Ficha 5 – Manipulação ao nível do bit

Tópicos abordados:
– Operações ao nível do bit

Duração prevista: 1 aula

1. Base binária, octal e hexadecimal

1.1. Base binária

A base binária faz uso de dois símbolos: 0 e 1. Um valor binário é composto por um conjunto de 0s e 1s. Na linguagem C não existe nenhuma forma nativa de representar um valor em formato binário.

Considerando um valor binário composto por $n+1$ bits, em que b_0 representa o bit menos significativo e b_n o bit mais significativo, tem-se:

$b_n \ldots b_i \ldots b_1 b_0$

A conversão de binário para decimal opera-se da seguinte forma:

$$valor\_em\_decimal = b_n.2^n + \ldots + b_i.2^i + \ldots + b_1.2^1 + b_0.2^0$$

Lab 1

Converta o valor binário `1010.0111` para decimal.

1.2. Base octal

A base octal faz uso de 8 símbolos: `0, 1, 2, 3, 4, 5, 6 e 7`, sendo, portanto, também conhecida como a base 8. Na linguagem C, um número inteiro é interpretado em base octal se tiver um zero à esquerda.

Por exemplo:

```
int a = 0124;    /* número 124 na base octal */
```

Considerando um valor em base octal composto por n+1 dígitos, em que d_0 representa o dígito menos significativo e d_n o dígito mais significativo, tem-se:

$d_n \ldots d_i \ldots d_1 d_0$

A conversão de octal para decimal opera-se da seguinte forma:

$$valor\_em\_decimal = d_n . 8^n + \ldots + d_i . 8^i + \ldots + d_1 . 8^1 + d_0 . 8^0$$

Lab 2

Qual o valor decimal da variável "valor_octal"?

```
int valor_octal = 01236;
```

Umas das vantagens da base octal é a facilidade de conversão para a base binária e vice-versa. A conversão da base octal para a base binária processa-se simplesmente convertendo cada dígito octal para a respetiva representação com 3 bits. Por exemplo, o valor octal 0234 é convertido para 010.011.100.

A operação inversa – conversão de binário para octal – processa-se substituindo grupos de 3 bits pelo respetivo valor octal. Os grupos de 3 bits são formados da direita para a esquerda, isto é, do bit menos significativo para o bit mais significativo.

Lab 3

a) Converta o valor octal 0766 para binário.

b) Converta o valor binário 011.110.101 para octal.

1.3. Base hexadecimal

A base hexadecimal faz uso de 16 símbolos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F, sendo, portanto, também conhecida como a base 16. Na linguagem C um número inteiro é interpretado em base hexadecimal se tiver o prefixo 0x.

Por exemplo:

```
int a = 0xAF;    /* número AF na base hexadecimal */
```

Considerando um valor em base hexadecimal composto por n+1 dígitos, em que d_0 representa o dígito menos significativo e d_n o dígito mais significativo, tem-se:

$d_n \ldots d_i \ldots d_1 d_0$

A conversão de hexadecimal para decimal opera-se da seguinte forma:

$$valor\_em\_decimal = d_n . 16^n + \ldots + d_i . 16^i + \ldots + d_1 . 16^1 + d_0 . 16^0$$

A conversão de hexadecimal para binário é similar à conversão de octal para binário. Contudo, dado que a base hexadecimal usa 16 símbolos, cada símbolo hexadecimal é representado por 4 bits. Por exemplo, o símbolo A é representado por `1010`.

A conversão de binário para hexadecimal também é trivial, agrupando-se os bits em grupos de 4, do bit menos significativo (à direita) para o mais significativo (à esquerda). Cada grupo de 4 bits é depois substituído pelo símbolo hexadecimal correspondente. Por exemplo, `1111` é substituído por `F`.

Lab 4

a) Converta o valor hexadecimal `0x912` para binário.

b) Converta o valor binário `111.0010.0001.0110` para hexadecimal.

2. Operadores para manipulação ao nível do bit

A linguagem C tem um conjunto de operadores que permitem a manipulação de valores ao nível do bit. No entanto, uma vez que na linguagem C não existe um tipo de dados que permita representar apenas um bit, estes operadores têm de ser aplicados a valores de pelo menos um byte.

2.1. Operador de negação (not)

Símbolo na linguagem C: ~ (*tilde*)

Descrição: o operador unário de negação devolve a negação do valor que lhe é passado. Assim, bits a 1 são colocados a 0 e bits a 0 são colocados a 1.

Lab 5

Considerando o código seguinte, qual é o valor de `B`?

```
short A = 0x1122;
short B = ~A;
```

2.2. Operador AND binário

Símbolo na linguagem C: &

Descrição: operador que requer dois operandos (`a & b`). O operador efetua a operação de AND em cada um dos bits correspondentes aos operandos `a` e `b`.

