PCS 3216
Sistemas de Programação

Aulas 19 e 20

Implementação de
Montadores Relocáveis e Ligadores

Exercício de planejamento

(repetido aqui por conveniência)

- Constate que são idênticas as operações de
  - Um **montador**, ao construir uma tabela de símbolos, estimulado pela identificação de rótulos declarados nos campos de rótulo (**definição**) e de operando (**referência**).
  - Um **ligador**, ao construir uma tabela de endereços simbólicos encontrados nos módulos que estão compondo um programa, estimulado pela identificação de símbolos marcados como *exports* ou *entry-points* (**definição**) e como *imports* ou *externals* (**referência**).
- Considerando como evento de entrada a identificação dos símbolos, utilize um motor de eventos, para com base nele, projetar, implementar e testar a lógica de um procedimento que :
  - Constrói uma tabela de símbolos, para uso por parte de um **montador** que recebe como entradas informações extraídas do programa-fonte por ele analisados.
  - Constrói uma tabela de resolução de referências externas, para uso por parte de um **ligador** cujas entradas provêm dos programas-objeto relocáveis por ele manipulados.
- Complete adequadamente esta especificação, para que ela possa facilitar a obtenção de partes operantes do montador e do ligador.

- Só para recordar, e para facilitar a retomada deste assunto, o slide anterior simplesmente repete aqui o enunciado do **exercício da aula 16**, que solicitava o **planejamento** conjunto da construção de **ligadores e de montadores relocáveis**, dada sua semelhança em muitos aspectos.

