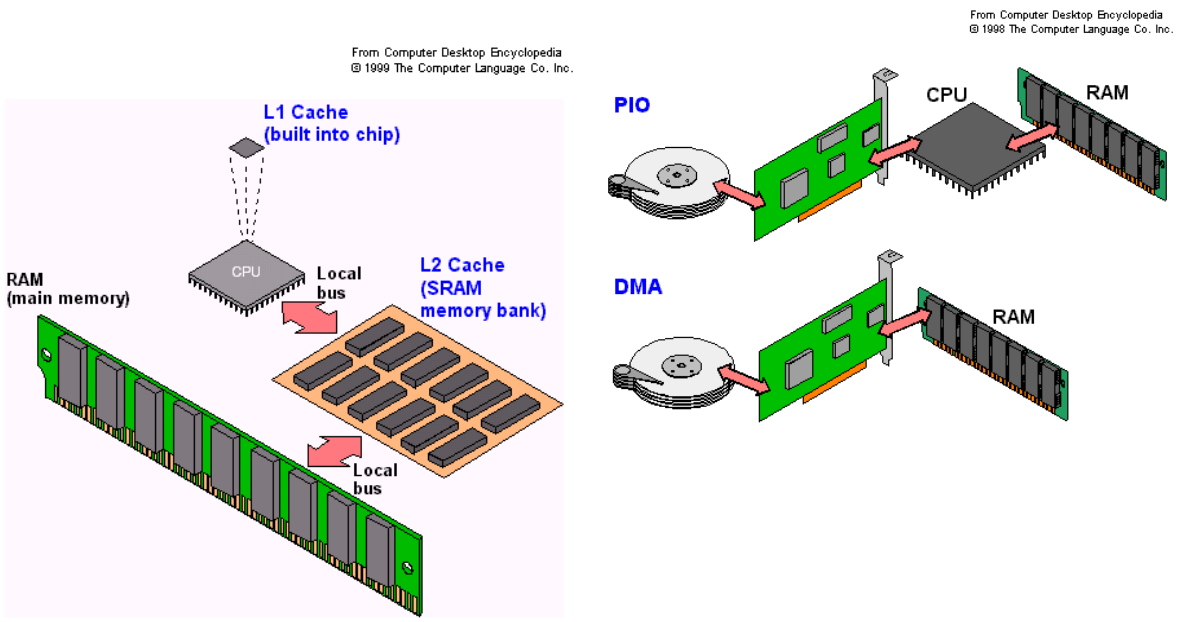
Tópico 5

Sistema de Memória: Tecnologia

Autores: José Raimundo de Oliveira e Wu Shin-Ting
DCA - FEEC - Unicamp
Agosto de 2019

Memórias são dispositivos capazes de reter dados binários, formados pelo alfabeto {0,1}. Elas, juntamente com a CPU, constituem um componente básico da máquina de von Neumann (Seção 4.2). Figura 5.1 ilustra um sistema de memória constituído de vários módulos encapsulados de circuitos integrados de distintas tecnologias, com custos e desempenhos bem variados. Temos as **memórias de acesso aleatório estáticas**, em inglês *static random-access memory* (SRAM), de diferentes velocidades, ultra-rápidas e menores (memória *cache* L1) e maiores e rápidas (memória *cache* L2), **memórias de acesso aleatório dinâmicas**, em inglês *dynamic random-access memory* (DRAM ou simplesmente RAM), e **dispositivos de armazenamento em massa**, como o **disco rígido,** em inglês *harddisk* (HD), e o **disco do estado sólido**, em inglês *state-solid disk* (SSD), fitas magnéticas, em inglês *tape*, CDs e DVDs. As transferências dos dados entre as memórias de massa e o barramento do sistema são gerenciadas pelos **controladores de disco**. Na figura são ainda mostrados dois modos de transferência via controladores de disco: modo de **entrada/saída programado**, em inglês *Programmed Input/Output* (PIO), e modo de **acesso direto à memória**, em inglês *Direct Memory Access* (DMA).



(a) (b)

Figura 5.1: Sistema de memória acoplado a uma CPU.

No Capítulo 4 vimos que um ciclo de instrução envolve três fases: busca, decodificação e execução. Dependendo da arquitetura do microprocessador, todas

as três fases podem envolver transferências de dados entre a CPU e a memória. As setas vermelhas na Figura 5.1 indicam os fluxos de dados através das conexẽos entre elas. A fase de busca corresponde ao intervalo de tempo em que uma instrução é transferida da memória para o registrador de instruções da CPU. Na fase de decodificação, operandos necessários para a execução de uma instrução podem estar na memória e precisam ser transferidos para os registradores. E na fase de execução, pode ser necessário armazenar os resultados das instruções na memória.

Vimos também no Capítulo 4 que um processador tem integrado nele um conjunto de **registradores** para armazenar as instruções e os dados, pois todas as operações, inclusive as de controle, são executadas com os operandos em registradores. Sendo o espaço de armazenamento dos registradores muito limitado, vimos que há instruções dedicadas para transferir os dados de uma memória de capacidade maior para os registradores antes da execução de uma instrução. Estas memórias que são acessíveis pelas instruções do processador são conhecidas por **memória principal**. Para tirar proveito das tecnologias disponíveis, a memória principal é organizada em diferentes níveis de *cache* (Figura 5.1). Voltaremos a discutir esta organização no Capítulo 6. Finalmente, há ainda memórias de massa denominadas usualmente de **memória secundária** (*storage*). Para transferências de dados entre as memórias secundárias e memória principal, usa-se usualmente os comandos de um sistema operacional. O sistema operacional, em conjunto com um **controlador de memória** ou **unidade de gerenciamento de memória**, consegue expandir virtualmente o tamanho da memória principal, fazendo transferências dos dados da memória de massa para a memória principal de forma transparente. Figura 5.2 mostra os conectores disponíveis numa placa-mãe onde se encontram a CPU, *chipset* e os módulos de memória. Observe que o BIOS (Seção 4.6) está armazenado numa **memória somente de leitura**, em inglês *read only memory* (ROM).

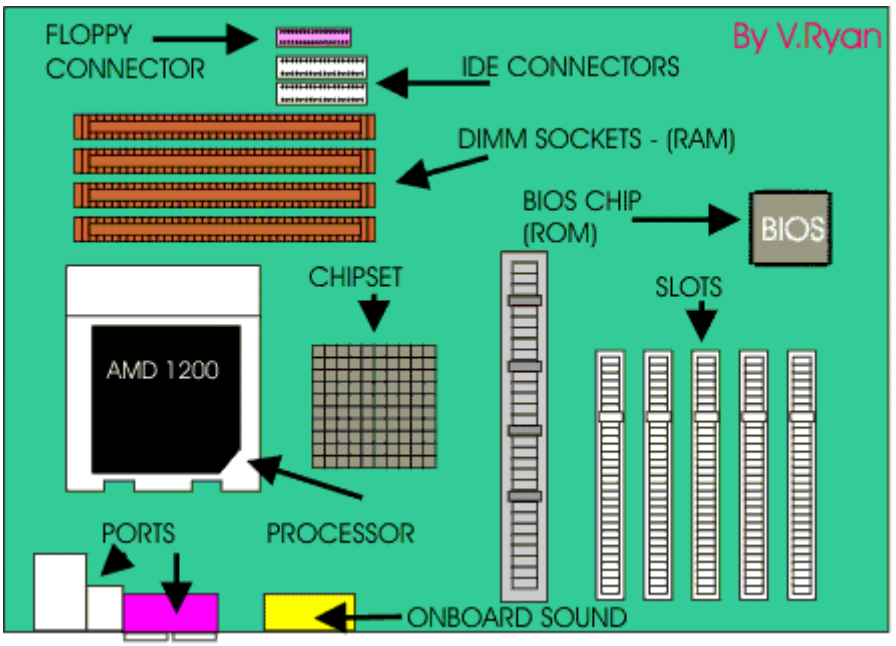


Figura 5.2: Placa-mãe com os conectores para CPU e memórias (Fonte: [1]).

Sintetizando, o sistema de memória de um sistema computacional é constituído por módulos de memória de diferentes tecnologias de diferentes capacidades de armazenamento e velocidades (Figura 5.3).

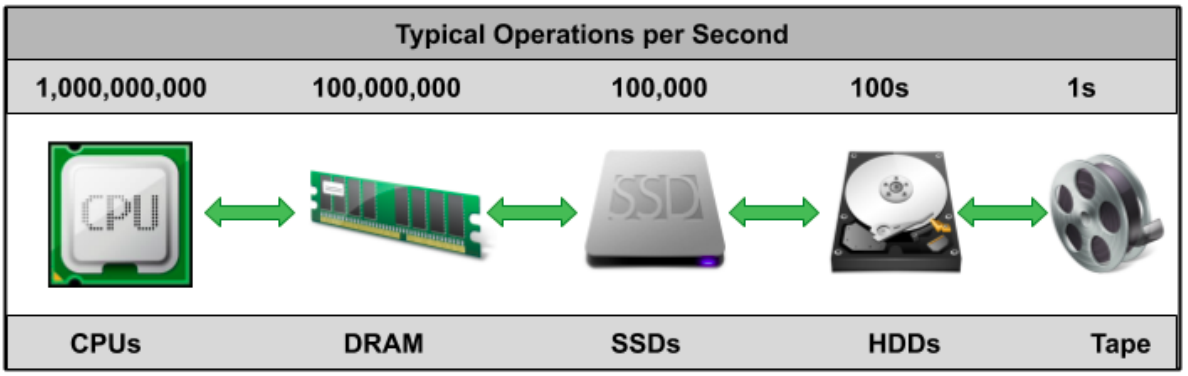


Fig. 4.3: Diferentes tecnologias num sistema de memória (Fonte: [2]).

Não menos importante, são os circuitos responsáveis pela **interconexão** entre estes elementos de memória, que são disponíveis como módulos independentes pelos seus fabricantes, e a CPU através de *chipsets* (Seção 4.7). Na Figura 5.4 são apresentados três padrões de barramento que podem estar presentes numa placa-mãe: barramento de **arquitetura do padrão industrial**, em inglês *Industry Standard Architecture* (ISA), pouco encontrado nos computadores pessoais modernos, barramento **local de VESA**, em inglês *VESA Local* (VL), cujo padrão foi estabelecido pela *Video Electronics Standards Association*, e barramento de **interconexão de componentes periféricos**, em inglês *Peripheral Component*

Interconnect (PCI), capaz de transmitir dados a uma taxa de até 132 MB/s e, hoje em dia, tomou o espaço dos dois barramentos anteriores.

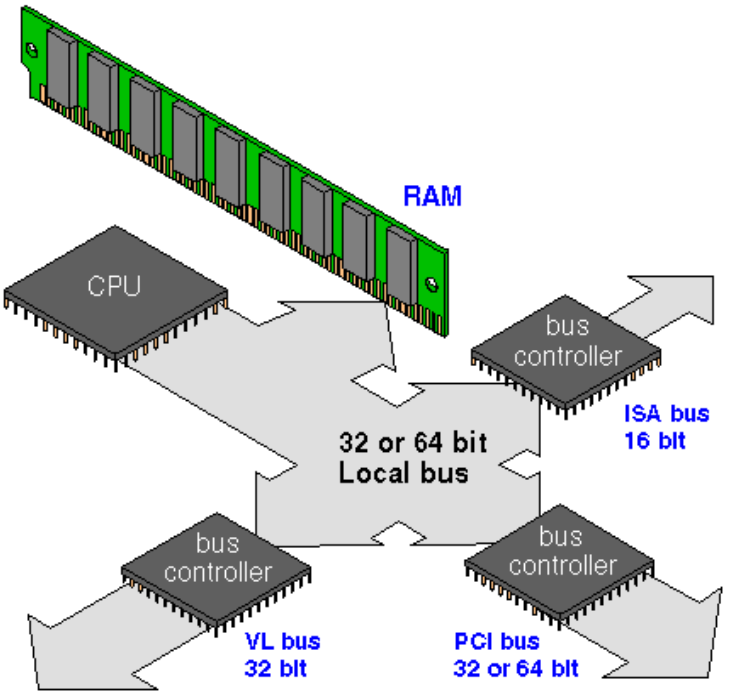


Figura 5.4; Diferentes barramentos de conexão entre a CPU e as memórias (Fonte: [3]).

Neste capítulo vamos fazer uma breve descrição de uma seleção de tecnologias representativas e mostrar a solução tecnológica para conectar estas tecnologias com uma CPU. Essa solução consiste em abstrair todos os dispositivos físicos em endereços que uma CPU consegue entender.

5.1 Tecnologias em Memória Principal

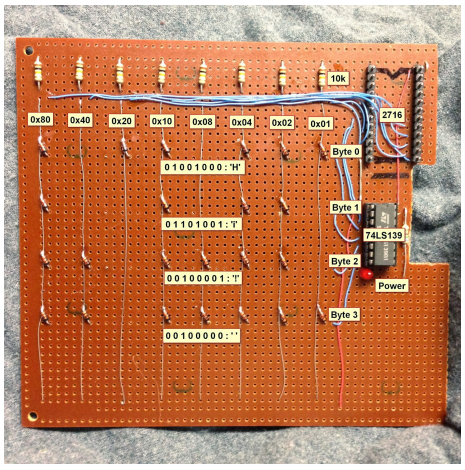
Como já comentamos, a principal função da memória principal é prover para a CPU as instruções e os dados necessários na execução de uma tarefa pré-programada. Por isso, ela precisa ser organizada de forma que tenha uma velocidade de acesso compatível com a velocidade de processamento da CPU para não virar um **gargalo da memória**, em inglês *memory bottleneck*, isto é, ser um ponto que causa a redução na taxa de transferência dos dados entre a CPU e a memória principal. Veremos no Capítulo 6 como podemos organizar as diferentes tecnologias de memória de maneira a nos aproximar do desempenho ótimo. Vale comentar que a tecnologia do barramento que conecta as duas unidades pode também ser um gargalo da memória. Preferivelmente, a sua velocidade de transferência não deve ser menor que a velocidade de processamento das duas unidades, pois o sistema sempre se “iguala por baixo” nivelando-se pela menor velocidade [4].

Dentre os módulos de memória que fazem parte da memória principal distinguem-se as **memórias voláteis** e **as não voláteis**. As memórias voláteis são aquelas que não retêm dados quando desenergizadas, enquanto as memórias não-voláteis retêm os dados mesmo sem alimentação. Mesmo que nunca seja necessário projetar ou construir um módulo de memória, é interessante que os projetistas de sistemas embarcados tenham uma compreensão melhor dos fatores que limitam o desempenho de cada tipo de memória.

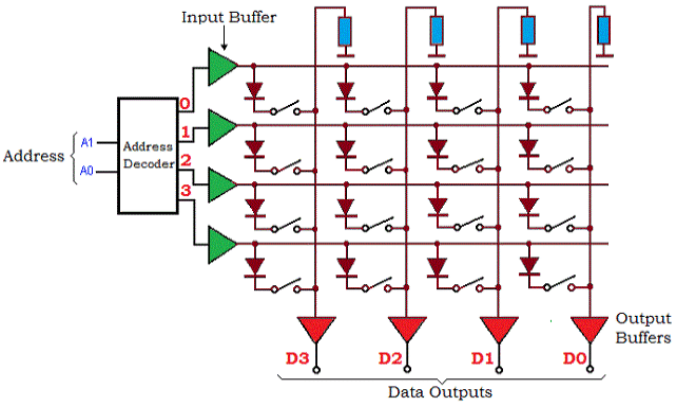
5.1.1 Memórias Não-Voláteis

As memórias não-voláteis se caracterizam por preservar seus dados gravados enquanto não forem deliberadamente apagados ou sobreescritos.

Memória somente de leitura, em inglês *Read Only Memory* (**ROM**), é um tipo de memória que, na sua concepção original, permite apenas a leitura dos dados. As primeiras versões eram circuitos combinacionais no formato matricial de *n-bits* de endereços (linhas) e *m-bits* de palavra (colunas), de maneira que quando uma palavra era selecionada os valores lógicos dos *m-bits* pré-definidos eram transferidos para a saída dos circuitos. Figura 5.5 apresenta uma implementação da memória ROM usando os diodos (componentes discretos). Note que os pontos A3-A0 e D3-D0 da Figura 5.5(b) correspondem aos pinos físicos através dos quais a memória se comunica com os outros dispositivos.



(a) Fonte: [7]



(b) Fonte: [6]

Figura 5.5: Memória ROM como uma matriz de diodos.

Com a invenção de circuitos integrados, surgiram as **memórias somente de leitura de máscara**, em inglês *mask read only memory* (**MROM**), em que os *bits* dos dados eram gravados no processo de fabricação, como mostra a Figura 5.6.(a). Cada linha

corresponde a um endereço da palavra e cada *bit* da palavra é “armazenado” em um transistor (bipolar ou CMOS). **Conexões baseadas na lógica OU**, em inglês *or-wired*, são feitas entre as linhas da memória de forma que a saída da memória seja uma das linhas selecionadas [5]. Na figura, o nível lógico é normalmente 1 (tensão positiva) e, quando uma linha é selecionada, as colunas que têm *bits* com nível lógico 0 são fechadas para o terra (tensão 0).

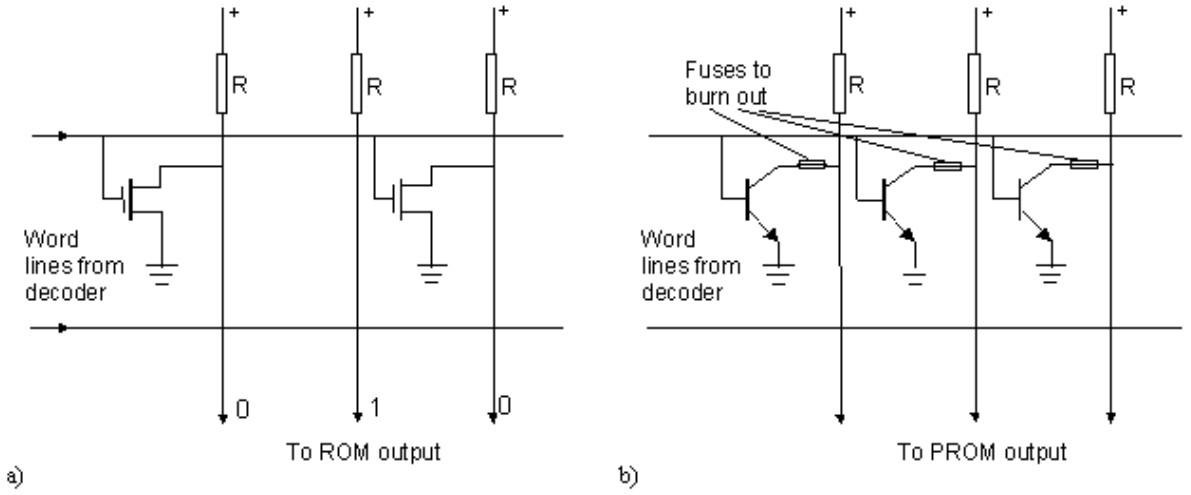


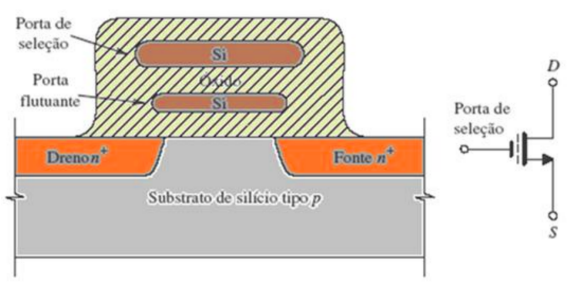
Figura 5.6: Diferença básica entre MROM e PROM.

O fato das MROM serem dedicadas, fabricadas junto com os seus dados no processo de fabricação, elas são mais compactas, baratas e rápidas. Hoje em dia, elas são usadas para armazenar sistemas operacionais, fontes das impressoras, dados de áudio de instrumentos musicais [8], enfim dados para os quais a probabilidade de modificação seja ínfima. São, porém, inviáveis as atualizações e as manutenções dos programas, sem implicar num novo processo de fabricação. E o custo para produções em baixa escala é muito alto.

Em 1956, Wen Tsing Chow inventou **memórias somente de leitura programáveis**, em inglês *programmable read only memory* (**PROM**s). Essas memórias podem ser programadas após o seu processo de fabricação, rompendo os pequenos fusíveis conectados em cada célula (Figura 5.6.(b)). Portanto, dispositivos de programação especiais são necessários, como mostra a Figura 5.7, e a programação é feita uma única vez. CY7C281A é uma memória CMOS PROM com capacidade de 1K x 8 *bits* [9]. Hoje em dia, estas memórias se encontram em consoles de *video games* e interfaces de multimídia de alta resolução [8]. Embora, sob o ponto de vista de fabricação, estas memórias flexibilizaram reuso de um mesmo circuito, elas continuam limitadas em termos de desenvolvimento de programas. A reprogramação não era permitida.

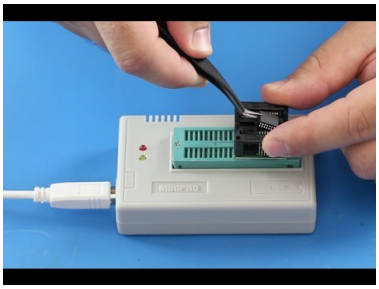
Figura 5.7: PROM e um gravador.

Em 1971, o problema de reprogramação foi solucionado pelo Dov Frohman com o desenvolvimento de **memórias somente de leitura programáveis e apagáveis**, em inglês *erasable programmable read only memory* (**EPROM**s), utilizando transistores MOSFET de portas flutuantes, em inglês *floating gate MOSFET* [14]. A carga armazenada nessa **porta flutuante**, em inglês *floating gate* (**FG**), controla o estado do *bit*. Sendo a porta flutuante "perfeitamente" isolada, o conteúdo dessas memórias pode ser limpo (em nível lógico "1") somente sob uma radiação de comprimento de onda ~4000 Angstroms (A) (ultravioleta) a uma intensidade de 12.000 uW/cm^2 durante cerca de vinte minutos. Para gravar (carregar elétrons em FG de) uma célula, é necessário aplicar uma tensão de 12-25V na porta de seleção. Isso força que alguns elétrons, que fluem da fonte para dreno, atravessem o material isolante entre o substrato e FG e alojem-se na FG, carregando esta porta com cargas negativas (em nível lógico "0") . Este fenômeno é conhecido por **injeção de elétrons quentes** (Figura 5.10.(b)). São permitidos em torno de milhares de ciclos de regravações e, uma vez gravado um conteúdo, essas memórias conseguem retê-lo por 10 a 20 anos. Para reduzir o custo de fabricação, há uma família de EPROMs, conhecida por **programável por uma vez**, em inglês *One Time Programmable* (OTP), que não tem a janela para apagar o conteúdo; portanto, o módulo é programável por uma única vez. Figura 5.8 mostra uma memória EPROM reprogramável e os acessórios (dispositivos para limpar e gravar os dados nela). As memórias EPROMs foram muito utilizadas no armazenamento das instruções em microcontroladores e de BIOS (Seção 4.6). EPROM 27C64 de 64K da *Microchip* é um exemplo desta classe de memória [10]. As limitações apresentadas por EPROMs são: (1) a demanda de um dispositivo dedicado para limpar o conteúdo da memória, (2) o tempo da operação de limpeza e (3) a extensão do conteúdo limpo em cada ciclo de regravação [8].



(a) Célula EPROM

(b) EPROM com janela de quartzo

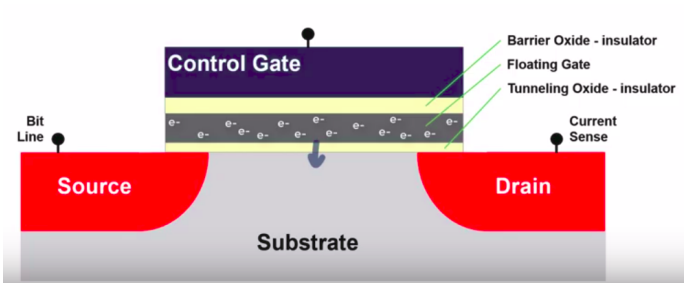


(c) gravador

(d) apagador

Figura 5.8: EPROM e seus acessórios (Imagem 4.8.(a) foi extraída de [40])

Em 1972, Yasuo Tarui, Yutaka Hayashi and Kiyoko Naga desenvolveram no Laboratório Eletrotécnico, do Instituto Nacional de Ciências e Tecnologias Industriais Avançadas no Japão, as **memórias somente de leitura programáveis e eletricamente apagáveis**, em inglês *electrically erasable programmable read only memory* (EEPROMs), utilizando o mecanismo de **tunelamento de Fowler-Nordheim** em MOSFET de portas flutuantes [15]. A ideia consiste em reduzir a distância entre o substrato e FG para cerca de ⅓ da espessura em EPROMs, de forma que essas memórias possam ser limpas eletricamente, aplicando internamente uma tensão alta, entre 12V a 20V, em torno de 1 segundo. Esta tensão força a migração das cargas armazenadas na porta flutuante (nível lógico “0”) para o substrato, de maneira que FG retorne ao seu estado original (nível lógico “1”) (Figura 5.9.(a)) [14]. Circuitos adicionais de controle do tunelamento precisaram ser integrados. Isso aumentou a área de ocupação de cada célula. Um exemplo da EEPROM com interface paralela é a memória AT28C256 da *Microchip* (Figura 5.9.(a)) [12]. Hoje em dia, EEPROM são amplamente aplicadas no armazenamento de instruções de tarefas dedicadas e reprogramáveis, como a BIOS (Seção 4.6) e os dados de calibração de equipamentos de teste [8]. A grande limitação da memória EEPROM é o tempo e a quantidade de *bytes* (alguns *bytes*) numa limpeza ou numa reprogramação. A sua vida útil é tipicamente um milhão de reprogramações.

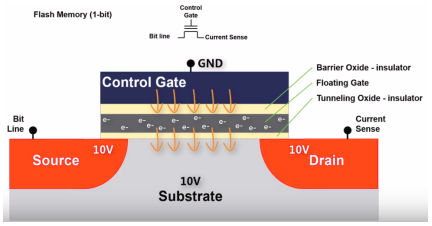


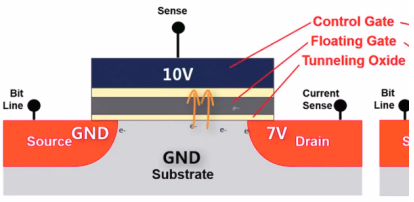
(a) Limpeza por tunelamento de Fowler-Nordheim (Fonte: [38])

(b) EEPROM encapsula

Figura 5.9: EEPROM.

Em 1980, Fujio Masuoka desenvolveu na Toshiba a memória **FLASH** a partir da EEPROM (Figura 5.9.(b)). As principais diferenças estão na forma como os dados são limpos e como os dados são gravados. Ao invés de limpeza por *byte*, aplica-se a tensão de limpeza em todo o substrato de forma que um bloco de células possa voltar ao estado original simultaneamente em "tempo de *flash* de uma câmera" (Figura 5.10.(a)). Note que todas células do bloco devem estar em "0" (programadas) antes de iniciar a limpeza. Ao invés de gravar o estado "0" numa célula por tunelamento, utiliza-se a técnica de injeção de elétrons quentes (Figura (4.10.(b)) [14]. E as células devem estar previamente limpas (em estado "1").





(a) Limpeza por bloco (Fonte: [38])

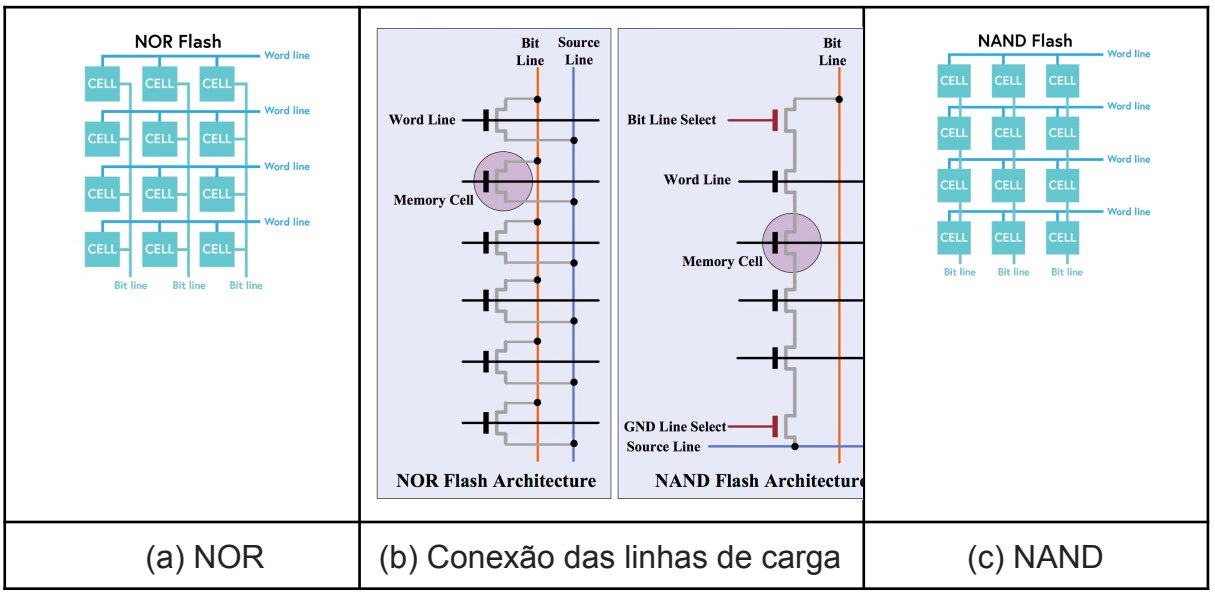
(b) Injeção de elétrons quentes (Fonte: [39])

(c) FLASH encapsulada

Figura 5.10: FLASH.