A tabela de verdade do AND binário é:

AND	0	1
0	0	0
1	0	1

Lab 6

Considerando o código seguinte, qual é o valor de `C`?

```
short A = 0x1122;
short B = 0x2211;
C = A & B;
```

2.3. Operador OR binário

Símbolo na linguagem C: |

Descrição: operador que requer dois operandos (`a | b`). O operador efetua a operação de OR em cada um dos bits correspondentes aos operandos `a` e `b`.

A tabela de verdade do OR binário é:

OR	0	1
0	0	1
1	1	1

Lab 7

Considerando o código seguinte, qual é o valor de `C`?

```
short A = 0x1122;
short B = 0x2211;
C = A | B;
```

2.4. Operador XOR binário

Símbolo na linguagem C: ^

Descrição: operador que requer dois operandos (`a ^ b`). O operador efetua a operação de XOR (eXclusive OR) em cada um dos bits correspondentes aos operandos `a` e `b`.

A tabela de verdade do XOR binário é:

XOR	0	1
0	0	1
1	1	0

Lab 8

Considerando o código seguinte, qual é o valor de `C`?

```
short A = 0x1122;
short B = 0x2211;
short C = A ^ B;
```

Lab 9

Considerando o código seguinte, qual é o valor de `B`?

```
short A = 0x1122;
short B = A ^ A;
```

Lab 10

Considerando o código seguinte, qual é o valor de `B`?

```
short A = 0x1122;
short B = A ^ (~A);
```

2.5. Operador deslocamento à esquerda (<<)

Na linguagem C, o operador deslocamento para a esquerda tem a seguinte sintaxe:

```
valor << n
```

O operador deslocamento à esquerda efetua uma translação em *n* posições dos bits para a esquerda do valor especificado. A Figura 1 ilustra uma operação de deslocamento para a esquerda em um bit do valor 0001.0010, resultando no valor 0010.0100. Note-se que, devido ao deslocamento à esquerda em 1 bit, o anterior bit mais significativo (bit mais à esquerda) é perdido, sendo acrescentado um bit a 0 para a posição do bit menos significativo (bit mais à direita, representado a azul na Figura 1). Caso o deslocamento fosse de *n* bits para a esquerda, perder-se-iam os *n* bits mais significativos, sendo ainda acrescentados *n* bits a 0 como bits menos significativos.

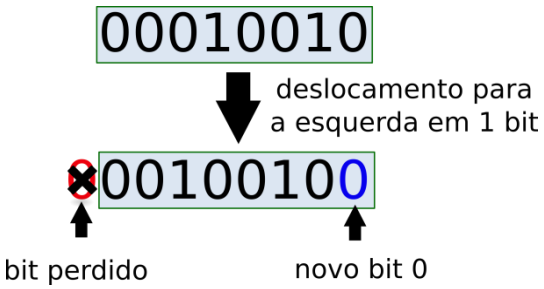

Figura 1: Exemplo de uma operação deslocamento para a esquerda em 1 bit

Quando realizada sobre a representação binária de um número inteiro, a operação deslocamento para a esquerda em *n* bits produz um resultado que corresponde à multiplicação por 2^n do valor inteiro original. Por exemplo, na Figura 1, o deslocamento em um bit para a esquerda do valor original 0001.0010, que corresponde ao inteiro 18 em base decimal, é transformado no valor 0010.0100, que corresponde ao valor 36 em base decimal, isto é, ao dobro do valor original. Esta propriedade do operador deslocamento para a esquerda é frequentemente empregue, especialmente em linguagens *assembler*, para realizar multiplicações de valores inteiros por 2^n, pois é bastante mais rápido do que o algoritmo de multiplicação entre dois números inteiros.

```
#include <stdio.h>
int main(void){
    unsigned int valor = 1;
    unsigned int valor_shift;
    size_t size_bits = sizeof(valor) * 8;
    unsigned int i;
    for(i = 0; i < size_bits; i++){
      valor_shift = valor << i;
      printf("[shift (valor << %02u)]%u\n", i, valor_shift);
    }
  return 0;
}
```

Listagem 1 – Operações deslocamento para a esquerda

Lab 11

O que faz o código apresentado na listagem 1? Execute-o e comente os resultados.

2.5.1 Máscara binária

A um padrão de bits empregue com um determinado fim dá-se o nome de máscara binária. Por exemplo, uma máscara que toma conhecimento do 3º bit menos significativo de um valor inteiro é formada por zeros, exceto no 3º bit menos significativo, que é 1.

