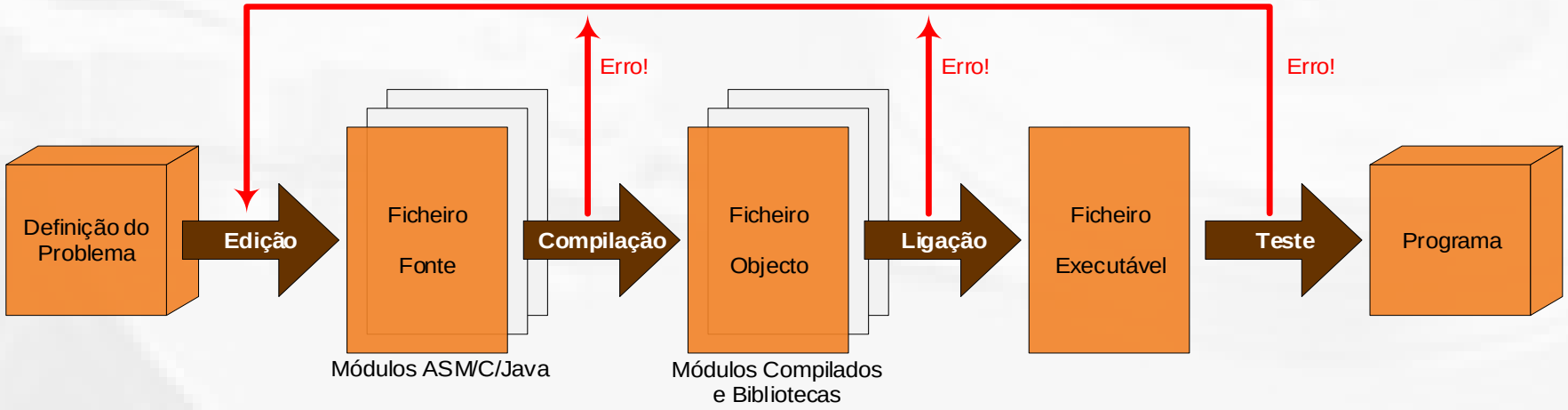
Desenvolvimento de Programas (em Linguagem *Assembly*) para o P16

Licenciatura em Engenharia Informática e de Computadores

1. Ciclo de Desenvolvimento de um Programa

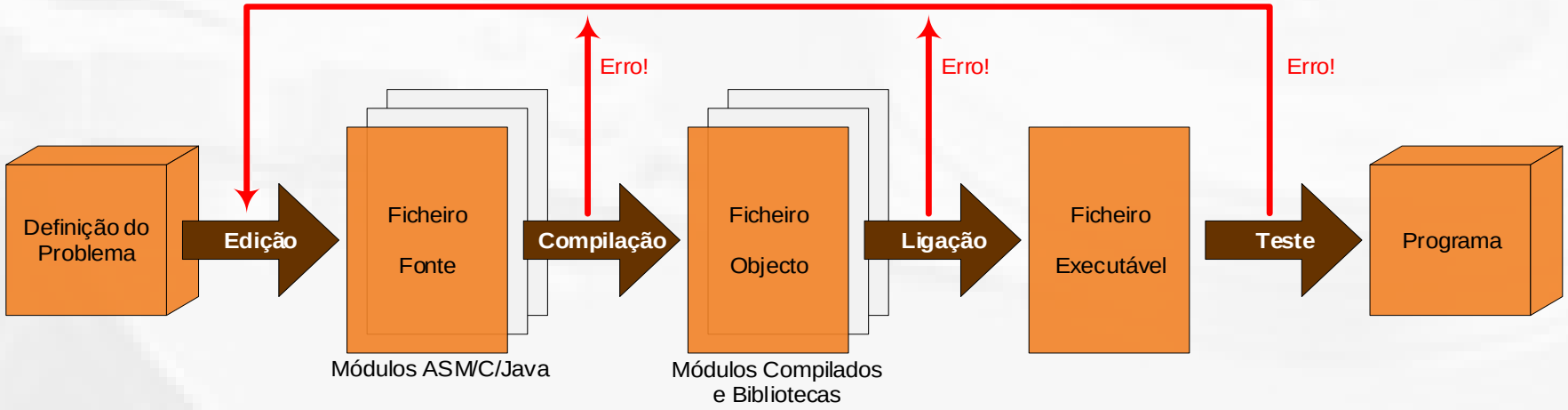
- Tradicionalmente, o ciclo de desenvolvimento de um programa/aplicação engloba quatro (4) fases distintas:



- *Editar o código fonte*
  Após a definição das especificações do problema e do(s) algoritmo(s) a usar para a sua solução, o(s) programador(es) começa(m) a codificar esse(s) algoritmos numa linguagem de programação em **ficheiros fonte**.