As células de uma memória FLASH são tipicamente conectadas como uma matriz bidimensional. Duas principais organizações das suas células são as que se assemelham com a organização da porta lógica NOR (**FLASH NOR**) e as com a organização da porta lógica NAND (**FLASH NAND**), como mostra a Figura 5.11. As **linhas de palavra**, em inglês *word line*, e as **linhas de *bit***, em inglês *bit line*, correspondem, respectivamente, às linhas e às colunas da conexão matricial das

células. As primeiras linhas são responsáveis pela carga das células e as segundas linhas controlam o fluxo de carga de uma célula para outra ao longo de uma coluna. Nas memórias FLASH NOR a linha de carga, em inglês *source line*, é conectada em cada célula, enquanto nas memórias NAND, esta linha de carga é conectada com até 8 células em série, como mostra a Figura 5.11.(b). Portanto, enquanto as memórias FLASH NOR permitem acessos por *bytes* ou palavras, as memórias FLASH NAND fazem acessos por páginas.



(a) NOR | (b) Conexão das linhas de carga | (c) NAND

Figura 5.11:: Organização e conexão das células em memórias FLASH (Fonte: [49]).

Tabela 5.1 sintetiza comparativamente as principais características das duas classes de FLASH [16]. Como as células de FLASH NOR são endereçáveis individualmente, acessos aleatórios de leitura são mais rápidos. Por outro lado, as gravações individuais de “0” em todas as células antes da limpeza tornam-na mais lenta. Além disso, as células em FLASH NOR são maiores. Por isso, FLASH NOR tem uma densidade menor, um custo maior e requer um tempo maior para gravar “0”. Hoje em dia, as FLASH NOR aplicam a técnica de injeção de elétrons quentes, e as FLASH NAND a técnica de tunelamento, para carregarem as cargas na porta flutuante (nível lógico “0”). A percentagem de blocos defeituosos é maior em FLASH NAND.

Feature	NOR Flash		NAND Flash	
	General	S70GL02GT	General	S34ML04G2
Capacity	8MB – 256MB	256MB	256MB – 2GB	256MB
Cost per bit	Higher	$6.57x10^{-9}$ USD/bit for 1ku	Lower	$2.533x10^{-9}$ USD/bit for 1ku
Random Read speed	Faster	120ns	Slower	30µS
Write speed	Slower		Faster	
Erase speed	Slower	520ms	Faster	3.5ms
Power on current	Higher	160mA (max)	Lower	50mA (max)
Standby current	Lower	200µA (max)	Higher	1mA (max)
Bit-flipping	Less common		More common	
Bad blocks while shipping	0%		Up to 2%	
Bad block development	Less frequent		More frequent	
Bad block handling	Not mandatory		Mandatory	
Data Retention	Very high	20 years for 1K program-erase cycles	Lower	10 years (typ)
Program-erase cycles	Lower	100,000	Higher	100,000
Preferred Application	Code storage & execution		Data storage	

Tabela 5.1: FLASH NOR vs. FLASH NAND (Fonte: [16]).

Pelo seu desempenho nos acessos de leitura e pela sua confiabilidade (Tabela 5.1), as FLASH NOR são utilizadas como memória principal pelos projetistas de sistemas embarcados. Figura 5.10.(c) mostra uma FLASH NOR encapsulada de maneira que tanto os endereços quanto os dados possam ser acessados em paralelo: o módulo MX29F800C [19]. As FLASH NAND, por sua vez, são aplicadas na construção de dispositivos de armazenamento de massa (**memória secundária**), como os dispositivos de armazenamento removíveis USB (*Universal Serial Bus*), os **cartões de memória** e os **discos de estado sólido**. Seus acessos de leitura e de escrita são feitos por blocos de páginas como quaisquer dispositivos de armazenamento de massa (Seção 5.2). Na Seção 5.4.2 apresentaremos as suas interfaces com um sistema computacional, como periféricos de armazenamento de massa.

Tabela 5.2 apresenta comparativamente as características das três principais classes de memórias não-voláteis programáveis vistos nesta seção[14].

	EPROM	EEPROM	FLASH NOR
Tamanho normalizado da célula	1.0	3.0	1.0-1.2
Mecanismo de programação	Injeção de elétrons quentes	Tunelamento FN	Injeção de elétrons quentes

Mecanismo de descarga da FG (*erase*)	Luz ultra-violeta	Tunelamento FN	Tunelamento FN
Tempo de descarga	20 minutos	5ms	1s
Extensão de limpeza	Toda pastilha	*byte*	Todo substrato
Ciclo de escrita (por célula)	< 100µs	5 ms	< 100µs
Ciclo de leitura	200ns	35ns	200ns

5.1.2 Memórias Voláteis

Memórias voláteis tem como sua principal aplicação a construção da memória principal de um sistema computacional. Estas memórias se caracterizam por não conseguirem reter o seu conteúdo removido quando não são energizadas. Além de serem mais rápidas do que os dispositivos de armazenamento de massa, elas podem proteger dados sensíveis, no sentido de que tais dados serem seguramente removidos quando se desliga a fonte de alimentação. Na concepção original, as memórias voláteis eram as memórias de acesso de leitura e de escrita em qualquer ordem, ou seja eram as **memórias de acesso aleatório**, em inglês *Random Access Memory* (**RAM**), cujas células eram baseadas em *latches* do tipo D.

Hoje em dia, distinguem-se duas grandes classes de memórias RAM: estáticas (sRAM) e dinâmicas (DRAM). As estáticas são as da tecnologia transistorizada (Figura 5.11), enquanto as dinâmicas utilizam a tecnologia capacitiva para reter os *bits* (Figura 5.12). Nesta seção vamos dar uma breve descrição dessas duas memórias.

5.1.2.1 Estáticas (sRAM)

Memórias **RAM estáticas**, em inglês *static RAM* (**sRAM**), tem acesso rápido e determinístico. São simples e confiáveis. A primeira RAM estática comercial, RAM estática 3101 Schottky TTL bipolar 64-*bit*, foi lançada pela Intel em 1969.

As RAMs estáticas operam em três modos:

- *standby*: modo de economia de energia.
- leitura: modo de acesso de leitura.
- escrita: modo de acesso de escrita.

Elas podem ser implementadas com a tecnologia bipolar (Figura 5.12.(a)) ou com a tecnologia CMOS (Figura 5.12.(b)). As memórias implementadas com os transistores bipolares consomem mais potência do que as da tecnologia CMOS, porém são mais rápidas. As da tecnologia CMOS são mais densas do que as da tecnologia bipolar. O consumo de potência das sRAMs baseadas em CMOs é quase desprezível quando a memória se encontra no estado ocioso, em inglês *idle*. Em relação às DRAMs, sRAMs tem mais elementos em cada célula, ou seja, são menos densas do que as DRAMs. As sRAMs são mais caras do que as DRAMs. Portanto, sRAMs são consideradas as memórias “nobres”, reservadas para tarefas específicas que requer alto desempenho e pouca memória, como a memória *cache* que veremos no Capítulo 5. Um exemplo de memórias estáticas é MCM6264C da Motorola [17].

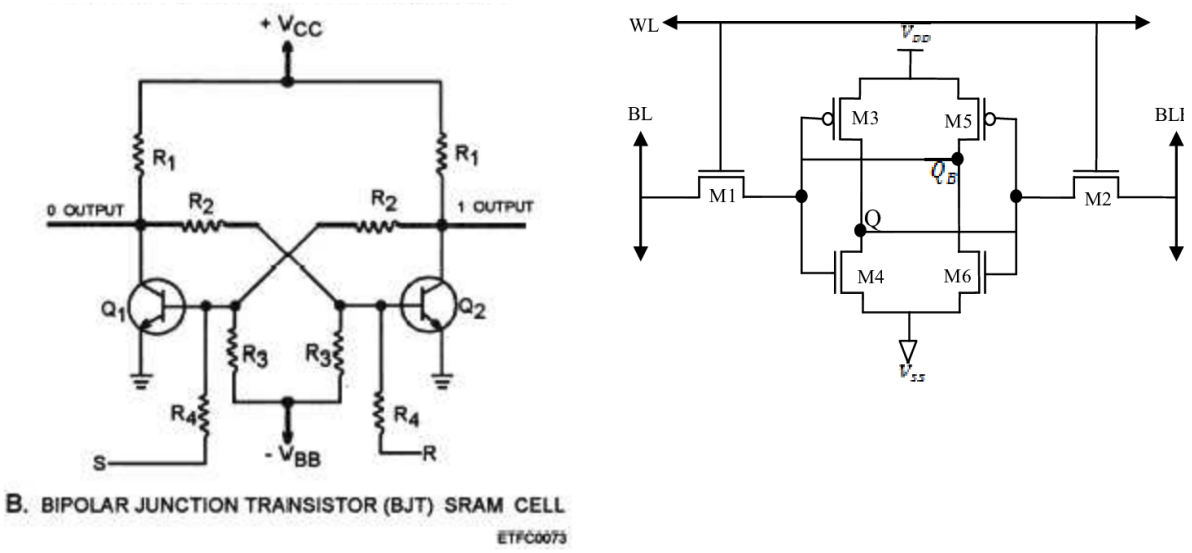
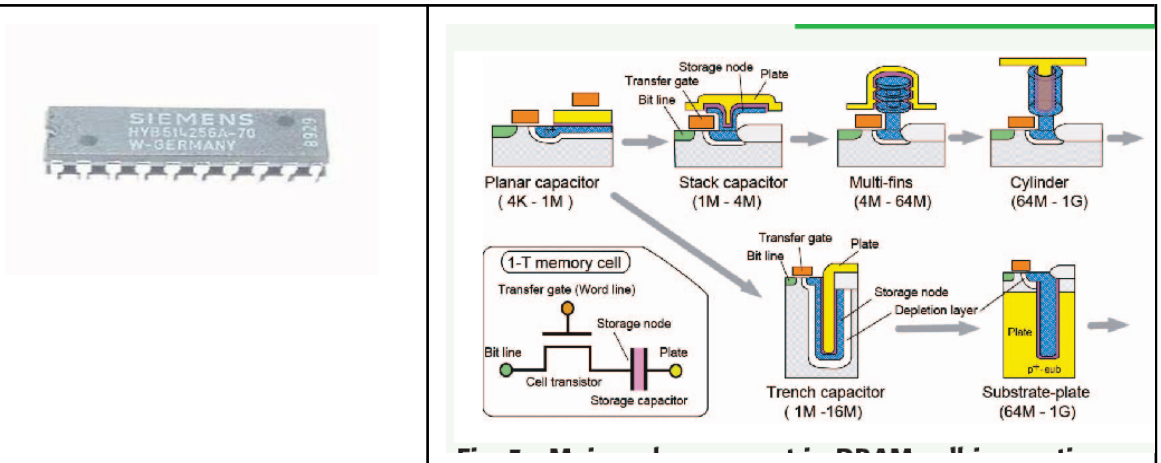


Figura 5.12: RAMs estáticas.

5.1.2.2. Dinâmicas (DRAM)

Memórias RAM dinâmicas, em inglês *dynamic RAM* (DRAM) armazenam os dados como carga de um capacitor de tecnologia MOS (Figura 5.12). Figura 5.12.(b) mostra diferentes tecnologias de capacitores aplicadas na construção de DRAMs. A ideia de se usar capacitores como uma forma de armazenamento de dados é conhecida desde a Segunda Guerra Mundial. A primeira DRAM comercial, com a capacidade de 1024x8bits, só foi lançada pela Intel em 1970.



(a) DRAM encapsulada	(b) Uma célula

Figura 5.12: DRAM.

Internamente, as células da memória são organizadas como um arranjo bi-dimensional cujas células são endereçáveis por dois índices (linha,coluna), como mostra a Figura 5.13. Como os capacitores descarregam em cada operação de leitura, o "**amplificador de sentido"**, em inglês *sense amplifier*, detecta a tensão da célula lida e restaura esta tensão antes do fim de cada ciclo de acesso de leitura. As células, que não são acessadas para leitura, também se descarregam pelas fugas, em inglês *leak*, naturais. Para assegurar a integridade dos dados, é integrado a DRAMs o mecanismo conhecido por **regeneração**, em inglês *refreshing*, periódica para a recarga por linha. O período de regeneração pode variar de 2ms a 128ms, controlado pelo Contador de *Refresh*, nas memórias modernas.

Como são poucos os elementos necessários para formar uma célula da DRAM (Figura 5.12.(b)), consegue-se alojar mais células numa pastilha de DRAM. Esta quantidade pode chegar na ordem de mega palavras, requerendo 20 ou mais *bits* para endereçar de forma unívoca todas as palavras da memória. Para encapsular essas memórias com os pinos resistentes o suficiente para conectá-las eletrica e mecanicamente com outros componentes de um sistema computacional, é preciso usar encapsulamentos que ocupam áreas muito maiores do que a da pastilha de silício. Diferente do módulo HYB514256BJL [18] mostrado na Figura 5.12.(a), a maioria das DRAMs multiplexa os seus endereços da coluna com os da linha, para reduzir a quantidade de pinos. Elas dispõem de dois sinais de *strobe*, *row address strobe* (/**RAS**) e *column address strobe* (/**CAS**), para controlar o armazenamento dos índices da linha e da coluna nos registradores *Row Address Buffer* e *Column Address Buffer*, como esquematizado na Figura 5.13. Note que as linhas do contador de regeneração e as linhas de endereçamento são multiplexadas, pois periodicamente uma fração de tempo de operação da memória é usada na regeneração por linhas.

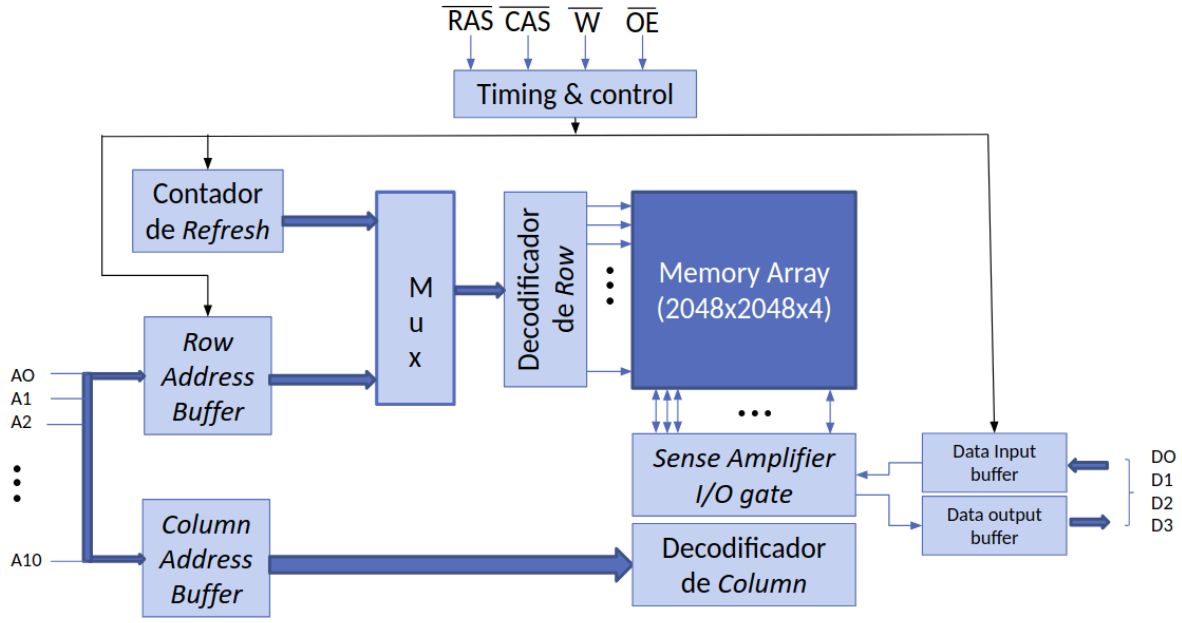


Figura 5.13: Organização de uma DRAM

Até agora consideramos que as memórias podem ser abstraídas como mostrada na Figura 5.5(b), em que os valores dos pinos de dados são resultados de uma combinação lógica dos valores dos pinos, em paralelo, de endereços. Com o multiplexamento de endereços, no mínimo dois ciclos de relógio são necessários para obter um endereço integral para iniciar o acesso. Além disso, a recarga das células acessadas entre dois acessos introduz também um atraso entre acessos sucessivos. Alternativas de **modos de operação** foram propostas para reduzir estes atrasos:

- **modo página**, em inglês *page mode*: permite sucessivos acessos de colunas mantendo fixo o índice da linha, incrementando somente o índice da coluna. Tempos de processamento de linhas são removidos (Figura 5.21).
- **modo saída de dados estendida**, em inglês *Extended Data Out* (EDO): permite que sucessivos acessos de colunas sejam feitos numa frequência pré-definida, sem aguardar que o ciclo predecessor conclua, ou seja, permite fazer **escrita precoce**, em inglês *early write*. Isso diminui o ciclo de acesso a uma página (Figura 5.22).
- **modo leitura-escrita**, em inglês *read-modify-write*: permite que seja feito um acesso de leitura seguido de um acesso de escrita numa mesma posição da memória (Figura 5.23).

A periodicidade de regeneração é padronizada pelo JEDEC (*Joint Electronics Device Engineering Council*) para cada tipo de DRAM. Tipicamente, ela gira em torno de dezenas de milisegundos. Vamos ver aqui as diferentes políticas de regeneração. Embora as regenerações periódicas sejam feitas por linhas, nas primeiras DRAMs os endereços das linhas de regeneração precisam ser fornecidas

pelos pinos de endereço da memória. Eram geradas externamente à memória. Esta técnica de regeneração é conhecida por **regeneração por somente RAS**, em inglês *RAS-only refreshing*. Com a integração do circuito gerador de endereços de regeneração nas DRAMs, o Contador de Refresh como mostra a Figura 5.13, tornou-se mais fácil desenvolver circuitos de controle de regeneração que reduza o intervalo de tempo gasto neste procedimento, como a **regeneração CAS antes da RAS**, em inglês *CAS-before-RAS refreshing*, em que os endereços das linhas a serem regeneradas são gerados automaticamente assim que a memória receber uma transição do sinal CAS de 1 para 0, seguida de uma transição do sinal RAS de 1 para 0. O intervalo entre as duas transições devem ser maior que o tempo de *setup* do sinal CAS (Figura 5.24). Um aprimoramento dessa técnica é a **regeneração transparente**, em inglês *hidden refresh cycle*, que executa a regeneração em paralelo aos ciclos de acesso de leitura ou escrita nos intervalos de ociosidade (Figura 5.25).

Um dos maiores problemas associados a DRAMs são os **erros de leitura**, ou seja quando os dados lidos não são iguais aos dados originalmente gravados [14]. As causas podem ser classificadas em: *hard* e *soft*. Os **erros *hard*** são aqueles causados pelas falhas do circuito do dispositivo, que se repetem sob mesmas condições. Os **erros *soft*** são erros decorrentes dos transientes operacionais, difíceis de serem previstos e reproduzidos. Para evitar danos causados por estes erros, são aplicadas as técnicas de detecção e/ou correção de erros na maioria das DRAMs modernas. As técnicas de detecção de erros identificam a ocorrência de erros, enquanto as técnicas de correção procuram fazer correções restaurando o estado original dos dados. Dentre as técnicas existentes na área de Telecomunicações, temos:

- **Código detector de erro binário** (*bit* de paridade): é adicionado um *bit* a mais a uma palavra. Este *bit* representa a paridade dos *bits* “1” que aparecem na palavra. Se o número é par o *bit* de paridade é “0”, caso contrário, “1”. Por exemplo, a palavra de 8 *bits*, 0b10111101, tem 6 (par) *bits* “1”, portanto ela é recodificada como 0b101111010, sendo o *bit* menos significativo o *bit* de paridade.
- **Código corretor de erros**, em inglês *Error Correcting Code* (ECC): são adicionados *bits* de redundância a uma palavra, de forma que o cômputo baseado nos *bits* de redundância nos permite determinar não só a presença de erros como a posição destes erros numa palavra. Distinguem-se duas classes de códigos corretores de erros: os códigos de bloco, em inglês *block codes*, e os códigos de convolução, em inglês *convolutional codes*. Os códigos de bloco codificam na base de blocos, enquanto os códigos convolucionais na base de *bits*. O código de bloco Hamming modificado foi implementado pela *Texas Instruments* no circuito integrado 74LS637 para 8 *bits*.

Originalmente, os passos de operação leitura, escrita e regeneração numa DRAM são executados de forma assíncrona em relação à CPU à qual ela estava conectada. Os sinais \RAS e \CAS são utilizados para efetuar sincronizações por acessos. Em 1992 a Samsung introduziu uma versão síncrona de DRAM, o circuito integrado KM48SL2000, em que todas as operações são sincronizadas com as bordas de subida do sinal de relógio da CPU, ou seja na taxa de dados simples, em inglês *single data rate*. Na arquitetura SDRAM a memória é segmentada em múltiplos bancos de memória o que facilita o entrelaçamento das operações e execução em *pipeling* (Seção 4.2.2). Em 1993, SDRAM foi formalmente estabelecido como um padrão pelo JEDEC (*Joint Electronics Device Engineering Council*).

Memórias com taxa de dados duplos, em inglês *double data rate*, conhecidas por DDR SDRAM foram apresentadas pela Samsung em 1997. Nesta nova versão, as transferências de dados são sincronizadas com as bordas de subida e de descida do sinal de relógio da CPU, duplicando a quantidade de dados transferidos mantendo a mesma frequência do relógio. As versões subsequentes, DDR2, DDR3 e DDR4, representam um aumento em 4x, 8x e 16x as transferências de dados por ciclo de relógio em relação a memórias SDRAMs. Figura 5.14 mostra a evolução das memórias dinâmicas desde 1996. Observe que a taxa máxima de transferência dos dados aumentou exponencialmente enquanto a frequência de operação da memória se mantém praticamente constante.

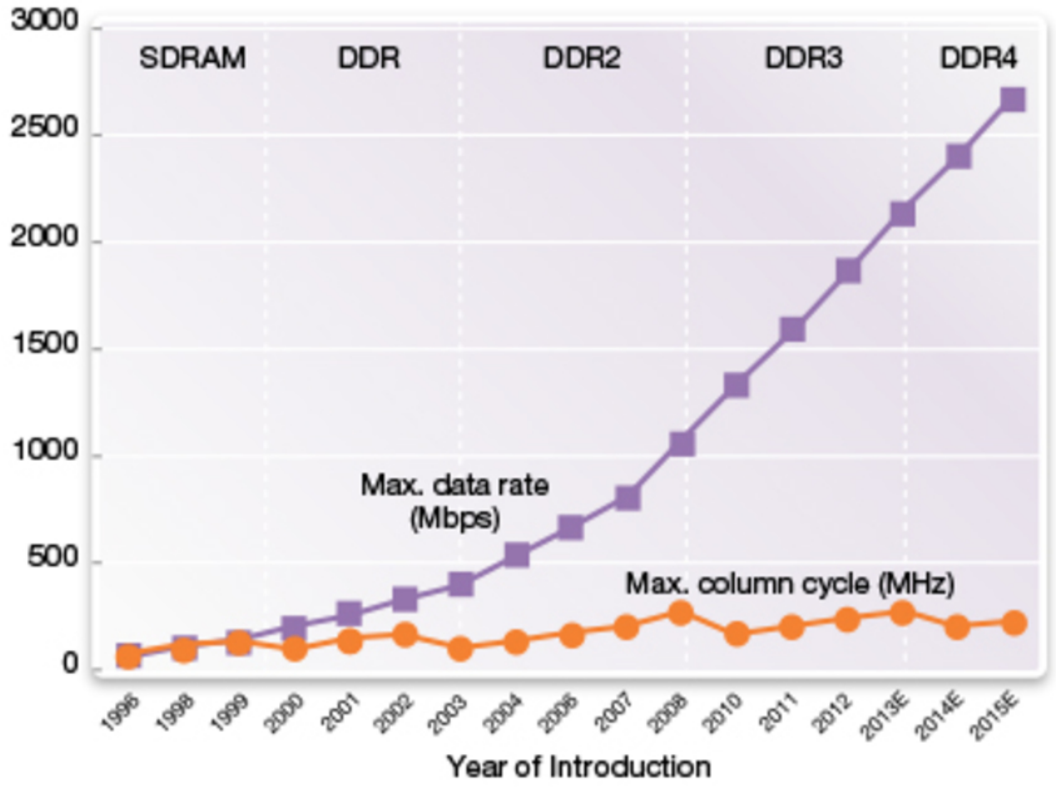
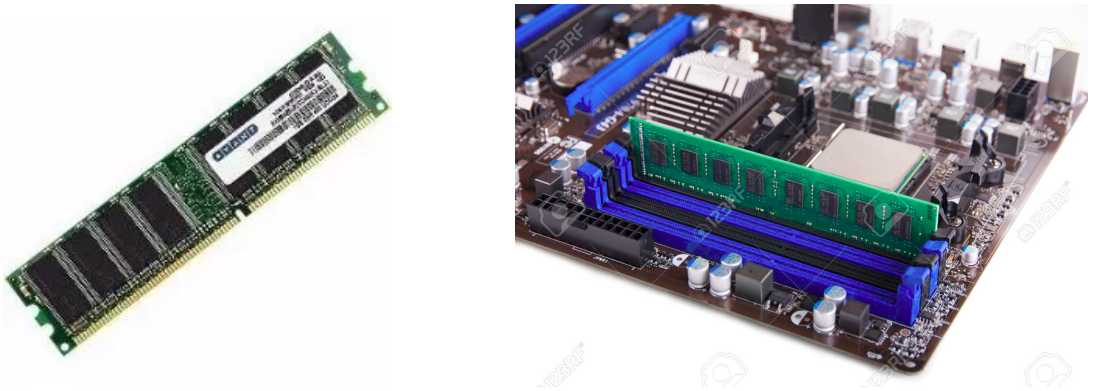


Figura 5.14: Evolução das memórias dinâmicas.

Hoje em dia, as DRAMs e suas variantes vem no formato de “pentes” de memória que são pequenas placas de circuito impresso onde estão soldados os módulos de memória. Estes módulos podem ser organizados em SIMM (*Single In-Line Memory Module*) ou DIMM (*Dual In-Line Memory Module*). Na primeira configuração os módulos são organizados em linha simples e na segunda, em linha dupla eletricamente isolada. Com isso, dobra-se a quantidade de pinos de acesso. As placas-mãe já vem com ranhuras, em inglês *slots*, específicas para alojá-las, como mostra a Figura 5.15.



(a) pente (b) pente numa placa-mãe

Figura 5.15: Pente de módulos de memórias dinâmicas.

5.2 Diagramas de Tempo

Todas as folhas técnicas de um módulo de memória incluem uma descrição funcional, a pinagem, as características elétricas, as características temporais, a capacidade de armazenamento em quantidade de palavras por tamanho da palavra, e os limites máximos, em inglês *maximum rating*, de operação. Dentre estas características, as características temporais são mais complexas, pois há uma interdependência temporal entre os sinais de endereço e os sinais de dados, e os instantes em que os endereços são colocados nos pinos da memória e em que os dados são disponíveis na saída do módulo da memória.

Diferente dos outros parâmetros, é difícil representar com precisão a dinâmica temporal de uma memória textualmente. Diagramas de tempo são recursos adotados nas folhas técnicas para descrever as principais relações temporais. Como projetistas de sistemas embarcados, precisamos saber extrair destes diagramas de tempo os valores necessários para programar corretamente os instantes em que os sinais de endereço devem ser enviados para as memórias selecionadas e os instantes que se deve acessar o conteúdo dos pinos de dados.

