

# QXD0133 - Arquitetura e Organização de Computadores II



**Universidade Federal do Ceará - Campus Quixadá**

Thiago Werlley  
thiagowerlley@ufc.br

18 de outubro de 2025

Capítulo 14

## Capítulo 14

# Memory Management Units

# Gerenciamento sem abstração

0	16380
---	-------

⋮

ADD	28
MOV	24
	20
	16
	12
	8
	4
JMP 24	0

# Gerenciamento sem abstração

0	16380
⋮	
ADD	28
MOV	24
	20
	16
	12
	8
	4
JMP 24	0 ←

# Gerenciamento sem abstração

0	16380
---	-------

⋮

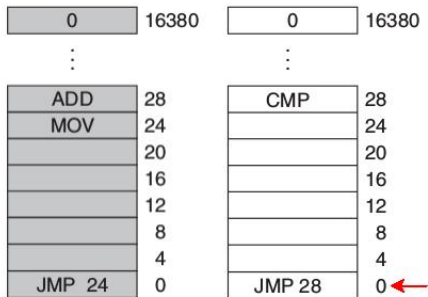
ADD	28
MOV	24
	20
	16
	12
	8
	4
JMP 24	0



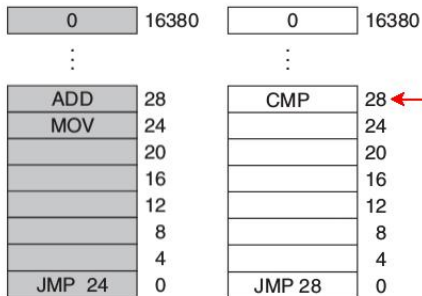
# Gerenciamento sem abstração

0	16380	0	16380
⋮		⋮	
ADD	28	CMP	28
MOV	24		24
	20		20
	16		16
	12		12
	8		8
	4		4
JMP 24	0	JMP 28	0

# Gerenciamento sem abstração

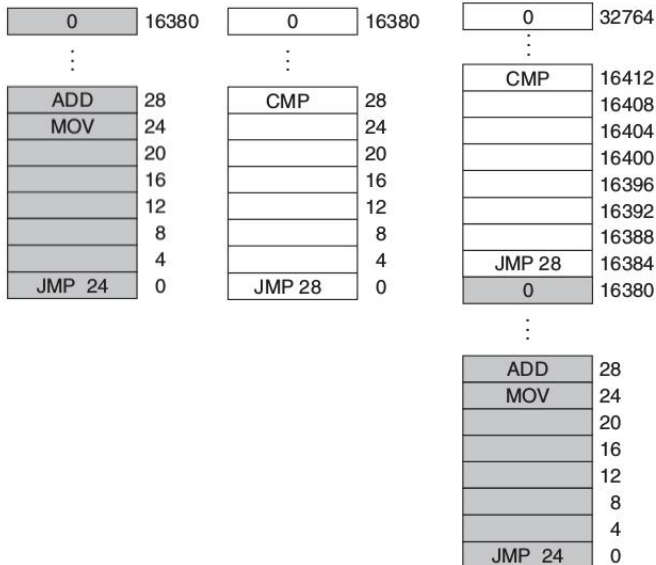


# Gerenciamento sem abstração

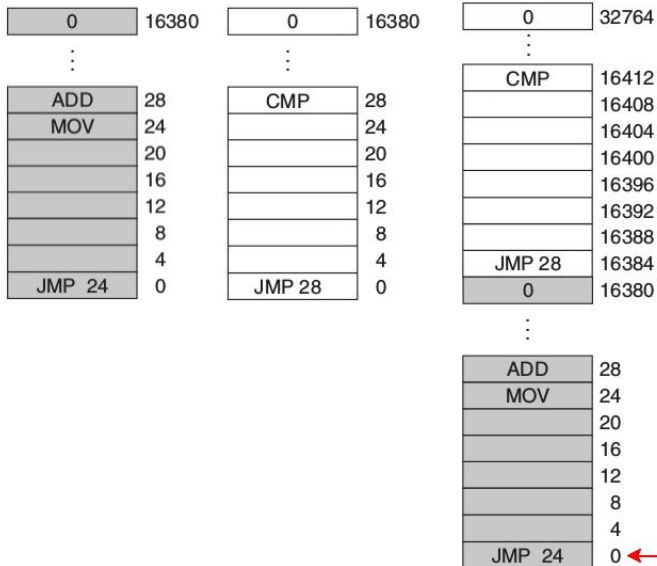




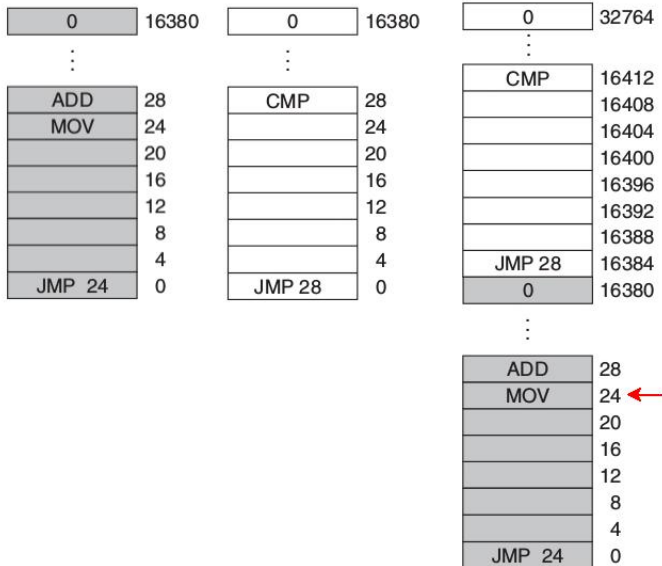
# Gerenciamento sem abstração



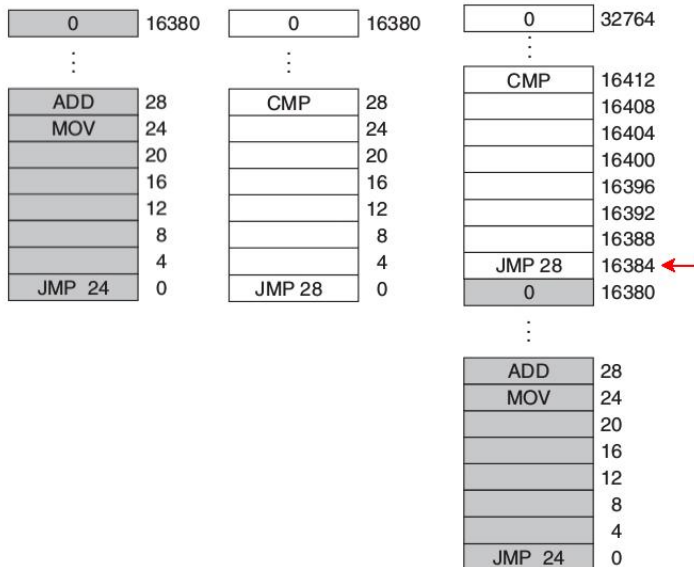
# Gerenciamento sem abstração



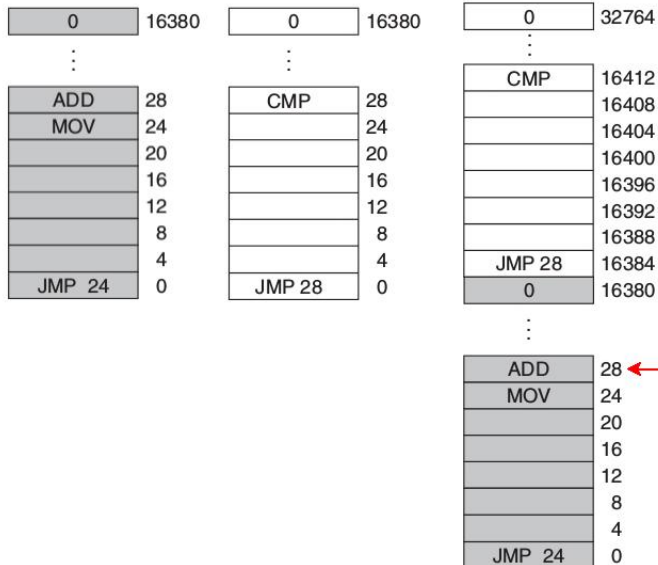
# Gerenciamento sem abstração



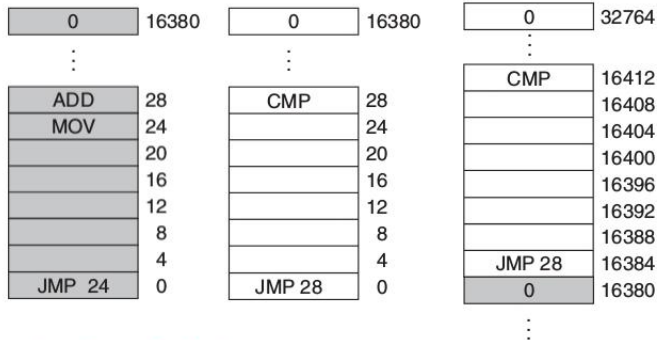
# Gerenciamento sem abstração



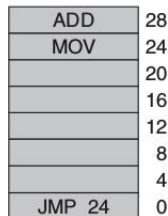
# Gerenciamento sem abstração



# Gerenciamento sem abstração



Ambos fazem referência absoluta de memória



# Gerenciamento sem abstração

0	16380	0	16380	0	32764
⋮		⋮		⋮	
ADD	28	CMP	28	CMP	16412
MOV	24		24		16408
	20		20		16404
	16		16		16400
	12		12		16396
	8		8		16392
	4		4		16388
JMP 24	0	JMP 28	0	JMP 28	16384
				0	16380
				⋮	

Ambos fazem referência absoluta de memória

Solução: adicionar 16384 a todos os endereçamentos do segundo programa

