Software Básico

nível de linguagem de montagem
(*assembly*)

Tanembaum, capítulo 7

UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA
COMPUTAÇÃO

Linguagem de montagem (Assembly)

- Abstração simbólica da linguagem de máquina
- Traduzida pelo programa *assembler*
- Mapeada diretamente em instruções de máquina
- Pseudo-instruções e macros auxiliares
  - Mnemônicos facilitam codificação
- Símbolos e rótulos simplificam endereçamento

Características

- Linguagem assembly pura
  - Cada comando corresponde a uma instrução de máquina
  - Uso de nomes e endereços simbólicos
  - Instruções mnemônicas: ADD, SUB ...
- Acesso a todas as características do HW alvo
  - Tudo que pode ser feito na máquina, pode ser feito em *assembly (diferentemente da linguagem de alto nível)*
  - Ex: testar bit de vai-um
- Restrita à família de processadores alvo
  - Programa em linguagem de alto nível pode ser recompilado para outra máquina

Por que Assembly?

- Programar em *assembly* é difícil
  - Demora mais pra escrever um programa
  - Depuração e manutenção complicadas
- Duas razões:
  - Desempenho
  - Acesso ao HW

Desempenho

- Em termos de tamanho: consumo de memória
  - crítico para sistemas embutidos
  - código em smart cards, telefones celulares

- Em termos de velocidade
  - Lei 90 – 10 (10% código : 90% do tempo de execução)
  - Implementar partes críticas em *assembly*
    - drivers de dispositivos,
    - rotinas de BIOS
    - laços internos de aplicações com restrições de desempenho críticas

Comparação entre programação em linguagens assembly e de alto nível

	Programadores-anos para produzir programa	Tempo de execução (seg)
Linguagem de alto nível	10	100
Linguagem de montagem	50	33 (= 100/3)
Abordagem mista (antes do ajuste)	10	100
10% críticos	1	90
Outros 90%	9	10
Abordagem mista (após ajuste)	15	40
10% críticos	1+5	30 (= 90/3)
Outros 90%	9	10

Outras experiências práticas

- MULTICS: 1o SO de tempo compartilhado
  - Função reescrita em 3 meses:
    - 26 vezes menor
    - 50 vezes mais rápida
  - Função reescrita em 2 meses:
    - 20 vezes menor
    - 40 vezes mais rápida

Acesso

- Algumas tarefas exigem acesso direto ao HW
  - Tratamento de interrupções do hardware
  - Controladores de dispositivos (sistemas de tempo real)
- Sistemas modernos (Linux)
  - Acesso às funções do processador para troca de contexto
  - Boot

Motivações reavaliadas

- Desempenho
  - É importante poder escrever um bom código de linguagem de montagem para cenários críticos
- Acesso – motivo mais importante
  - Mesmo assim, apenas em casos extremos
  - Uma linguagem como C dá bom acesso
- Tamanho: às vezes código de montagem é única saída devido à escassez de memória

Motivações reavaliadas

- Compilador deve ou produzir saída para assembler ou ele mesmo tem que executar processo de montagem
  - Alguém tem que escrever os compiladores

- Estudo de arquiteturas
  - *Assembly* está diretamente relacionada ao hardware
  - Permite entender como a máquina realmente trabalha do ponto de vista do hardware

Formato das Instruções

- Está diretamente associado ao *hardware*
  - Porém, possuem características comuns
- Instruções *Assembly*
  - Representam os comandos da máquina
  - Usualmente, quatro campos
    - Label, operação, operandos, comentários
- Pseudo-instruções: comandos para o assembler
  - Usadas para reservar espaço para dados
  - Apenas tipos básicos

Exemplos

Pentium 4

Label	Opcode	Operands	Comments
FORMULA:	MOV	EAX,I	; register EAX = I
	ADD	EAX,J	; register EAX = I + J
	MOV	N,EAX	; N = I + J
I	DD	3	; reserve 4 bytes initialized to 3
J	DD	4	; reserve 4 bytes initialized to 4
N	DD	0	; reserve 4 bytes initialized to 0

(a)

Motorola 680x0

Label	Opcode	Operands	Comments
FORMULA	MOVE.L	I, D0	; register D0 = I
	ADD.L	J, D0	; register D0 = I + J
	MOVE.L	D0, N	; N = I + J
I	DC.L	3	; reserve 4 bytes initialized to 3
J	DC.L	4	; reserve 4 bytes initialized to 4
N	DC.L	0	; reserve 4 bytes initialized to 0

(b)

Sparc

Label	Opcode	Operands	Comments
FORMULA:	SETHI	%HI(I),%R1	! R1 = high-order bits of the address of I
	LD	[%R1+%LO(I)],%R1	! R1 = I
	SETHI	%HI(J),%R2	! R2 = high-order bits of the address of J
	LD	[%R2+%LO(J)],%R2	! R2 = J
	NOP		! wait for J to arrive from memory
	ADD	%R1,%R2,%R2	! R2 = R1 + R2
	SETHI	%HI(N),%R1	! R1 = high-order bits of the address of N
	ST	%R2,[%R1+%LO(N)]	
I:	.WORD 3		! reserve 4 bytes initialized to 3
J:	.WORD 4		! reserve 4 bytes initialized to 4
N:	.WORD 0		! reserve 4 bytes initialized to 0

(c)

UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Campo de label

- Usado para atribuir um nome simbólico para uma variável ou endereço (para desvio)
  - Campo opcional

- Formato
  - Posição fixa ou separador
  - Tamanho fixo (6-8 caracteres) ou variável

Campo de operação

- Representações simbólicas das instruções do hardware
  - Critério de quem fez o montador
    - Intel: MOV
    - Motorola: MOVE
    - Sun: ST e LD
  - Variantes em função do tamanho do dado
    - Motorola: MOVE.L / MOVE.W / MOVE.B
    - Intel: EAX / AX / AH ou AL

Campo de operandos

- Especifica os alvos das operações
  - Endereços: instruções de desvio
  - Variáveis
    - posição de memória sendo acessada
    - Assemblers variam em como reservam espaço para dados (DD na Intel vs. .WORD na Sparc)
  - Constantes
    - valores para operações aritméticas (endereçamento imediato)
  - Registradores
    - variáveis locais do processador

