# PCS 3216
# Sistemas de Programação

João José Neto

Aula 10 – Montadores de um passo e
Alguns aspectos de sua implementação

# MONTADORES DE UM SÓ PASSO

# Montadores de dois passos

- Os montadores de dois passos, como foi mencionado antes, dividem as tarefas de montagem dos programas simbólicos entre os dois passos em que são estruturados
  - Especializando cada um deles
    - Primeiro passo – coleta de informações sobre os símbolos
    - Segundo passo – geração de código objeto
  - Isso propicia a obtenção de programas potentes e eficientes.

# Algumas Inconveniências

- Todavia, muitas vezes, deseja-se construir montadores para máquinas ou situações em que a dupla leitura física do texto simbólico é indesejável. Por exemplo:
  - Quando os periféricos de leitura/gravação disponíveis são muito lentos
  - Quando o montador é intensamente utilizado, ou seja, a operação de montagem de programas simbólicos é uma atividade muito intensa no sistema
  - Quando o sistema desempenha muitas atividades e está sobrecarregado.
  - Quando se deseja que a operação de montagem seja muito rápida, por exemplo, em uma escola, para atendimento muito frequente de atividades discentes.

# Dois passos

- Em montadores de dois passos, esse problema costuma ser atenuado:
  - Memorizando uma cópia do texto simbólico (em disco, por exemplo) durante a sua leitura, no primeiro passo;
  - Posteriormente, para completar a operação de montagem, tal imagem do programa-fonte construída no primeiro passo é (re)lida, desta vez a partir do disco, pelo segundo passo do montador (só o fato de essa releitura ser feita a partir de um arquivo em disco e não de periféricos lentos já colabora significativamente para uma redução substancial do tempo gasto pelo montador na atividade de releitura do programa fonte).

# Uso em máquinas sem memória de massa

- Há casos em que se usa, alternativamente, a opção de fazer a operação de montagem em um único passo:
  - Por imposição de decisões de projeto
  - Quando não há disponibilidade de um dispositivo de memória de massa (disco, por ex.) para esta tarefa
  - Quando não se dispõe de periféricos de entrada/saída rápida para a leitura/gravação do programa fonte ou do programa objeto.
  - Quando limitações ou exigências da aplicação inviabilizam o uso de outras soluções

# Montadores de um só passo

- Para isto, são construídos os chamados *montadores de passo único*, que leem o programa fonte uma só vez.
- Há dois tipos principais de montadores de um só passo:
  - Montadores *load and go* (ou *assemble and go*), em que o código-objeto é gerado diretamente na memória, ficando disponível para ser executado imediatamente
  - Montadores *que geram código-objeto carregável*, a ser posteriormente introduzido na memória para execução. Nesta modalidade, o código-objeto assume a forma de um script numérico de preenchimento da memória, que determina como depositar na memória o código-objeto. Essa ordem reflete a exata sequência em que as referências à frente, na memória, foram sendo conhecidas pelo montador, durante o processo de montagem do programa-fonte.

