

Curso de C

Operações com Bits

Aritmética em Bits

Soma:

Soma de dígitos binários:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ e "vai um"}$$

$$\begin{array}{r}
 \begin{array}{cccccc}
 1 & 1 & 1 & 1 & 1 & (vai\ um) \\
 \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \\
 & 0 & 1 & 1 & 0 & 1 \\
 + & 1 & 0 & 1 & 1 & 1 \\
 \hline
 = & 1 & 0 & 0 & 1 & 0 & 0
 \end{array}
 \end{array}$$

Aritmética em Bits

Subtração:

Subtração de dígitos binários:

- $0 - 0 = 0$
- $0 - 1 = 1$ e "empréstimo 1"
- $1 - 0 = 1$
- $1 - 1 = 0$

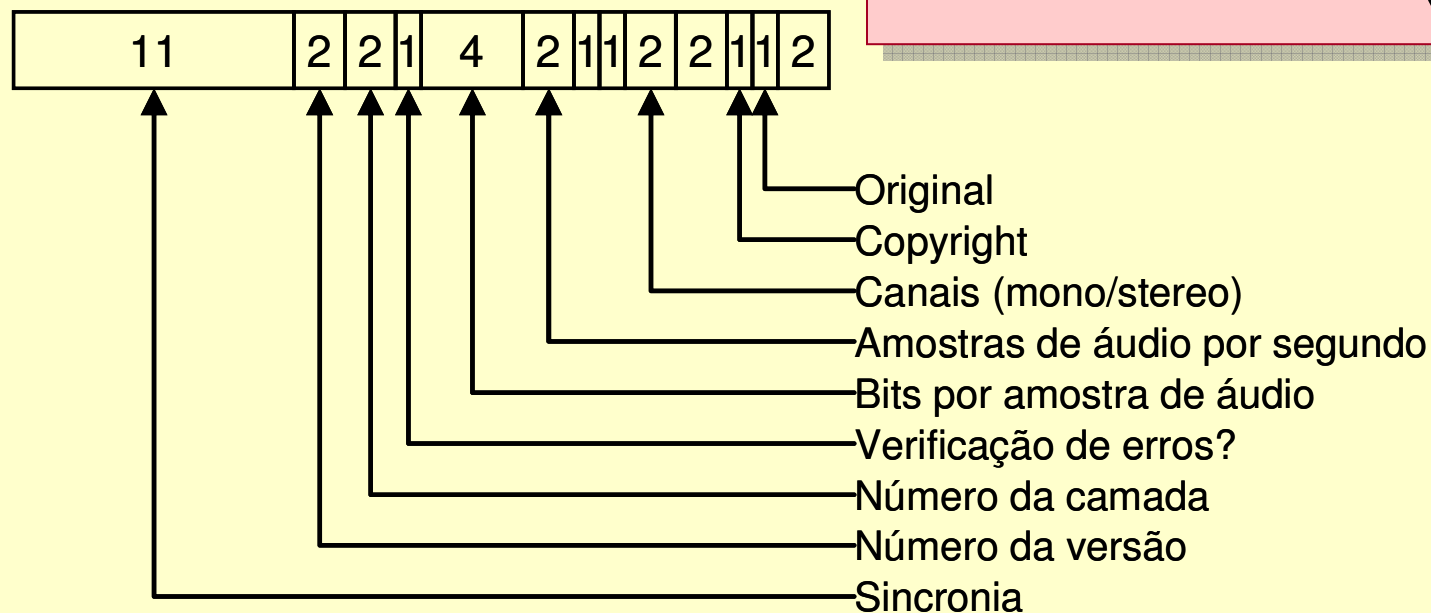
(empresta um)

1	1	0	1	1	1	0
-		1	0	1	1	1
<hr/>						
=	1	0	1	0	1	1

Operações em Bits

Exemplo:

Cabeçalho de MP3:



Objetivos:

