Arquitetura de Computadores 2022/2023

Ficha 5

Tópicos: *Representação de dados, tipos e dimensões. Representação de reais (IEEE 754).*

Exercícios sobre bases numéricas e representação de dados

Parte I - Números inteiros

1. Considere a representação binária dos valores presentes na memória de um computador com palavras de 7 bits. Complete a tabela interpretando esses bits como:

- **números sem sinal**, apresentando o valor em base 10 (decimal) e em base 16 (hexadecimal);
- **caracteres** na norma ASCII;
- **inteiros com sinal** em complemento para 2.

binário	s/sinal dec.	s/sinal hex	car. ASCII	c/sinal Dec.
010 1010				
100 0110				
100 0011				
101 0100				
000 1010				
010 0001				
000 1000				

2. Apresente a representação binária (em complemento para dois) dos valores decimais apresentados na tabela.

decimal	5 bit	8 bit	12 bit
5			
-3			
-16			
-1			
16			
260			

3. Para o caso da representação de números em palavras de 7 bits indique:

a) Quantos valores diferentes se podem representar em cada palavra, para números em binário, em hexadecimal e em decimal, com ou sem sinal?

b) Quantos caracteres diferentes serão no máximo possíveis representar?

c) Se a palavra representar um endereço de memória, quantos endereços diferentes serão, no máximo, possíveis representar?

d) Quais são o maior e o menor números que se podem representar sem sinal (dê a resposta em base 10 e em base 2)?

e) Quais são o maior e o menor números representáveis com sinal em complemento para 2 (dê a resposta em base 10 e em base 2)?

4. a) Complete o programa *exercicio4.c* para imprimir o tamanho em bytes de cada um dos tipos seguintes: `char`, `short`, `int`, `unsigned int`, `long`, `unsigned long`, `long long`, `float`, `double`. Para isso, utilize:

- "`sizeof (T)`", operador que devolve o tamanho ocupado por *T* em bytes (o valor devolvido é um `size_t` que está definido como `unsigned int` ou `unsigned long`, dependendo da arquitectura) †;
- "`printf("tamanho %zu\n", N)`", função que imprime "*tamanho N*" mudando de linha de seguida, onde *N* é o valor que será afixado em base dez.

Nota: ao contrário do Java, alguns dos tipos em C dependem da arquitetura onde o programa é executado (32 bits vs. 64 bits).

b) Complete o programa anterior para escrever também os valores limite dos tipos dados, indicados pelas constantes seguintes definidas em `limits.h`: `SHRT_MIN`, `SHRT_MAX`, `INT_MIN`, `INT_MAX`, `UINT_MAX`, `LONG_MIN`, `LONG_MAX`, `ULONG_MAX`, `LLONG_MIN`, `LLONG_MAX`, `ULLONG_MAX`.
Quais são os menores valores para as variantes *unsigned* (que não têm constante definida)?
Note que no *printf* deve indicar na formatação o tipo de dados que quer escrever e respetiva representação de acordo com o valor da constante. Assim, deve usar `%u` para `unsigned int`, `%ld` para `long int`, etc. (veja o manual no terminal com o comando "`man 3 printf`", ou documentação online sobre o C). Resumo:

prefixo de dimensão	tipo base (para escrever em base dez)
h (short), l (long), ll (long long)	u (unsigned), d ou i (int), f (float ou double)

Para compilar o seu programa use o comando:

```
cc exercicio4.c -o ex4
```

Para executar o programa executável gerado utilize o comando:

```
./ex4
```

5. Considere os operadores para sequências de bits: **&** , **|** , **~** , **>>** e **<<** da linguagem C. Complete programa *exercicio5.c* para escrever no ecrã os resultados de expressões que usam esses operadores sobre uma variável `int x`, para obter os resultados a seguir indicados:

a) Colocar o bit 1 de `b` a 1, mantendo os outros inalterados;
b) Colocar os 4 bits mais significativos de `b` a 0, mantendo os outros inalterados;
c) Colocar o bit 2 de `b` a 0, mantendo os outros inalterados;
d) Determinar se o bit 0 de `b` é 0 ou 1;
e) Multiplicar `b` por 16 (sem usar os operadores de multiplicação nem de soma);
f) Dividir `b` por 4 (divisão inteira, sem usar os operadores de divisão nem de subtração);
g) Multiplicar `b` por 12 (sem usar o operador de multiplicação);

† Em arquiteturas de 32 bits é normal que `size_t` esteja definido como `unsigned int`

Teste, escrevendo o resultado para `b` com os valores: `127`, `86`, `1`, `-1`. Para tal escreva em base dez e em hexadecimal (dado que não tem uma função para escrever um valor em binário) o valor inicial de `b` e o resultado de cada expressão. No printf, para escrever em hexadecimal deve usar %x em vez de %u. Justifique os resultados.

