Do alto-nível ao *assembly*

Compiladores, Aula Nº 2

João M. P. Cardoso

Viagem

- Como são implementadas as estruturas computacionais em *assembly*?
  - Revisão dos conceitos relacionados com a programação em *assembly* para o MIPS R3000 (ver disciplina de Arquitectura de Computadores)
- Fizemos o papel de um compilador. Agora vamos aprender, entre outras coisas, a fazer um compilador.

Do alto-nível ao *assembly*

- Alvo: MIPS R3000

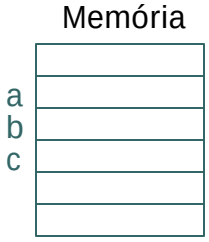
```
Int sum(int A[], int N) {
  Int i, sum = 0;
  For(i=0; i<N; i++) {
    sum = sum + A[i];
  }
  return sum;
}
```

```
# $a0 armazena o endereço de A[0]
# $a1 armazena o valor de N
Sum:    Addi    $t0, $0, 0          # i = 0
        Addi    $v0, $0, 0          # sum = 0
Loop:   beq     $t0, $a1, End       # if(i == 10) goto End;
        Add     $t1, $t0, $t0       # 2*i
        Add     $t1, $t1, $t1       # 2*(2*i) = 4*i
        Add     $t1, $t1, $a0       # 4*i + base(A)
        Lw      $t2, 0($t1)         # load A[i]
        Add     $v0, $v0, $t2       # sum = sum + A[i]
        Addi    $t0, $t0, 1         # i++
        J       Loop                # goto Loop;
End:    jr      $ra                 # return
```


Do alto-nível ao *assembly*

- Variáveis globais:
  - Armazenadas na memória
  - Para cada uso de uma variável global o compilador tem de gerar instruções *load/store*

```
int a, b, c;

... fun(...) {
   ...
}
```

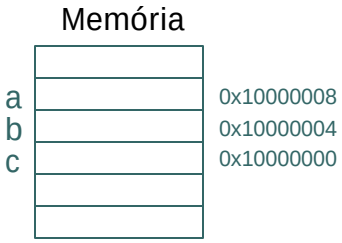


Variáveis Globais

```
.data   0x10000000
a:      .space  4
b:      .space  4
c:      .space  4
        .text
fun:    …
        la      $t1, a
        lw      $t1, 0($t1)
        la      $t2, b
        lw      $t2, 0($t2)
        add     $t3, $t2, $t1
        la      $t4, c
        sw      $t3, 0($t4)
        …
```

Alocação de memória

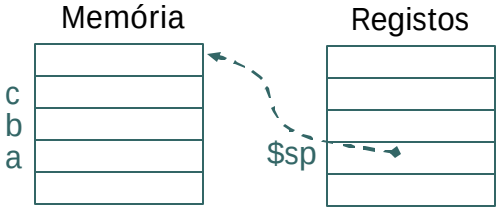
```
Int a, b, c;

void fun() {
  c = a + b;
}
```



Do alto-nível ao *assembly*

- Conceito de chamada a procedimentos
  - Cada procedimento tem estados
    - Variáveis locais
    - Endereço de retorno
  - Estado é guardado na área de memória designada por pilha de chamadas (é utilizado um registo para apontar para a posição actual da pilha)
- Pilha de chamadas
  - A pilha de chamadas encontra-se no topo da memória
  - A pilha cresce para baixo

```
void fun() {
   int a, b, c;
   ...
   c = a + b;
   ...
}
```

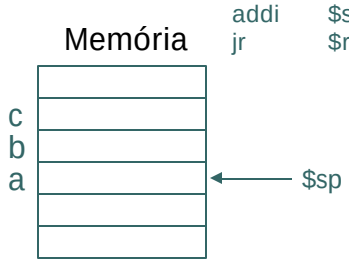


Variáveis Locais

- Exemplo:

```
fun:    addi    $sp, $sp, -12     } Reserva espaço na pilha
        …
        lw      $t1, 0($sp)       } Load a
        lw      $t2, 4($sp)       } Load b
        add     $t3, $t2, $t1     } a + b
        sw      $t3, 8($sp)       } store c
        …
        addi    $sp, $sp, 12      } liberta espaço na pilha
        jr      $ra
```

```
void fun() {
   int a, b, c;
   ...
   c = a + b;
   ...
}
```