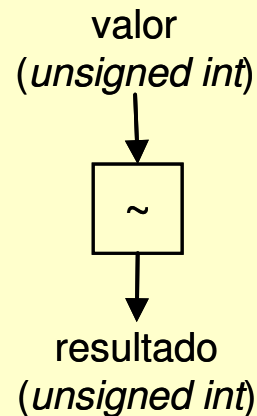
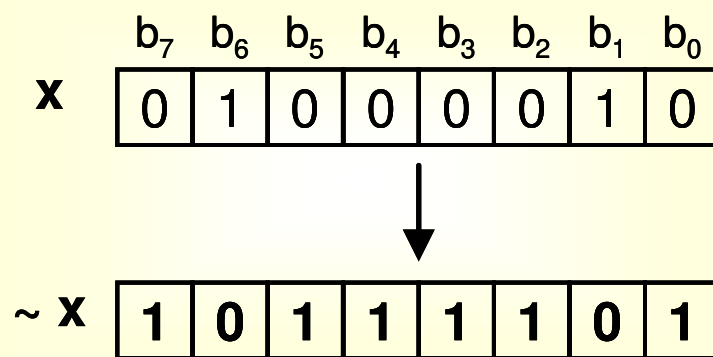
- Verificar estado de um bit
- Atribuir um bit
- Extrair um intervalo (número)

Representação: **unsigned int** (32 bits)

Operações em Bits

NÃO binário:

Inverte todos os bits:



Operador: \sim

Tabela Verdade:

$\sim 0 \rightarrow 1$

$\sim 1 \rightarrow 0$

Não confundir com o operador !

Operações em Bits

NÃO binário:

```
unsigned int v, not_binario_v, not_logico_v;
```

```
leBinario(&v);
```

```
not_binario_v = ~v;
```

```
not_logico_v = !v;
```

```
escreveBinario(not_binario_v);
```

```
escreveBinario(not_logico_v);
```

Resultado:

Entrada:

00010111

Saída:

$\sim v \rightarrow 11101000$

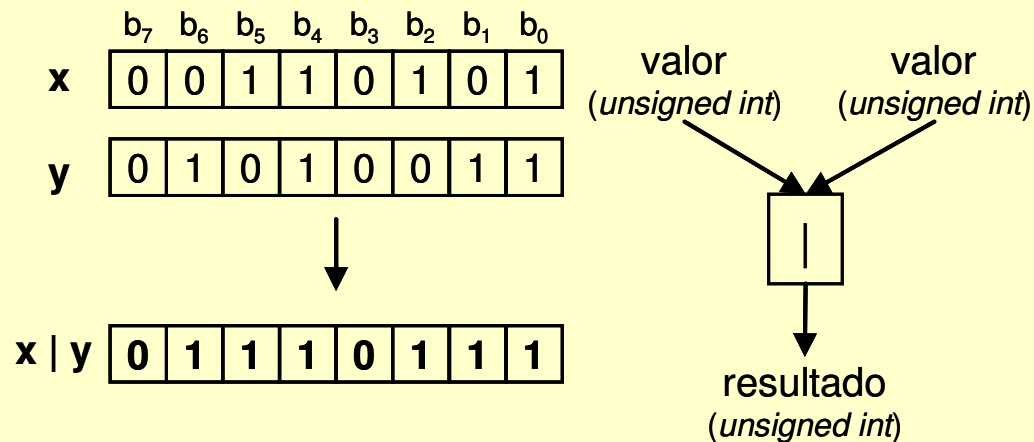
$!v \rightarrow 00000000$

Not01

Operações em Bits

OU binário:

Combina pares de bits:



Operador: |

Tabela Verdade:

0		0	→	0
0		1	→	1
1		0	→	1
1		1	→	1

Não confundir com o operador ||

Operações em Bits

OU binário:

```
unsigned int u, v;  
unsigned int or_binario, or_logico;
```

```
leBinario(&u);  
leBinario(&v);
```

```
or_binario = u | v;  
or_logico = u || v;
```

```
escreveBinario(or_binario);  
escreveBinario(or_logico);
```

Resultado:

Entrada:

u = 00011001

v = 00010101

Saída:

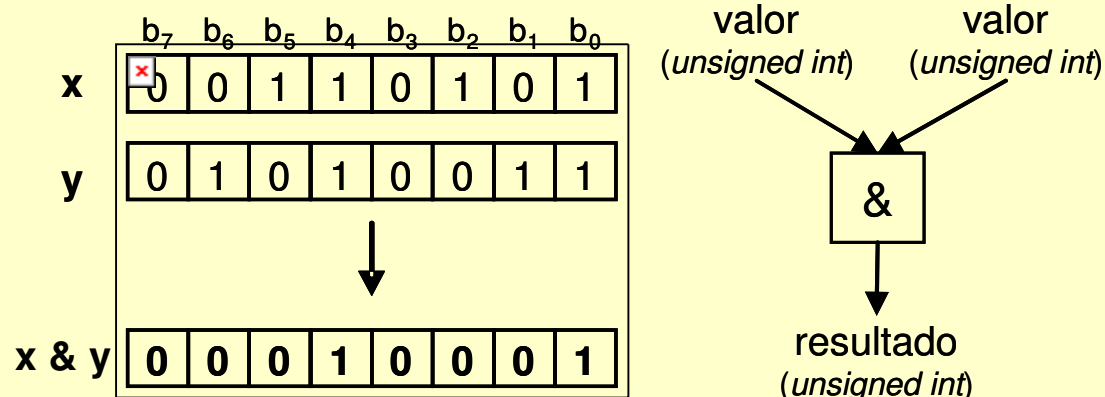
u | v → 00011101

u || v → 00000001

Operações em Bits

E binário:

Unifica pares de bits:



Operador: &

Tabela Verdade:

0	&	0	→	0
0	&	1	→	0
1	&	0	→	0
1	&	1	→	1

Não confundir com o operador &&

Operações em Bits

E binário:

```
unsigned int u, v;  
unsigned int and_binario, and_logico;
```

```
leBinario(&u);  
leBinario(&v);
```

```
and_binario = u & v;  
and_logico = u && v;
```

```
escreveBinario(and_binario);  
escreveBinario(and_logico);
```

Resultado:

Entrada:

u = 00001100

v = 00001010

Saída:

u & v → 00001000

u && v → 00000001

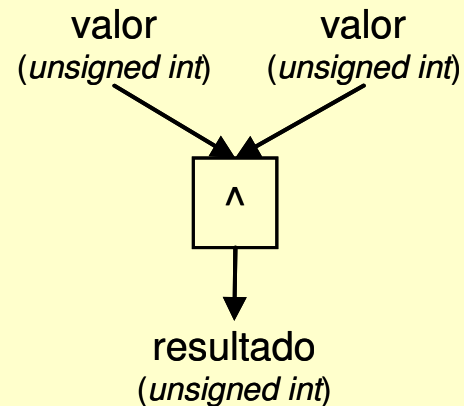
And01

Operações em Bits

OU exclusivo:

Exclui pares de bits iguais:

	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
x	0	0	1	1	0	1	0	1
y	0	1	0	1	0	0	1	1
	↓							
x ^ y	0	1	1	0	0	1	1	0



Operador: ^

Tabela Verdade:

0	^	0	→	0
0	^	1	→	1
1	^	0	→	1
1	^	1	→	0

Operações em Bits

OU EXCLUSIVO binário:

```
unsigned int u, v;  
unsigned int xor_binario;
```

```
leBinario(&u);  
leBinario(&v);
```

```
xor_binario = u ^ v;
```

```
escreveBinario(xor_binario);
```

Resultado:

Entrada:

