

Software Básico

Características

Observações sobre nível de Hardware

Registradores no Nível ISA

Registradores

Registradores Ponteiros

Formato das Instruções

Critérios de projeto

Endereçamento

Endereçamento imediato

Endereçamento direto

Endereçamento de registrador

Endereçamento indireto de registrador

Endereçamento indexado

Endereçamento de base indexado

Endereçamento de pilha

Ortogonalidade e modos de endereçamento

Tipos de Instruções

Instruções de movimentação de dados

Operações diádicas

Operações monádicas

Comparações e desvios condicionais

Controle de laço

Entrada e saída

Fluxo de Controle

Assembly

Entradas para Chamadas de Sistema

Jumpers

Operações Matemáticas

Macros

Definição

Macros x Chamada

Características

Parâmetros

Processador de Macros

Resumo

Exemplo Macro

O montador

Primeira passagem

Segunda Passagem

Tabela de Símbolos

A TABELA HASH

Ligação e Carregamento

Resolvendo a relocação e a referência externa

Módulo Objeto

Ligação em duas passagens

Tempo de vinculação e relocação dinâmica

Ligação Dinâmica

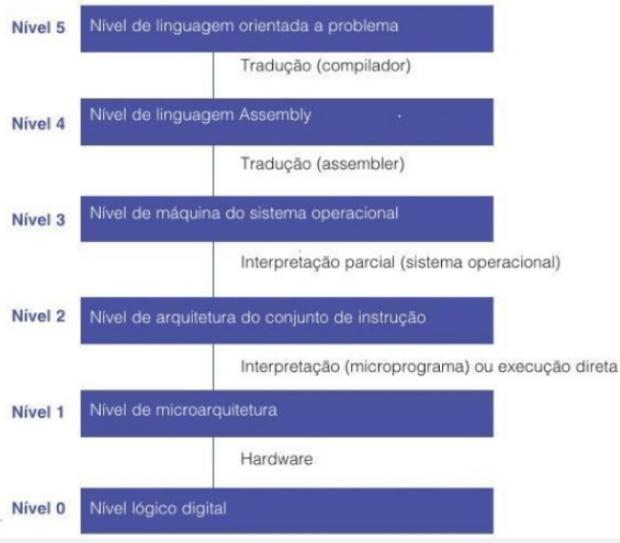
A LIGAÇÃO DINÂMICA NO MULTICS

Software Básico

Características

- Softwares escritos em linguagem de montagem em geral consomem menos memória e tempo de processamento que softwares escritos em outras linguagens, embora a complexidade de tempo desses softwares também possa ser um fator determinante
- A linguagem de montagem é mais poderosa que outras linguagens de alto nível, ou seja, pode fazer coisas que as linguagens de alto nível não podem
- Softwares desse tipo são mais difíceis de programar, e de concertar. Por exemplo, gasta-se mais tempo para se resolver um problema usando-se a linguagem de montagem do que usando-se uma linguagem de alto nível

NÍVEIS DE ABSTRAÇÃO



- Em qual nível está o software básico?
- Em que nível se desenvolve um sistema de informação?
- Onde fica a separação entre software e hardware?

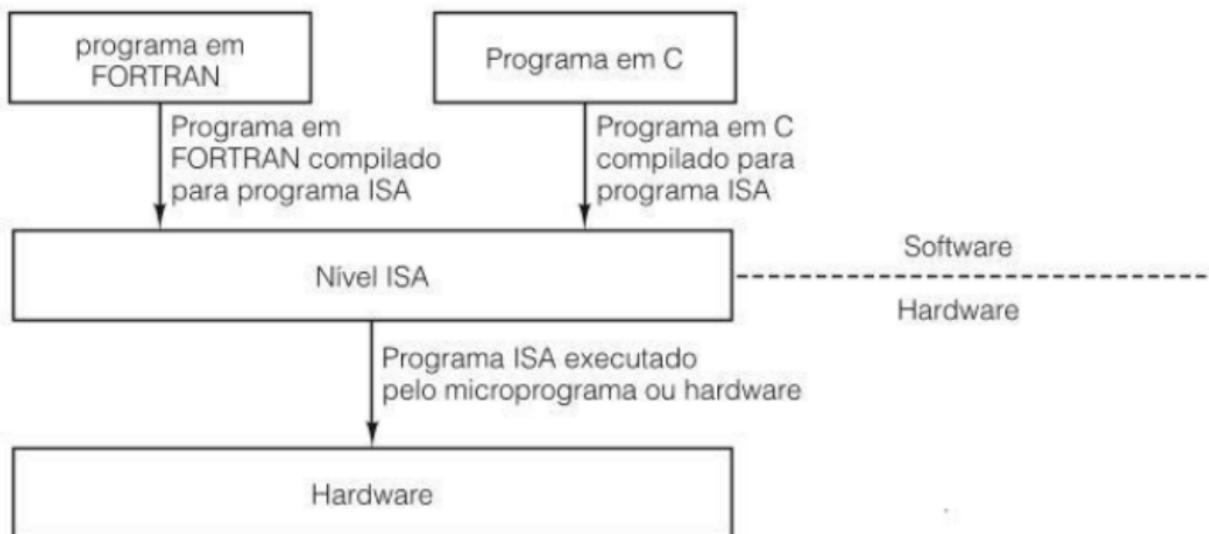
Observações sobre nível de Hardware

- **O hardware real.** Isso consiste em transistores configurados como circuitos lógicos.
- **O nível do micropograma.** Microinstruções controlam o fluxo de dados através da máquina e controlam ativação do hardware real (por exemplo, diga ao circuito ADDER para somar dois números). Uma série de microinstruções, chamadas de micropograma, são usadas para implementar cada uma das instruções na máquina conjunto de instruções da linguagem. A microprogramação permite ao

projeta implementar facilmente um conjunto específico de máquinas instruções e é, portanto, um conceito-chave na construção de uma família compatível de computadores.

