TÉCNICO LISBOA

Compiladores[1]

Geração de Código II

Capítulo 11 - "Compiladores: Da Teoria à Prática"

Prof. Alberto Abad

IST - Universidade de Lisboa

2021/2022

[1] Slides adaptados de Prof. Pedro T. Monteiro (2017/2018)

Consultem o Manual de Referência !

https://web.tecnico.ulisboa.pt/~david.matos/w/pt/index.php/Postfix_Reference_Guide

Frame Pointer

Frame Pointer (**FP**) - indica na pilha o contexto local de uma função

Frame Pointer

Frame Pointer (**FP**) - indica na pilha o contexto local de uma função

- Na entrada de função:
  - o **FP** guarda o valor do **SP**

Frame Pointer

Frame Pointer (**FP**) - indica na pilha o contexto local de uma função

- Na entrada de função:
  - o **FP** guarda o valor do **SP**
- Durante a execução:
  - o **SP** pode variar
  - o **FP** mantém-se constante
  - os argumentos da função e as variáveis locais podem ser acedidas através de um *offset* fixo

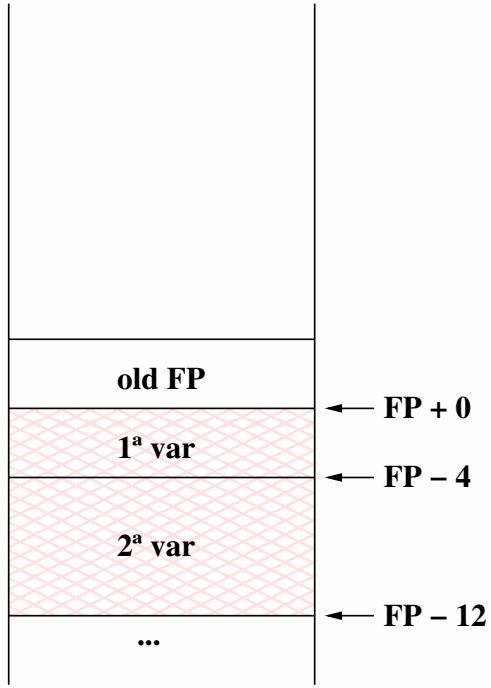
Variáveis locais

- Existem apenas no contexto de uma função
- Residem na pilha
- Acessíveis através de um *offset*
- *offset* é negativo relativamente ao **FP**

Variáveis locais

Variáveis locais

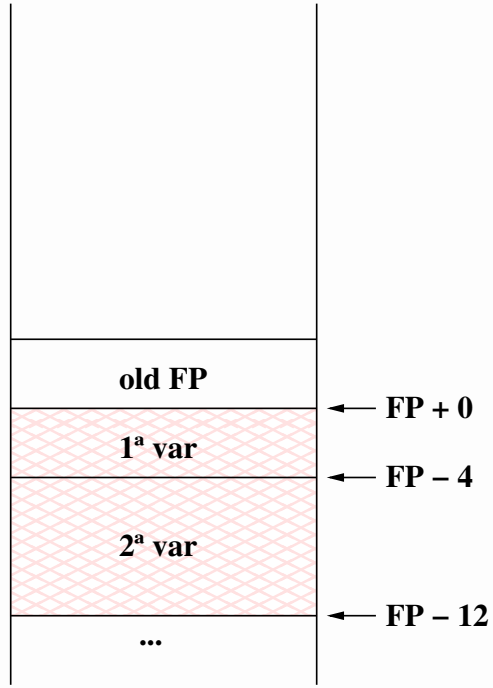
- Existem apenas no contexto de uma função
- Residem na pilha
- Acessíveis através de um *offset*
- *offset* é negativo relativamente ao **FP**



Variáveis locais

Variáveis locais

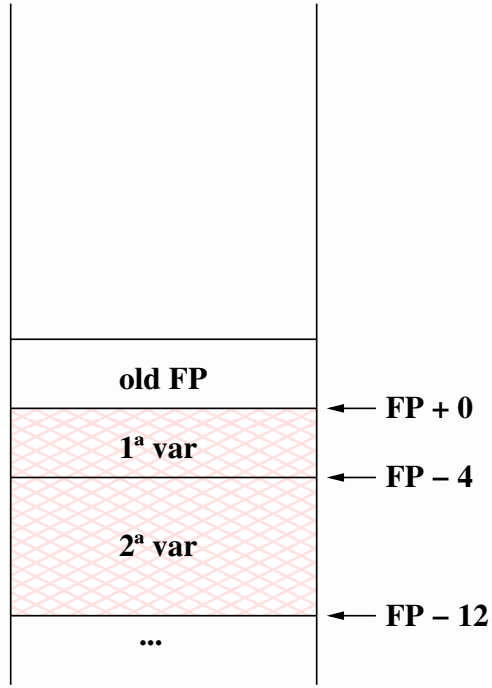
- Existem apenas no contexto de uma função
- Residem na pilha
- Acessíveis através de um *offset*
- *offset* é negativo relativamente ao **FP**



