

Introdução à Software Básico: Montadores - parte 1

Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade de Brasília

Montadores

- 1 Introdução
- 2 Definição de uma máquina hipotética
- 3 Funções de um Montador
- 4 O processo de Montagem
- 5 O algoritmo de duas passagens

Montador

- Nos primórdios da computação, os primeiros programas foram escritos em Linguagem de Máquina, isto é, as instruções eram armazenadas diretamente na memória, em formato binário
- Esse trabalho de programação era feito através de botões ou ligações por cabos.
- Já na segunda geração das linguagens de programação surgem os primeiros tradutores. Que traduzem uma linguagem formada por símbolos mnemônicos (linguagem de Montagem) e a linguagem binária da máquina:
 - os chamados Montadores (do inglês, “Assembler”).

Montador

- Como vimos anteriormente, o processo de “montagem” recebe como entrada um arquivo texto contendo o código fonte do programa em linguagem de montagem e gera como saída um arquivo binário contendo o código de máquina e outras informações relevantes para a execução do código gerado.
- Para entendermos o processo de montagem, vamos precisar de algumas “ferramentas”, as quais veremos a seguir.

Descrição da máquina hipotética

- Para nos auxiliar no estudo de montadores (e também ligadores e carregadores), nós vamos definir uma máquina hipotética e um conjunto (reduzidíssimo) de instruções para ela.
- Embora o nosso conjunto de instruções seja bem reduzido, veremos que programas completos podem ser criados
- O objetivo aqui é apresentar a linguagem assembly de maneira informal, complementado posteriormente.

Descrição da máquina hipotética

- A nossa máquina hipotética terá:
 - 1 registrador acumulador de 16 bits. Chamaremos de **ACC**.
 - 1 registrador apontador de instruções de 16 bits. Chamaremos de **Program counter** ou **PC**.
 - Memória de 216 palavras de 16 bits;
 - A nossa máquina hipotética pode reconhecer 3 formatos de instruções:
 - Formato 1: Opcode
 - Formato 2: Opcode Endereço
 - Formato 3: Opcode Endereço_1 Endereço_2

Código de Operação ou OPCODE

- Identifica a operação a ser realizada pelo processador.
- É o campo da instrução cujo **valor binário** identifica a operação a ser realizada (é o código binário da instrução) .
- Este código é a entrada no **decodificador de instruções** na unidade de controle.
- Cada instrução deverá ter um código único que a identifique.

Uma máquina hipotética

mnemônico		código da máquina		espaço que a inst ocupará em mem
Opcode Simbólico	Opcode Numérico	Tamanho (palavras)	Ação	
ADD	01	2	$ACC \leftarrow ACC + mem(OP)$	
SUB	02	2	$ACC \leftarrow ACC - mem(OP)$	
MUL	03	2	$ACC \leftarrow ACC \times mem(OP)$	
DIV	04	2	$ACC \leftarrow ACC \div mem(OP)$	
JMP	05	2	$PC \leftarrow OP$	
JMPN	06	2	Se $ACC < 0$ então $PC \leftarrow OP$	
JMPN	07	2	Se $ACC > 0$ então $PC \leftarrow OP$	
JMPZ	08	2	Se $ACC = 0$ então $PC \leftarrow OP$	
COPY	09	3	$mem(OP2) \leftarrow mem(OP1)$	
LOAD	10	2	$ACC \leftarrow mem(OP)$	
STORE	11	2	$mem(OP) \leftarrow ACC$	
INPUT	12	2	$mem(OP) \leftarrow entrada$	
OUTPUT	13	2	$saída \leftarrow mem(OP)$	
STOP	14	1	Suspende a execução	

Figura: Conjunto de instruções da máquina hipotética

Conjunto de instruções

- O conjunto de instruções de alguns processadores, como por exemplo o Intel 8080, não possui instruções para multiplicação ou divisão
- Portanto, um programa em linguagem de máquina, nestes processadores, não pode fazer multiplicação ou divisão diretamente (somente através de combinação de outras instruções).
- Já um programa em linguagem de nível mais alto pode implementar comandos de multiplicação (ou divisão) que combinem uma série de instruções binárias do conjunto de instruções para fazer uma multiplicação (ou divisão) através de repetidas somas (subtrações) ou por deslocamento de bits (Ex. C).

