PCS 3216
Sistemas de Programação

Aulas 17 e 18

Programação simbólica relocável

LINGUAGENS SIMBÓLICAS RELOCÁVEIS E SEUS MONTADORES

- Para implementar os conceitos envolvidos na programação relocável, é necessário enriquecer as linguagens simbólicas com um **conjunto adicional de pseudo-instruções**.
- Estas pseudo-instruções se destinam a proporcionar ao programador **ferramentas** através das quais possa ele **especificar** ao montador as diversas **condições** em que deseja **definir os programas** a serem desenvolvidos.

Pseudo-instruções p/ linguagens relocáveis

- Surgem assim **pseudo-instruções típicas das linguagens simbólicas relocáveis**, entre as quais as mais importantes se referem a funções tais como:
  - definição de **pontos de acesso** a um módulo
  - definição dos **símbolos externos** a serem referenciados por um módulo
  - **identificação do módulo**
  - definição do **endereço inicial de execução** do módulo
  - definição de **origem relativa**
  - definição de **equivalências** relativas entre símbolos
  - definição de **separações físicas no código-objeto**
- **Detalham-se a seguir** essas pseudo-instruções.

Definição de pontos de acesso a um módulo

- Através desta pseudo-instrução, o programador indica, em seu módulo, os **rótulos que devem ser feitos globais**.
- Em outras palavras, indica **através de quais pontos do módulo os demais módulos podem interagir** com ele.
- Como uma imagem comparativa bem simplificada, os pontos de acesso funcionam como uma espécie de interface ou **conector de acesso**, disponível para ser utilizado por outros módulos.

Definição dos símbolos externos referenciados por um módulo

- Através desta pseudo-instrução, o programador indica ao montador **que utilizará os rótulos especificados, e estes devem, para isso, ter sido declarados como pontos de aceso em outros módulos**.
- Para que um programa possa ser construído e posteriormente executado corretamente, cada um dos **símbolos externos** assim declarados devem ser, portanto **declarados como pontos de acesso**, em algum **dos outros módulos** a partir dos quais se constrói o programa.

Identificação de um módulo

- Através desta pseudo-instrução, o programador tem a possibilidade de **dar nome** ao módulo, e também de **indicar o particular tipo** de módulo de que se trata.
- Costuma-se encontrar, nos montadores usuais, a possibilidade de identificar diversos **tipos de módulos**, entre os quais destacam-se:
  - **programa principal**;
  - ***overlay***;
  - **função** ou **sub-rotina**;
  - **rotina de biblioteca**;
  - ***driver* de entrada/saída** com interrupção;
  - **programa privilegiado do sistema** operacional etc.

Definição do endereço inicial de execução do módulo

- Esta pseudo-instrução permite **registrar**, no programa-objeto, para uso do Sistema Operacional, a **informação** do endereço inicial de execução do módulo, para sua correta iniciação.
- Quando um módulo define uma **sub-rotina**, os **pontos de acesso** a tal sub-rotina, declarados em pseudo-instrução específica, indicam os seus endereços iniciais de execução.
- No caso de ***overlays*** e **programas principais**, entretanto, é necessário indicar o endereço de partida para o módulo, visto que usualmente seu endereçamento não é feito da mesma forma que o das sub-rotinas.
- Em alguns montadores, essa função é incorporada à pseudo-instrução que indica o **final físico do programa**, na qual esta informação pode ser declarada, na forma de um operando.

Definição de uma origem relativa

- Uma extensão da pseudo-instrução de origem, utilizada em programação absoluta, dá ao programador a possibilidade de **definir**, para seu código, **origens** que sejam **relativas a algum endereço simbólico** qualquer.
- Os montadores encontrados nas máquinas usuais apresentam esta pseudo-instrução implementada em diferentes graus de complexidade, permitindo desde uma origem relativa apenas a **símbolos definidos internamente ao módulo**, até origens relativas a **símbolos globais,** logo**, referentes a outros módulos**.
- Apesar de o módulo ser relocável, alguns montadores permitem definir, inclusive, **origens absolutas**, o que dá ao programador a liberdade de preencher áreas globais de memória, com endereços físicos predeterminados.

