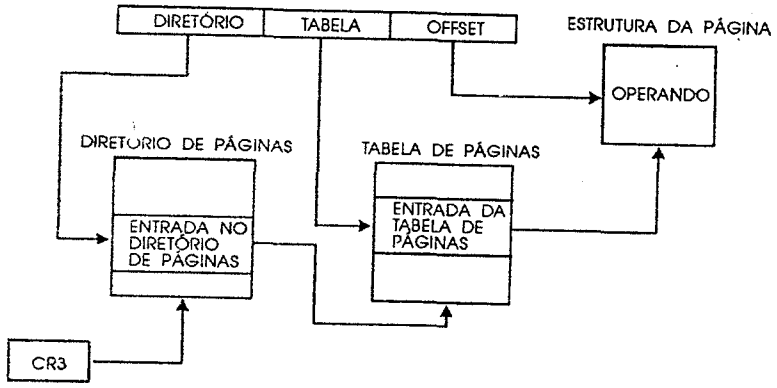


Figura 3.39 - Tradução de Páginas

TABELA DE PÁGINAS

Uma tabela de páginas é uma estrutura com entradas de 32 bits cada. Uma tabela de páginas é ela mesma uma página, e contém 4096 bytes de memória, ou seja, 1 Kilobyte de entradas de 32 bits. Todas as páginas, incluindo diretório e tabelas, são alinhadas a 4 Kilobytes.

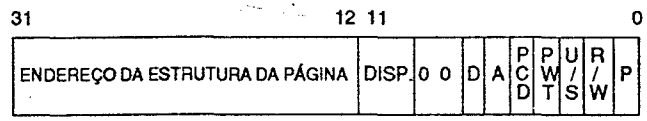
Dois níveis de tabelas são utilizados para o endereçamento de uma página de memória. O primeiro nível é chamado de diretório de página. Ele endereça até 1K de tabela de páginas no segundo nível. Uma tabela de páginas no segundo nível endereça até 1K de páginas na memória física. Todas as tabelas são endereçadas por um diretório, pode-se endereçar portanto 1M ou 2^{20} páginas. Como cada página contém 4K ou 2^{12} bytes, as tabelas de um diretório de páginas podem englobar todo os espaço de endereçamento físico do processador 80486 ($2^{20} \times 2^{12} = 2^{32}$).

O endereço físico do diretório de páginas em uso é armazenado no registrador CR3, conhecido por registrador de base do diretório de páginas (PDBR). O software de gerenciamento de memória tem a opção de utilizar-se de

um único diretório de páginas para todas as tarefas, um para cada tarefa ou uma combinação de ambos.

ENTRADAS NAS TABELAS DE PÁGINAS

Em qualquer nível das tabelas de páginas, as entradas têm o mesmo formato.



P	- PRESENTE
R/W	- LEITURA/ESCRITA
U/S	- USUÁRIO/SUPERVISOR
PWT	- PÁGINA TRANSPARENTE À ESCRITA
PCD	- DESABILITAÇÃO DE CACHE DE PÁGINA
A	- ACESSADO
D	- SUJO (ESCRITA EM CURSO)
DISP	- DISPONÍVEL PARA USO DE PROGRAMADORES DE SISTEMA

NOTA: ZERO INDICA BIT RESERVADO A INTEL. NÃO UTILIZE

Figura 3.40 - Formato de uma entrada na tabela de páginas.

Endereço da Estrutura da Página

O endereço da estrutura da página é o seu endereço de base. Em uma entrada da tabela de páginas, os 20 bits superiores são usados para especificar um endereço da estrutura da página, e os 12 bits inferiores fornecem informações de controle e status para a página. No caso do diretório de página, o endereço da estrutura da página é o endereço de uma tabela de página. Num segundo nível de tabela de página, o endereço da estrutura da página corresponde ao endereço da página que contém instruções ou dados.

Bit de Presença

O bit de presença informa se um endereço da estrutura da página em uma entrada da tabela de páginas está mapeando uma página na memória física. Quando em 1, indica que a página encontra-se na memória.

Quando o bit de presença está em zero, a página não encontra-se em memória , e todo o restante da entrada da tabela de página fica disponível para o sistema operacional, por exemplo, recuperar a informação da página perdida (a partir do disco ou de outro meio qualquer de armazenamento). A figura 3.41, apresenta o formato de uma entrada na tabela de páginas quando o bit de presença é igual a zero.

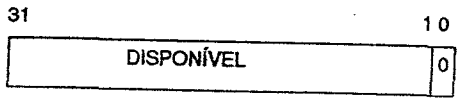


Figura 3.41 - Formato de uma entrada para uma página ausente.

Se o bit de presença estiver zerado em qualquer nível das tabelas de páginas quando uma tentativa é realizada para a tradução de endereço, uma exceção de falha de página é gerada. Em sistemas que suportam a demanda de página em memória virtual, a seguinte seqüência de eventos deverá ocorrer:

1. O sistema operacional copia a página do disco para a memória física.
2. O sistema operacional carrega o endereço da estrutura da página na entrada da tabela e iguala a 1 seu bit de presença. Outros bits como R/W também devem ser atualizados.
3. Como uma cópia da antiga entrada na tabela de páginas pode permanecer no TLB (translation lookaside buffer), o sistema operacional deve esvaziá-lo.
4. O programa que gerou a exceção é então reinicializado.

Como não existe o bit de presença no registrador CR3 para indicar quando o diretório de páginas não está residente em memória, o CR3 deverá sempre apontar um diretório presente na memória física.

Bits "Acessado" e "Modificado"

Estes bits fornecem informações referentes ao uso das páginas em qualquer nível das tabelas. O bit "acessado" reporta leitura ou escrita em curso numa determinada página ou no segundo nível de tabela de páginas. O bit "modificado" reporta uma escrita em uma página.

Com exceção do bit "modificado" em uma entrata do diretório de páginas, estes bits são igualados a 1 pelo hardware; entretanto, o processador não zera nenhum destes bits. O processador atualiza os bits "acessados" em ambos os níveis de tabelas de páginas antes de uma leitura ou escrita numa página. O processador coloca o bit "modificado" em 1 no segundo nível de tabela de páginas antes de uma operação de escrita em endereço mapeado pela entrada da tabela de páginas. O bit "modificado" nas entradas do diretório é indefinido.

O sistema operacional pode usar o bit "acessado" quando precisa criar espaço livre em memória, mandando uma página ou o segundo nível de tabela

de página para o disco. Apagando periodicamente os bits "acessados" nas tabelas de páginas, ele pode ver quais páginas foram utilizadas recentemente. Aquelas que não o foram, são candidatas a saírem da memória e serem armazenadas em disco.

O sistema operacional pode utilizar o bit "modificado" para controlar o envio de páginas para o disco. Sempre que uma página é trazida do disco, o bit deve ser zerado. Caso a página necessite retornar ao disco, uma verificação no bit "modificado" irá informar se ela sofreu ou não alterações e portanto se a operação de colocá-la em disco é ou não necessária

Bits de Leitura/escrita e de Supervisor/usuário

Os bits de leitura/escrita e de supervisor/usuário são utilizados para a verificação de proteção que é feita simultaneamente à tradução do endereço da página acessada.

Bits de Controle do Cache do Nível de Página

Os bits PCD e PWT são usados para o gerenciamento do cache do nível de página. O software pode controlar o sistema de cache de páginas individuais ou do segundo nível de tabelas de páginas, a partir destes bits.

TRANSLATION LOOKASIDE BUFFER

O processador armazena as páginas utilizadas mais recentemente em um cache interno chamado Translation Lookaside buffer ou TLB. Um ciclo de barramento é realizado apenas quando a página solicitada não se encontra no TLB.

O TLB é invisível para o programa aplicativo, mas não para o sistema operacional. Ele deve sempre atualizar o TLB quando alguma entrada nas tabelas de páginas for alterada. Caso contrário, dados antigos que não receberam alterações serão utilizados na tradução de endereços. Todavia, uma alteração em entradas que não se encontrem em memória não requerem a atualização do TLB, uma vez que o mesmo somente retém páginas que se encontrem presente em memória.

O TLB é atualizado quando o registrador CR3 é carregado. Isto pode ocorrer de dois modos:

- **1.** Explicitamente pela instrução MOV . Exemplo:

  MOV CR3,EAX

- **2.** Implicitamente carregado por uma troca de tarefas que altera o conteúdo do CR3.

Uma única entrada no TLB pode ser descartada através da instrução INVLPG. O que é muito útil nos casos em que somente o mapeamento de uma página foi alterado.

Combinando Segmento e Tradução de Página

A figura a seguir sintetiza ambos os estágios da tradução de um endereço lógico para um físico (ver figura 3.42 e figura 3.43). As opções disponíveis em ambos os estágios de tradução do endereçamento podem ser usadas para fornecerem diferentes estilos de gerenciamento de memória.