- **O conjunto de instruções da máquina.** As instruções mais básicas que o computador pode entender e executar.
- **Linguagem assembler simbólica.** Uma linguagem de programação simbólica cujas instruções têm um para um correspondência com uma linguagem de máquina específica.
- **O sistema operacional.** O sistema operacional tem a função de gerenciar e alojar os recursos do computador (tempo de CPU, dispositivos de E / S, arquivos de dados) para os usuários.
- **Idiomas de Alto Nível.** Como C ++ ou Java ou pacotes de aplicativos, como gerenciadores de bancos de dados

O NÍVEL ISA



- Um limiar de separação entre hardware e software

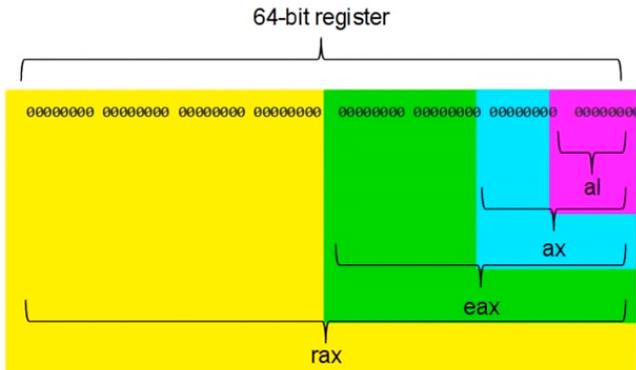
Registradores no Nível ISA

- Categorias
 - Uso geral e uso específico
- Program Status Word (PSW)
 - É o registrador de flags
 - Vários de seus bits têm função específica
 - N – Marcado quando o resultado é negativo
 - Z – Marcado quando o resultado é zero
 - V – Marcado quando ocorre um overflow

- C – Marcado quando ocorre um carry
- P – Marcado quando o resultado é par

Registradores

Registers



8-bit	16-bit	32-bit	64-bit
al	ax	eax	rax
bl	bx	ebx	rbx
cl	cx	ecx	rcx
dl	dx	edx	rdx
sil	si	esi	rsi
dil	di	edi	rdi
bpl	bp	ebp	rbp
spl	sp	esp	rsp
r8b	r8w	r8d	r8
r9b	r9w	r9d	r9
r10b	r10w	r10d	r10
r11b	r11w	r11d	r11
r12b	r12w	r12d	r12
r13b	r13w	r13d	r13
r14b	r14w	r14d	r14
r15b	r15w	r15d	r15

Registradores	Função
EAX	Principal registrador aritmético
EBX	Ponteiros
ECX	Execução de laços
EDX	Multiplicação e divisão
ESI e EDI	Ponteiros de origem e de destino de cadeias de dados
EBP	Extended Base Pointer
ESP	Extended Stack Pointer
De CS a GS	Segmentos
EIP	Extended Instruction Pointer
EFLAGS	Program Status Word (PSW)

Registradores Ponteiros

Ponteiros também são registradores que armazenam dados. Eles "pontam" para dados, ou seja, armazenam seu endereço de memória.

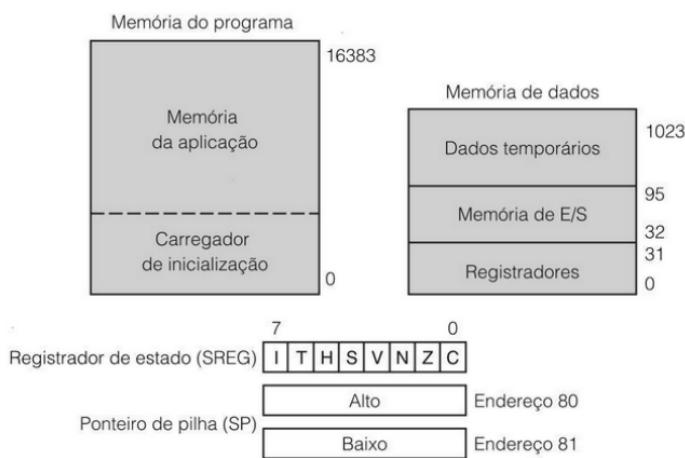
Nome do Ponteiro	Significado	Descrição
rip (eip, ip)	Index Pointer	Aponta para o próximo endereço a ser executado no fluxo de controle.
rsp (esp, sp)	Stack Pointer	Aponta para o endereço topo da stack.
rbp(ebp, bp)	Stack Base Pointer	Aponta para a base da stack.
...

REGISTRADORES DO OMAP4430

Registrador	Nome alternativo	Função
R0–R3	A1–A4	Mantém parâmetros para o procedimento que está sendo chamado
R4–R11	V1–V8	Mantém variáveis locais para o procedimento atual
R12	IP	Registrador de chamada dentro do procedimento (para chamadas de 32 bits)
R13	SP	Ponteiro de pilha
R14	LR	Registrador de ligação (endereço de retorno para função atual)
R15	PC	Contador de programa

REGISTRADORES E MEMÓRIA DO ATMEGA168

- O registrador SREG



- Um bit de habilitação de interrupção
- Carga e armazenamento
- O bit auxiliar de vai-um
- O bit de sinal
- O bit de overflow
- O flag de negativo
- O flag de zero
- O bit de vai-um

Formato das Instruções

Critérios de projeto

- Compatibilidade com arquiteturas anteriores
- Se a memória é rápida, uma arquitetura baseada em pilha é uma boa escolha
- Se a memória é lenta, pode-se optar por ter-se muitos registradores
- Instruções curtas ocupam menos espaço de memória, porém são mais difíceis de codificar
- Deve haver espaço para a codificação de todas as instruções existentes e ainda as possíveis futuras
- Quanto maior a resolução da memória, mais longas são as instruções (simplicidade x eficiência)



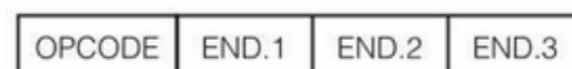
(a)



(b)



(c)



(d)

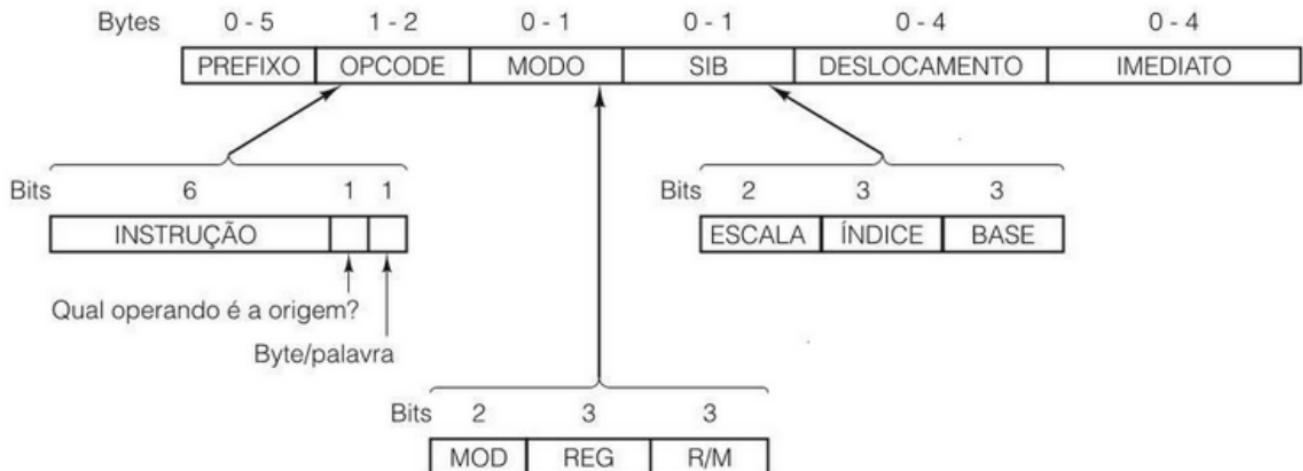
OPCODE Identifica o que a instrução deverá fazer

