Laboratório de Sistemas
Embarcados e Distribuídos

Conjunto de Instruções
Parte VI – Geração de Código Executável

Histórico de revisões

Revisão	Data	Responsável	Descrição
0.1	-x-	Prof. Cesar Zeferino	Primeira versão
0.2	04/2017	Felipe Viel	Revisão do modelo
0.3	09/2022	Thiago Felski Pereira	Revisão para o RISC-V

Observação: Este material foi produzido por pesquisadores do Laboratório de Sistemas Embarcados e Distribuídos (LEDS – Laboratory of Embedded and Distributed Systems) da Universidade do Vale do Itajaí e é destinado para uso em aulas ministradas por seus pesquisadores.

Introdução

- **Objetivo**
  - Conhecer o processo de geração de um código executável, identificando os programas utilizados e as funções de cada um deles

- **Conteúdo**
  - Geração de código executável
  - Compilador
  - Montador
  - Ligador
  - Carregador

Introdução

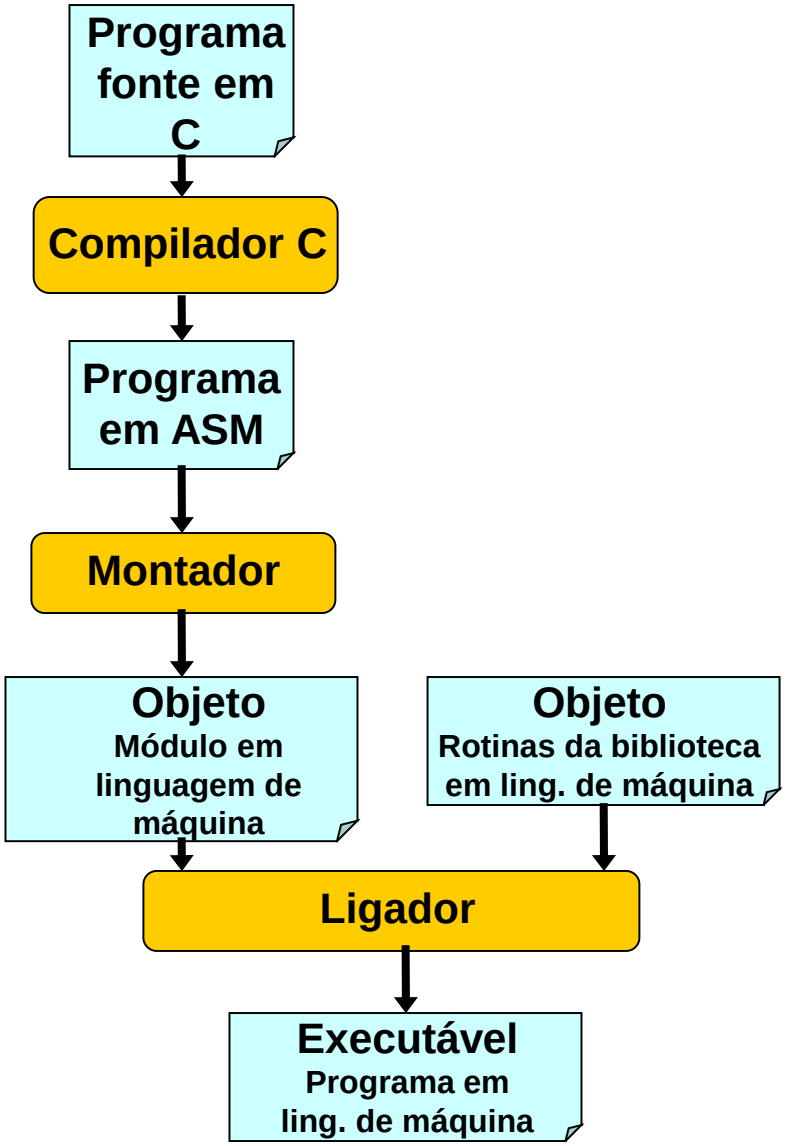
Bibliografia

- PATTERSON, David A.; HENNESSY, John L. Abstrações e tecnologias computacionais. *In*: _____. **Organização e projeto de computadores**: a interface hardware/software. 4. ed. Rio de Janeiro: Campus, 2014. cap. 2. Disponível em: <http://www.sciencedirect.com/science/book/9788535235852>. Acesso em: 13 mar. 2017.