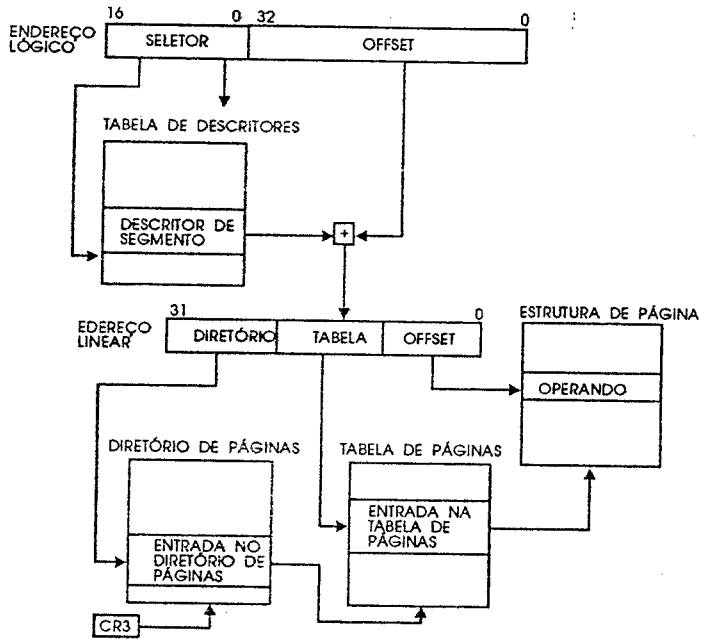


Figura 3.42 - Segmento e tradução de endereço de página.

MODELO FLAT

Ao se utilizar o 80486 com programas escritos para estruturas sem segmentação, é desejável desabilitarmos este mecanismo do processador. O 80486 não possui um controle de modo que permita sua utilização sem segmentos, todavia o mesmo efeito pode ser obtido, mapeando-se os segmentos de código, dados e stack na mesma faixa de endereços lineares. Os offsets de 32 bits usados pelo 80486 podem cobrir todo o espaço de endereçamento linear.

Isto pode ser feito a partir do mecanismo de paginação. Se mais de um programa é executado ao mesmo tempo, este mecanismo permite a cada programa dispor de seu próprio espaço de endereços.

A arquitetura do 80486 permite a interposição de segmentos e páginas, e é possível a existência de situações nas quais um segmento cobre várias páginas e vice-versa. É importante observar que não existe a necessidade de alinhamento de endereços entre os limites das páginas e dos segmentos, todavia, caso isto seja forçado no projeto do sistema, teremos o desempenho máximo do processador.

A figura 3.43, apresenta um modelo de simplificação do mecanismo de gerenciamento da memória a partir da associação de uma tabela de páginas para cada segmento. Uma única entrada no diretório de páginas é fornecida para um segmento, providenciando informações de controle de acesso para o mesmo.

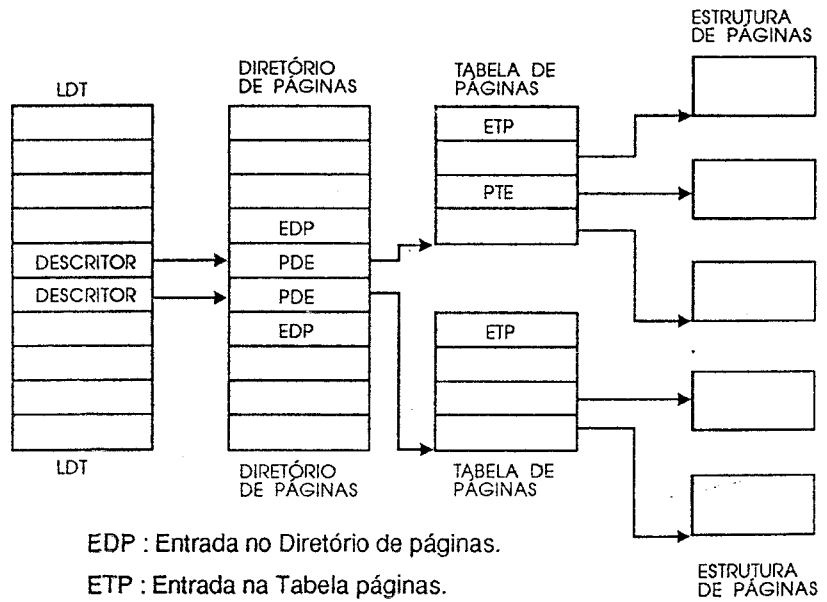


Figura 3.43 - Uma página associada a um segmento próprio.

PROTEÇÃO

O mecanismo de proteção é necessário para assegurarmos a operação de várias tarefas simultaneamente (o multitarefamento). A proteção pode ser utilizada para prevenir que uma tarefa interfira em outra indevidamente.

A proteção funciona tanto para páginas quanto para segmentos. Dois bits em um registrador do 80486 definem o nível de privilégio do programa que está sendo rodado (chamado de nível de privilégio corrente ou CPL). O CPL é verificado durante a tradução da segmentação e da paginação para prevenção de acessos indesejáveis.

Embora não existam meios de desabilitarmos os mecanismos de proteção, podemos obter o mesmo resultado designando-se o nível 0 de privilégio (o mais alto nível) para todos os seletores de segmento, descritores e entradas das tabelas de páginas.

Proteção em Nível de Segmento

Uma das primeiras características que a proteção nos fornece é a de asseguramos que o mau funcionamento de uma rotina não infligirá danos a outra. Dispomos deste modo, de uma importante ferramenta para o desenvolvimento e depuração de programas pois um software depurador ou o próprio sistema operacional, por exemplo, pode-se manter em memória após a interrupção indevida de um programa, permitindo-nos analisar quais as causas da parada do processamento. Em produção, este mecanismo nos permite mais segurança de processamento e possbilita ao sistema operacional uma oportunidade de reinicializar rotinas recuperadas.

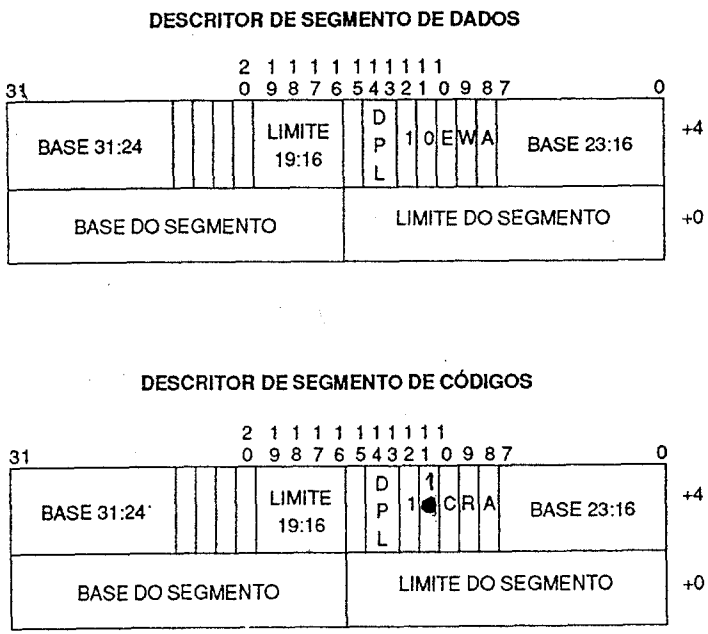
Cada referência à memória é controlada para certificar-se de que satisfaz o mecanismo de proteção. Todos os exames são feitos antes do início do ciclo de barramento. Qualquer violação impede o ciclo e gera uma exceção. Como este exame é realizado simultaneamente à tradução do endereço, não há comprometimento da performance. Ao todo existem cinco controles de proteção:

1. Controle de tipo.
2. Controle de limite.
3. Restrições ao domínio endereçável.
4. Restrições aos pontos de entrada das rotinas.
5. Restrições ao conjunto de instruções.

As violações à proteção geram uma exceção. Neste capítulo, detalhamos estas violações que conduzem à exceção.

Descritores de Segmento e Proteção

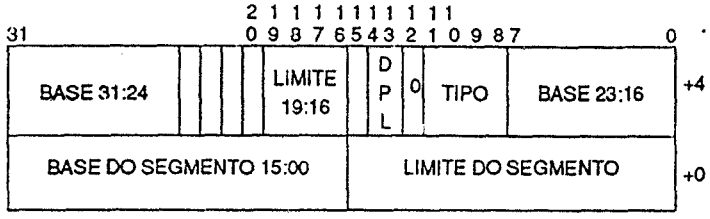
A figura 3.44, mostra os campos de um descritor de segmento que são usados pelo mecanismo de proteção. Os bits individuais são referenciados pelo nome de suas funções.



A	Acessado
C	Conforme (relativo ao nível de privilégio)
DPL	(nível de privilégio do descritor)
E	(expansão para baixo)
R	(habilitado à leitura)
LIMITE	(limite do segmento)
W	(habilitado à escrita)

Figura 3.44 - Campos dos descritores usados para proteção (parte).

DESCRITOR DE SEGMENTO DE SISTEMA



DPL (NÍVEL DE PRIVILÉGIO DO DESCRITOR)
LIMITE (LIMITE DO SEGMENTO)

Figura 3.44 - Campos dos descritores usados para proteção

Os parâmetros de proteção são colocados no descritor quando da sua criação. Via de regra, os programas aplicativos não precisam se preocupar com estes parâmetros.

