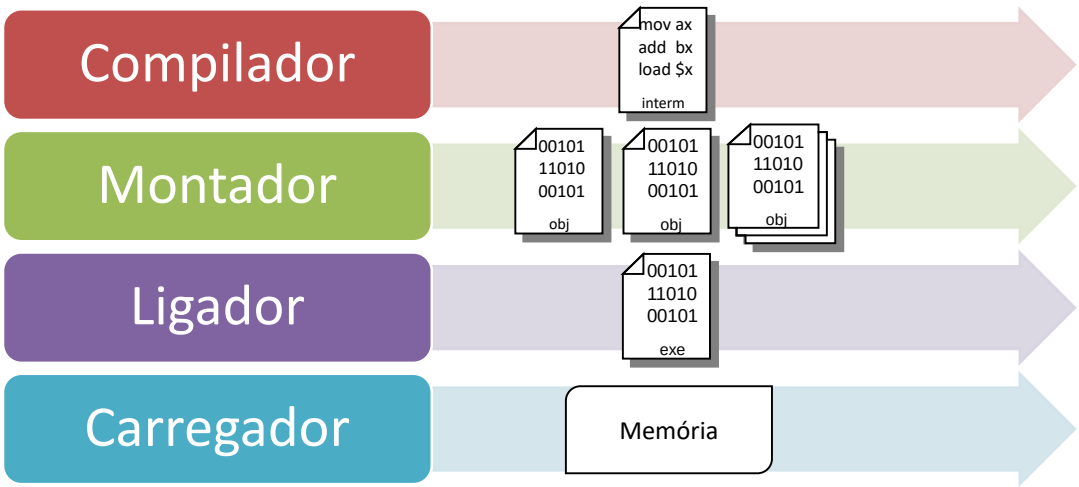
Compiladores e Computabilidade

Prof. Leandro C. Fernandes
UNIP – Universidade Paulista, 2018

Visão geral do processo



Visão geral do processo

- Fluxo da construção até a execução de um programa:
  - Compiladores
    - Responsável pela tradução de um programa descrito em linguagem de alto nível para
  - Montadores (*Assembler*)
    - Responsáveis pela tradução do programa escrito em linguagem assembly
    - Resultado é um programa em linguagem de máquina
  - Ligadores (*Linker* ou *Linkeditor*)
    - Responsável por unir diferentes partes de um programa
    - Permite modularizar o desenvolvimento de um programa
  - Carregadores (*Loader*)
    - Transfere o programa para a Memória Principal

MONTADORES

Montadores (*Assemblers*)

- **Tarefa:** Traduzir programas escritos em linguagem de montagem (*assembly*) para linguagem de máquina

- As funções da montagem compreendem:
  - Substituir os mnemônicos pelos opcodes do conjunto de instruções do processador.
  - Determinar de maneira absoluta ou relativa (em termos do valor do registrador *Program Counter*) o endereço de destino dos rótulos.
  - Reservar espaço para dados de acordo com o tipo associado a cada variável.
  - Gerar constantes em memória para variáveis e constantes, determinando o valor associado ao modo de endereçamento do operando.

Tradução e montagem

Programa em Linguagem de Alto Nível
int a,b,c;
read(a)
read(b)
c = a + b;
write(c);

Programa em Linguagem de Montagem (Assembly)		
Rótulo	*Mnemônico*	*Oper*
	INPUT	N1
	INPUT	N2
	LOAD	N1
	ADD	N2
	STORE	N3
	OUTPUT	N3
	STOP	
N1:	SPACE	
N2:	SPACE	
N3:	SPACE	

Programa em Linguagem de Máquina		
End.	*Opcod*	*Oper*
00	12	13
02	12	14
04	10	13
06	01	14
08	11	15
10	13	15
12	14	
13	??	
14	??	
15	??	