- Edições anteriores
  - Patterson e Hennessy (2005, cap. 2)
  - Patterson e Hennessy (2000, cap. 3)

1 Suporte a procedimentos

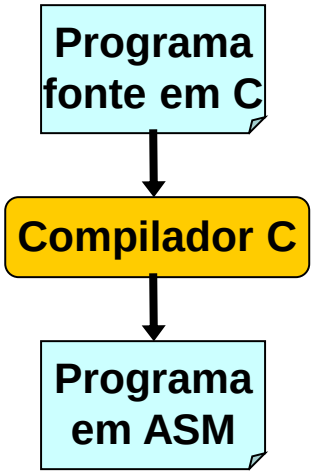
- **Geração do código executável**
  - A partir de um código fonte escrito em uma linguagem de alto nível (ex: C), são necessários três programas utilitários:
    - **Compilador**
    - **Montador**
    - **Ligador**



2 Compilador

Função

- Transforma um programa escrito em uma linguagem de alto nível (ex: C ou Pascal) em um equivalente expresso na linguagem de montagem (assembly)



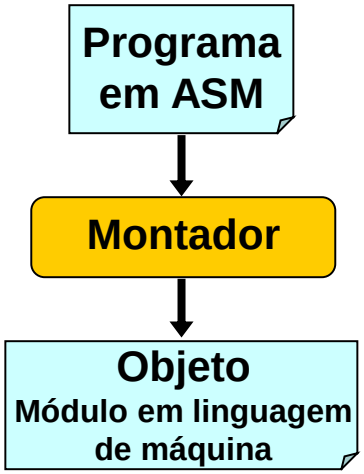
OBS

- Alguns compiladores produzem diretamente o módulo-objeto, eliminando a necessidade de usar um montador

3 Montador

Função

- Transforma um programa expresso na linguagem de montagem (assembly) para a linguagem de máquina gerando um código não executável denominado arquivo-objeto (ou módulo-objeto)



- O montador precisa saber os endereços que correspondem a cada um dos **labels** do programa. Isso é feito por meio de uma tabela chamada tabela de símbolos que armazena pares de símbolo/endereço

3 Montador

- **Exemplo de estrutura de arquivo-objeto (UNIX)**
  - Cabeçalho
  - Segmento de texto
  - Segmento de dados
  - Informações sobre relocação
  - Tabela de símbolos
  - Informações para análise de erros

3 Montador

- **Exemplo de estrutura de arquivo-objeto (UNIX)**
  - **Cabeçalho**
    - Descreve o tamanho e a posição das demais parte do arquivo
  - **Segmento de texto**
    - Contém o código na linguagem de máquina
  - **Segmento de dados**
    - Contém os dados que forem necessários ao programa (estáticos ou dinâmicos)
  - **Informações sobre relocação**
    - Identificam instruções e dados que dependem de endereços absolutos, por ocasião da carga do programa na memória
  - **Tabela de símbolos**
    - Contém labels não-definidos durante a montagem (ex: referências externas)
  - **Informações para análise de erros**
    - Descrição concisa de como os módulos foram compilados de modo que depurador possa associar instruções de máquina com arquivos fonte em C

3 Montador

Pseudo-instruções

- Além do conjunto de instruções definidas na linguagem de máquina do processador o montador oferece ao programador as chamadas pseudo-instruções

- Uma pseudo-instrução é traduzida para uma instrução do processador

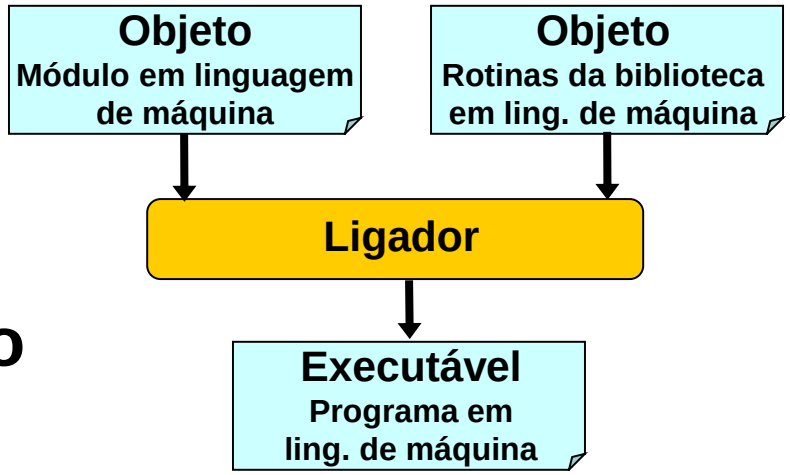
- Exemplo
  - **move t0,t1** ⟶ **add t0,zero, t1**

3 Montador

- **Outros tipos de pseudo-instruções (ver Apêndice A)**
  - Aritméticas e Lógicas
  - Manipulação de Constantes
  - Comparação
  - Carga
  - Armazenamento
  - Movimento de dados
  - Ponto-flutuante

4 Ligador (ou linker)

- **Função**
  - Liga os módulos-objeto e gera um programa executável



- **Passos básicos do trabalho do ligador**
  - Colocar os módulos de código e de dados simbolicamente na memória
  - Determinar os endereços dos rótulos de dados e instruções
  - Resolver as referências internas e externas