Quando um programa armazena um seletor de segmento em um registrador, simultaneamente o 80486 armazena seu endereço base e a informação referente à proteção. Cada parte invisível de um registrador de segmento tem espaço para as informações de base, limite, tipo e nível de privilégio. Enquanto estas informações estiverem no registrador, o controle sobre as mesmas será exercido sem comprometimento da performance.

CONTROLE DE TIPO

Além dos descritores de códigos e dados de programas aplicativos, o 80486 possui descritores para segmentos de sistema e gates. Que são estruturas utilizadas no gerenciamento de tarefas, exceções e interrupções. A tabela 3.11 lista todos os tipos definidos para segmentos de sistema e gates. Observe que nem todos os descritores definem segmentos; descritores de gates armazenam ponteiros para pontos de entrada em processos.

Os campos de tipo dos descritores de dados e de códigos incluem bits que em primeiro lugar definem o propósito do segmento (ver figura 3.44):

- ❑ O bit de "habilitado à escrita" no descritor de segmento de dados controla se um programa pode ou não escrever no segmento.
- ❑ O bit de "habilitado à leitura" em descritor de segmento de códigos especifica se um programa pode acessá-lo para leitura, que poderá ser executada de duas formas:

1. A partir do registrador CS informado pelo prefixo da instrução.
2. Carregando-se seletor para o descritor em um dos registradores de dados (DS, ES, FS ou GS).

O controle de tipo permite a detecção de tentativas de acesso a segmentos por meios não planejados pelo programador. O processador examina a informação de tipo em duas ocasiões possíveis:

1. Quando o seletor para um descritor é carregado num registrador de segmento. Certos registradores de segmento somente comportam tipos específicos de descritores; por exemplo:

- ❑ O registrador CS somente pode ser carregado com seletores de segmentos executáveis.
- ❑ Os seletores de segmentos de códigos não autorizados à leitura, não podem ser carregados em registradores de segmentos de dados.
- ❑ Somente seletores para segmentos habilitados à escrita podem ser carregados no registrador de segmento de stack (SS).

2. Determinados segmentos somente podem ser utilizados por instruções a partir de vias predefinidas, por exemplo:

- ❑ Nenhuma instrução pode escrever em um segmento executável.
- ❑ Nenhuma instrução pode escrever num segmento de dados não habilitado à escrita.
- ❑ Nenhuma instrução pode ler um segmento de códigos não habilitado à leitura.

Tipo	Descrição
0	Reservado
1	TSS 80286 disponível
2	LDT - Tabela de descritores locais
3	TSS 80286 ocupada
4	Gate de chamada
5	Gate de tarefa
6	Gate de interrupção 80286
7	Gate de bloqueio 80286
8	Reservado
9	TSS 80486 disponível
10	Reservado
11	TSS 80486 ocupada
12	Gate de chamada 80486
13	Reservado
14	Gate de interrupção 80486
15	Gate de tarefa 80486

Tabela 3.11 - Tipos de segmentos de sistema e de gate

CONTROLE DE LIMITE

O campo de limite em um descritor de segmento, evita o acesso a endereços que não pertençam a ele. O valor deste campo depende do bit de granularidade (Bit G). Para segmentos de dados, o limite também depende do bit de direção de expansão do segmento (Bit E). O bit E é a designação de um dos bits do campo de tipo para descritores de segmentos de dados.

Quando o bit G é igual a zero, o limite corresponde ao valor expresso pelos 20 bits do campo limite. Neste caso, o valor estará entre 0 e 0FFFFFH (2^{20} - 1 ou 1 Megabyte). Quando o bit G é igual a 1, o processador multiplica o valor do campo de limite por um fator de 2^{12}. E neste caso, o limite varia entre 0FFFH (2^{12} - 1 ou 4 Kbytes) até 0FFFFFFFF (2^{32} - 1 ou 4 Gigabytes). Observe que nesta situação, os 12 bits menos significativos do endereço não são verificados em relação ao limite; pois mesmo quando o limite é igual a zero, o offset poderá assumir valores entre 0 e 4095.

Para todos os tipos de segmento, com exceção dos que possuem expansão para baixo (segmentos de stacks), o valor do limite é igual ao tamanho

do segmento menos 1. Uma exceção de proteção- geral ocorre em um destes casos:

- Tentativa de acesso a byte em memória em endereço maior do que o limite.
- Tentativa de acesso a word em memória em endereço maior do que o limite - 1.
- Tentativa de acesso a doubleword em endereço maior do que o limite - 3 .

No caso de segmentos de dados com expansão para baixo, o limite possui a mesma função mas é interpretado de maneira diferente. A faixa de offsets válidos está entre (limit + 1) e (2^{32} - 1). Um segmento com expansão para baixo tem o tamanho máximo quando o limite é igual a zero.

Além do controle do limite do segmento, existe a verificação do limite para as tabelas de descritores. Os registradores GDTR e IDTR possuem campos de 16 bits para seus respectivos limites. O processador se utiliza desses valores para evitar o acesso a descritores que não existem por parte de softwares ou por conseqüência de erro de programação. O limite de uma tabela informa o último byte válido da mesma. Como cada descritor possui oito bytes, uma tabela com N descritores terá limite igual a 8N-1.

NÍVEIS DE PRIVILÉGIO

O mecanismo de proteção reconhece 4 níveis de priviléigo, numerados de 0 a 3. Onde o nível de maior privilégio é o zero e o de menor o 3. Se todos os outros controles de proteção são satisfeitos, um programa que tentar acessar um segmento com nível de privilégio menor que o do segmento, produz uma exceção de proteção geral.

Embora não exista bit de controle que permita desativar o mecanismo de proteção, podemos simular este efeito, atribuindo-se a todos o nível de privilégio zero. Observe que o bit PE (Protect enable) do registrador CR0 não é um bit de habilitação do mecanismo de proteção, ele é utilizado para habilitar o modo protegido de operação do processador, no qual toda a arquitetura de 32 bits está disponível. No modo real, o 80486 opera como um 8086, evidentemente mais rápido.

Os níveis de privilégio podem ser usados para tornar um sistema muito mais confiável. Dando-se ao sistema operacional o nível mais alto de privilégio, ele fica protegido de danos causados por erros de programação (bugs) em outros softwares. Se um programa é interrompido devido a esses erros, o sistema operacional tem a oportunidade de apresentar mensagens de erros, ou mesmo tentar recuperar a rotina que apresentou problemas.

Podemos também proteger o sistema operacional de erros em programas de acesso a periféricos (device drivers) e, indiretamente, de erros nos próprios periféricos. Deste modo temos a possibilidade de reportar erros de acesso a estes dispositivos. Em um nível mais baixo, podemos proteger estas rotinas de erros em programas aplicativos que poderiam danificar informações em elementos críticos como o disco rígido. Os níveis de privilégio, portanto, podem ser definidos para o sistema operacional, como o mais alto; para os device drivers, o intermediário, e finalmente para os programas aplicativos, o mais baixo.

A figura a seguir, mostra como estes níveis devem ser interpretados a partir de anéis de proteção. O centro é para segmentos que contêm os softwares mais críticos, em geral o kernel do sistema operacional. Os outros anéis são para os programas menos críticos.

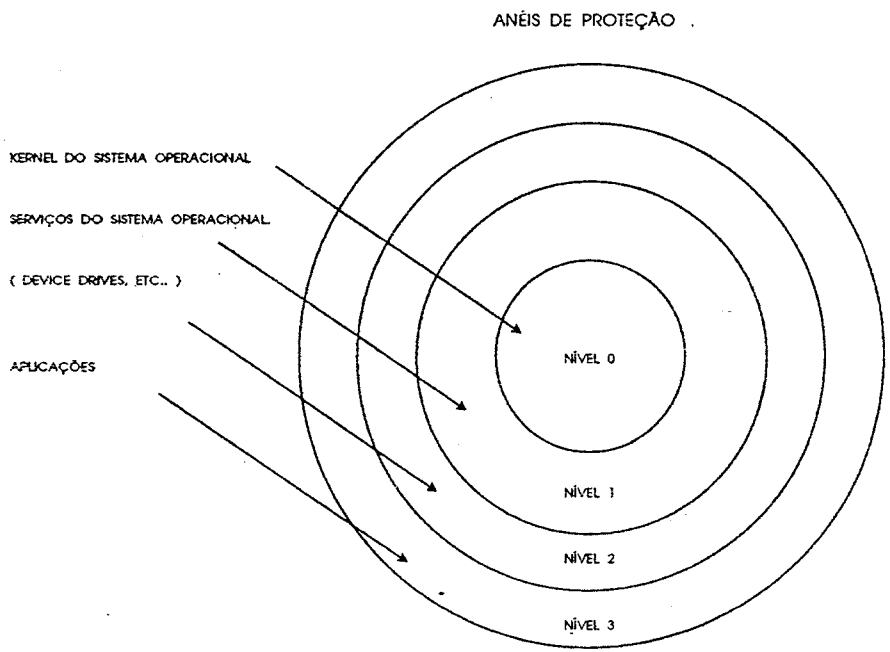


Figura 3.45 - Anéis de proteção

As estruturas de dados a seguir, são as que armazenam os níveis de privilégio:

- Os dois bits menos significativos do registrador CS contêm o nível de privilégio corrente CPL (Current Privilege Level). Corresponde ao nível de privilégio do programa que está sendo rodado. Os dois bits menore do SS contêm uma cópia do CPL. Normalmente, o CPL é igual ao nível de privilégio do segmento de códigos onde se encontra a instrução que está sendo buscada. O CPL muda quando o controle é transferido para um segmento de códigos com diferente nível de privilégio.
- Os descritores de segmentos possuem um campo chamado Nível de privilégio do Descritor DPL (Descriptor Privilege Level). O DPL corresponde ao nível do segmento.
- Os seletores de segmento possuem um campo chamado de nível de privilégio requisitado RPL (Requested Privilege Level). O RPL representa o nível de privilégio da rotina que criou o seletor. Se o RPL é menos privilegiado do que o CPL, ele é sobreposto ao CPL. Quando um programa mais privilegiado recebe um seletor de segmento a partir de um programa de menor privilégio, o RPL faz com que o acesso à memória seja realizado em um nível menos privilegiado.