Definição de equivalências relativas

- A pseudo-instrução de equivalência pode **relacionar um símbolo** novo, a ser definido pela pseudo-instrução, **com um endereço simbólico** qualquer, envolvendo tipos arbitrários de **endereçamentos: absoluto, relativo, simbólico.**
- Naturalmente, se a equivalência referenciar **símbolos globais**, sua **resolução** só poderá ser efetuada **após a ligação** dos módulos envolvidos, dificultando, portanto, a operação do ligador e do relocador.

Definição de fragmentações físicas no código-objeto

- Alguns montadores permitem que o programador indique quais **trechos do seu código** se referem às áreas de **programa, de dados, de pilha, de apontadores, de *common* etc.**
- Conforme o sistema computacional a que pertence o montador, essas divisões podem ou não ter sentido.
- A pseudo-instrução que especifica tal fragmentação é utilizada apenas para **indicar ao montador em relação a qual das bases de relocação** existentes se referem os endereços relativos definidos no texto simbólico.

Programas modulares e linguagem simbólica relocável

- **Programas modulares** são aqueles **não atrelados a posições fixas de memória** nas quais devam ser carregados para poderem ser executados.
- Uma **linguagem simbólica relocável** pode ser obtida quase simplesmente **adicionando à linguagem absoluta** um conjunto de **pseudo-instruções** similar ao anteriormente **descrito**, bem como estendendo e **reinterpretando as antigas pseudo-instruções**.
- A linguagem assim obtida torna-se bastante **adequada à criação de programas modulares**, já que incorpora todos os **conceitos específicos** anteriormente propostos **para a obtenção de programas modulares**.

Montadores para programas relocáveis

- Neste caso, cabe ao montador **gerar programas-objeto relocáveis** a partir do texto-fonte simbólico relocável.
- Devido à presença de informações (**meta-dados**) adicionais, necessárias à correta interpretação do código-objeto por parte dos programas **ligadores, alocadores e relocadores**, o formato do código-objeto relocável torna-se mais elaborado que o utilizado para códigos-objeto absolutos.
- Assim, o **montador** deve traduzir as informações contidas nas **instruções e nas pseudo-instruções** de que se compõe o texto simbólico (fonte), e representá-las integralmente em formato compatível com os demais programas de sistema.

Meta-dados

- Assim, os **programas-objeto** gerados pelos montadores das linguagens **relocáveis** devem apresentar uma **estrutura própria**, diferente da que foi vista no caso dos programas-objeto absolutos.
- Isto se deve ao fato de que os **programas-objeto relocáveis** apresentarem um formato que comporta **meta-dados**, informações adicionais que em sua maior parte carecem de significado no caso das linguagens absolutas.
- Por essa razão, a **estrutura de um programa-objeto relocável** é muito **mais complexa** que a de um programa absoluto, pois costuma, em geral, refletir a estrutura do texto-fonte simbólico que lhe deu origem.