4 Ligador (ou linker)

Observações importantes

- O ligador usa informações de relocação e a tabela de símbolos de cada módulo-objeto para resolver todos os labels indefinidos
- O ligador edita os módulos, substituindo endereços antigos por novos
- O ligador produz um arquivo executável que tem um formato parecido com o de um módulo-objeto, mas exclui várias seções (referências não resolvidas, informações de relocação, tabela de símbolos, informações p/ o depurador), ou seja, possui as seguintes seções
  - **Cabeçalho**
  - **Segmento de texto**
  - **Segmento de dados**

4 Ligador (ou linker)

Exemplo de ligação de dois módulos (A e B)

- Módulo A

Cabeçalho do arquivo objeto	**Nome**	**Procedimento A**	
	Tamanho do texto	**100_{hexa}**	
	Tamanho dos dados	**20_{hexa}**	
Segmento de texto (código)	Endereço	Instrução	
	0	**`lw a0, 0(gp)`**	
	4	**`jalr zero, 0`**	
	...	**...**	
Segmento de dados	Endereço	Dado	
	0	**`(X)`**	
	...	**...**	
Informação de relocação	Endereço	Instrução	Dependência
	0	**lw**	**X**
	4	**jalr**	**B**
Tabela de símbolos	Label	Endereço	
	X	**??**	
	B	**??**	

4 Ligador (ou linker)

Exemplo de ligação de dois módulos (A e B)

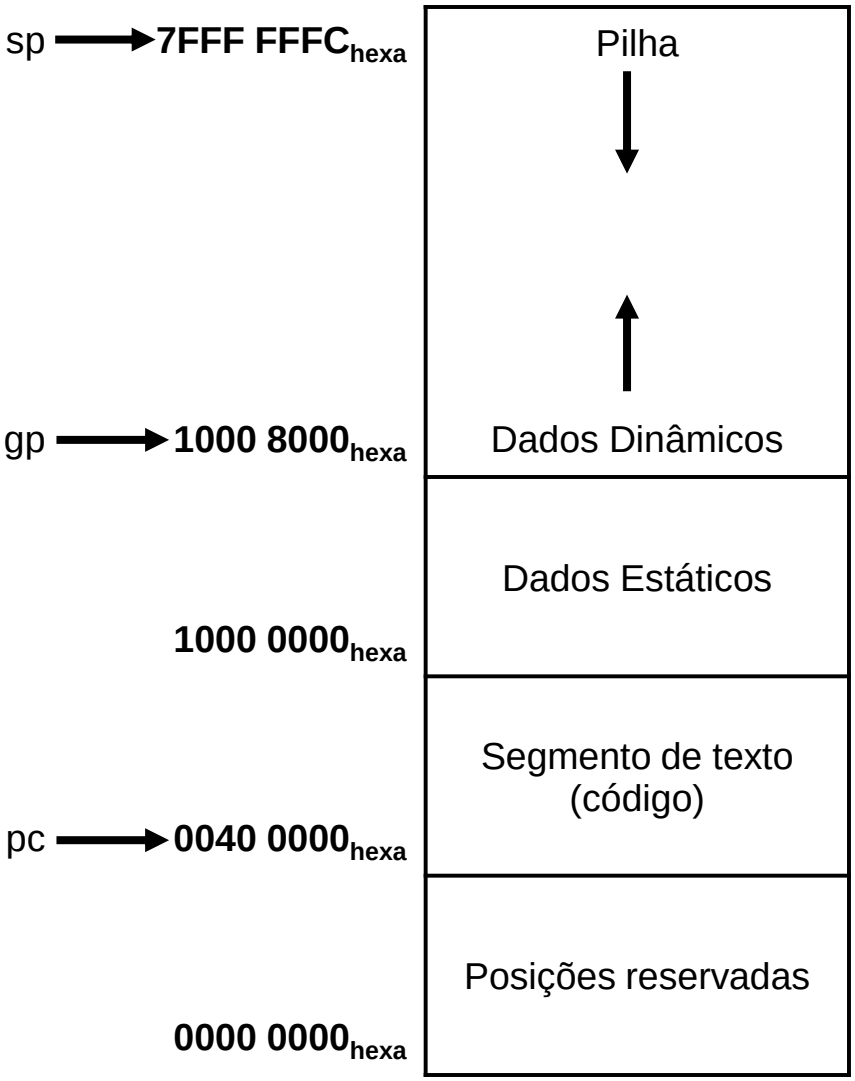
- Módulo B

Cabeçalho do arquivo objeto	**Nome**	**Procedimento B**	
	Tamanho do texto	**200_{hexa}**	
	Tamanho dos dados	**30_{hexa}**	
Segmento de texto (código)	Endereço	Instrução	
	0	**`sw a1, 0(gp)`**	
	4	**`jalr zero, 0`**	
	...	**...**	
Segmento de dados	Endereço	Dado	
	0	**`(Y)`**	
	...	**...**	
Informação de relocação	Endereço	Instrução	Dependência
	0	**sw**	**Y**
	4	**jalr**	**A**
Tabela de símbolos	Label	Endereço	
	Y	**??**	
	A	**??**	