O controle do nível de privilégio é feito sempre que o seletor de um descritor é armazenado em registrador de segmento. As verificações realizadas em transferências entre segmentos, são diferentes daquelas feitas para os acessos a dados, por este motivo, os itens seguintes discriminam em separado estes controles.

Restringindo o Acesso a Dados

O acesso a um operando em memória é feito armazenando-se um seletor de segmento em um dos registradores de segmentos de dados (DS, ES, FS, GS ou SS). O processador controla os níveis de privilégios dos segmentos. A verificação é feita no instante em que o seletor é carregado. A figura 3.46, mostra três níveis de privilégio entrando neste sistema de verificação.

Os três níveis controlados são:

1. O CPL do programa (nível de privilégio corrente). Que se encontra nos dois bits menos significativos do registrador CS.
2. O DPL, nível de privilégio do descritor do segmento onde se encontra o operando.

3. O RPL, nível de privilégio solicitado do seletor utilizado para especificar o segmento que possui o operando. Encontra-se nos dois bits menos significativos do registrador de segmento usado para acesso ao operando (DS, ES, FS, GS ou SS). Caso o operando encontre-se no stack, o RPL é o mesmo do CPL.

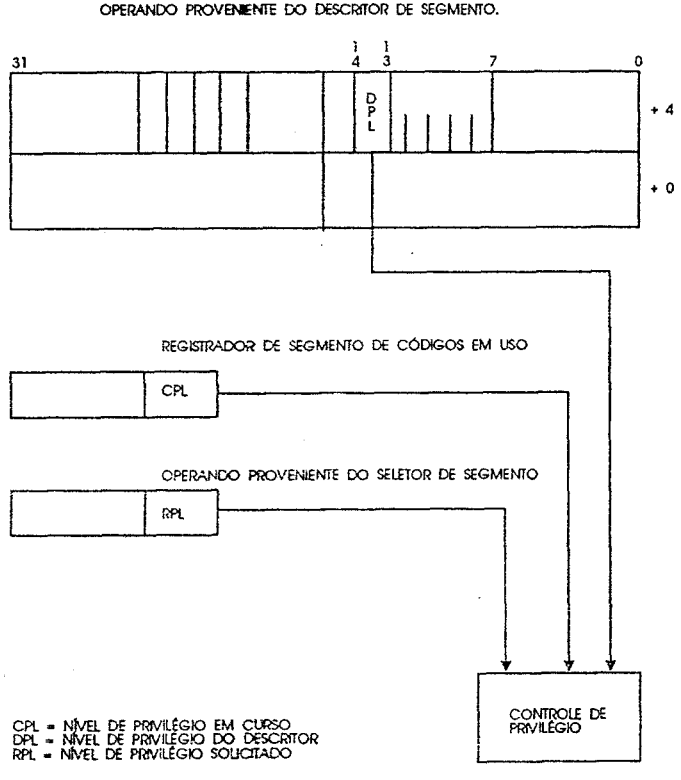


Figura 3.46 - Controle de privilégio para o acesso a dados.

As instruções podem carregar um registrador de segmento somente se o DPL do segmento possuir o mesmo ou menor privilégio do que o CPL e o RPL doseletor.

O domínio de endereçamento de uma tarefa varia conforme o seu CPL. Quando o CPL é nível zero, os dados de todos os níveis de privilégio estão disponíveis, quando o CPL é 1, somente os dados com níveis entre 1 e 3 estão disponíveis; quando é 3, somente os dados de nível 3 são acessáveis.

ACESSANDO DADOS EM SEGMENTOS DE CÓDIGOS

Em algumas situações, somos forçados a armazenar dados em segmentos de códigos (ex.: programas em ROM). Normalmente fazemos isto para valores constantes. Um segmento definido como código não pode ser escrito, a menos que um segmento de dados seja mapeado para o mesmo espaço. Temos os seguintes métodos de acesso a dados em segmentos de códigos:

1. Carregar registrador de segmento de dados com um seletor que aponte para segmento de características não-conforme, habilitado para leitura e executável.
2. Carregar registrador de segmento de dados com um seletor que aponte para segmento de características conforme, habilitado para leitura e executável.
3. Usar prefixo de segmento de código para a leitura de segmento executável, habilitado à leitura e cujo seletor já se encontra no registrador CS.

As mesmas regras de acesso a dados se aplicam no primeiro caso. No segundo, o acesso é sempre válido pois o nível de privilégio de um segmento de códigos definido como "conforme" é efetivamente o mesmo do CPL, independente do DPL. E finalmente, o caso 3 é sempre válido pois o DPL do segmento de códigos selecionado pelo registrador CS é o CPL.

Restrições às Transferências de Controle

As formas mais comuns de transferências de controle são a partir das instruções de JMP, CALL e RET dentro do segmento de códigos em uso e, portanto, sujeitas apenas à verificação de limite. O processador avalia se o acesso excede ou não o limite do segmento atual. Este limite encontra-se no próprio registrador CS (parte invisível), e portanto não interfere na performance do processamento.

As instruções CALL e JMP para outros segmentos (far) obrigam o processador a realizar o controle do nível de privilégio. Existem duas formas para uma instrução CALL ou JMP se referenciar a outro segmento:

1. O operando seleciona um descritor de outro segmento executável.
2. O operando seleciona o descritor de um "gate".

A figura 3.47, apresenta dois níveis de privilégios diferentes entrando no mecanismo de controle para uma transferência que não se utiliza de um gate:

1. O CPL, nível de privilégio corrente.
2. O DPL do descritor do segmento de códigos destino.

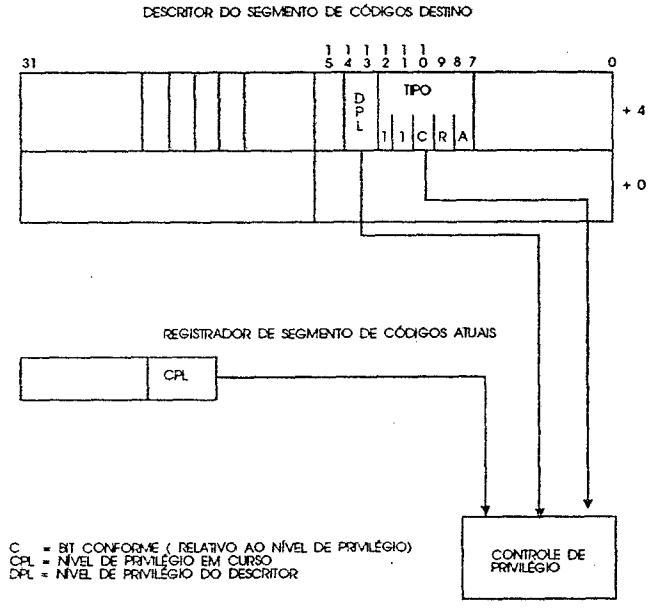


Figura 3.47 - Privilégio para transferências sem gate

Normalmente, o CPL é igual ao DPL do segmento que o processador está executando. Entretanto, o CPL pode ser menos privilegiado do que o DPL se o segmento de códigos atual é do tipo conforme (indicado pelo campo de tipo do seu descritor). Um segmento conforme roda no nível de privilégio da rotina que o chamou. O processador mantém um registro do CPL armazenado no registrador CS; este valor pode ser diferente do DPL no descritor do segmento de códigos em uso.

O processador somente permite um JMP ou CALL diretamente para outro segmento se todas as regras abaixo forem satisfeitas:

- O DPL do segmento é igual ao CPL corrente.
- O segmento de códigos é do tipo conforme e o DPL é mais privilegiado do que o CPL.

Segmentos tipo "conforme" são usados para programas como, por exemplo, bibliotecas matemáticas que dão suporte a aplicações e não necessitam acessar instalações de sistema. Quando o controle é transferido para um segmento "conforme", o CPL não muda, mesmo que o seletor utilizado para endereçar o segmento tenha um RPL diferente. Esta é a única condição na qual o CPL pode ser diferente do DPL do segmento de códigos em uso.

A maior parte dos segmentos são do tipo não-conforme. Para estes segmentos o controle pode ser transferido sem um gate, somente para segmentos de códigos com o mesmo nível de privilégio. Algumas vezes é necessário, entretanto, transferir o controle para níveis mais privilegiados. Isso é possível através de instruções de CALL via descritores conhecidos por "call-gates". Uma instrução JMP nunca pode transferir o controle para um segmento não-conforme cujo DPL não é igual ao CPL.