LOCAL - carrega o endereço de um *offset* para a pilha

Variáveis locais

- Existem apenas no contexto de uma função
- Residem na pilha
- Acessíveis através de um *offset*
- *offset* é negativo relativamente ao **FP**



LOCAL - carrega o endereço de um *offset* para a pilha
Depois o endereço é lido/escrito da forma habitual:

- **LDINT**, **LDDOUBLE**, ...
- **STINT**, **STDOUBLE**, ...

Endereçamento de variáveis locais

Endereçamento de variáveis locais

```
v = 2;
```

```
INT 2
DUP32
LOCAL -4
STINT
TRASH 4
```

Endereçamento de variáveis locais

```
v = 2;
```

```
INT 2
DUP32
LOCAL -4
STINT
TRASH 4
```

```
v = x;
```

```
LOCAL -8
LDINT
DUP32
LOCAL -4
STINT
TRASH 4
```

Definição de função

```
void fname(int a, char * b) {
   int i;
   double d;
   ...
}
```

Funções

Definição de função

```
void fname(int a, char * b) {
   int i;
   double d;
   ...
}
```

TEXT

ALIGN

GLOBAL fname, FUNC

LABEL fname

ENTER ?

Funções

Definição de função

```
void fname(int a, char * b) {
   int i;
   double d;
   ...
}
```

```
TEXT
ALIGN
GLOBAL fname, FUNC
LABEL fname
ENTER 12
;; evaluate function body
LEAVE
RET
```

Definição de função

```
void fname(int a, char * b) {
    int i;
    double d;
    ...
}
```