4 Ligador (ou linker)

Exemplo de ligação de dois módulos (A e B)

- Mapa de memória



4 Ligador (ou linker)

Exemplo de ligação de dois módulos (A e B)

Arquivo executável

Cabeçalho do arquivo executável	**Tamanho do texto**	**300_{hexa}**
	Tamanho dos dados	**50_{hexa}**
Segmento de texto (código)	Endereço	Instrução
	$0040\ 0000_{hexa}$	`lw a0, 8000hexa (gp)`
	$0040\ 0004_{hexa}$	`jal 400100hexa`
	...	...
	$0040\ 0100_{hexa}$	`sw a1, 8020hexa (gp)`
	$0040\ 0104_{hexa}$	`jalr 400000hexa`
	...	...
Segmento de dados	Endereço	Dado
	$1000\ 0000_{hexa}$	(X)
	...	...
	$1000\ 0020_{hexa}$	(Y)
	...	...

NOTA: O imediato 8000_{hexa}, na primeira instrução do programa, é um valor negativo que somado ao conteúdo de $gp (1000 8000_{hexa}) resulta em $1000\ 0000_{hexa}$ que é a posição de X

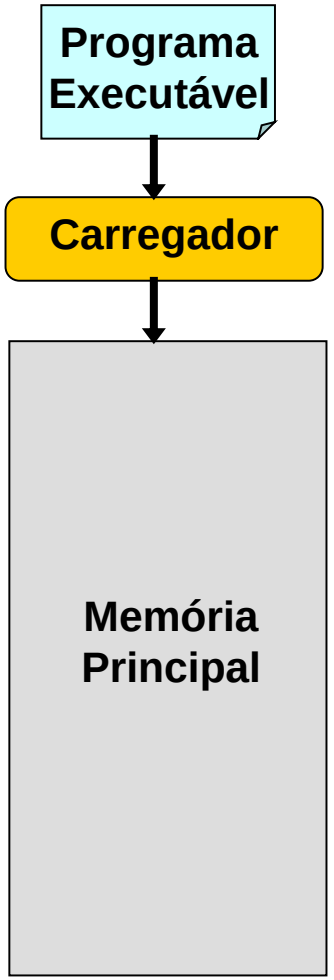
5 Carregador

Função

- Transfere o programa executável do disco para a memória principal

Carregamento de um executável no Unix

- O carregador lê o cabeçalho para determinar os tamanhos dos segmentos de texto e de dados
- Cria espaço de endereços com o tamanho necessário
- Copia as instruções e os dados para a memória
- Copia os parâmetros (se houver algum) para a pilha do programa principal
- Inicializa os registradores (incluindo o sp)
- Desvia para a rotina de inicialização que copia os parâmetros nos registradores de argumento e chama a rotina principal



6 Exemplo no RARS


```
int main(){
    int x = 5;
    int y = 2;
    int z = 0;

    z = x + y;
}
```

```
.data
    x : .word 5
    y : .word 2
    z : .word 0
.text
main:
    la   t3, x
    lw   t4, 0(t3)      # t4=x

    la   t3, y
    lw   t5, 0(t3)      # t5=y

    la   t3, z

    add  t6,t4, t5   #z=x+y
    sw   t6, 0(t3)
```

6 Exemplo no RARS

- **Programa após a utilização das ferramentas (Toolchains)**

Address	Code	Basic	Source
0x00400000	0x0fc10e17	auipc x28,0x0000fc10	8: la t3, x
0x00400004	0x000e0e13	addi x28,x28,0	
0x00400008	0x000e2e83	lw x29,0(x28)	9: lw t4, 0(t3) #t4 = x
0x0040000c	0x0fc10e17	auipc x28,0x0000fc10	11: la t3, y
0x00400010	0xff8e0e13	addi x28,x28,0xfffffff8	
0x00400014	0x000e2f03	lw x30,0(x28)	12: lw t5, 0(t3) #t5 = y
0x00400018	0x01ee8fb3	add x31,x29,x30	14: add t6, t4, t5 #t6 = x + y
0x0040001c	0x0fc10e17	auipc x28,0x0000fc10	16: la t3, z
0x00400020	0xfece0e13	addi x28,x28,0xffffffec	
0x00400024	0x01fe2023	sw x31,0(x28)	17: sw t6, 0(t3) #z = t6