EP - MIPS OAC1

Edgar Henrique de Oliveira Lira
Victor Sacchi

Organização e Arquitetura MIPS

Linguagem de montagem

- Linguagem Intermediária
- Sistemas embarcados
- Bibliotecas em várias linguagens e SO
- Programação lenta
- Difícil manutenção

Elementos da linguagem de montagem

- Rótulos
- Mnemônicos
- Operandos
- Comentários

```
cond:  div $a1, $a2 # Comentário
```

Tipos de Sentenças MIPS

Instrução

- Instrução padrão do MIPS

Diretiva

- .text
- .data
- .asciiz

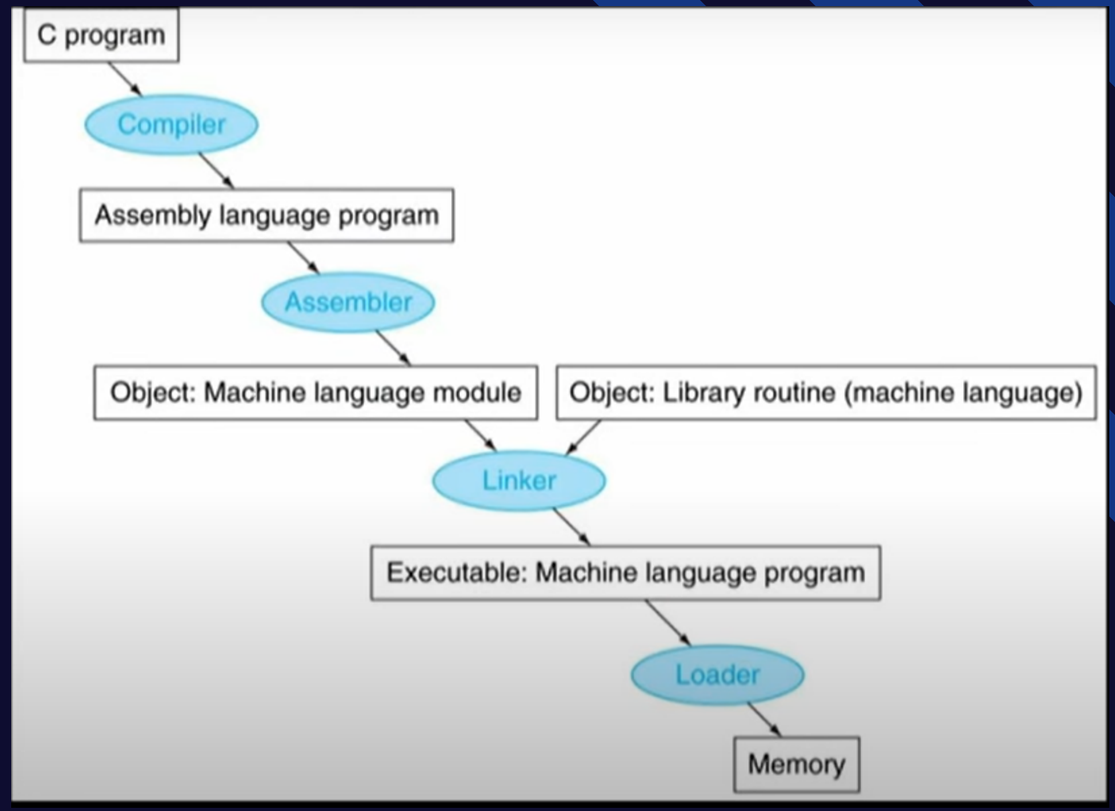
Macro

- .macro
- Evita ações repetitivas

Comentário

Compilação de Arquivo C

Link-Editor Montador MIPS



Principios de Projeto MIPS

- Simplicidade favorece regularidade
- Menor é mais rápido
- Faça o caso comum rápido
- Um bom projeto exige bons compromissos.

Formatos das instruções

Instrução	Campos					
Formato	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Tipo R	op	rs	rt	rd	shamt	funct
Tipo I	op	rs	rt	Endereço/Imediato		
Tipo J	op	Endereço de Destino				

Uso convencional registradores MIPS

Registradores	Uso
$a0… $a3	Usados em argumentos
$t0… $t7	Registradores temporários, valores não precisam ser salvos entre chamadas
$zero	Constante 0, usado em diferentes operações que usa-se 0
$s0… $s7	Registradores com dados que precisam ser salvos entre as chamadas
$sp	Stack Pointer, aponta para o último local na pilha
$ra	Usado como endereço de retorno de uma chamada
$v0 e $v1	Avaliação de expressão e resultado de uma função