Nesta seção vamos mostrar como ver as interdependências dos sinais comentados na Seção 5.2 através dos diagramas de tempo.

5.2.1 Memórias Não-Voláteis e Memórias Estáticas

Quando as interfaces dos endereços e dos dados de uma memória são paralelas, podemos abstrair a organização das ROM e FLASH NOR como um conjunto de sinais de alimentação, de controle, de endereços e de dados. Figura 5.16 ilustra a pinagem do módulo de EEPROM AT28C256 de capacidade 32Kx8 *bits*:

- *Memory enable*: sinal de habilitação do módulo, em inglês *chip enable* (/CE) ou *chip select* (/CS).
- *Read//Write*: modo de acesso, de leitura (R) ou de escrita (W), equivalente ao sinal write enable (/WE).
- *Output Enable*: habilitação do *buffer* de saída (/OE).
- *Memory address*: correspondem aos endereços (A0...A14).
- Data: correspondem aos dados (I/O0 … I/O7).
- Alimentação: VCC (fonte) e GND (terra).

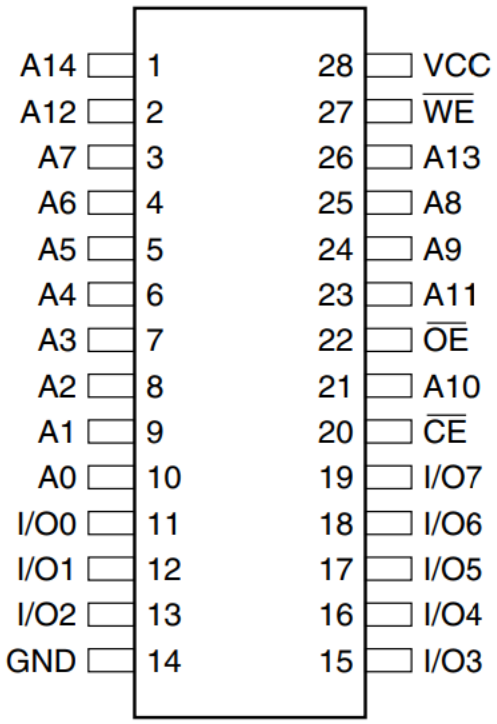


Figura 5.16: Pinagem do módulo EEPROM AT28C256 (Fonte: [12]).

Vale comentar que a técnica de endereçamento das palavras dentro de um módulo de memória é sequenciamento linear de 0 a 2^n, onde n é a quantidade de *bits* de endereços. Por exemplo, o módulo de memória na Figura 5.16 tem um espaço de endereços de 0 a 2^{15} (15 pinos de endereço) e cada endereço tem uma palavra de 8 *bits* (1 *byte*) (8 pinos de dados).

Figura 5.17 apresenta um diagrama de tempo típico de interdependência destes sinais num módulo de memória que está sincronizado com um sinal de relógio *Clock*. Observe que o sinal R//W está no nível lógico “0” indicando que o modo de operação é leitura durante todo o ciclo de acesso. Os endereços devem estar disponíveis nos pinos de endereço da memória, antes de habilitar a memória na borda de subida em T1. Vimos na Seção 5.1 que o tempo de processamento varia com a tecnologia da memória em análise. Para o caso da Figura 5.17, o circuito da memória gerou dados válidos depois de 2 *clocks*. Observe os deslocamentos relativos dos sinais no tempo (eixo x). Estes deslocamentos variam de módulo para módulo. Eles são as **restrições temporais** que existem entre os sinais e são fornecidos pelos fabricantes nas folhas técnicas, em inglês *datasheets*.

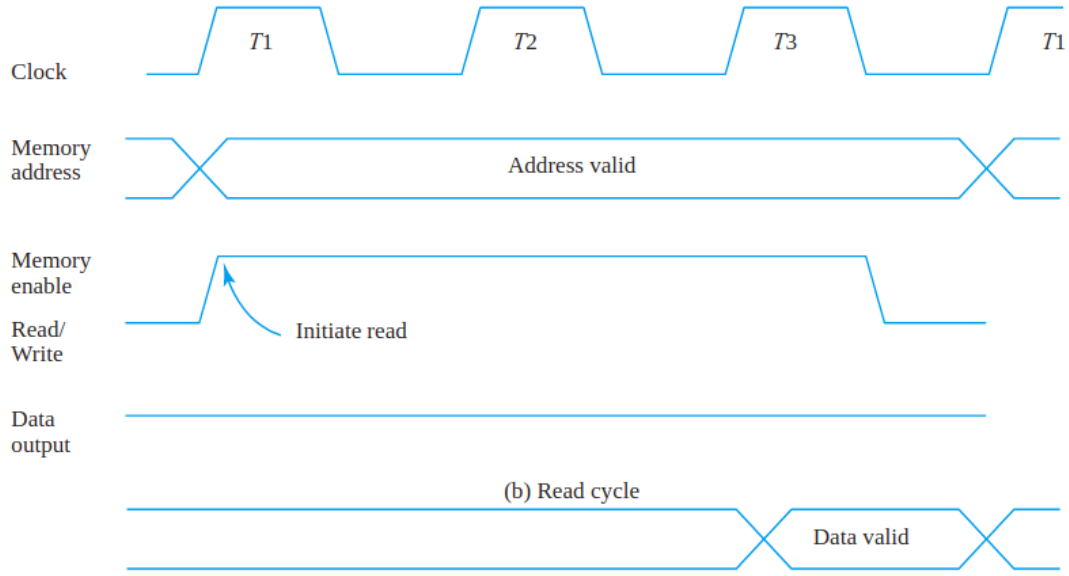


Figura 5.17: Ciclo de leitura de sRAMs, ROMs e FLASH NOR.

Na Figura 5.18 é apresentado o diagrama de tempo do ciclo de escrita das sRAMs, ROMs e FLASH NOR. Note que agora o sinal R//W está no nível lógico 0 e os dados devem estar disponíveis nos pinos de dados, como os endereços nos pinos de endereços. Estes dados devem ser mantidos nos pinos até que sejam transferidos, em inglês *latched,* para os registradores internos da memória.

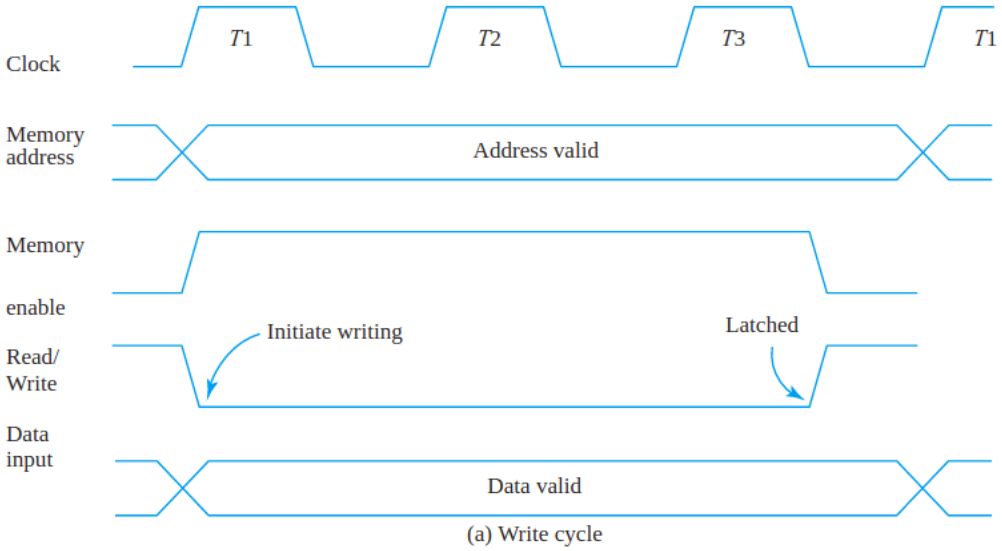


Figura 5.18: Ciclo de escrita de sRAMs, ROMs e FLASH NOR.

Nas Figuras 5.17 e 5.18 as operações das memórias são sincronizadas com um sinal de relógio (Clock). Quando as operações são assíncronas, o sinal de relógio é substituído pelo sinal de *address strobe* (AS) para indicar a disponibilidade de endereços válidos no barramento.

Não foram incluídas as temporizações dos sinais de programação das memórias EPROM, a tensão externa maior na porta de controle V_{PP} e o sinal de habilitação de programação /PGM.

5.2.2 Memórias Dinâmicas

Vimos na Seção 5.1.2.2 que as DRAMs assíncronas tem os seus endereços multiplexados em linhas e colunas. Figura 5.10 ilustra um diagrama de tempo típico de uma memória DRAM. A disposição de uma linha e de uma coluna na entrada de um módulo de memória é sinalizada pelos sinais /RAS e /CAS. Na borda de descida de /RAS e de /CAS são transferidos, respectivamente, o endereço da linha e o da coluna para a memória. O sinal de habilitação de escrita, em inglês *Write Enable* (/WE), em “1”, indica que o modo de operação é de leitura. Portanto, o *buffer* de saída é habilitado (/OE = 0) para transferir os dados válidos aos pinos Dout. Os valores dos tempos t_*, como t_{RAS} (largura do pulso /RAS) e t_{RC} (tempo do ciclo de acesso aleatório), são especificados nas folhas técnicas de cada módulo de memória. Eles correspondem às **restrições temporais**, usualmente mínimas, típicas e máximas que os fabricantes recomendam para que o módulo opere corretamente.

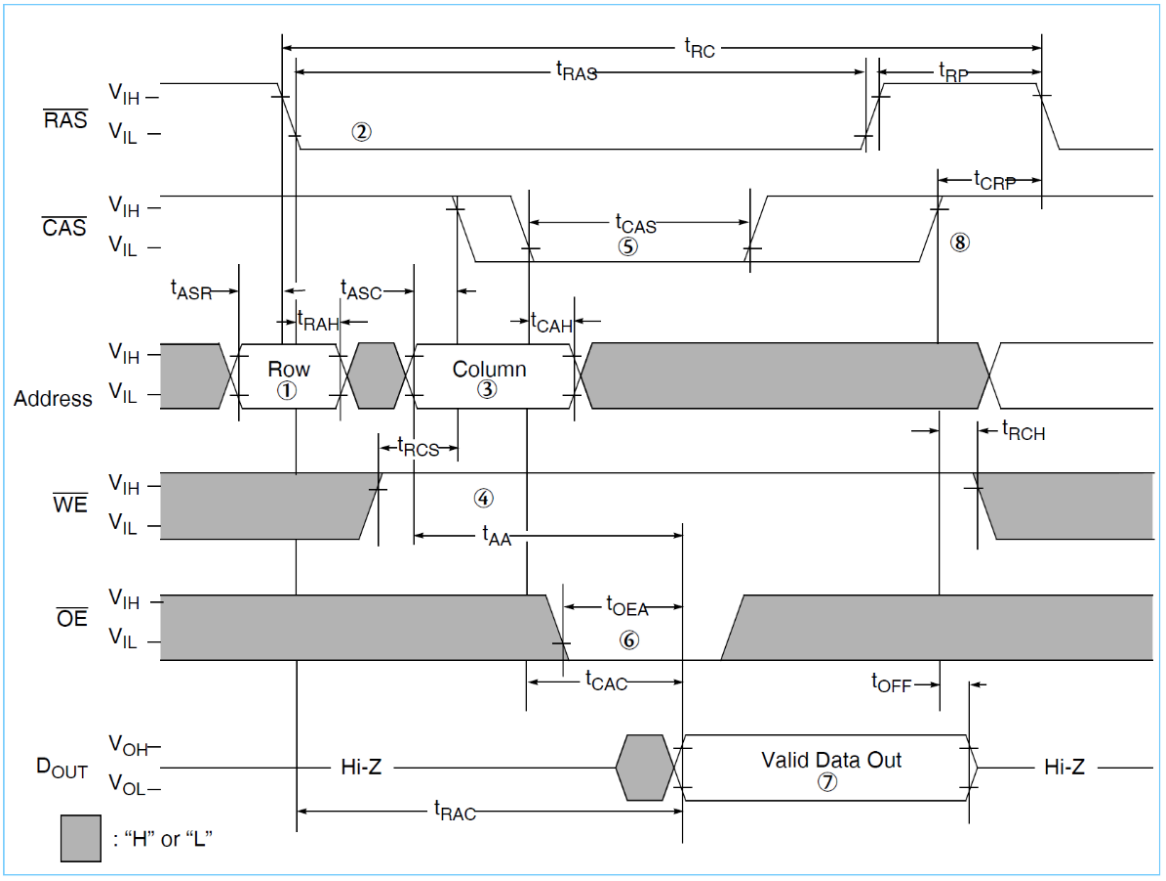


Figura 5.19: Ciclo de leitura típico de uma DRAM.

Figura 5.20 mostra as interdependências dos tempos entre vários sinais associados a uma memória DRAM no ciclo de acesso de escrita. Como no ciclo de acesso de leitura, a linha e a coluna são transferidas para a memória nas bordas de descida de /RAS e /CAS, respectivamente. Sendo um ciclo de escrita, o sinal /WE deve estar em “0” e os dados a serem gravados devem estar disponíveis antes de completar a transferência do endereço.

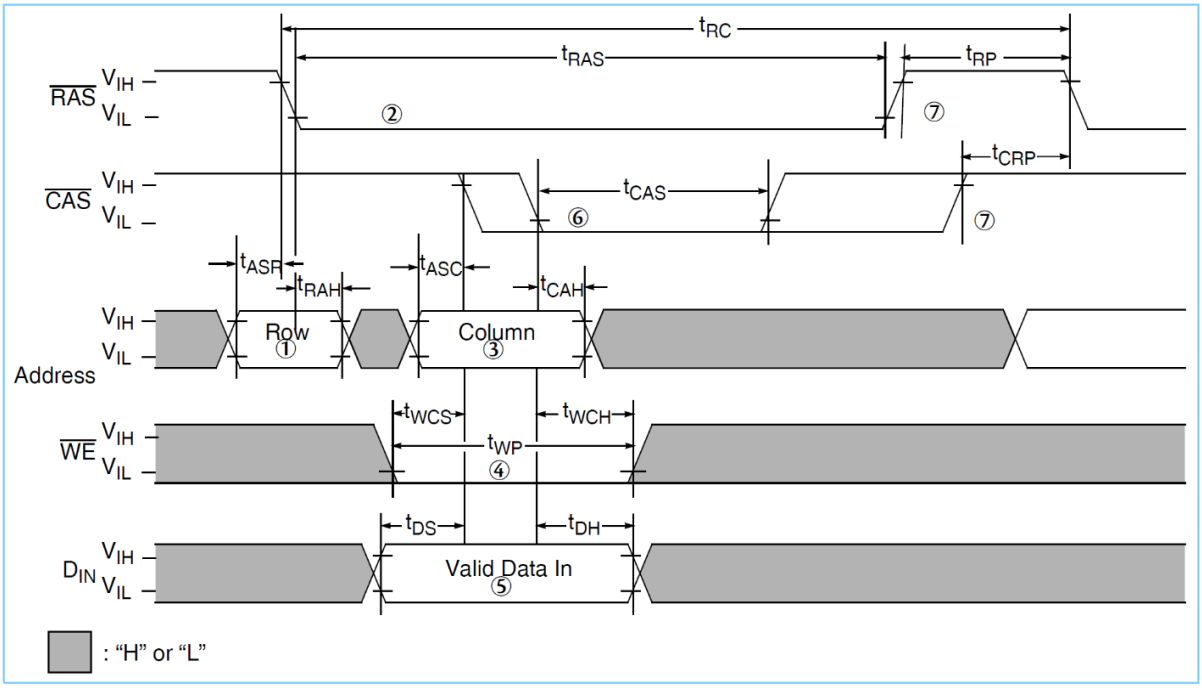


Figura 5.20: Ciclo de escrita típico de uma DRAM.

Embora os modos de acesso por página, EDO e *read-modify-write* tenham cedido espaço para o modo síncrono SDRAM e DDR SDRAM, decidimos apresentar os seus diagramas típicos de tempo para exercitar a leitura das relações temporais dos sinais. Na Figura 5.21 é mostrado o diagrama de tempo do modo de acesso de leitura (/WE=1) por página, em que a linha é carregada uma vez (uma borda de descida em /RAS) e uma sequência de colunas carregadas (várias bordas de descida em /CAS). Para cada combinação (*Row*,*Column n*), é habilitado o *buffer* de saída (/OE=0) e o dado correspondente D_{OUT} é colocado nos pinos de dados.

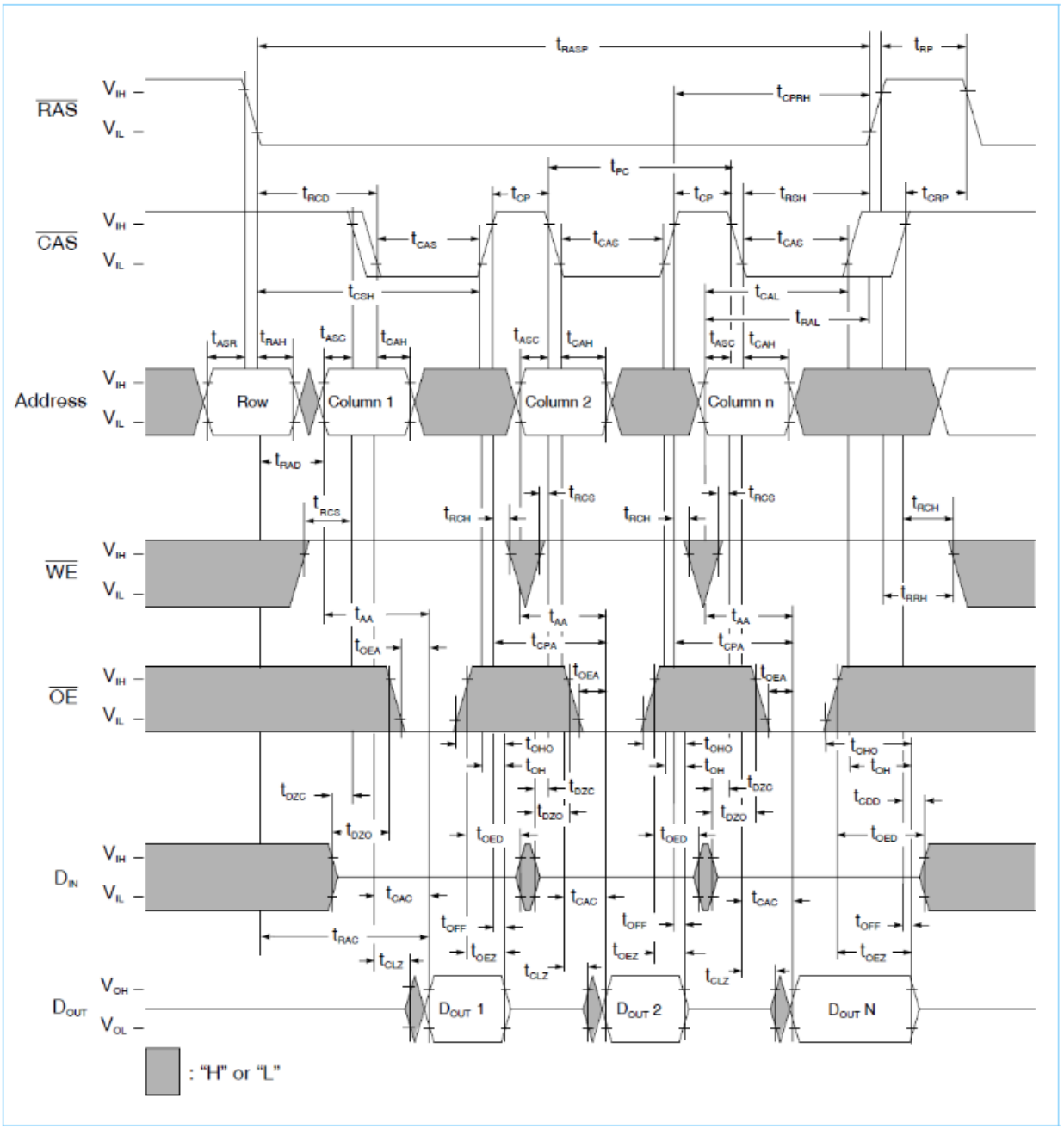


Figura 5.21: Modo de acesso Página.

Comparando a Figura 5.21 e a Figura 5.22, pode-se observar que o modo de acesso EDO é bem similar ao modo de acesso página. A única diferença está na transição do sinal /CAS antes da finalização do acesso anterior, permanecendo o sinal de habilitação do *buffer* de saída sempre habilitado (/OE=0).

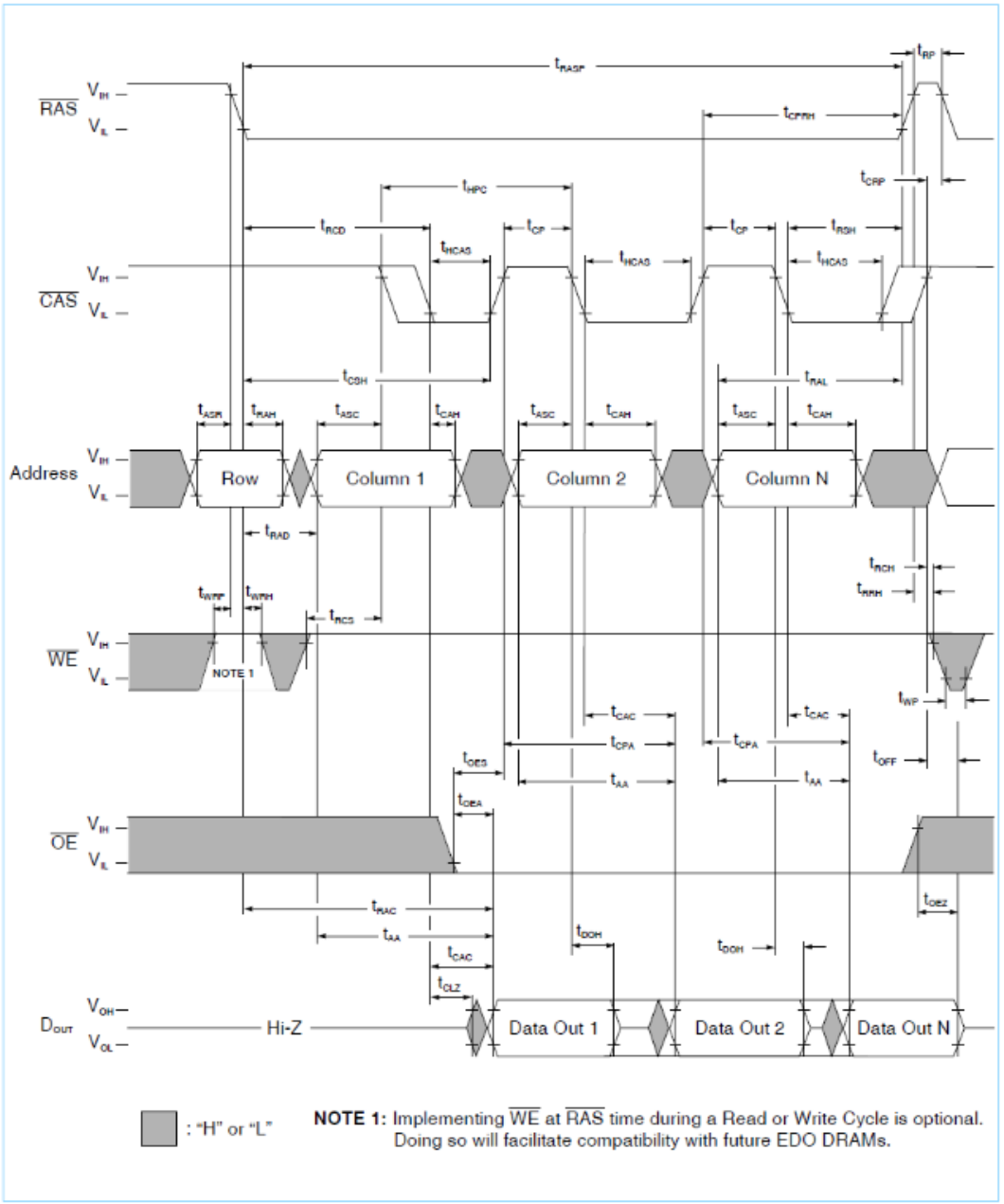


Figura 5.22: Modo de acesso EDO.

No modo de acesso *read-modify-write*, a Figura 5.23 mostra que, uma vez decodificado o endereço da memória (uma borda de descida em /RAS e uma borda de descida em /CAS), é feito um acesso de leitura com os dados nos pinos D_{OUT}, seguido de um acesso de escrita com os dados D_{IN} nos mesmos pinos. Neste caso, chamamos atenção para o controle da multiplexação dos pinos de dados entre dois tipos de dados.

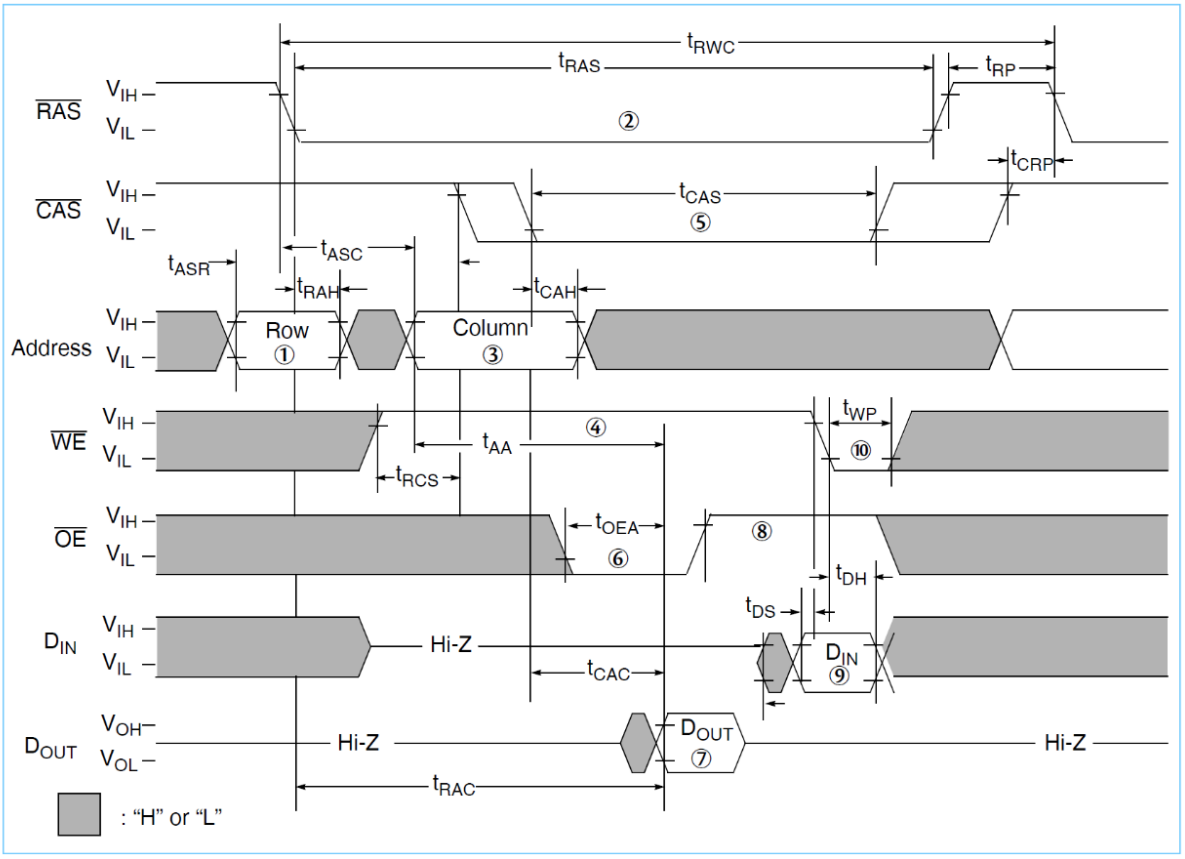


Figura 5.23 : Modo de acesso *read-modify-write*.

Figura 5.24 representa a relação temporal dos sinais /RAS e /CAS para iniciar uma regeneração. Deve-se gerar uma borda de descida em /CAS e aguardar no mínimo um intervalo correspondente a t_{CSR} (tempo de setup do CAS) para gerar uma borda de descida em /RAS.