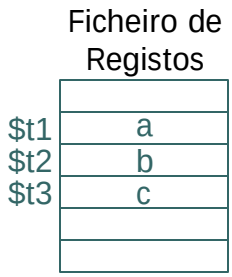
Variáveis Locais

- Acesso aos registos internos do processador é muito mais rápido
  - Mas os registos internos são em número limitado
  - E por isso nem todas as variáveis locais podem ser armazenadas nesses registos
- No passado a atribuição de registos internos do processador a variáveis locais era feita pelo programador:
  - A linguagem C tem uma palavra reservada para orientar o compilador: **register** (*e.g.*, register int c;)
- Hoje os compiladores são muito mais eficientes
  - Essa atribuição é lhe inteiramente delegada

Variáveis Locais

- Utilização de registos internos

```
void fun() {
   int a, b, c;
   ...
   c = a + b;
   ...
}
```

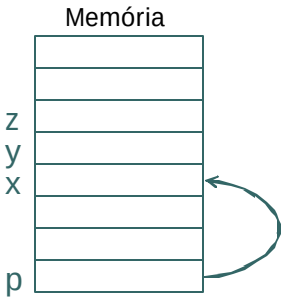



```
fun:    ...
        add     $t3, $t2, $t1
        ...
        jr      $ra
```


Do alto-nível ao *assembly*

- Implementar Registos
  - Registos contêm vários campos
- Cada estrutura é armazenada em posições contíguas de memória

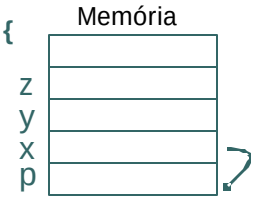
```
typedef struct {
  int x, y, z;
} foo;

foo *p;
```



Do alto-nível ao *assembly*

- Exemplo com estrutura local:

```
typedef struct {
  int x, y, z;
} foo;

fun() {
  foo *p;
  p->x = p->y + p->z;
}
```

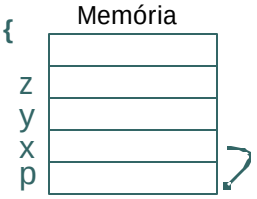



```
fun:  addi  $sp, $sp, -16   } Reserva espaço na pilha
      ...
      lw    $t1, 0($sp)     } Endereço de p
      addi  $t1, $t1, 8     } address p->y
      lw    $t2, 0($t1)     } Load p->y
      lw    $t1, 0($sp)     } Endereço de p
      addi  $t1, $t1, 12    } address p->z
      lw    $t3, 0($t1)     } Load p->z
      add   $t3, $t2, $t3   } p->y + p->z
      lw    $t1, 0($sp)     } Endereço de p
      addi  $t1, $t1, 4     } address p->x
      sw    $t3, 0($t1)     } store em p->x
      ...
      addi  $sp, $sp, 16    } liberta espaço na pilha
      jr    $ra
```


Do alto-nível ao *assembly*

- Exemplo com estrutura local (optimizado):

```
typedef struct {
  int x, y, z;
} foo;

fun() {
  foo *p;
  p->x = p->y + p->z;
}
```




```
fun:  addi  $sp, $sp, -16   } Reserva espaço na pilha
      ...
      lw    $t2, 8($sp)     } Load p->y
      lw    $t3, 12($sp)    } Load p->z
      add   $t3, $t2, $t3   } p->y + p->z
      sw    $t3, 4($sp)     } store em p->x
      ...
      addi  $sp, $sp, 16    } liberta espaço na pilha
      ...
```