Endereços

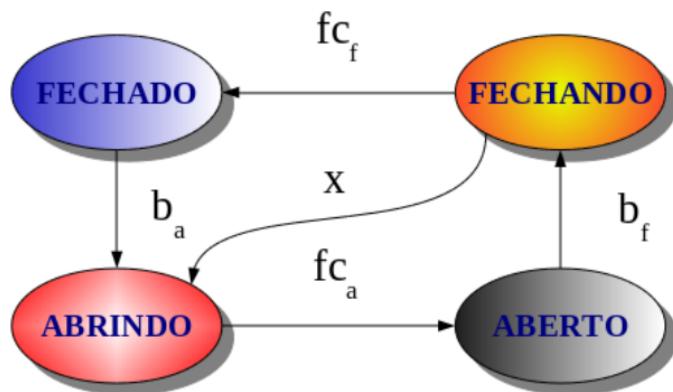
- Podem ser um, dois, três, ou não haver no formato da instrução
- Podem ser de tamanho fixo ou de tamanho variável.
- Simplicidade e economia de espaço são características a serem balanceadas

Core i7

- Seis campos de comprimento variável, sendo cinco deles opcionais
- Não existem instruções com dois endereços de memória
- A compatibilidade contribuiu para o aumento da complexidade do formato de instruções



O PROJETO DO CONTROLADOR



Endereçamento

Endereçamento imediato

- A informação pode ser encontrada no corpo da própria instrução, sem que seja necessário fornecer um endereço de onde a informação pode ser encontrada. Deve ser um valor constante.

```
MOV AX, 5 ;armazena o valor decimal 5 no acumulador.
```

- Exige menos processamento, pois não é necessário fazer um acesso extra à memória em busca da informação
- O tamanho da informação fica limitado ao tamanho do referido campo da instrução

Endereçamento direto

- O usuário fornece a localização real (*offset*) de uma variável. Isso é feito tipicamente especificando nome da variável. O nome de uma variável é igualado ao *offset* da variável no segmento de dados.

```
INC [GRADE]           ;Add 1 to the variable GRADE
INC [FIELD + 4]       ;Add 1 to the memory location that is 4 bytes after the variable
FIELD
INC [BYTE PTR 32]    ;Add 1 to the byte at offset 32 decimal
```

- A instrução sempre acessa a mesma posição de memória, ou seja, embora o valor possa mudar, seu endereço não muda
- O valor do endereço precisa ser conhecido em tempo de compilação. Por isso, seu uso é restrito

Endereçamento de registrador

- Especifica-se um registrador ao invés de uma posição de memória

```
MOV AX,BX ; Move o conteúdo do registrador BX para o registrador AX.
```

- É o tipo de endereçamento mais comum, por ser rápido e resultar em operações curtas
- É utilizado por compiladores na otimização de desempenho

Endereçamento indireto de registrador

- O usuário coloca o *offset* para uma variável num desses registros: SI, DI, BX, BP. A instrução então usa esse registro para acessar indiretamente os dados. Referência aos dados de acesso SI, DI, BX no segmento de dados. Referência aos dados de acesso do BP no segmento da pilha. Linguagens de alto nível usam esse modo para implementar ponteiros.

```
MOV BX,var ; BX = endereço da variável var na memória.
ADD AX,[BX] ; Adiciona o conteúdo da memória apontada por BX em AX.
```

- O operando especificado vem da memória ou vai para ela, mas seu endereço não está ligado à instrução. Em vez disso, está contido em um registrador
- Pode-se referenciar a memória com um tamanho menor de instrução, pois o endereço completo de memória não fica na instrução, mas sim no registrador

Endereçamento indexado

- É o nome que se dá ao endereçamento de memória que fornece um registrador (explícito ou implícito) mais um deslocamento constante

- Pode-se usar um ponteiro de memória em um registrador, mais um pequeno índice na instrução, ou o inverso, ou seja, um ponteiro de memória na instrução, mais um pequeno índice no registrador
- O código conhecido como “byte code” interpretado pela máquina virtual java usa a primeira estratégia

Endereçamento de base indexado

- O endereço de memória é calculado somando-se dois registradores mais um deslocamento (opcional). Neste caso, um dos registradores é a base e o outro é o índice
- Ao colocar-se o endereço de A em R5 e o endereço de B em R6, as duas primeiras instruções em laço do exemplo anterior poderiam ser escritas como:

```
MOV R4, (R2 + R5)
AND R4, (R2 + R6)
```

Endereçamento de pilha

- Pode fazer com que uma instrução não tenha nenhum endereço em seu corpo (desejável, por motivos de eficiência).
- O endereçamento de pilha pode ser implícito ao tipo de instrução utilizado
- Notação infixa e notação polonesa invertida
 - “X + Y * Z” em notação infixa pode ser escrito como Y Z * X + em notação polonesa invertida
- A pilha é útil para avaliação de expressões escritas em notação polonesa invertida
 - Parênteses não são necessários
 - Não é necessário definir precedência de operadores, o que pode ser considerado arbitrário

Ortogonalidade e modos de endereçamento

- Deveria existir um número mínimo de formatos de instruções, para que o compilador produzisse bom código.
- Todos os opcodes deveriam permitir todos os modos de endereçamento
- Todos os registradores deveriam estar disponíveis para todos os modos de endereçamento de registrador

	Bits	8	1	5	5	5	8
1		OPCODE	0	DESTINO	ORIGEM1	ORIGEM2	
2		OPCODE	1	DESTINO	ORIGEM1		DESLOCAMENTO
3		OPCODE				DESLOCAMENTO	

