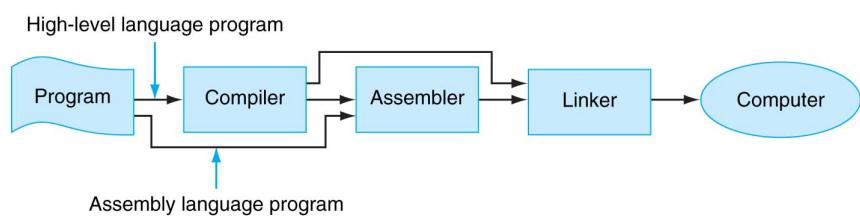


Arquitetura de Computadores I

O processo de assemblagem

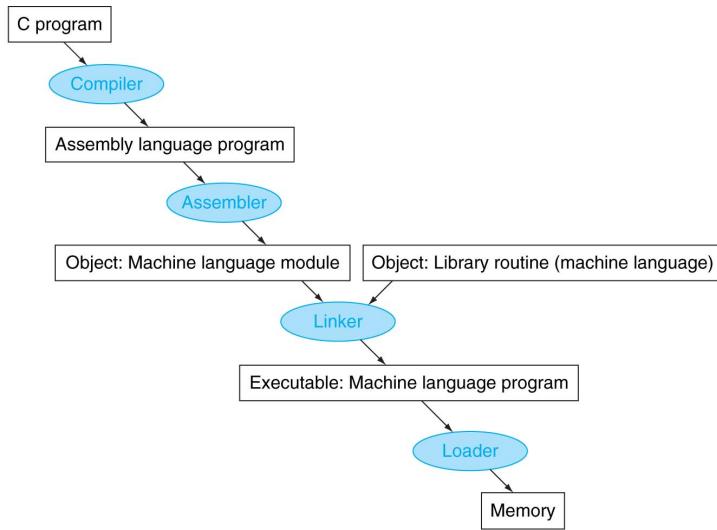
António de Brito Ferrari

ferrari@ua.pt



Assembly: diretamente escrito pelo programador ou resultado da tradução feita pelo compilador de uma linguagem de “alto nível”

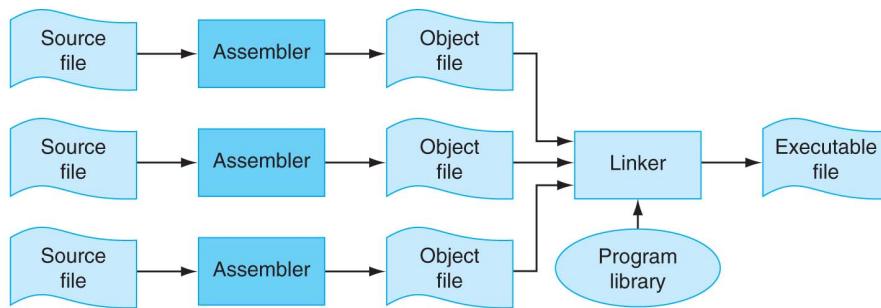
Tradução linguagem de alto nível para código executável



ABF - AC I - MIPS assembler

3

O Processo de “assemblagem”



ABF - AC I - MIPS assembler

4

Tarefas do assembler

- Seguir Diretivas
- Traduzir Pseudoinstruções em instruções nativas
- Construir a lista de labels e endereços (***Symbol table***)
- Gerar linguagem máquina
- Crear ***Object File***

Diretivas

- As diretivas indicam ao assembler como traduzir o programa mas não geram instruções máquina
- **.text (addr)**: indica que o que segue é para ser colocado no segmento de texto (a partir de `addr`)
- **.align n**: alinhar o dado que segue com 2^n byte boundary; `align 2` ⇒ word boundary seguinte
- **.globl sym**: declara `sym` global (`sym` pode ser referenciado noutras ficheiros)
- **.data (addr)**: os items subsequentes são colocados no user data segment (a partir de `addr`)
- **.asciiz str**: armazenar a string `str` na memória inserindo o carácter null no final

Pseudoinstruções

Pseudoinstrução

- move \$t0, \$t1
- blt \$t0, \$t1, L1
- li \$t0, 0x12345
- add \$t1, 0x12345

Instrução nativa

- add \$t0, \$zero, \$t1
- slt \$at, \$t0, \$t1
- bne \$at, \$zero, L1
- lui \$t0, 0x1
- ori \$t0, \$t0, 0x2345
- lui \$at, 0x1
- ori \$at, \$at, 0x2345
- add \$t1, \$t1, \$at

\$at (r1) – registo reservado para armazenar variáveis temporárias do assembler

ABF - AC I - MIPS assembler

7

Assembler de duas passagens

- O assembler percorre duas vezes o texto do programa (duas passagens):
 1. Constrói a *Symbol Table*
 2. Gera o código máquina
- Labels
 - Local – o objeto apenas pode ser usado no ficheiro
 - Global (Externo) – pode ser referenciado por outros ficheiros para além daquele em que é definido