TEXT

```
ALIGN
```

GLOBAL fname, FUNC

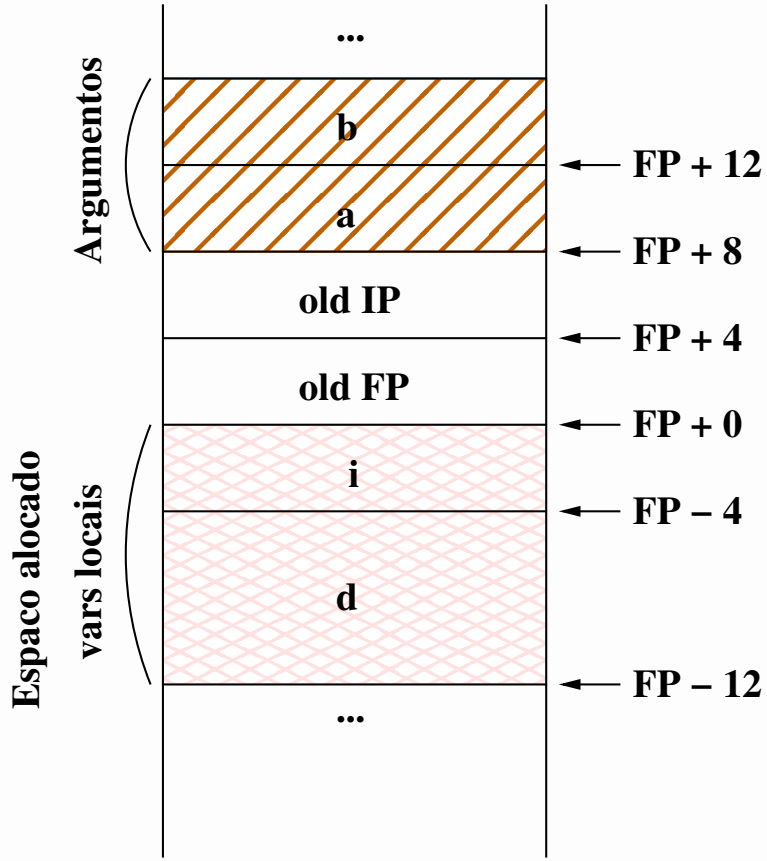
LABEL fname

ENTER 12

```
;; evaluate function body
```

LEAVE

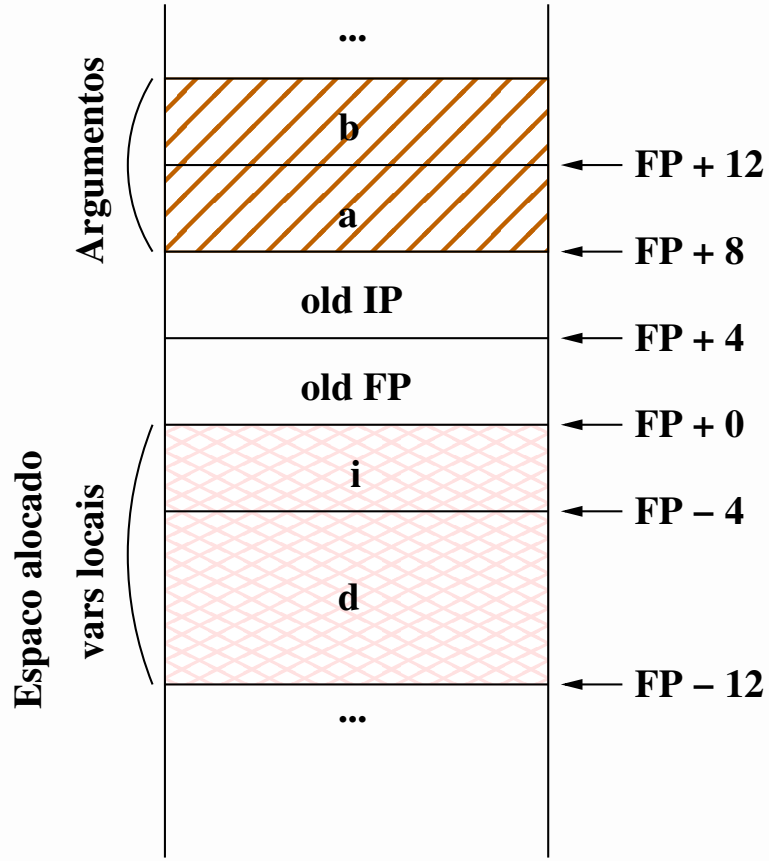
RET



Definição de função

```
void fname(int a, char * b) {
    int i;
    double d;
    ...
}
```

```
TEXT
ALIGN
GLOBAL fname, FUNC
LABEL fname
ENTER 12
;; evaluate function body
LEAVE
RET
```



Q: Como o copilador sabe o tamanho do ENTER?

Funções

- **LEAVE** - restaura o **FP** anterior e apaga a pilha até ao **FP**
- **RET** - restaura o **IP** anterior e apaga-o da pilha

Funções

- **LEAVE** - restaura o **FP** anterior e apaga a pilha até ao **FP**
- **RET** - restaura o **IP** anterior e apaga-o da pilha

Q: Como é que o valor de retorno é passado?

Funções

- **LEAVE** - restaura o **FP** anterior e apaga a pilha até ao **FP**
- **RET** - restaura o **IP** anterior e apaga-o da pilha

Q: Como é que o valor de retorno é passado?

- **STFVAL** - retira o valor da pilha para o registo **EAX/ST0**
- **LDFVAL** - retira o valor do registo **EAX/ST0** para a pilha

TÉCNICO LISBOA

Declaração de variável funcional (global)

```
var vname = (int a, [text] b) -> void :
   int i
   double d
   ...
```

Funções em L22

Declaração de variável funcional (global)

```
var vname = (int a, [text] b) -> void :
   int i
   double d
   ...
```

```
DATA
ALIGN
GLOBAL vname, OBJ
LABEL vname
SADDR _F123
...
```

```
...
TEXT _F123
ALIGN
LABEL _F123
ENTER 12
;; evaluate function body
LEAVE
RET
```

Funções em L22

Declaração de variável funcional (global)

```
var vname = (int a, [text] b) -> void :
   int i
   double d
   ...
```

```
DATA
ALIGN
GLOBAL vname, OBJ
LABEL vname
SADDR _F123
...
```

```
...
TEXT _F123
ALIGN
LABEL _F123
ENTER 12
;; evaluate function body
LEAVE
RET
```

Q: Como seria com variáveis funcionais locais?