Fita-objeto relocável típica

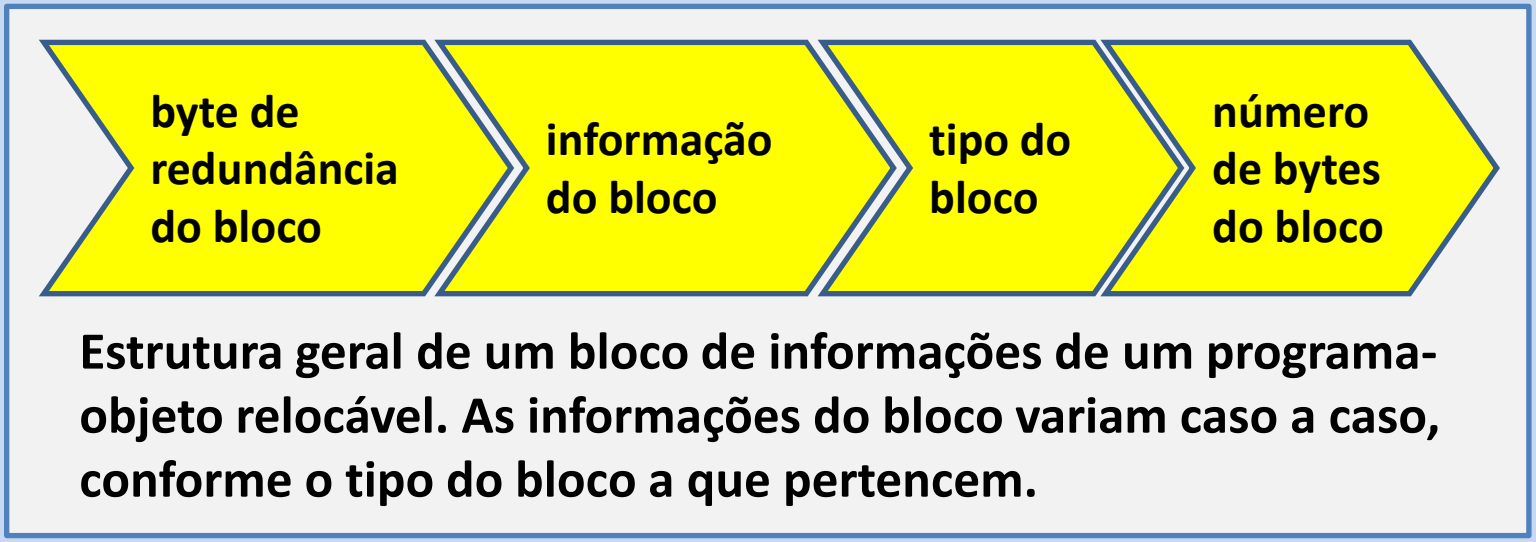
- Por essa razão, os inúmeros **computadores comercialmente disponíveis** apresentam-se muito **diversificados** em relação aos **formatos** adotados para os **programas-objeto relocáveis** que utilizam em seus sistemas de programação.
- No entanto, há uma **grande semelhança conceitual** entre eles, podendo por isso o conteúdo de um **programa-objeto relocável típico** ser esboçado através do esquema a seguir:

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.

- Como se pode notar, o código-objeto relocável consta de uma **sequência de blocos de informação**, a partir dos quais é possível construir, ao menos em parte, um programa executável.
- **Cada** um desses **bloco**s deve conter **um** dentre os diversos **tipo**s **particular**es **de informação** originalmente existentes **no** código simbólico que constitui o **programa-fonte**.
- **Cada um** desses tipos de **informação** contribui para a composição do programa-objeto, e **deve ser tratado de forma específica** pelo montador.

Bloco de informações típico

- Um **bloco de informações típico** é muito **similar** ao que se utiliza na representação do **código-objeto absoluto**, e sua estrutura é esboçada a seguir:



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.

Estrutura de um bloco genérico de informação

- Um **bloco genérico de código-objeto relocável** é, portanto, **iniciado sempre** com a informação de seu **comprimento**, **seguida pela indicação do** particular **tipo do bloco** em questão.
- Portanto, nesta indicação informa-se, por meio de um **código numérico convencionado**, se se trata de um bloco de identificação de **pontos de acesso**, ou um bloco de **símbolos externos**, ou de **dados** (neste caso, sequências binárias codificadas, representando informação sobre o conteúdo da memória), **ou** de **finalização**.