Formato de uma linha fonte:

- O nosso programa fonte, em pseudo-assembler, utiliza o seguinte formato:
<rótulo>: <operação> <operandos> ;<comentários>

Exemplo

Algoritmo 1 Exemplo 1

```
1: ROT: INPUT N1
2:      COPY N2,N1 ;isso é um comentário
3: N1: SPACE
4: N2: SPACE
```

Exemplo

O programa a seguir utiliza o conjunto de instruções definidos acima para ler dois números da entrada padrão (teclado) e imprimir a soma dos mesmos na saída padrão (vídeo):

- $N3 = N1 + N2$;

Algoritmo 2 Exemplo 2

INPUT	N1	; Lê o primeiro número
INPUT	N2	; Lê o segundo número
LOAD	N1	; Carrega N1 no acumulador
ADD	N2	; Soma está no acumulador
STORE	N3	; Coloca soma N1+N2 em N3
OUTPUT	N3	; Escreve o resultado
STOP		; Termina a execução do programa
N1:	SPACE	; Reserva espaço para o primeiro número
N2:	SPACE	; Reserva espaço para o segundo número
N3:	SPACE	; Reserva espaço para o resultado

Conjunto de instruções

- São instruções para o próprio montador, isto é, uma instrução da linguagem assembler para que o montador modifique o seu comportamento ou realize alguma tarefa.
- Exemplo:
 - diretivas podem ser usadas para alocar espaço para dados ou variáveis;
 - instruir o montador a incluir arquivos fontes;
 - estabelecer o ponto de início do programa, etc.
- Inicialmente vamos utilizar duas diretivas que usaremos nos exemplos e nos nossos programas:
 - **CONST** → Instrui o montador a gerar dados em memória (leitura apenas)
 - **SPACE** → Reserva espaço para os dados (leitura e gravação)

Exemplo 3

- Crie um programa, utilizando o conjunto de instruções definidos da nossa máquina hipotética que compute a área de um triângulo.
- O programa deverá ler os dados com INPUT e mostrar o resultado (Sabemos que a área é dada por: $(Base \times Altura)/2$)

Solução

Algoritmo 3 Exemplo 3

INPUT	B		; Lê a base
INPUT	H		; Lê a altura
LOAD	B		; Carrega B no acumulador
MULT	H		; ACC tem BxH
DIV	DOIS		; ACC tem o resultado de $(B+H)/2$
STORE	R		; Coloca soma $(B+H)/2$ em R
OUTPUT	R		; Escreve o resultado
STOP			; Termina a execução do programa
B:	SPACE		; Reserva espaço para a base
H:	SPACE		; Reserva espaço para a altura
R:	SPACE		; Reserva espaço para o resultado
DOIS:	CONST	2	; Reserva espaço para uma constante

- A linguagem assembler que estamos utilizando é também chamada de linguagem simbólica pura, pois cada declaração gera exatamente uma instrução de máquina.

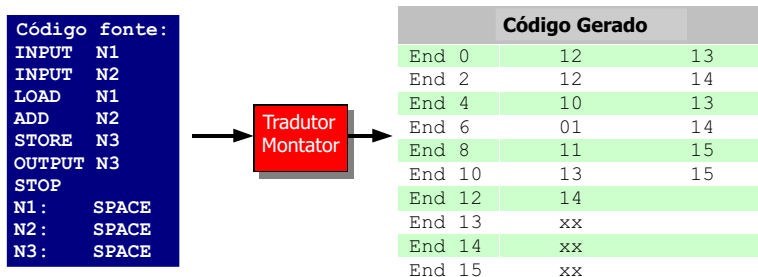


Figura: Exemplo de Montagem

Nota

Valores numéricos aparecem em decimal e o valor “xx” representa um valor qualquer ocupando uma palavra de memória (16 bits).