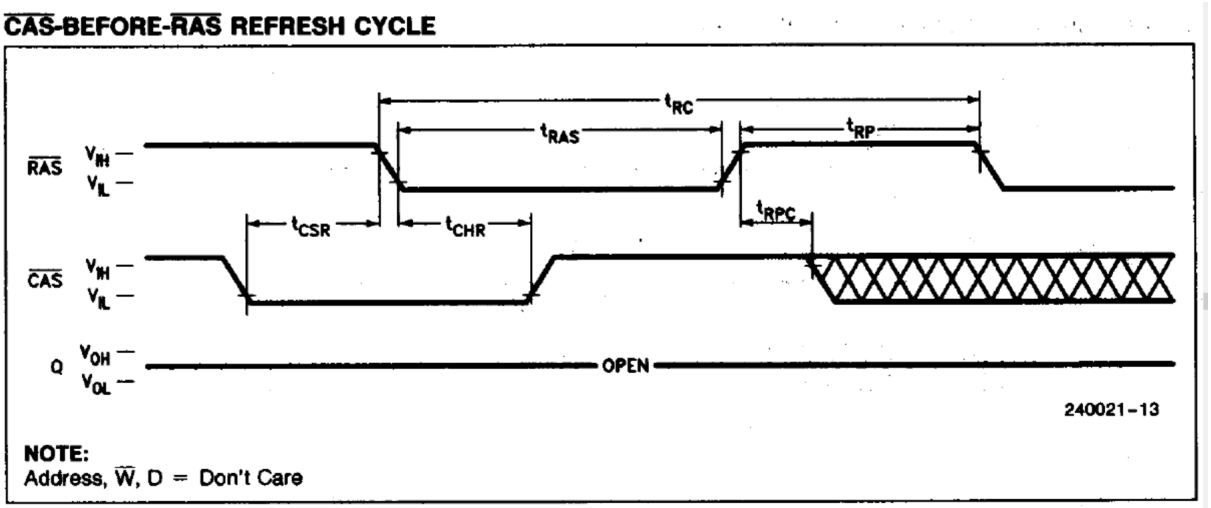


Figura 5.24: Regeneração CAS-antes-RAS.

Finalmente, Figura 5.25 mostra a regeneração transparente, enquanto os dados válidos (sinal D0) são mantidos nos pinos de saída.

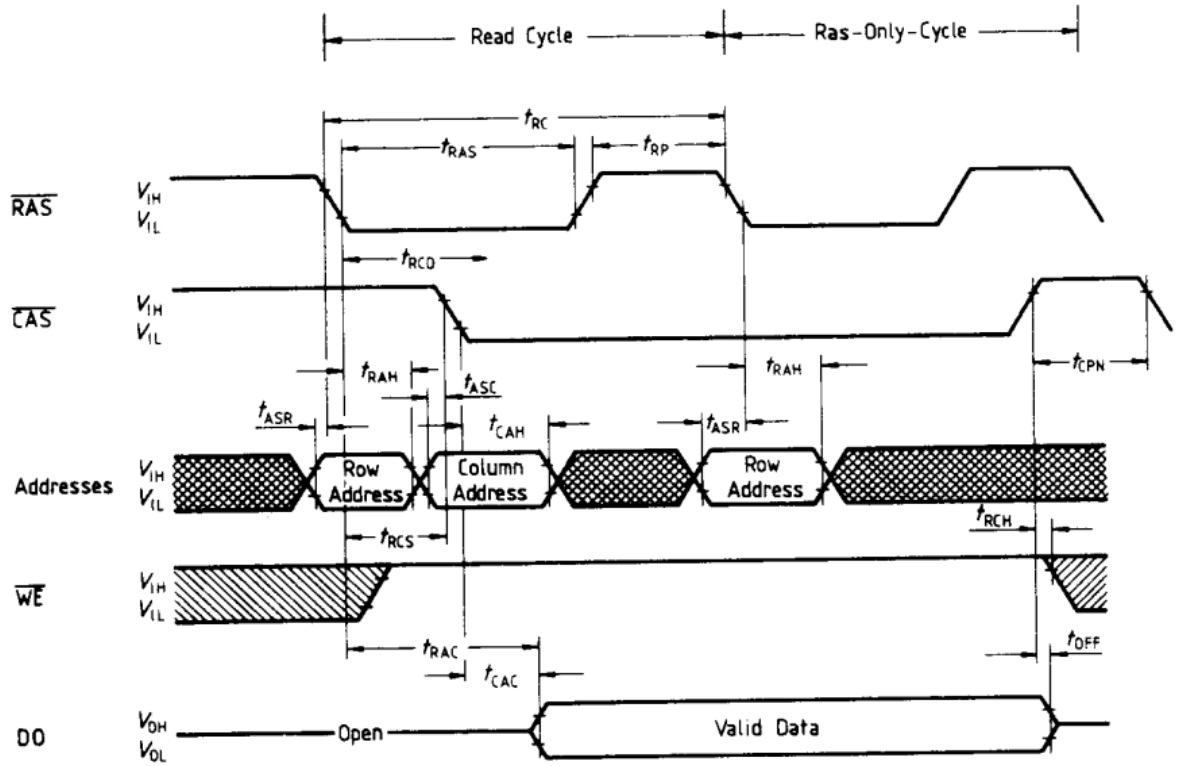


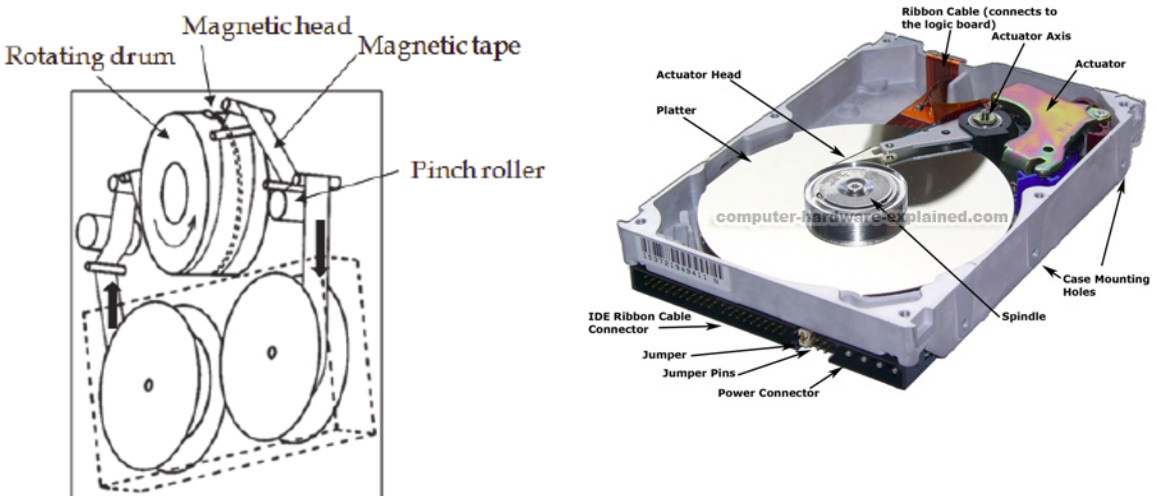
Figura 5.25: Regeneração transparente.

5.3 Tecnologias em Memórias Secundárias

As memórias de armazenamento em massa são não-voláteis, mas não são acessadas como “memória” nas arquiteturas apresentadas na Figura 4.7. Elas são consideradas como dispositivos de entrada (*input*) e saída (*output*), acessíveis através da ponte sul[1] (Seção 4.7). Mesmo assim, decidimos incluí-las neste capítulo por se tratar de dispositivos providos da função de armazenamento de um grande volume de dados, relevante a organização de um sistema de memória que detalharemos no Capítulo 4. É importante, porém, frisar que, sob o ponto de vista de projetos de interface com a CPU tanto em *hardware* quanto em *software*, as memórias secundárias devem ser tratadas como dispositivos E/S interfaceadas por **controladores** (*drives*) gerenciados por um sistema operacional (Seção 4.8.1). Para projetos de sistemas embarcados diretamente sobre *firmwares* ou *bootloader*, a programação desta interface pode ser trabalhosa. Uma compreensão melhor do funcionamento destes dispositivos pode ajudar nesta tarefa quando necessária.

[1] A partir da série 5 da Intel, a arquitetura de *hub* da Intel, em inglês *Intel Hub Architecture* (IHA), foi substituída pelo *Hub* de Controlador de Plataforma, em inglês *Platform Controller Hub* (PHC). Muitas funções da ponte norte, como conexões com a memória principal, foram integradas na CPU.

Diferentes da maioria das memórias principais, os acessos aos dados das memórias secundárias não são aleatórios, em inglês *random access*. Os dados são acessados **sequencial**mente, por blocos, à medida que o cabeçote desloca continuamente sobre a trilha onde eles são gravados, como é numa fita magnética ilustrada na Figura 5.26. No caso dos discos, o tempo de acesso pode ser melhorado se associarmos a cada trilha do disco um cabeçote de forma que o modo de **acesso** às trilhas possa ser **direto** (Figura 5.26).



(a) fita magnética (b) disco rígido

Fig. 4.26: Memórias secundárias (de massa)

5.3.1 Discos Magnéticos

O primeiro disco rígido foi construído pela IBM em 1956. Era constituído de 50 discos magnéticos contendo 50.000 setores, sendo que cada um suportava 100 caracteres alfanuméricos, totalizando uma capacidade de 5 *megabytes* [21].

Os discos magnéticos consistem de um conjunto de pratos giratórios e um conjunto de braços mecânicos, cada um provido de um cabeçote de leitura e gravação numa das suas extremidades. Figura 5.26.(b) ilustra os componentes de um disco rígido, em inglês *harddisk* (HD). O braço consegue se mover sobre o prato posicionando o cabeçote sobre diferentes pontos da superfície do prato (primeiro sobre a trilha e depois sobre o setor da Figura 5.27). O prato, por sua vez, é coberto por uma camada magnética extremamente fina, e o cabeçote, um eletroimã bem pequeno e preciso cobrindo uma área menos de centésimos de milímetros de diâmetro.

Através do campo criado por este cabeçote, as moléculas de óxido de ferro da superfície do prato se alinham conforme a polaridade do campo. Alternando direção da corrente do eletroimã, podemos criar dois campos opostos de alinhamento das

moléculas. Aos dois estados de alinhamento são associados os dois níveis lógicos “0” e “1”. Portanto, cada *bit* corresponde a um conjunto de moléculas alinhadas sob o cabeçote móvel em discos magnéticos (canto superior da Figura 5.27). Quanto maior for a densidade do disco, menor deverá a área do cabeçote e a quantidade de moléculas usadas para gravar um *bit*.

Fisicamente, os fabricantes formatam os discos em **trilhas**, **setores** e **cilindros**. Os setores são as menores unidades de acesso. Hoje em dia o tamanho típico de um setor é 512 *bytes*. As trilhas são faixas circulares concêntricas sobre um prato. Elas são divididas em setores. E quando há mais de um disco empilhado no eixo, chamamos de cilindro um conjunto de trilhas concêntricas de mesmo raio. A quantidade de cabeçotes é igual ao número de trilhas por cilindro. Na Figura 5.27 temos 3 trilhas por cilindro.

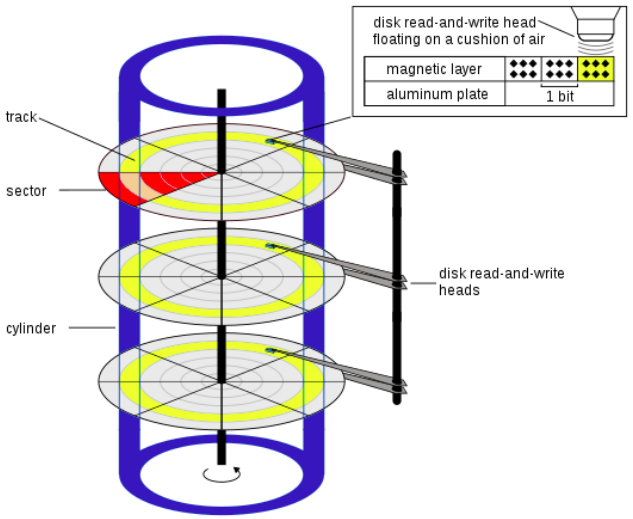


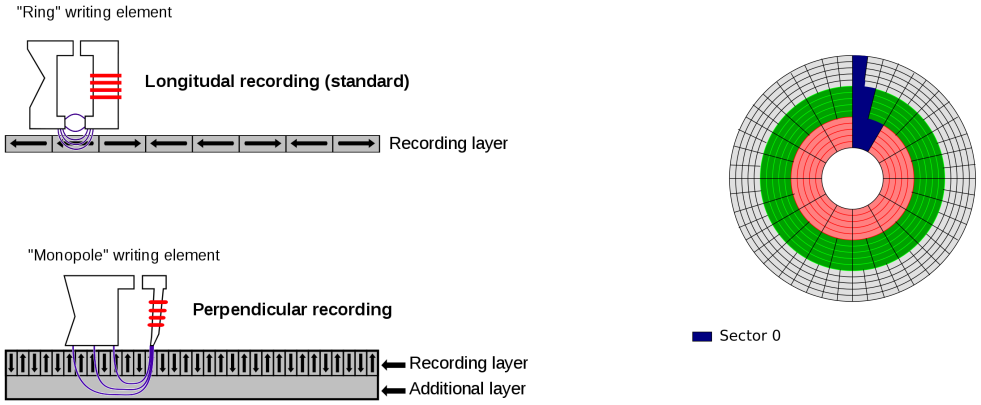
Figura 5.27: Formatação física dos discos: cilindros, trilhas e setores.

Para gravar uma sequência de *bits* 0 e 1, a polaridade do cabeçote é alternada à medida que o prato gira sob o cabeçote. Dependendo da variação dos *bits*, a polaridade do cabeçote pode alternar algumas milhões de vezes por segundo. E para ler os dados gravados, o cabeçote capta o campo magnético gerado pelas moléculas alinhadas enquanto o prato gira. Através das pequenas correntes induzidas nas bobinas, o controlador integrado no disco rígido consegue codificar as correntes medidas em *bits* 0 e 1 e disponibilizar os *bits* codificados na sua saída.

Os acessos são feitos por setores. Ao invés de usar a sequência linear de 0 até 2^n endereços (Seção 5.2.1), usava-se inicialmente a técnica **cilindro-cabeçote-setor**, em inglês *Cylinder-head-sector* (**CHS**), para endereçar os setores. Esta técnica exigia que seja conhecida a posição física dos setores. Nos discos modernos, o esquema de **endereçamento de bloco lógico**, em inglês *Logical block addressing*

(**LBA**), é mais aplicado para especificar de forma sequencial os blocos de dados nos *storages*.

Divisões otimizadas, como **gravação de *bits* por zona**, em inglês *zone bit recording* (ZBR), tem sido exploradas para aumentar a capacidade dos discos. Figura 5.28 mostra como o tamanho do setor 0, em azul, pode variar conforme a distância da sua trilha em relação ao eixo de rotação [23].



(a) Gravação magnética (b) ZBR

Figura 5.28: Gravação por setor.

Controladores microcontrolados tem sido integrados aos discos rígidos modernos para simplificar a interface dos discos magnéticos sob o ponto de vista de programação. Figura 5.29 ilustra um HD com um microcontrolador PIC integrado nele. Vale comentar que o primeiro setor do HD é reservado para armazenar o endereço do sistema operacional, de forma que, quando um computador é ligado, o seu HD é inicializado pelo POST (Seção 4.6) e o BIOS consegue localizar o endereço do sistema operacional acessando o HD e inicializar o seu carregamento na memória principal.

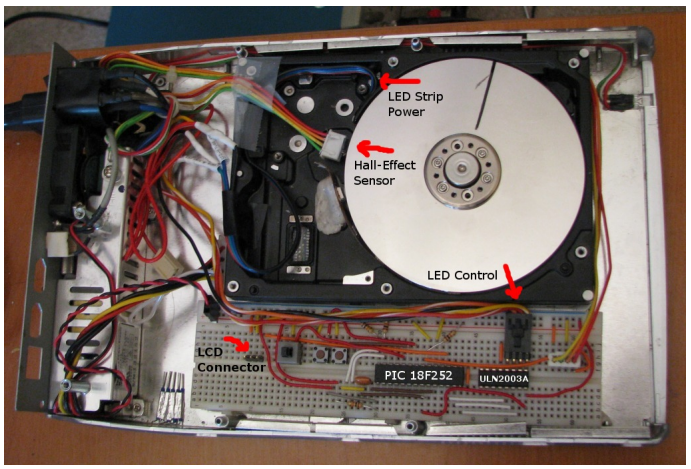


Figura 5.29: HD com controle microcontrolado.

A capacidade de um disco rígido, atualmente disponível no mercado para uso doméstico/comercial, varia de 80 a 8000 GB, assim como aqueles disponíveis para empresas e grandes servidores, de mais de 8 TB. No entanto, **há uma discrepância entre a capacidade física reconhecida pelo sistema operacional e a capacidade fornecida pelos fabricantes**. As indústrias adotam o Sistema Internacional de Unidades nas suas especificações. Este sistema trabalha com potências e dez e no mundo binário dos sistemas digitais usa-se potências de dois [21].

Visando a aumentar o desempenho, a segurança dos dados e a facilidade na recuperação de conteúdo perdido, foi proposto em 1988 por D. A. Petterson e colegas [24] um sistema de armazenamento composto por vários discos individuais com redundância de dados, de maneira que, em caso de falhas em um disco, os demais vão continuar em funcionamento. Isso evita perda de dados e interrupções no acesso aos dados. Este sistema é conhecido por **arranjos redundantes de discos independentes**, em inglês *redundant array of independent disks* (**RAID**). Hoje em dia existem várias organizações de RAID, cada uma tem uma finalidade. Alguns são mais apropriados para desempenho e outros para segurança [25]:

- RAID 0: os dados são divididos em pequenos segmentos e distribuídos pelos discos conforme a sua frequência de acessos. Isso pode aumentar o desempenho.
- RAID 1: caracteriza-se pelo espelhamento de dados de um disco em outro.
- RAID 2: integra o mecanismo de detector de erros nos discos. Muitos discos modernos já vem com este mecanismo integrado.
- RAID 3: além dos dados originais são armazenados os dados de paridade (Seção 5.1.2.2) para controle de erros.
- RAID 4: os dados são distribuídos entre os discos com dados de paridade centrados num disco dedicado.
- RAID 5: os dados de paridade são distribuídos entres os discos, ao invés de serem armazenados num disco dedicado.
- RAID 6: semelhante a RAID 5, porém usa o dobro de *bits* de paridade, garantindo a integridade dos dados caso até 2 dos HDs falhem ao mesmo tempo.

Para explorar melhor as características que cada organização de RAID oferece, muitos sistemas de armazenamento combinam as estratégias de dois RAIDs, como RAID 01 (RAID 0+1), RAID 10 (RAID 1+0), RAID 50 (RAID 5+0) e RAID 100 (RAID 10+0). Figura 5.30 ilustra a organização RAID 100 para um sistema composto de 8 discos.

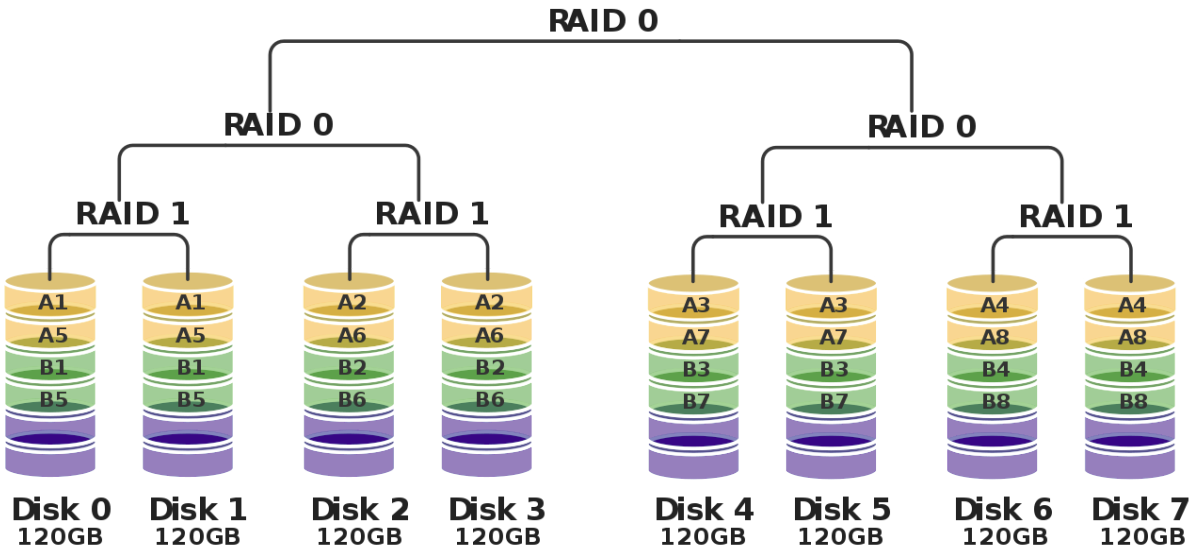
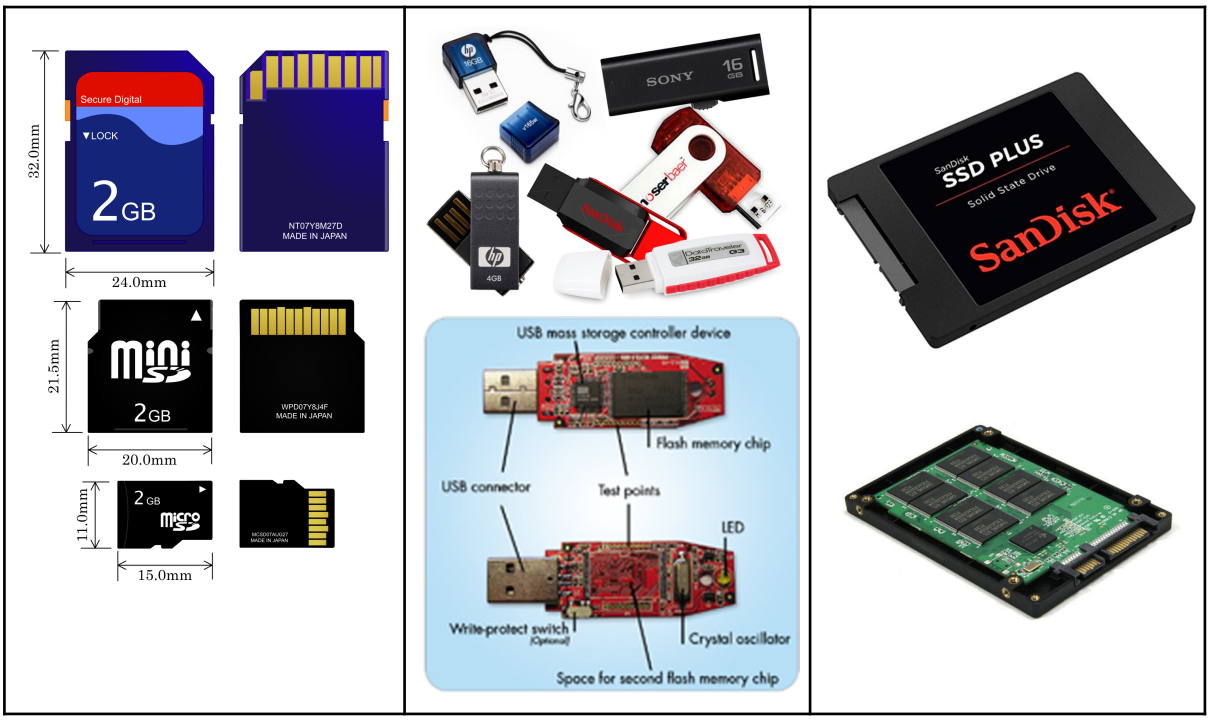


Figura 5.30: RAID 100

5.3.2 Discos de Tecnologia FLASH NAND

Os discos magnéticos rígidos (HD) têm sido usados predominantemente como dispositivos de armazenamento de massa em computadores pessoais. No entanto, como vimos na Seção 5.1.1, com a evolução da tecnologia de memórias FLASH NAND, estas tornaram competitivas e vem ganhando cada vez mais o espaço.

Comercialmente, as memórias FLASH NAND são encontradas num dos seguintes formatos (Figura 5.31): cartões de memória, em inglês *Secure Digital* (SD) *card*, *pendrives* ou *flashdrives*, e discos de estado sólido, em inglês *state-storage drive*.



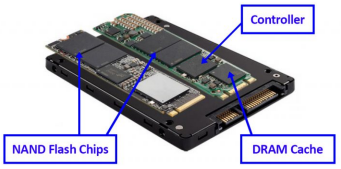
		
(a) cartão de memória	(b) *pendrives* [46]	(c) SSD [47]

Figura 5.31: Dispositivos de armazenamento de massa baseada na tecnologia FLASH NAND.

5.3.3 Discos Ópticos

Os discos ópticos são ainda usados como mídias de gravação. Os mais conhecidos são os *Compact Discs* (CDs) e *Digital Versatile Discs* (DVDs). São usualmente feitos de 4 (somente de leitura) a 6 camadas (regraváveis):

- camada de rótulo, podendo ser de papel ou impresso.
- camada de plástico com função protetora
- camada reflexiva com altos e baixos relevos (mídias somente de leitura)/camada reflexiva dom relevo plano (mídias regraváveis)
- camada dielétrica para dissipação do calor do laser durante a gravação (mídias regraváveis)
- camada gravável-regravável (mídias regraváveis)
- camada de policarbonato.

A gravação destes dispositivos se baseia no princípio de alterar a refletividade da superfície, de maneira que seja possível distinguir dois estados lógicos “0” e “1”. Na leitura, um raio laser é disparado perpendicularmente ao disco. Este raio é refletido de volta para o leitor e as variações em alto e baixo relevo (mídias somente de leitura) ou pontos transparentes ou opacos (mídias regraváveis) permitem que o controlador do dispositivo detecte sequência de 0 e 1 gravado, como ilustra a Figura 5.32.

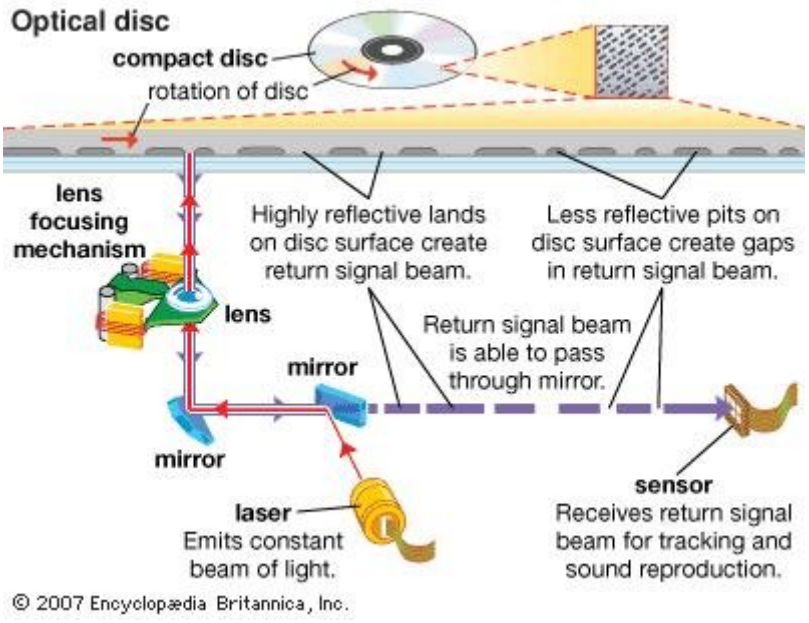


Figura 5.32: Leitura de um disco óptico.

5.4 Interfaces das Memórias Secundárias

Os dispositivos de armazenamento de massa são disponíveis comercialmente como módulos completamente independentes. Na maioria dos projetos, só precisamos entender as interfaces desses dispositivos e o *chipset* da placa-mãe em que eles serão conectados para projetar as conexões.

Embora a arquitetura de *chipset* tenha evoluída da IHA (Figura 4.22) para PCH (Figura 5.33), mantém-se o mesmo o conceito de separar as conexões paralelas rápidas (entre a CPU e a memória principal) e as conexões, muitas vezes seriais, mais lentas (entre a CPU e os periféricos de entrada/saída). Ao compararmos as Figuras 3.22 e Figura 5.33, podemos dizer que o que efetivamente alterou foi uma reorganização das funções da ponte norte, distribuindo-as entre a CPU e PCH para melhorar a largura da banda. A fim de conectarmos corretamente os dispositivos de interesse, precisamos ter uma noção dos padrões de controladores e de interface suportados pelos *chipsets*. É interessante observar que entre as características almejadas estão a velocidade e a confiabilidade na transferência. E os esforços acadêmicos e industriais resultaram numa série de padrões de interfaces.