ABF - AC I - MIPS assembler

8

Tradução C ➡️ Assembly

```

int f, g, y; // global           .data
f:             f:
g:             g:
y:             y:
.text
main:
    addi $sp, $sp, -4      # stack frame
    sw   $ra, 0($sp)       # store $ra
    addi $a0, $0, 2         # $a0 = 2
    sw   $a0, f             # f = 2
    addi $a1, $0, 3         # $a1 = 3
    sw   $a1, g             # g = 3
    jal  sum               # call sum
    sw   $v0, y             # y = sum()
    lw   $ra, 0($sp)       # restore $ra
    addi $sp, $sp, 4         # restore $sp
    jr  $ra                # return to OS
sum:
    add  $v0, $a0, $a1      # $v0 = a + b
    jr  $ra                # return

```

ABF - AC I - MIPS assembler

9

Symbol Table

Symbol	Address
f	0x10000000
g	0x10000004
y	0x10000008
main	0x00400000
sum	0x0040002C

ABF - AC I - MIPS assembler

10

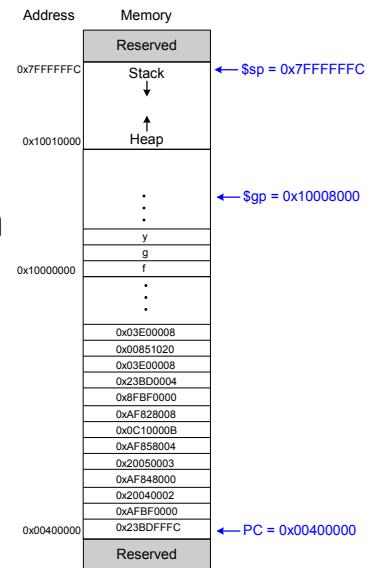
Ficheiro Executável

Executable file header	Text Size	Data Size
	0x34 (52 bytes)	0xC (12 bytes)
Text segment	Address	Instruction
	0x00400000	0x23BDFFFF
	0x00400004	sw \$ra, 0 (\$sp)
	0x00400008	addi \$a0, \$0, 2
	0x0040000C	sw \$a0, 0x8000 (\$gp)
	0x00400010	addi \$a1, \$0, 3
	0x00400014	sw \$a1, 0x8004 (\$gp)
	0x00400018	jal 0x0040002C
	0x0040001C	sw \$v0, 0x8008 (\$gp)
	0x00400020	lw \$ra, 0 (\$sp)
	0x00400024	addi \$sp, \$sp, -4
	0x00400028	jr \$ra
	0x0040002C	add \$v0, \$a0, \$a1
	0x00400030	jr \$ra
Data segment	Address	Data
	0x10000000	f
	0x10000004	g
	0x10000008	y

ABF - AC I - MIPS assembler

11

Programa na memória



ABF - AC I - MIPS assembler

12

Formato do ficheiro objeto (UNIX)

Object file header	Text segment	Data segment	Relocation information	Symbol table	Debugging information
--------------------	--------------	--------------	------------------------	--------------	-----------------------

Object File Header – descreve o tamanho e posição dos outros campos do ficheiro objeto

Text segment – contém o código máquina das rotinas no ficheiro fonte.

Este código pode não ser executável devido a referências externas.

Data segment – contém a representação em binário dos dados no ficheiro fonte.

Os dados podem estar incompletos devido a referências externas.

Relocation information – identifica as instruções e dados que dependem dos endereços absolutos quando o programa é carregado em memória.

Esta informação é necessária porque o assembler não sabe que posições de memória o código e os dados vão ocupar quando forem ligados com o resto do programa.

Symbol table – os labels que ainda não estão definidos (external references,...) e aqueles que estando definidos são globais (i.e. referenciados noutras ficheiros objeto)

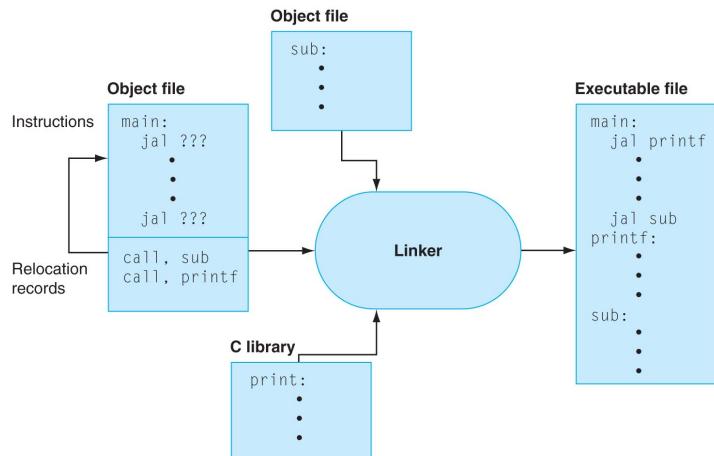
Debugging information – informação que permite associar instruções máquina com as instruções no ficheiro fonte (source file)

