GUILHERME MARTINS BIZERRA

EXERCÍCIO PRÉ-PROVA

LLP-TELEM 2024.1

1. O que é Manipulação de Bits?

A manipulação de bits refere-se ao processo de alterar ou examinar o conteúdo de uma variável ou dado no nível mais básico, isto é, no nível de bits individuais. Em vez de trabalhar com dados como unidades de 8, 16, 32 ou 64 bits (bytes, palavras, etc.), a manipulação de bits trata diretamente de cada bit individual dentro desses dados. Essa abordagem é fundamental para otimizar o uso de memória e para operações de baixo nível que são críticas em programação de sistemas, jogos, criptografia e outras áreas.

2. Qual a Importância da Manipulação de Bits?

A manipulação de bits é importante por várias razões:

- **Eficiência**: Permite operações muito rápidas e econômicas em termos de tempo e memória, já que manipular bits diretamente pode ser mais rápido do que trabalhar com dados de maior granularidade.
- **Controle Preciso**: Proporciona um controle mais preciso sobre dados, essencial para tarefas como configuração de hardware, protocolos de comunicação e programação em sistemas embarcados.
- **Compactação de Dados**: Permite armazenar múltiplos valores lógicos em uma única variável. Por exemplo, uma variável de 8 bits pode armazenar 8 flags booleanos.
- **Algoritmos e Criptografia**: É fundamental em algoritmos de criptografia e compressão, onde operações de bit são usadas para criar e manipular dados de maneira segura e eficiente.

3. Quais são as Operações Básicas de Manipulação de Bits?

As principais operações básicas de manipulação de bits incluem:

- **AND Bit a Bit (&)**: Realiza uma operação lógica "E" entre dois bits. O resultado é 1 se ambos os bits forem 1, caso contrário, é 0.
- **OR Bit a Bit (|)**: Realiza uma operação lógica "OU" entre dois bits. O resultado é 1 se pelo menos um dos bits for 1, caso contrário, é 0.
- **XOR Bit a Bit (^)**: Realiza uma operação lógica "OU exclusivo" entre dois bits. O resultado é 1 se os bits forem diferentes, e 0 se forem iguais.
- **NOT Bit a Bit (~)**: Inverte todos os bits de uma variável, transformando 0s em 1s e 1s em 0s.
- **Deslocamento à Esquerda (<<)**: Move todos os bits de uma variável para a esquerda por um número especificado de posições, preenchendo com 0s à direita.

- **Deslocamento à Direita (>>)**: Move todos os bits de uma variável para a direita por um número especificado de posições, preenchendo com 0s ou com o bit de sinal, dependendo do tipo de deslocamento.

```c
#include <stdio.h>

// Função para exibir os bits de um número
void exibirBits(unsigned int num) {
    for (int i = sizeof(num) * 8 - 1; i >= 0; i--) {
        printf("%d", (num >> i) & 1);
        if (i % 4 == 0) printf(" ");
    }
    printf("\n");
}

int main() {
    unsigned int a = 29;  // Exemplo de número: 29 em binário é 0001 1101
    unsigned int b = 15;  // Exemplo de número: 15 em binário é 0000 1111

    printf("Número a: %u\n", a);
    printf("Bits de a: ");
    exibirBits(a);

    printf("Número b: %u\n", b);
    printf("Bits de b: ");
    exibirBits(b);

    // Operações bit a bit
    printf("a & b: ");
    exibirBits(a & b); // AND

    printf("a | b: ");
    exibirBits(a | b); // OR

    printf("a ^ b: ");
    exibirBits(a ^ b); // XOR

    printf("~a: ");
    exibirBits(~a);    // NOT (inversão de bits)

    printf("a << 2: ");
    exibirBits(a << 2); // Deslocamento à esquerda por 2 bits

    printf("a >> 2: ");
    exibirBits(a >> 2); // Deslocamento à direita por 2 bits

    return 0;
}
```

```
Número a: 29
Bits de a: 0000 0000 0000 0000 0000 0000 0001 1101
Número b: 15
Bits de b: 0000 0000 0000 0000 0000 0000 0000 1111
a & b: 0000 0000 0000 0000 0000 0000 0000 1101
a | b: 0000 0000 0000 0000 0000 0000 0001 1111
a ^ b: 0000 0000 0000 0000 0000 0000 0001 0010
~a: 1111 1111 1111 1111 1111 1111 1110 0010
a << 2: 0000 0000 0000 0000 0000 0000 0111 0100
a >> 2: 0000 0000 0000 0000 0000 0000 0000 0111
```