ADD	28
MOV	24
	20
	16
	12
	8
	4
JMP 24	0

# Gerenciamento sem abstração

0	16380	0	16380	0	32764
⋮		⋮		⋮	
ADD	28	CMP	28	CMP	16412
MOV	24		24		16408
	20		20		16404
	16		16		16400
	12		12		16396
	8		8		16392
	4		4		16388
JMP 24	0	JMP 28	0	JMP 28	16384
				0	16380
				⋮	

Ambos fazem referência absoluta de memória

Solução: adicionar 16384 a todos os endereçamentos do segundo programa

**REALOCAÇÃO ESTATICA**

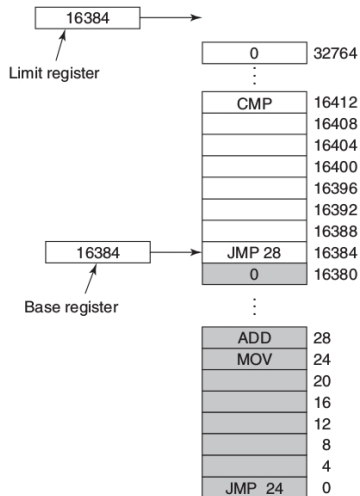
ADD	28
MOV	24
	20
	16
	12
	8
	4
JMP 24	0



## Registradores-base e registradores-limite

- Solução simples da **realocação dinâmica**
- O que ela faz é mapear o **espaço de endereçamento** de cada processo em uma parte diferente da **memória física**
- Dois registradores de hardware especiais
  - Registradores-base
  - Registradores-limite
- Cada referência à memória tem acrescido o valor base
  - Executado pelo hardware da CPU
  - Também verifica se resultado é menor que limite

# Registradores-base e registradores-limite



- Solução simples que dá a cada processo um espaço exclusivo
- Em alguns casos apenas o SO pode acessar os registradores
- Desvantagem: a cada acesso deve ser executada
  - Operação de adição
  - Operação de comparação

# Memória Virtual

- Endereçamento → Abstração → Registradores-base/-limite
- Como gerenciar softwares que demandam muita memória?
- Possível solução: sobreposição (*overlay*)
  - Gerenciador de sobreposição → Carregado inicialmente
  - Programa dividido em módulos → Sobreposições
  - Troca e coexistência de sobreposições: HD ↔ Memória
  - Troca realizada pelo SO
  - Divisão do programa → Programador → + complexidade