Invocação de variável funcional (global)

```
int<int, int> vname
...
vname(0, 1)
...
```

TÉCNICO LISBOA

Invocação de variável funcional (global)

```
int<int, int> vname
...
vname(0, 1)
...
...
INT 1
INT 0
ADDR vname
LDINT
BRANCH
TRASH 4
...
```

Invocação de variável funcional (global)

```
int<int, int> vname
...
vname(0, 1)
...
...
INT 1
INT 0
ADDR vname
LDINT
BRANCH
TRASH 4
...
```

Q: Como seria com variáveis funcionais locais?

TÉCNICO LISBOA

Invocação de variável funcional (global)

```
int<int, int> vname
...
vname(0, 1)
...
...
INT 1
INT 0
ADDR vname
LDINT
BRANCH
TRASH 4
...
```

Q: Como seria com variáveis funcionais locais?

Q: Como seria com funções externas foreign?

Aritmética de ponteiros

Um vector é um ponteiro para a base acrescido de um deslocamento

Aritmética de ponteiros

Um vector é um ponteiro para a base acrescido de um deslocamento

A soma de um ponteiro p com um inteiro i:

- é um ponteiro p (contém endereço)
- deslocado de `i * sizeof(tipo apontado pelo ponteiro)`

Aritmética de ponteiros

Um vector é um ponteiro para a base acrescido de um deslocamento

A soma de um ponteiro p com um inteiro i:

- é um ponteiro p (contém endereço)
- deslocado de `i * sizeof(tipo apontado pelo ponteiro)`

`v[i]` $\equiv$ `v+i`

Aritmética de ponteiros

Um vector é um ponteiro para a base acrescido de um deslocamento

A soma de um ponteiro *p* com um inteiro *i*:

- é um ponteiro *p* (contém endereço)
- deslocado de `i * sizeof(tipo apontado pelo ponteiro)`

`v[i]` ≡ `v+i`

```
LOCAL -4 ; v
LDINT
LOCAL -8 ; i
LDINT
INT 4    ; 8 bytes if double, 4 otherwise
MUL
ADD
```

Alocação de memória

```
int *i = [3];
```

Alocação de memória

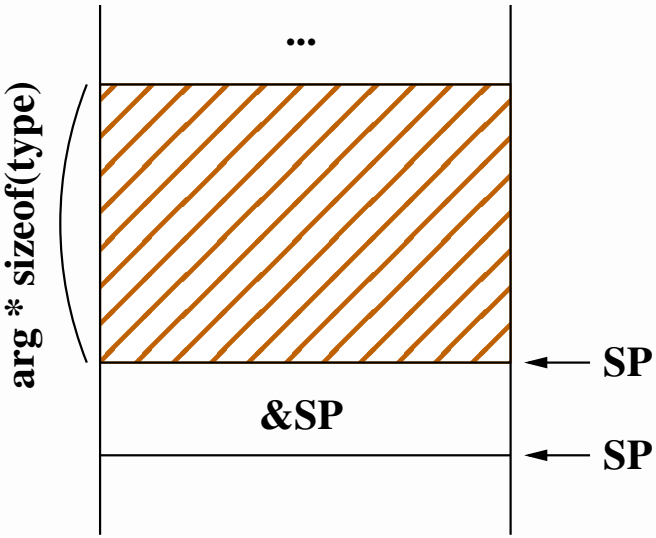
```
int *i = [3];
```

...
ALLOC
SP

Alocação de memória

```
int *i = [3];
```

...
ALLOC
SP



Exercício

```
int *click(int *x, int dim) {
  int *res, i;
  for (i = dim-2, res = x+dim-1; i >= 0; i--)
    if (x[i] > *res) res = &x[i];
  return res;
}
```

Exercício

```
int *click(int *x, int dim) {
  int *res, i;
  for (i = dim-2, res = x+dim-1; i >= 0; i--)
    if (x[i] > *res) res = &x[i];
  return res;
}
```

```
TEXT
ALIGN
GLOBAL click, FUNC
LABEL click ; x@+8 dim@+12
ENTER 8     ; res@-4 i@-8

LOCAL 12
LDINT
INT 2
SUB
DUP32
LOCAL -8
STINT
TRASH 4

LOCAL 8
LDINT
LOCAL 12
LDINT
INT 1
SUB
INT 4
MUL
ADD
DUP32
LOCAL -4
STINT
TRASH 4
```