Tarefas básicas do montador

- Tradução de pseudo-instruções por *opcodes* numéricos e expansão de “macros”
- Representação do programa em linguagem de máquina: instruções e dados
- Reservar espaço para os dados
- Determinação dos endereços de memória referenciados pelo programa
- Registro das informações necessárias à **ligação** do programa:
 - Tabela de Definições - símbolos externos usados no módulo
 - Tabela de Uso – Símbolos exportados e seus atributos

O processo de Montagem (Assembler)

- Como vimos, um programa em linguagem *assembly* puro consiste de uma série de declarações de uma linha
- Desta forma, seria natural imaginar um montador que lesse o código fonte (*assembly*), linha a linha, e gerasse o código de máquina em um outro arquivo
- Esse processo poderia ser repetido até que todas as declarações do arquivo fonte fossem traduzidas em linguagem de máquina

- Infelizmente, o processo não é tão simples.
- Para ver isso, imagine uma situação na qual temos um desvio (*jump*) no programa para um rótulo R dentro do programa.
- O montador não poderá gerar o código de máquina para essa instrução até que o endereço do rótulo R seja conhecido.
- O rótulo R pode estar definido no final do programa, o que leva o montador a ler o programa fonte inteiro para poder encontrar o rótulo (e então gerar o endereço)

- Outro exemplo, e no exercício anterior onde os identificadores para as variáveis (os rótulos de N1,N2 e N3) foram declarados no final

Código fonte:

```
INPUT  N1
INPUT  N2
LOAD   N1
ADD    N2
STORE  N3
OUTPUT N3
STOP
N1:    SPACE
N2:    SPACE
N3:    SPACE
```

Código Gerado

End 0	12	13
End 2	12	14
End 4	10	13
End 6	01	14
End 8	11	15
End 10	13	15
End 12	14	
End 13	xx	
End 14	xx	
End 15	xx	

- O problema que vimos no slide anterior é conhecido como *forward reference problem* – ou problema de referência posterior
- Como resolver a questão de “referências posteriores”?
- Existem duas formas:
 - 1 Ler o código uma única vez utilizando mecanismos para resolver o problema de referências posteriores
 - Torna o montador mais complexo
 - 2 Ler o código duas vezes
 - A primeira leitura identifica os símbolos
 - A segunda, gera o código de máquina
 - Aumenta o Tempo de leitura. Precisamos ler todo o programa fonte duas vezes

O processo de Montagem (Assembler)

- Por sua simplicidade, a maioria dos assembler utiliza o processo de ler o código fonte duas vezes
- Cada leitura do código fonte é chamada de **passagem**
- O tradutor que lê o código fonte duas vezes é chamado de tradutor de duas passagens (ou *two-pass translator*)
- Seguindo a mesma nomenclatura, um montador que lê o código fonte duas vezes é chamado de montador de duas passagens (*two-pass assembler*)