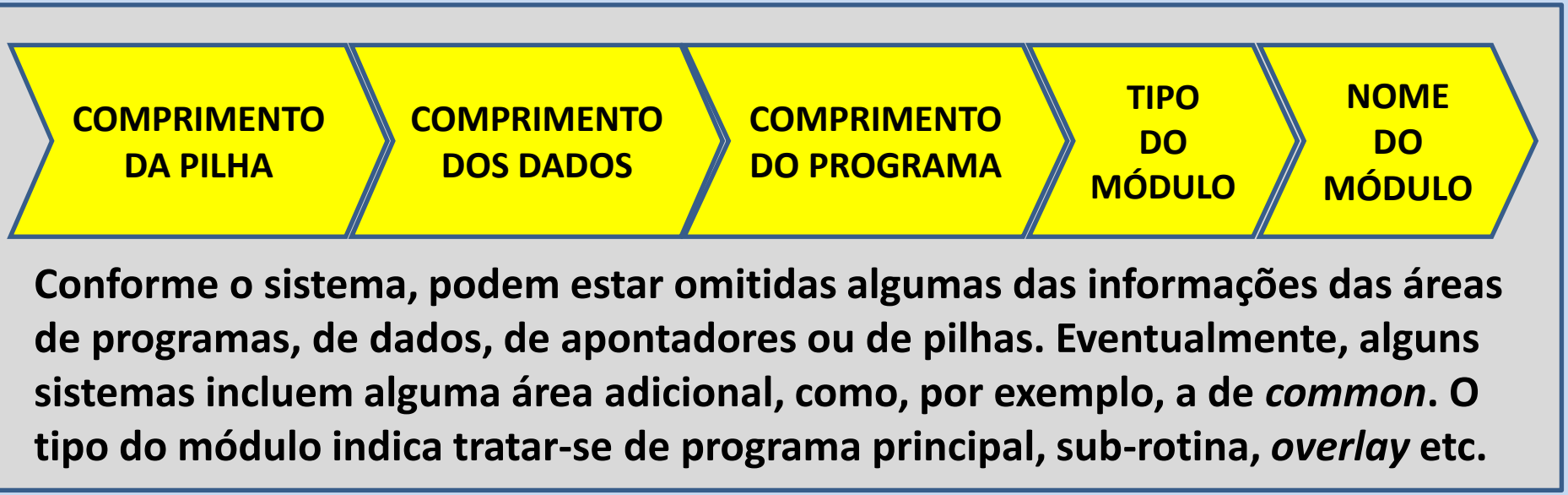
- **A seguir**, conforme será descrito adiante, aparecem as **informações, organizadas de acordo com o tipo de bloco** a que se referem.
- **Finaliza-se o bloco** com um **byte de redundância**, (**tipicamente, um byte de *checksum***) a exemplo do que era utilizado nos programas-objeto absolutos, para melhorar a confiabilidade da leitura do programa-objeto pelos programas de sistema.

Definição do módulo

- Através da **pseudo-instrução de definição do módulo**, um programa-objeto é introduzido por um **bloco de identificação**, que contém as informações fornecidas pelo programador ao montador através da **pseudo-instrução de definição do módulo**.

Bloco de identificação

- A figura abaixo detalha a estrutura típica de um **bloco de identificação** de um programa relocável.



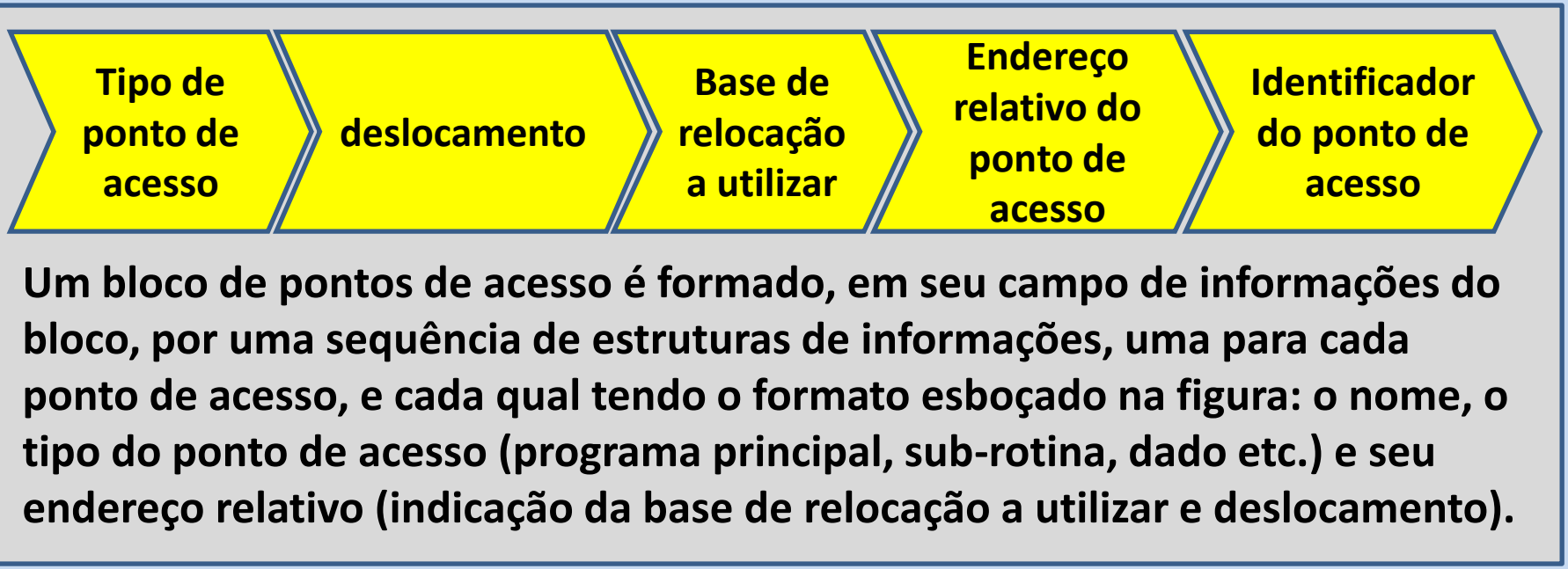
- Note-se que, nesta figura e nas seguintes, **só** estão representadas as **informações do particular bloco** em observação, e não os outros três campos, mostrados anteriormente.