Conceitos Básicos

- Sintaxe típica de programas de montagem:
  [rótulo] [operação][operando1] [, operando2] ; comentário

- Rótulo:
  - É um elemento de marcação empregado para definir um endereço dentro da área de código.
- Operação:
  - **Opcodes simbólicos**: mnemônico que representa uma instrução
  - **Pseudo instrução**: Operação que orienta a atuação do montador
- Operandos
  - Registradores, constantes ou endereços de memória

Formato dos Comandos

- Formato MASM (*Microsoft Macro Assembly*)

Label	Opcode	Operandos	Comentários
FORMULA:	MOV	EAX,I	; registrador EAX = I
	ADD	EAX,J	; registrador EAX = I + J
	MOV	N,EAX	; N = I + j
I	DD	3	; reserva 4 bytes, inicializado com 3
J	DD	4	; reserva 4 bytes, inicializado com 4
N	DD	0	; reserva 4 bytes, inicializado com 0

Cálculo de N = I + J em assembly do Pentium IV

Formato dos Comandos

- Cálculo de N = I + J

Label	Opcode	Operandos	Comentários
FORMULA	MOV.L	I, D0	; registrador D0 = I
	ADD.L	J, DO	; registrador D0 = I + J
	MOV.L	D0, N	; N = I + j
I	DC.L	3	; reserva 4 bytes, inicializado com 3
J	DC.L	4	; reserva 4 bytes, inicializado com 4
N	DC.L	0	; reserva 4 bytes, inicializado com 0

Cálculo de N = I + J em assembly do Motorola 680X0

Formato dos Comandos

Label	Opcode	Operandos	Comentários
FORMULA:	SETHI	%HI(I), %R1	! R1 = bits de alta ordem do endereço I
	LD	[%R1+%LO(I)], %R1	! R1 = I
	SETHI	%HI(J), %R2	! R2 = bits de alta ordem do endereço J
	LD	[%R2+%LO(J)], %R2	! R2 = J
	NOP		! Espera J estar disponível na memória
	ADD	%R1, %R2, %R2	! R2 = R1 + R2
	SETHI	%HI(N), %R1	! R1 = bits de alta ordem do endereço N
	ST	%R2, [%R1+%LO(N)]	
I:	.WORD	3	; reserva 4 bytes, inicializado com 3
J:	.WORD	4	; reserva 4 bytes, inicializado com 4
N:	.WORD	0	; reserva 4 bytes, inicializado com 0

Cálculo de N = I + J em assembly do SPARC

Funções Básicas do Montador

- Substituir os mnemônicos por instruções com opcodes numéricos
  - Segue uma tabela de associações
    - Relaciona o mnemônico com a instrução alvo
- Substituir os endereços simbólicos por endereços numéricos
  - Saltos e desvios como endereços absolutos ou como deslocamento relativo ao *program counter*
  - Constantes (com os valores associados ao ponto do código)
- Reservar espaço para dados
  - A quantidade de memória é determinada de acordo com o tipo associado a cada variável
- Gerar constantes em memória
  - Variáveis e constantes (valor associado ao modo de endereçamento do operando)

Montadores de Duas Passagens

- Realiza duas passagens no arquivo fonte.
- Primeira passagem:
  - Reconhece símbolos definidos pelo programador
  - Constrói tabela com símbolos e seus respectivos valores
- Segunda passagem:
  - Geração do código objeto
- Estrutura de dados empregada
  - Tabela de instruções
  - Tabela de diretivas (pseudo-instruções)
  - Tabela de símbolos
  - Contador de posições

Tabela de Instruções

- Tabela de códigos de operação (parcial) de um montador para o Pentium 4

Opcode	1º operando	2º operando	Opcode em Hexa	Compr.	Classe
AAA	- x -	- x -	37	1	6
ADD	EAX	imediato 32bits	05	5	4
ADD	registrador	registrador	01	2	19
AND	EAX	imediato 32bits	25	5	4
AND	registrador	registrador	21	2	19

A classe da instrução define a tradução da instrução, cada classe sendo associada a procedimentos específicos de tradução do montador.

Primeira passagem

	COPY	ZERO, OLDER	**00**	09	??	??
	COPY	ONE, OLD	**03**	09	??	??
	INPUT	LIMIT	**06**	12	??	
	OUTPUT	OLD	**08**	13	??	
FRONT:	LOAD	OLDER	**10**	10	??	
	ADD	OLD	**12**	01	??	
	STORE	NEW	**14**	11	??	
	SUB	LIMIT	**16**	02	??	
	JMPP	FINAL	**18**	07	??	
	OUTPUT	NEW	**20**	13	??	
	COPY	OLD, OLDER	**22**	09	??	??
	COPY	NEW, OLD	**25**	09	??	??
	JMP	FRONT	**28**	05	??	
FINAL:	OUTPUT	LIMIT	**30**	13	??	
	STOP		**32**	14		
ZERO:	CONST	0	**33**	00		
ONE:	CONST	1	**34**	01		
OLDER:	SPACE		**35**	??		
OLD:	SPACE		**36**	??		
NEW:	SPACE		**37**	??		
LIMIT:	SPACE		**38**	??		