Campo de comentário

- Usado para acrescentar informações relevantes ao programa
  - Facilitar a compreensão do mesmo
  - Programa assembly é praticamente incompreensível sem comentários

- Formato geral
  - Indicador de início (; ou !)
  - Até o final da linha

UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Pseudo-instruções

- Comandos para o montador, não instruções
  - Diretivas de assember
- Relacionadas com o modo de operação da arquitetura e do montador
  - Alocação de variáveis
    - ex: DD no Pentium, .WORD no SPARC
  - Criação de macros e subrotinas
  - Definição de escopo de uma macro / subrotina / variável
    - PUBLIC / EXTERN (local por default)
  - Definição de segmentos
  - Constantes
- Veja exemplos do assember MASM do Pentium 4 (Tabela 7.2)

Pseudo-instruções:

- Exemplos do MASM
  - Definir um novo símbolo igual a uma expressão dada

```
BASE  EQU   2000
LIMIT   EQU  4 * BASE + 2000
```

  - Alocar e definir armazenamento para um ou mais bytes ou palavras (16 bits) ou double (32 bits)

```
TABLE DB 11, 23, 49
```

Aloca espaço para 3 bytes e os inicializa com 11, 23 e 49, respectivalmente
Define símbolo TABLE e o ajusta com o endereço onde 11 é armazenado

Definição de Macro

- Bloco de código de uso repetitivo
- Ao invés de realmente reescrevê-lo, define-se a macro
  - Identifica as instruções a serem repetidas
  - Código é expandido em cada ocorrência
- Mais eficiente que chamada de procedimento quando conjunto de instruções é pequeno mas frequente

Definição, Chamada e expansão

- São necessários:
  - Cabeçalho com o nome da macro;
  - Bloco de instruções;
  - Pseudo-instrução indicando o término da macro.
- O montador salva a macro em uma tabela e quando esta é chamada, *substitui* a chamada pelo código.
  - Chamada de macro: utilização de um nome de macro como opcode
  - Expansão de macro: substituição pelo corpo da macro

Exemplo (Pentium 4)

```
MOV     EAX,P
MOV     EBX,Q
MOV     Q,EAX
MOV     P,EBX

MOV     EAX,P
MOV     EBX,Q
MOV     Q,EAX
MOV     P,EBX
```

(a)

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

        SWAP

        SWAP
```

(b)

Chamada e expansão

- A expansão ocorre durante o processo de montagem e não durante a execução do programa.
- O código gerado pelo exemplo (a) é o mesmo do gerado pelo exemplo (b).
- Assembler efetuado em 2 passos (conceitualmente):
  - As definições de macro são salvas e as chamadas são expandidas gerando um novo código em linguagem assembly
  - Código é processado pelo assembler como se fosse o original.

UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Macros com parâmetros

```
MOV    EAX,P
MOV    EBX,Q
MOV    Q,EAX
MOV    P,EBX

MOV    EAX,R
MOV    EBX,S
MOV    S,EAX
MOV    R,EBX
```

(a)

```
CHANGE    MACRO P1, P2
          MOV EAX,P1
          MOV EBX,P2
          MOV P2,EAX
          MOV P1,EBX
          ENDM