- Observe-se a presença, entre os elementos componentes deste bloco, não apenas das informações fornecidas pela pseudo-instrução de definição do módulo, mas também das **informações de comprimento** das várias áreas de que se compõe o programa, obtidas como **resultado** do processamento **do primeiro passo** do montador.

- Isto faz com que, no caso de esquemas de montagem em **dois passos**, só ao final do processamento do primeiro passo do processo de montagem se disponha da informação completa para a geração deste bloco, e portanto de qualquer parte do código-objeto relocável.
- Para montadores organizados em lógica de um **único passo**, uma prática que pode ser utilizada é a de omitir tais informações do bloco de identificação, incorporando-as na ocasião oportuna a algum outro elemento do programa-objeto, como, por exemplo, o bloco de finalização.

Blocos de pontos de acesso

- A estrutura dos blocos de **pontos de acesso** está esquematizada na figura seguinte:



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).

Observações

- Para cada ponto de acesso, devem ser registradas as informações esquematizadas na figura, podendo um bloco de pontos de acesso incluir um ou mais desses pontos de acesso ao módulo.
- Alguns montadores geram **um bloco para cada ponto de acesso** existente, enquanto outros agrupam-nos, **ou** em um **bloco único** ou em conformidade com a sequência em que são encontradas as suas declarações nas pseudo-instruções em que são definidos, no texto-fonte.

- Analogamente ao caso do bloco de identificação, é óbvio que, no caso geral, informações sobre **endereços relativos** só ficam totalmente disponíveis **ao final** do processamento **do primeiro passo** do montador.
- Para montadores de **um único passo**, às vezes torna-se mais cômodo **gerar por último** todas estas informações, em vez de gerá-las no início do programa-objeto, como foi apresentado inicialmente.