# Funcionalidade idêntica

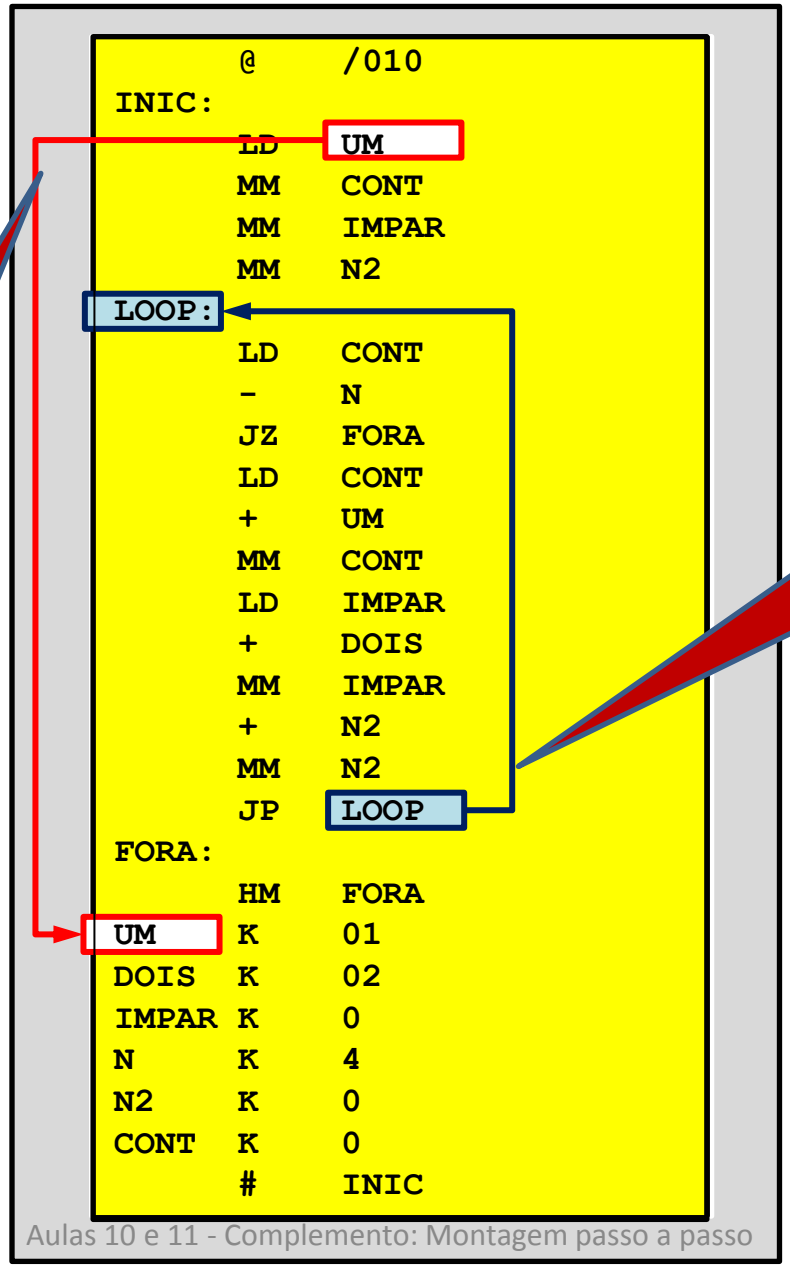
- Os montadores de um único passo devem executar funcionalmente todas as tarefas que foram descritas para os montadores de dois passos.
- Deverão fazê-lo, entretanto, sem exigir para isso mais de uma única leitura física do texto simbólico, por parte do montador.

# Problema principal: Referências à frente

- Assim, será necessário resolver novamente alguns problemas que já foram solucionados pelo esquema em dois passos, os quais, porém, voltam à tona pela imposição de não seja feita mais que uma única operação de leitura física do programa-fonte simbólico.
- O problema mais sério a ser resolvido surge na ocasião da montagem de instruções de referência à memória, e decorre da falta de informação completa para a obtenção do código de máquina para essas instruções quando elas se apresentam com operandos simbólicos correspondentes a endereços ainda não definidos (referências simbólicas à frente).

# Exemplos de referências à memória

Neste ponto o símbolo UM está indefinido, desconhecendo-se seu endereço, portanto.




Uma Referência para a frente

Uma Referência para trás

Já aqui o símbolo LOOP está definido, pois apareceu acima como rótulo, portanto desde aquele ponto seu endereço já estava determinado.

# Uma Solução Prática

- Em montadores de um só passo, soluciona-se esta dificuldade através da memorização dessas instruções incompletas em uma tabela de códigos pendentes.
- Os elementos desta tabela referem-se ao conjunto de todas as instruções que apresentem operandos com endereços ainda não resolvidos, e devem registrar todas as informações conhecidas acerca dos respectivos operandos
- A finalidade desses registros é a de propiciar o preenchimento dessas informações assim que se tornarem disponíveis, durante a montagem do programa, ou seja, na ocasião da definição do símbolo do qual dependem.
- Assim, o código binário completo pode ser finalmente montado e gerado no programa-objeto, ou então, diretamente depositado na posição correta de memória.

# O código-objeto a ser gerado

- Por questões práticas, o **código-objeto** a ser gerado pelo montador de um único passo deve ter **formato compatível** com os códigos-objeto adotados para o montador de dois passos, assim como para os demais programas do sistema de programação (loader, dumper, etc).
- Assim, um programa-objeto deverá constar de uma sequência de **um ou mais blocos** (binários ou hexadecimais), cada qual contendo os **códigos numéricos** correspondentes às instruções e dados do programa, acompanhados de **metadados** informando o **endereço** de memória associado a esses códigos, o **número de bytes** do bloco, e um byte de redundância contendo um "**checksum**", que costuma ser o byte menos significativo da somatória de todos os bytes que compõem o bloco.
- O formato de cada bloco é o seguinte (= ao usado no caso de 2 passos):
  - Uma **separação física** (normalmente sequência de zeros binários)
  - Número N de **bytes do bloco** (maior que zero, obrigatoriamente)
  - **Endereço inicial** associado ao primeiro byte de código (n bytes para memórias com endereços de $2^n$ bits) No caso, 2 bytes (a memória tem 4096 bytes)
  - N bytes corespondentes ao **código numérico** contido no bloco
  - 1 byte de **checksum**, contendo o complemento da soma de todos os demais, de modo que seja nulo o último byte da soma de todos os bytes do bloco, incluindo o de checksum.