Exercício

```
int *click(int *x, int dim) {
  int *res, i;
  for (i = dim-2, res = x+dim-1; i >= 0; i--)
    if (x[i] > *res) res = &x[i];
  return res;
}
```

```
TEXT
ALIGN
GLOBAL click, FUNC
LABEL click ; x@+8 dim@+12
ENTER 8     ; res@-4 i@-8

LOCAL 12
LDINT
INT 2
SUB
DUP32
LOCAL -8
STINT
TRASH 4

LOCAL 8
LDINT
LOCAL 12
LDINT
INT 1
SUB
INT 4
MUL
ADD
DUP32
LOCAL -4
STINT
TRASH 4
```

```
ALIGN
LABEL forcond
LOCAL -8
LDINT
INT 0
GE
JZ forend
```

Exercício

```c
int *click(int *x, int dim) {
  int *res, i;
  for (i = dim-2, res = x+dim-1; i >= 0; i--)
    if (x[i] > *res) res = &x[i];
  return res;
}
```

```
TEXT
ALIGN
GLOBAL click, FUNC
LABEL click ; x@+8 dim@+12
ENTER 8     ; res@-4 i@-8

LOCAL 12
LDINT
INT 2
SUB
DUP32
LOCAL -8
STINT
TRASH 4

LOCAL 8
LDINT
LOCAL 12
LDINT
INT 1
SUB
INT 4
MUL
ADD
DUP32
LOCAL -4
STINT
TRASH 4

ALIGN
LABEL forcond
LOCAL -8
LDINT
INT 0
GE
JZ forend


LOCAL +8
LDINT
LOCAL -8
LDINT
INT 4
MUL
ADD
LDINT ; x[i] = *(x+i)
LOCAL -4
LDINT ; res
LDINT ; * res
GT
JZ ifend
```

Exercício

```
int *click(int *x, int dim) {
  int *res, i;
  for (i = dim-2, res = x+dim-1; i >= 0; i--)
    if (x[i] > *res) res = &x[i];
  return res;
}
```

```
TEXT
ALIGN
GLOBAL click, FUNC
LABEL click ; x@+8 dim@+12
ENTER 8     ; res@-4 i@-8

LOCAL 12
LDINT
INT 2
SUB
DUP32
LOCAL -8
STINT
TRASH 4

LOCAL 8
LDINT
LOCAL 12
LDINT
INT 1
SUB
INT 4
MUL
ADD
DUP32
LOCAL -4
STINT
TRASH 4
```

```
ALIGN
LABEL forcond
LOCAL -8
LDINT
INT 0
GE
JZ forend


LOCAL +8
LDINT
LOCAL -8
LDINT
INT 4
MUL
ADD
LDINT ; x[i] = *(x+i)
LOCAL -4
LDINT ; res
LDINT ; * res
GT
JZ ifend
```

```
LOCAL 8
LDINT
LOCAL -8
LDINT
INT 4
MUL
ADD ; x+i=&x[i]
DUP32
LOCAL -4
STINT
TRASH 4
```

Exercício

```
int *click(int *x, int dim) {
  int *res, i;
  for (i = dim-2, res = x+dim-1; i >= 0; i--)
    if (x[i] > *res) res = &x[i];
  return res;
}
```

```
TEXT
ALIGN
GLOBAL click, FUNC
LABEL click ; x@+8 dim@+12
ENTER 8     ; res@-4 i@-8

LOCAL 12
LDINT
INT 2
SUB
DUP32
LOCAL -8
STINT
TRASH 4

LOCAL 8
LDINT
LOCAL 12
LDINT
INT 1
SUB
INT 4
MUL
ADD
DUP32
LOCAL -4
STINT
TRASH 4
```

```
ALIGN
LABEL forcond
LOCAL -8
LDINT
INT 0
GE
JZ forend


LOCAL +8
LDINT
LOCAL -8
LDINT
INT 4
MUL
ADD
LDINT ; x[i] = *(x+i)
LOCAL -4
LDINT ; res
LDINT ; * res
GT
JZ ifend
```

```
LOCAL 8
LDINT
LOCAL -8
LDINT
INT 4
MUL
ADD ; x+i=&x[i]
DUP32
LOCAL -4
STINT
TRASH 4


ALIGN
LABEL ifend

ALIGN
LABEL forincr
LOCAL -8
LDINT
DUP32
INT 1
SUB
LOCAL -8
STINT
TRASH 4
JMP forcond
```