6. Escreva uma função em C que permita escrever a representação binária de um valor do tipo `int`:

```
void printBin (int val);
```

7. Descarregue o ZIP disponível no CLIP, extraia os ficheiros e leia atentamente o código do programa **`ints-comp2.c`**. Constatará que, não considerando as duas linhas comentadas, o programa é uma versão abreviada daquele que lhe foi pedido que escrevesse num exercício anterior: imprime a dimensão ocupada em memória por (i) um inteiro com sinal (**`int`**) e (ii) sem sinal (**`unsigned int`**), e os valores máximo e mínimo que podem ser representados em cada caso:

a) Compile e execute o programa. Observe com atenção o valor exibido no **`printf`** que mostra o valor da variável **`in`** – naturalmente, é o mesmo valor que foi exibido no primeiro **`printf`**, i.e., o valor de **`MAX_INT`**

b) **NÃO** remova nenhum comentário, pense apenas no seguinte: que acha que aconteceria se removesse apenas as marcas **`//`** de comentário na linha identificada como **`Linha 1`**? Que valor seria mostrado no ecrã?

c) Agora, remova apenas as marcas **`//`** de comentário na linha identificada como **`Linha 1`**, compile e execute o programa. Acertou? Parabéns, está a compreender bem o que se passa com a representação em complemento para 2. Não acertou? Tente agora perceber o que aconteceu; se não conseguir, espere pela explicação do seu professor…

d) **NÃO** remova mais nenhum comentário, pense apenas no seguinte: que acha que aconteceria se removesse as marcas **`//`** de comentário na linha identificada como **`Linha 2`**? Que valor seria mostrado no ecrã?

e) Finalmente, remova também as marcas **`//`** de comentário na linha identificada como **`Linha 2`**, compile e execute o programa. Acertou? Parabéns, para além de compreender bem o que se passa com a representação em complemento para 2, compreende também o papel da especificação de formato de impressão do **`printf`**. Não acertou? Tente agora perceber o que aconteceu; se não conseguir, espere pela explicação do seu professor...

8. Uma nota final: compile o programa assim,

```
gcc -Wall -ftrapv -o ints-comp2 ints-comp2.c
```

e execute-o. Veja agora o que aconteceu… Leia o manual online do **`gcc`** e procure a explicação da opção **`ftrapv`** (utilize a barra com a palavra ftrapv, isto é, escreva **`/ftrapv`**, para procurar rapidamente a palavra no texto). Espero que compreenda… senão, aguarde pela explicação do seu docente.

Parte II - Números reais

Observações: *O objetivo desta secção é aprofundar/experimentar uma representação de números reais – também designada em computação como representação (de números) em vírgula flutuante.*

9. Considere a representação de inteiros e em vírgula flutuante IEEE 754 de precisão simples. Indique os valores representados pelos seguintes padrões de 32 bits:

binário	int. s/sinal dec	int c/sinal dec	FP/IEEE dec
01000001010110000000000000000000			

11000001101010000000000000000000			
			-3,25
			2,4

10. Admita que numa variável do tipo `float` coloca o valor 0,1. (considere a norma IEEE 754)

a) Qual será a representação binária desse valor em memória? *Se achar que está a obter muitas casas decimais, ou se a sua calculadora deixar de ter precisão suficiente, passe à alínea seguinte.*

b) Comprove o resultado que obteve completando o programa em C **`floats.c`** para que imprima cada um dos bits da variável. Note que o último **`printf`** imprime 20 dígitos do valor efetivamente guardado e mostra o erro da conversão.

Sugestões: no programa **`floats.c`**, a função **`printBin`** assume que a sua estratégia para imprimir os bits passa por "apanhar" um bit da variável **`val`** e processá-lo e, no final, deslocar (*shift*) **`val`** para apanhar o próximo; pode, contudo, usar outra estratégia: definir uma máscara e usá-la para apanhar um bit, etc., e depois deslocar os bits da máscara – e não mexer em **`val`**. A decisão é sua 😊

c) Complete o programa anterior para afixar no ecrã o valor inteiro representado no expoente.