DETALHAMENTO DO PROJETO

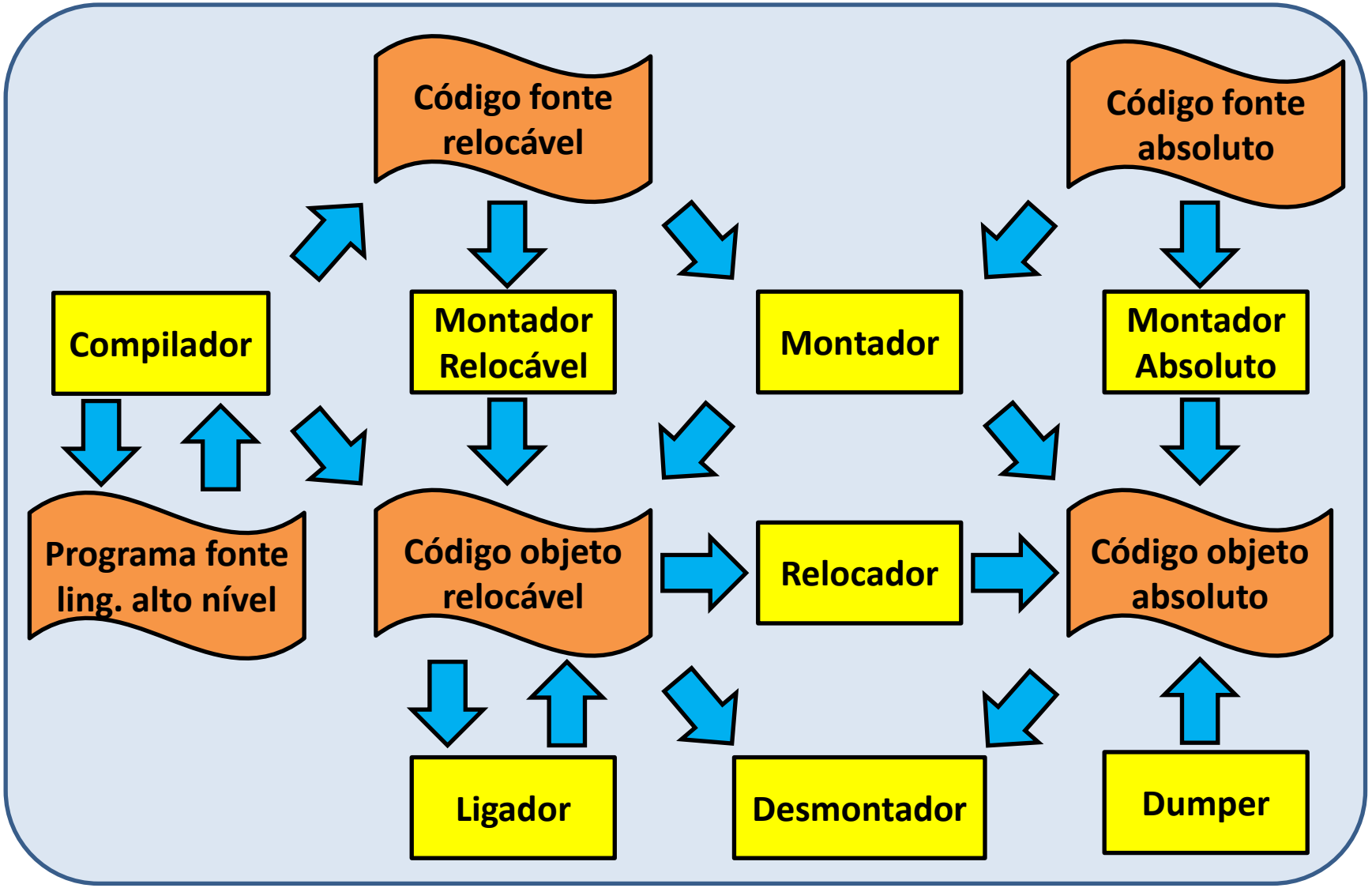
Objetivo

- O objetivo principal desta aula é encaminhar a implementação de um montador para linguagem simbólica relocável e de um ligador-relocador para converter em código executável (absoluto) os programas-objeto relocáveis a ele submetidos.

Compatibilidades

- Como esses programas de sistema são utilizados conjuntamente, é indispensável que suas interfaces sejam totalmente compatíveis entre si.
- A figura seguinte evidencia, no âmbito dos sistemas de programação, a necessidade de haver compatibilidade de formatos para que possa haver intercâmbio de informação entre os programas de sistema.
- Destacam-se os formatos dos textos-fonte e dos códigos-objeto absoluto e relocável, tanto binários como nos formatos numéricos ASCII, decimal, hexadecimal e octal.

Algumas Exigências de Compatibilidade de formatos no Sistema de Programação



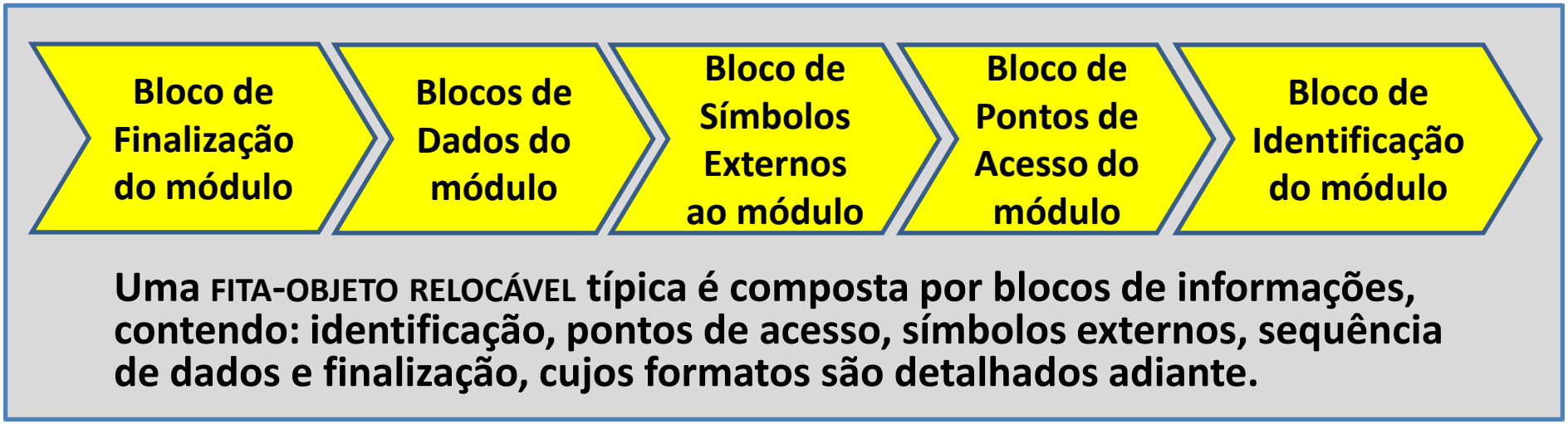
Formato do código-objeto relocável

- Entre os **ligadores** e os **montadores** de linguagens simbólicas **relocáveis**, esta compatibilidade deve ser especialmente respeitada adotando-se exatamente um formato único para os códigos-objeto relocáveis, pois constituem o formato adotado tanto para a saída dos montadores como para a entrada dos ligadores e desmontadores.