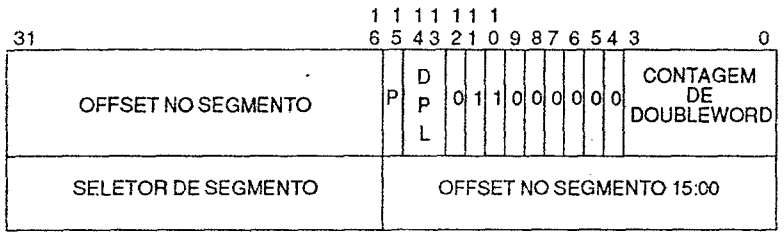
Descritores Gate

O controle de transferências entre segmentos de diferentes níveis de privilégio é feito a partir de descritores chamados de gate. Existem quatro tipos de descritores gate:

- Gates de chamada (Call).
- Gates de bloqueio (Traps).
- Gates de interrupções.
- Gates de tarefas.

A figura 3.48, ilustra o formato de um gate de chamada (CALL).

GATE DE 32 BITS PARA CHAMADA



DPL NÍVEL DE PRIVILÉGIO DO DESCRITOR.
P PRESENÇA DO SEGMENTO

Figura 3.48 - Gate de chamada

Um gate de chamada possui duas funções principais:

1. Definir a ponte de entrada de um processo.
2. Especificar o nível de privilégio requerido para se entrar em um processo.

Os gates de chamadas são usados por instruções de JMP e de CALL do mesmo modo que os descritores de segmentos de códigos. Quando o hardware reconhece que o seletor de segmento para o destino referencia-se a um descritor de gate, a operação da instrução é determinada pelo conteúdo do gate de chamada. Este pode encontrar-se tanto na GDT quanto na LDT, e não pode estar na IDT.

Os campos de seletor e offset do gate formam um ponteiro para o ponto de entrada de um processo. Um gate de chamada garante que todas as transferências para outros segmentos irão para um ponto de entrada válido ao invés de cair no meio de uma rotina ou, o que seria muito pior, no meio de uma instrução. O operando da instrução de transferência de controle não é o seletor de segmento e o offset dentro do segmento para o ponto de entrada da rotina. Ao invés disso, o seletor de segmento aponta para um descritor de gate, e o offset não é utilizado. A figura 3.49, mostra esta forma de endereçamento.

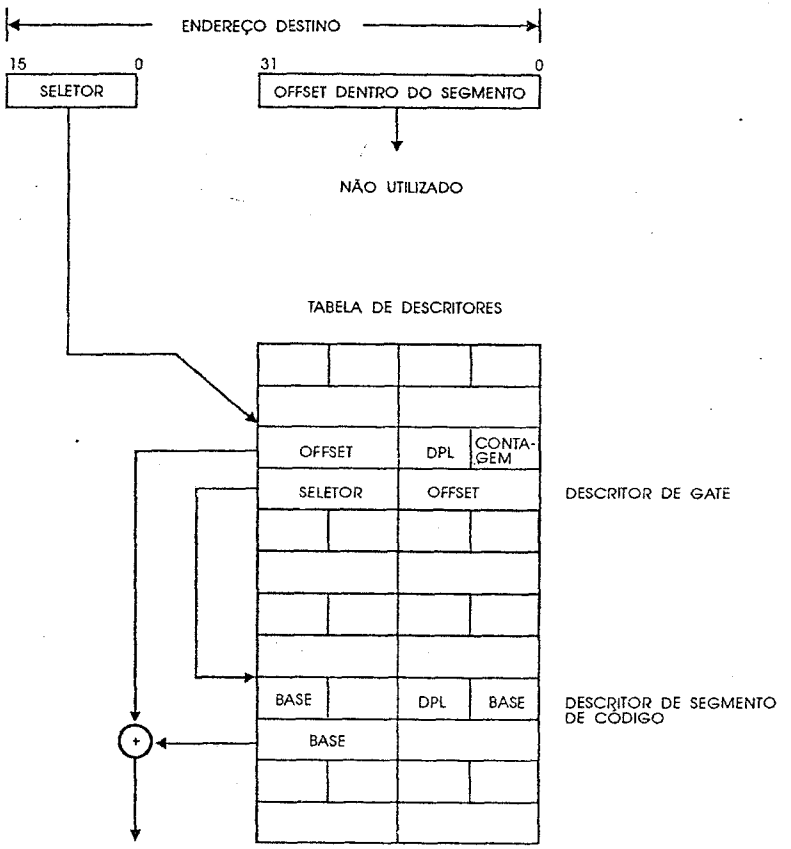


Figura 3.49 - Mecanismo do gate de chamada.

A figura 3.50 mostra os quatro níveis de privilégios diferentes utilizados no controle de uma transferência via gate de chamada.

Eles são os seguintes:

1. O CPL, nível de privilégio corrente.
2. O RPL, nível de privilégio solicitado do seletor de segmento usado para especificar o gate de chamada.
3. O DPL, nível do descritor do gate de chamada.
4. O DPL do descritor do segmento de códigos destino.

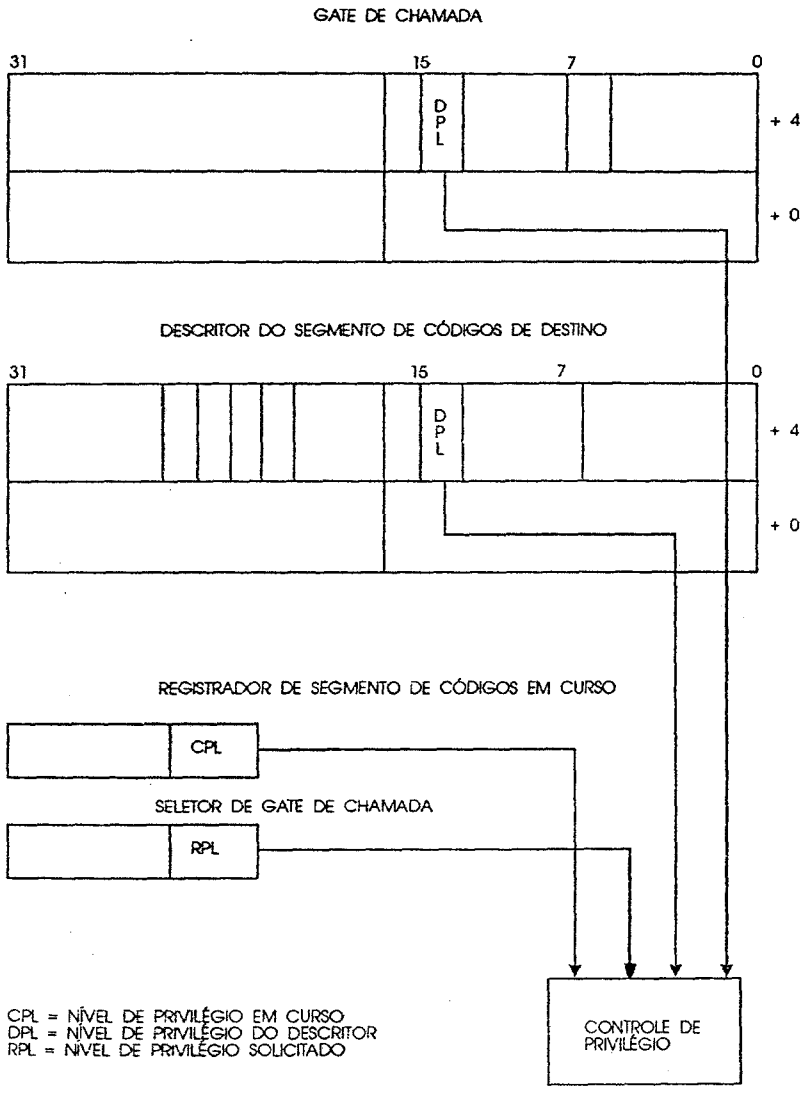


Figura 3.50 - Verificação de privilégio com gate de chamada

O campo DPL do descritor do gate determina a partir de qual nível de privilégio o gate pode ser utilizado. Um segmento de códigos pode ter várias rotinas de diferentes níveis de privilégio. Por exemplo, um sistema operacional pode ter rotinas de acesso ao disco que estão disponíveis a outro softwares e pode ter rotina de acesso exclusivo do próprio sistema operacional, como as rotinas de inicialização de drivers para dispositivos.

Os gates podem ser utilizados para transferências de controle no mesmo nível ou para níveis de maior privilégio. Somente instruções CALL podem ser utilizadas em transferências a níveis menos privilegiados. A instrução de JMP somente pode ser utilizada quando o nível é o mesmo ou para segmentos de códigos cujo tipo é "conforme". Mesmo nesse caso, não poderá ser de um nível menos privilegiado.

Para uma instrução JMP a um segmento não-conforme, as regras abaixo devem ser satisfeitas ao mesmo tempo:

MAX (CPL,RPL) <= DPL do gate.

DPL do segmento de códigos destino = CPL.

Para uma instrução CALL (ou para um JMP a segmento "conforme"), as regras são as seguintes:

MAX (CPL,RPL) <= DPL do gate.