Tipos de Instruções

Instruções de movimentação de dados

- Implementa operações do tipo $A \leftarrow B$
- Deveria ser entendida como operação de cópia de dados, mas não de movimentação de dados

Operações diádicas

- Utilizam dois operandos (adição, subtração, multiplicação, divisão, e operações lógicas)
- A operação lógica AND é útil para remover bits
- A operação lógica OR é útil para inserir bits

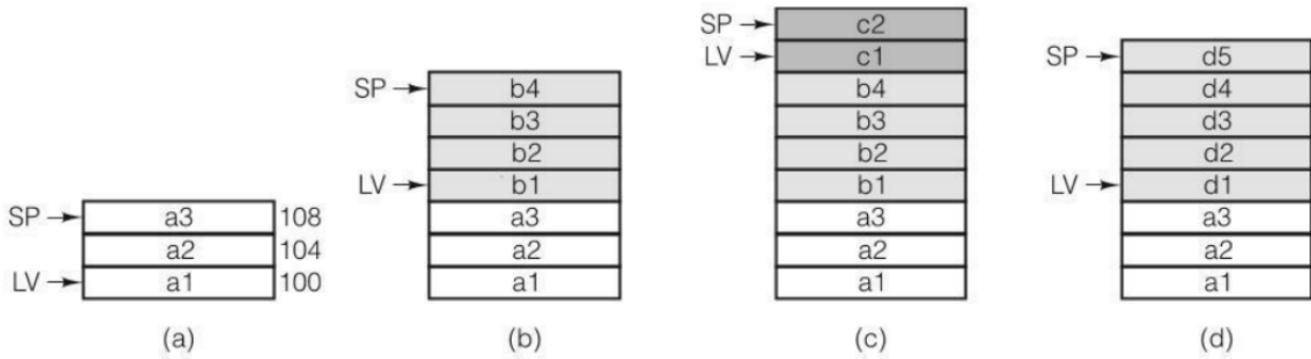
10110111 10111100 11011011 10001011	A
11111111 11111111 11111111 00000000	B (máscara)
<hr/>	
10110111 10111100 11011011 00000000	A AND B
<hr/>	
00000000 00000000 00000000 01010111	C
<hr/>	
10110111 10111100 11011011 01010111	(A AND B) OR C

Operações monádicas

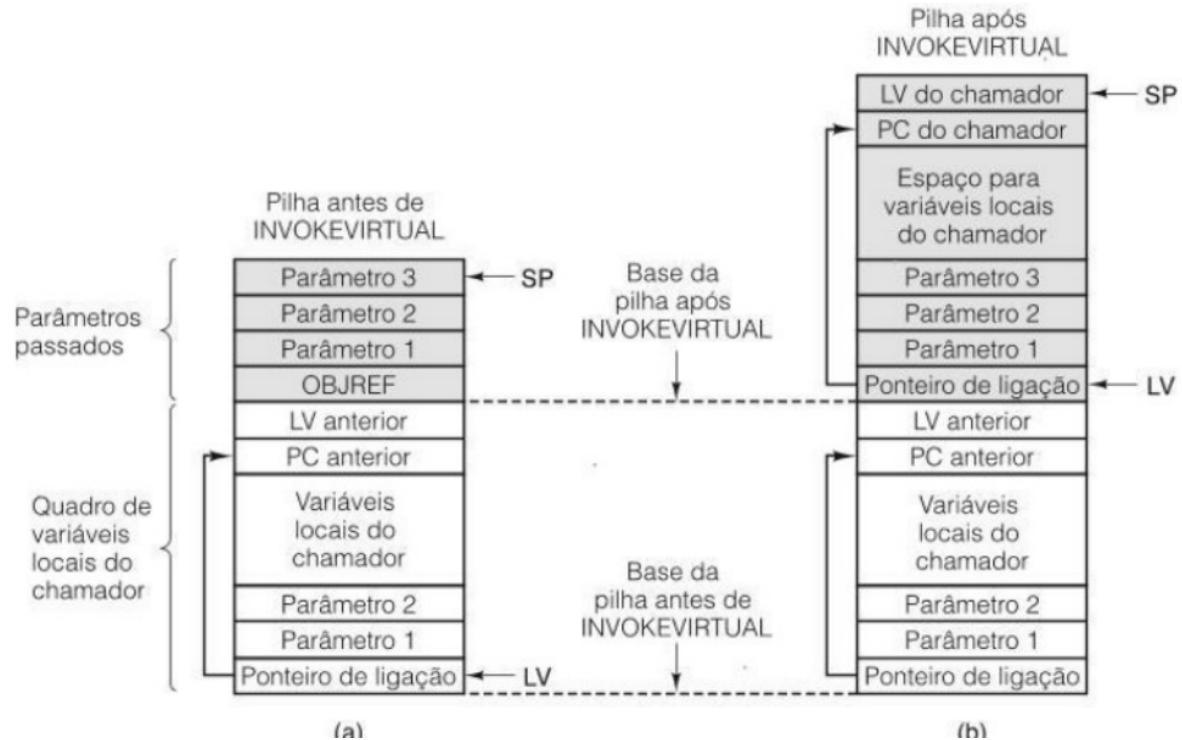
- Possuem um único operando e produzem um único resultado. Exemplo: rotação e deslocamento de bits
- Podem ser úteis para acelerar certos cálculos aritméticos. Por exemplo: $18n = 16n + 2n$
- Algumas operações diádicas podem ser transformadas em operações monádicas por questões de eficiência. Exemplos: CLR, INC, NEG