# Memória Virtual

- Método concebido por Fotheringham em 1961: **Memória Virtual**
- Generalização do conceito de registradores-base/-limite
- Espaço de endereçamento dividido em blocos: **Páginas**
  - Cada página: série contínua de endereços
  - Mapeadas na memória física
  - Nem todas presentes na memória física
- Quando programa referencia endereçamento:
  - Página mapeada → Hardware executa mapeamento
  - Página não mapeada → SO alertado
- Também apropriado para ambientes com multiprogramação
  - Troca de CPU durante carregamento de página

# Introdução

- Ao criar um sistema embarcado multitarefa, faz sentido **ter uma maneira fácil de gravar, carregar e executar tarefas** de aplicativos independentes.
- Muitos dos sistemas embarcados de hoje **usam um sistema operacional para simplificar esse processo**.
- Os sistemas operacionais mais avançados usam uma **unidade de gerenciamento de memória (MMU)** baseada em **hardware**.
- Um dos **principais serviços fornecidos por uma MMU** é a capacidade de gerenciar tarefas como programas independentes executados em seu próprio espaço de memória privada.

# Introdução

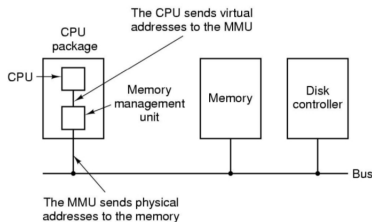
- A MMU fornece os recursos necessários para habilitar a **memória virtual** - um espaço de memória adicional independente da memória física conectada ao sistema.
- A **MMU atua como um tradutor, que converte os endereços de programas e dados compilados para serem executados na memória virtual nos endereços físicos reais** em que os programas são armazenados na memória principal física.
- Esse processo de conversão permite que os programas sejam executados com os mesmos endereços virtuais enquanto são mantidos em locais diferentes na memória física.

# Introdução

- Essa visão dupla da memória resulta em dois tipos de endereços distintos:
  - **endereços virtuais** - são atribuídos pelo compilador e lincador quando localiza um programa na memória
  - **endereços físicos** - são usados para acessar os componentes de hardware reais da memória principal, onde os programas estão localizados fisicamente.

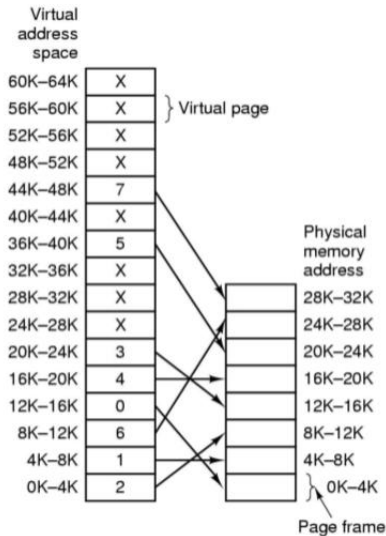
# Paginação

- Técnica utilizada pela maioria dos sistemas com memória virtual
- Cada programa
  - Conjunto de endereços virtuais = Espaço de endereçamento virtual
- Sem memória virtual: Virtual = Físico
  - Ler/Escriver → Direto no barramento de memória
- Com memória virtual
  - Endereço virtual → Memory Management Unit (MMU)
  - MMU: Mapeamento Virtual → Físico



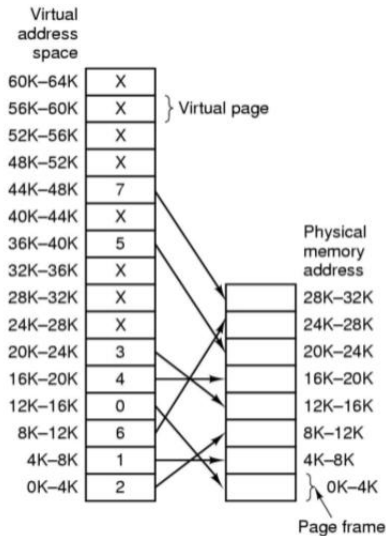


# Paginação: Exemplo



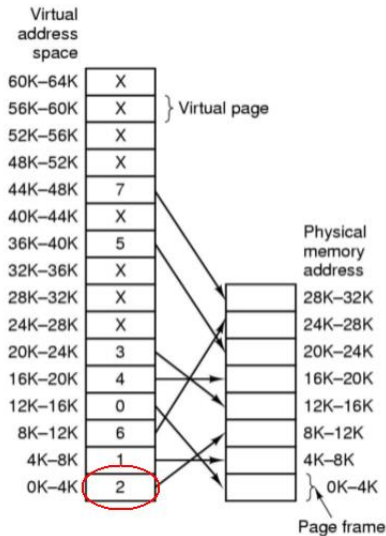
- Endereços virtuais de 16 bits (64KB)
- Apenas 32KB de memória física
- Memória física dividida em molduras
- Memória virtual
  - Divisão em páginas (*pages*)
  - Cada página = 4KB
- Memória física
  - Divisão em molduras (*page frames*)
  - Cada moldura de página = 4KB
- HD ↔ Memória: Páginas completas

# Paginação: Exemplo



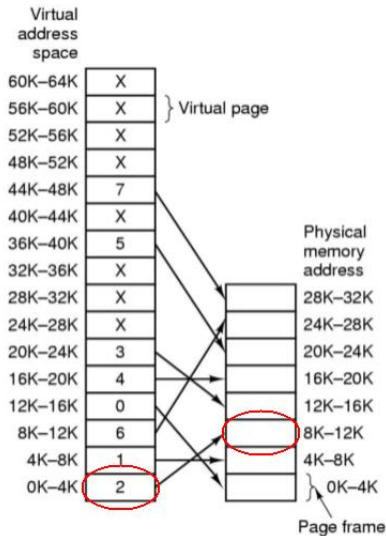
- Instrução: MOV REG,0
  - Endereço 0 → MMU
  - Endereço 0 → Moldura 2
  - Instrução efetiva:  
MOV REG,8192
- Instrução: MOV REG,32780
  - Endereço 32780 → MMU
  - Não mapeada
  - MMU: Falta de página → SO
  - SO salva uma moldura no HD
  - SO carrega página na moldura