1. Ciclo de Desenvolvimento de um Programa

- Tradicionalmente, o ciclo de desenvolvimento de um programa/aplicação engloba quatro (4) fases distintas:

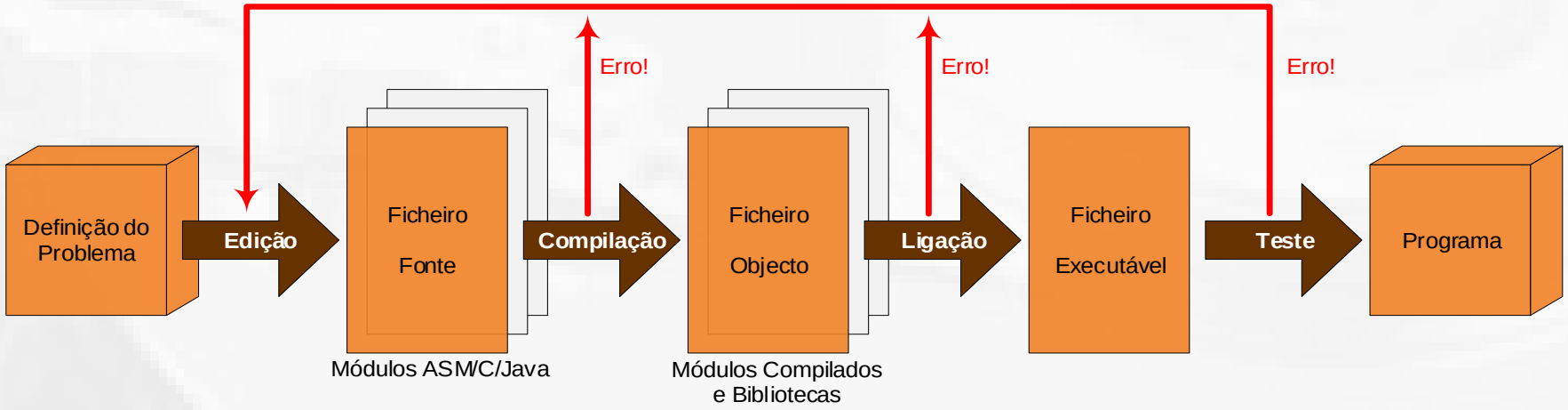


- *Compilar (assemblar) o programa*
  Através do uso de um compilador (*assembler*) são verificadas as regras sintáticas e semânticas da linguagem e é gerado um **ficheiro objeto** para cada ficheiro fonte do programa. Os ficheiros objeto contêm código máquina do CPU do sistema numa forma localizável.

III. Desenvolvimento de Programas para o P16

1. Ciclo de Desenvolvimento de um Programa

- Tradicionalmente, o ciclo de desenvolvimento de um programa/aplicação engloba quatro (4) fases distintas:

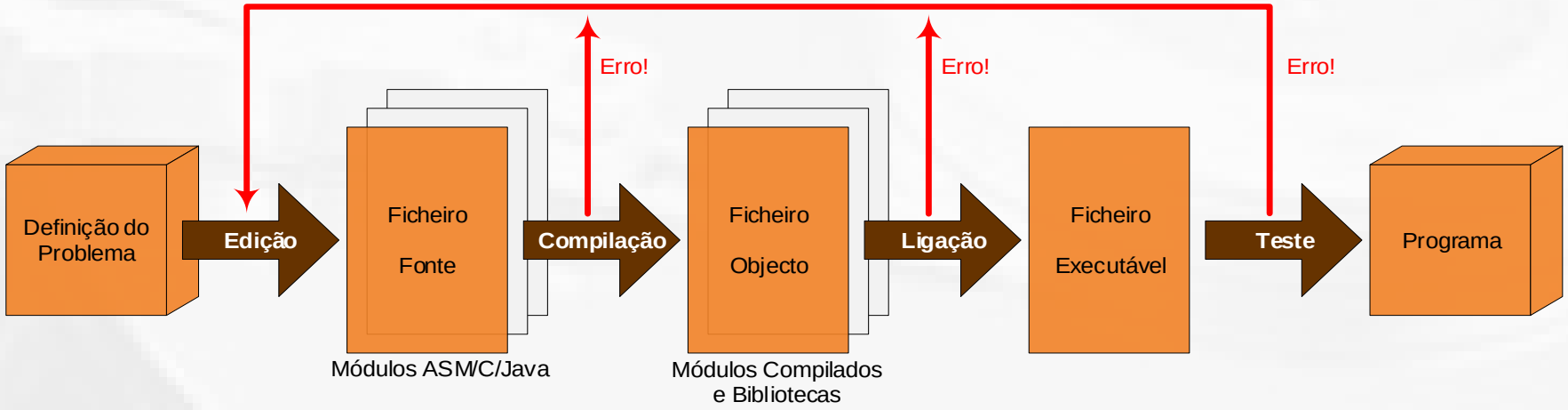


- *Ligar os ficheiros objeto*
  O *linker* faz a ligação entre os diversos ficheiros objetos que compõem o programa e as bibliotecas definidas na linguagem de programação adotada, procedendo ainda à realocação do código. É esta ferramenta que gera o **ficheiro executável**.

III. Desenvolvimento de Programas para o P16

1. Ciclo de Desenvolvimento de um Programa

- Tradicionalmente, o ciclo de desenvolvimento de um programa/aplicação engloba quatro (4) fases distintas:



- *Teste e depuração de erros*
  Recorre-se a um *debugger* e/ou um conjunto de vetores de teste para validar o correto funcionamento do programa contido no ficheiro executável.

2. Ficheiros Fonte

- São ficheiros que contêm um texto que implementa o algoritmo solução para um dado problema, sendo esse texto escrito numa linguagem de programação e obedecendo às regras semânticas e sintáticas dessa linguagem.
- Um programa é composto por um ou mais ficheiros fonte, denominados **módulos**, em que cada ficheiro contém uma parte do código do programa.
- Os ficheiros fonte que compõe um programa podem ser escritos na mesma, ou em diferentes, linguagens de programação.
- Os programas desenvolvidos no âmbito da disciplina são compostos por apenas um ficheiro fonte, escrito em linguagem *assembly* do P16.

2. Ficheiros Fonte

- Módulos escrito em linguagem *assembly* do P16
  - O código escrito em ficheiros fonte é representado por uma sequência de expressões que devem respeitar a seguinte sintaxe:

    `[label][instruction ][ comment]`

    - `label`

      Símbolo que referencia um endereço de memória, podendo ser composto por letras, dígitos e o carácter _, no início ou no meio. É seguido do carácter **:**

    - `instruction`

      Pode ser uma instrução *assembly* P16 ou uma diretiva do assembler.

    - `comment`

      Identifica os comentários ao código. O carácter **;** indica comentário até ao fim da linha, enquanto os caracteres **/*** e ***/** identificam um bloco de comentário, que pode ser multi-linha.

```
loop:
        ldr r1, [r0, 0]   ; Comentário de linha
        …
        b   loop
        …
```

2. Ficheiros Fonte

- Recomendações para escrita de módulos em *assembly* do P16
  - O programa deve ser escrito em letra minúscula, excetuando os identificadores de constantes que devem ser escritos em letra maiúscula.
  - Nos identificadores formados por várias palavras usa-se o caráter "_" (sublinhado) como separador.
  - O texto do programa deve ser disposto na forma de uma tabela com quatro colunas, separadas por caracteres TAB:
    - Na 1.ª coluna inserem-se apenas os símbolos (*labels*), se existirem;
    - Na 2.ª coluna inserem-se as mnemónicas das instruções ou diretivas.
    - Na 3.ª coluna inserem-se os parâmetros das instruções ou diretivas.
    - Na 4.ª coluna inserem-se os comentários até ao fim da linha.
  - Cada linha deve conter apenas um símbolo, instrução ou diretiva.
  - As linhas com símbolos não devem conter instruções ou diretivas, por forma a garantir-se uma tabulação alinhada e assim evitar-se separações na sequência de instruções.