DPL do segmento de códigos destino <= CPL.

Caso contrário, uma exceção de proteção-geral é produzida.

MUDANÇA DE STACK

A transferência via CALL para um nível mais privilegiado faz o seguinte:

1. Altera o CPL.

2. Transfere o controle da execução.

3. Altera o stack.

Todos os níveis de proteção dos anéis mais internos (níveis 0, 1 e 2), possuem seu próprio stack para recebimento de chamadas de níveis menos privilegiados. Caso o stack fosse providenciado pela rotina chamadora, e não dispussesse de tamanho suficiente para uso da rotina chamada, esta última poderia interromper-se involuntariamente por insuficiência de stack. Ao invés disso, programas com menor nível de privilégio, são impedidos de causar danos a outros de maior privilégio a partir da criação de novo stack quando uma chamada é feita a um nível mais privilegiado. O novo stack é criado, copiam-se parâmetros, salvam-se conteúdos de registradores e a execução prossegue normalmente.

Quando o processo termina, os conteúdos dos registradores preservados são restaurados no stack original.

O processador encontra o espaço necessário para a criação do novo stack a partir de segmentos de estado de tarefa (TSS - task state segment), conforme a figura 3.51. Cada tarefa tem o seu próprio TSS. O TSS contém ponteiros iniciais de stacks para os anéis de proteção internos. O sistema operacional é responsável pela criação de cada TSS e a inicialização de seus ponteiros de stack. Um ponteiro inicial consiste de um seletor de segmento e um valor inicial para o registrador ESP (offset inicial dentro do segmento). Os ponteiros de stack iniciais são valores estritamente de leitura. O processador não os altera enquanto a tarefa é executada. Estes ponteiros são utilizados somente para se criar um novo stack quando chamadas são feitas para níveis mais privilegiados. Estes stacks desaparecem quando a rotina chamada retorna. Na próxima ocasião em que a rotina for chamada, um novo stack será criado, utilizando-se o ponteiro inicial do stack.

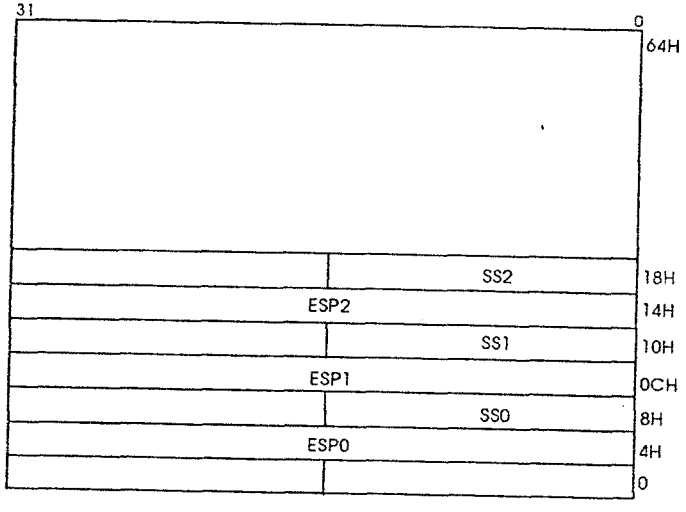


Figura 3.51 - Ponteiro de stack inicial em uma TSS.

Quando um gate de chamada é utilizado para a alteração de níveis de privilégio, um novo stack é criado pelo carregamento de um endereço a partir do TSS. O processador se utiliza do DPL do código de destino (o novo CPL) para selecionar o ponteiro inicial do stack dos níveis 0, 1 ou 2.

O DPL do novo segmento de stack deve ser igual ao novo CPL; caso isto não ocorra, será gerada uma exceção de "falha de stack". É responsabilidade do sistema operacional criar stacks e descritores de segmentos de stacks para todos os níveis de privilégios em que ele é usado. O descritor de segmento de stack deve especificar o seu tipo como de leitura e escrita. E deve ter um limite suficiente para comportar o conteúdo dos registradores SS e ESP, o endereço de retorno, os parâmetros e as variáveis temporárias requeridas pela rotina chamada.

Como nas chamadas, os parâmetros são copiados no novo stack, e podem ser acessados pela rotina solicitada a partir dos mesmos endereços relativos que seriam usados caso a mudança de stack não tivesse ocorrido. O campo do contador no gate de chamada informa ao processador quantas doublewords (máximo 31) copiar do stack da rotina chamadora para o da rotina chamada. Se o contador for zero, nenhum parâmetro será copiado.

Caso mais de 31 doublewords de parâmetros devam ser passadas para a rotina chamada, um desses parâmetros pode ser utilizado como ponteiro para uma estrutura de dados, ou então o conteúdo salvo dos registradores SS e ESP podem ser usados para acessar parâmetros no espaço do antigo stack.

O processador executa os seguintes procedimentos relativos ao stack quando de uma chamada entre dois níveis de privilégios diferentes:

VERIFICAÇÃO

1. É feita uma no tamanho do stack da rotina chamada para certificar-se de que possui tamanho suficiente para comportar os parâmetros e preservar o conteúdo dos registradores; caso contrário, é gerada uma exceção de "falha de stack".
2. O antigo conteúdo dos registradores SS e ESP são armazenados no stack da rotina chamada em duas doublewords (os 16 bits do registrador SS são estendidos a 32 bits; com a word superior zerada não a utilize pois trata-se de informação considerada reservada pela Intel).
3. Os parâmetros são copiados do stack da rotina chamadora para o stack da rotina chamada.
4. Um ponteiro para a instrução seguinte à CALL (antigo conteúdo dos registradores CS e EIP), é armazenado no novo stack. Os conteúdos dos registradores SS e ESP após a chamada apontam para este ponteiro no stack.

A figura 3.52 ilustra a estrutura do stack antes, durante e depois de uma chamada bem sucedida entre níveis.

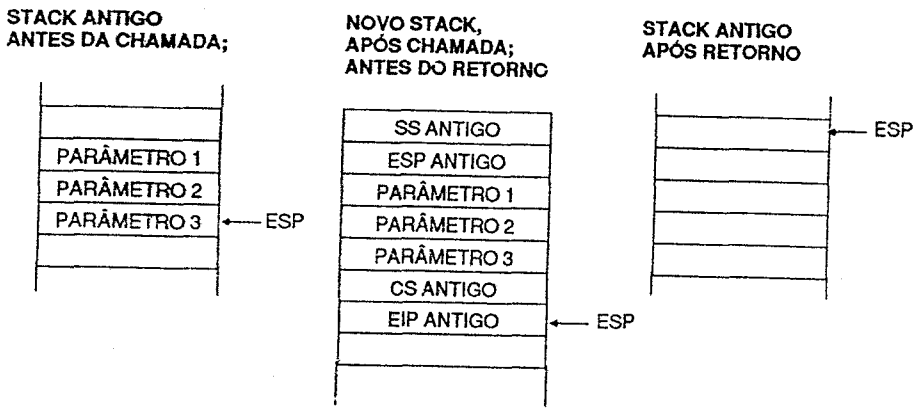


Figura 52 - Estrutura do stack em uma chamada entre níveis

O TSS não possui um ponteiro de stack para o nível de privilégio 3 e, isto ocorre porque um processo com este nível de privilégio não pode ser chamado por uma rotina menos privilegiada. O stack para ele é preservado pelos conteúdos dos registradores SS e ESP que são armazenados no stack do nível chamado a partir do nível 3.

Uma chamada a partir de um gate não controla os valores das words copiadas para o novo stack. A rotina chamada deve verificar cada parâmetro para avaliar sua validade.

RETORNANDO DE UM PROCESSO

As formas "próximas" (nears) da instrução RET transferem o controle somente dentro do segmento de códigos em uso, portanto estão sujeitas apenas aos controles de limites. O offset para a instrução seguinte à CALL é recuperado do stack e armazenado no registrador EIP. O processador então verifica se este offset não excede o limite do segmento.

A forma "distante" (far) da instrução RET, recupera o endereço armazenado no stack pela instrução CALL mais recente. Sob condições normais, o ponteiro de retorno é válido porque foi gerado por uma instrução CALL ou INT. Todavia, o processador realiza a verificação de privilégio por causa da possibilidade da rotina em curso ter alterado o ponteiro ou falhado na correta manutenção do stack. O RPL do descritor de segmento de código recuperado do stack pela instrução de retorno deve ter o nível de privilégio da rotina chamadora.

Um retorno para outro segmento pode alterar os níveis de privilégios, mas somente para níveis menos privilegiados. Quando uma instrução RET encontra um RPL menos privilegiado do que o CPL, um retorno através de níveis de privilégios ocorre, o que demanda os seguintes passos:

1. Os controles listados na tabela a seguir, são feitos e os registradores CS, EIP, SS e SP são armazenados com seus antigos valores preservados anteriormente no stack.