Exercício

```c
int *click(int *x, int dim) {
  int *res, i;
  for (i = dim-2, res = x+dim-1; i >= 0; i--)
    if (x[i] > *res) res = &x[i];
  return res;
}
```

```
TEXT
ALIGN
GLOBAL click, FUNC
LABEL click ; x@+8 dim@+12
ENTER 8     ; res@-4 i@-8

LOCAL 12
LDINT
INT 2
SUB
DUP32
LOCAL -8
STINT
TRASH 4

LOCAL 8
LDINT
LOCAL 12
LDINT
INT 1
SUB
INT 4
MUL
ADD
DUP32
LOCAL -4
STINT
TRASH 4

ALIGN
LABEL forcond
LOCAL -8
LDINT
INT 0
GE
JZ forend


LOCAL +8
LDINT
LOCAL -8
LDINT
INT 4
MUL
ADD
LDINT ; x[i] = *(x+i)
LOCAL -4
LDINT ; res
LDINT ; * res
GT
JZ ifend

LOCAL 8
LDINT
LOCAL -8
LDINT
INT 4
MUL
ADD ; x+i=&x[i]
DUP32
LOCAL -4
STINT
TRASH 4


ALIGN
LABEL ifend

ALIGN
LABEL forincr
LOCAL -8
LDINT
DUP32
INT 1
SUB
LOCAL -8
STINT
TRASH 4
JMP forcond

ALIGN
LABEL forend
LOCAL -4
LDINT

STFVAL32
LEAVE
RET
```

Arranque do programa

Funções:

- Normalmente, as funções nomeadas existem no contexto global
- Em L22:
  - Apenas existe uma função "nomeada": o main
  - O início e fim do main está delimitado pelas keywords begin e end
  - O resto de funções não são nomeadas e podem existir funções dentro de funções (Atenção: **TEXT**)

Arranque do programa

Funções:

- Normalmente, as funções nomeadas existem no contexto global
- Em L22:
  - Apenas existe uma função "nomeada": o main
  - O início e fim do main está delimitado pelas keywords begin e end
  - O resto de funções não são nomeadas e podem existir funções dentro de funções (Atenção: **TEXT**)

Ponto de entrada: Label _main

```
TEXT
ALIGN
GLOBAL _main, FUNC
LABEL _main
ENTER ...
```

Geração de código assembly

Conversão directa de código POSTFIX para código assembly:

- compilador pf2asm (`https://web.tecnico.ulisboa.pt/~david.matos/w/pt/images/d/d9/Pf2asm-202206081024.tar.bz2`)

Geração de código assembly

Conversão directa de código POSTFIX para código assembly:

- compilador pf2asm (`https://web.tecnico.ulisboa.pt/~david.matos/w/pt/images/d/d9/Pf2asm-202206081024.tar.bz2`)

O projecto gera directamente código assembly!

- Visitor `postfix_writer`, chama métodos
  `cdk::basic_postfix_emitter`
  `cdk::postfix_ix86_emitter`
- Opção `-g` para modo debug

Geração de código assembly

Conversão directa de código POSTFIX para código assembly:

- compilador pf2asm (`https://web.tecnico.ulisboa.pt/~david.matos/w/pt/images/d/d9/Pf2asm-202206081024.tar.bz2`)

O projecto gera directamente código assembly!

- Visitor `postfix_writer`, chama métodos
  `cdk::basic_postfix_emitter`
  `cdk::postfix_ix86_emitter`
- Opção `-g` para modo debug

Vantagens do pf2asm:

- Geração directa de protótipos em código POSTFIX
- Permite a realização de testes

Geração de código assembly

Exemplo - A-01-1-N-ok.l22

```
begin
  writeln "string"
  return 0
end
```

Geração de código assembly

Exemplo - A-01-1-N-ok.l22

```
begin
  writeln "string"
  return 0
end
```

```
TEXT
ALIGN
GLOBAL _main, FUNC
LABEL _main
ENTER 0
RODATA
ALIGN
LABEL _L2
SSTRING "string"
TEXT
ADDR _L2
CALL prints
TRASH 4
CALL println
INT 0
STFVAL32
JMP _L1
LABEL _L1
LEAVE
RET
EXTERN println
EXTERN prints
```

Geração de código assembly

Exemplo - A-01-1-N-ok.l22

```
begin
  writeln "string"
  return 0
end
```

```
TEXT
ALIGN
GLOBAL _main, FUNC
LABEL _main
ENTER 0
RODATA
ALIGN
LABEL _L2
SSTRING "string"
TEXT
ADDR _L2
CALL prints
TRASH 4
CALL println
INT 0
STFVAL32
JMP _L1
LABEL _L1
LEAVE
RET
EXTERN println
EXTERN prints
```

```
segment .text
align   4
global  _main:function
_main:
        push    ebp
        mov     ebp, esp
        sub     esp, 0
segment .rodata
align   4
_L2:
        db      "string", 0
segment .text
        push    dword $_L2
        call    prints
        add     esp, 4
        call    println
        push    dword 0
        pop     eax
        jmp
dword _L1
_L1:
        leave
        ret
extern  println
extern  prints
```