Símbolo	Valor
FRONT	10
FINAL	30
ZERO	33
ONE	34
OLDER	35
OLD	36
NEW	37
LIMIT	38

Segunda passagem

	COPY	ZERO, OLDER	**00**	09	33	35
	COPY	ONE, OLD	**03**	09	34	36
	INPUT	LIMIT	**06**	12	38	
	OUTPUT	OLD	**08**	13	36	
FRONT:	LOAD	OLDER	**10**	10	35	
	ADD	OLD	**12**	01	36	
	STORE	NEW	**14**	11	37	
	SUB	LIMIT	**16**	02	38	
	JMPP	FINAL	**18**	07	30	
	OUTPUT	NEW	**20**	13	37	
	COPY	OLD, OLDER	**22**	09	36	35
	COPY	NEW, OLD	**25**	09	37	36
	JMP	FRONT	**28**	05	10	
FINAL:	OUTPUT	LIMIT	**30**	13	38	
	STOP		**32**	14		
ZERO:	CONST	0	**33**	00		
ONE:	CONST	1	**34**	01		
OLDER:	SPACE		**35**	??		
OLD:	SPACE		**36**	??		
NEW:	SPACE		**37**	??		
LIMIT:	SPACE		**38**	??		

Símbolo	Valor
FRONT	10
FINAL	30
ZERO	33
ONE	34
OLDER	35
OLD	36
NEW	37
LIMIT	38

LIGAÇÃO (LINKER)

O papel do *Linker*

- Programas podem ser constituídos de vários procedimentos que podem estar separados em diferentes arquivos.
- Antes da execução, todos os procedimentos já traduzidos e montados devem ser agrupados para criar um único programa, para que então possa ser carregado na memória principal e colocado em execução.
- O software que faz a ligação é denominado Ligador (Linker).

O papel do *Linker* (cont.)

- O *linker* produz um programa único chamado Módulo Absoluto de Carga, a partir dos vários módulos objeto que foram obtidos pela tradução dos vários procedimentos (módulos fonte).
  - O Módulo de Carga contém todos os módulos ligados em um único programa executável.
- Um software denominado Carregador (*Loader*) é responsável por carregar o Módulo de Carga na memória principal para executá-lo.

O papel do *Linker* (cont.)

- Para criar o Módulo de Carga, o *Linker* une todos os Módulos Objeto em um único espaço de endereçamento:
  - Todas as referências a endereços devem ser atualizadas (Problema de Relocação);
    - Este problema é inexistente quando a memória é segmentada;
  - Quando existe um Procedimento A que chama a um Procedimento B, o endereço absoluto de B só é conhecido após a ligação (Problema de Referência Externa).

Estrutura de um módulo objeto

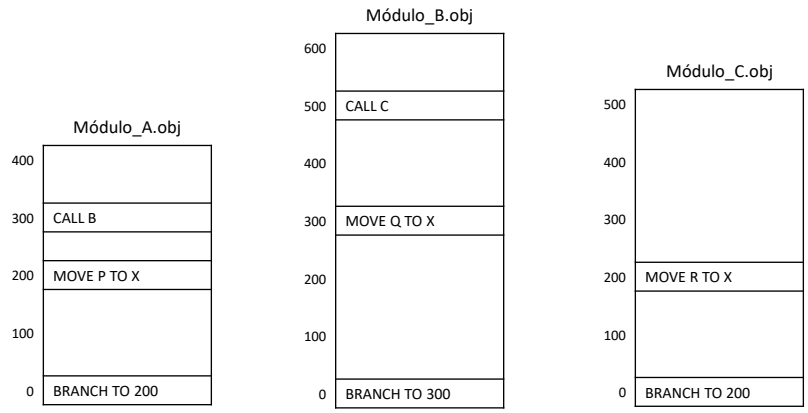
Cabeçalho	• Dado pela identificação de tipo, tamanho do código e eventualmente o arquivo de origem;
Código gerado	• Contém as instruções e os dados em formato binário;
Relocação	• Contém as posições no código onde ocorrerá mudanças quando for definida a posição de carregamento.
Tabela de Símbolos	• Lista de símbolos globais definidos no módulo e símbolos externos, que devem vir de outros módulos;
Depuração	• Contém referências para o código fonte (ex: número de linha e nomes de identificadores).