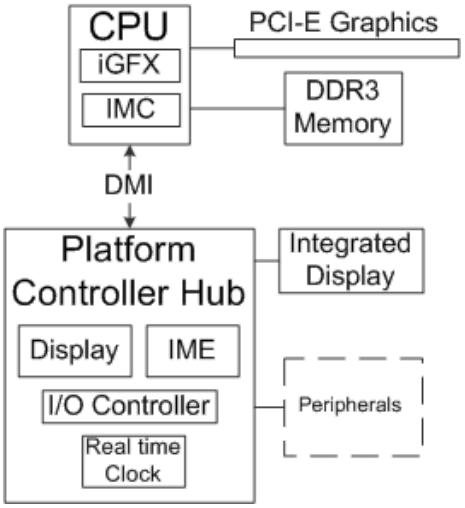


Figura 5.33: *Hub* de Controlador de Platafoma (PCH) (Fonte: [26]).

Neste curso decidimos nos limitar em interfaces para as unidades de memória FLASH NAND, amplamente usadas como memórias removíveis, e as unidades de HD, presentes em todos os sistemas computacionais e usadas para expandir o espaço de endereços da memória principal através de uma unidade de gerenciamento de memória (Seção 5.6).

5.4.1 Discos Rígidos

Os seguintes tipos de interfaces são encontrados nos discos rígidos comerciais:

- **IDE** (*Integrated Drive Eletronics*)/**ATA** (*Advanced Technology Attachment*) [27]: foi o primeiro padrão de interface dos HDs, em que o controlador e o disco são integrados. Os primeiros HDs com interface IDE foram lançados por volta de 1986. Permite conectar a uma mesma interface dois discos no esquema mestre/escravo. Os cabos tem 40 fios, 16 para enviar e receber dados em paralelo, 7 terra, e 1 para seleção do modo mestre/escravo (Figura 5.34). Com a introdução de SATA em 2003, foi retroativamente renomeado para PATA (*Parallel ATA*).

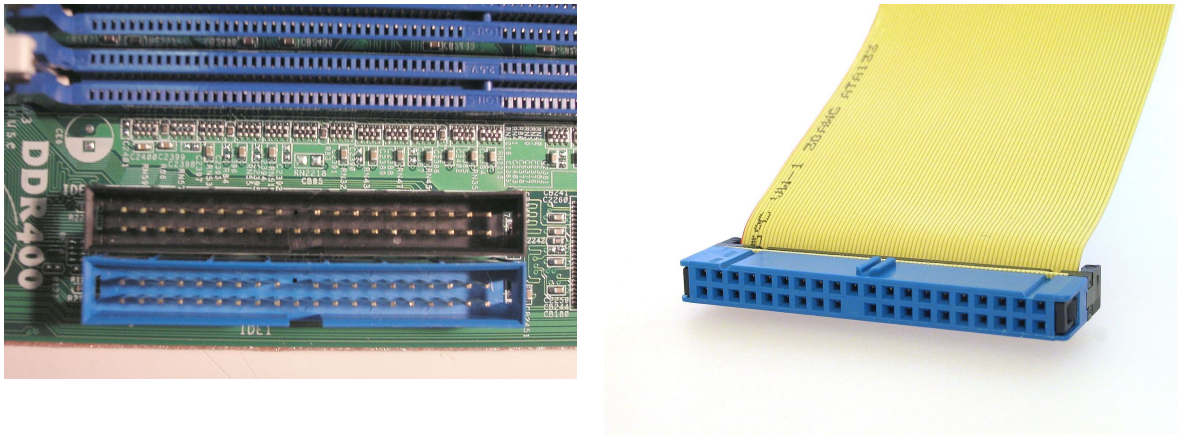


Figura 5.34: Interface ATA (Fonte: [27]).

- **SATA** (*Serial AT Attachment*) [28]: é o sucessor da tecnologia ATA, substituindo a interface paralela pela serial. Os cabos são formados por dois pares de fios (um para transmissão e outro para recepção) e mais três fios de terra, totalizando 7 fios (Figura 5.35). Usando transmissão diferencial (Figura 5.41.(b)), reduziu interferências eletromagnéticas entre os fios. Isso permitiu aumentar a velocidade de transferência, reduzir o diâmetro dos fios e melhorar a ventilação do gabinete. Reduzindo os fios, viabilizou a troca do dispositivo enquanto ligado, em inglês *hot-swap*. Há diferentes gerações da interface SATA. Elas diferem essencialmente em velocidade de transferência.

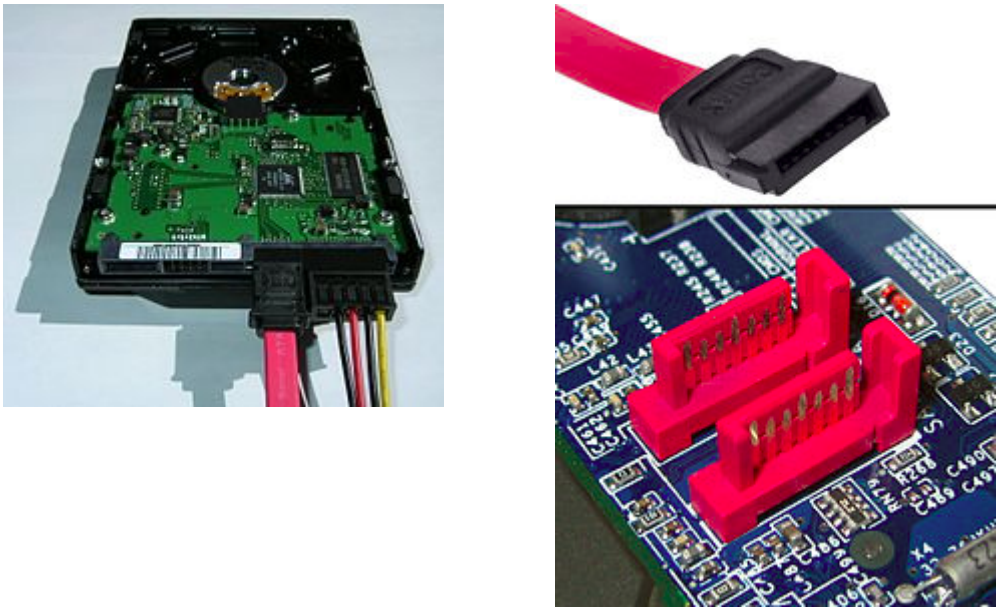
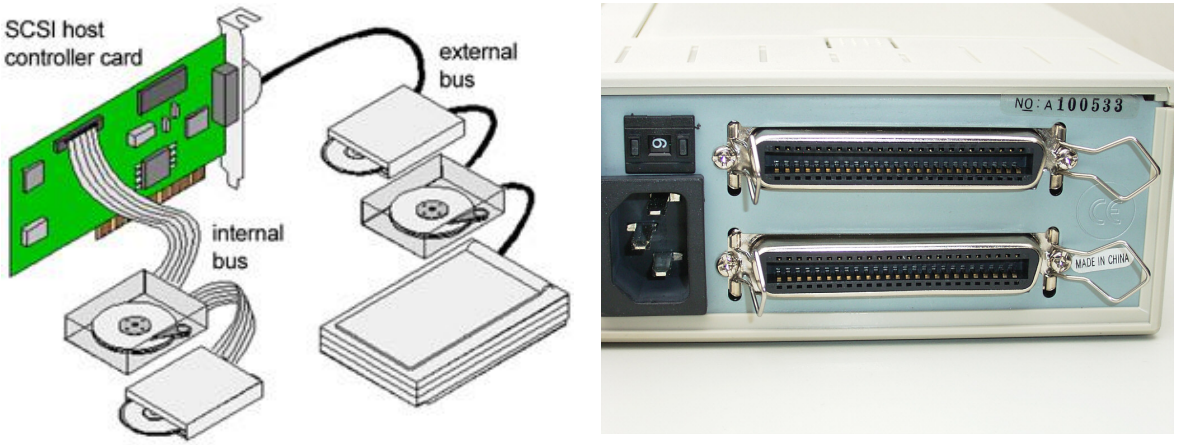


Figura 5.35: Interface SATA.

- **SCSI** (Small Computer System Interface) [29]: foi padronizada em 1986. Embora seja mais comumente encontrada como interface de discos rígidos e unidades de fita, foi originalmente concebida para conectar paralelamente

uma larga gama de periféricos. Ele permite conexões de até 15 dispositivos (Figura 5.36), identificados por um código binário, chamado ID SCSI. As transmissões ocorrem, porém, sempre entre dois dispositivos de cada vez. Diferentes versões de SCSI diferem na quantidade de fios de dados e na velocidade de transmissão.



(a) conexões de até 15 dispositivos (b) conectores SCSI

Figura 5.36: Interface SCSI (Fonte: [29]).

- **SAS** (*Serial Attached SCSI*) [30]: é a versão serial de SCSI, como mostra a Figura 5.37, com vantagens de maior velocidade na transferência de dados, *hot-swapping* e melhor controle dos erros na transmissão.

(a) conectores SASI (b) *Storage* com discos SASI

Figura 5.37: Interface SASI (Fonte: [30]).

- **FC** (*Fiber Channel*) [31]: é o sucessor serial de SCSI em mercados corporativos. Foi aprovado pela ANSI em 1994. Adotando fibras ópticas

como cabos de conexão[2], é um protocolo sem perda e de altíssima taxa de transferência (na ordem de *gigabits* por segundo). Portanto, é tipicamente usado para conexão dos discos de *storage* com os servidores nas **redes de áreas de storage**, em inglês *storage area networks* (**SAN**) em centros de dados comerciais. Figura 5.38 ilustra uma interface FC.

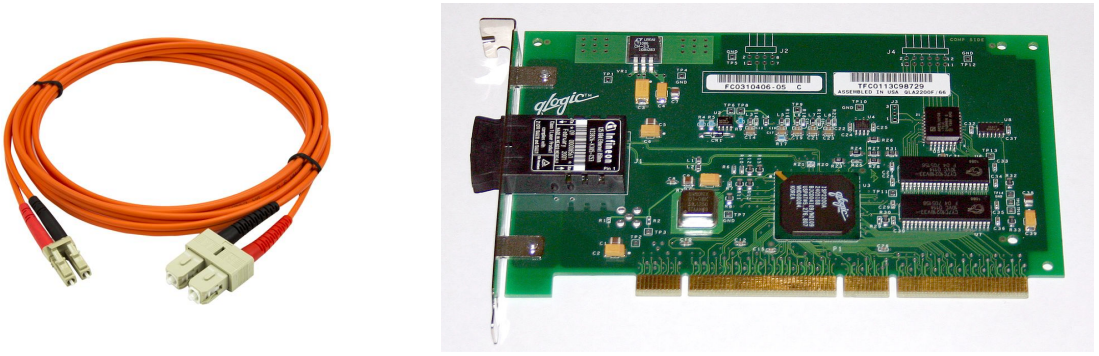


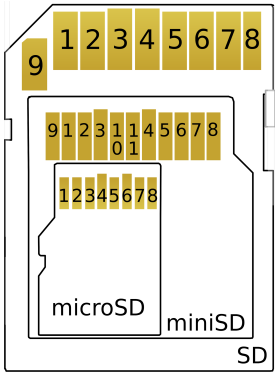
Figura 5.38: Interface FC (Fonte: [31]).

5.4.2 Flash NAND

Na seção 5.1.1 apresentamos alguns dispositivos baseados em tecnologia FLASH NAND e na seção 5.3.2 comentamos que eles estão sendo cada vez mais competitivos em relação aos discos rígidos. Pois, eles têm a vantagem de não precisarem de um cabeçote (parte móvel mecânico) para fazer as leituras e as gravações. Por isso, são cada vez mais considerados nos projeto de sistemas embarcados.

Em termos de interfaces, os cartões de memória diferem muito dos *pendrives*. Os cartões de memória utilizam pinos de contato para se conectarem com um sistema computacional, enquanto os *pendrives* utilizam o conector *Universal Serial Board* (USB). Na Figura 5.39. (a) são apresentados os três tamanhos típicos dos cartões de memória SD e nas Figuras 5.39.(b) e 5.39.(c) as suas respectivas pinagens. Observe que todos os cartões possuem uma interface nativa (coluna SD) e uma interface de comunicação de protocolo SPI (coluna SPI). No Capítulo 9 é apresentado em maiores detalhes o protocolo de comunicação SPI, muito difundido em sistemas embarcados.

[2] Há implementações com cabos de cobre.



PIN	SD	SPI
1	CD/DAT3	CS
2	CMD	D1
3	VSS1	VSS1
4	VDD	VDD
5	CLK	SCLK
6	VSS2	VSS2
7	DAT0	D0
8	DAT1	X
9	DAT2	X

PIN	SD	SPI
1	DAT2	X
2	CD/DAT3	CS
3	CMD	DI
4	VDD	VDD
5	CLK	SCLK
6	VSS	VSS
7	DAT0	DO
8	DAT1	X

(a) tamanhos (b) pinagem de SD e mini (c) pinagem de micro

Figura 5.39: Interfaces de cartões de memória (Fonte: [44])

O conector típico de um *pendrive* é o conector USB. As conexões USB são do padrão Plugar e Usar, em inglês *Plug and Play* (PnP), facilitando enormemente a sua instalação. Com o objetivo de padronizar as conexões de uma grande variedade de periféricos, o padrão industrial da USB foi desenvolvido em 1995, mas só começou a ser usado nos computadores de 1997. Desde então várias versões foram lançadas, aperfeiçoando tanto na potência quanto na taxa de transferência. Por exemplo, velocidade de transmissão variou de 1.5 M*bits*/s (versão 1.0) até 10G*bits*/s (versão 3.0). Para acompanhar esta variação das taxas de transmissão, os conectores USB têm evoluídos também. Distinguem-se 5 tipos de conectores: tipo-A (12mmx4.5mm), tipo-B (8.45mmx7.36mm), tipo-C (6.8mmx2.4mm), micro e mini (6.8mmx1.8-3mm). O tipo-C é o formato mais moderno, simétrico e reversível. Figura 5.40 sintetiza os diferentes formatos de conectores para diferentes versões de USB.

(a) 1.0, tipo A e tipo B

(b) 2.0, tipo A

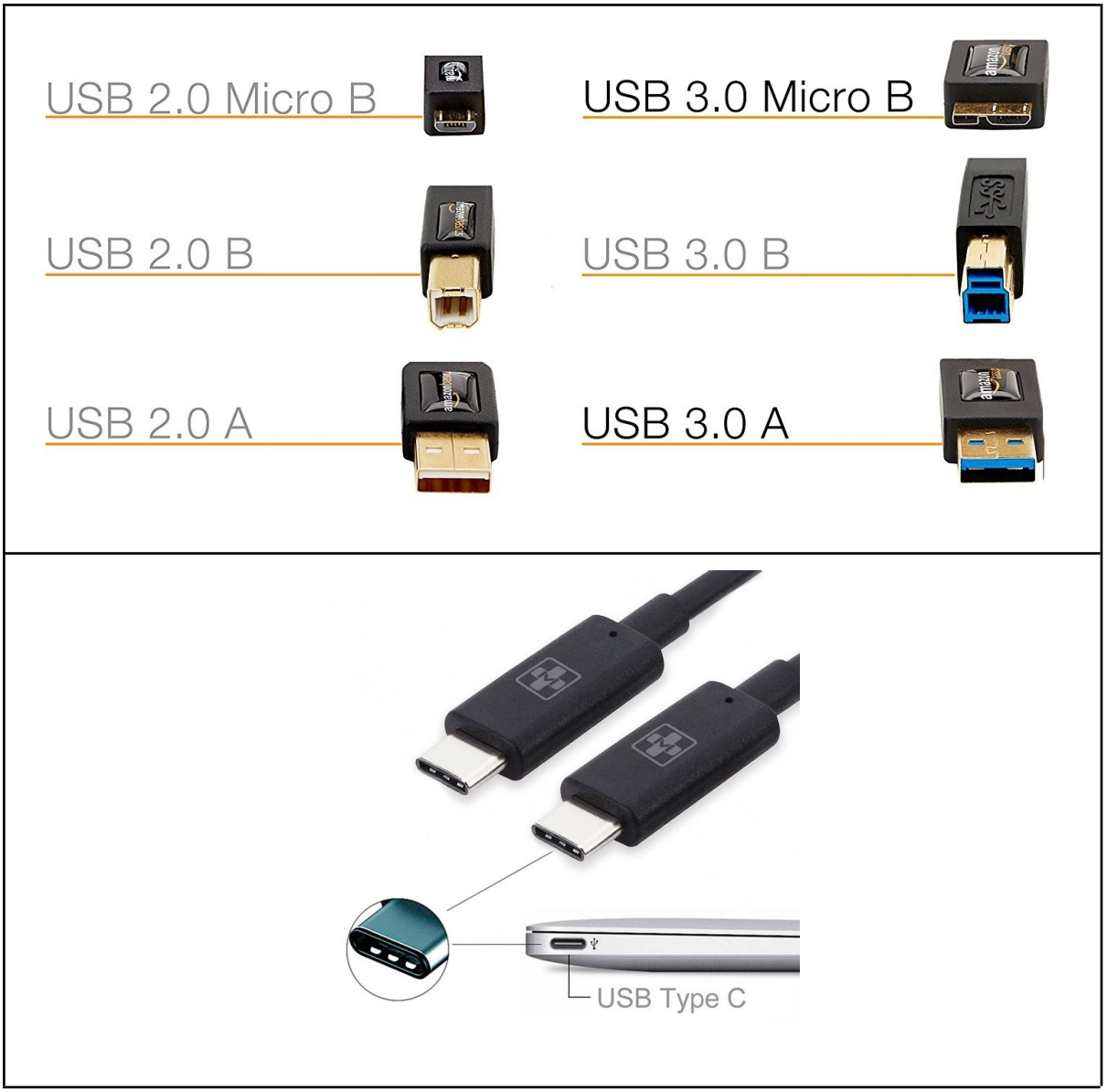
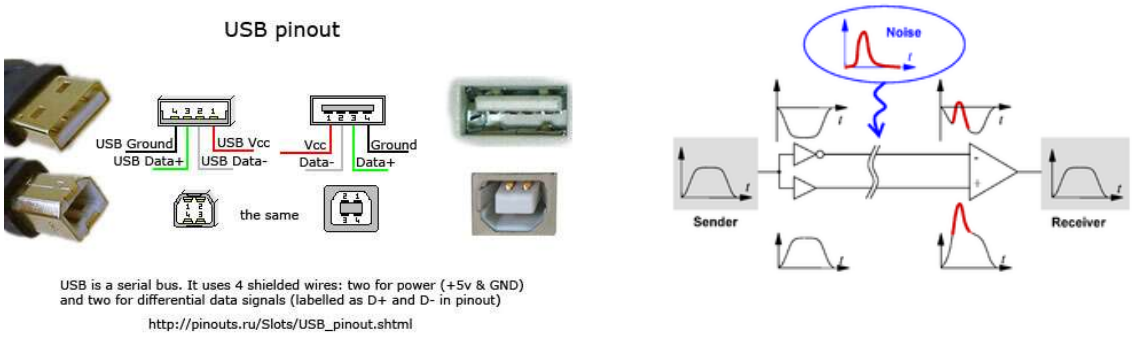


Figura 5.40: Interface USB (Fonte: [45]).

Uma conexão USB é formada por 4 fios: 2 de alimentação e 1 par diferencial para dados. A alimentação típica da USB é 5V em corrente contínua, e a corrente acionável pelos pinos varia de 0,5A (USB 1.0) até 0.9A (USB 3.0). Figura 5.41.(b) ilustra a técnica de transmissão de sinais diferenciais. Esta técnica consiste em enviar as versões complementares de um mesmo sinal com a finalidade de identificar potenciais ruídos ao longo da transmissão e removê-los no receptor.



(a) Pinagem (b) Sinais diferenciais

Figura 5.41: Pinagem da USB (Fonte: [46]).

Os discos de estado sólido (SSD) diferem dos cartões e dos *pendrives* principalmente na velocidade de transferência. Em princípio, todos estes dispositivos são baseados na mesma tecnologia FLASH NAND. No entanto, pelas dimensões disponíveis nos cartões SD e nos *pendrives*, os fabricantes não conseguem integrar às pastilhas de memória circuitos complexos que otimizem as transferências dos dados como nos discos SSD. Em cada SSD, há um controlador integrado, indicado pela seta azul na Figura 5.38.(c), que procura balancear o uso das células da memória melhorando a organização dos dados armazenados. Às vezes é integrada a memória *cache*, também destacada pela seta azul na Figura 5.38.(c), para aprimorar o desempenho dos seus acessos. Como estes discos têm dimensões bem menores do que os discos rígidos, é mais fácil acomodá-los próximos do barramento PCI (Figura 4.22) e conectá-lo com o barramento PCI-expresso de velocidade muito maior do que a velocidade da interface SATA do disco rígido. Por isso, substituindo o disco rígido pelo SSD, o tempo de *boot* de um sistema pode reduzir drasticamente nos sistemas computacionais modernos [48].

5.5 Decodificador de Endereços

Vimos no Capítulo 4 que o tamanho dos endereços da memória representáveis nas instruções varia entre microprocessadores. Para um tamanho de *n bits*, é possível representar 2^n endereços distintos, ou seja, endereçar todos os *bytes* de uma memória de 2^n *bytes*. No entanto, dificilmente encontra-se disponível no mercado módulos encapsulados de memória exatamente no tamanho desejado. Usualmente encontramos módulos de tamanho menor, 2^m, com o seu próprio espaço de endereços entre 0 a 2^m palavras, endereçáveis por m pinos (Seção 5.2.1). Estas palavras podem ser de 1 *bit*, 4 *bits*, ou 8 *bits*. Como podemos, então, compatibilizar a capacidade de endereçamento de um microprocessador de n pinos de endereços

e k *bytes* de dados com os endereços dos módulos de memória de m pinos de endereços com palavras de 1 *bit*, 4 *bits* ou 1 *byte*?

Uma solução clássica foi, então, juntar vários módulos menores de memória e mapeá-los em sub-espaços de endereços representáveis pelas instruções da CPU e implementar uma **lógica de mapeamento dos endereços** representáveis na CPU em endereços físicos de cada módulo de memória, especificados pelos fabricantes. Na Figura 5.42 ilustra o mapeamento do espaço de endereços 0x0000-0x3FFF de cada uma das 4 pastilhas de memórias 2^{14} *bytes* (1 ROM e 3 RAMs) no espaço de endereços de um microprocessador de 8 *bits*, capaz de endereçar 2^{16} *bytes*.

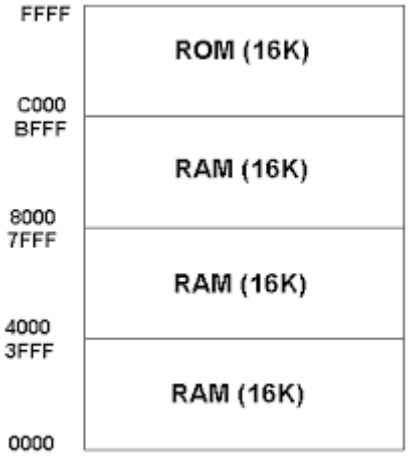


Figura 5.42: Mapa de memória

A ideia é muito simples: conectar todos os j pinos de dados dos módulos de memória com o barramento de dados, através de um *transceiver*[3], e controlar a habilitação destes módulos através do sinal de seleção (/CS) (Seção 5.2.1). Sendo usualmente a quantidade de *bits* necessários para endereçar todos os *bytes* de um módulo de memória muito menor do que a quantidade de *bits* de endereços que a CPU utiliza, separa-se os m *bits* menos significativos para endereçar os 2^m *bytes* dentro do módulo da memória. Na Figura 5.42 seriam os *bits* A13-A0. Os *bits* A15, A14 do espaço de endereços da CPU são utilizados para gerar os sinais de seleção /CS dos 4 módulos:

$$/CS_{ROM} = /A14 \wedge /A15$$
$$/CS_{RAM(em\ cima)} = A14 \wedge /A15$$
$$/CS_{RAM(meio)} = /A14 \wedge A15$$
$$/CS_{RAM(embaixo)} = A14 \wedge A15$$

[3] É um dispositivo bidirecional (trans(-mitter/re-)ceiver) capaz de acionar em relação a um barramento os sinais em duas direções (entrada e saída).

O circuito combinacional que gera os sinais de seleção a partir dos sinais de endereços são chamados de **decodificadores de endereços**. Figura 5.43 ilustra a ideia de partição de uma palavra de endereços em duas regiões: os *bits* menos significativos para endereçar as palavras dentro de um módulo de memória e os *bits* mais significativos para endereçar sub-espaços de endereços alocados para diferentes conjuntos de módulos de memória disponíveis comercialmente.

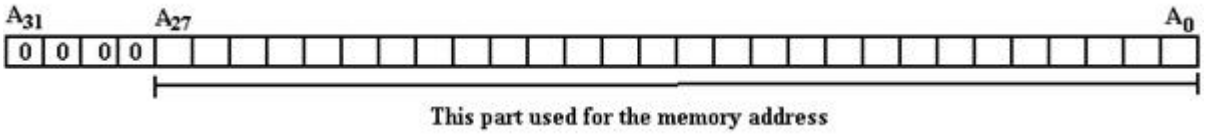


Figura 5.43: Partição dos *bits* de endereços processáveis pela CPU em 2 regiões (Fonte: [51]).

Figura 5.44 ilustra um outro circuito de decodificador de endereços implementado com a porta lógica AND. Os *bits* de endereços A0-A10 do barramento em azul são utilizados para acessar as palavras dentro do módulo da memória. E o restante dos *k bits* mais significativos, A11-A19, são usados para ativar /CS do módulo. Observe que todos os 9 *bits* mais significativos são utilizados na função lógica do sinal /CS:

$$/CS = A11 \wedge A12 \wedge A13 \wedge A14 \wedge A15 \wedge A16 \wedge A17 \wedge A18 \wedge A19$$

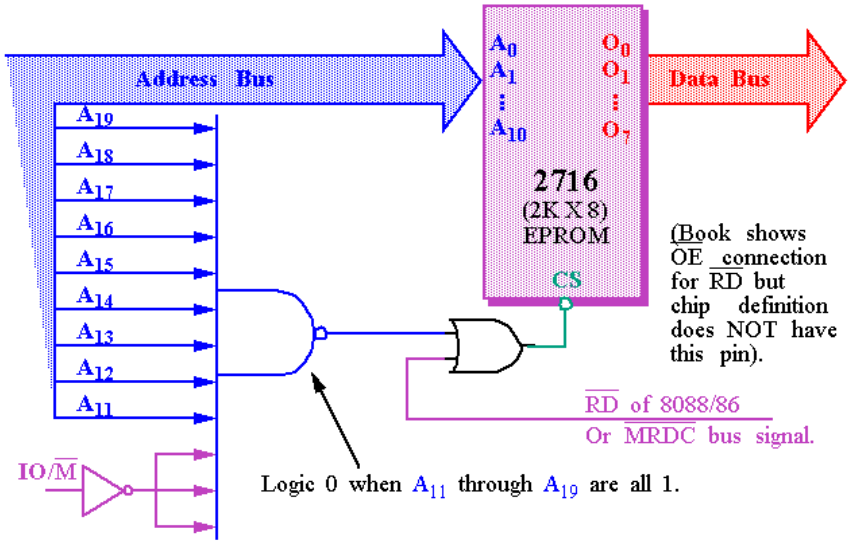
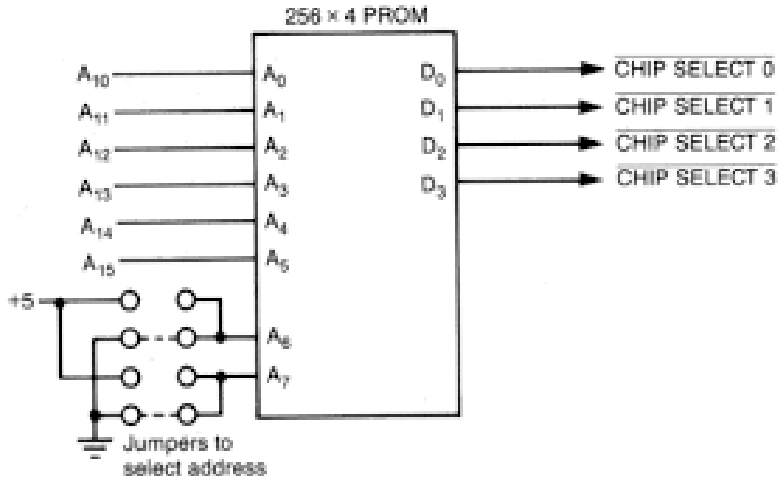


Figura 5.44: Decodificador de endereços: mapeamento do espaço de endereços da CPU ao espaço de endereços físico de um módulo de memória (Fonte: [32]).