Estrutura Interna de um Montador

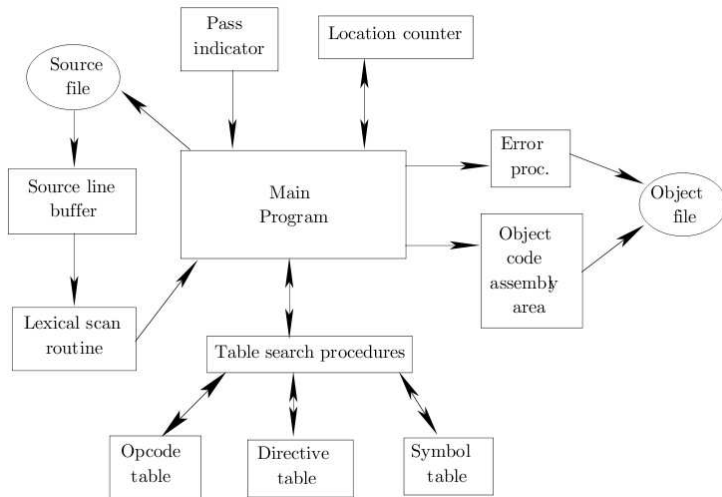


Figura: Estrutura Interna de um Montador

Montador de duas passagens

- Iniciaremos a nossa discussão do pelo algoritmo de duas passagens
- O processo executado pelo montador em cada passagem é descrito como segue:
- **Primeira Passagem**
 - Na primeira passagem, o montador coleta informações de definições de rótulos, símbolos, etc, e os armazena em uma tabela
- **Segunda Passagem**
 - Na segunda passagem, os valores (endereços) dos símbolos já são conhecidos e cada declaração pode, então, ser “montada”

O Algoritmo de duas passagens

Primeira Passagem

- A função principal da primeira passagem é a construção da Tabela de Símbolos
- **Tabela de símbolos (TS):** contém todos os símbolos definidos no programa e seus atributos
- **Símbolo:** Um símbolo é um rótulo ou o nome de uma variável declarada com um nome simbólico

Exemplo

Algoritmo 4 Exemplo 4

```
1: ...  
2: N1: CONST 0  
3: N2: SPACE  
4: L: OUTPUT X
```


Estruturas de Dados

- Além da tabela de símbolos, o montador utiliza as seguintes estruturas de dados:
- **Tabela de instruções:**
 - contém os opcodes simbólicos acompanhadas das informações para gerar código de máquina
 - Basicamente, esta tabela contém o formato da instrução e opcode numérico.
 - Esta tabela está pronta, isto é, faz parte do código do montador.

Estruturas de Dados

- **Contador de linhas:**
 - indica a linha do programa fonte que está sendo analisada no momento (útil para impressão de mensagens de erro)
- **Contador de posições (location counter):**
 - indica a posição de memória a ser ocupada (preenchida) pelo código de máquina.
- **Tabela de símbolos (TS):**
 - contém todos os símbolos definidos no programa e seus atributos. Essa tabela é criada na 1ª passagem e usada na 2ª
- **Tabela de diretivas:**
 - contém as diretivas da linguagem. Em geral, para cada diretiva existe uma rotina que implementa as ações correspondentes. Esta tabela também está pronta.
 - Nos exemplos, nós vamos assumir que as diretivas CONST and SPACE ocupam 1 palavra de memória (16 bits)

Algoritmo da primeira passagem:

```
contador_posição = 0
contador_linha = 1
Enquanto arquivo fonte não chegou ao fim, faça:
    Obtém uma linha do fonte
    Separa os elementos da linha:
        rótulo, operação, operandos, comentários
    Ignora os comentários
    Se existe rótulo:
        Procura rótulo na TS (Tabela de Símbolos)
    Se achou: Erro → símbolo redefinido
    Senão: Insere rótulo e contador_posição na TS
    Procura operação na tabela de instruções
    Se achou:
        contador_posição = contador_posição + tamanho da instrução
    Senão:
        Procura operação na tabela de diretivas
        Se achou:
            chama subrotina que executa a diretiva
            contador_posição = valor retornado pela subrotina
        Senão: Erro, operação não identificada
    contador_linha = contador_linha + 1
```

Figura: Primeira Passagem

Exemplo Primeira Passagem 1

■ Código fonte:

```
INPUT      N1
INPUT      N2
LOAD       N1
ADD        N2
STORE      N3
OUTPUT     N3
STOP
N1:  SPACE
N2:  SPACE
N3:  SPACE
```

■ Tabela de Símbolos

```
N1:  13
N2:  14
N3:  15
```

Exemplo Primeira Passagem 2

- Considere o programa abaixo e assuma que a entrada seja 8.
- Para o programa você deverá:
 - Mostrar a saída
 - Tente identificar o que está sendo computado
 - Utilizar o algoritmo de primeira passada para criar a tabela de símbolos