3. Ficheiros Objeto

- Um ficheiro objeto contém o programa descrito num ficheiro fonte codificado na linguagem máquina do CPU do sistema alvo.
- Num ficheiro fonte todos os endereços referenciados no código são relativos ao próprio módulo e não ao programa global, pelo que código máquina gerado se diz **localizável**.
- Para além do código máquina correspondente às funções e procedimentos definidos pelo módulo, o ficheiro objeto inclui ainda informação necessária para ajudar na ligação de todos os módulos que compõem o programa, nomeadamente:
  - informação de realocação para as instruções e dados que referenciam endereços absolutos;
  - uma tabela das funções e procedimentos disponibilizados pelo módulo;
  - uma tabela de funções e procedimentos usados pelo módulo mas que deverão estar definidos noutro lado;
  - informação de *debugging* para facilitar a ligação ao ficheiro objeto correspondente.

4. Ficheiro Executável

- Um ficheiro executável contém o programa descrito na linguagem máquina do CPU do sistema alvo.
- Opcionalmente, também poderá conter informação para depuração de erros, o que só deverá acontecer aquando da fase de desenvolvimento do programa.

```
Sections
Index    Name              Addresses       Size
0        .text             0000 - 000B     000C 12

Symbols
Name                       Type       Value       Section
loop                       LABEL      0002 2      .text
var                        LABEL      000A 10     .text

Code listing
   1 0000 A060                    mov        r0, var
   2                    loop:
   3 0002 0100                    ldr        r1, [r0, 0]
   4 0004 91A0                    add        r1, r1, 1
   5 0006 0120                    str        r1, [r0, 0]
   6 0008 FC5B                    b          loop
   7                    var:
   8 000A 0B00                    .word 11
```

5. Organização do Código Compilado

- O código máquina gerado encontra-se agrupado em secções, ou seja, sequências de endereços de memória contínuos e sem espaços.
- Dentro de uma secção, todos os endereços são calculados em função do endereço base dessa secção, pelo que se diz que o código é **localizável**.
- Cada ficheiro executável para o P16 contém, tipicamente, várias secções que correspondem no programa ao:
  - Código (`.startup` e `.text`);
  - Dados iniciados (`.data`);
  - Dados não iniciados (`.bss`)
  - Pilha (`.stack`).
- A primeira secção do ficheiro deve ser a secção `.startup`, que começa no endereço 0 do ficheiro objeto, sendo seguida pelas secções `.text`, `.data`, `.bss` e `.stack` de uma forma contínua.

5. Organização do Código Compilado

- A secção `.startup`
  - Destina-se a preparar o ambiente de execução do programa antes que este comece a sua execução.
  - Para um sistema embebido, esta ação implica a realização de várias tarefas
    - Iniciação do sistema e dos periféricos indispensáveis ao seu funcionamento.
      - » Programação de vetores de exceção, iniciação de temporizadores ou controladores de interrupção, etc.
    - Iniciação das zonas de memória utilizadas pelo programa
      - » *Stack*, por iniciação do *stack pointer*
      - » Preenchimento com o valor 0 das posições de memória afetas a variáveis globais não iniciadas (secção `.bss`)

5. Organização do Código Compilado

- A secção `.startup`
  - Exemplo de conteúdo

```
          .section .startup
reset:
          b         _start
irq:
          b         .

_start:

          ; Iniciar o stack pointer
          ldr       sp, addr_stack_top

          ; Invocar a função main, sem argumentos (argc=0; argv=NULL)
          mov       r0, 0
          mov       r1, 0
          bl        main

          b         .           ; Bloquear a execução

addr_stack_top:
          .word     stack_top
```