          CHANGE P, Q

          CHANGE R, S
```

(b)

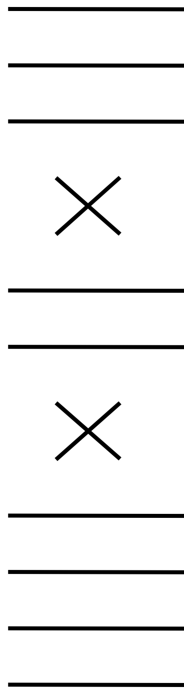
UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Subrotina

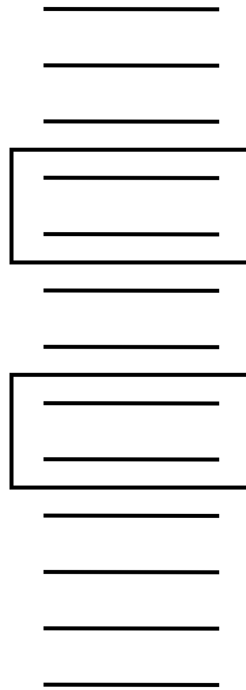
- É uma chamada a procedimento.
- Onde houver uma chamada, o procedimento é invocado separadamente e ao seu término, retorna para o programa que originou esta chamada.
- Ocorre um *desvio* no código.

Macro x Subrotina

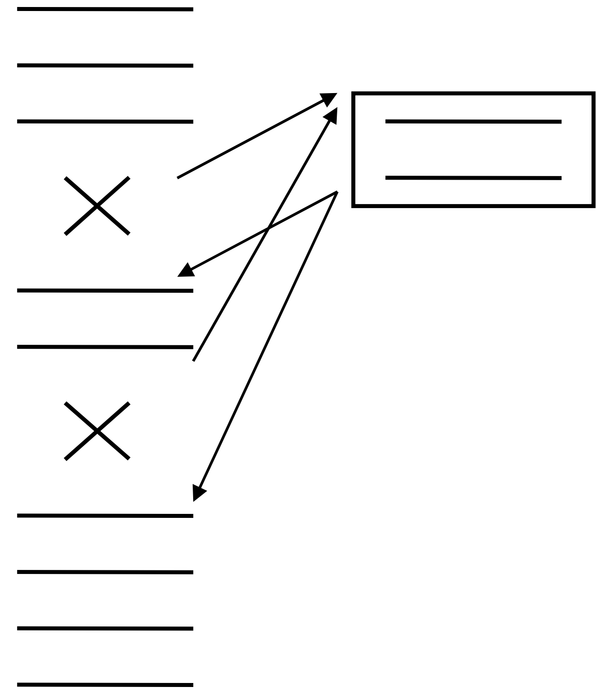
Programa



Macro



Subrotina

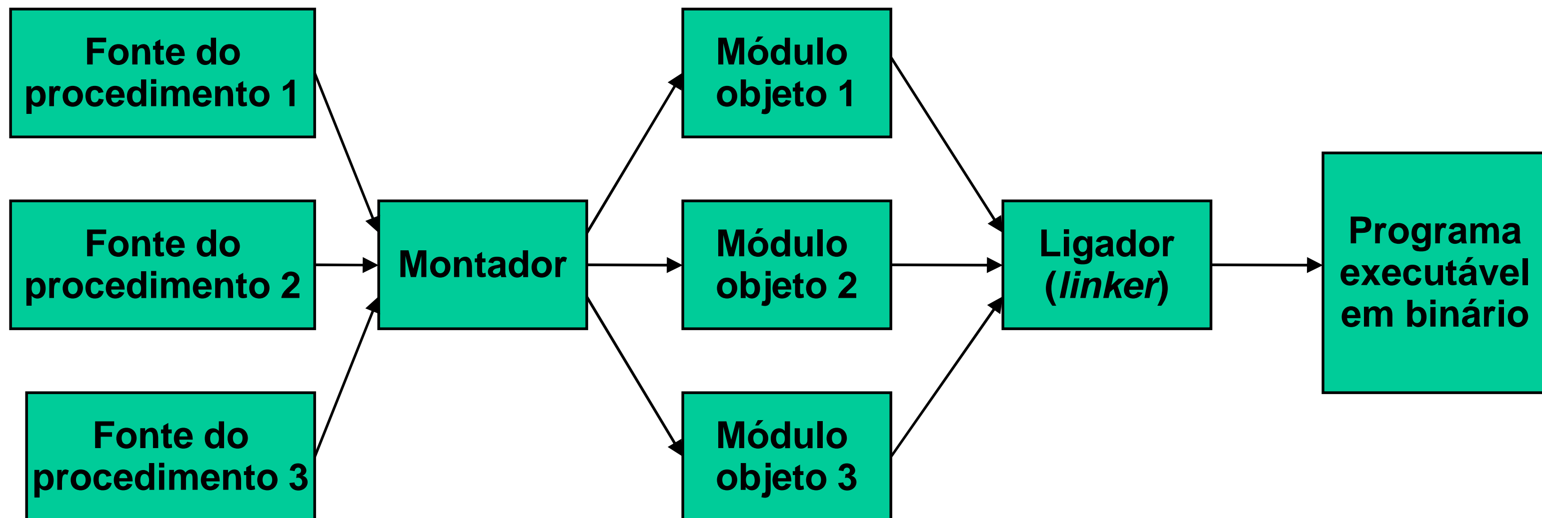


UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

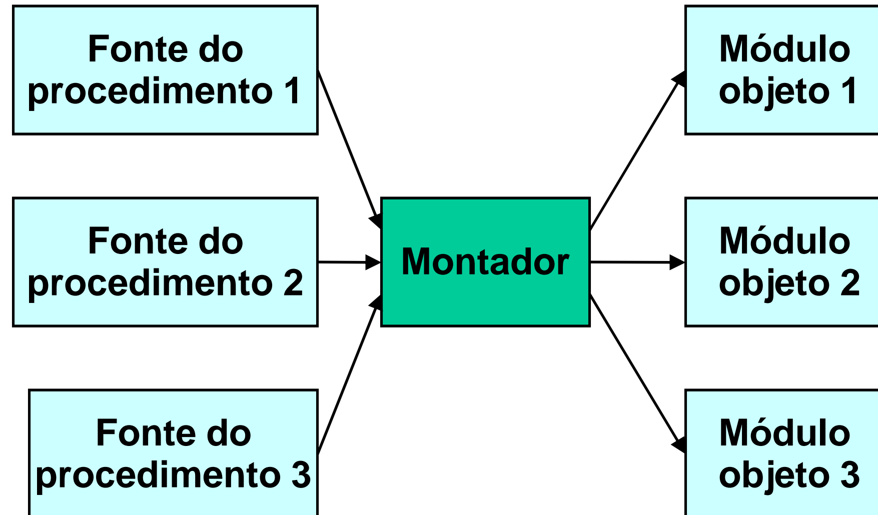
Tradução completa

- Constituída de duas fases (duas passagens):
  1. Montagem dos procedimentos a partir dos arquivos com código em *assembly*.
  2. Ligação dos módulos-objetos resultantes.
- Ao final, teremos um programa binário executável.

Tradução completa



O processo de montagem



- Entrada: arquivos com código *assembly*
- Saída: linguagem de máquina "anotada"
  - Referências a outros módulos
  - Reservas de espaço

UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Montagem em duas passagens

- Tradução imediata nem sempre é possível
  - Problema da referência antecipada:
    - Ex: desvio para endereço L no início, mas L definido no final
- Primeira passagem:
  - Contagem do espaço ocupado por cada instrução
  - Construção da *tabela de símbolos* (*labels*)
  - Salva definição de macros e expande chamadas à medida que são encontradas
- Segunda passagem:
  - Geração do programa objeto para o *Linker*
  - Resolução de símbolos e referências

Montagem em duas passagens

- Duas alternativas:
  - Alternativa 1: ler o programa de entrada 2 vezes.
    - armazenar os símbolos em uma tabela
    - traduzir o programa
    - solução simples