Além das portas lógicas, pode-se utilizar na implementação dos decodificadores de endereços os CIs decodificadores, como 74LS138 [33], as memórias ROMs programáveis e arranjos de lógicas programáveis -- *field programmable gate array* (FPGA), *programmable logic array* (PLA) e *programmable array logic* (PAL) [14]. Custo, compatibilidades funcionais, elétricas e temporais são alguns fatores

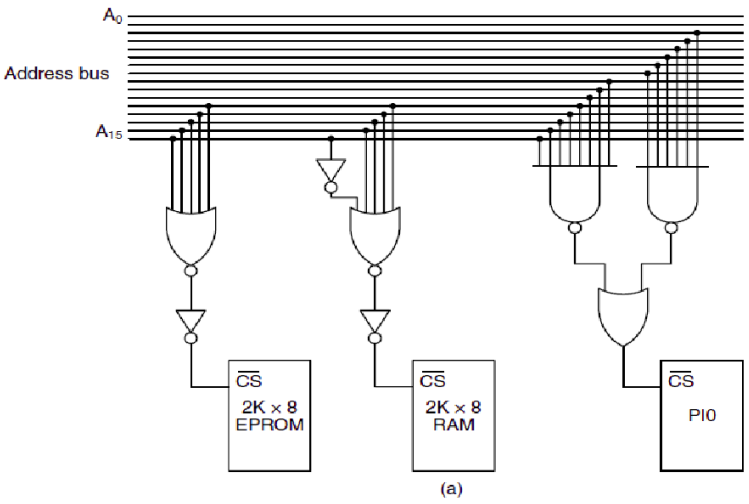
considerados na escolha de uma das tecnologias disponíveis. Como projetistas, não devemos nos esquecer da **latência de propagação do sinal de /CS em relação aos sinais da dados** pelos decodificadores de endereços. Fig. 5.45 ilustra a implementação de um decodificador de endereços com uso de uma memória PROM. Esta solução é mais flexível do que circuitos lógicos de decodificação, porém apresenta uma latência maior. Os 6 *bits* mais significativos do espaço de endereços da CPU foram usados no circuito de decodificação dos sinais de seleção dos módulos /{CHIP SELECT x}.



]

Figura 5.45: Decodificador de endereços implementado com uma memória ROM (Fonte: [34]).

Quando todos os *bits* da parte mais significativa são utilizados na ativação do módulo, dizemos que é uma **decodificação completa**. E, quando parte dos *bits* mais significativos é usada para selecionar os módulos de memória, dizemos que é uma **decodificação parcial**. Na Figura 5.46 os módulos de memória tem uma capacidade de 2K, cujos *bytes* são endereçáveis pelos *bits* A0-A10. Dos restantes *bits* A11-A15, todos eles foram usados na síntese dos sinais /CS dos dois módulos na Figura 5.46.(a). Dizemos então que a decodificação foi completa. Enquanto na Figura 5.46.(b), somente os *bits A14* e *A15* foram usados para decodificação. Por isso, é classificada como decodificação parcial.



(a) Decodificação completa

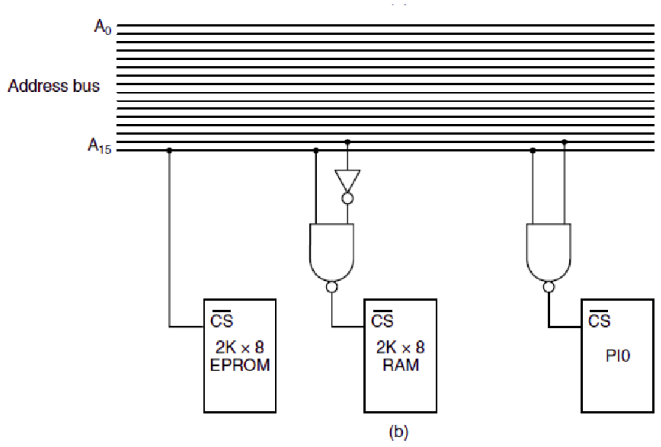


Figure 3-61. (a) Full address decoding. (b) Partial address decoding.

(b) Decodificação parcial

Figura 5.46: Decodificação completa e parcial.

Lembre-se de que apresentamos na Seção 4.1.3 que os endereços das palavras alinhados são sempre em potência de dois: 1 *byte*, 2 *bytes* (*halfword*), 4 *bytes* (*word*) e 8 *bytes* (*double word*)? Em termos de endereçamentos, considerando que a menor unidade endereçável do sistema de memória, os endereços de dois *bytes* adjacentes diferem no *bit* menos significativo A0, os endereços dos dois *halfwords* adjacentes diferem no *bit* A1, os endereços dos dois *words* adjacentes diferem em A2 e dos dois *double words* em A3. Ou seja, os *bits* de endereços que distinguem

os tipos de dados a serem endereçados são os *bits* menos significativos de um endereço. Como os módulos de memória disponíveis no mercado apresentam uma grande variação no tamanho das suas palavras, tipicamente 1 *bit*, 4 *bits* e 8 *bits*, cabe aos projetistas adequar agrupar os módulos de memória para formar uma palavra endereçável por um processador.

Vamos ilustrar com um caso de dois módulos de memória de 8 *bits* e um barramento de dados de 16 *bits*, apresentado na Figura 5.47, e que o processador suporta dois tipos de alinhamento, em *bytes* e em *halfwords*. Observe os *bits* de endereços utilizados para endereçar os *bytes* dentro de um módulo de ROM de 8K na Figura 5.47: A1-A13, e não A0-A12! Por quê? O *bit* menos significativo A0 foi utilizado para a síntese do sinal /LRD (*Low Read*), que habilita a leitura do módulo à direita. O inverso de A0, /BHE (*Byte High Enable*), foi usado para a síntese do sinal /HRD, que habilita a leitura do módulo à esquerda. Com isso, podemos selecionar os 16 *bits* (*halfword*) simultaneamente se setarmos A0=/BHE=1; do contrário, o valor do A0 controla acessos individuais de cada módulo. Para os endereços pares, /LRD é ativo e os dados D0-D7 deste módulo são transferidos para o barramento. E para os endereços ímpares, o sinal /HRD é ativo e os dados do módulo correspondente transferidos para o barramento. Olhando do lado da CPU, o módulo à direita armazena as palavras de endereços pares e o módulo à esquerda, as palavras de endereços ímpares.

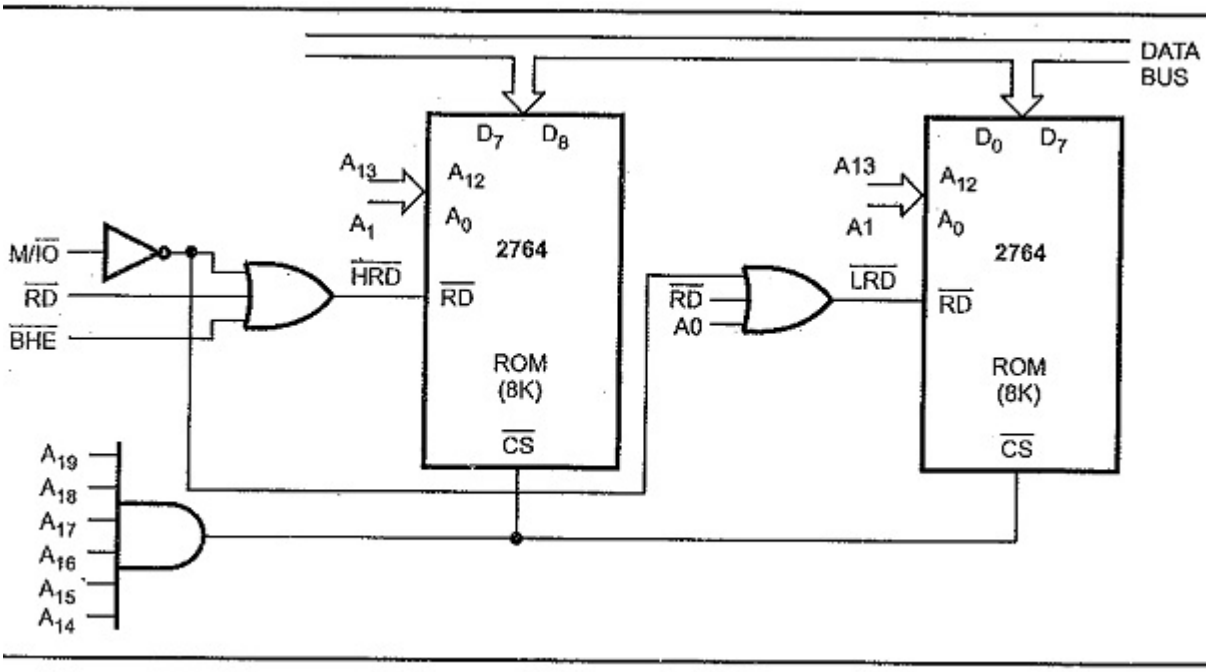


Fig. 10.12 Absolute decoding

Figura 5.47: Distinção dos endereços pares e ímpares num mesmo subespaço de endereços (Fonte: [37]).

Qual é a ideia que está por trás do esquemático na Figura 5.47? Para suportar os diferentes tamanhos naturais das palavras dos microporcessadores e diferentes alinhamentos permitidos, a palavra de endereço processável pela CPU foi dividida em 3 regiões (Figura 5.48): (1) os *bits* mais significativos para seleção dos módulos mapeados num subespaço endereçável pela CPU, (2) os *bits* menos significativos para endereçar submódulos dentro de um mesmo subespaço que correspondem aos diferentes alinhamentos dentro do tamanho natural[4], e (3) os *bits* no meio para endereçar as palavras dentro de um módulo de memória. Em termos práticos, o circuito de síntese dos *bits* mais significativos em subespaços de endereços é o decodificador de endereços, e os seletores de módulos dentro de um subespaço, isto é os *bits* menos significativos, são usados na síntese dos sinais de habilitação de leitura e escrita (R//W) de cada módulo.

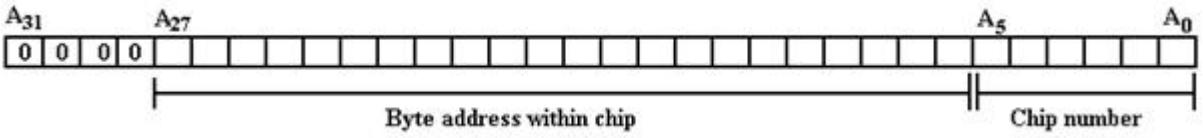


Figura 5.48: Partição de *bits* de endereços processáveis pela CPU em 3 regiões (Fonte:[51]).

Temos ainda o endereçamento por bloco de módulos de memória de forma que o conteúdo do mesmo endereço interno de todos estes módulos possa ser acessado simultaneamente. Figura 5.49 mostra uma técnica de decodificação por bloco em que os 3 *bits* mais significativos, A19-A17, são usados para selecionar os 4 módulos, 2 EPROMs e 2 RAMs, e os restantes 13 *bits*, as 2^{13} (8K) palavras da memória. Note que os 2 módulos EPROM compartilham o mesmo sinal de seleção Y7 e os 2 módulos RAM compartilham o sinal Y0. Ambos os sinais de seleção são obtidos com a decodificação dos sinais A19, A18 e A17 através do decodificador 74138. Vale observar que o projetista deste decodificador usou mais *bits* do que os necessários para endereçar os quatro módulos de memória. Esta estratégia é muito utilizada pelos projetistas como uma forma de prever futuras expansões. Neste caso em particular, até 6 novos blocos (Y1-Y6) podem ser adicionados ao sistema. Observe ainda que os *bits* A1-A13 foram utilizados para endereçar as palavras em cada módulo, ficando A0 dedicado à seleção dos *bytes* em cada subespaço de endereços de palavras de 2 *bytes*.

[4] Tamanho, em *bits*, dos registradores disponíveis na CPU.

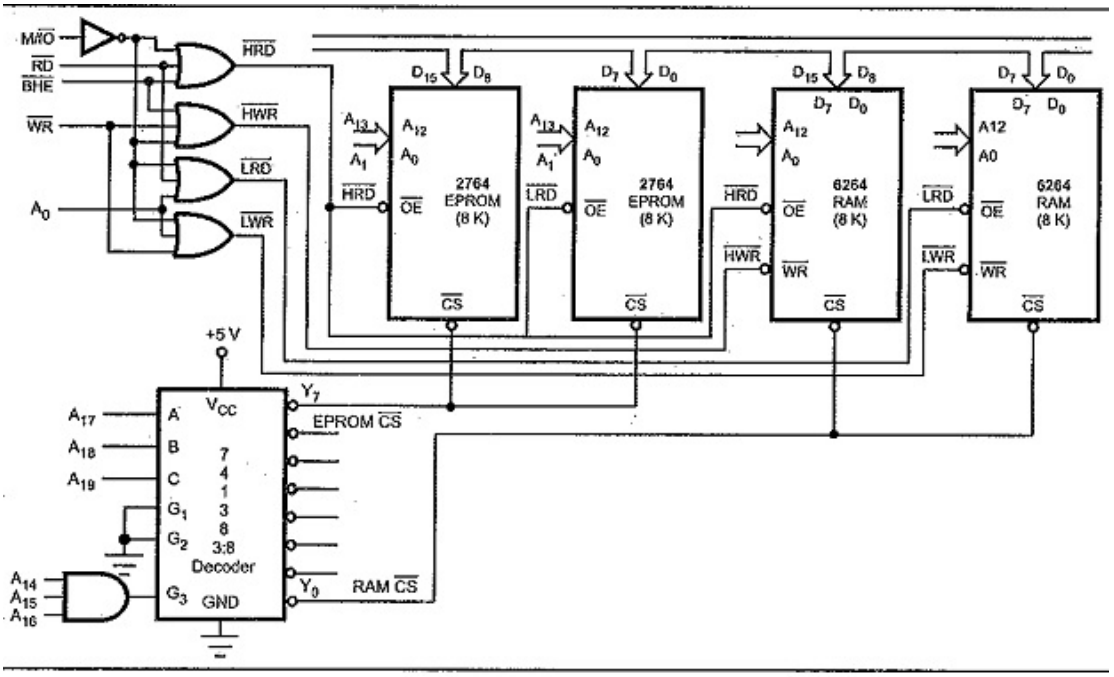


Fig. 10.14 Block decoding

Figura 5.49: Decodificação por bloco (Fonte: [37]).

5.6 Compatibilidade Temporal

(baseado no cap. 4 de [14])

Tipicamente, os módulos de memória tem os pinos de endereços e de dados para acessar cada palavra contida em cada um dos seus endereços. Há ainda os sinais de controle para controlar o fluxo de dados como vimos na Seção 5.2.1. Embora os nomes dos pinos possam variar, sempre existe um pino de controle de habilitação do módulo (para evitar contenção do barramento[5]) e um pino de controle da direção de fluxo dos dados (leitura ou escrita). Vimos na Seção 5.5 que os sinais de controle dos módulos de memória são, de fato, funções lógicas dos sinais de controle dos processadores, a fim de atenderem as solicitações de instruções e de dados necessários para a execução de um programa na CPU, Figura 5.50 mostra uma implementação das funções lógicas entre os sinais de um processador 68000 de 16 *bits* [62] e dois módulos de memória 6116 de 2Kx8 *bits* [53]. Para assegurar que o sistema opere corretamente, devemos analisar ainda a compatibilidade entre as restrições temporais dos sinais gerados pela CPU e as restrições temporais dos sinais esperados pelos módulos da memória, detalhadamente discutida no capítulo 4 em [14]. Um resumo das principais ideias é apresentado nesta seção.

[5] Contenção do barramento é quando há mais de um dispositivo ativo acionando o barramento.

Na Seção 5.2 vimos os diagramas de tempo dos módulos de memória, agora vamos ver como relacioná-los com os diagramas de tempo dos processadores para que as ações individuais de cada dispositivo sejam consistentes com a execução da tarefa programada. Esta habilidade de relacionar as restrições temporais é fundamental no projeto de um sistema digital sequencial.

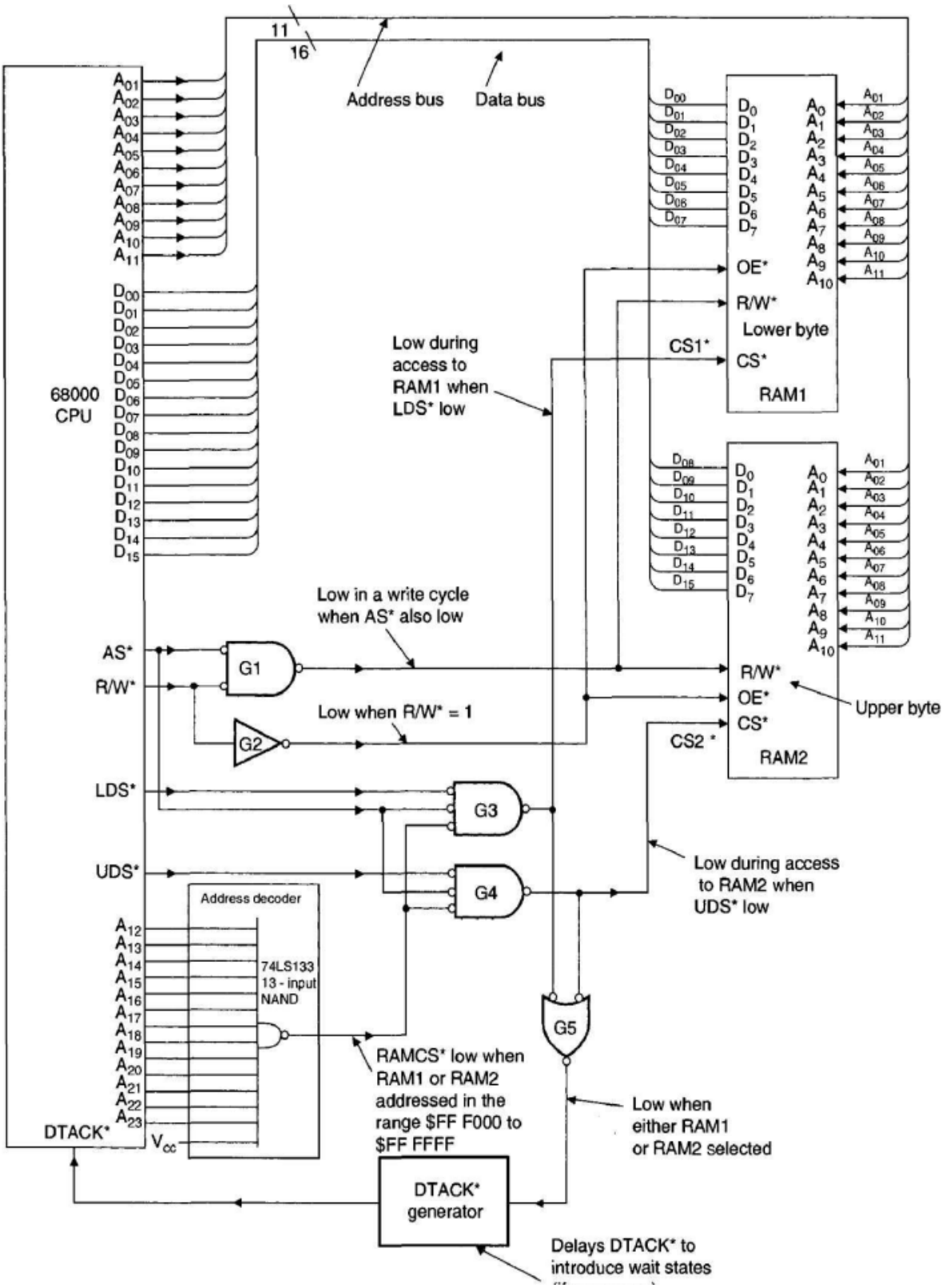


Figura 5.50: Conexão entre um processador 68000 e 2 módulos de memória 6116 (Fonte: [14]).

Para simplificar a nossa análise, vamos desprezar os tempos de propagação dos sinais. Essencialmente, nos ciclos de acessos à memória a CPU precisa colocar os endereços válidos no barramento de endereços e, conforme o modo do ciclo ser de leitura ou de escrita, a memória ou a CPU precisam colocar os dados no barramento de dados. Para que ambos os dispositivos consigam ler corretamente os endereços/dados do barramento, é necessário assegurar que o tempo de *setup*[6] e o tempo de *hold*[7] em relação ao sinal que atua como evento de relógio seja compatível com o dispositivo que esteja fazendo leituras.

Observe na Figura 5.50 que os módulos de memória estão mapeados no espaço de endereços 0xFFF000 a 0xFFFFFF. Os pinos /UDS e /LDS (A0) são usados para endereçar o *byte* menos (RAM1) e o mais significativo (RAM2) de uma palavra de 16 *bits* que a CPU acessa em cada ciclo de leitura. Os sinais de controle da memória foram sintetizadas a partir das seguintes funções lógicas:

/CS1 = /(LDS ΛAS Λ/(A23 ΛA22 ΛA21 ΛA20 ΛA19 ΛA18 ΛA17 ΛA16 ΛA15 ΛA14 ΛA13 ΛA12))
/CS2 = /(UDS ΛAS Λ/(A23 ΛA22 ΛA21 ΛA20 ΛA19 ΛA18 ΛA17 ΛA16 ΛA15 ΛA14 ΛA13 ΛA12))
/OE = /(R//W)
$R//W_{mem}$ = /(AS Λ$R//W_{CPU}$)

Além disso, a CPU espera uma ativação do sinal /DTACK para ela “entender” que os dados estão disponíveis no barramento de dados. Como o módulo 6116 não gera este sinal, uma lógica foi adicionada:

/DTACK = /(CS1 V CS2).

5.6.1 Ciclo de Leitura

Vamos analisar primeiro ciclos de leitura. Figura 5.51 apresenta a sequência de ações que o 68000 executa num ciclo de leitura, cuja representação gráfica em diagrama de tempo é mostrada na Figura 5.52. Figura 5.53 mostra o diagrama de tempo de um ciclo de leitura do módulo de memória.

Com base nas funções lógicas que relacionam os sinais de controle da CPU e da memória, podemos verificar as seguintes restrições temporais relevantes para o funcionamento correto do sistema:

- **acesso ao endereço:** como a memória 6116 não armazena o endereço é importante que os endereços válidos estejam disponíveis no barramento durante o ciclo de acesso à memória (t_{RC}, Figura 5.53).

$$3.5/8MHz - t_{CLAV} + t_{CHADZ} = 437,5ns - 70ns + 80ns > t_{RC}$$

[6] Tempo de *setup* é o tempo mínimo que uma entrada síncrona deve estar estável antes da ocorrência do evento de relógio (sinal de *clock*) para assegurar que o sinal seja apropriadamente amostrado.
[7] Tempo de *hold* é o tempo mínimo que uma entrada síncrona deve estar estável após a ocorrência do evento de relógio (sinal de *clock*) para assegurar que o sinal seja apropriadamente amostrado.

Se este tempo for maior do que o ciclo de operação da CPU (4 ciclos de relógio, Figura 5.52), **estados de espera** devem ser adicionados no sinal /DTACK.

- **acesso ao dado**: pela Figura 5.52 o evento de relógio para a memória ler um dado do barramento é a borda de descida do sinal de *clock* em S6 (3 pulsos de relógio, Figura 5.52). Como (1) o tempo de *setup* dos dados em relação a este sinal é t_{DICL}, (2) os endereços só ficam válidos no barramento de endereços após t_{CLAV} da borda de descida do sinal de *clock* em S0 e (3) a memória coloca dado válido no barramento de dados após t_{AA} do instante em que os endereços ficam válidos (Figura 5.53), a seguinte restrição temporal deve ser satisfeita

  3 ciclos de *clock* = 3*125ns = 375ns > $t_{CLAV} + t_{AA} + t_{DICL} > 70ns + t_{AA} + 15ns$

  Ou seja,

  $$375ns - 70ns - 15\ ns = 290ns > t_{AA}.$$

  Precisamos ainda verificar o tempo de *hold* destes dados para assegurar que eles sejam lidos corretamente. Pela Figura 5.52, a CPU espera que os dados fiquem por t_{SHDI} após a borda de subida de /AS. Analisando a função lógica de /CSx, concluímos que /CSx também transitam do nível 0 para o nível 1 neste instante se desprezarmos os tempos de propagação dos sinais. O tempo t_{CHZ} (Figura 5.53) deve, portanto, satisfazer a seguinte restrição:

  $$t_{CHZ} > t_{SHDI} = 0ns.$$

- **contenção de barramento**: observe que a memória começa ocupar o barramento de dados t_{CLZ} após a borda de descida do sinal /CSx. Isso significa que precisamos assegurar que a CPU não acesse o barramento a partir deste instante para evitar contenção de barramento. Como o sinal /DTACK é derivado do sinal /CSx, se o tempo t_{ASI} que a CPU aguarda pelos dados válidos após a borda de descida de /DTACK (Figura 5.52) satisfizer a seguinte restrição temporal:

  $$t_{ASI}=20ns > t_{CLZ}$$

  não ocorrerá garantidamente a contenção.

  É preciso também verificar se o tempo máximo t_{CHZ} em que a memória mantém os dados no barramento de dados após a desativação do sinal /CSx possa causar a contenção do barramento. Como /CSx é derivado de /AS e ele fica no nível 1 após t_{CLSH} da borda de descida do sinal de *clock* em S6, então

  $$t_{CLSH} + t_{CHZ} = 70ns + t_{CHZ} < 0.5/8MHz = 0.5*125ns = 62.5ns.$$

  Ou seja,

  $$t_{CHZ} < 62.5ns - 70ns < -7.5ns,$$

  podendo a memória ocupar o barramento enquanto a CPU inicia o próximo ciclo de operação (estado S0). Felizmente, a duração do estado S0 é 62.5ns

e neste estado todos os sinais da CPU estão “flutuando” (inativos). A pequena “invasão” não causaria problema.