5. Organização do Código Compilado

- A secção `.stack`
  - Destina-se a definir a zona de memória que será utilizada para *stack*.
  - *O endereço de início da zona de memória corresponde ao endereço em que a secção `.stack` será localizada.*
  - A dimensão do *stack*, em bytes, é estabelecida usando a diretiva `.space`.
  - *O símbolo `stack_top` identifica o extremo superior da zona de memória, que deverá ficar associado ao registo stack pointer (R13) pelo facto de a convenção prever um stack com filosofia full descendente.*

```
          .equ STACK_SIZE, 1024        ; Dimensão do stack (em bytes)

          ; Definir a zona de stack do programa
          .section .stack
stack_bottom:
          .space   STACK_SIZE
stack_top:
```

6. Ferramentas de Compilação e Depuração de Erros

- O *assembler* `PAS`
  - Aplicação que se destina a processar ficheiros escrito em *assembly* P16 e a gerar ficheiros executáveis para o processador P16.
  - É um assembler didático que, por essa razão, processa apenas um ficheiro fonte e localiza logo o programa. Este processamento é feito numa única passagem de leitura sobre o ficheiro fonte.
  - O processamento também envolve a verificação sintática e semântica do código escrito nos ficheiros fonte, sendo geradas mensagens de aviso e de erro quando essas regras não são cumpridas.
  - O *assembler* `PAS` dispõe de um pré-processador que, apesar de não ser tão avançado como os pré-processadores de C/Java, permite o tratamento de expressões e de pseudo-instruções.
  - Para além do ficheiro executável com o código máquina, o *assembler* `PAS` também gera um ficheiro de texto com informação sobre o processo de compilação.
  - Apesar de poder gerar ficheiros executáveis com diferentes formatos, o formato que será usado na produção de programas para o processador P16 é o Intel Hex.

6. Ferramentas de Compilação e Depuração de Erros

- O *assembler* `PAS`
  - *Expressões*
    - Uma expressão pode aparecer em qualquer instrução onde se espere uma constante e produz um valor absoluto operando valores numéricos ou símbolos com valor numérico.
    - Na sintaxe ARM as constantes são precedidas por # e podem ser declaradas de diversas formas:
      * Decimal (*Exemplo*: 34)
      * Hexadecimal (*Exemplo*: **0x**22)
      * Octal (*Exemplo*: **o**42)
      * Binário (*Exemplo*: **0b**100010)
      * Caracteres entre ‘ ’ (*Exemplo*: **‘**K**’**)
    - As expressões podem ainda produzir valores com recurso a operadores:
      * Unários: - e ~
      * Binários: +, -, *, /, %, &, |, << e >>

      Exemplos: #’a’-’A’ #2*4+3 #2<<3+1

6. Ferramentas de Compilação e Depuração de Erros

- O *assembler* `PAS`
  - *Diretivas mais utilizadas*

.align n	Insere bytes a zero até um endereço múltiplo de 2^n.
.space size[, fill]	Insere `size` bytes com o valor `fill`.
.ascii “string”	Insere os caracteres que compõem a string.
.asciz “string”	Insere os caracteres que compõem a string com terminação \0.
.byte expression	Insere o valor especificado representado a 8 bits (1 byte).
.word expression	Insere o valor especificado representado a 16 bits (2 bytes).
.equ nome, valor	Atribui `valor` ao símbolo `nome`.
.section name	Define uma secção com o nome `name`.
.text	Define uma secção especial, destinada às instruções.
.data	Define uma secção especial, destinada às variáveis globais

6. Ferramentas de Compilação e Depuração de Erros

- O *assembler* `PAS`
  - *Linha de comandos:*

```
pas [-h] [-v] [-s sectionname=address] [-o exefile] srcfile
```

`-h`	Imprime uma mensagem com as opções disponíveis
`-v`	Imprime a versão da aplicação
`-s`	Define que a secção `sectioname` será localizada no endereço `address`
`-o`	Definir o nome do ficheiro de saída (por omissão é srcfile.hex)
`srcfile`	Ficheiro fonte do programa

- A ordem pela qual os ficheiros e as opções são especificados é aleatória.
- A especificação das opções é case *sensitive*.