Resolução do Problema
Código em C

Números Complexos

Números Deficientes

Números Perfeitos

```c
#include <stdio.h>
int main(void) {

  int x = 0, soma = 0;
  scanf("%d", &x);

  for (int i = 1; i < x; i++){
    if (x % i == 0) soma = soma + i;
    }


  if(soma > x)
    printf("numero abundante\n");

  if(soma == x)
    printf("numero perfeito\n");

  if(soma < x)
    printf("numero deficiente\n");

  return 0;
}

```

Código em Assembly

Função Main e escanear Inteiro

```
.data
# Variaveis que guardam uma String
digite: .asciiz "Digite um número:"
perfeito: .asciiz "Numero Perfeito"
deficiente: .asciiz "Numero Deficiente"
abundante: .asciiz "Numero Abundante"

.text
.globl main
main:
      li $a1,0                # inicia o local do numero que sera digitado
      li $a2,0                # inicia o contador
      li $a3,0                # inicia o lugar onde ficara a soma dos inteiros

      jal scanInt             # vai para função de escanear um inteiro

      jal divisores           # chama divisores
      jal resultado           # Chama o resultado

      li $v0,10               # termina
      syscall                 # chama a função de v0

#Recebe um inteiro digitado pelo usuário
scanInt:

      la $a0, digite          # Da um load no endereço de digite no registrador $a0
      li $v0, 4               # Carrega a instrução 4 para o registrador $v0
      syscall                 # Executa a intrução que esta no registrador $v0
      li $v0, 5               # Carrega a instrução 5 para o registrador $v0
      syscall                 # Executa a intrução que esta no registrador $v0
      move $a1, $v0           # Move os dados de $v0 para $a1
      jr $ra
```

Função de Somar divisores do Int

```
divisores:

      add $a2,$a2,1                   # soma 1 do contador
        blt $a2,$a1,cond              # verifica se chegou no final
        j divisores_fim               # vai para o final

cond:
      div $a1, $a2
      mfhi $s4
      beq $s4, 0, salva_e_soma
      j divisores

salva_e_soma:

        subu $sp,$sp,8          # Abre um espaco na pilha
        sw $a2,4($sp)           # Salva o $a0
        sw $ra,0($sp)           # Salva o endereco de retorno
        jal divisores           # chama o fatorial
        lw $ra,0($sp)           # recupera o valor original do $a2 em $t0
        lw $t0,4($sp)           # recupera o valor original do $a2 em $t0
        addu $sp,$sp,8          # retira o espaco ocupado na pilha
        add $a3,$a3,$t0         # soma

divisores_fim:
        jr $ra                          # retorna
```

Rotinas para o print da classificação do número

```
# Função que chama a função do resultado baseado na comparação entre a soma dos divisores do número e o próprio número
resultado:

      beq $a3, $a1, nPerfeito       # Vai para a funçao nPerfeito
      blt $a3, $a1, nDeficiente     # Vai para a função nDeficiente
      bgt $a3, $a1, nAbundante      # Vai para a função nAbundante


nPerfeito: # Printa Numero perfeito na tela

      li $v0, 4               # Carrega a instrução 4 para o registrador $v0
      la $a0, perfeito        # Carrega a string de perfeito para o registrador $a0
      syscall                 # Mostra na tela o conteudo do registrador $a0
      jr $ra                  # Vai para função encerra

nDeficiente: # Printa Numero deficiente na tela

      li $v0, 4               # Carrega a instrução 4 para o registrador $v0
      la $a0, deficiente      # Carrega a string de deficiente para o registrador $a0
      syscall                 # Mostra na tela o conteudo do registrador $a0
      jr $ra

nAbundante: # Printa Numero abundante na tela

      li $v0, 4               # Carrega a instrução 4 para o registrador $v0
      la $a0, abundante       # Carrega a string de abundante para o registrador $a0
      syscall                 # Mostra na tela o conteudo do registrador $a0
      jr $ra
```

Instruções utilizadas no MIPS

- j
- jal
- jr
- li
- add
- addu
- sub
- move
- la
- lw
- sw
- beq
- bgt
- blt
- mfhi
- div
- subu

Resumo das instruções

j - jal - jr

Desviam para alguma rotina do código

beq - bgt - blt

Desviam condicionalmente para alguma rotina do código, recebendo 2 operandos para comparação

add - addu - sub - subu

Operações aritméticas, recebem 1 operando de destino e 2 de origem

mfhi - move - la - li - lw - sw

Movem informações de um registrador para outro