Arquitetura de Computadores

Diretivas do compilador: constantes, dados do programa iniciados e não iniciados, secções

João Pedro Patriarca (jpatri@cc.isel.ipl.pt), Gabinete F.0.23 do edifício F

ISEL, ADEETC, LEIC

Sumário

- Definição de constantes

- Definição de dados do programa
  - Sem iniciação
  - Com iniciação
  - Alinhamento

- Organização de um programa em secções

Definição de constantes

- Diretiva `.equ <symbol>,<value>`
  - Associa um símbolo a um valor
  - Não envolve alocação de memória

```
#define   INT16_MIN   0x8000
…
int16_t val = INT16_MIN;
```

```
   .equ INT16_MIN, 0x8000
   …
   ; opção 1
   mov  r0, #INT16_MIN & 0xFF
   movt r0, #INT16_MIN >> 8
   …
   ; opção 2
   ldr  r0, int16_min_val
   …
int16_min_val:
   .word  INT16_MIN
```

Definição de dados do programa
Sem iniciação

- Diretiva `.space <len>[, <val>]`
  - Aloca `len` *bytes* iniciados com o valor 0 ou cada *byte* com o valor `<val>`

```
uint8_t a[20];


int16_t b[20];


uint32_t c;
```

```
a:  .space 20


b:  .space 40


c:  .space 4


d:  .space 10, 2 ; aloca 10 bytes e
cada byte tem o valor 2
```

Definição de dados do programa
Com iniciação

- Diretiva `.byte <val>[, <val>]*`
  - Define 1 ou mais *bytes* em memória expressos na base 10, 16 e caractere
  - Cada *byte* ocupa 8 bits

- Diretiva `.word <val>[, <val>]*`
  - Define 1 ou mais *words* em memória expressos na base 10, 16 e caractere
  - Cada *word* ocupa 16 bits

- Diretiva `.ascii “<str>”[, “<str>”]*`
  - Define sequência com 1 ou mais caracteres em memória
  - Cada caractere ocupa 8 bits

- Diretiva `.asciz “<str>”[, “<str>”]*`
  - Define sequência com 1 ou mais caracteres em memória
  - A última sequência termina com o valor 0
  - Cada caractere ocupa 8 bits

Definição de dados do programa
Com iniciação::Exemplos

```
uint8_t a = 10;

int16_t b = -10;

uint32_t c = 0x12345678

int8_t d[] = {0, -1, ‘0’, 0xAA};

int16_t e[] = {0, -1, ‘0’, 0xAAFF};

char f = ‘a’;

char g[] = “xpto”;
```

```
a:  .byte 10

b:  .word -10

c:  .word 0x5678, 0x1234

d:  .byte 0, -1, ‘0’, 0xAA

e:  .word 0, -1, ‘0’, 0xAAFF

f:  .ascii ‘a’

g:  .asciz “xpto”
```

Definição de dados do programa
Alinhamento

- Diretiva `.align <n>`
  - Avança contador de localização até um valor múltiplo de 2^n, ou seja, o novo valor terá zero nos 'n' *bits* de menor peso.
  - O compilador do P16 aceita apenas n=0|1
  - Garante o alinhamento de *words* e de instruções a seguir à definição de um número ímpar de *bytes*

```
int8_t a = 3;
uint16_t b = 0x34;
```

```
0108 05    a:   .byte 5
0109 00         .align 1
010A 34 00 b:   .word 0x34
```

Organização de um programa em secções

- As secções são lógicas e permitem a organização de um programa, nomeadamente, distinguir claramente código do programa de dados do programa e da estrutura *stack*
  - Permite ainda estabelecer diferentes privilégios no acesso a cada secção (não considerado no P16)

- Secções suportadas pelo compilador P16
  - Secção `.text`: inclui programa aplicacional e constantes
  - Secção `.data`: inclui dados do programa iniciados (variáveis globais iniciadas)

- O compilador suporta a definição de novas secções através da diretiva `.section <name>`

- O compilador suporta ainda fragmentos dispersos das mesmas secções; após compilação junta todos os fragmentos da mesma secção no mesmo bloco

- O compilador não suporta um programa constituído por vários ficheiros com código fonte, ou seja, um programa corresponde sempre a um único ficheiro

Organização de um programa em secções
Outras secções típicas

- Nomes de outras secções típicas num programa

Secção	Descrição
`.section .startup`	Código de arranque (*bootstrap*)
`.section .bss`	Dados do programa (variáveis globais não iniciadas – no arranque do programa têm o valor 0)
`.section .stack`	Estrutura de dados do tipo *stack*

Organização de um programa em secções

Secção `.startup`

- Deve ficar localizada no endereço 0
  - Define o ponto de entrada (*entry point*) após *reset*
  - Define o ponto de entrada (*entry point*) do vetor de interrupção
- Inicia registo *Stack Pointer* (SP)
- Inicia secção `.bss` com zeros (desnecessário se na definição dos dados for garantida a iniciação com zeros)
- Produz chamada para o programa principal (`main`)
  - Tratando-se, tipicamente, de uma chamada para secção diferente, a chamada não deve depender da distância da definição da função `main`