Comparações e desvios condicionais

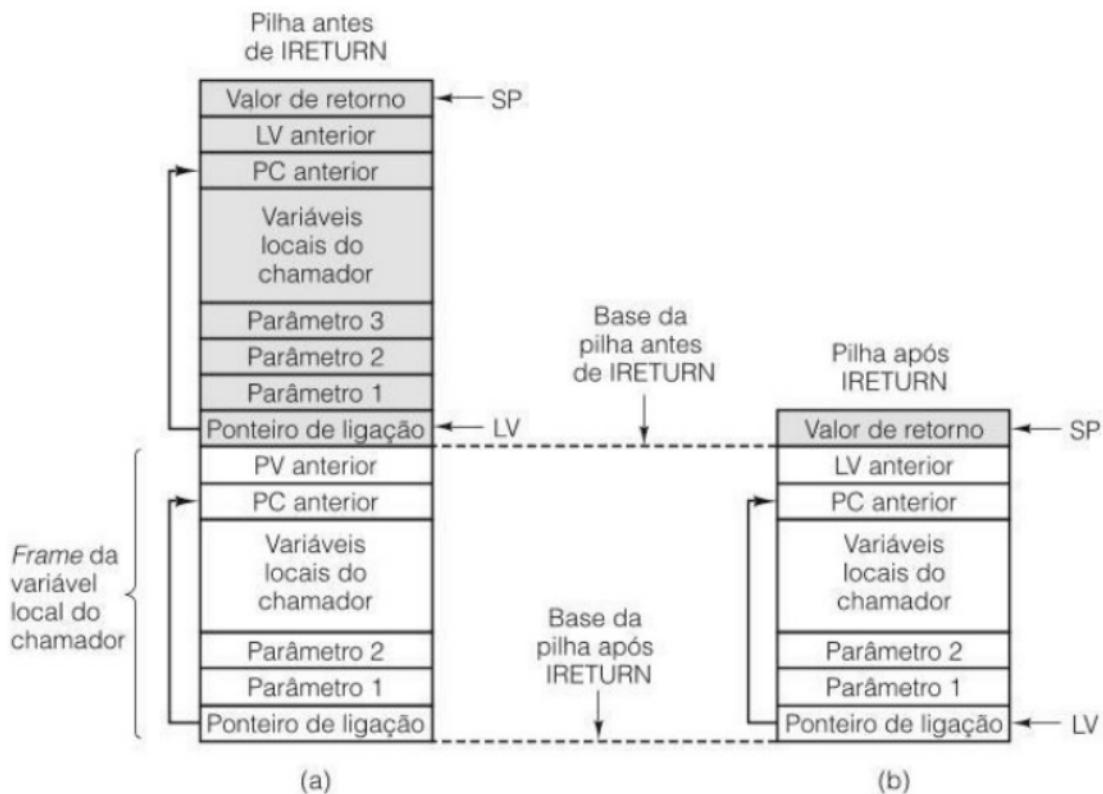
- Na maioria das arquiteturas, operações CMP são utilizadas para fazer comparações e ajustar um ou mais bits no registrador de status.
- Uma comparação do tipo JMP é utilizada para desviar o fluxo de controle testando um ou mais bits no registrador de status.
- Uma comparação CMP tipicamente utiliza dois operandos, enquanto que uma operação JMP tipicamente utiliza um operando RÓTULO, para onde se quer desviar.
- Instruções para chamada de procedimento
 - O endereço de retorno é salvo na pilha.
- Instruções para chamada de procedimentos
 - Tipicamente chamadas de CALL, onde um dos operandos é o procedimento chamado
 - Exemplo: a) Enquanto A está ativo, b) Após A chamar B, c) Após B chamar C, d) Após B e C retornarem, e A chamar D



- Chamando métodos na JVM



- Retornando de um método na JVM



Controle de laço

- Tipicamente implementadas com comparações, instruções goto, e incrementos ou decrementos de contadores
- Exemplo: a) teste no final do laço, e b) teste no início do laço

```
i = 1;
L1:    primeira declaração;
.
.
.
última declaração;
i = i + 1;
if (i < n) goto L1;
```

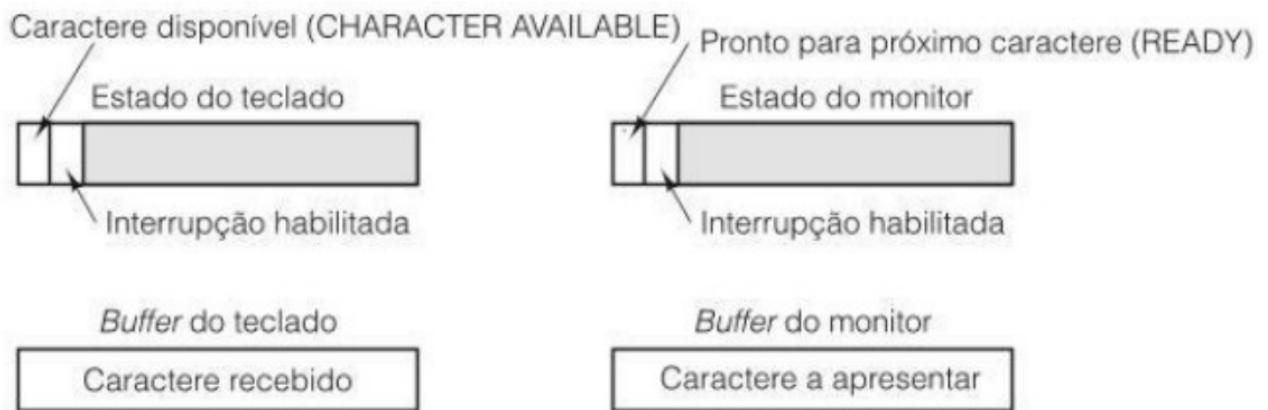
(a)

```
i = 1;
L1:    if (i > n) goto L2;
primeira declaração;
.
.
.
última declaração;
i = i + 1;
goto L1;
L2:
```

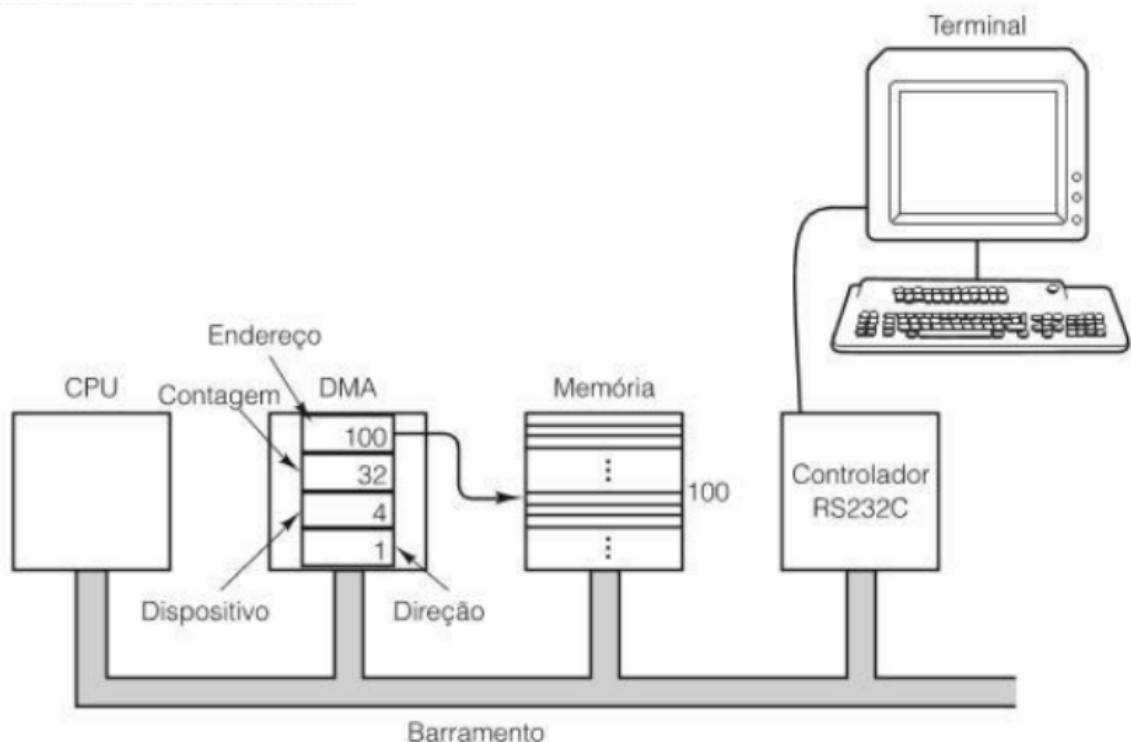
(b)