Tipo de controle	Tipo de exceção	Código de erro
Topo do stack deve estar dentro dos limites do segmento de stack	Stack	0
Topo do stack + 7 deve estar dentro dos limites do segmento de stack	Stack	0
RPL do segmento onde se encontra o código de retorno deve ser maior do que o CPL	Proteção	CS de retorno
Seletor do segmento do código de retorno não deve ser nulo	Proteção	CS de retorno
Descritor do segmento do código de retorno deve estar dentro dos limites da tabela de descritores	Proteção	CS de retorno
Descritor do segmento de retorno deve ser segmento de código	Proteção	CS de retorno
Segmento de código de retorno está presente	Segmento ausente	CS de retorno
DPL do segmento tipo "não conforme" de código de retorno deve ser igual ao RPL do seletor de segmento de código de retorno, ou o DPL do segmento tipo "conforme" de código de retorno deve ser menor do que ou igual ao RPL do seletor do segmento de código de retorno	Proteção	CS de retorno
ESP+N+15 deve estar dentro dos limites do segmento de stack	Falha de Stack	CS de retorno
Seletor de segmento na posição ESP+N+12 não deve ser nulo	Proteção	CS de retorno
Descritor de segmento na posição ESP+N+12 deve estar dentro dos limites da tabela do descritor	Proteção	CS de retorno
Descritor do segmento de stack deve ser de leitura e escrita	Proteção	CS de retorno
Segmento de stack deve estar presente	Falha de Stack	CS de retorno
DPL do segmento de stack antigo deve ser igual ao RPL do segmento de código antigo	Proteção	CS de retorno
Seletor do segmento de stack antigo deve ter RPL igual ao DPL do segmento de stack antigo	Proteção	CS de retorno

Onde N é o valor do operando imediato fornecido pela instrução RET

Tabela 3.12 - Controle de retorno interníveis.

2. Os antigos conteúdos dos registradore SS e ESP (provenientes do stack), são ajustados pelo número de bytes indicados na instrução RET. O ESP resultante não é comparado com o limite do segmento de stack. Se o valor do ESP é superior ao limite, este fato não é reconhecido até a próxima operação com o stack.
3. São verificados os conteúdos dos registradores DS, ES, FS e GS. Caso algum deles se referencie a um segmento cujo DPL é menor do que o novo CPL (excetuando-se os segmentos de códigos tipo "conforme"), o registrador é carregado com um seletor nulo. A instrução RET por si própria não sinaliza uma exceção neste caso, entretanto, qualquer referência subsequente à memória, usando um registrador de segmento cujo seletor é nulo, produzirá uma exceção de proteção geral. Isto evita códigos menos privilegiados acessarem segmentos mais privilegiados, usando seletores deixados em registradores de segmento por rotinas mais privilegiadas.

Instruções Reservadas Para o Sistema Operacional

Instruções que podem afetar o mecanismo de proteção ou interferir na performance do sistema podem ser executadas somente por processos confiáveis. O 80486 possui duas classes destas instruções:

1. Instruções privilegiadas. Usadas no controle do sistema.
2. Instruções sensitivas. Usadas em operações de entrada e saída ou com referência a entradas e saídas.

INSTRUÇÕES PRIVILEGIADAS

As instruções que afetam instalações protegidas podem ser executadas somente quando o nível de privilégio é 0 (o mais privilegiado). Caso um outro nível tente executá-las, será gerada uma exceção de proteção geral. Estas instruções incluem:

CLTS - Apagar indicador de mudança de tarefa (Clear Task-Switched Flag).

HLT - Parar o processador (Halt processor).

LGDT - Carregar o registrador da tabela de descritores globais (Load GDT register).

LIDT - Carregar o registrador da tabela de descritores de interrupções (Load IDT register).

LLDT - Carregar o registrador da tabela de descritores locais (Load LDT Register)

LMSW - Carregar a word de status do processador (Load Machine Status Word).

LTR - Carregar o registrador de tarefa (Load Task Register).

MOV to/from CR0 - Mover de/para o registrador de controle (Move to Control Register 0).

MOV to/from DRn - Mover de/para registrador de depuração (Move to debug Register n).

MOV to/from TRn - Mover de/para registrador de teste "n" (Move to Test Register n)

INSTRUÇÕES SENSITIVAS

Instruções que se relacionam com dispositivos de entrada e saída, precisam ser protegidas, mas ao mesmo tempo, precisam ser utilizadas por níveis de privilégios diferentes de zero.

Instruções Para Validação de Ponteiros

A validação de ponteiros é necessária para que se mantenha a isolação entre níveis. E consiste nos seguintes passos:

1. Verificar se o fornecedor do ponteiro está autorizado a acessar o segmento.
2. Controlar se o tipo do segmento é compatível com seu uso.
3. Verificar se o offset do ponteiro excede o limite do segmento.

Embora o 80486 automaticamente realize os controles 2 e 3 durante a execução da instrução, o software deve auxiliá-lo no primeiro controle. A instrução ARPL existe com este propósito. O programa também pode utilizar os passos 2 e 3 para violações em potencial, ao invés de aguardar pela geração de exceções. As instruções LAR, LSL, VERR, e VERW existem com este objetivo.

Um controle adicional de alinhamento pode ser aplicado no modo usuário. Quando ambos os bits: AM no CR0 e o flag AC são iguais a 1, uma referência à memória desalinhada provoca exceção. Isto pode ser útil para programas que usam os dois bits mais baixos dos ponteiros para identificar o tipo de estrutura de dados.

LAR - Carregar direitos de acesso (Load Access Rights) utilizada para verificar se um ponteiro referencia-se a segmento cujo tipo e nível de privilégio são compatíveis. A instrução LAR tem um operando: um descritor de segmento cujo direito de acesso é para ser verificado. O descritor do segmento deve estar

habilitado para leitura a um nível de privilégio menor do que o CPL e o RPL do seletor. Se o descritor é habilitado para leitura, a instrução LAR mascara a segunda doubleword do descritor com o valor 00FxFF00H, e armazena o resultado no registrador de 32 bits de destino e iguala o indicador ZF a 1. O x indica que os quatro bits correspondentes do valor armazenado são indefinidos. Uma vez carregado, o direito de acesso pode ser testado. Todos os tipos válidos de descritores podem ser testados pela instrução LAR. Se o CPL ou o RPL é maior do que o DPL, ou se o seletor do segmento exceder o limite da tabela de descritores, nenhum direito de acesso é retornado, e o flag ZF é zerado. Um segmento de códigos do tipo "conforme" pode ser acessado a partir de qualquer nível de privilégio.

LSL - Carregar limite do segmento (load Segment Limit), permite ao software testar o limite de um descritor de segmento. Se o descritor referenciado pelo seletor de segmento (em memória ou registrador) está habilitado para leitura no CPL, a instrução LSL carrega o registrador de 32 bits especificado com o limite no formato de 32 bits com granularidade de byte, calculado a partir do campo de limite e do bit de granularidade do descritor. Isto somente pode ser feito para descritores que descrevem segmentos (de dados, códigos, tarefa, estado e tabelas de descritores locais); descritores de gates são inacessíveis. A tabela 3.13 lista em detalhes quais tipos de descritores são válidos ou não. A insterpretação do limite é função do tipo de segmento. Por exemplo, segmentos de dados com expansão para baixo (segmentos de stack) tratam o limite de forma diferente dos outros tipos de segmentos. Para as instruções LAR e LSL, o ZF será igual a um se executado com sucesso, e zero se ocorreu algum problema.

VALIDAÇÃO DE DESCRITOR

O processador 80486 tem duas instruções, VERR e VERW, que determinam se um seletor de segmento aponta para um segmento que pode ser lido ou escrito usando o CPL. Nenhuma destas instruções ocasionam falhas de proteção se o segmento não pode ser acessado.

Código do tipo	Tipo do descritor	Válido ?
0	Reservado	Não
1	Reservado	Não
2	LDT	Sim
3	Reservado	Não
4	Reservado	Não
5	Gate de tarefa	Não
6	Reservado	Não

Tabela 3.13 - Tipos de descritores para a instrução LSL. (parte)

Código do tipo	Tipo do descritor	Válido ?
8	Reservado	Não
9	TSS disponível do 80486	Sim
A	Reservado	Não
B	TSS ocupada do 80486	Sim
C	Gate de chamada do 80486	Não
D	Reservado	Não
E	Gate de interrupção do 80486	Não
F	Gate de bloqueio	Não

Tabela 3.13 - Tipos válidos de descritores para a instrução LSL.

VERR - Verificação para leitura (Verify for Reading), verifica um segmento para leitura e iguala a 1o flag ZF se o mesmo estiver habilitado para leitura usando o CPL. A instrução VERR verifica o seguinte:

- Se o Seletor de segmento aponta para um descritor dentro dos limites da GDT ou da LDT.
- Se o seletor de segmento indexa um descritor de segmento de códigos ou de dados.
- Se o segmento está habilitado à leitura e tem um nível de privilégio compatível.

O controle do privilégio para segmentos de dados ou de códigos "não-conforme" verifica que o DPL deve ser menos privilegiado do que o CPL ou o RPL do descritor. Para segmentos do tipo "Conforme", o nível de privilégio não é verificado.

VERW - Verificação para a escrita (Verify for Writing), fornece a mesma capacidade que a instrução VERR mas para a verificação de habilitação de escrita. A instrução VERW iguala a 1 o ZF caso o segmento possa ser escrito. A instrução verifica se o descritor está dentro dos limites, é um descritor de segmento, está habilitado para escrita e tem 1 DPL menos privilegiado do que ou o CPL ou o RPL do seletor. Segmentos de códigos nunca podem ser escritos, independente de serem "conforme" ou não.