Comando: `pf2asm file.pf` → `file.asm`

TÉCNICO LISBOA

Yasm Modular Assembler

- Assembler para a arquitectura Intel x86
- Permite múltiplos formatos de output (ELF, win32, ...)
  https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

Comando: `yasm -felf -o file.o file.asm` → `file.o`

Geração de um executável

Linker:

- Ligação dos módulos binários e bibliotecas num executável
- Linux: GNU `ld`

Geração de um executável

Linker:

- Ligação dos módulos binários e bibliotecas num executável
- Linux: GNU `ld`

A biblioteca RTS fornece funções para interação:

- com o ambiente de execução
- com o programador

Manual da RTS:
https://web.tecnico.ulisboa.pt/~david.matos/w/pt/index.php/Manual_da_RTS

Geração de um executável

Linker:

- Ligação dos módulos binários e bibliotecas num executável
- Linux: GNU `ld`

A biblioteca RTS fornece funções para interação:

- com o ambiente de execução
- com o programador

Manual da RTS:

https://web.tecnico.ulisboa.pt/~david.matos/w/pt/index.php/Manual_da_RTS

Comando:

```
ld -m elf_i386 -o file file.o -L$HOME/compiladores/root/usr/lib -lrts
→ file
```

Exercício

```
static double v[] = { 1.41, 2.71, 3.14 };
int f(double v[], int s, double d) {
  for (int i = 0; i < s; i++)
    if (v[i] == d) return i;
  return -1;
}
int main() {
  return f(v, 3, 3.14);
}
```

Exercício

```
static double v[] = { 1.41, 2.71, 3.14 };
int f(double v[], int s, double d) {
  for (int i = 0; i < s; i++)
    if (v[i] == d) return i;
  return -1;
}
int main() {
  return f(v, 3, 3.14);
}
```

```
DATA
ALIGN
LABEL v
SDOUBLE 1.41
SDOUBLE 2.71
SDOUBLE 3.14
```

Exercício

```
static double v[] = { 1.41, 2.71, 3.14 };
int f(double v[], int s, double d) {
  for (int i = 0; i < s; i++)
    if (v[i] == d) return i;
  return -1;
}
int main() {
  return f(v, 3, 3.14);
}
```

```
DATA
ALIGN
LABEL v
SDOUBLE 1.41
SDOUBLE 2.71
SDOUBLE 3.14

TEXT
ALIGN
GLOBAL f, FUNC
LABEL f
ENTER 4
; i@-4 v@8 s@+12 d@+16
```

Exercício

```
static double v[] = { 1.41, 2.71, 3.14 };
int f(double v[], int s, double d) {
  for (int i = 0; i < s; i++)
    if (v[i] == d) return i;
  return -1;
}
int main() {
  return f(v, 3, 3.14);
}
```

```
DATA
ALIGN
LABEL v
SDOUBLE 1.41
SDOUBLE 2.71
SDOUBLE 3.14


TEXT
ALIGN
GLOBAL f, FUNC
LABEL f
ENTER 4
; i@-4 v@8 s@+12 d@+16
```

```
INT 0
LOCAL -4
STINT
ALIGN
LABEL forcond
LOCAL -4
LDINT
LOCAL 12
LDINT
LT
JZ forend

LOCAL +8
LDINT
LOCAL -4
LDINT
INT 8
MUL
ADD
LDDOUBLE
```

Exercício

```
static double v[] = { 1.41, 2.71, 3.14 };
int f(double v[], int s, double d) {
  for (int i = 0; i < s; i++)
    if (v[i] == d) return i;
  return -1;
}
int main() {
  return f(v, 3, 3.14);
}
```

```
DATA
ALIGN
LABEL v
SDOUBLE 1.41
SDOUBLE 2.71
SDOUBLE 3.14


TEXT
ALIGN
GLOBAL f, FUNC
LABEL f
ENTER 4
; i@-4 v@8 s@+12 d@+16
```

```
INT 0
LOCAL -4
STINT
ALIGN
LABEL forcond
LOCAL -4
LDINT
LOCAL 12
LDINT
LT
JZ forend

LOCAL +8
LDINT
LOCAL -4
LDINT
INT 8
MUL
ADD
LDDOUBLE
```

```
LOCAL 16
LDDOUBLE
DCMP
INT 0
EQ
JZ ifend

LOCAL -4
LDINT
STFVAL32
LEAVE
RET
ALIGN
LABEL ifend
```