Formato do código-objeto absoluto

- O mesmo se pode afirmar quanto ao formato do código-objeto absoluto, que deve guardar perfeita compatibilidade entre a entrada do loader (bootstrap) e as saídas do dumper, do montador absoluto e do ligador-relocador que estamos desenvolvendo.

Informações já conhecidas sobre os formatos



Uma FITA-OBJETO RELOCÁVEL típica é composta por blocos de informações, contendo: identificação, pontos de acesso, símbolos externos, sequência de dados e finalização, cujos formatos são detalhados adiante.

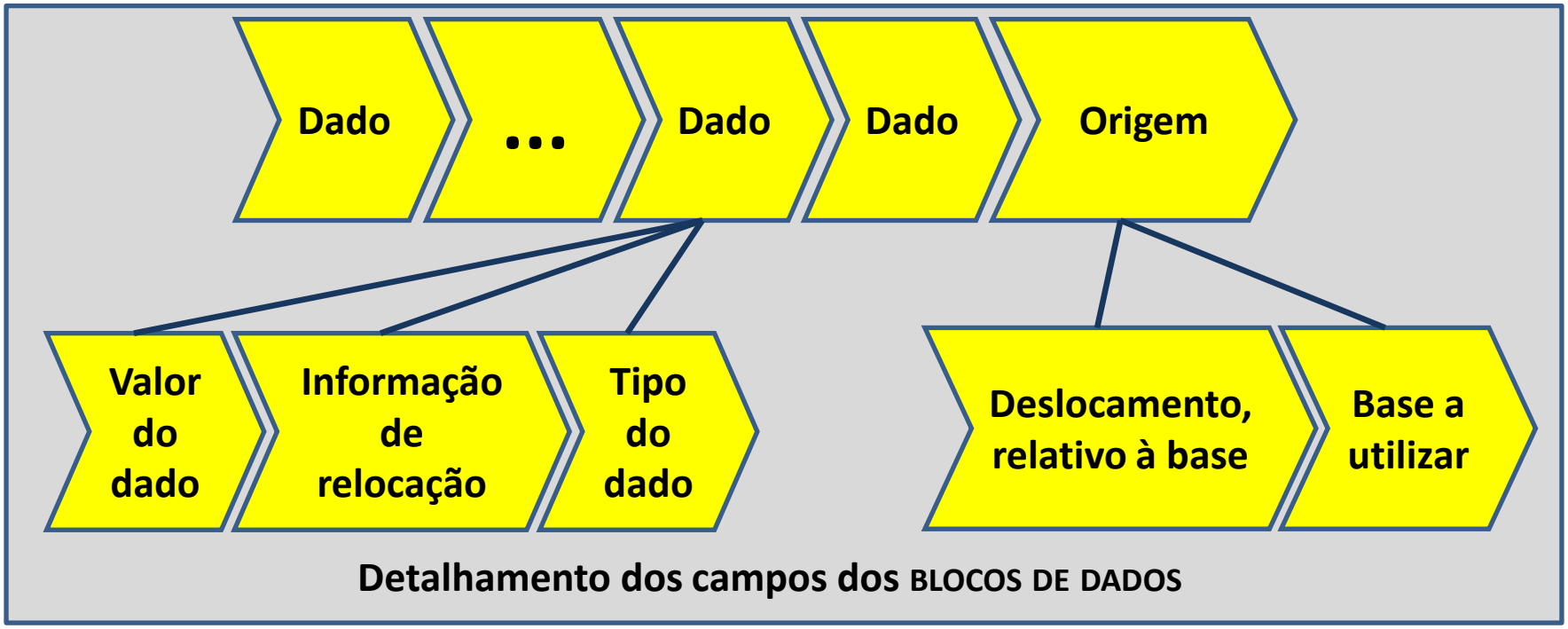
Estrutura geral de um BLOCO DE INFORMAÇÕES de um programa-objeto relocável. As informações do bloco variam caso a caso, conforme o tipo do bloco a que pertencem.