  - Alternativa 2: ler o programa de entrada 1 vez
    - gerar código intermediário e armazenar em memória
      - remover comentários
      - manter somente o essencial
    - pode ser mais eficiente (memória X tempo de E/S)

Passo 1

- Principal papel: montar **tabela de símbolos**
  - Símbolo: label ou valor ao qual é atribuído nome simbólico
  - ex: BUFSIZE EQU 8192
- Contabiliza espaço para instruções
  - ILC – *Instruction Location Counter*
    - Incrementado do tamanho de cada instrução traduzida
    - Usado para definir endereços de instruções com label
    - Iniciado com 0 e incrementado do comprimento da instrução para cada instrução processada
- Memoriza localização dos símbolos encontrados durante o processo: tabela

Exemplo

Rótulo	Opcode	Operandos	Comentários	Comprimento	ILC
MARIA:	MOV	EAX,I	EAX=I	5	100
	MOV	EBX,J	EBX=J	6	105
ROBERTA:	MOV	ECX,K	ECX=K	6	111
	IMUL	EAX,EAX	EAX = I*I	2	117
	IMUL	EBX,EBX	EBX = J*J	3	119
	IMUL	ECX,ECX	ECX = K*K	3	122
MARILYN:	ADD	EAX,EBX	EAX = I*I + J*J	2	125
	ADD	EAX,ECX	EAX = I*I + J*J + K*K	2	127
STEPHANY:	JMP	DONE	Branch to DONE	5	129

Passo 1

- Usa no mínimo 3 tabelas internas
  - Tabela de símbolos
  - Tabela de pseudo-instruções
  - Tabela de opcodes
- Opcional: tabela de literais
  - Quando não tem suporte para operandos imediatos
  - Constantes (ex: ´5´, ´9´);
  - Após passo 1: tabela ordenada e duplicatas removidas

Tabela de símbolos

- Criada no passo 1 para utilização no passo 2
- Informações coletadas sobre símbolos
  - Valor (endereço ou valor numérico)
  - Tamanho do campo de dados
  - Informação de relocação: o símbolo muda de valor se o programa for carregado em um endereço diferente?
  - Regras de escopo: acessível ou não fora do procedimento?

Símbolo	Valor	Outras
MARIA	100	
ROBERTA	111	
MARILYN	125	
STEPHANY	129	