# Paginação: Exemplo



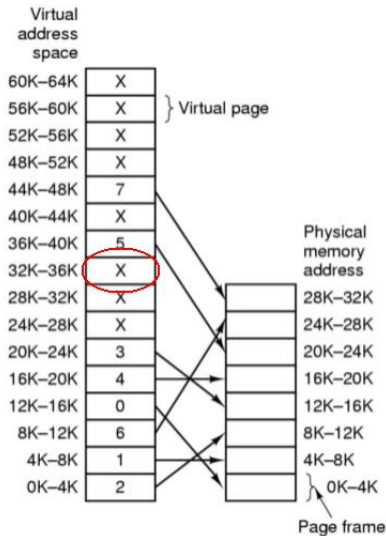
- Instrução: MOV REG,0
  - Endereço 0 → MMU
  - Endereço 0 → Moldura 2
  - Instrução efetiva:  
MOV REG,8192
- Instrução: MOV REG,32780
  - Endereço 32780 → MMU
  - Não mapeada
  - MMU: Falta de página → SO
  - SO salva uma moldura no HD
  - SO carrega página na moldura

# Paginação: Exemplo



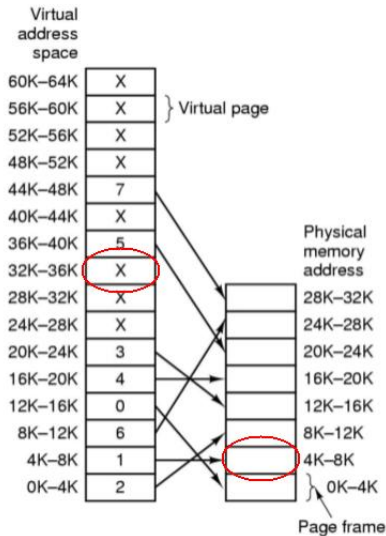
- Instrução: MOV REG,0
  - Endereço 0 → MMU
  - Endereço 0 → Moldura 2
  - Instrução efetiva:  
MOV REG,8192
- Instrução: MOV REG,32780
  - Endereço 32780 → MMU
  - Não mapeada
  - MMU: Falta de página → SO
  - SO salva uma moldura no HD
  - SO carrega página na moldura

# Paginação: Exemplo



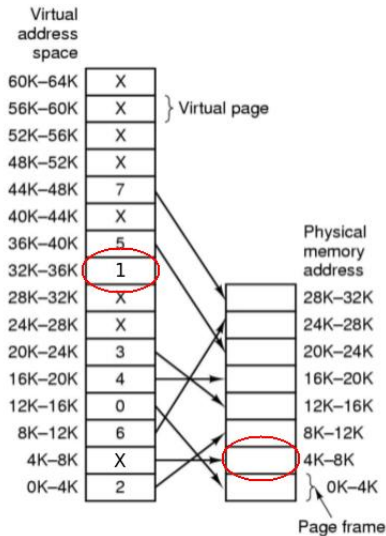
- Instrução: MOV REG,0
  - Endereço 0 → MMU
  - Endereço 0 → Moldura 2
  - Instrução efetiva:  
MOV REG,8192
- Instrução: MOV REG,32780
  - Endereço 32780 → MMU
  - Não mapeada
  - MMU: Falta de página → SO
  - SO salva uma moldura no HD
  - SO carrega página na moldura

# Paginação: Exemplo



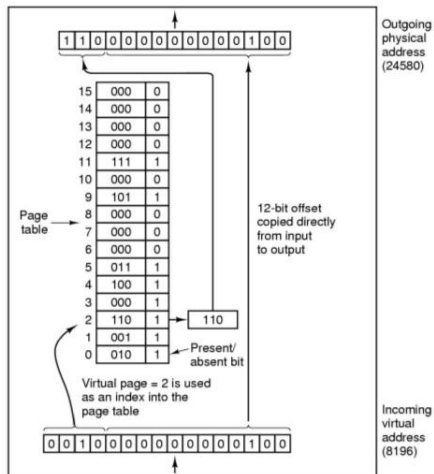
- Instrução: MOV REG,0
  - Endereço 0 → MMU
  - Endereço 0 → Moldura 2
  - Instrução efetiva:  
MOV REG,8192
- Instrução: MOV REG,32780
  - Endereço 32780 → MMU
  - Não mapeada
  - MMU: Falta de página → SO
  - SO salva uma moldura no HD
  - SO carrega página na moldura

# Paginação: Exemplo



- Instrução: MOV REG,0
  - Endereço 0 → MMU
  - Endereço 0 → Moldura 2
  - Instrução efetiva:  
MOV REG,8192
- Instrução: MOV REG,32780
  - Endereço 32780 → MMU
  - Não mapeada
  - MMU: Falta de página → SO
  - SO salva uma moldura no HD
  - SO carrega página na moldura

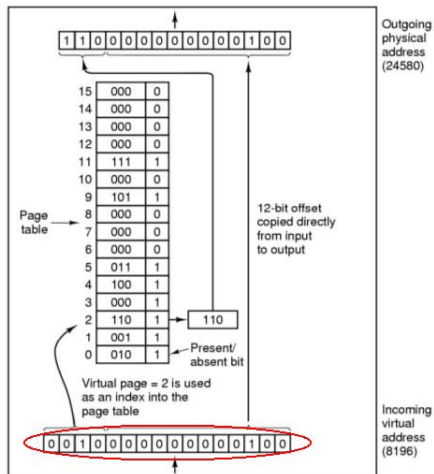
# Funcionamento MMU



- Endereço de 16 bits chega à MMU
- Duas partes:
  - 4 bits → Página (16 páginas)
  - 12 bits → Deslocamento ( $2^{12}$  posições)
- Tabela de páginas:
  - Posição da página
  - Mapeado/Não Mapeado
- Endereço de 15 bits da memória física