Entrada e saída

- Entrada e saída programadas: ocorre desperdício de tempo de processamento, e não é indicado quando vários processos estão em execução



- Acesso direto à memória: rouba ciclos do barramento, mas ainda é melhor que fazer uma interrupção por byte



Fluxo de Controle

```

public void torres(int n, int i, int j){
    int k;
    if (n == 1)
        System.out.println("Mova um disco de " + i + "para" + j)
    else{
        k = 6 - i - j;
        torres(n - 1, i, k)
        torres(1, i, j)
        torres(n - 1, k, j)
    }
}

```

- Exceções são implementadas na forma de chamadas a procedimentos com endereços fixos na memória quando algo de errado ocorre
- São exemplos de exceções:
 - Transbordo de inteiros (overflow)
 - Violação de proteção
 - Opcode indefinido
 - Transbordo de pilha
 - Tentativa de iniciar dispositivo de E/S inexistente
 - Tentativa de buscar palavra de memória de endereço ímpar
 - Divisão por zero.
- O tratamento de interrupções ocorre da seguinte forma para um dispositivo de E/S (ações do hardware)
 - O controlador de dispositivo ativa uma linha de interrupção no barramento
 - A CPU ativa um sinal de reconhecimento de interrupção no barramento assim que estiver pronta
 - Quando o controlador de dispositivo percebe o seu reconhecimento, coloca seu ID no barramento
 - A CPU salva o ID do controlador em algum lugar
 - A CPU salva o PSW e contador de programa na pilha
 - Em seguida, a CPU usa o ID para localizar o novo contador de programa de tratamento de interrupção em uma tabela.
- O tratamento de interrupções ocorre da seguinte forma para um dispositivo de E/S (ações do software)
 - A primeira coisa que a rotina de tratamento de interrupção faz é salvar todos os registradores em algum lugar, de maneira a recordá-los mais tarde
 - O número do terminal é identificado a partir de algum registrador de dispositivo
 - Outros registradores de dispositivos também podem ser lidos, e tratamento de erros podem ser feitos, caso necessário
 - As variáveis ptr e count são atualizadas a medida que os caracteres são retirados do buffer
 - Um código especial é informado ao dispositivo ou ao controlador de interrupção que a mesma acabou
 - Todos os registradores salvos são restaurados

Assembly

Sections

All x86_64 assembly files have three sections, the “.data” section, the “.bss” section, and the “.text” section.

The data section is where all data is defined before compilation.

The bss section is where data is allocated for future use.

The text section is where the actual code goes.

```
section .data
    text db "Hello, World!",10

section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, text
    mov rdx, 14
    syscall

    mov rax, 60
    mov rdi, 0
    syscall
```

Labels

A “label” is used to *label* a part of code.

Upon compilation, the compiler will calculate the location in which the label will sit in memory.

Any time the name of the label is used afterwards, that name is replaced by the location in memory by compiler.

```
section .data
    text db "Hello, World!",10

section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, text
    mov rdx, 14
    syscall

    mov rax, 60
    mov rdi, 0
    syscall
```

Global

The word “global” is used when you want the linker to be able to know the address of some a label.

The object file generated will contain a link to every label declared “global”.

In this case, we have to declare “_start” as global since it is required for the code to be properly linked.

```
section .data
    text db "Hello, World!",10

section .text
    global _start
_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, text
    mov rdx, 14
    syscall

    mov rax, 60
    mov rdi, 0
    syscall
```

Entradas para Chamadas de Sistema

Argumento	Registradores
ID	rax
1	rdi
2	rsi
3	rdx
4	r10
5	r8
6	r9

Jumpers

je <label>	; if <op1> == <op2>
jne <label>	; if <op1> != <op2>
jl <label>	; signed, if <op1> < <op2>
jle <label>	; signed, if <op1> <= <op2>
jg <label>	; signed, if <op1> > <op2>
jge <label>	; signed; if <op1> >= <op2>
jb <label>	; unsigned, if <op1> < <op2>
jbe <label>	; unsigned, if <op1> <= <op2>
ja <label>	; unsigned, if <op1> > <op2>
jae <label>	; unsigned, if <op1> >= <op2>

Operações Matemáticas

Nome da Operação	Nome da Operação (signed)	Descrição
add a, b	-	$a = a + b$
add a, b	-	$a = a - b$
mul reg	imul reg	$rax = rax * reg$
div reg	idiv reg	$rax = rax / reg$
neg reg	-	$reg = -reg$
inc reg	-	$reg = reg + 1$
dec reg	-	$reg = reg - 1$
adc a, b	-	$a = a + b + CF$
sbb a, b	-	$a = a - b - CF$

Operações de Stack

Operação	Efeito
push reg/valor	Push um valor no topo da stack.
pop reg	Pop um valor do topo da stack.
mov reg, [rsp]	Armazena o valor <i>peek</i> em reg.

Macros

Definição

Uma macro pode ser definida, basicamente, como um texto a ser substituído por outro texto.

Exemplo:

```
SWAP MACRO
    mov EAX, P
    mov EBX, Q
    mov Q, EAX
    mov P, EBX
ENDM

SWAP

SWAP
```

- Uma **chamada** macro é feita quando ela é utilizada na forma de um opcode;
- Uma **expansão** de macro é feita quando é substituída pelo seu corpo. (Durante a montagem)

- A expansão da macro ocorre durante a montagem, mas não durante a execução do programa.
- Macro é um **bloco** de texto que recebe um nome. O **nome** é substituído pelo **bloco** de instrução durante a montagem.