6. Ferramentas de Compilação e Depuração de Erros

- O *assembler* `PAS`
  - *Exemplo de ficheiro fonte processado pelo* `PAS`

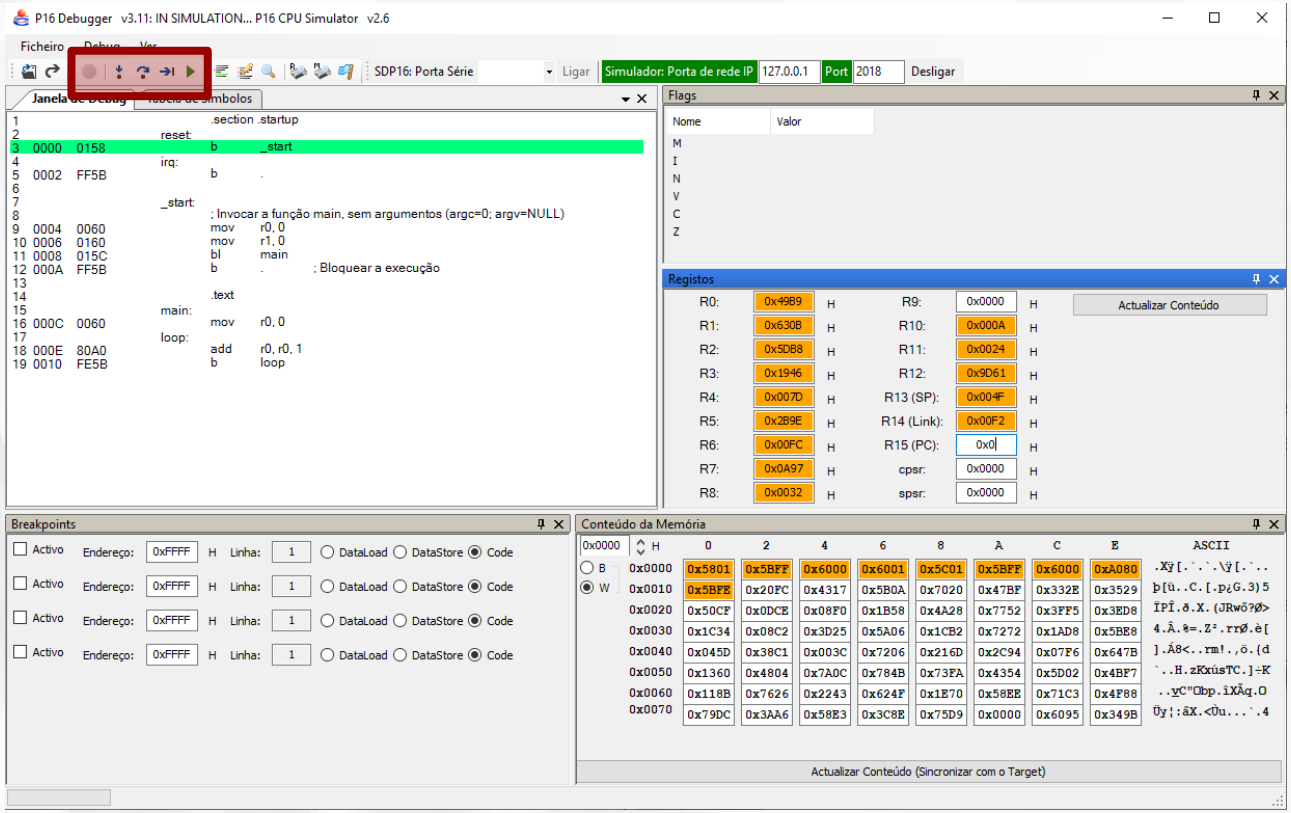
```
            .section .startup
reset:
            b           _start
irq:
            b           .

_start:
            ; Invocar a função main, sem argumentos (argc=0; argv=NULL)
            mov         r0, 0
            mov         r1, 0
            bl          main
            b           .           ; Bloquear a execução

            .text
main:
            mov         r0, 0
loop:
            add         r0, r0, 1
            b           loop
```

  - *Exemplo de invocação de* `pas`*:*

```
pas -s .startup=0 prog.s
```

6. Ferramentas de Compilação e Depuração de Erros

- O *debugger* P16Debugger
  - Aplicação para Microsoft Windows (versão 64 bits) que permite fazer a depuração de erros em programas escritos para o processador P16.



6. Ferramentas de Compilação e Depuração de Erros

- O *debugger* P16Debugger
  - Para além de permitir interação com o simulador P16Simulator, esta aplicação também é capaz de interagir com a plataforma SDP16.