# Geração do código-objeto

- Do ponto de vista de geração do código-objeto, é possível ao montador de um único passo **gerar códigos incompletos**, para as instruções com operandos não resolvidos, utilizando apenas as informações disponíveis na ocasião da leitura da instrução simbólica.
- A utilidade da geração desse código incompleto nessa ocasião é a de **reservar área de memória** para o código definitivo, cujo endereço e formato são conhecidos, mas cujo valor exato ainda não foi determinado.
- **Na ocasião da definição** do símbolo do qual dependem, as instruções cujos códigos ainda estejam inacabadas **podem ser completadas** com as informações faltantes, obtendo-se dessa forma o código final correspondente à instrução.

# Lista de Pendências

- A **Lista de Pendências** costuma ser organizada como uma lista ligada, cujos elementos, além do **ponteiro para o próximo elemento** da lista, contêm no seu campo de informação triplas tais como (**EI, OPI, DI**), nas quais referenciam, respectivamente:
  - EI - **endereço de memória** associado à instrução correspondente
  - OPI - **código de operação** referente à instrução
  - DI - **deslocamento** (translação) a ser sofrido pelo endereço de definição do símbolo referenciado para formar corretamente o operando da instrução.

# Estruturas de dados

- Possível implementação da lista de pendências
  - **Listas ligadas** implementam muito confortavelmente as listas de pendências, que representam o conjunto dos **códigos incompletos** presentes no programa-objeto em construção.
  - **A cada símbolo**, associa-se uma **lista de referências à frente ainda não resolvidas**, cada qual acompanhada de eventuais informações complementares sobre o operando associado.
  - Os elementos da lista de pendências são a ela **incorporados** toda vez que for encontrada uma **referência a um símbolo indefinido**, e são dela **removidos** sempre que o montador **determinar o endereço** a que se refira um desses símbolos.
  - Em algumas implementações, as listas de pendências são fisicamente armazenadas na mesma estrutura de dados da tabela de símbolos, funcionando assim como sua extensão.

# Inclusão de pendências

- Na ocasião da inclusão de mais um elemento na lista de referências, pode-se verificar se o símbolo referenciado já está definido ou não.
- Se o símbolo não for encontrado, ou então, mesmo se for encontrado mas estiver indefinido, cria-se e inclui-se na lista de pendências associada a esse símbolo uma tripla da forma (EI, OPI, DI), conforme foi referido anteriormente. Nesse caso, nenhum código pode ser gerado nesta ocasião.
- Caso o símbolo já tenha sido encontrado, e se ele estiver marcado como já definido, então o respectivo endereço numérico já será conhecido, portanto nada de inédito haverá para ser memorizado, podendo portanto ser gerado diretamente o código definitivo para essa instrução, sem alterar a lista de pendências.

# Resolução de pendências

- Posteriormente, na ocasião da definição do símbolo, a lista de pendências a ele associado deverá ter cada um dos seus elementos processado e eliminado, por meio da geração do código definitivo correspondente.
- Para isso constrói-se, para cada um dos elementos da lista, um bloco de código-objeto contendo as informações registradas na lista de pendências, complementadas com a informação do endereço de memória associado ao símbolo recém-definido.
- Após a geração de tal código, os elementos da lista de pendências referentes ao símbolo em questão podem ser descartados.

# Geração descontínua de código

- Do ponto de vista da geração de código, se esta for efetuada diretamente na memória, não haverá qualquer necessidade de processamento adicional.
- Se, entretanto, a geração for feita em meio externo, como por exemplo em fita ou arquivo, certamente haverá uma descontinuidade na sequência de bytes gerados, em termos dos endereços de memória ocupados.
- Como consequência direta, o programa objeto resultante da montagem em passo único, ao contrário do que geralmente ocorre na montagem em dois passos, será composto de um número maior de blocos de código, e o comprimento desses blocos em geral será menor.

# Esvaziamento do bloco de código

- Durante a geração de código, uma área de memória reservada a essa geração vai sendo preenchida até que o seu **comprimento máximo** seja atingido, ou até que se **altere o endereço de origem** do código.
- Neste último caso, deverá ser forçado o **esvaziamento** do bloco incompleto de código, em construção, e o **início do preenchimento** de um novo bloco.
- Isso deve ser feito para liberar espaço no bloco de código, cedendo área para a geração das informações de preenchimento de lacunas anteriormente criadas pelo montador, associadas a referências à frente representadas pelas pendências registradas.
- Obviamente, é possível usar uma lógica mais complexa para evitar a geração de alguns desses blocos adicionais, mas isso não será considerado no presente estudo.