BLOCO DE IDENTIFICAÇÃO - Conforme o sistema, certas informações das áreas de programas, dados, apontadores ou pilhas podem ser omitidas. Eventualmente, alguns sistemas incluem alguma área adicional, como, por exemplo, a de *common*. O tipo do módulo indica tratar-se de programa principal, sub-rotina, *overlay* etc.

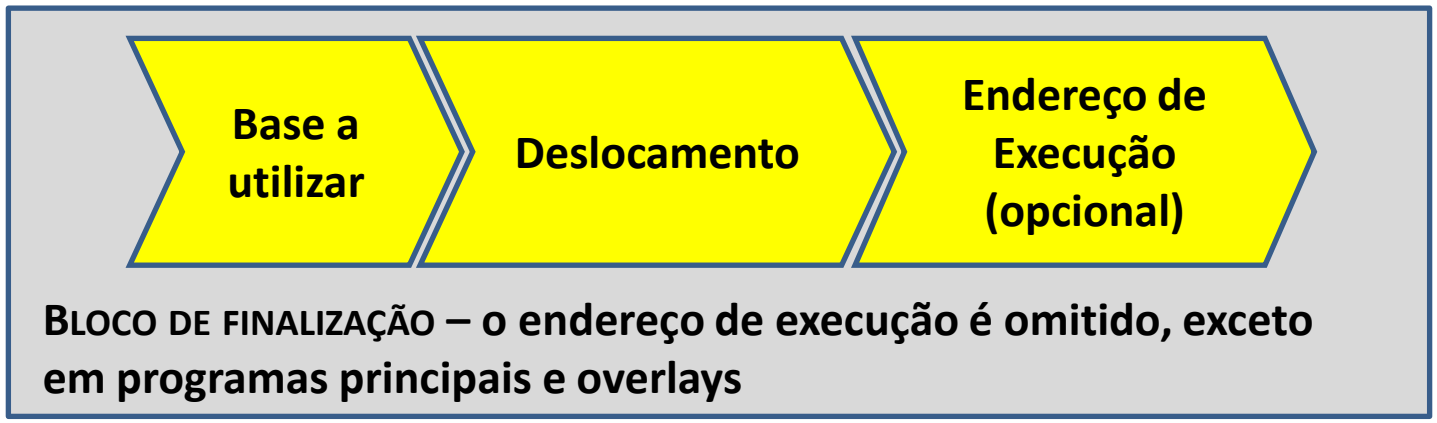
Um BLOCO DE PONTOS DE ACESSO é formado, em seu campo de informações do bloco, por uma sequência de estruturas de informações, uma para cada ponto de acesso, e cada qual tendo o formato esboçado na figura: o nome, o tipo do ponto de acesso (programa principal, sub-rotina, dado etc.) e seu endereço relativo (indicação da base de relocação a utilizar e deslocamento).

FORMATO PARA PASSO ÚNICO - Observar a inversão na sequência dos blocos em relação ao esquema em dois passos, para facilitar a geração. Notar a separação física inserida propositalmente entre o último bloco de dados e o restante das informações da fita, para facilitar o manuseio desta na época da ligação e relocação.

Alguns montadores registram neste BLOCO DE SÍMBOLOS EXTERNOS apenas símbolos externos efetivamente referenciados no programa, e não todos os declarados nas pseudo-instruções de definição de símbolos externos. Outros geram um bloco separado para cada símbolo.



Detalhamento dos campos dos BLOCOS DE DADOS



BLOCO DE FINALIZAÇÃO – o endereço de execução é omitido, exceto em programas principais e overlays

Formato fonte

- É também bastante evidente que, como alguns dos módulos do sistema de programação também geram informação na forma de **textos-fonte**, os formatos adotados devem também ser uniformes.
- Mais que uma **uniformidade de formato**, cabe ressaltar a necessidade de **uniformidade de linguagem**, de forma que os comandos de qualquer programa sejam **correta e igualmente interpretados** por todo o sistema.
- É o que acontece no caso dos **compiladores**, que também podem, entre outras possibilidades, **gerar programas-fonte como saídas**.