Tabela de símbolos

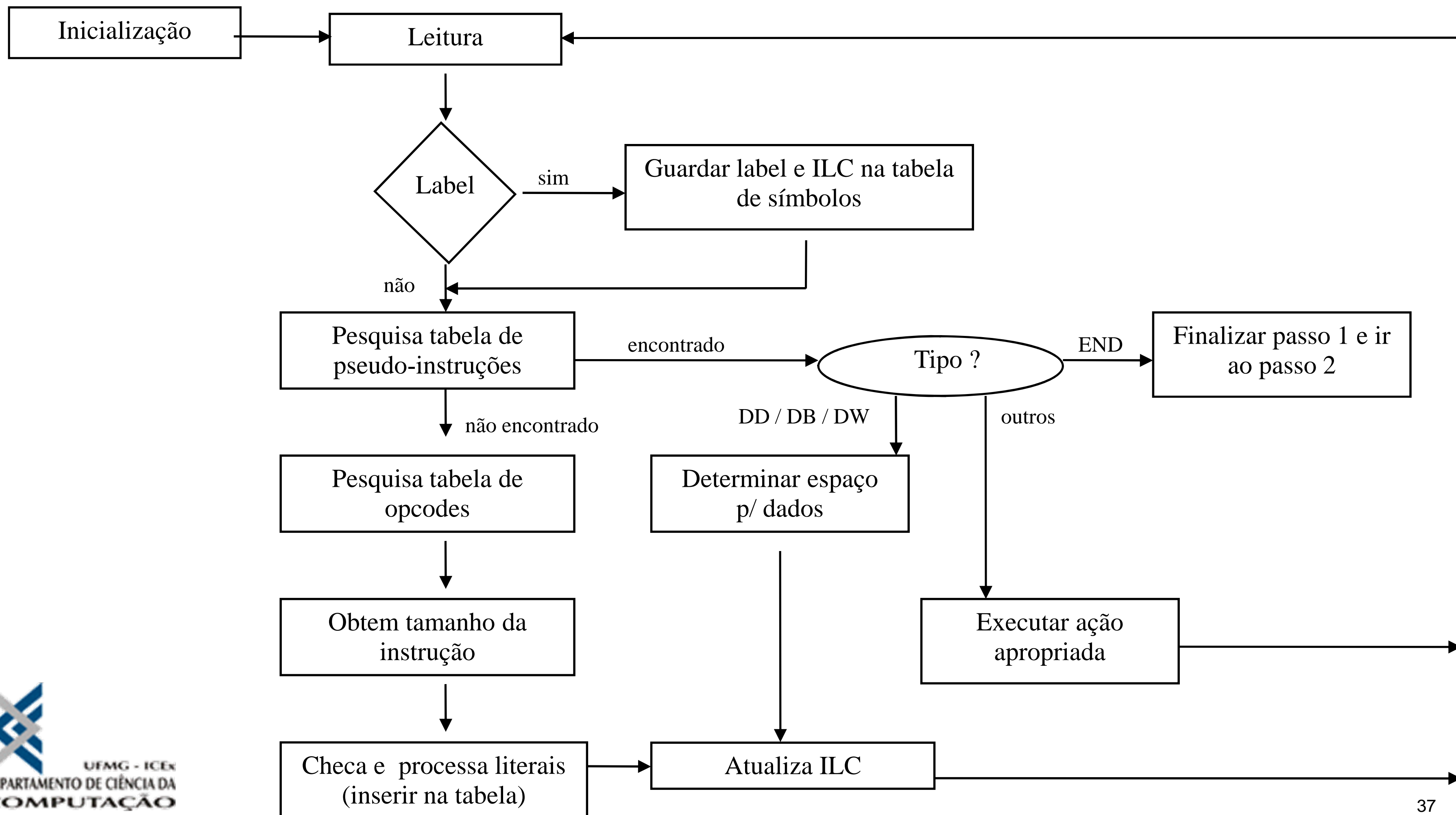
- Várias maneiras de organizar a tabela
  - Memória associativa: par (símbolo,valor):
    - Forma mais simples: vetor de registros
    - Forma mais esperta: tabela hash

Tabela de opcodes

- Uma entrada para cada opcode simbólico na linguagem assembly
  - opcode simbólico,
  - operandos,
  - valor numérico do opcode,
  - comprimento
  - classe da instrução (número e tipo dos operandos)

Opcode	1º Operando	2º Operando	Opcode (hexa)	Comprimento da instrução	Classe
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

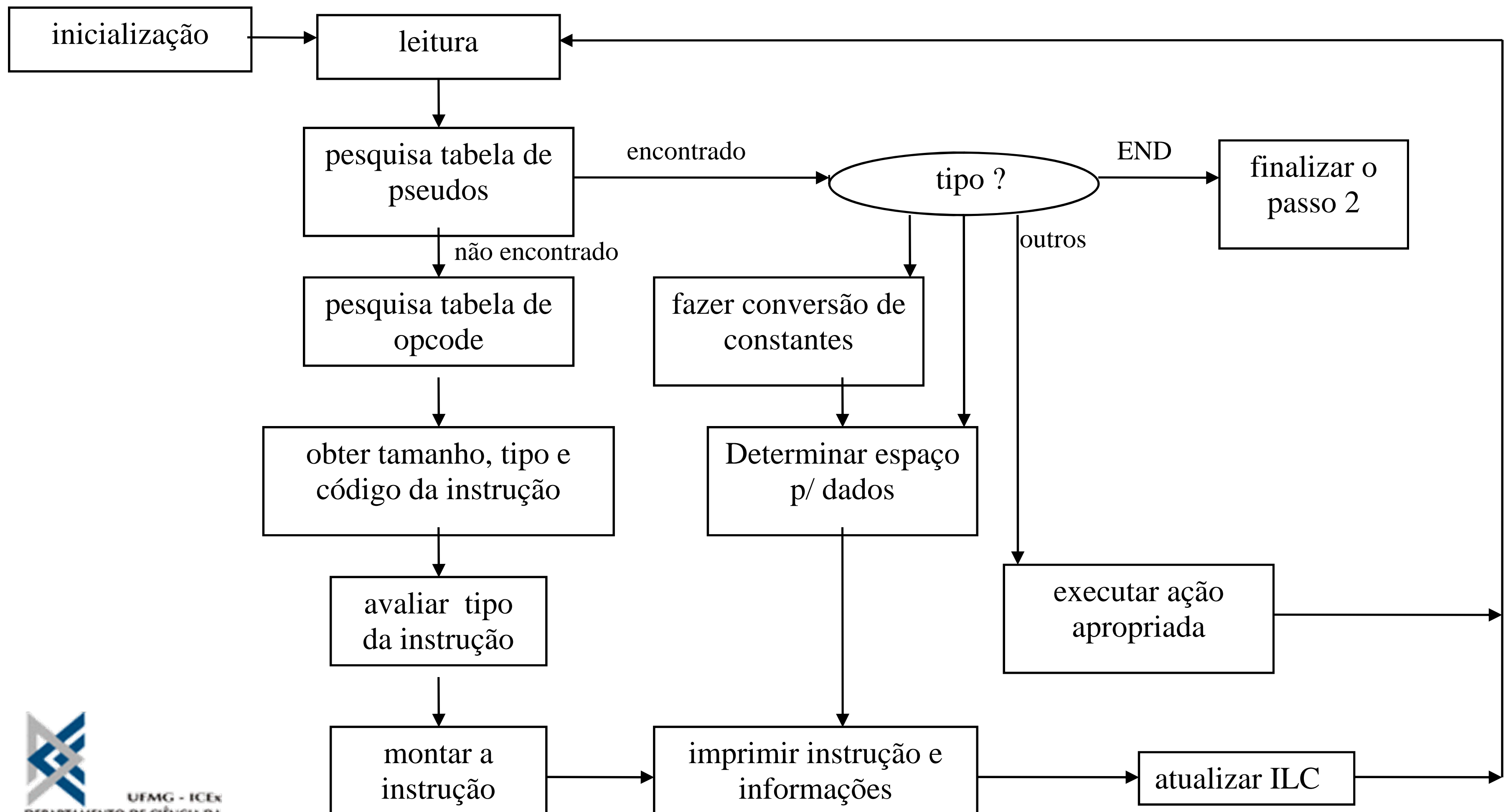
Passo 1



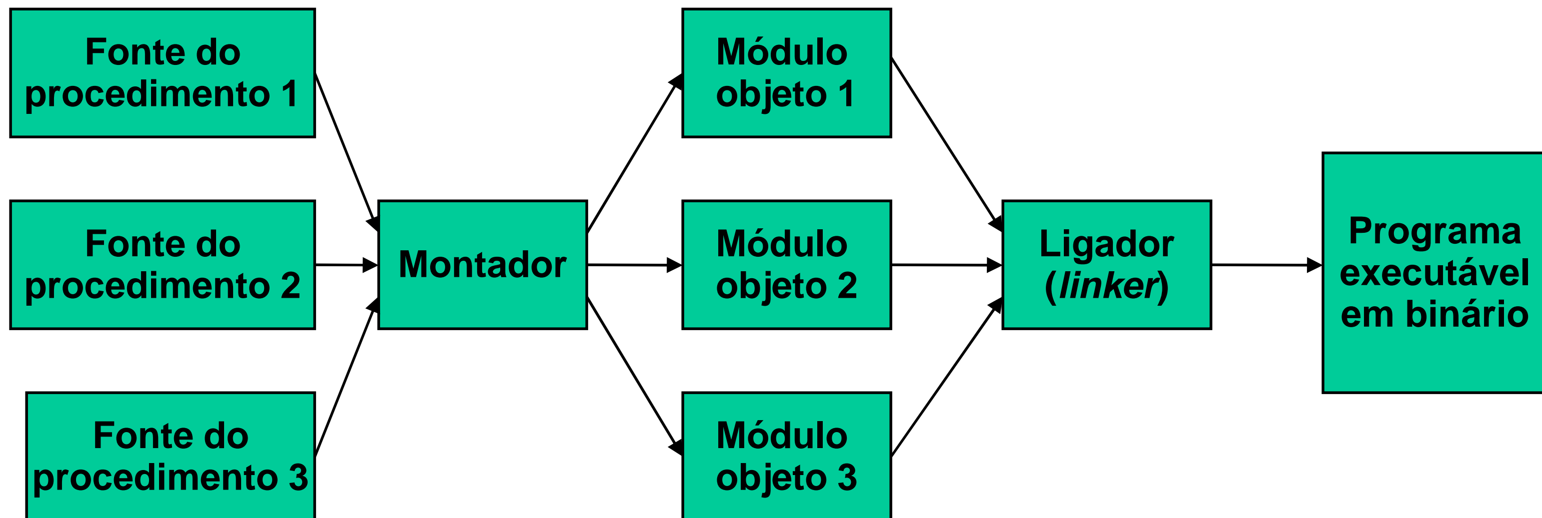
Passo 2

- Tradução propriamente dita do programa
  - Produz código
  - Produz também informação extra que o ligador precisa pra ligar procedimentos montados em tempos diferentes
- Leitura sequencial do código
  - Acessos às tabelas de instruções e de símbolos
  - Geração de código binário
- Tratamento de erros
  - Símbolos desconhecidos ou definidos múltiplas vezes
  - Opcode fornecido com # insuficiente/excessivo de operandos
  - Declaração END faltante, etc...

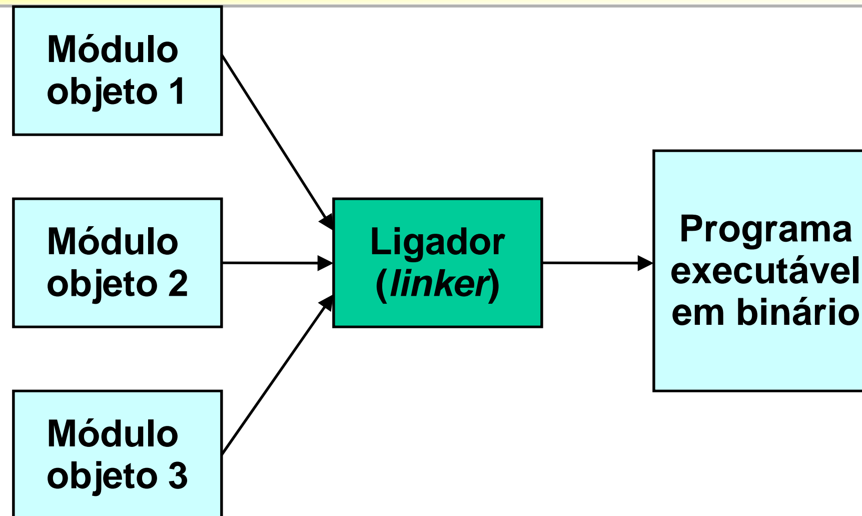
Passo 2



Tradução completa



Ligação dos módulos



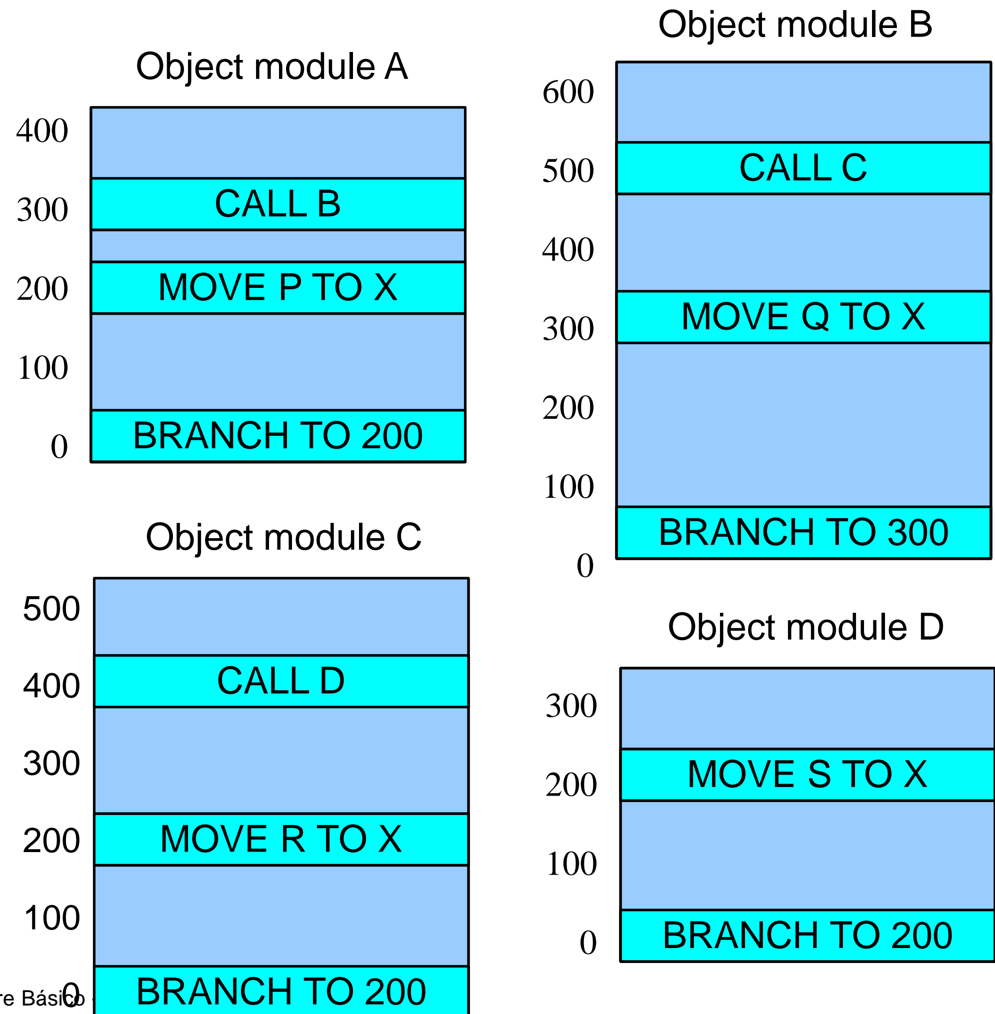
- Para simplificar o desenvolvimento, cada programa pode ser dividido em módulos, cada um com alguns procedimentos.
- Para um programa poder ser executado, todos os procedimentos devem ser ligados.

Ligador

- Funções:
  - Ligação: une os procedimentos e resolve as referências entre os módulos.
  - Alocação: separa espaço na memória para o programa.
  - Relocação: ajusta os endereços que dependem da posição do programa na memória.

Exemplo de ligação

Módulos isolados:



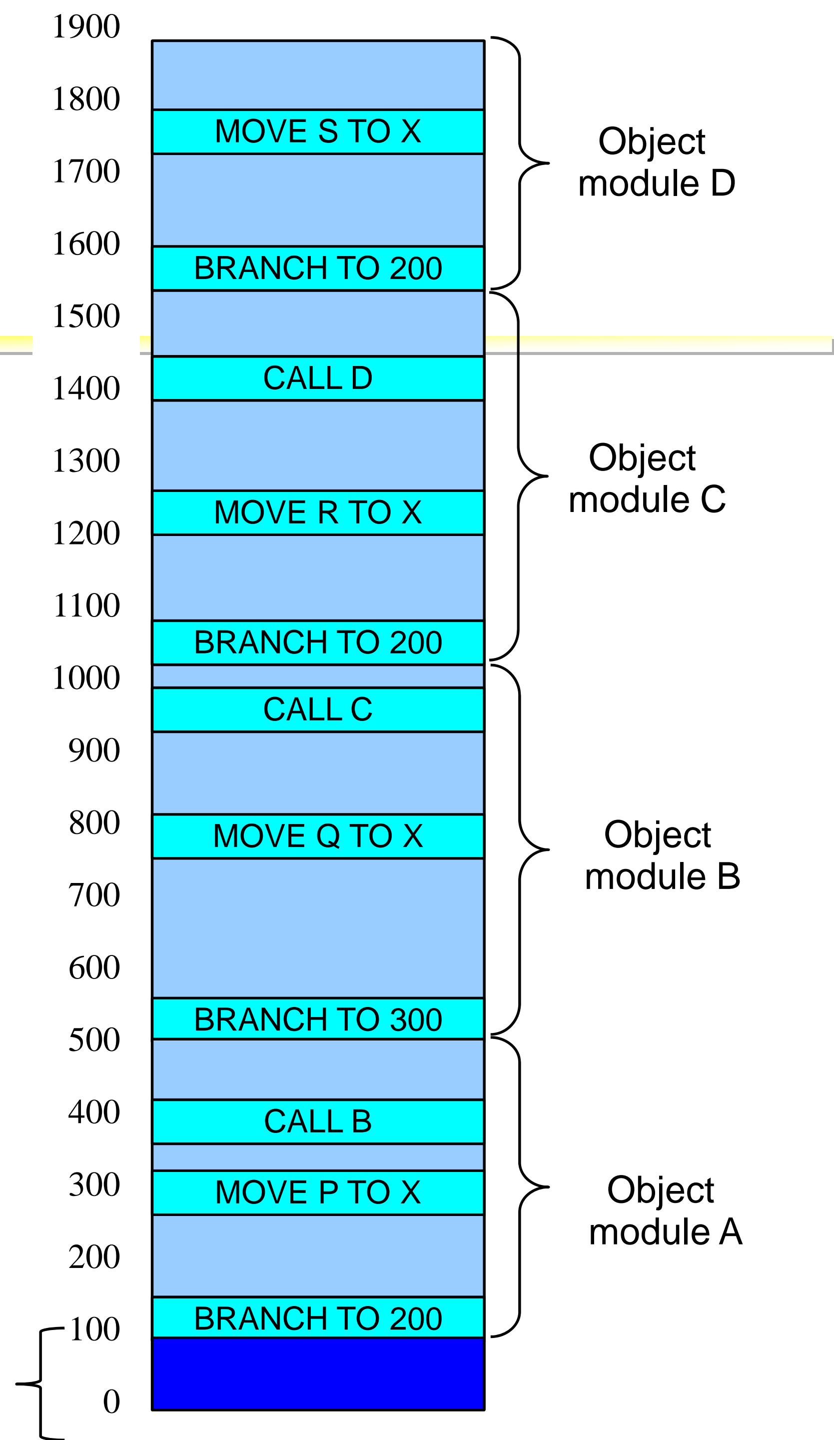
UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Exemplo de ligação

Para executar o programa, o ligador traz os módulos para a memória principal para formar a imagem do programa binário executável.

O que acontece se o programa for executado assim?

Seção da memória com início em 0 é usada para vetores de interrupção, comunicação com sistema operacional, etc



Problemas tratados pelo ligador

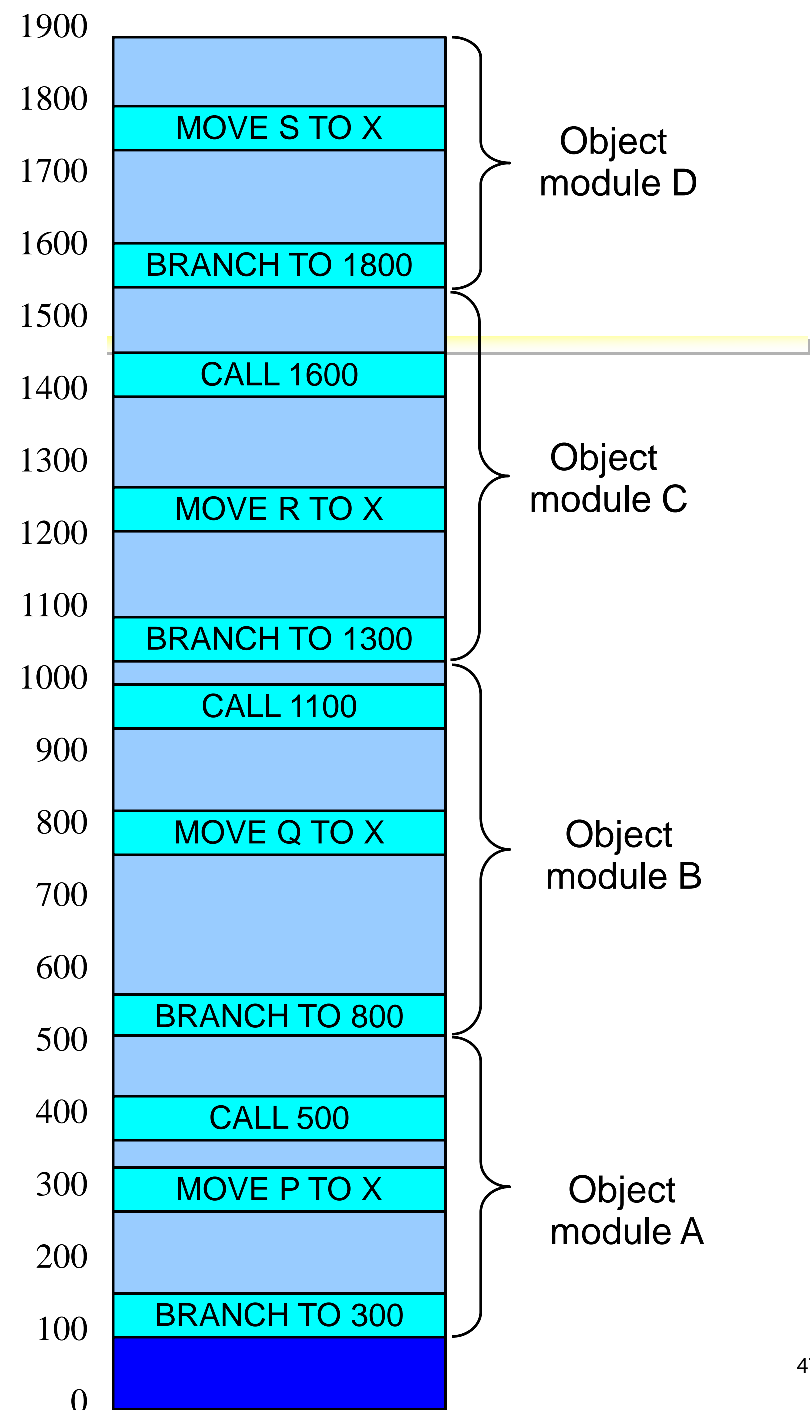
- Problema da relocação: todas as instruções de referência à memória falhariam pois contém endereços determinados assumindo cada módulo com um espaço de endereçamento separado
  - Se espaço de endereçamento segmentado em tese não haveria problema
    - Família Pentium tem suporte em hardware
    - Porém OS/2 é único SO que tem suporte a segmentação
- Problema de referência externa: assembler não tem como saber endereço de um procedimento externo (CALL B), pois este só é conhecido em tempo de ligação

Fusão dos espaços de endereços dos módulos objetos em um único espaço

1. Constrói tabela com módulos objetos e seus comprimentos
2. Com base na tabela, designa um endereço de início a cada módulo objeto
3. Para cada instrução com referência à memória, adiciona ao endereço sendo referenciado uma constante de relocação iugal ao endereço de início do módulo em questão
4. Para cada instrução que referencia procedimentos externos, insere os endereços destes procedimentos no lugar adequado

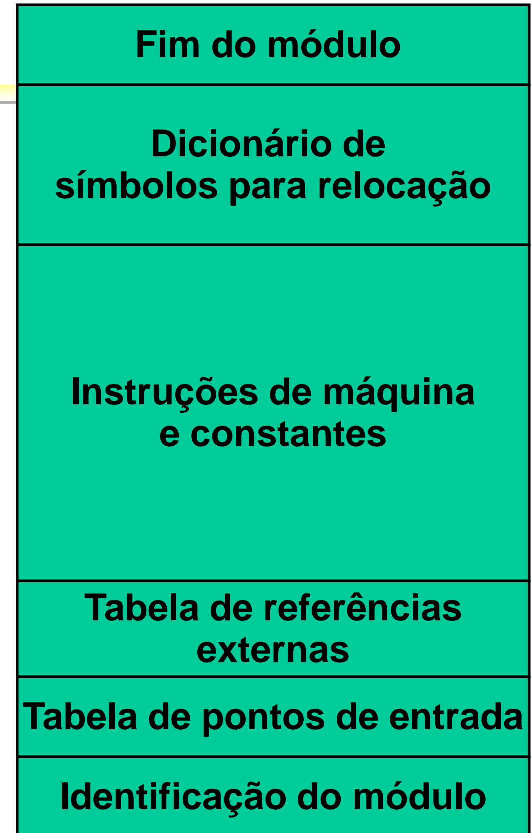
Exemplo de ligação

Módulos relocados:



UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Estrutura de módulos objetos



- Identificação: nome e tamanho das partes do módulo
- Pontos de entrada (nome e endereço)
  - Símbolos internos que podem ser referenciados de outros módulos- PUBLIC