Variante, para montagem em um passo

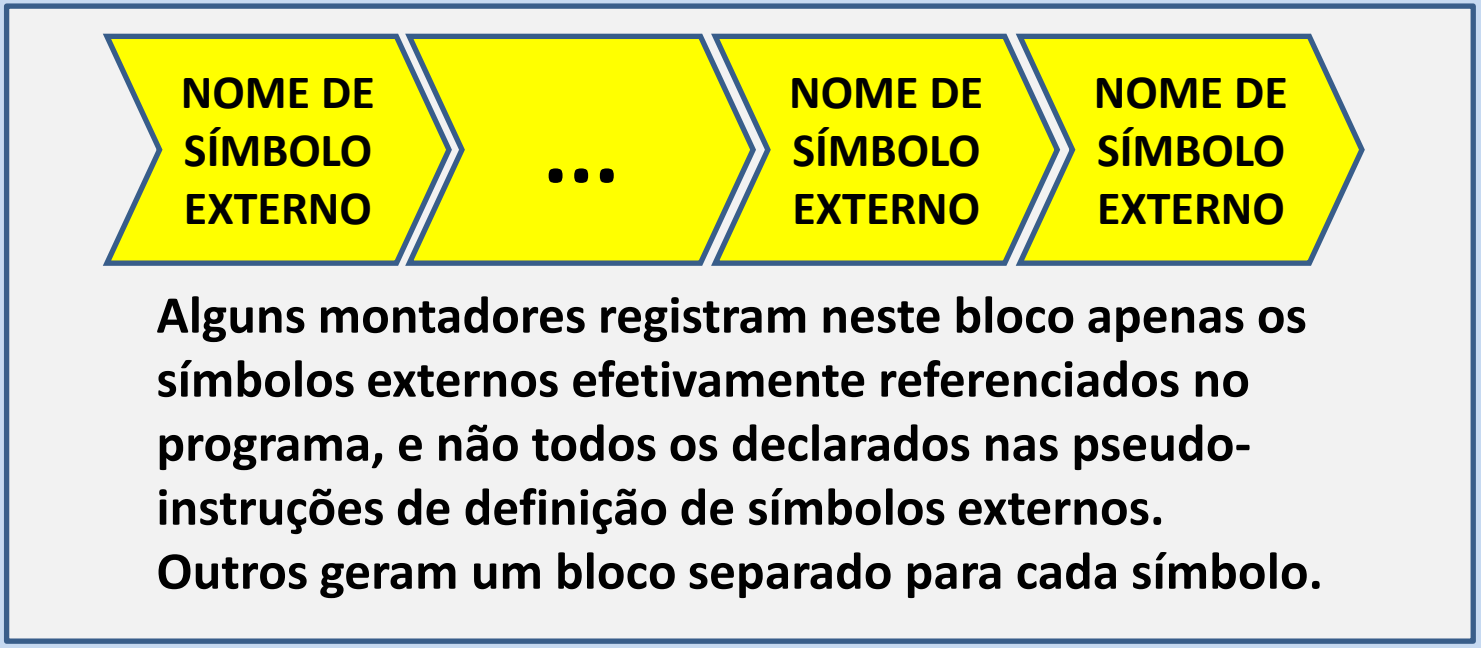
- Uma possível alternativa para o **formato da fita-objeto**, adequada para a geração por montadores de **um só passo**, é mostrada no esquema seguinte:

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

- Os blocos de **símbolos externos** coletam as informações declaradas nas correspondentes pseudo-instruções de definição de símbolos externos.
- Não se dispondo de nenhuma outra informação sobre símbolos externos a não ser apenas os seus **nomes**, os blocos de símbolos externos não contêm nada mais que um conjunto de tais identificadores, como mostrado a seguir.

Informação de um bloco de símbolos externos

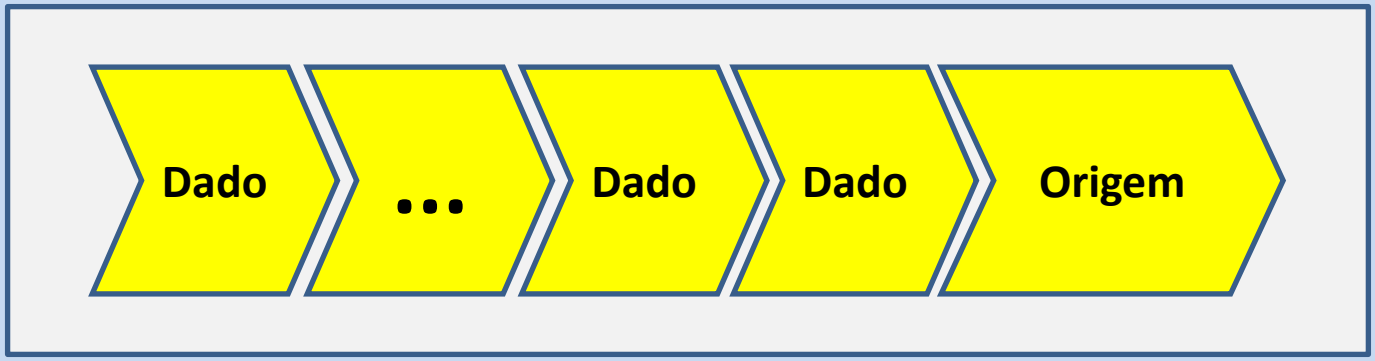
- A área de informação dos **blocos de símbolos externos** é bastante simples, conforme se pode observar no esquema seguinte:



- Estruturalmente, os **blocos de dados** são os mais complexos, pois devem conter as informações necessárias e suficientes para a identificação da função de cada ***byte*** do programa, para que se possa construir, a partir de tais informações, um programa absoluto executável.
- Funcionalmente, esses blocos muito se **assemelham**, como foi mencionado, aos blocos de **programa-objeto absoluto**.

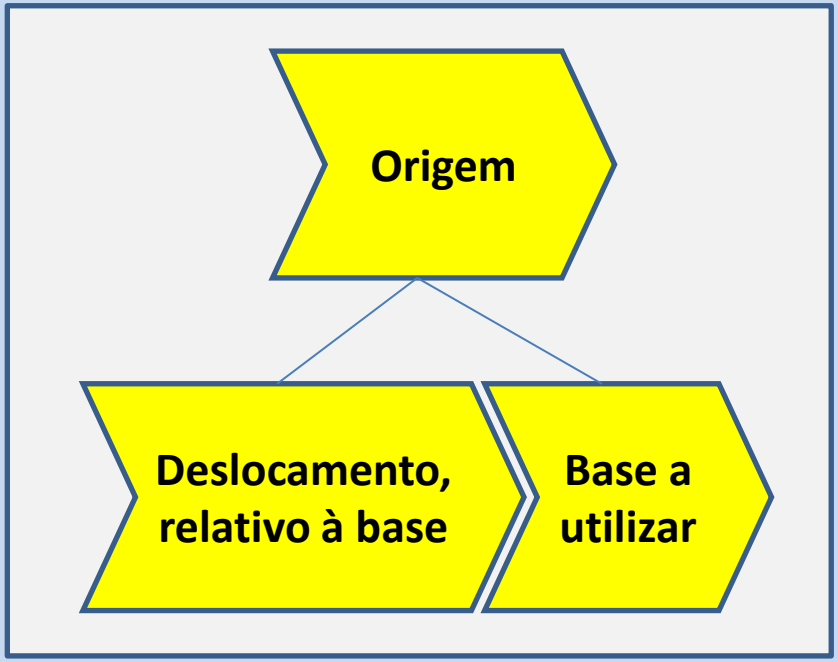
Informação de um bloco de dados

- Um bloco de dados contém os seguintes campos de informação (ver figura a seguir):
  - A origem , que indica a posição (relativa) de memória a partir da qual os dados são carregados.
  - Em seguida, consta uma sequência de descritores dos dados, os quais deverão ser sequencialmente depositados na memória a partir do endereço expresso como origem.