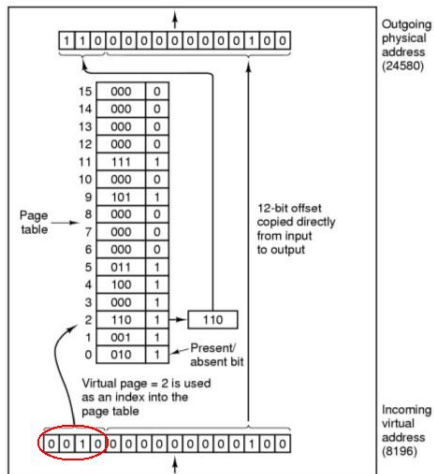


# Funcionamento MMU



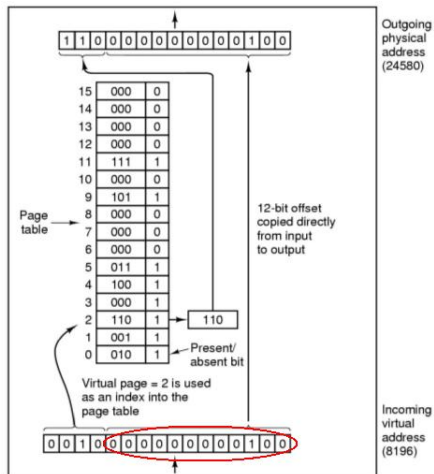
- Endereço de 16 bits chega à MMU
- Duas partes:
  - 4 bits → Página (16 páginas)
  - 12 bits → Deslocamento ( $2^{12}$  posições)
- Tabela de páginas:
  - Posição da página
  - Mapeado/Não Mapeado
- Endereço de 15 bits da memória física

# Funcionamento MMU



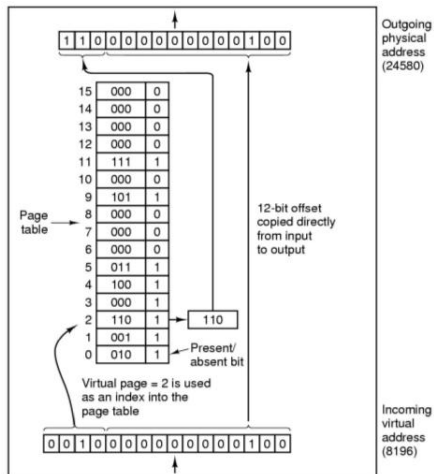
- Endereço de 16 bits chega à MMU
- Duas partes:
  - 4 bits → Página (16 páginas)
  - 12 bits → Deslocamento ( $2^{12}$  posições)
- Tabela de páginas:
  - Posição da página
  - Mapeado/Não Mapeado
- Endereço de 15 bits da memória física

# Funcionamento MMU



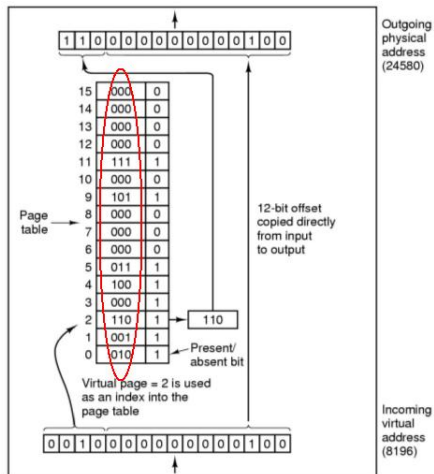
- Endereço de 16 bits chega à MMU
- Duas partes:
  - 4 bits → Página (16 páginas)
  - 12 bits → Deslocamento ( $2^{12}$  posições)
- Tabela de páginas:
  - Posição da página
  - Mapeado/Não Mapeado
- Endereço de 15 bits da memória física

# Funcionamento MMU



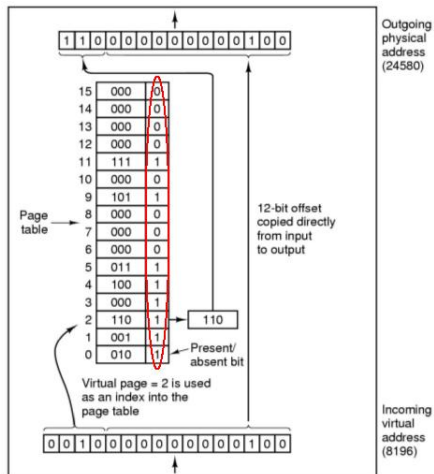
- Endereço de 16 bits chega à MMU
- Duas partes:
  - 4 bits → Página (16 páginas)
  - 12 bits → Deslocamento ( $2^{12}$  posições)
- Tabela de páginas:
  - Posição da página
  - Mapeado/Não Mapeado
- Endereço de 15 bits da memória física

# Funcionamento MMU



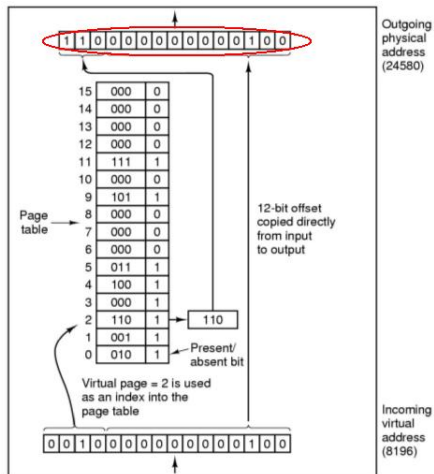
- Endereço de 16 bits chega à MMU
- Duas partes:
  - 4 bits → Página (16 páginas)
  - 12 bits → Deslocamento ( $2^{12}$  posições)
- Tabela de páginas:
  - Posição da página
  - Mapeado/Não Mapeado
- Endereço de 15 bits da memória física