13

Link editor ou linker

- Pega em diferentes programas objeto (p.ex. Programa e rotinas da biblioteca que invoca) e liga-os
- 3 passos de processamento:
 1. Procurar nas bibliotecas as rotinas usadas pelo programa
 2. Determinar as posições de memória que o código de cada módulo ocupará e relocalizar as respetivas instruções ajustando as referências absolutas
 3. Resolver as referências entre ficheiros

O *linker*: resolução das referências externas



ABF - AC I - MIPS assembler

15

Object file header			
	Name	Procedure A	
	Text size	100 _{hex}	
	Data size	20 _{hex}	
Text segment		Address	Instruction
	0	1w \$a0, 0(\$gp)	
	4	jal 0	
	
Data segment		0	(X)
	
	Relocation information	Address	Instruction type
	0	1w	X
	4	jal	B
	Symbol table		Label Address
	X	-	
	B	-	
Object file header			
Name	Procedure B		
	Text size	200 _{hex}	
	Data size	30 _{hex}	
Text segment		Address	Instruction
	0	sw \$a1, 0(\$gp)	
	4	jal 0	
	
Data segment		0	(Y)
	
	Relocation information	Address	Instruction type
	0	sw	Y
	4	jal	A
	Symbol table		Label Address
	Y	-	
	A	-	

Linking de ficheiros objeto

Exemplo:

Símbolos a atualizar na “linkagem”:
Endereços das procedures A e B
Instruções que referenciam as
Words X e Y

16

Executable file header		
	Text size	300 _{hex}
	Data size	50 _{hex}
Text segment	Address	Instruction
	0040 0000 _{hex}	lw \$a0, 8000 _{hex} (\$gp)
	0040 0004 _{hex}	jal 40 0100 _{hex}

	0040 0100 _{hex}	sw \$a1, 8020 _{hex} (\$gp)
	0040 0104 _{hex}	jal 40 0000 _{hex}

Data segment	Address	
	1000 0000 _{hex}	(X)

	1000 0020 _{hex}	(Y)

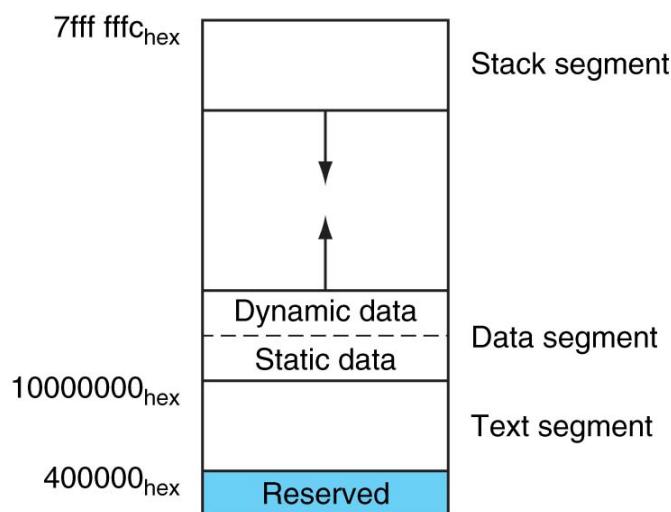
MIPS Text Segment 40 0000_{hex} Data Segment 1000 0000_{hex}	Código de A colocado a partir de 40 0000 _{hex} Dados de A (X) colocado em 1000 0000 _{hex} - MIPS assembler	Código de B colocado a partir de 40 0100 _{hex} Dados de B (Y) colocado em 1000 0020 _{hex}

Código Executável

Loader (UNIX)

1. Lê o header do ficheiro executável para determinar o tamanho dos segmentos de texto e dados
2. Cria um espaço de endereçagem para o programa (texto + dados + stack)
3. Copia instruções e dados para o espaço de endereçagem atribuído
4. Copia os argumentos passados ao programa para o stack
5. Inicializa os registos ($\$sp =$ topo do stack)
6. Salta para a rotina de inicialização que copia os argumentos do stack para os registos e invoca o programa. Quando este termina e retorna a rotina de inicialização termina o programa com a system call *exit*

Mapa da memória



Programa assembly

```

.text
.align 2
.globl main
main:
    subu    $sp, $sp, 32
    sw     $ra, 20($sp)
    sd     $a0, 32($sp)
    sw     $0, 24($sp)
    sw     $0, 28($sp)

loop:
    lw      $t6, 28($sp)
    mul   $t7, $t6, $t6
    lw      $t8, 24($sp)
    addu  $t9, $t8, $t7
    sw     $t9, 24($sp)
    addu  $t0, $t6, 1
    sw     $t0, 28($sp)
    ble   $t0, 100, loop
    la      $a0, str
    lw      $a1, 24($sp)
    jal   printf
    move  $v0, $0
    lw      $ra, 20($sp)
    addu  $sp, $sp, 32
    jr     $ra
}

.str:
    .asciiz "The sum from 0 .. 100 is %d\n"

```

Symbol Table

Símbolo	Endereço
main	
loop	
str	

ABF - AC I - MIPS assembler

21