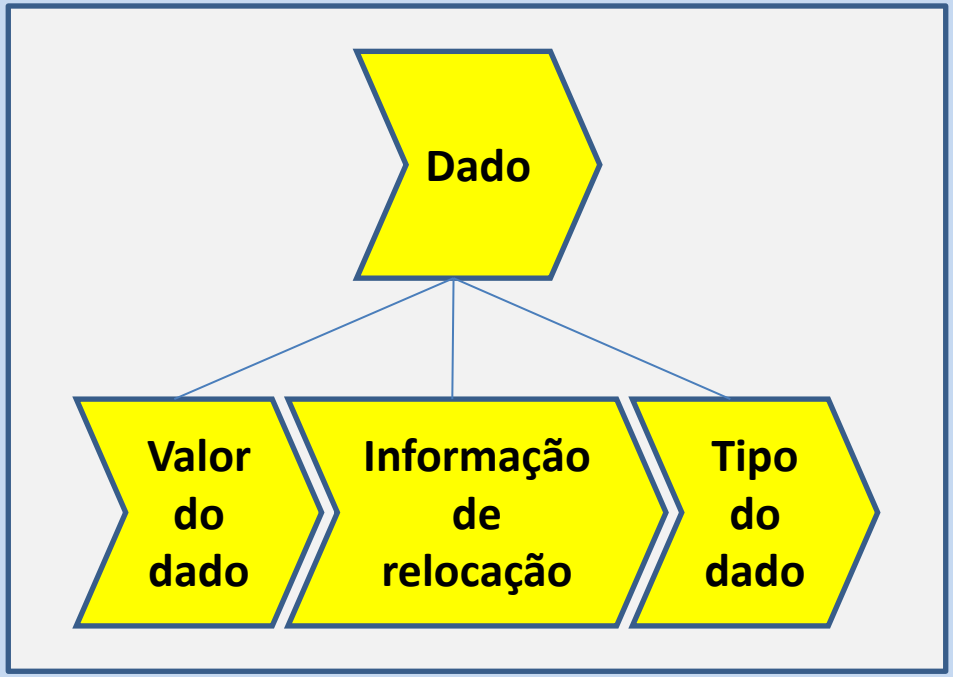
Detalhamento das informações de origem

- A informação de origem é composta de:
  - uma indicação de base associada ao endereço relativo (que pode inclusive indicar endereços absolutos)
  - um deslocamento em relação a essa base.



Detalhamento das informações de dado

- **Cada dado apresenta:**
  - **uma indicação de tipo**
    (constante, endereço relativo, endereço externo simbólico etc.)
  - **uma informação de relocação**
    (se necessário, indicando a base a ser utilizada)
  - **um valor**
    (a ser relocado, se for o caso).



- Nesses blocos, os **dados** podem apresentar-se como sendo **de diferentes tipos**, e cada um deve ser descrito de acordo com o correspondente tipo.
- Havendo **sequências** de dados, todos **no mesmo formato**, ou **na mesma base** de relocação, costuma ser conveniente **destacar a informação** comum, para conseguir um **código-objeto mais compacto**.
- Isso pode proporcionar **economia de código-objeto**, e tende a **simplificar o processamento** posterior, a ser executado **pelo ligador, alocador e relocador**.

- Assim, o montador pode gerar os blocos de dados de acordo com os tipos dos dados a ele apresentados no programa-fonte.
- Isto costuma ser feito pela **eliminação de informação desnecessária** e pelo **acréscimo de complementos** que sejam essenciais à análise do programa-objeto por parte dos demais programas do sistema encarregados de manipulá-los.

- O último tipo de bloco que ainda resta para ser analisado é o **bloco de finalização do programa**, que, em geral, não inclui nenhuma informação, exceto nos casos de programas principais ou de ***overlays***, em que esse bloco poderá exibir um campo contendo a informação do endereço a partir do qual o módulo deve iniciar sua execução.

Estrutura de um bloco de finalização

- **Blocos de finalização** apresentam, em seu campo de informações, a estrutura esboçada na figura para casos de **módulos executáveis** (**programas principais** e ***overlays***).
- Para **sub-rotinas** e outros módulos não diretamente executáveis, este campo de informações é **omitido** ou deixado **em branco**.

Objetivo do montador relocável

- Cabe aos montadores de linguagens simbólicas relocáveis **gerar**, a partir de textos-fonte simbólicos, **programas-objeto** que apresentem a estrutura anteriormente descrita.
- Adicionalmente, as **listagens** por eles geradas deverão apresentar para o usuário, em formato legível (hexadecimal, octal etc.), todas as **informações contidas no código-objeto**.

Listagens

- Nessas listagens deverão constar as **informações de relocação de cada endereço**, relacionando a lista de pontos de acesso e seus respectivos endereços relativos.
- Adicionalmente, deverão incluir a indicação da **base de relocação** a ser utilizada em cada caso, relacionar os **endereços simbólicos externos** referenciados no programa, e dar indicações sobre os **requisitos de memória** exigidos pelo módulo para cada uma das partes que o compõem.