# Funcionamento MMU



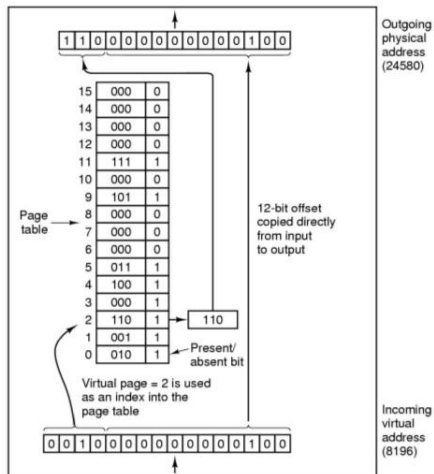
- Endereço de 16 bits chega à MMU
- Duas partes:
  - 4 bits → Página (16 páginas)
  - 12 bits → Deslocamento ( $2^{12}$  posições)
- Tabela de páginas:
  - Posição da página
  - Mapeado/Não Mapeado
- Endereço de 15 bits da memória física

# Funcionamento MMU



- Endereço de 16 bits chega à MMU
- Duas partes:
  - 4 bits → Página (16 páginas)
  - 12 bits → Deslocamento ( $2^{12}$  posições)
- Tabela de páginas:
  - Posição da página
  - Mapeado/Não Mapeado
- Endereço de 15 bits da memória física

# Funcionamento MMU



- Endereço de 16 bits chega à MMU
- Duas partes:
  - 4 bits → Página (16 páginas)
  - 12 bits → Deslocamento ( $2^{12}$  posições)
- Tabela de páginas:
  - Posição da página
  - Mapeado/Não Mapeado
- Endereço de 15 bits da memória física



# MPU para MMU

- Uma região é uma maneira conveniente de organizar e proteger a memória.
- As regiões estão ativas ou inativas:
  - região ativa - contém código ou dados em uso atual pelo sistema;
  - região inativa - contém código ou dados que não estão em uso atual, mas provavelmente se tornarão ativos em pouco tempo.
- Uma região inativa é protegida e, portanto, inacessível à tarefa atual em execução.
- A principal diferença entre uma MPU e uma MMU é a adição de hardware para suportar a memória virtual.

## Como a memória virtual funciona?

- Um sistema embarcado usando MPU compila e executa cada tarefa em áreas de endereços fixas distintamente diferentes na memória principal.
- Em uma MMU, as tarefas podem ser executadas mesmo que sejam compiladas e lincadas para execução em regiões com endereços sobrepostos na memória principal.
- O suporte à memória virtual na MMU permite a construção de um sistema embarcado que possui vários mapas de memória virtual e um único mapa de memória física.

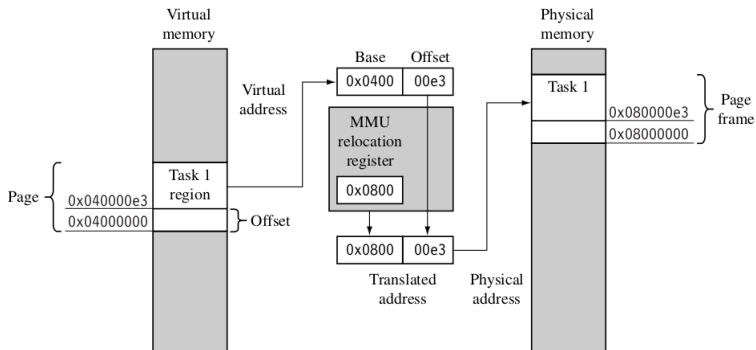
## Como a memória virtual funciona?

- Cada tarefa recebe seu próprio mapa de memória virtual com o objetivo de compilar e linkar o código e os dados que compõem a tarefa.
- Uma camada do kernel gerencia o posicionamento das várias tarefas na memória física para que elas tenham um local distinto na memória física diferente do local virtual no qual foi projetado para executar.
- Para permitir que as tarefas tenham seu próprio mapa de memória virtual, o hardware da MMU executa a **realocação de endereços**, convertendo a saída do endereço de memória pelo núcleo do processador antes de atingir a memória principal.

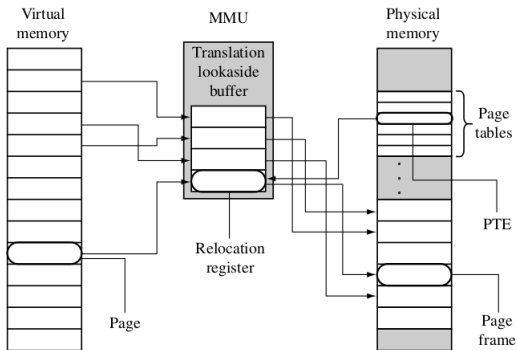
# Como a memória virtual funciona?

- A maneira mais fácil de entender o processo de conversão é imaginar um registrador de realocação localizado na MMU entre o núcleo e a memória principal.
- Quando o núcleo do processador gera um endereço virtual, a MMU pega os bits mais significativos do endereço virtual e os substitui pelo conteúdo do registrador de realocação para criar um endereço físico.
- A parte menos significativa do endereço virtual é um offset que se traduz em um endereço específico na memória física.
- O intervalo de endereços que podem ser traduzidos usando esse método é limitado pelo tamanho máximo do offset do endereço virtual.

# Maapeamento de tarefa na MV para a MF usando um registrador de realocação



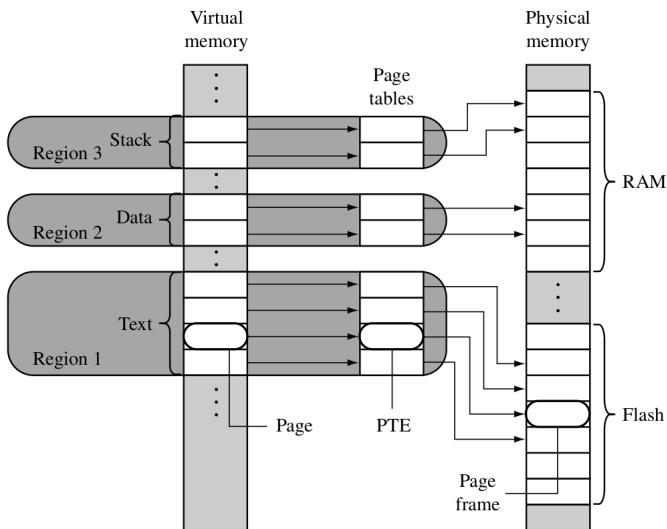
# Os componentes de um sistema de memória virtual



# Definindo regiões usando páginas

- Em uma MMU, as regiões são definidas como grupos de tabelas de páginas e são controladas completamente no software como páginas sequenciais na memória virtual.
- Uma região pode ser definida como um conjunto sequencial de entradas da tabela de páginas.
- O local e o tamanho de uma região podem ser mantidos em uma estrutura de dados de software, enquanto os dados reais de conversão e informações de atributo são mantidos nas tabelas de páginas.