Toda definição de macro possui:

- Um cabeçalho;
- Um corpo;
- Um símbolo que delimita seu fim.

Macros x Chamada

Item	Macro	Call Procedure
Quando a chamada é feita?	Durante a Montagem	Durante a execução do programa
O corpo inserido no .o em todos os lugares que a chamada é feita?	Sim	Não
Uma call proced. é inserida no .o e executada mais tarde?	Não	Sim
Utiliza instruções de retorno ao final da chamada?	Não (Corpo é substituído)	Sim (ret)
Quantidade de cópias do corpo aparecem no .o	Uma por chamada de macro	Uma

Características

- Podem produzir rótulos repetidos;
- Macros podem chamar outras macros;
- Macros podem ser recursivas.

Parâmetros

- Formais (Definição macro)

```
change macro p1, p2
    mov eax, p1
    mov ebx, p2
endm
```

- Reais (Chamada macro)

```
change P, Q
```

Processador de Macros

- Possui uma tabela com o **nome** das macros e outra tabela com o **corpo** das macros.
- Quando uma macro é encontrada, o processador de macros passa a ler o corpo da macro substituindo os parâmetros formais pelos reais.

Resumo

Call Procedure	Macro
Chamada durante execução	Durante Montagem
Executada mais tarde	Substitui nome de macro com bloco de instruções
Instrução de retorno	

Exemplo Macro

Definição

```
%macro print 2 ;2 Parâmtros
    mov eax, sys_write
    mov edi, std_out
    mov esi, %1
    mov edx, %2
    syscall
%endmacro
```

Chamada

```
print sum, 1
```

Definição

```
change macro p1, p2
    mov eax, p1
    mov ebx, p2
    mov p2, eax
    mov p1, ebx
endm
```

Chamada

```
change R, S
```

O montador

Primeira passagem

- O contador de localização de instrução (ILC)

Rótulo	OPCODE	Operando	Comprimento	ILC
MARIA:	mov	eax, I	5	100
	mov	ebx, J	6	105

- Tabela de Símbolos
- Tabela de Pseudo Instruções
- Tabela de Literais (Opcional)
- A tabela de opcodes

OPCODE	Primeiro Operando	Segundo Operando	OpCode Hexa	Comprimento da Instrução	Classe da Instrução
AAA	-	-	37	1	6
ADD	EAX	immed32	05	5	4
ADD	reg	reg	01	2	19
AND	EAX	immed32	25	5	4
AND	reg	reg	21	2	19

Segunda Passagem

- Produz o Código Objeto (.O)
- Passa informações adicionais ao ligador
- Verificação de Erros

Tabela de Símbolos

- Em geral, usa-se uma memória associativa, que pode ser de busca linear, binária, ou usando uma tabela hash;
- A busca linear pode ser muito lenta;
- A busca binária pode exigir ordenação da tabela de símbolos;
- A tabela hash faz a ponderação entre tamanho da tabela e eficiência de busca, e é muito usada na prática.

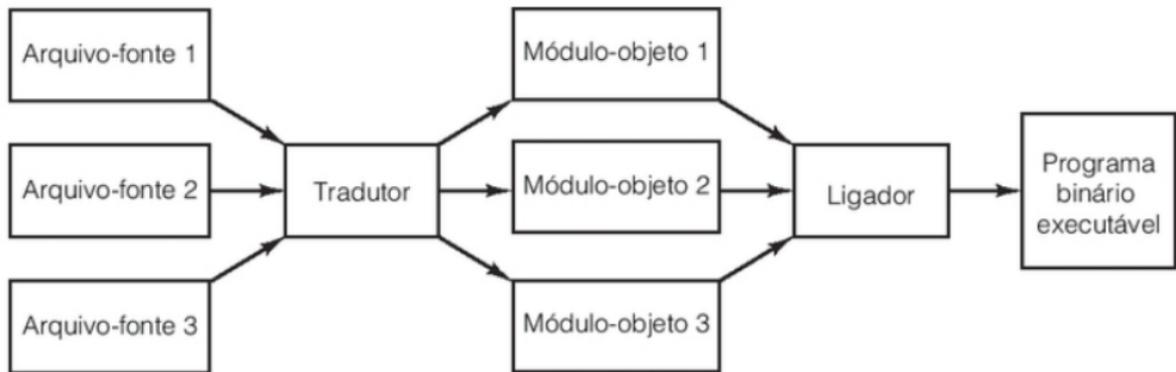
A TABELA HASH

- Símbolos com o mesmo número hash são armazenados na mesma posição da tabela, em forma de uma lista ordenada;
- O cálculo do número hash pode variar de metodologia para metodologia.

Ligaçāo e Carregamento

- Antes da Execução, o ligador encontra e interliga todos os procedimentos traduzidos.
- Tradução completa requer 2 etapas
 1. Compilação ou montagem de Procedimentos

