Trabalho de Programação de Sistemas
Prof André Flores dos Santos
Pedro Henrique de Brito Canabarro

1) Explique o que faz um montador e onde ele fica no fluxo de construção (fonte → objeto → executável).

O montador, em um contexto de construção do software, ele é um programa que traduz o código em linguagem de "assembly" para código de máquina, ele faz a conversão para sequências de bits que o computador consegue executar diretamente. Ele fica no fluxo após a compilação e o processo de criação do código objeto, sendo o responsável por gerar o código executável a partir das instruções simbólicas, permitindo que o programa final possa ser carregado e executado pela máquina.

2) Compare montadores de 1-pass e 2-pass (como resolvem rótulos, prós e contras, quando usar cada um).

Os montadores de uma passagem processam o código em uma única varredura, armazenando os rótulos indefinidos na tabela de símbolos para resolução posterior, mas são limitados em otimização e não conseguem resolver referências cruzadas.

Quando usar um montador de uma passagem? Quando estiver tratando de um projeto mais simples, onde o código-fonte é pequeno, o número de referências cruzadas é baixo, ou a velocidade de montagem é a prioridade.

Os montadores de duas passagens usam duas varreduras: a primeira coleta informações e rótulos, e a segunda resolve dependências e gera o código final, permitindo otimizações significativas. A escolha depende do ambiente, a uma passagem é mais rápida e simples, enquanto as duas passagens são mais robustas para códigos complexos com referências cruzadas.

Quando usar um montador de duas passagens? Quando estiver tratando de um projeto mais complexo e um projeto grande, que exigem a resolução de dependências entre diferentes partes do código, e onde a otimização do código é um requisito importante.

3) Defina a tabela de símbolos (SYMTAB) e porque ela é necessária.

A tabela de símbolos é uma estrutura de dados usada pelos compiladores para armazenar informações sobre os identificadores (nomes de variáveis, funções, etc.) usados em um programa. Ela permite que o compilador associe cada identificador a seus atributos, como tipo, escopo e endereço na memória, facilitando a verificação de erros, a geração de código e a otimização do programa.

A tabela de símbolos é essencial para o processo de compilação por várias razões

- Verificação de tipos: a tabela de símbolos permite que o compilador verifique se os identificadores estão sendo usados corretamente, de acordo com suas declarações e o contexto do programa. Isso ajuda a detectar erros como o uso de variáveis não declaradas ou a atribuição de valores de tipos incompatíveis.
- Escopo: A tabela de símbolos ajuda a gerenciar o escopo dos identificadores, ou seja, a área do programa onde um identificador é válido. Isso evita conflitos de nomes e garante que cada identificador seja usado no contexto correto.
- Otimização: A tabela de símbolos também pode ser usada para otimizar o código gerado. Por exemplo, ela pode ajudar a identificar variáveis que são usadas com frequência e podem ser armazenadas em registradores para acesso mais rápido.
- Depuração: As informações armazenadas na tabela de símbolos são úteis durante a fase de depuração, permitindo que o programador veja o valor de variáveis e outros identificadores em pontos específicos do programa.

PARTE B DO TRABALHO

```
; exemplo.asm — mini-ISA (MOV/ADD/JMP/HLT)
 Cada MOV/ADD/JMP ocupa 2 bytes; HLT ocupa 1 byte.

        MOV R0, #5  ; carrega 5 em R0
        ADD R0, #3   ; R0 = 5 + 3 = 8
        JMP END      ; salto para rótulo ainda não definido (referência futura)

 LOOP:  ADD R0, #1  ; incrementa R0 (exemplo de referência para trás)
        JMP LOOP    ; volta para LOOP

END:    HLT          ; fim
```

1) Endereçamento por linha, começando em 0 (quanto cada instrução ocupa, bytes?).

MOV R0, #5 (ocupa 2 bytes: 0 e 1)
ADD R0, #3 (ocupa 2 bytes: 2 e 3)
JMP END (ocupa 2 bytes: 4 e 5)
LOOP:
ADD R0, #1 (ocupa 2 bytes: 6 e 7)
JMP LOOP (ocupa 2 bytes: 8 e 9)
HLT (ocupa 1 byte: 10)

2 + 2 + 2 + 2 + 2 + 1 = 11 bytes no total.

2) SYMTAB com os endereços de LOOP e END.

A tabela de símbolos armazena os rótulos e os endereços de memória onde eles são definidos. O LOOP está no endereço 6 e o END está no endereço 10.

3) Explicar, em 4–6 linhas, como o 2-pass e o 1-pass resolvem o JMP END.

(2-pass) No primeiro passo, o montador lê todo o código apenas para construir a tabela de símbolos, mapeando LOOP para o endereço 6 e o END para o endereço 10. No segundo passo, ele lê o código novamente para gerar o código de máquina. Ao encontrar JMP END, ele consulta a SYMTAB já completa, obtém o endereço 10 para END e gera a instrução de salto corretamente.

(1-pass) Ao encontrar JMP END, como o endereço de END é desconhecido, o montador gera a instrução com o campo de endereço em branco e adiciona END a uma lista de referências pendentes para aquele local. Quando o montador finalmente encontra a definição do rótulo END no endereço 10, ele utiliza uma técnica chamada *backpatching* (retro-correção) para voltar à instrução JMP e preencher o endereço que faltava.