Sintetizando, os parâmetros da memória 6116 satisfazem as restrições temporais da CPU 68000 no ciclo de leitura:

Parâmetro da memória	Restrições de 68000 em 8 MHz	Valor	Operação da memória
t_{RC}	< 3,5 ciclos de *clock* -t_{CLAV} + t_{CHADZ}	447.5ns	200ns
t_{AA}	< 3 ciclos de *clock*-t_{CLAV}-t_{DICL}	290ns	200ns
t_{CHZ}	> t_{SHDI}	0ns	50ns
	< 1 ciclo de relógio-t_{CLSH} -t_{CHZ}	55ns	
t_{CLZ}	< t_{ASI}	20ns	15ns

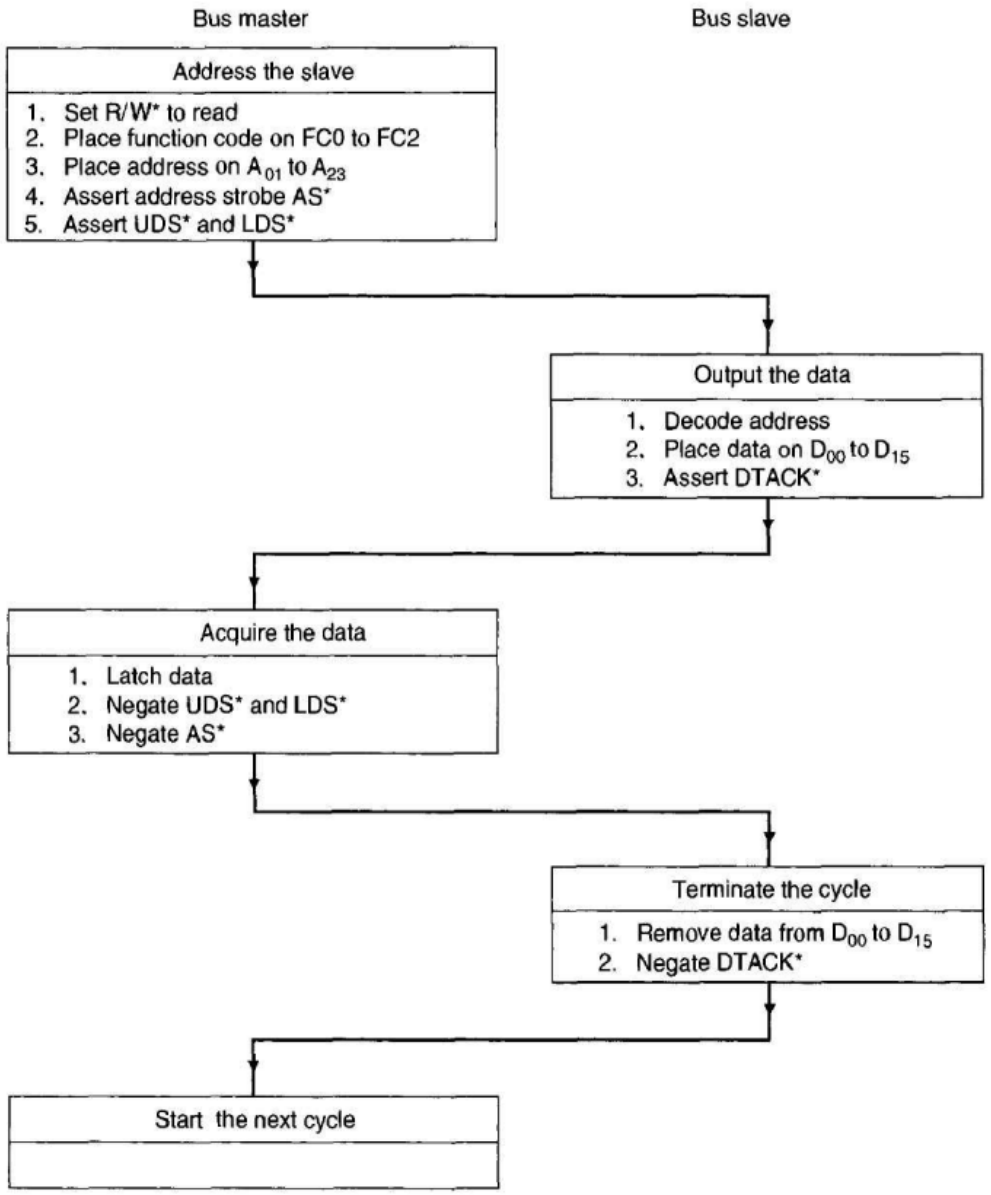
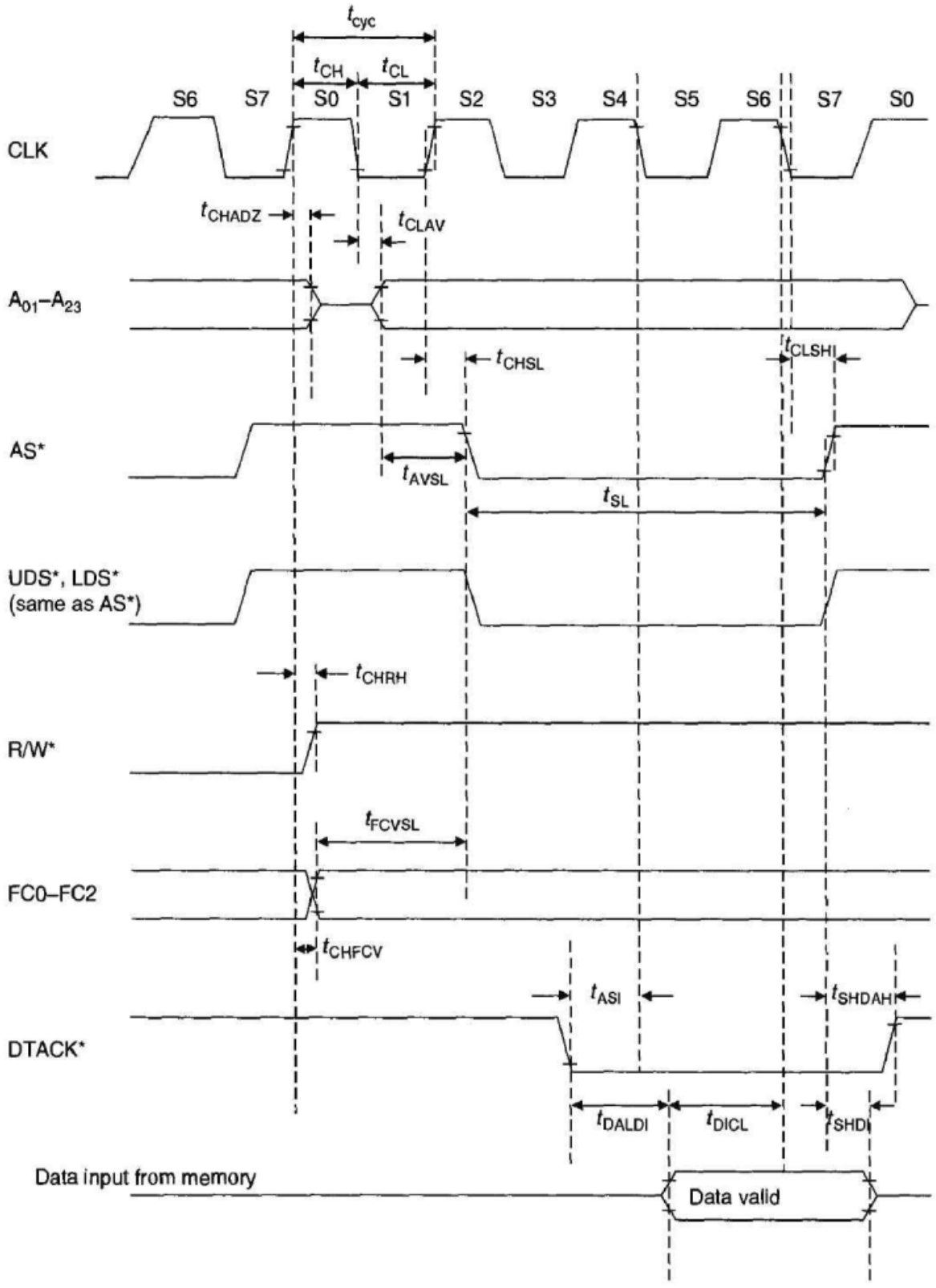
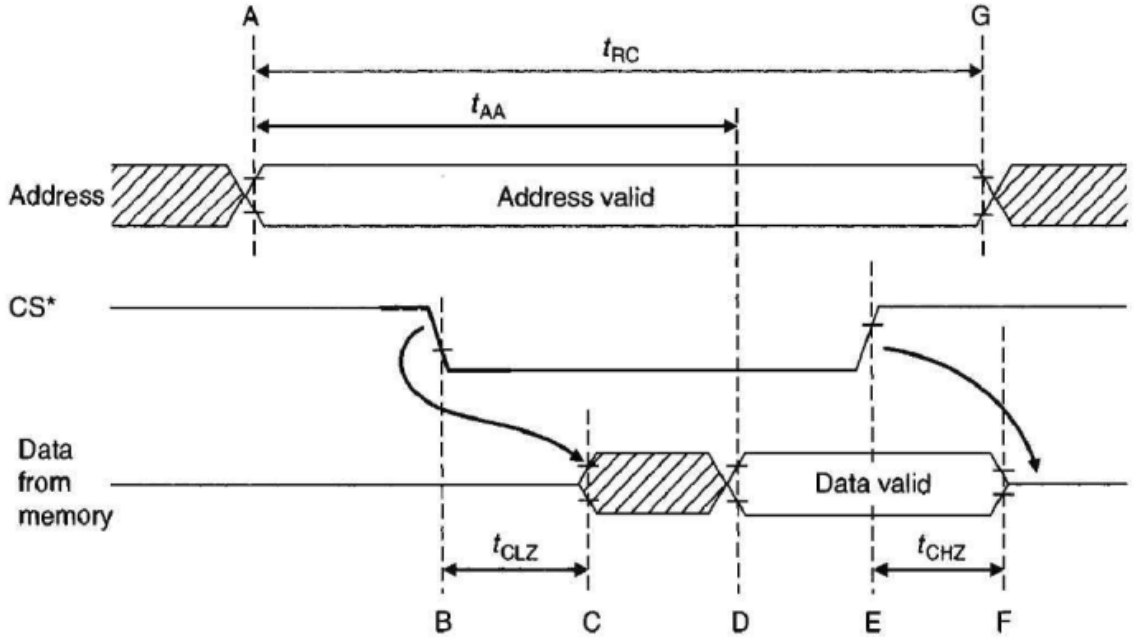


Figura 5.51: Descrição funcional de um ciclo de leitura de 68000 (Fonte: [14]).


t_{cyc}
t_{CH}
t_{CL}
S6
S7
S0
S1
S2
S3
S4
S5
S6
S7
S0
CLK
t_{CHADZ}
t_{CLAV}
A_{01}–A_{23}
t_{CHSL}
t_{CLSHI}
AS*
t_{AVSL}
t_{SL}
UDS*, LDS*
(same as AS*)
t_{CHRH}
R/W*
t_{FCVSL}
FC0–FC2
t_{CHFCV}
t_{ASI}
t_{SHDAH}
DTACK*
t_{DALDI}
t_{DICL}
t_{SHDI}
Data input from memory
Data valid

Parameter Name	Symbol	Minimum	Maximum
Clock period	t_{cyc}	125	250
Clock width (low)	t_{CL}	55	125
Clock width (high)	t_{CH}	55	125
Clock high to address bus high impedance	t_{CHADZ}		80
Clock low to address valid	t_{CLAV}		70
Address valid to AS* low	t_{AVSL}	30	
Clock high to AS*, DS* low	t_{CHSL}	0	60
Clock low to AS*, DS* high	t_{CLSH}		70
AS*, DS* with low	t_{SL}	240	
Clock high to R/W* high	t_{CHRH}	0	70
Clock high to function code valid	t_{CHFCV}		70
Function code valid to AS* low	t_{FCVSL}	60	
Asynchronous input DTACK* setup time	t_{ASI}	20	
AS*, DS* high to DTACK* high	t_{SHDAH}	0	245
Data in to clock low setup time	t_{DICL}	15	
DS* high to data invalid (data hold time)	t_{SHDI}	0	
DTACK low to data in setup time	t_{DALDI}		90

Figura 5.52: Representação gráfica da descrição funcional na Figura 5.51 e restrições temporais para 68000 operando em 8MHz (Fonte: [14]).



Note: R/W* is high for the duration of the read cycle and OE* is low. The read access is controlled entirely by CS*.

Parameter	Mnemonic	Minimum	Maximum
Read cycle time	t_{RC}	200 ns	
Address access time	t_{AA}		200 ns
Chip select to output not floating	t_{CLZ}		15 ns
Chip deselect to output floating	t_{CHZ}	0 ns	50 ns

Figura 5.53: Diagrama do ciclo de leitura da memória 6116 (Fonte: [14]).

5.6.2 Ciclo de Escrita

Figura 5.54 mostra o fluxo de execução de um ciclo de escrita pelo microprocessador 68000.

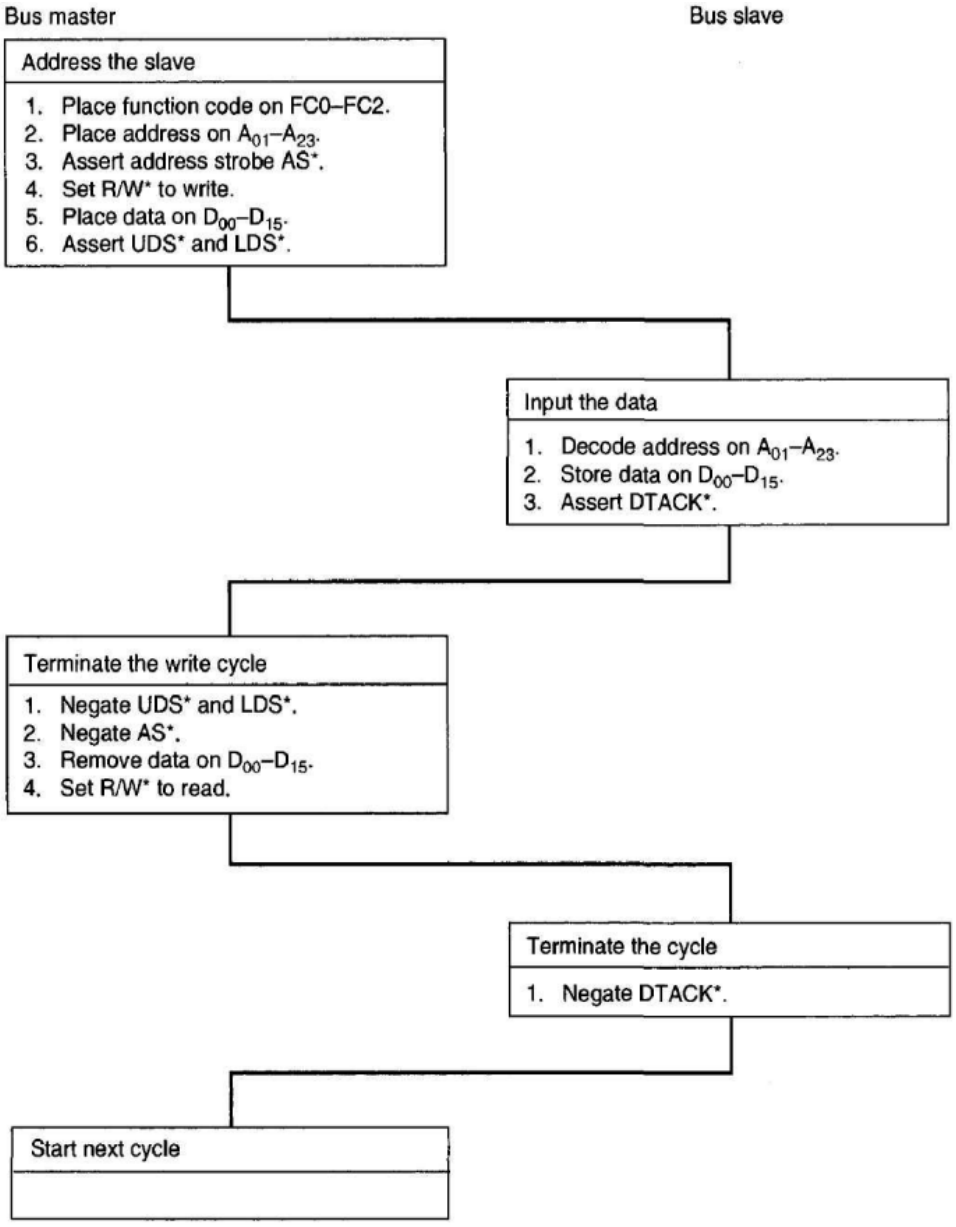
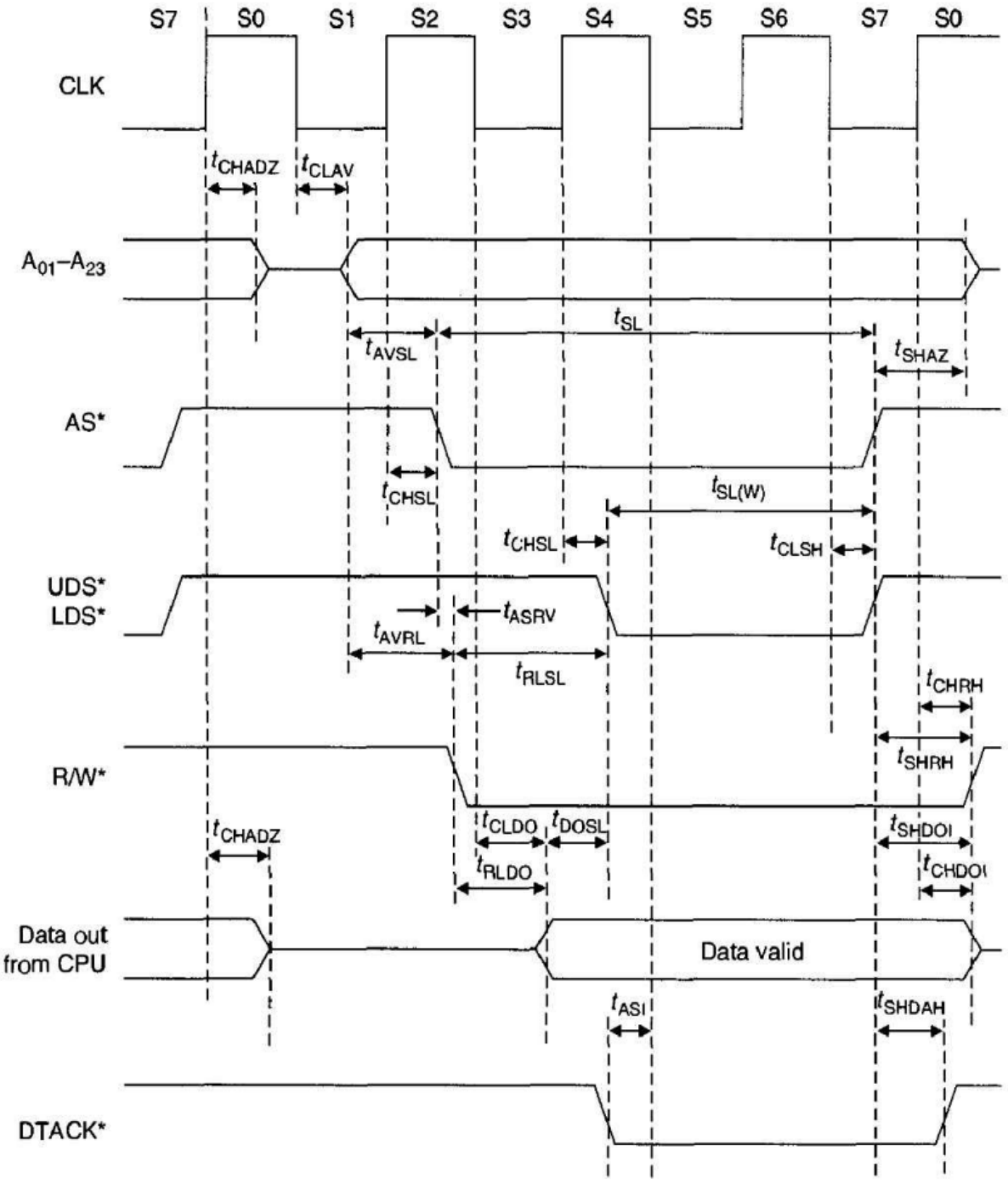


Figura 5.54: Descrição funcional de um ciclo de escrita de 68000 (Fonte: [14]).

Uma versão gráfica enfatizando as relações temporais dos sinais é mostrado no diagrama de tempo da Figura 5.55.

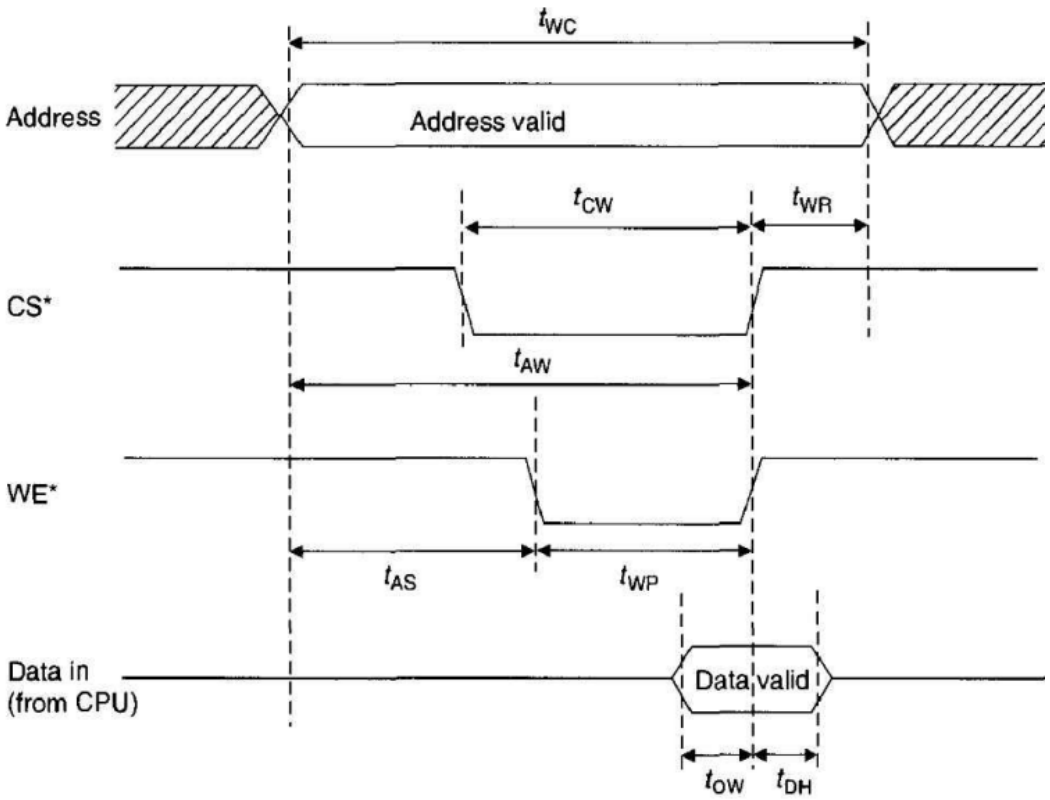


Parameter Name	Symbol	8 MHz	
		Minimum	Maximum
Clock period	t_{cyc}	125	250
Clock high to data and address bus high impedance	t_{CHADZ}		80
Clock low to address valid	t_{CLAV}		62
Address valid to AS* asserted	t_{AVSL}	30	
AS* asserted	t_{SL}	270	
AS*, DS* negated to address bus high impedance	t_{SHAZ}	40	
Clock high to AS*, DS* asserted	t_{CHSL}	3	60
Clock low to AS*, DS* negated	t_{CLSH}		62
DS* asserted in write cycle	$t_{SL(W)}$	140	
AS* asserted to R/W* low	t_{ASRV}		10
Address valid to R/W* low	t_{AVRL}	20	
R/W* low to DS* asserted	t_{RLSL}	80	
Clock high to R/W* high	t_{CHRH}	0	55
AS*, DS* negated to R/W* high	t_{SHRH}	40	
Clock low to data out valid	t_{CLDO}		62
Data out valid to DS* asserted	t_{DOSL}	40	
AS*, DS* negated to data out invalid	t_{SHDOI}	40	
Data out hold from clock high	t_{CHDOI}	0	
R/W* low to data out valid	t_{RLDO}	30	
Asynchronous input setup time (DTACK* setup)	t_{ASI}	10	
AS*, DS* negated to DTACK* negated (asynchronous hold)	t_{SHDAH}	0	240

Note: Some of the parameters in this table differ from those of Table 4.4. The reader should be aware that the
may change when the manufacturer publishes an updated data sheet.

Figura 5.55: Representação do ciclo de escrita da Figura 5.54 em diagrama de tempo (Fonte: [14]).

Para podermos analisar a compatibilidade temporal de memória precisamos ter acesso às restrições temporais mostradas no diagrama de tempo do ciclo de escrita da memória na Figura 5.56.



Parameter Name	Symbol	Minimum	Maximum
Write cycle time	t_{WC}		150
Chip select low to end of write	t_{CW}	90	
Write recovery time	t_{WR}	10	
Address valid to end of write	t_{AW}	120	
Address setup time	t_{AS}	20	
Write pulse width	t_{WP}	90	
Data setup time	t_{DW}	40	
Data hold time	t_{DH}	10	

Figura 5.56: Diagrama de tempo do ciclo de escrita da memória 6116 (Fonte: [14]).

Análogo ao ciclo de leitura, precisamos agora assegurar que os dados colocados pela CPU no barramento estão temporalmente compatíveis com os tempos da memória. A memória requer que os dados tenham um tempo de *setup* t_{DW} e de *hold* t_{DH} em relação à borda de subida de /CS e R//WE_{mem}. Pela lógica que usamos para gerar o sinal R//WE_{mem} estas bordas coincidem com a borda /AS. Portanto, a seguinte restrição temporal deve ser satisfeita:

$$t_{SL(W)} + t_{DOSL} = 140ns + 40ns > t_{DW}.$$

E o tempo de *hold* garantido pela CPU é no mínimo

$$t_{SHDOI} = 40ns > t_{DH}.$$

O ciclo de escrita completo da memória deve ser menor que o intervalo de tempo em que os endereços colocados pela CPU são válidos. Isso garante que um ciclo de operação da CPU não seja atrasado:

$$t_{WC} < t_{AVSL} + t_{SL} + t_{SHAZ} < 30ns + 270ns + 40ns = 340ns.$$

Diferente do ciclo de leitura a largura do pulso ativo de /CS deve satisfazer a restrição temporal:

$$t_{CW} < t_{SL(W)} = 140ns.$$

Como o sinal $R//W_{mem}$ é derivado de /AS e $R//W_{CPU}$, a largura do pulso ativo de $R//W_{mem}$ é

$$t_{WP} < t_{SL} - t_{ASRV} = 270ns - 10ns = 260ns.$$

E o intervalo de tempo entre o instante da CPU colocar endereços válidos até $R//W_{mem}$ ficar ativo baixo deve ser maior que t_{AS}, ou seja,

$$t_{AS} < t_{AVRL} = 20ns.$$

Finalmente, o tempo mínimo para que o barramento de endereços seja desativado pela CPU deve ser compatível com o tempo requerido pela memória:

$$t_{WR} < t_{SHAZ} = 40ns.$$

A tabela abaixo sintetiza comparativamente a compatibilidade temporal entre os ciclos de escrita de 68000 e da memória 6116 [14].

Memory Parameter	Parameter Expressed in 68000 Terms	Value	Required	Excess
t_{WC}	$t_{AVSL} + t_{SL} + t_{SHAZ}$	$30 + 270 + 40 = 340$	150 min	190
t_{CW}	$t_{SL(w)}$	140	90 min	50
t_{WR}	t_{SHAZ}	40	10 min	30
t_{AW}	$t_{AVSL} + t_{SL}$	$30 + 270 = 300$	120 min	180
t_{AS}	t_{AVRL}	20	20 min	0
t_{WP}	$t_{SL} - t_{ASRV}$	$270 - 10 = 260$	90 min	170
t_{DW}	$t_{DOSL} + t_{SL(w)}$	$40 + 140 = 180$	40 min	140
t_{DH}	t_{SHDOI}	40	10 min	30

5.7 Unidade de Gerenciamento de Memória

Embora o espaço de endereços representáveis na CPU tenha aumentado muito com o lançamento de microprocessadores de 32 ou 64 *bits*, a demanda pela memória tem aumentado ainda mais rapidamente. Exigências por multiprocessamento e multiprogramação nos sistemas operacionais tem aumentado paralelamente, de forma que o espaço de endereços representáveis na CPU não consegue mais cobrir a demanda. Uma solução foi definir um **espaço de endereçamento virtual** dividido em unidades denominadas **páginas**, em inglês *pages*. As unidades correspondentes na memória física são chamadas **molduras de página**, em inglês *page frames* [36].

O programa que precisa ser executado não precisa mais se limitar à quantidade máxima das molduras de página de um sistema computacional. Com base nos endereços virtuais das suas instruções, a **unidade de gerenciamento de memória**, em inglês *memory management unit*, consegue mapear os endereços virtuais em endereços físicos da memória principal como mostra a Figura 5.57. Caso seja detectada a falta da página na memória principal, é gerado o evento de interrupção de **falta de página**, em inglês *page fault*, para substituir uma moldura de página, por exemplo a menos utilizada, pela página faltante, transferindo-a do disco para a memória principal de forma totalmente transparente para o usuário. Nos sistemas computacionais de mesa, esta tarefa é sempre executada pelo sistema operacional [36].

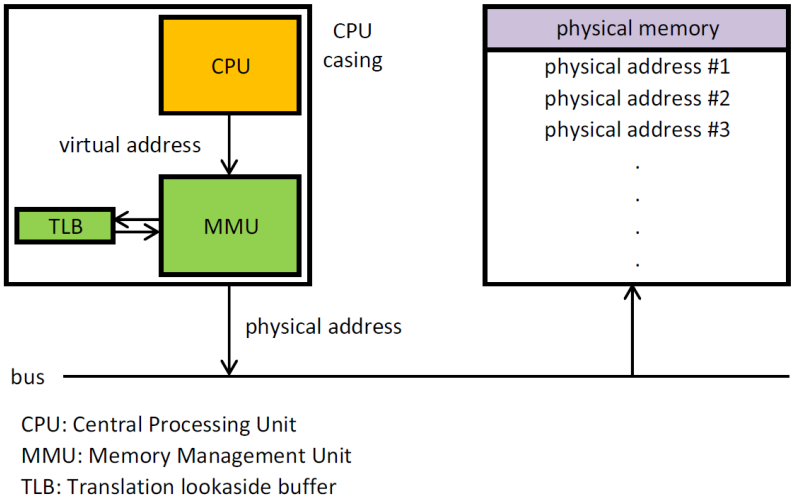


Figura 5.57: Espaço de endereçamento virtual (Fonte:[35]).

Figura 5.58 mostra o mapeamento de um espaço de endereçamento virtual muito maior do que o espaço de endereçamento físico por **paginação**.