2. Ligação de módulos objeto

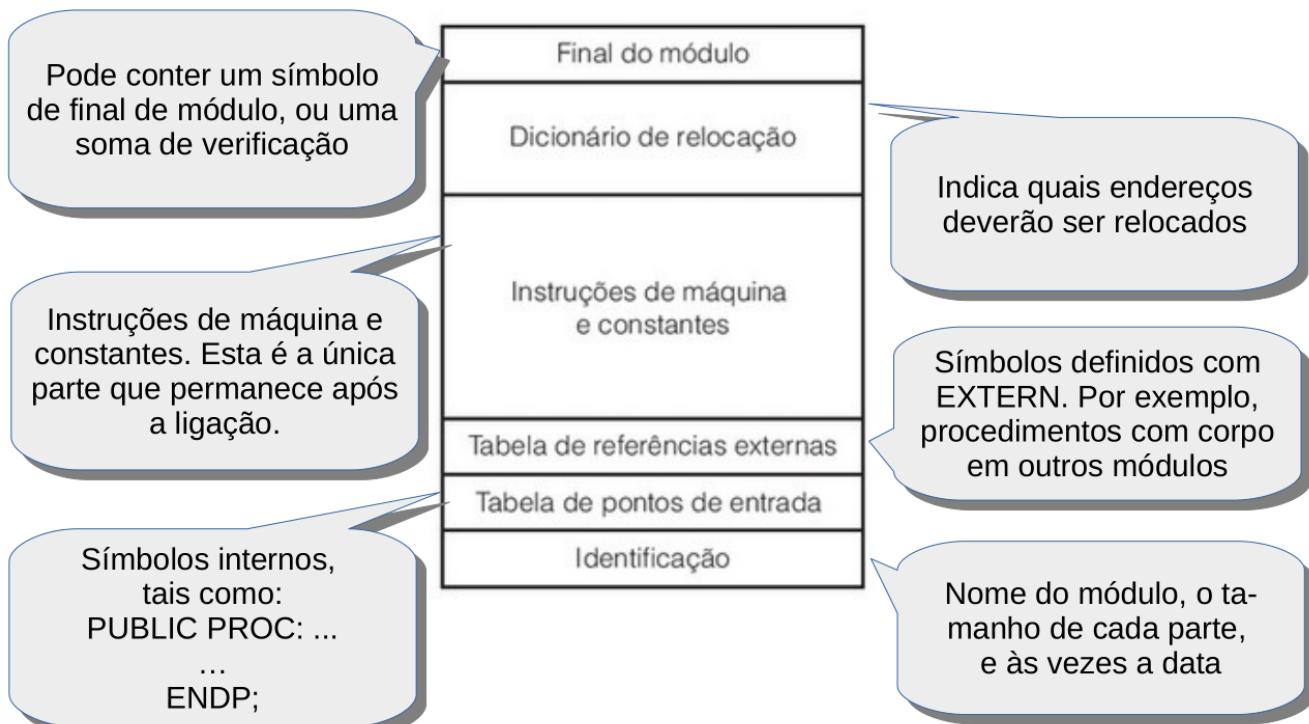


- Não ocorre mudança de nível durante a ligação. (Somente após compilação ou montagem)
- Código Fonte é montado em partes, caso contrário uma pequena alteração no código fonte provocaria um grande esforço de compilação

Resolvendo a relocação e a referência externa

- Problema ocorre pois a imagem do arquivo binário executável foi carregada, mas ainda não está pronta para ser executada
- Constrói-se uma tabela com todos os seus módulos objetos e seus comprimentos;
- Um endereço de início de cada módulo objeto é calculado com base na tabela;
- Uma constante de relocação, igual ao endereço de início de cada módulo, é adicionada a cada referência à memória;
- Os endereços de procedimento são adicionados às instruções chamadoras

Módulo Objeto



Ligaçāo em duas passagens

- Primeira Passagem: Constrói-se uma tabela de **nomes** e **tamanhos** dos **módulos objeto**; e uma tabela de **símbolos**, global, com todos os pontos de entrada e referências externas.
- Segunda Passagem: É feita a **relocação** e **ligação** de um módulo objeto por vez.

Tempo de vinculação e relocação dinâmica

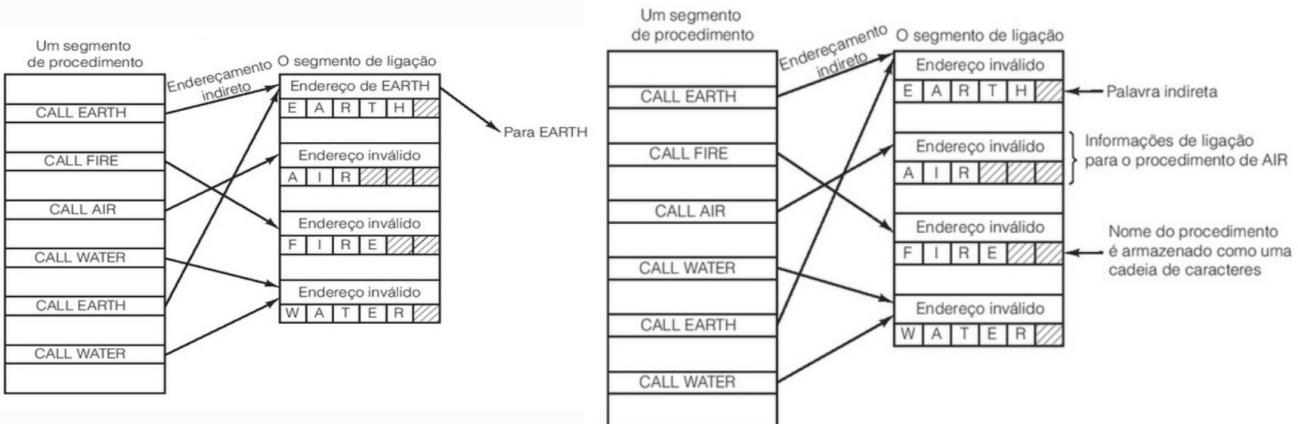
- Um programa pode mudar de posição na memória
- Caso aconteça, é necessário atribuir novos endereços à símbolos pre-existentes
- Custo desse processo pode ser muito alto
- Momento que tal atualização é feita é chamada de tempo de vinculação

Ligaçāo Dinâmica

- Consiste em ligar um determinado procedimento apenas quando ele é chamado pela primeira vez;
- Isso acontece por questões de eficiência, tendo em vista que muitos procedimentos são raramente executados;
- O primeiro sistema a utilizar a ligação dinâmica foi o MULTICS.

A LIGAÇÃO DINÂMICA NO MULTICS

Um segmento de ligação é utilizado; um bloco de informações para cada procedimento; armazena-se o endereço virtual de cada procedimento;



A LIGAÇÃO DINÂMICA NO WINDOWS

- Todas as versões do Windows usam ligação dinâmica, e DLLs (Dynamic link library);
- A DLL é um conjunto de procedimentos que podem ser carregados na memória, e acessados por vários processos ao mesmo tempo.
- DLLs são usadas para poupar-se espaço de memória principal e de disco;
- Usando-se DLLs, cada biblioteca aparece apenas um única vez em disco, e também na memória principal;
- Uma DLL não possui a função principal, que é chamada pelo sistema operacional;
- Uma DLL possui funções específicas que são chamadas na vinculação e desvinculação de processos;
- A vinculação pode ser implícita, via biblioteca de importação, ou explícita;
- Uma DLL não possui identidade própria, como acontece com processos e threads.

A LIGAÇÃO DINÂMICA NO UNIX

- Muito semelhante às DLLs do Windows; no Unix, são chamadas de bibliotecas compartilhadas;
- Apenas a ligação implícita é suportada;
- Uma biblioteca compartilhada é formada de duas partes: uma biblioteca hospedeira, e uma biblioteca alvo;
- Com pequenas diferenças de detalhes, mas essencialmente com os mesmos conceitos das DLLs do Windows.