Linguagem-fonte simbólica

- Uma compatibilidade muito desejável é aquela que se pode estabelecer entre as linguagens simbólicas absoluta e relocável.
- Isto pode ser obtido facilmente
  - Padronizando a notação usada para a representação das instruções de máquina nos dois casos.
  - Identificando claramente o tipo dos operandos (constante, endereço simbólico, absoluto ou relocável...)
  - Unificando o programa montador, para que possa tratar igualmente programas-fonte absolutos e relocáveis.
  - Implementando um conjunto de pseudo-instruções bem escolhido, de modo que essa unificação se viabilize.

Número de passos

- Sabe-se bem que montadores de dois passos são menos complexos e menos artificiosos.
- Embora não seja obrigatório, trata-se de uma boa opção para este projeto.

Linguagem de implementação

- Desejando-se construir um montador auto-residente e auto-compilável, uma possibilidade é a de implementá-lo na própria linguagem da MVN.
- Todavia, não há muito suporte disponível para isso, logo convém elaborar um montador cruzado, escrito em uma linguagem de alto nível qualquer, para ser executado em outra máquina, gerando para a MVN um código-objeto compatível com os que já foram construídos.
- A grande disponibilidade de compiladores, montadores e outras ferramentas de desenvolvimento também dá ao usuário a liberdade de escolha do ambiente e da linguagem de implementação considerados mais adequados.

Recomendação

- Para evitar atrasos e outras dificuldades desnecessárias, **não** escolha neste momento linguagens ou ambientes de desenvolvimento com os quais não tenha familiaridade.
- É prudente **agilizar o trabalho** privilegiando o uso de alguma linguagem de programação com a qual se tenha **fluência**, e um ambiente de desenvolvimento de cujo uso já se tenha **prática**.
- Definitivamente, **esta não é uma boa ocasião para aventuras exploratórias** nessa área, por ser desnecessário e não haver tempo para inevitáveis dúvidas, contratempos, enganos e correções.

A LINGUAGEM SIMBÓLICA

A Linguagem Simbólica

- As linguagens absolutas vinculam o código a posições fixas de memória.
- Programas escritos em linguagem absoluta só funcionam nos endereços para os quais foram projetados, devido às referências absolutas de endereçamento neles contidas.
- Não é prática a construção de bibliotecas de uso geral baseadas em linguagens absolutas.
- Contorna-se o problema introduzindo-se o conceito de relocabilidade.
- Programas relocáveis apresentam referências a endereços não resolvidos.
- Neles, a resolução dos endereços (vinculação a posições fixas) é postergada.
- Endereços não resolvidos (relocáveis) são endereços relativos à posição de memória a partir da qual o programa irá ser alocado.
- Isto viabiliza a construção de bibliotecas relocáveis de programas e rotinas.
- A programação relocável propicia o desenvolvimento de programas em módulos separados relativamente independentes.
- Introduzindo-se o conceito de endereçamento simbólico, é possível, através da mútua referencia dos módulos através de rótulos simbólicos, compô-los de acordo com a necessidade, obtendo assim os programas desejados.

Pseudo-instruções p/ montador absoluto

- São formas sintáticas parecidas com as instruções de máquina, que, na realidade, propiciam ao programador a passagem de informações acerca do código fonte para o montador.
  - A maioria das pseudo-instruções não geram código
  - Servem para controlar a montagem do código de máquina
- Algumas pseudo-instruções típicas:
  - **ORG** – (origem) informa o endereço de início do programa
  - **END** – (término) indica o final físico do texto
  - **EQU** – (equivalência) define sinônimos
  - **DB** – (*define byte*) preenche bytes inicializados
  - **DW** – (*define word*) preenche palavras inicializadas
  - **DS** – (*define storage*) vetor na memória, não inicializada