Tarefas do *Linker*

1. Construir uma tabela com todos os módulos objeto e seu respectivos comprimentos.
2. Atribuir um Endereço de Carga a cada módulo objeto.
3. Relocar todas as instruções que contêm um endereço adicionando uma Constante de Relocação (que corresponde ao endereço inicial de cada módulo).
4. Encontrar todas as instruções que referenciam outros procedimentos e inserir nelas o endereço absoluto dos mesmos.

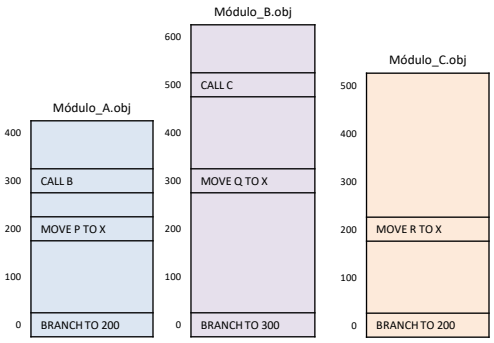
A junção dos módulos

- Cada módulo é montado considerando um espaço de endereçamento próprio e assim começando no endereço 0 (zero).



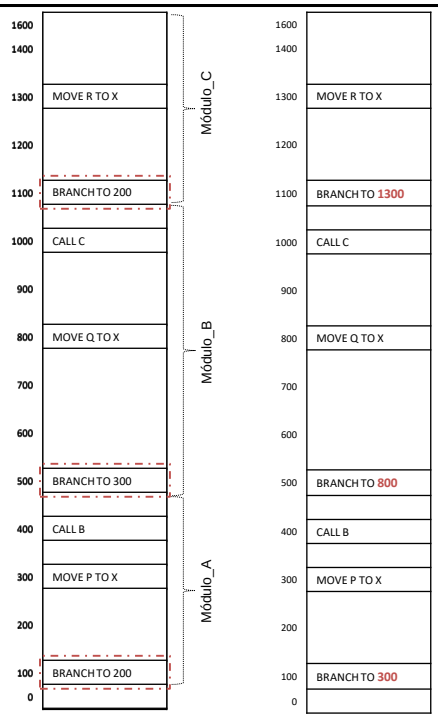
1. Construção da tabela dos módulos
2. Atribuição dos endereços de carga

- Supondo que o endereço inicial de montagem seja 100, os endereços dos módulos são calculados para que sejam posicionados na imagem binária.



Módulo	Tamanho	Endereço inicial
A	400	100
B	600	500
C	500	1100

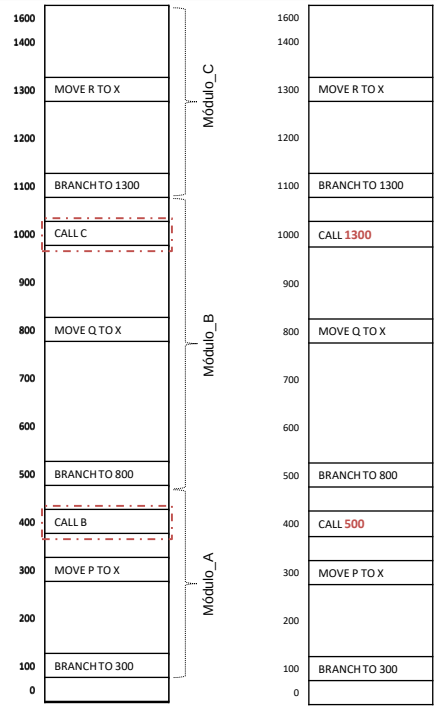
3. Realocar endereços



- As instruções devem ter suas referências atualizadas de acordo com os endereços que os módulos ocupam dentro da imagem binário.
- Esse processo é feito somando-se uma constante (que corresponde ao endereço base do módulo) as endereços.

Módulo	Tamanho	Endereço inicial
A	400	100
B	600	500
C	500	1100

4. Realocar endereços



- As chamadas a procedimentos também devem ter suas referências atualizadas.
- Os endereços são estabelecidos com base no endereço absoluto dentro do módulo mais o endereço base ocupado pelo módulo na imagem binária.