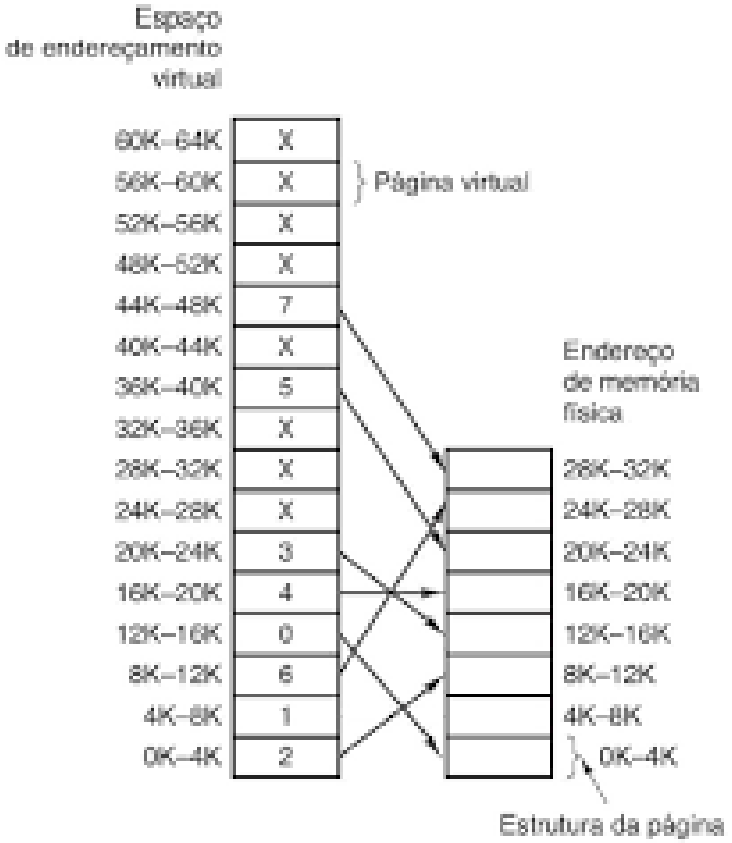


Figura 5.58: Paginação.

Além de controlar o mapeamento entre um espaço físico de memória principal menor do que um espaço virtual endereçável pela CPU, a unidade de gerenciamento de memória facilita:

- o controle do mapeamento entre um espaço físico da memória principal maior do que um espaço virtual endereçável pela CPU.
- a proteção de acessos não autorizados aos espaços de memória.
- o aumento na eficiência do uso do espaço de memória.
- o uso transparente do sistema de memória.

Não sendo a unidade de gerenciamento de memória o foco desta disciplina, não vamos aprofundar aqui as políticas de troca de páginas nem de acessos aos sub-espaços de memória. Voltaremos no Capítulo 6 às técnicas de substituição de dados faltantes no contexto de memória *cache*.

5.8 Abstração em Programação de Alto Nível: Tipos de Dados

Na seção 4.1.1 vimos que os dados armazenados na memória são abstraídos em instruções e operandos. Através dos modos de endereçamento suportados pelo microprocessador, pode-se distinguir operandos de tamanhos diferentes. Porém, pelo volume de dados que a maioria dos aplicativos modernos lida, programar um microprocessador em que se precisa atentar para os tamanhos de todos os dados e elaborar modos de endereçamento apropriados é uma tarefa tediosa e propensa a muitos erros. Na seção 4.8.2 vimos que estes detalhes são hoje abstraídos através dos **compiladores**, que traduzem programas em linguagem de alto nível para linguagem de máquina, com a tarefa de otimizar a alocação dos espaços de memória para os dados com base nas instruções dadas pelos programadores e nas regras de alinhamento dos acessos (Seções 4.1.3 e 5.5).

Nesta seção vamos ver como os tamanhos dos operandos e seus alinhamentos são abstraídos em linguagem de alto nível, mais especificamente em C, e como podemos empregar os nossos conhecimentos acerca da arquitetura e da organização do sistema de memória para aprimorar o nosso uso deste recurso através dos comandos da linguagem C.

Em linguagem de alto nível, os dados, sejam eles **variáveis** ou **constantes**, são associados a algum **tipo de dados.** Um tipo de dado especifica o espaço de memória reservado para armazenar o dado, o seu alinhamento nos acessos, a forma como se deve interpretar o código binário contido neste espaço, e as operações lógico-aritméticas que podem ser aplicadas sobre tal código binário. A linguagem C suporta 5 tipos básicos: **char**, **int** (inteiros), **float** (valores de precisão simples em ponto flutuante), **void** (endereço de um valor de qualquer tipo), **double** (valores de precisão dupla em ponto flutuante).

É interessante observar que os valores do tipo **char** são sempre representados por 1 *byte* (8 *bits*), enquanto o tamanho dos valores do tipo **int** normalmente corresponde ao tamanho natural dos registradores, ou seja, de um ***word*** de um processador. Num processador de 16 *bits*, com registradores de 16 *bits*, um valor do tipo **int** é representado por 16 *bits*. E num núcleo de 32 *bits*, é então representado por 32 *bits*. Neste último caso, um valor representado por 16 *bits* é usualmente conhecido como de tamanho de um ***halfword*** e um valor representado por 64 *bits*, ***doubleword***. (Compare estes temos com as definições dadas nas seções 4.1.3 e 5.5!) Dizemos, portanto, que o espaço de memória ocupado pelo tipo **int** é dependente da máquina, e, a partir dele, infere-se o espaço de memória dos tipos **float**, **void** (mesma quantidade de memória do tipo **int**) e **double** (duas vezes a quantidade de memória do tipo **int**).
Os tipos de dados podem ser acompanhados de **qualificadores**, que são palavras-chave que modificam as propriedades dos tipos de dados básicos. Vamos introduzir neste capítulo dois tipos de qualificadores que PODEM ajudar um compilador na otimização do uso do espaço de memória para os dados do nosso programa:

1. Em relação ao sinal: **signed** e **unsigned**;

2. Em relação ao tamanho: **short** e **long**;

Os qualificadores de sinal podem ser utilizados, por exemplo, para estabelecer que uma determinada variável **int** receba apenas valores positivos (**unsigned**). No caso do modificador **signed**, não é necessário utilizá-lo, pois, com exceção do tipo **char**, as variáveis são inicialmente **signed**, representados em complemento de 2.

Os qualificadores de tamanho alteram o tamanho do tipo de dado básico, podendo ampliá-lo (**long**) ou diminuí-lo (**short**). Estes são utilizados quando, em geral, sabe-se que o tamanho básico do tipo de dados excede ou não é suficiente para representar os valores a serem processados.

Os qualificadores **signed**, **unsigned**, **long** e **short** são aplicáveis ao **int**, ou seja, além de termos a distinção entre valores inteiros com sinal e sem sinal, podemos definir através dos qualificadores a quantidade de inteiros representáveis, maior (**long**) ou menor (**short**). Eles não podem ser aplicados ao **float**, pois estes tem formatos padronizados como IEEE 754 [43]. Ao **double** pode-se aplicar apenas o **long**. E ao **char** só se aplicam os qualificadores de sinal: com sinal (**signed**) ou sem sinal (**unsigned**).

A linguagem C exige que os programas contenham declarações que especifiquem os tipos e qualificadores de dados das variáveis que serão utilizadas ao longo dos programas. Ao longo de um programa é possível converter os tipos de dados declarados implícita e explicitamente [41]. Por exemplo, uma operação envolvendo operandos com um mesmo tipo de dado, tem como resultado o mesmo tipo de dado. Um exemplo é **a divisão de dois números inteiros, como ⅔, que em C apresenta 0 como resposta**, ao invés de um número decimal 0.6666...

Em operações que contêm operandos de tipos diferentes, os valores dos operandos são **convertidos implicitamente** para o mesmo tipo antes da operação ser executada. Tipos mais simples são promovidos implicitamente para tipos mais complexos. Usualmente, os operandos têm os seus tipos convertidos para o tipo comum dos operandos de acordo com a seguinte ordem de precedência: int → unsigned int → float → double → long double, como mostra Fig. 5.52. Quando os operandos de uma operação aritmética são do tipo char ou short, eles são implicitamente convertidos para o tipo int ou unsigned int antes da execução da operação.

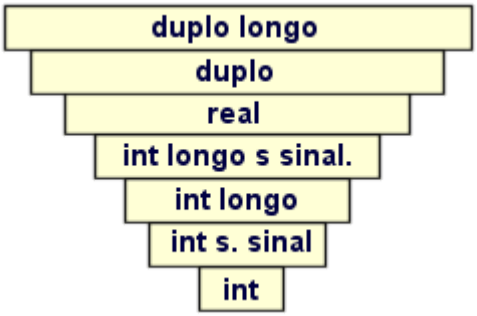


Figura 5.52*:* Hierarquia dos tipos de dados (Fonte: [42])

Ao atribuir o resultado de uma operação a uma outra variável de tipo diferente, ocorre também uma **conversão implícita** do tipo de dado do lado direito do sinal de atribuição para o tipo de dado da variável no lado esquerdo. Para o caso de divisão mostrado anteriormente, a variável no lado esquerdo do sinal de atribuição receberá 0.0 se for do tipo **float**.

Além da **conversão implícita**, a linguagem C tem um operador, conhecido por ***casting***, que **altera um tipo de dado explicitamente**. O formato deste operador aplicado sobre o resultado de uma "expressão" é:

(tipo de dado) expressão

O ***casting*** de um tipo tem a maior precedência entre os operadores. Ele é utilizado para que uma variável de um determinado tipo de dado possa ser interpretada como um outro tipo de dado distinto, sem que haja mudança nos tipos de dado das variáveis utilizadas. O processador consegue truncar ou estender automaticamente os *bits* quando o tamanho do espaço de memória variar no procedimento de ***casting***. Este operador é muito aplicado nos programas de sistemas embarcados para fazer a conversão entre os tipos apropriados para maximizar o fluxo de acessos com os periféricos e os tipos processáveis pelos programas disponíveis..

A falta de uma especificação precisa do tipo **int** prejudica a portabilidade de um código. Um código que funciona bem num microprocessador de 16 *bits* (tipo **int** tem tamanho de 16 *bits*) com os resultados das operações truncados/aproximados em 16 *bits* pode apresentar outros resultados quando o processamento ocorre num processador de 32 *bits* (tipo **int** ocupa 32 *bits*). Para que os resultados sejam invariantes em relação à representação natural de cada processador, faz-se uso da diretiva **typedef** para redefinir os tipos de dados de acordo com o tamanho de espaço efetivo de memória ocupado e não de acordo com o tipo nativo de cada processador.

Tipos redefinidos mais populares adotam a seguinte convenção: **s** para representar se o tipo é estático ou não, **v** para representar se o tipo é volátil ou não, **u** para representar "sem sinal" (*unsigned*) e **intx** para representar um valor inteiro ocupando um espaço de x *bits*. É comum ainda encontrar no nome o sufixo "t", denotando "*type*". Seguem-se algumas redefinições encontradas em projetos de

software de sistemas embarcados:

```
#typedef int8_t signed char ;
#typedef uint8_t unsigned char ;
#typedef vint8_t volatile signed char ;
#typedef vint8_t volatile signed char ;
#typedef vuint8_t volatile unsigned char ;
#typedef int16_t  signed short ;
#typedef uint16_t unsigned short ;
#typedef int32_t signed int ;
#typedef uint32_t unsigned int ;
#typedef sint32_t static signed int ;
#typedef suint32_t static unsigned int ;
```

5.9 Exercícios

1. O que são transistores de portas flutuantes? Quais são as duas tecnologias associadas a estes transistores aplicadas na fabricação de memórias ROM programáveis e memórias FLASH?
2. Quais são as principais diferenças entre as memórias EPROM, EEPROM e FLASH?
3. Em termos de organização, qual é a principal diferença entre uma FLASH NOR e uma FLASH NAND em termos de acessos às células? Qual delas tem acessos de leitura mais rápidos? Qual delas tem acessos de escrita mais rápidos?Justifique em termos da organização das células?
4. Dadas as memórias sRAMs, DRAMs, *Flash* e EPROMs. Qual delas é mais densa em termos de capacidade de armazenamento por área? Qual delas é mais rápida em termos de transferência de bytes por segundo? Qual delas é a mais lenta? Qual delas é a mais recente? Qual delas é a mais volátil? Justifique em termos da tecnologia usada.
5. Em relação ao projeto de uma interface com os módulos de memória de um sistema computacional, por quê o projeto de interface com DRAMs é mais complexo do que com as sRAMs?
6. Por quê as memórias DRAMs precisam de regenerações? Por quê a técnica de regeneração *RAS-only* é menos usada do que a técnica *CAS-before-**RAS*? [14]
7. Qual é a diferença básica entre uma memória DRAM *single data rate* e *double data rate*?
8. Consulte a folha de dados da memória FLASH NOR SST39SF010A [50] e responda:
    a. qual é a tensão de alimentação? quais são as correntes máximas no ciclo de leitura e nos ciclos de gravação?
    b. qual é o tempo típico de leitura?

c. quais são os tempos típicos de escrita (por *byte*, por *setor* e por módulo) ?
d. quantos pinos são alocados para endereços e quantos bits para dados? São condizentes com a especificação da capacidade de armazenamento da memória 1Mbit?
e. a diferenciação das três diferentes formas de gravação, por *byte*, por setor e por módulo, se dá por sequências de comandos em *software*. Quais são estas sequências de comandos?
f. Qual é o tempo que se deve manter os dados no barramento enquanto os endereços são alterados no barramento de endereços durante o ciclo de leitura?
g. Durante a programação dos bytes, a temporização pode ser controlado pelo sinal \WE (*write enable*) ou \CE (*chip enable*). Quais são os tempos de *setup* e *hold* destes sinais nos ciclos de gravação? Quais são os tempos de *setup* e *hold* dos dados nos pinos DQx?
h. há dois *bits* de estado de operação da memória FLASH: *polling bit* (indica a finalização de um ciclo de gravação) e *toggle bit* (indica o estado de leitura da memória FLASH). Através de quais *bits* de dados se consegue obter estas informações?

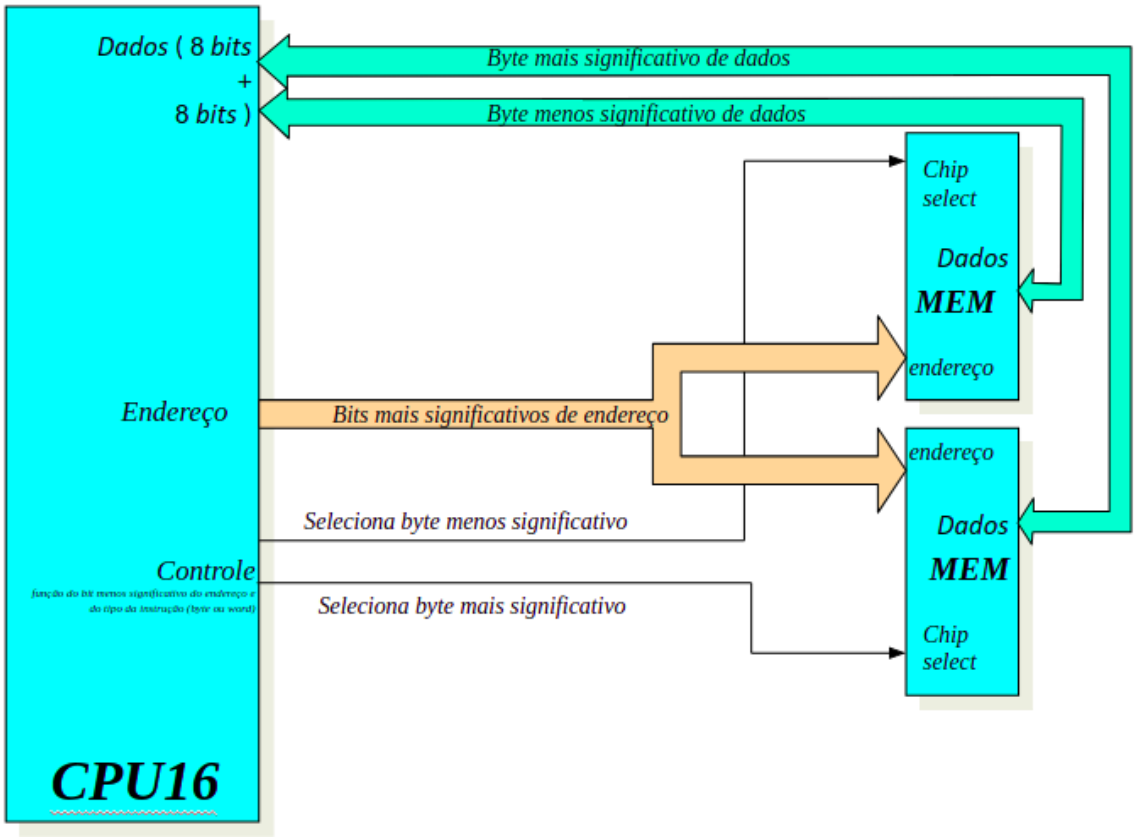
9. Compare os diagramas de tempo das Figuras 5.19 e 5.21 e destaque as diferenças de um ciclo de leitura de uma palavra de um acesso no modo de página.
10. Compare os diagramas de tempo das Figuras 5.21 e 5.22. Destaque as diferenças nos sinais \OE e D_{out} . Com base nestas diferenças, explique por memórias DRAM EDO são mais rápidas nos acessos por página do que as DRAMs sem EDO.
11. Qual é a principal diferença entre um cartão de memória SD e um *pendrive*?
12. A interface da USB é bem simples: dois pinos de alimentação e dois pinos diferenciais de dados. Qual é a função dos sinais diferenciais de dados?
13. Há várias versões de interface USB, diferindo em velocidade e potência dos sinais. Há ainda diferentes conectores para USB. Todas as versões suportam o conector do tipo A. Isso significa que as características elétricas do conector do tipo A são iguais para todas as versões?
14. É fato ou fake: “O disco de estado sólido (SSD) inicializa um sistema computacional muito mais rápido em comparação com um disco rígido (HD) convencional”?
15. Supondo que a frequência da CPU seja 3.1 GHz e o barramento FSB opere até a frequência 1333MHz. Qual memória você compraria para que o seu sistema computacional apresente um melhor desempenho? Uma memória DDR3 32GB a 800MHz ou uma memória 16GB a 1333MHz? Justifique.
16. Um projetista de um sistema de memória usou a técnica de decodificação de endereços parcial para mapear 3 módulos de 1Mbytes de RAM a partir dos

endereços 0x20 0000, 0x40 0000 e 0x80 0000, conectando diretamente os *bits* de endereços A21, A22 e A23 ao pino \CS (*chip select*) do primeiro, segundo e terceiro módulo através de portas inversoras. Qual é o problema deste circuito decodificador? [14]

17. Explique a seguinte afirmação “Os projetistas de um decodificador de endereços procuram usar mais *bits* mais significativos na decodificação parcial para facilitar expansões futuras.” considerando o mapa dos seguintes módulos de memória

    ROM1 2^{10} *bytes* a partir do endereço 0x00 0000
    RAM 2^{10} *bytes* a partir do endereço 0x20 0000
    ROM2 2^{13} *bytes* a partir do endereço 0x40 0000
    I/O_1 1 *byte* a partir do endereço 0x80 0000
    I/O_2 1 *byte* a partir do endereço 0xC0 0000

18. Dadas uma CPU de 16 *bits* e duas pastilhas de memória idênticas de 8 *bits*, cujas conexões são esquematizadas no diagrama de blocos abaixo. Cada instrução é representada por 2 *bytes*. Considerando que o barramento de endereços tenham 8 *bits*, o barramento de dados 16 *bits* e o espaço de endereçamento da CPU seja 0x0000 a 0xFFFF. Projete um decodificador de endereços que permita fazer acessos aos dados de 16 *bits* endereçados pela CPU num único ciclo de memória.
    a. O valor <endereço> deve ser igual ou diferente? O que você entende por “*bits* mais significativos de endereços”? Faz sentido para o seu projeto?
    b. De onde deve ser derivado o sinal “Selecione *byte* menos significativo”?
    c. De onde deve ser derivado o sinal “Selecione *byte* mais significativo”?
    d. Como se pode gerar um sinal de controle que acesse uma palavra de 2 *bytes*?



19. Projete um decodificador de endereços completo para os seguintes mapas de memória [14]:
    a. RAM1 0x00 0000 - 0x00 FFFF
       RAM2 0x01 0000 - 0x03 FFFF
       I/0_1 0xE8 0000 - 0xE8 001F
       I/O_2 0xE8 0000 - 0xE8 003F
    b. ROM1 0x00 0000 - 0x00 3FFF
       ROM2 0x00 4000 - 0x00 7FFF
       ROM3 0x00 8000 - 0x00 BFFF
       RAM 0x04 0000 - 0x07 FFFF
       I/O 0x08 0000 - 0x0800FF
    c. ROM 0x00 0000 - 0x07 FFFF
       RAM 0x80 0000 - 0x9F FFFF
       I/O 0xA0 0000 - 0xA0 0FFF
20. Projete um decodificador de endereços parcial para os seguinte mapas de memória [14]:
    a. ROM 0x00 0000 - 0x03 FFFF
       RAM 0x06 0000 - 0x06 FFFF
       I/O 0x07 0000 - 0x07 0FFF
    b. 2Mbytes de ROM a partir de endereço 0x00 0000 usando módulos de 2M x 4

512 Kbytes de RAM a partir do endereço 0x80 0000 usando módulos de 128K x 8

c. 4 Mbytes de ROM a partir do endereço $0x00 0000 usando módulos de 1M x 8
   8 Mbytes de RAM a partir do endereço 0x80 0000 usando módulos de 4M x 4
   1 Mbytes de RAM a partir do endereço 0x60 0000 usando 512K x 8

21. Quando se executa um programa cujo tamanho seja maior do que o espaço de endereços da memória principal, o que você poderia fazer para atenuar o problema da queda de desempenho pela frequência de paginação?
22. Quais são os tipos de dados suportados pela linguagem C? Como estes tipos de dados são alinhados com os espaços da memória em termos de *bytes*?
23. Por quê o uso de qualificadores unsigned pode “instruir” o compilador a otimizar o uso do espaço de memória para representar o domínio de valores de uma aplicação?

4.8 Referências

[1] http://www.technologystudent.com/comps/comp2.htm
[2] Petros Koutoupis.Data in a Flash, Part IV: the Future of Memory Technologies. https://www.linuxjournal.com/content/data-flash-part-iv-future-memory-technologies
[3] https://www.yourdictionary.com/local-bus
[4] Jacob Andrew. What Combination of Processor Speed Memory & Hard Drive Capacity Will Give the Best Performance? https://smallbusiness.chron.com/combination-processor-speed-memory-hard-drive-capacity-give-performance-70115.html
[5] ARK-ENG course. Implementation of ROM (read-only) semiconductor memories. https://edux.pjwstk.edu.pl/mat/264/lec/main92.html
[6] https://electricalfundablog.com/read-only-memory-rom/
[7] Hackaday. Making a diode matrix rom. https://hackaday.com/2013/10/18/making-a-diode-matrix-rom/
[8] Electricalfundablog. Read Only Memory (ROM) - Working, Types, Applications, Advantages e Disadvantages. https://electricalfundablog.com/read-only-memory-rom/
[9] Cypress. CY7C281A CMOS 1kx8 PROM. http://pdf.datasheetcatalog.com/datasheet2/a/0a91lwhqh7h8l9929lly1q9f057y.pdf
[10] Microchip. 27C64 64K (8K x 8) CMOS EPROM. https://www.jaapsch.net/psion/pdffiles/Eprom008k_datasheet_27C64.pdf
[11] Wikipedia. Read-only Memory. https://en.wikipedia.org/wiki/Read-only_memory
[12] Atmel. AT28C256 256K. http://ww1.microchip.com/downloads/en/DeviceDoc/doc0006.pdf

[13] Wikipedia. Flash memory. https://en.wikipedia.org/wiki/Flash_memory
[14] Alan Clements. Microprocessor Systems Design: 68000 Hardware, Software, and Interfacing. ISBN 0-534-94822-7
[15] Vei-Han Chan e D.K.Y. Liu. An enhanced erase mechanism during channel Fowler-Nordheim tunneling in flash EPROM memory devices. Março, 1999. 10.1109/55.748914.
[16] Avinash Aravindan. Flash 101: NAND Flash vs NOR Flash. https://www.embedded.com/design/prototyping-and-development/4460910/Flash-101--NAND-Flash-vs-NOR-Flash
[17] Motorola. MC6264C. http://users.ece.utexas.edu/~valvano/Datasheets/MCM6264.pdf
[18] Siemens. HYB514256BJL. http://pdf.datasheetcatalog.com/datasheet/siemens/HYB514256BJL-50.pdf
[19] Macronix. MX29F800C. http://www.macronix.com.tw/Lists/Datasheet/Attachments/7650/MX29F800C%20T-B,%205V,%208Mb,%20v1.3.pdf
[20] Computer Hope. Computer memory history. https://www.computerhope.com/history/memory.htm
[21] Wikipedia. Unidade de disco rígido. https://pt.wikipedia.org/wiki/Unidade_de_disco_r%C3%ADgido
[22] Wikipedia. Cylinder-head-sector. https://en.wikipedia.org/wiki/Cylinder-head-sector
[23] Wikipedia. Zone bit recording. https://en.wikipedia.org/wiki/Zone_bit_recording
[24] David A. Patterson, Garth A. Gibson e Randy H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). SIGMOD de 1988: pp. 109–16.
[25] Wikipedia. RAID. https://pt.wikipedia.org/wiki/RAID
[26] Wikipedia. Platform Controller Hub. https://en.wikipedia.org/wiki/Platform_Controller_Hub
[27] Wikipedia. ATA. https://pt.wikipedia.org/wiki/ATA
[28] Wikipedia. Serial ATA. https://pt.wikipedia.org/wiki/Serial_ATA
[29] Wikipedia. SCSI. https://pt.wikipedia.org/wiki/SCSI
[30] Wikipedia. Serial Attached SCSI. https://en.wikipedia.org/wiki/Serial_Attached_SCSI
[31] Wikipedia. Fiber Channel. https://en.wikipedia.org/wiki/Fibre_Channel
[32] http://ece-research.unm.edu/jimp/310/slides/8086_memory2.html
[33] Motorola. SN54/74LS138. https://ecee.colorado.edu/~mcclurel/sn74ls138rev5.pdf
[34] http://research.cs.tamu.edu/prism/lectures/mbsd/mbsd_l16.pdf
[35] https://en.wikipedia.org/wiki/Memory_management_unit#cite_note-TanenMOS-1
[36] Tanenbaum, A. S. Sistemas Operacionais Modernos. ISBN: 978-8543005676. Pearson Universidade. 2015
[37] EEEGuide.com. Addressing Decoding Techiques in 8086 Microprocessors. http://www.eeeguide.com/address-decoding-techniques-in-8086-microprocessor/
[38] https://www.youtube.com/watch?v=2QyKVU1IRZg
[39] https://www.youtube.com/watch?v=dZSZQd8uCjU
[40] http://www.demic.fee.unicamp.br/~alexdant/aularom.pdf
[41] Robson R. Linhares. Aspectos básicos de linguagem C. http://www.dainf.cefetpr.br/~robson/prof/common/c/aspec.htm

[42] Técnicas de Linguagem de Programação. Conceitos Gerais - Tipos de dados. http://www.prof2000.pt/users/famaral/ig/tlp/variaveis.htm
[43] Wikipedia. IEEE754. https://en.wikipedia.org/wiki/IEEE_754
[44] OpenLabPro. Interfacing Microcontrollers with SD Card. https://openlabpro.com/guide/interfacing-microcontrollers-with-sd-card/
[45] Redação Olhar Digital. A diferença entre USB 2.0, 3.0, Tipo-A, Tipo-C e outros formatos e padrões. https://olhardigital.com.br/dicas_e_tutoriais/noticia/a-diferenca-entre-usb-2-0-3-0-tipo-a-tipo-c-e-outros-formatos-e-padroes/83296
[46] D. Mohankumar. USB - How it works. https://www.electroschematics.com/4856/usb-how-things-work/
[47] Miguel Leiva-Gomez. SD Card vs. SSD: What is really the difference? https://www.maketecheasier.com/sd-card-vs-ssd/
[48] Tom Brant. SSD vs. HDD: What is the difference? https://www.pcmag.com/article/297758/ssd-vs-hdd-whats-the-difference
[49] Zachary Painter. NAND Flash Memory Technology: The Basics of a Flash Memory Cell. https://blog.silicon-power.com/index.php/guides/nand-flash-memory-technology-basics/
[50] Folha de dados de SST39SF010A/SST39SF020A/SST39SF040. http://ww1.microchip.com/downloads/en/DeviceDoc/20005022C.pdf
[51] http://edwardbosworth.com/My5155Textbook_HTM/MyText5155_Ch08_V06.htm
[52] Motorola. MC68000 Technical Summary. http://datasheets.chipdb.org/Motorola/68000/mc68000.pdf
[53] Cypress. CY6116A CY6117A datasheet. https://docs.isy.liu.se/pub/VanHeden/DataSheets/6116.pdf