Pseudo-instruções p/ montador relocável

- As diferenças entre textos simbólicos absolutos e relocáveis de linguagem simbólica reside no conjunto de pseudo-instruções disponíveis e sua interpretação.
- Pseudos típicas de montadores relocáveis são as seguintes:
  - **ENTRY** – define pontos de acesso (*entry-points*) a um módulo
  - **EXTERNAL** – indica *entry-points* de outros módulos, referenciados no texto
  - **NAME** – define o nome do módulo corrente.
  - **ORG** – define nova origem do código, usualmente relocável.
  - **END** – indica o final físico do módulo, e o endereço de execução (prog.principal)
  - **EQU** – define sinônimos, atribui nome a endereços (expressões).
  - **SET** – dá nome e altera o valora de variáveis em tempo de montagem.
  - **AIF** – efetua desvio condicional testando variáveis em tempo de montagem.
  - **ANOP** – define rótulos, específicos para desvios em tempo de montagem.
  - **AGO** – executa desvio incondicional de montagem para diante.
  - **AGOB** – executa desvio incondicional de montagem para trás.
  - **OPDEF** – redefine mnemônicos.
- Algumas pseudos acima podem ser utilizadas na programação absoluta.
- O montador relocável traduz linguagem simbólica relocável em código-objeto relocável.

IMPLEMENTAÇÃO DO MONTADOR

Implementação de montadores absolutos de dois passos

- Os montadores de dois passos são os mais conhecidos e mais difundidos.
- Conceitualmente, esses programas são os mais simples, exigindo portanto menores esforços para sua construção e implantação.
- Adicionalmente, desta simplicidade resulta um programa mais compacto e econômico.

Montador Absoluto, passo 1

- O primeiro passo do montador destina-se especialmente às tarefas seguintes:
  - Construção da **tabela de símbolos**
  - Consulta à **tabela de mnemônicos**
  - Construção da **tabela de equivalências**
  - Cálculo de **endereços** das instruções
  - **Teste de consistência** da tabela de símbolos
  - Detecção de **erros**
  - Geração de **mapas de memória**
  - Geração de tabelas de referências cruzadas

Montador Absoluto, passo 2

- No segundo passo, o montador se dedica especialmente às seguintes atividades:
  - Consulta à **tabela de códigos**
  - **Montagem do código**-objeto
  - Avaliação das **expressões dos operandos**
  - Geração de **listagens** formatadas
  - Geração de **programa-objeto**

Principais Estruturas de Dados

- **Tabela de símbolos**
  (símbolo - endereço - definido - referenciado)
- **Extensão** da tabela de símbolos para a geração das **referências cruzadas**
  (linha de definição - *link* para ordem alfabética - ponteiro para lista de referências)
- **Lista de referências** para referências cruzadas
  (*link* - número da linha)
- **Tabela de mnemônicos e códigos**
  (mnemônico - código - classe)
- **Tabela de equivalências**
  (símbolo - *link*)
- **Área de saída**
  (bloco de código objeto gerado)

Lógica do montador absoluto de 2 passos

- Contador de instruções (C.I.) ← 0. Passo ← 1.
- Leitura de uma linha, ignorando comentários; se passo =2 então listar a linha;
- Se a linha tiver rótulo:
  - Procurar o rótulo na tabela dos símbolos.
  - Se já existe e foi definido, gerar mensagem de erro .
  - Se já existe Indefinido, atribuir-lhe o endereço do contador de instruções .
  - Se não existe, inserir na tabela, e atribuir o endereço do contador de instruções .
  - Marcar como definido.
  - Atualizar a tabela de referências cruzadas.
- Analisar o mnemônico:
  - Procurar na tabela de mnemônicos .
  - Se não achar, gerar mensagem de erro.
  - Atualizar contador de instruções conforme o tipo de mnemónico.
  - Se o mnemônico exigir operando:
    - analisar o operando
      - Incluir eventuais símbolos novos na tabela de símbolos .
      - Se não constaram na tabela, marcar como indefinidos.
      - Atualizar a tabela de referências cruzadas.
    - Avaliar operando (expressão)
    - Se passo = 2 e não for pseudo instrução, Montar código objeto.
  - Se for pseudo instrução:
    - ORG – Modificar contador de instrução conforme o valor do operando.
    - DB,DW – Se passo =2, gerar o código objeto associado .
    - EQU – Se passo = 1, atualizar a tabela de equivalências .
    - END – Se passo = 1, fazer passo ← 2. Se não, terminar a montagem.
- Voltar à leitura de nova linha.