Módulo	Tamanho	Endereço inicial
A	400	100
B	600	500
C	500	1100

Algoritmo de Ligação

A maioria dos *Linkers* requer dois passos:

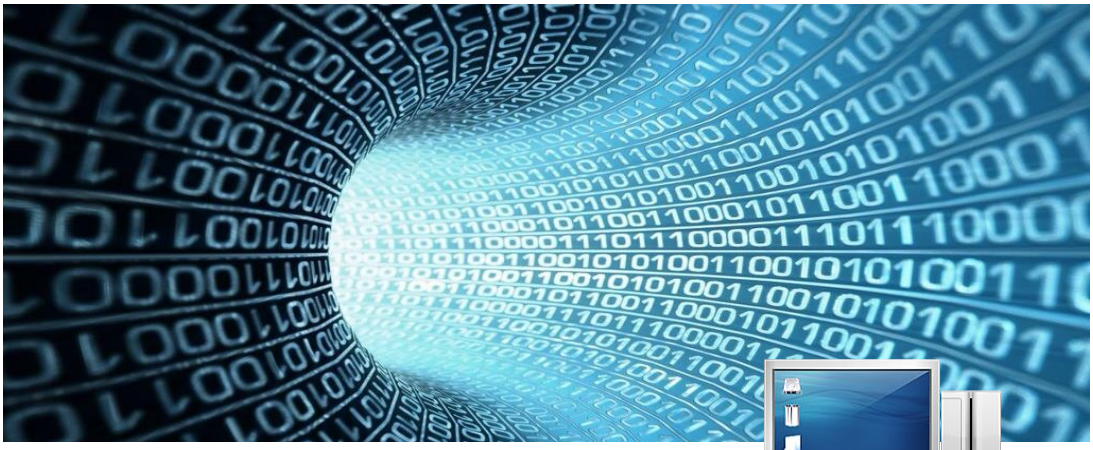
- Primeira passagem:
  - Ler todos os módulos objeto, construir uma Tabela de nomes e comprimentos de módulos e uma Tabela global de símbolos internos e externos.
- Segunda passagem:
  - Ler os módulos objeto, relocá-los e ligá-los para formar um único módulo.

Tempo de Mapeamento (*Binding Time)*

- **Tempo de Mapeamento:** tempo no qual um endereço real da memória principal é atribuído a um símbolo.
  - Quando o programa é escrito, quando o programa é carregado, quando a instrução que usa o endereço é executada, etc.
- Em Sistemas de Tempo Compartilhado, programas podem ser carregados em endereços diferentes, quando executados em instantes diferentes.
  - Devem ser relocados dinamicamente.
- Solução:
  - Mapear símbolos em endereços virtuais e alterar a tabela de páginas para realizar o novo mapeamento virtual-físico.

Ligação Dinâmica

- Trata-se do processo de ligação de procedimentos pré-compilados separadamente e que são ligados no momento em que estes são chamados pela primeira vez (durante a execução do programa).
- Vantagens:
  - Procedimentos que raramente utilizados são ligados apenas se forem necessário.
  - Permitem um melhor aproveitamento da memória virtual.



CARREGADORES (LOADER)

Carregador (*Loader*)

- **Função:** Responsável por copiar um programa para a Memória Principal e preparar a sua execução.

- Tarefas principais
  - Verificar se o programa existe
  - Dimensionar a quantidade de memória necessária
  - Solicitar ao SO a quantidade de memória necessária
  - Copiar o código para a memória
  - Ajustar os endereços do código executável

Carregador Absoluto

- Considera que programa é carregado sempre no mesmo endereço.
  - Referências específicas a um endereço
  - Referências definidas em tempo de projeto
    - Pelo programador ou pelo compilador/montador
  - Referências criadas pelo programador
    - Programador deve conhecer a plataforma
    - Inclusão/alteração de instruções implica em redefinição de endereços
  - Referências criadas pelo compilador/montador
    - Emprega símbolos para representar as referências

Carregador Relocador

- Programa carregado de forma absoluta impõe restrições.
  - O que fazer caso haja outro programa ocupando a área de memória desejada?
- Solução:
  - Trabalhar com endereços relativos a ponto conhecido
    - Exemplo: o endereço de início do programa
- Carga do programa na posição k da memória
  - Implica em adicionar a constante k a cada uma das referências do programa

Carregador Dinâmico

- Problema do emprego de carregador relocador:
  - Não adequado para situações de swapping
    - Translado de processos entre Memória Principal e disco pode maximizar o uso do processador
    - Processos não necessariamente retornam a mesma posição de memória anterior
- Solução:
  - Executar relocação no momento em que a posição for referenciada
  - Endereços devem ser relativos ao início do módulo na memória
    - Usa registrador específico (registrador de base)