A representação binária da máscara é:

`0…0100`

A máscara pode ser empregue numa operação de AND binário para determinar o valor do 3º bit menos significativo de uma variável A, tal como é mostrado no fragmento de código seguinte:

```
int A;
int mask = 1 << 2;    /* Cria padrão de bits 0…0100 */
(…)
int is_bit_3_lsb_set = (A & mask);
```

Listagem 2 – Criação e verificação de uma máscara binária

Lab 12

Indique o código C que permite a criação das seguintes máscaras binárias, que devem ter 32 bits cada:

a) `0...01`

b) `0...10`

c) `0...10101`

2.6. Operador deslocamento à direita (>>)

O operador deslocamento para a direita funciona de forma similar ao operador de deslocamento para a esquerda, alterando-se somente o sentido do deslocamento. Assim, na operação de deslocamento para a direita em *n* bits, há lugar à deslocação em *n* posições dos bits para a direita. A Figura 2 ilustra uma operação de deslocamento para a direita. Na linguagem C, o operador deslocamento para a direita é representado por >> e, à semelhança do operador deslocamento para a esquerda, requer dois operandos. Do lado esquerdo do operador fica o operando cujo valor irá ser alvo da operação de deslocamento para a direita. Por sua vez, o operando do lado direito indica de quantos bits deve o valor inicial ser deslocado.

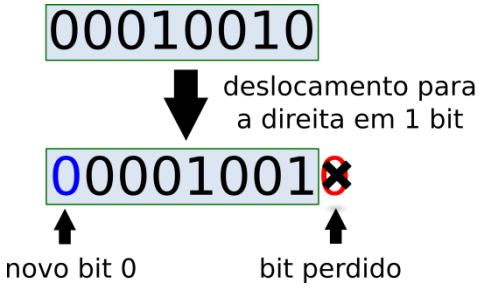

Figura 2: Exemplo de uma operação deslocamento para a direita em 1 bit

2.6.1 Limitação do operador deslocamento à direita (>>)

Na linguagem C, uma limitação do operador >> envolve o bit mais à esquerda (bit mais significativo) que deve ser acrescentado pelo operador quando ocorre um deslocamento para a direita. De facto, no caso de um valor inteiro **com sinal** representado em complementos de dois, o bit mais à esquerda (bit mais significativo) corresponde ao sinal do número: 0 indica um número positivo e 1 indica um número negativo. Assim, a operação deslocamento para a direita não pode simplesmente acrescentar um bit zero em lugar do bit mais significativo, pois tal poderá resultar num valor com sinal diferente do valor inicial. No caso da linguagem C, não está definido qual o bit a ser inserido como bit mais significativo pelo operador deslocamento à direita quando lida com inteiros com sinal. Deste modo, o comportamento fica dependente do compilador empregue.

```
#include <stdio.h>

int main(void){
  int positive = 998;
  int negative = -998;
  int positive_shift, negative_shift;
  int i;
  for(i = 0; i < 4; i++){
    printf("===[shift right %d]===\n", i);
    positive_shift = positive >> i;
    negative_shift = negative >> i;
    printf("positive_shift = %d\n", positive_shift);
    printf("negative_shift = %d\n", negative_shift);
  }
  return 0;
}
```

Listagem 3 – Operação deslocamento à direita em inteiros positivos e negativos

Lab 13

Execute o código da listagem 3 e indique qual é o bit (0 ou 1) que é inserido quando se efetua uma operação de deslocamento à direita. Justifique.

3. Exercícios

3.1. Para a aula

1. Elabore a função `bin_to_decimal`, que deve ter o seguinte protótipo:

```
int bin_to_decimal(const char *bin_s_ptr);
```

A função recebe uma *string* com `0`s e `1`s que representa um valor binário, devolvendo o valor inteiro corresponde ao número binário expresso pela *string*.

2. Elabore a função `is_bit_n_set`, que deve ter o seguinte protótipo:

```
int is_bit_n_set(int input, size_t bit_n);
```

A função deve devolver `1` se o valor indicado pelo parâmetro `input` tiver o `bit_n-`*`ésimo`* bit a `1`, e `0` caso contrário. O parâmetro `bit_n` indica o bit cujo valor deve ser determinado, sendo que `bit_n` com o valor a `0` corresponde ao bit menos significativo, `bit_n` com o valor a `1` corresponde ao 2º bit menos significativo e assim sucessivamente.

Sugestão: uso de máscara binária combinada com o operador AND.