IMPLEMENTAÇÃO DO LIGADOR

Ligadores (*linkers*)

- Cuidam da resolução de **endereço simbólicos externos**.
- Complementam funcionalmente os alocadores
- Sistemas antigos fundiam **ligadores** e **alocadores** em um único programa (***linking loaders***)

Funções de um ligador

- **Resolver endereços** simbólicos externos.
- **Unir módulos** gerados separadamente **resolvendo referências** mútuas.
- **Construir fitas-objeto relocáveis** como resultado da união de fitas correspondentes a módulos menores.
- Gerar **mensagens de erro** de ligação (símbolo duplamente definido).
- Gerar **tabelas de símbolos externos resolvidos**.
- Gerar **listagem de símbolos externos** referenciados **não resolvidos**.

- Permitem o **uso de bibliotecas**, viabilizando a ligação entre módulos.
- Permitem o **desenvolvimento gradativo** de sistemas, através da **ligação parcial** dos módulos (não exige que todos os símbolos externos estejam resolvidos)
- Os ***linking loaders*** exigem resolução total de símbolos externos e relocam o código objeto, **gerando fitas objeto absolutas**
- Os **ligadores preservam todas as tabelas** de símbolos externos no código objeto.

Estrutura lógica dos ligadores

- Como os montadores, a lógica dos ligadores tem como finalidade a resolução de endereços simbólicos.
- Existe uma perfeita correspondência entre:

Montador	**Ligador**
Campo de **rótulo**	Símbolo no bloco de **pontos de acesso**
Símbolo no campo de **operandos**	Símbolo no bloco de **símbolos externos**
Referência **simbólica**	Referência a **símbolo externo**
Tabela de símbolos	**Tabela de resolução de símbolos externos**

Comparação entre ligador e montador

- Devido a essa semelhança, o programa ligador apresenta uma forte **analogia estrutural** com o programa montador.
- Suas entradas e saídas são programas relocáveis completos
- Alguns ligadores permitem promover **ligações** até mesmo **com programas já alocados ou já ligados**.

Esboço da lógica de um ligador de dois passos – PASSO 1

- Iniciar as tabelas de endereço simbólicos externos referenciados e de pontos de acesso.
- Iniciar bases de relocação local em zero.
- Enquanto houver fitas relocáveis a ligar
  - Ler uma fita. Guardar uma cópia para uso no passo 2.
  - Extrair dos blocos de pontos de acesso os símbolos e endereços e preencher a tabela de pontos de acesso com os endereços relativos corrigidos pelas bases de relocação.
  - Usando as informações extraídas do bloco de identificação, atualizar as bases de relocação.

  [Nesse ponto, as Tabelas de Pontos de Acesso e de Símbolos Externos já estão devidamente preenchidas.]

Esboço da lógica de um ligador de dois passos - PASSO 2

- Iniciar as bases de relocação em zero.
- A partir do material coletado no passo 1, gerar blocos de identificação, de pontos de acesso, e de símbolos externos.
- Reler as fitas e locáveis utilizadas no Passo 1. Para cada uma:
  - Extrair informações dos blocos de dados .
  - Atualizar essas informações:
    - Aplicando as bases de relocação local .
    - Substituindo referências simbólicas externas resolvidas por endereços relativos e em seguida aplicando-lhes as bases de relocação locais correspondentes.
  - Montar blocos de dados corrigidos.
  - Gerar os blocos montados .
  - Ignorar blocos de finalização .
- Gerar bloco de finalização do programa construído.
- Gerar mapas de memória.

Enunciado

- Visando à obtenção de uma parte operante de um sistema de programação para a MVN, resta construir dois programas: um montador com capacidade de processar programas simbólicos absolutos e relocáveis, e um ligador-relocador compatível.
- Ambos esses programas deverão ser construídos com base no motor de eventos já desenvolvido, e podem ser escritos em qualquer linguagem de alto nível.