- Tabelas de referências externas
  - Símbolos referenciados pelo módulo, ausentes do mesmo - EXTERN
- Instruções, variáveis, constantes (código objeto)
  - Única parte carregada em memória
  - Outras partes é para uso do ligador
- Dicionário para relocação: contém endereços que devem ser relocados (quais instruções na parte 4 precisam ser relocadas)
- Fim de módulo: marcação especial e (talvez) soma de verificação

Tarefas do Ligador

- Maioria dos ligadores requer duas passagens
- Primeira passagem:
  - Colocação de todos os módulos na memória
  - Contabilização dos espaços ocupados
  - Determinação dos endereços exportados
  - Verificação de referências externas
  - Constrói tabelas de nomes, comprimentos dos módulos, tabelas com pontos de entrada e referências externas
- Segunda passagem:
  - Relocação: Alteração de endereços em cada módulo
  - Ligação efetiva

Não está tudo resolvido ainda....

- Um programa pode ser carregado/descarregado da memória principal muitas vezes durante a execução
  - Paginação: partes do programa (páginas) podem ser retirados da memória principal para serem recarregados depois

- O que acontece com as referências à memória?

Tempo de vínculação

- Momento em que é determinado o endereço de memória principal correspondente a um símbolo

- Se uma instrução com endereço de memória for movida após a vinculação, ela será incorreta

- Dois problemas:
  - Mapeamento de símbolo para endereço virtual
  - Mapeamento de endereço virtual para endereço físico

Oportunidades para vinculação

- Durante a escrita do programa
- Durante a tradução *assembly*
- Durante a ligação (geração do executável)
- Durante a carga do programa na memória
- Quando um registrador base é carregado
- Quando a instrução com o endereço é executada

Relocação dinâmica