Exercício

```
static double v[] = { 1.41, 2.71, 3.14 };
int f(double v[], int s, double d) {
  for (int i = 0; i < s; i++)
    if (v[i] == d) return i;
  return -1;
}
int main() {
  return f(v, 3, 3.14);
}
```

```
DATA
ALIGN
LABEL v
SDOUBLE 1.41
SDOUBLE 2.71
SDOUBLE 3.14

TEXT
ALIGN
GLOBAL f, FUNC
LABEL f
ENTER 4
; i@-4 v@8 s@+12 d@+16

INT 0
LOCAL -4
STINT
ALIGN
LABEL forcond
LOCAL -4
LDINT
LOCAL 12
LDINT
LT
JZ forend

LOCAL +8
LDINT
LOCAL -4
LDINT
INT 8
MUL
ADD
LDDOUBLE

LOCAL 16
LDDOUBLE
DCMP
INT 0
EQ
JZ ifend

LOCAL -4
LDINT
STFVAL32
LEAVE
RET
ALIGN
LABEL ifend

ALIGN
LABEL forincr
LOCAL -4
LDINT
DUP32
INT 1
ADD
LOCAL -4
STINT
TRASH 4
JMP forcond
```

Exercício

```
static double v[] = { 1.41, 2.71, 3.14 };
int f(double v[], int s, double d) {
  for (int i = 0; i < s; i++)
    if (v[i] == d) return i;
  return -1;
}
int main() {
  return f(v, 3, 3.14);
}
```

```
DATA
ALIGN
LABEL v
SDOUBLE 1.41
SDOUBLE 2.71
SDOUBLE 3.14

TEXT
ALIGN
GLOBAL f, FUNC
LABEL f
ENTER 4
; i@-4 v@8 s@+12 d@+16

INT 0
LOCAL -4
STINT
ALIGN
LABEL forcond
LOCAL -4
LDINT
LOCAL 12
LDINT
LT
JZ forend

LOCAL +8
LDINT
LOCAL -4
LDINT
INT 8
MUL
ADD
LDDOUBLE

LOCAL 16
LDDOUBLE
DCMP
INT 0
EQ
JZ ifend

LOCAL -4
LDINT
STFVAL32
LEAVE
RET
ALIGN
LABEL ifend

ALIGN
LABEL forincr
LOCAL -4
LDINT
DUP32
INT 1
ADD
LOCAL -4
STINT
TRASH 4
JMP forcond

ALIGN
LABEL forend

INT -1
STFVAL32
LEAVE
RET
```

Exercício

```c
static double v[] = { 1.41, 2.71, 3.14 };
int f(double v[], int s, double d) {
  for (int i = 0; i < s; i++)
    if (v[i] == d) return i;
  return -1;
}
int main() {
  return f(v, 3, 3.14);
}
```

```
DATA
ALIGN
LABEL v
SDOUBLE 1.41
SDOUBLE 2.71
SDOUBLE 3.14


TEXT
ALIGN
GLOBAL f, FUNC
LABEL f
ENTER 4
; i@-4 v@8 s@+12 d@+16
```

```
INT 0
LOCAL -4
STINT
ALIGN
LABEL forcond
LOCAL -4
LDINT
LOCAL 12
LDINT
LT
JZ forend

LOCAL +8
LDINT
LOCAL -4
LDINT
INT 8
MUL
ADD
LDDOUBLE
```

```
LOCAL 16
LDDOUBLE
DCMP
INT 0
EQ
JZ ifend

LOCAL -4
LDINT
STFVAL32
LEAVE
RET
ALIGN
LABEL ifend
```

```
ALIGN
LABEL forincr
LOCAL -4
LDINT
DUP32
INT 1
ADD
LOCAL -4
STINT
TRASH 4
JMP forcond


ALIGN
LABEL forend

INT -1
STFVAL32
LEAVE
RET
```

```
TEXT
ALIGN
GLOBAL _main, FUNC
LABEL _main
ENTER 0

DOUBLE 3.14
INT 3
ADDR v

CALL f
TRASH 16

LDFVAL32

STFVAL32
LEAVE
RET
```

Questões?

Dúvidas?