3. Elabore a função `bit_n_to_zero`, que deve ter o seguinte protótipo:

```
int bit_n_to_zero(int input, size_t bit_to_zero);
```

A função deve devolver um inteiro correspondente ao valor de `input` com o `bit_to_zero-`*`ésimo`* bit a `0`, mantendo os restantes bits de `input` inalterados. Um valor de zero para o parâmetro `bit_to_zero` indica que se pretende "zerar" o bit menos significativo (bit `0`). Por exemplo, a chamada da função abaixo mostrada pretende zerar o 4º bit menos significativo do conteúdo guardado na variável `valor`.

```
int valor = …
int valor_com_bit3_zero = bit_n_to_zero(valor, 3);
```

Sugestão: uso de máscara binária combinada com o operador AND.

3.2. Extra-aula

4. Elabore a função `count_ones`, que deve ter o seguinte protótipo:

```
int count_ones(int input);
```

A função deve devolver o número de bits com o valor `1` da variável `input`. Por exemplo, a chamada da função abaixo mostrada deve devolver o valor 6.

```
int ones = count_ones(0x512A);
```

Sugestão: uso de máscara binária combinada com o operador AND e o operador de deslocamento à direita.

5. Como sabe, o protótipo da função `open` é:

```
int open(const char *pathname, int flags, int mode_t mode);
```

Como o nome indica, o parâmetro `flags` é empregue para passar *flags*, isto é, uma variável inteira em que cada bit está associado a uma determinada propriedade. Por exemplo, a *flag* `O_APPEND` especifica que a abertura do ficheiro deve ser feita por forma a colocar o ficheiro em modo de acréscimo.

a) Através da página do manual eletrónico da função `open` identifique as constantes que podem ser empregues para especificar o parâmetro `flags`. Deve ainda elaborar um programa onde é mostrado – em decimal, hexadecimal e em binário – o valor de cada constante.

b) Elabore a função `descodifica_flags`, com o seguinte protótipo:

```
void descodifica_flags(int flags)
```

A função deve receber um valor que representa *flags* similares às que podem ser passadas à função `open` e identificar na saída padrão as *flags* que estão ativas no valor passado pelo parâmetro `flags`.

Por exemplo:

```
O_APPEND: **ativo**
O_CREAT: ---
O_DIRECTORY: ---
```

6. (pergunta adaptada do 2º teste prático 2017-18)

Pretende-se que implemente, recorrendo à linguagem C, uma aplicação que permita efetuar sequências de lançamentos de moedas. A aplicação **head_tail** deve gerar números aletórios entre 0 e 1 de forma a simular o lançamento de uma moeda, resultando em Tail (Coroa) ou Head (Cara), respetivamente.Esta recebe os seguintes argumentos de entrada:

- -s, --seed <INT>: número inteiro empregue para inicializar o gerador de números aleatórios **[opção não obrigatória]**
- -f, --flips <INT>: número inteiro com o número de lançamentos de moeda **[opção obrigatória]**

A opção "-s/--seed" é empregue para inicializar o gerador de números aleatórios com um número inteiro, que deve estar entre 0 e 216-1. Caso a opção "-s/--seed" não seja especificada (é uma opção não obrigatória), o gerador de números aleatórios deve ser inicializado com o valor 1. A opção "-f/--flips" representa o número de lançamentos a efetuar e que deve estar entre 0 e 255. Todos os parâmetros passados à aplicação devem ser validados.

Esta aplicação deverá utilizar um **inteiro sem sinal de 32 bits** para representar os resultados obtidos nos lançamentos de forma ordenada. Cada um dos bits deste inteiro corresponde a um lançamento de moeda, sendo que o valor 1 representa Head e o valor 0 Tail. Concretamente, o bit menos significativo codifica o resultado do primeiro lançamento, o segundo bit menos significativo codifica o resultado do segundo lançamento e assim sucessivamente. Esta aplicação não deverá realizar sequências de lançamentos inferiores a 1 ou superiores a 16. Caso seja solicitada uma sequência fora destes limites, a aplicação deverá apresentar uma apropriada mensagem de erro no canal de erro padrão.

No final, esta deverá mostrar por ordem os resultados de cada um dos lançamentos simulados na saída padrão. Em caso de erro, deverá ser apresentada uma mensagem de erro apropriada no canal de erro padrão.

Considere o seguinte exemplo que ilustra o funcionamento da aplicação.

```
$ ./head_tail -s 1 -f 3
Requesting 3 coin flips...
Results:
[01] Head
[02] Tail
[03] Head
```

4. Bibliografia

[1] Patrício Domingues. “Manipulação ao nível do bit na Linguagem C”. Revista Programar, Número 50, pp. 26-35, Setembro 2015, ISSN 1647 0710. http://bit.ly/2dmD74H

[2] Steve Oualline, “Practical C Programming – Chapter 11: Bit Operations”, O'Reilly Media, Inc.", 1997, ISBN: 1-56592-306-5

[3] Patrício Domingues, Apontamentos das aulas teórico-práticas de Programação Avançada – Capítulo “Bits & Bytes”.