INTEGRIDADE DE PONTEIRO E RPL

O nível de privilégio solicitado (RPL) pode evitar o uso acidental de ponteiros que danifiquem códigos mais privilegiados a partir de códigos menos privilegiados.

O processador 80486 automaticamente verifica qualquer seletor carregado em um registrador de segmento para certificar-se de que seu RPL permite o acesso.

Para aproveitar a verificação do RPL feita pelo processador, a rotina chamada precisa apenas verificar se todos os seletores que lhe foram passados têm RPL com o mesmo, ou com menor nível de privilégio do que o CPL da rotina chamadora. Isto garante que o seletor não é mais privilegiado do que sua origem. Se um seletor é utilizado para o acesso a segmento cuja fonte não estaria habilitada para acessá-lo diretamente, isto é, o RPL é menos privilegiado do que o DPL, uma exceção de proteção geral é produzida quando o seletor é carregado em um registrador de segmento.

ARPL - Alterar o nível de privilégio solicitado (Adjust Requested Privilege Level), adapta o campo de RPL de um seletor de segmento para ser menos privilegiado do que seu valor original, o mesmo acontecendo no caso de ele se encontrar em um registrador de segmento. Os campos de RPL encontram-se nos dois bits menos significativos do seletor de segmento e do registrador. Caso o ajuste altere o RPL do descritor, o flag ZF é igualado a 1, caso contrário é zerado.

Proteção do Nível da Página

A proteção se aplica tanto para segmentos quanto para páginas. Quando o modelo flat para segmentação de memória é utilizado, a proteção do nível da página evita a interferência de um programa em outro.

Cada referência à memória é controlada para certificar-se de que satisfaz o mecanismo de proteção. Todos os controles são feitos antes do ciclo de memória ser iniciado, qualquer violação evita o início do ciclo e resulta em uma exceção. Como a verificação é realizada em paralelo à tradução do endereço, não existe nenhuma perda de performance. Existem dois níveis de proteção a páginas:

1. Restrição ao domínio endereçável.
2. Controle de tipo.

ENTRADAS DAS TABELAS DE PÁGINAS ARMAZENAM PARÂMETROS DE PROTEÇÃO

A figura a seguir, destaca os campos da entrada da tabela da página que controlam o acesso às páginas. Os controles são aplicados tanto no primeiro quanto no segundo nível de tabelas.

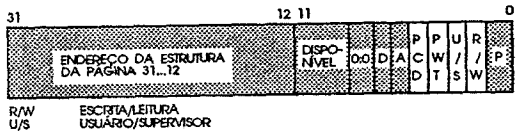


Figura 3.53 - Campos de uma Entrada na Tabela de Páginas

Restrição ao Domínio Endereçável

O privilégio é interpretado de maneira diferente para páginas e segmentos. Nos segmentos, temos quatro níveis de privilégios, que variam de 0 (mais privilegiado) até 3 (menos privilegiado). Com as páginas existem dois níveis:

1. Nível supervisor: (U/S = 0), para sistemas operacionais, outros softwares de sistema (device drivers), e dados de sistema protegidos (como as tabelas de páginas).
2. Nível usuário (U/S=1), para códigos de aplicações e dados.

Os níveis de privilégios utilizados para a segmentação encontram-se mapeados naqueles usados na paginação. Se o CPL é 0, 1 ou 2, o processador está rodando em nível supervisor. Se o CPL é 3, o processador está em nível usuário. Quando em modo supervisor, todas as páginas podem ser acessadas. Quando o processador está rodando em modo usuário, somentes as páginas neste nível são acessáveis.

Controle de Tipo

Somente dois tipos de páginas são reconhecidos pelo mecanismo de proteção:

1. Com acesso apenas para leitura (R/W = 0).
2. Com acesso para leitura e escrita (R/W = 1).

Quando o processador está rodando em nível supervisor com o bit WP no registrador CR0 em zero (seu estado seguinte à inicialização), todas as páginas são habilitadas para escrita e leitura (a proteção de escrita é ignorada). Quando em nível usuário, somente as páginas que pertencem a este nível e estão marcadas como habilitadas à leitura e escrita é que podem ser modificadas. Páginas em nível usuário podem sempre ser lidas. As páginas em nível supervisor não podem ser nem lidas nem escritas pelo nível usuário. Será produzida uma exceção de proteção geral sempre que uma tentativa de violação às regras de proteção for realizada.

Ao contrário do 386DX, o processador 80486 permite a páginas do modo usuário serem protegidas contra o acesso do modo supervisor. Igualando-se o bit WP do registrador CR0 a 1, habilita-se a sensibilidade do modo supervisor em relação às páginas do nível usuário protegidas contra escrita.

ASSOCIANDO A PROTEÇÃO DE AMBOS OS NÍVEIS DE TABELAS DE PÁGINAS

Para uma dada página, os atributos de proteção de sua entrada no diretório de páginas (primeiro nível de tabela), pode ser diferente daqueles da entrada do segundo nível de tabela. O 80486 controla a proteção da página examinando os parâmetros dos dois níveis: diretório de páginas e tabela de páginas. A tabela 3.14 mostra a proteção fornecida por possíveis associações de atributos de proteção quando o bit WP é igual a zero.

Entrada no diretório de Páginas		Entrada na tabela de páginas		Efeito combinado	
Privilégio	Tipo de acesso	Privilégio	Tipo de acesso	Privilégio	Tipo de acesso
Usuário	Somente para leitura	Usuário	Somente para leitura	Usuário	Somente para leitura
Usuário	Somente para leitura	Usuário	Leitura e escrita	Usuário	Somente para leitura
Usuário	Leitura e escrita	Usuário	Somente para leitura	Usuário	Somente para leitura
Usuário	Leitura e escrita	Usuário	Leitura e escrita	Usuário	Leitura e escrita
Usuário	Somente para leitura	Supervisor	Somente para leitura	Supervisor	Leitura e escrita
Usuário	Somente para leitura	Supervisor	Leitura e escrita	Supervisor	Leitura e escrita
Usuário	Leitura e escrita	Supervisor	Somente para leitura	Supervisor	Leitura e escrita
Usuário	Leitura e escrita	Supervisor	Leitura e escrita	Supervisor	Leitura e escrita
Supervisor	Somente para leitura	Usuário	Somente para leitura	Supervisor	Leitura e escrita
Supervisor	Somente para leitura	Usuário	Leitura e escrita	Supervisor	Leitura e escrita
Supervisor	Leitura e escrita	Usuário	Somente para leitura	Supervisor	Leitura e escrita
Supervisor	Leitura e escrita	Usuário	Leitura e escrita	Supervisor	Leitura e escrita
Supervisor	Somente para leitura	Supervisor	Somente para leitura	Supervisor	Leitura e escrita
Supervisor	Somente para leitura	Supervisor	Leitura e escrita	Supervisor	Leitura e escrita
Supervisor	Leitura e escrita	Supervisor	Somente para leitura	Supervisor	Leitura e escrita
Supervisor	Leitura e escrita	Supervisor	Leitura e escrita	Supervisor	Leitura e escrita

Tabela 3.14 - Proteção de diretório com tabela de páginas

NOTAS SOBRE A PROTEÇÃO DE PÁGINAS

Alguns acessos são verificados como se fossem do nível 0, independente do valor do CPL:

- Acesso a segmento de descritores (LDT, GDT, TSS e IDT).
- Acesso a stacks internos durante uma instrução de CALL, ou exceções e interrupções quando ocorre mudança no nível de privilégio.

Associando Proteção de Segmentos e de Páginas

Quando a paginação está habilitada, o processador primeiro interpreta a proteção de segmento, e em seguida a proteção da página. Em qualquer dos procedimentos, uma vez identificada violação aos mecanismos de proteção, é gerada uma exceção e o processamento não continua. Se uma exceção é gerada pela segmentação, não o será pela paginação na mesma operação.

MULTITAREFAMENTO

O processador 80486 fornece suporte de hardware para a implementação de multitarefamento. Uma tarefa é um programa que está rodando ou aguardando ser rodado enquanto outro é executado. A tarefa é invocada por interrupção, exceção, desvio (jump) ou chamada (call). A transferência de execução feita por uma das formas citadas, permite que o estado da tarefa corrente seja preservado antes do início da nova tarefa. Isto é possível a partir de determinados tipos de entradas existentes nas tabelas de descritores. Existem dois tipos de descritores relacionados com tarefas: descritores de segmento de estado de tarefa e gates de tarefas. Quando a execução é passada por um desses dois tipos de descritores, um chaveamento de tarefa ocorre.

O chaveamento de tarefa é semelhante a uma chamada (call), com a diferença de maior número de informações, relativas ao estado do processador, preservadas. A chamada a um processo preserva no stack o conteúdo dos registradores de uso geral além do registrador EIP. O processo pode, com a preservação destes registradores, chamar a si mesmo. Quando isto é possível, o processo é chamado de reentrante.

A mudança de tarefa transfere a execução completamente para um novo ambiente: o ambiente de uma tarefa em particular. Isto requer a preservação de quase todos os registradores do processador, como o EFLAGS, por