- Programas podem mudam de lugar na memória
  - Sistemas de espaço de endereçamento único
- Endereços internos podem ser alterados
  - Tabelas de tradução (paginação)
  - Registrador de relocação (endereços relativos)
    - Contém endereço de memória física do início do programa
    - Hardware adiciona endereço de relocação a todos endereços de memória antes de enviá-los a memória
    - Atualizado toda vez que programa recarregado
  - Acessos relativos ao PC
- Memória virtual acaba com esse problema!
  - Ligador responsável pelo mapeamnto símbolo : endereço virtual

Ligação Dinâmica

- Ligação de um módulo somente quando um de seus procedimentos é chamado pela primeira vez
  - Evita carregar em memória, módulos ue poderão não ser utilizados (procedimentos raramente chamados)
- Pioneiro: MULTICS
- Outros
  - DLL no Windows: economia de memória pois código é compartilhado entre aplicações
  - Bibliotecas compartilhadas no Unix
    - Ex: biblioteca padrão do C

MULTICS

- Cada programa tem um segmento de ligação: tabela com par (nome, endereço virtual) para cada procedimento
  - Os endereços virtuais são iniciados com valor inválido
- Instruções CALL apontam para entrada correspondente na tabela
- Quando a primeira chamada a um procedimento externo é executada:
  - Acesso a endereço inválido causa exceção para ligador dinâmico
  - Módulo é carregado e seu endereço virtual é atualizado na tabela
  - Chamadas subsequentes funcionam normalmente

Resolução de endereços

- Escrita do programa: endereços diretos
- Tradução do programa: resolução de nomes
- Durante a ligação: referências externas
- No momento da carga: relocação
- Na carga de um registrador de indexação
- Sob controle do sistema: relocação dinâmica
- Na execução de uma instrução: ref. relativas
- Carga e ligação sob demanda: DLLs