# Maapeamento de páginas à frame de páginas

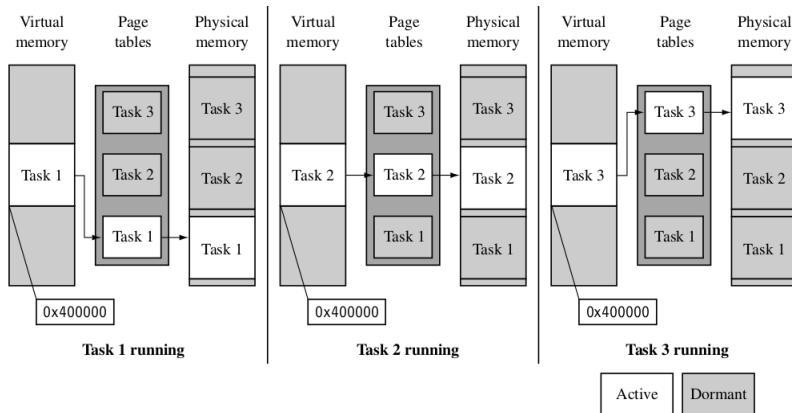




# Multitarefa e MMU

- As tabelas de páginas podem residir na memória e não serem mapeadas para o hardware MMU.
- Uma maneira de criar um sistema multitarefa é criar conjuntos separados de tabelas de páginas, cada um mapeando um espaço de memória virtual exclusivo para uma tarefa.
- Para ativar uma tarefa, o conjunto de tabelas de páginas da tarefa específica e seu espaço de memória virtual são mapeados para uso pela MMU.
- Os outros conjuntos de tabelas de páginas inativas representam tarefas inativas.
- Essa abordagem permite que todas as tarefas permaneçam residentes na memória física e ainda estejam disponíveis imediatamente quando ocorrer uma alternância de contexto para ativá-la.

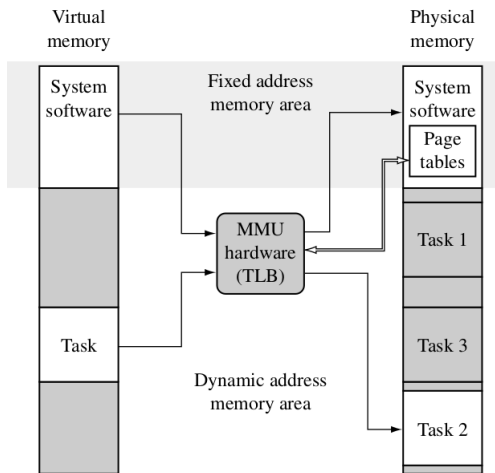
# Memória virtual de um contexto de tarefa do usuário



# Multitarefa e MMU

- Para alternar entre tarefas, são necessárias as seguintes etapas:
  - ➊ Salvar o contexto da tarefa ativa e colocar a tarefa em um estado inativo.
  - ➋ Limpar as caches
  - ➌ Limpar o TLB para remover as traduções da tarefa que está sendo retirada.
  - ➍ Configurar a MMU para usar novas tabelas de páginas que traduzem a área de execução da memória virtual para o local da tarefa acordada na memória física.
  - ➎ Restaurar o contexto da tarefa acordada.
  - ➏ Continuar a execução da tarefa restaurada.

# Organização da memória em um sistema de MV



# Detalhes da MMU ARM

- O MMU do ARM executa várias tarefas:
  - converte endereços virtuais em endereços físicos;
  - controla a permissão de acesso à memória;
  - determina o comportamento individual do cache e o buffer de gravação para cada página na memória.
- Quando a MMU está desativada, todos os endereços virtuais são mapeados individualmente para o mesmo endereço físico.
- Se o MMU não conseguir converter um endereço, ele gera uma exceção de cancelamento.
- A MMU abortará apenas as falhas de conversão, permissão e domínio.

# Tabela de Páginas

- O hardware do ARM MMU possui uma arquitetura de tabela de página multinível. Existem dois níveis de tabela de páginas: nível 1 (L1) e nível 2 (L2).
- Existe uma tabela de página única de nível 1, conhecida como tabela da página principal L1, que pode conter dois tipos de entrada:
  - Ponteiros para o endereço inicial das tabelas de nível 2;
  - Entradas na tabela de páginas para traduzir páginas de 1 MB.

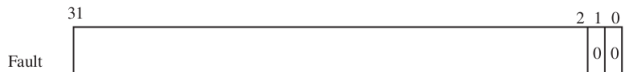
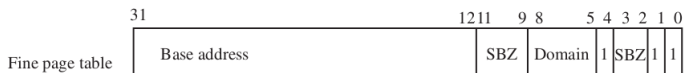
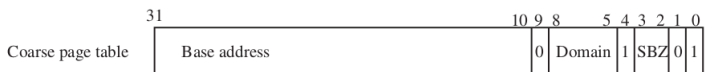
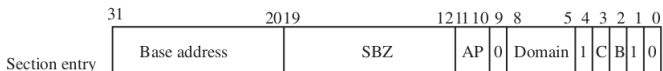
Name	Type	Memory consumed by page table (KB)	Page sizes supported (KB)	Number of page table entries
Master/section	level 1	16	1024	4096
Fine	level 2	4	1, 4, or 64	1024
Coarse	level 2	1	4 or 64	256

# Entradas da tabela de páginas de nível 1

A tabela de páginas de nível 1 aceita quatro tipos de entrada:

- Uma entrada de tradução de seção de 1 MB
- Uma entrada de diretório que aponta para uma tabela de páginas L2 fina (*fine*)
- Uma entrada de diretório que aponta para uma tabela de páginas L2 grosseira (*coarse*)
- Uma entrada de falha que gera uma exceção de cancelamento

# Entradas da tabela de páginas de nível 1



SBZ = should be zero

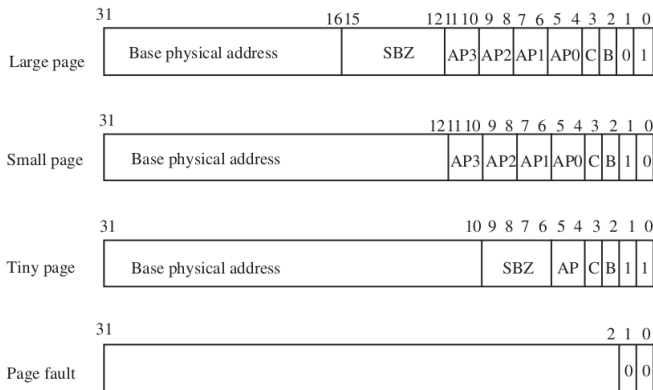


## Entradas da tabela de páginas de nível 2

Existem quatro entradas possíveis usadas nas tabelas de páginas L2:

- Uma entrada de página (*large*) define os atributos para um frame de página de 64 KB.
- Uma entrada de página (*small*) define um frame de página de 4 KB.
- Uma entrada de página (*tiny*) define um frame de página de 1 KB.
- Uma entrada de página de falha gera uma exceção de cancelamento.

# Entradas da tabela de páginas de nível 2



SBZ = should be zero

## Selecionando um tamanho de página para o seu sistema embarcado

- Quanto menor o tamanho da página, mais frames de páginas haverá em um determinado bloco de memória física.
- Quanto menor o tamanho da página, menor a fragmentação interna.
- Quanto maior o tamanho da página, maior a probabilidade de o sistema carregar códigos e dados referenciados.
- Cada tabela de página consome 1KB de memória ao usar o L2 *coarse*. Cada tabela de página do L2 *fine* consome 4KB. Cada tabela de página L2 traduz 1MB de espaço de endereço. Assim, o tamanho máximo de tabela de página usada por tarefa é:

$$((|tarefa|/1\text{MB})+1)*(|tabela\ de\ página\ de\ L2|)$$

# Translation Lookaside Buffer

- O TLB é uma cache especial de traduções de página usadas recentemente.
- O TLB mapeia uma página virtual para um frame de página ativo e armazena dados de controle que restringem o acesso à página.
- O TLB é uma cache e, portanto, possui uma política de substituição de linha TLB.
- Nos processadores ARM, o TLB usa um algoritmo round-robin para selecionar qual registrador de realocação substituir em caso de falha no TLB.
- O TLB suporta dois tipos de comandos: você pode dar um *flash* no TLB e bloquear traduções no TLB.

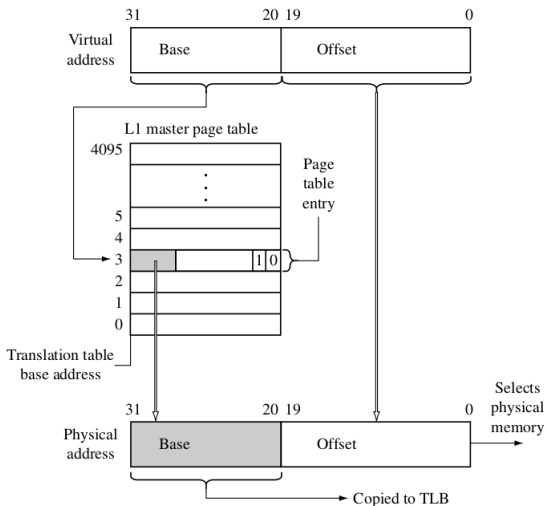
# Translation Lookaside Buffer

- Se o TLB não tiver uma tradução válida, temos um TLB *miss*.
- A MMU manipula automaticamente as falhas do TLB em hardware, pesquisando as tabelas de páginas na memória principal (**caminhada de tabela de páginas**) em busca de traduções válidas e carregando-as em uma das 64 linhas do TLB.
- Se houver um PTE válido, o hardware copia o endereço de conversão do PTE para o TLB e gera o endereço físico para acessar a memória principal.

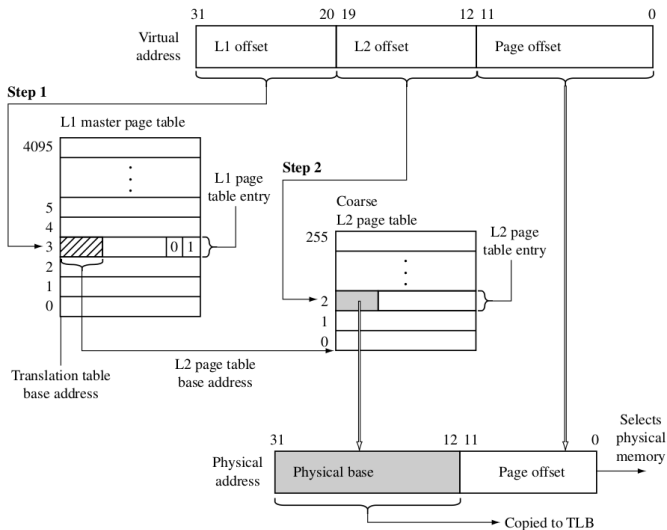
# Translation Lookaside Buffer

- Se, no final da pesquisa, houver uma entrada de falha na tabela da página, o hardware da MMU gerará uma exceção de cancelamento.
- O custo de uma falha é geralmente de um ou dois ciclos de acesso à memória principal.
- O número de ciclos depende da tabela de páginas em que os dados de conversão são encontrados.

# Single-Step Page Table Walk



# Two-Step Page Table Walk





# QXD0133 - Arquitetura e Organização de Computadores II



**Universidade Federal do Ceará - Campus Quixadá**

Thiago Werlley  
thiagowerlley@ufc.br

18 de outubro de 2025

Capítulo 14