Organização de um programa em secções

Secção `.startup` :: exemplo

```
   .section .startup
   b     _start  ; reset entry
   b     .       ; irq entry
_start:
   ldr   sp, addr_stack_top
   mov   r0, pc
   add   lr, r0, #4
   ldr   pc, addr_main
   b     .
addr_main:
   .word main
addr_stack_top:
   .word stack_top
```

```
   .text
main:
   push  lr
   …
   pop   pc

   .equ STACK_SIZE 128
   .section .stack
   .space STACK_SIZE
stack_top:
```

Organização de um programa em secções
Secção `.bss`

- O espaço em memória para cada variável é alocado com a diretiva `.space <n>`
- A diretiva `.space` inicia o espaço alocado com 0s, por omissão

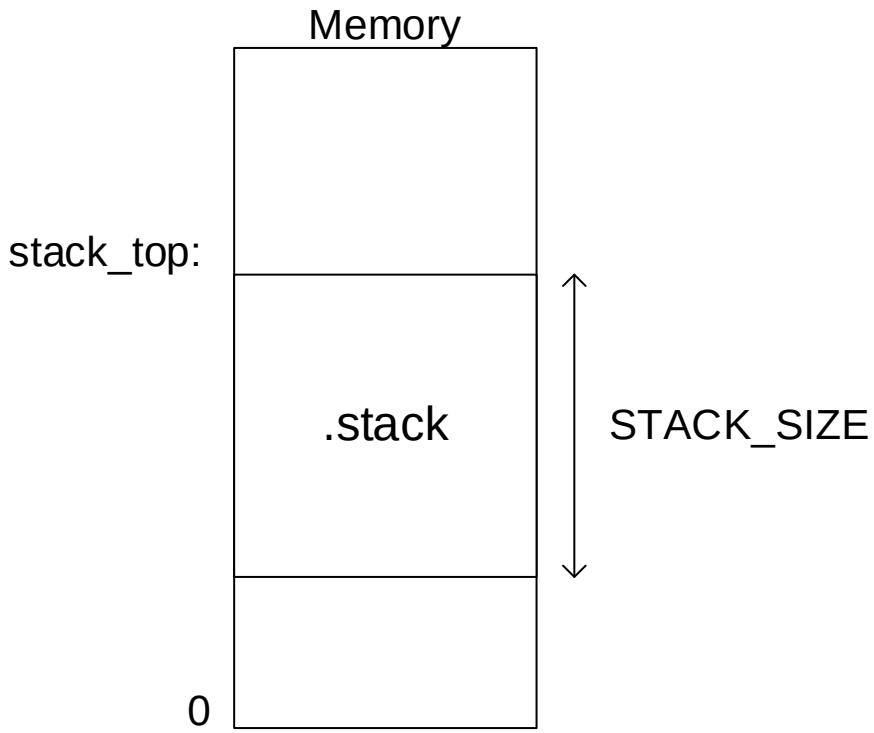
```
// Variáveis globais não iniciadas
uint8_t a;
int16_t b;
char c;
int32_t d;
char e[10];
uint16_t f[4];
```

```
                .section .bss
2000 00     a:  .space 1
2001 00         .align 1
2002 00 00  b:  .space 2
2004 00     c:  .space 1
2005 00         .align 1
2006 00 00  d:  .space 4
     00 00
200A 00 00  e:  .space 10
     …
2013 00
                .align 1
2014 00 00  f:  .space 8
     …
201B 00
```

Organização de um programa em secções
Secção `.stack`

```
    .equ STACK_SIZE 1024

    .section .stack
    .space STACK_SIZE
stack_top:
```



Acesso a dados do programa

- Acesso a dados definidos em outras secções não deve depender da distância entre a instrução que realiza o acesso e a respetiva definição do dado
- A distância da instrução `LDR rd, imm6` ao endereço da variável (`addr_xxx`) não pode ultrapassar 128 *bytes* (64 *words*)

```
   .text
   ldr   r0, addr_var1
   ldr   r0, [r0]
   …
   ldr   r0, addr_var2
   ldrb  r0, [r0]
   …
```

```
   …
   ldr   r0, addr_var3
   strb  r1, [r0]
   …
addr_var1: .word   var1
addr_var2: .word   var2
addr_var3: .word   var3
```

```
   .data
var1:
   .word 0x1234
var2:
   .byte 0x5E

   .bss
var3:
   .space 1
```

Posições relativas das secções no espaço de endereçamento do P16

- As posições relativas sugeridas para o P16 seguem o modelo usado pelo compilador da GNU para ARM
- As posições das secções são definidas pela ordem que aparecem no código fonte
  - Este comportamento por omissão pode ser alterado através de opções do compilador

```
.section .startup
…
.text
…
.data
…
.section .bss
…
.section .stack
…
```