# *Backtracking*

- Esta técnica pode ser classificada na classe dos algoritmos de ***backtracking***, em que o trabalho não é efetuado linearmente, mas sofre **descontinuidades** para correções de códigos incompletos analisados anteriormente, e que estão portanto fora de ordem.
- O conteúdo do programa-objeto assim construído **reflete a ordem** na qual a memória é preenchida pelo montador à medida que este vai determinando os endereços associados aos diversos símbolos.
- Preserva, portanto, a ordem de preenchimento da memória, na sequência exata em que as informações forem sendo coletadas ou construídas **ao longo do trabalho de montagem**.

# Ações de Retro-Preenchimento

- A execução das ações de "***backtracking***" costuma, portanto, envolver as seguintes ações:
  - **Salvamento do endereço** corrente de geração do código.
  - **Retroendereçamento** à posição ocupada pela instrução pendente que está sendo resolvida.
  - **Geração de bloco** de substituição (**preenchimento** ou **correção**) do código incompleto pela versão completa, agora conhecida, da instrução pendente.
  - **Abertura de novo bloco** de geração de código, restaurando a origem ao endereço inicialmente salvo, se for o caso (desnecessário se não houver alteração na origem do código gerado).

# Interpretação

- Assim, o **código gerado** pelo montador de um passo não representa obrigatoriamente uma imagem do conteúdo da memória.
- Trata-se, na realidade, de um código que funciona como um "***script***" de preenchimento da memória.
- Naturalmente, esse programa de preenchimento é interpretado pelo ***loader*** (carregador) absoluto, o qual se encarrega de executar as ações apropriadas.
- Ao final dessa interpretação, os **bytes do código-objeto** representado nesse programa estará devidamente **carregado nos endereços corretos** da memória, pronto para a execução, se for o caso.

# Listagens

- Um outro problema advindo da supressão de um passo na lógica do montador, manifesta-se nas **operações de listagem** do programa simbólico, ao lado do correspondente programa numérico.
- Optando-se por **efetuar a listagem à medida que se monta o programa**, surge um problema: havendo falta de informação, os códigos numéricos correspondentes às pendências não poderão ser listados na ocasião.
- Para salvar a listagem, pode-se **ao final da montagem imprimir a** tabela de endereços que tenham estado presentes na **lista de pendências** durante a montagem.
- Isso completa a informação impressa, mas o uso de uma listagem desse tipo continua **desconfortável**, pois registra mais o histórico da montagem do que o seu resultado final.

# Listagem de códigos pendentes

- Muitos montadores de um passo imprimem os códigos gerados exatamente na mesma ordem em que os mesmos foram sendo gerados no programa objeto.
- Em outras palavras, os códigos pendentes são impressos logo após os símbolos dos quais dependem terem sido definidos (ocorrerem como rótulos).
- Outros montadores imprimem códigos incompletos, indicando que estão pendentes naquele ponto.
- Ao final da montagem, imprimem o conteúdo da tabela de símbolos, com cujo auxílio o usuário pode compor manualmente os endereços omitidos.

# Listagens melhores

- As duas opções mencionadas fornecem listagens que se mostram incômodas para o usuário.
- Alguns montadores mais sofisticados, de um passo, geram em disco arquivos contendo uma imagem da listagem do programa e do código associado.
- Nesses arquivos, tais montadores alteram o conteúdo da imagem da listagem toda vez que for efetuada alguma resolução das referências à frente.
- Com esta última solução, é possível obter, com montadores de um passo, listagens idênticas às produzidas pelos de dois passos.

# Desvantagem

- Como desvantagem principal do uso de montadores em um só passo, em geral é necessário ter à disposição uma considerável área de memória de massa.

- Incorporando as soluções apresentadas às ideias já utilizadas na lógica do montador de dois passos, é possível obter um montador capaz de efetuar em um único passo a montagem dos seus programas-fonte.

# Conclusão

- Há naturalmente vantagens e desvantagens de se utilizar, na confecção de um montador, uma lógica de um ou de dois passos.
- Normalmente, deve-se levar em conta a finalidade a que o montador se destina, a frequência com que será utilizado, o tempo de retorno desejado, o tamanho médio dos programas que serão traduzidos, e o tamanho máximo dos mesmos, que irá servir para definir a dimensão de suas tabelas e áreas de dados, e mesmo, neste caso particular, o tipo mais adequado de montador.