	COPY	ZERO,	OLDER
	COPY	ONE,	OLD
	INPUT	LIMIT	
	OUTPUT	OLD	
FRONT:	LOAD	OLDER	
	ADD	OLD	
	STORE	NEW	
	SUB	LIMIT	
	JMPP	FINAL	
	OUTPUT	NEW	
	COPY	OLD,	OLDER
	COPY	NEW,	OLD
	JMP	FRONT	
FINAL:	OUTPUT	LIMIT	
	STOP		
ZERO:	CONST	0	
ONE:	CONST	1	
OLDER:	SPACE		
OLD:	SPACE		
NEW:	SPACE		
LIMIT:	SPACE		

Solução

```
      COPY      ZERO,  OLDER
      COPY      ONE,   OLD
      INPUT     LIMIT
      OUTPUT    OLD
FRONT: LOAD     OLDER
      ADD       OLD
      STORE     NEW
      SUB       LIMIT
      JMPP      FINAL
      OUTPUT    NEW
      COPY      OLD,   OLDER
      COPY      NEW,   OLD
      JMP       FRONT
FINAL: OUTPUT    LIMIT
      STOP
ZERO:  CONST    0
ONE:   CONST    1
OLDER: SPACE
OLD:   SPACE
NEW:   SPACE
LIMIT: SPACE
```

Mostra a Série de Fibonacci
8 → 1,1,2,3,5,8

Símbolo	Valor
FRONT	10
FINAL	30
ZERO	33
ONE	34
OLDER	35
OLD	36
NEW	37
LIMIT	38

Exemplo Primeira Passagem 3

- Crie um programa que compute o fatorial de um número.
- Utilize o algoritmo da primeira passagem para gerar a tabela de símbolos.

Solução

- Esse programa computa o fatorial do valor lido pela entrada padrão

Código Fonte

```

      INPUT      N
      LOAD      N
FAT:   SUB       ONE
      JNPZ      FIM
      STORE     AUX
      MULT      N
      STORE     N
      LOAD      AUX
      JNP       FAT
FIM:   OUTPUT   N
      STOP
AUX:   SPACE
N:     SPACE
ONE:   CONST    1
    
```

N	ACC	AUX
5	5	
5	$5-1=4$	4
20	$4*5=20$	4
20	4	4

N	ACC	AUX
20	4	4
20	$4-1=3$	3
60	$3*20=60$	3
60	3	3

N	ACC	AUX
60	3	3
60	$3-1=2$	2
120	$2*60=12$	2
120	2	2

N	ACC	AUX
120	2	2
120	$2-1=1$	1
120	$1*120=12$	1
120	1	1

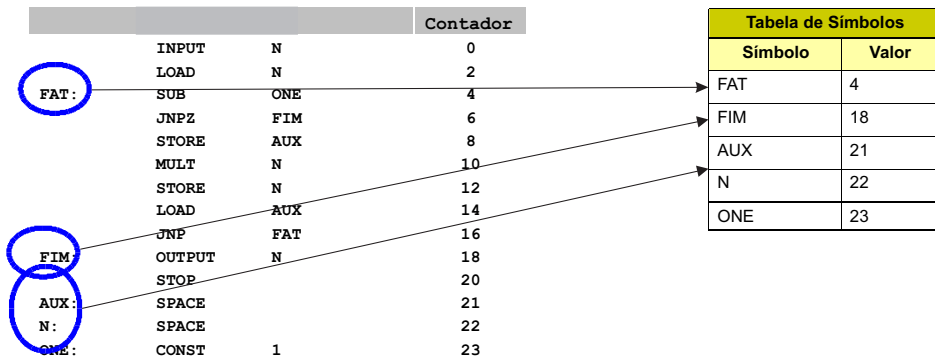
N	ACC	AUX
120	1	1
120	$1-1=0$	1

Input → 5

Output → **120**

Solução

- Vamos ilustrar o algoritmo de duas passagens com um programa que computa o fatorial de um número lido do fluxo de entrada padrão



Próxima Aula

Montador passagem única