u = 00001100

v = 00001010

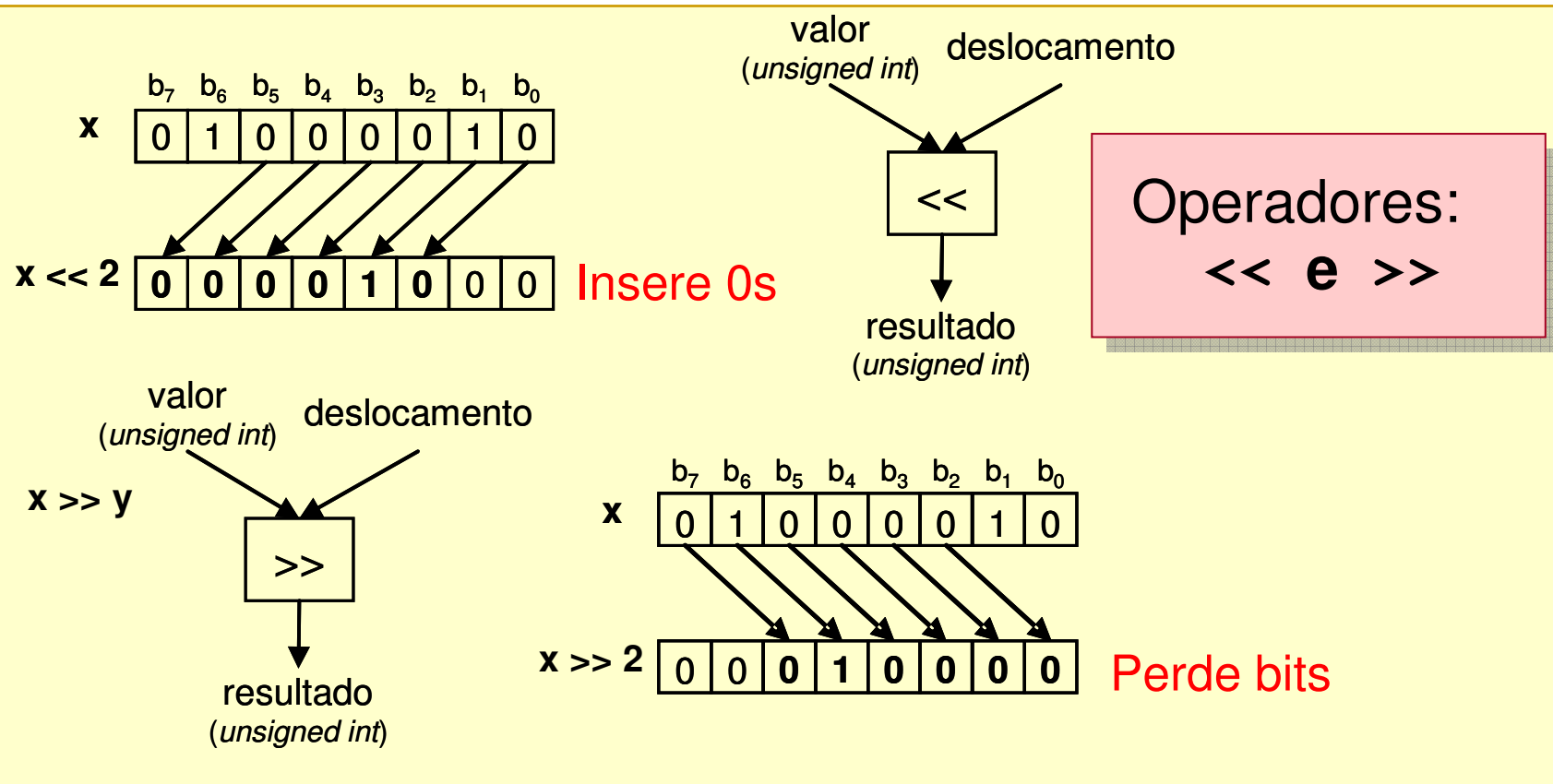
Saída:

u ^ v → 00000110

Xor01

Operações em Bits

Deslocamento de bits:



Operações em Bits

Deslocamento binário:

```
unsigned int v;  
unsigned int v_direita, v_esquerda;
```

```
leBinario(&v);
```

```
v_esquerda = v << 2;
```

```
v_direita = v >> 3;
```

```
escreveBinario(v_esquerda);
```

```
escreveBinario(v_direita);
```

Resultado:

Entrada:

$v = 10010010$

Saída:

$v \ll 2 \rightarrow 01001000$

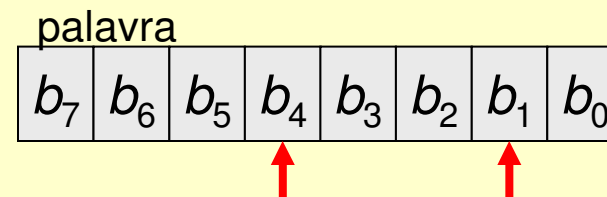
$v \gg 3 \rightarrow 00010010$

Operações em Bits

Máscara de bits:

- Selecionar alguns bits de uma palavra

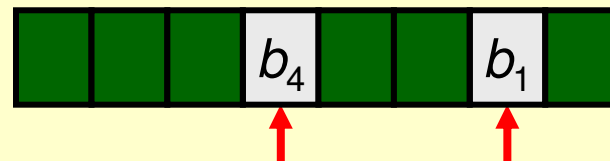
Bits desejados: 1 e 4



Máscara:



Seleção dos bits:



Operações em Bits

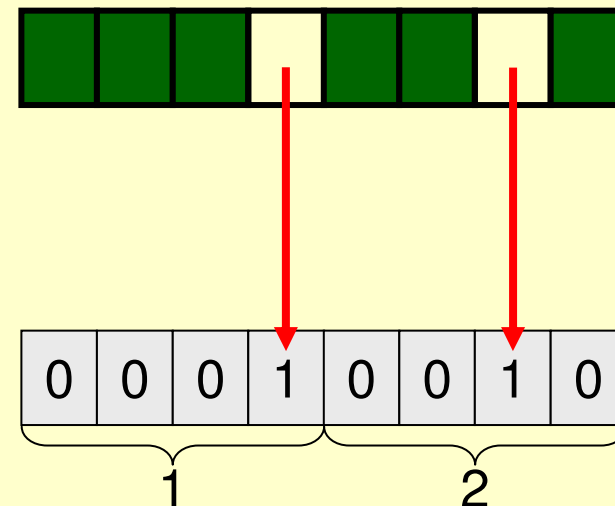
Representação de máscara de bits:

- Máscara: número inteiro e binário

Máscara:

Lacunas: dígito 1
Demais: dígito 0

Representação numérica:

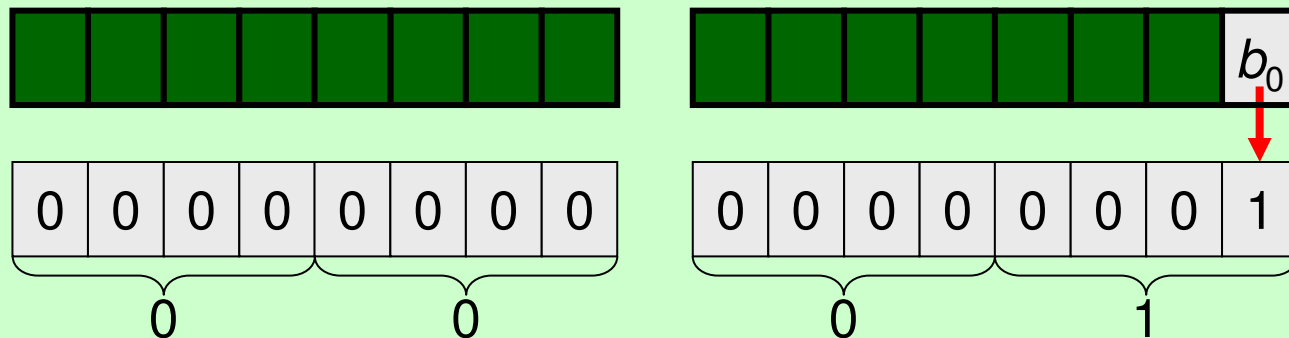


Em decimal: $00010010_2 = 2^4 + 2^1 = 18$
Em hexa: $= 12_{16}$

Operações em Bits

Exemplo de máscara:

- Selecionar primeiro bit:



Representação numérica:

Em decimal: 00000000 00000001₂ = $2^0 = 1$

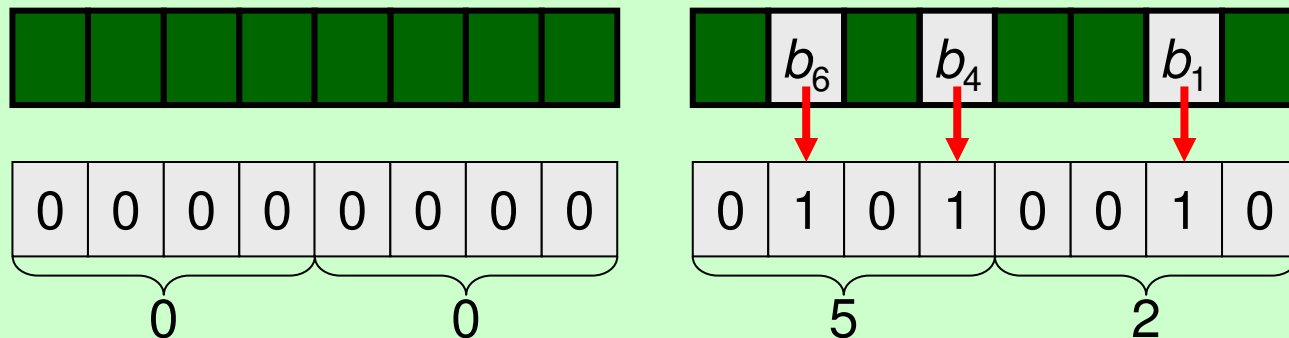
Em hexa = 1₁₆

Em C: 0x0001

Operações em Bits

Exemplo de máscara:

- Selecionar bits 1, 4 e 6:



Representação numérica:

Em decimal: 00000000 01010010₂ = $2^6 + 2^4 + 2^1 = 82$

Em hexa

= 52₁₆

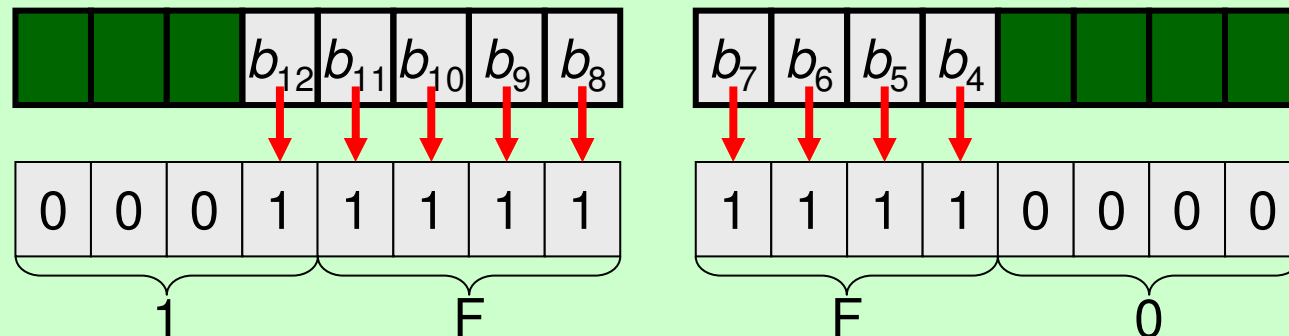
Em C:

0x0052

Operações em Bits

Exemplo de máscara:

- Intervalo de bits de 4 a 12:



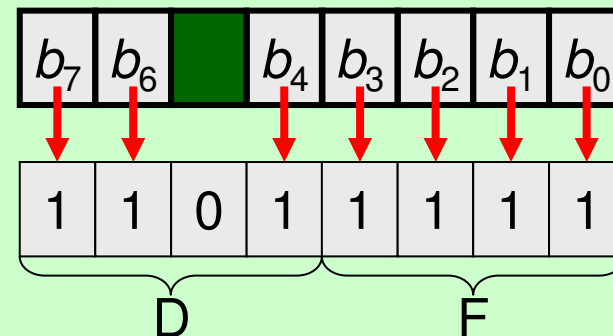
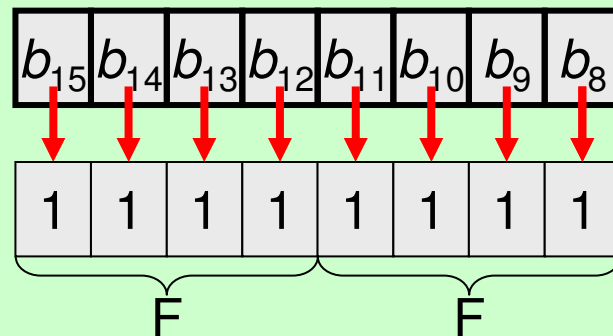
Representação numérica:

Em decimal: $00011111\ 11110000_2 = 8176$
 Em hexa $= 1FF0_{16}$
 Em C: $0x1FF0$

Operações em Bits

Exemplo de máscara:

- Todos, exceto bit 5:



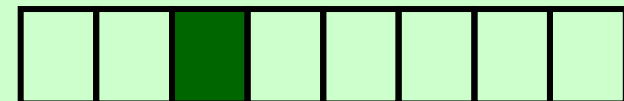
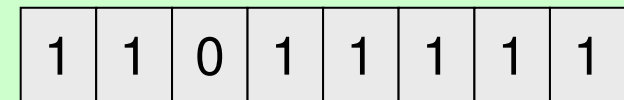
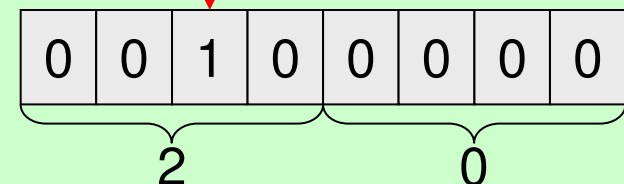
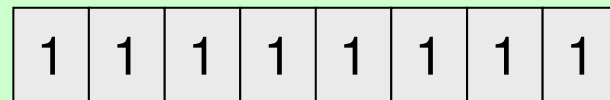
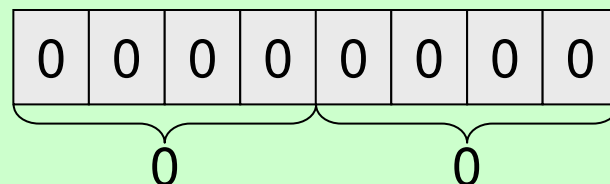
Representação numérica:

Em decimal: $11111111\ 11011111_2 = 65503$
 Em hexa $= \text{FFDF}_{16}$
 Em C: **0xFFDF**

Operações em Bits

Exemplo de máscara:

- Todos, exceto bit 5:



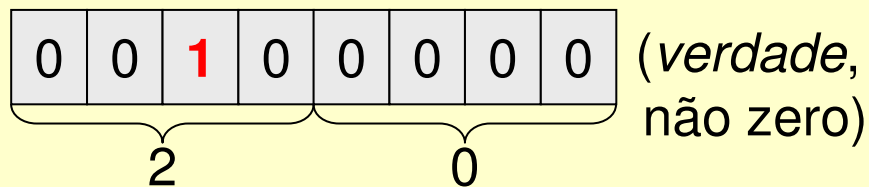
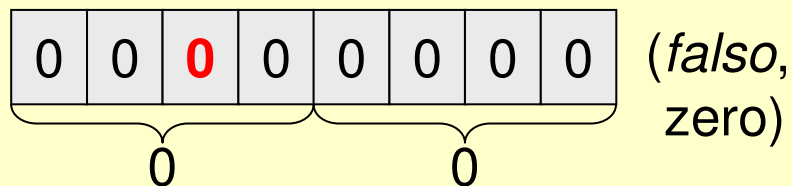
\sim 0x0020

= 0xFFDF

Operações em Bits

Verificar valor de um bit:

- Procedimento:
 - Descobrir máscara
 - Aplicar E binário
 - Verificar com IF



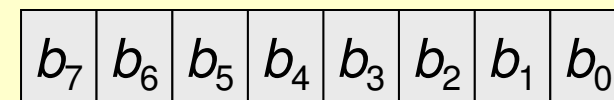
E binário

0 & 0 → 0

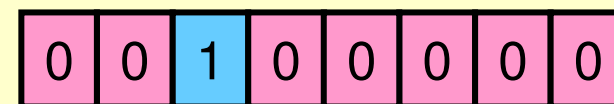
1 & 0 → 0

0 & 1 → 0

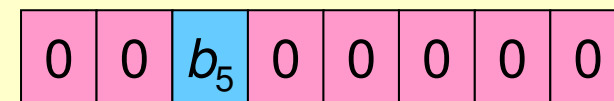
1 & 1 → 1



& (E binário)



=



Operações em Bits

Exemplo:

- Verificar valor do bit 5:

```
unsigned int v;  
leBinario(&v);
```

Máscara para
00000000 0010000

```
if (v & 0x0020) {  
    printf("Bit 5 é 1");  
} else {  
    printf("Bit 5 é 0");  
}
```

LerBit01

Operações em Bits

Alterar o valor de um bit:

- Mudar para 1:
 - Descobrir máscara
 - Aplicar OU binário

OU binário

0		0	→	0
1		0	→	1
0		1	→	1
1		1	→	1

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
-------	-------	-------	-------	-------	-------	-------	-------

| (OU binário)

0	0	1	0	0	0	0	0
---	---	---	---	---	---	---	---

=

b_7	b_6	1	b_4	b_3	b_2	b_1	b_0
-------	-------	---	-------	-------	-------	-------	-------

Operações em Bits

Alterar o valor de um bit:

- Mudar para 0:
 - Descobrir máscara
 - Inverter máscara
(não binário)
 - Aplicar E binário

E binário

0 & 0 → 0

1 & 0 → 0

0 & 1 → 0

1 & 1 → 1

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
-------	-------	-------	-------	-------	-------	-------	-------

& (E binário)

1	1	0	1	1	1	1	1
---	---	---	---	---	---	---	---

=

b_7	b_6	0	b_4	b_3	b_2	b_1	b_0
-------	-------	---	-------	-------	-------	-------	-------

Operações em Bits

Alterar o valor de um bit:

- Trocar valor de um bit:
 - Descobrir máscara
 - Aplicar OU exclusivo

OU exclusivo

0 ^ 0 → 0

1 ^ 0 → 1

0 ^ 1 → 1

1 ^ 1 → 0

$b_7 \ b_6 \ b_5 \ b_4 \ b_3 \ b_2 \ b_1 \ b_0$

^ (OU exclusivo)

0 0 1 0 0 0 0 0

=

$b_7 \ b_6 \ \overline{b_5} \ b_4 \ b_3 \ b_2 \ b_1 \ b_0$

Operações em Bits

Exemplo:

- Mudar o valor do bit 5:

```
unsigned int u, v, w, x;
```

```
// Mudar para 1:
```

```
v = u | 0x0020;
```

```
// Mudar para 0:
```

```
w = u & (~0x0020);
```

```
// Inverter:
```

```
w = u ^ 0x0020;
```

AtribuirBit01

Operações em Bits

Ler um intervalo:

- Procedimento:
 - Descobrir máscara do intervalo
 - Filtrar bits com E binário
 - Deslocar bits para direita

E binário

0	&	0	→	0
1	&	0	→	0
0	&	1	→	0
1	&	1	→	1

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
-------	-------	-------	-------	-------	-------	-------	-------

& (E binário)

0	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---

=

0	b_6	b_5	b_4	0	0	0	0
---	-------	-------	-------	---	---	---	---

>> 4

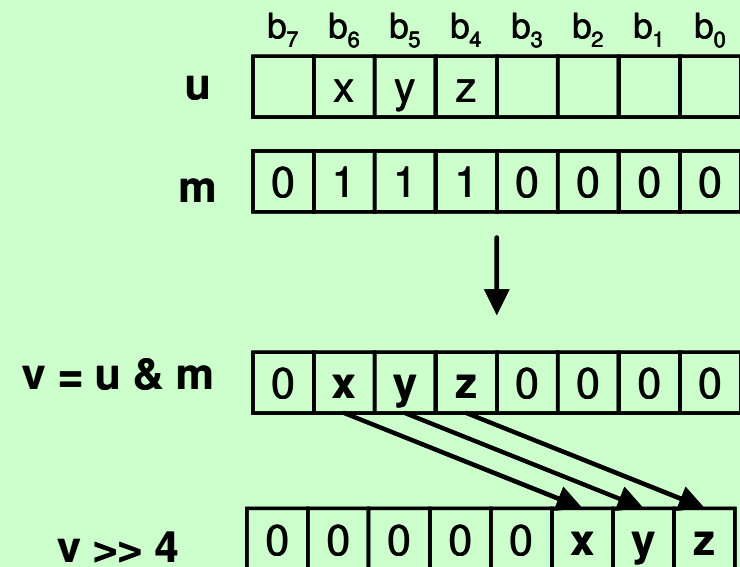
0	0	0	0	0	b_6	b_5	b_4
---	---	---	---	---	-------	-------	-------

Operações em Bits

Exemplo:

- Extrair número no intervalo 4 - 6

```
unsigned int u, v, w;
// Extrair intervalo
v = u & 0x0070;
// Deslocar:
w = v >> 4
```



LerFaixaBits01

Operações em Bits

Atribuir um intervalo de bits:

- Passo 1:
 - Apagar bits na palavra
 - Descobrir máscara invertida do intervalo
 - Aplicar E binário para apagar

E binário

0 & 0 → 0

1 & 0 → 0

0 & 1 → 0

1 & 1 → 1

b_7	x_6	x_5	x_4	b_3	b_2	b_1	b_0
-------	-------	-------	-------	-------	-------	-------	-------

& (E binário)

1	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---

=

b_7	0	0	0	b_3	b_2	b_1	b_0
-------	---	---	---	-------	-------	-------	-------

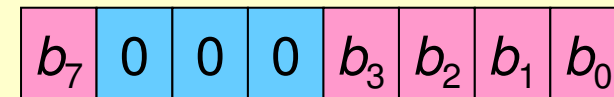
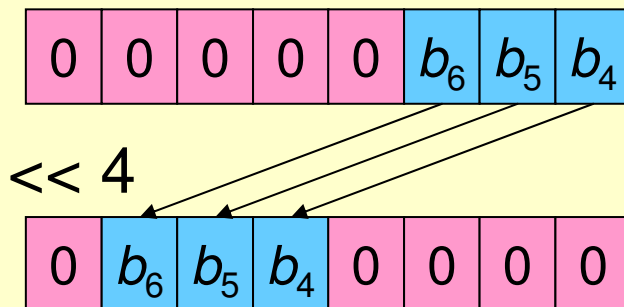
Operações em Bits

Atribuir um intervalo de bits:

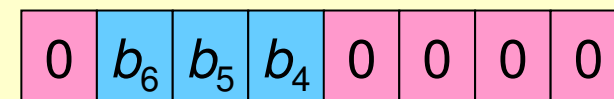
- Passo 2:
 - Deslocar para a esquerda
 - Aplicar OU binário para copiar bits

OU binário

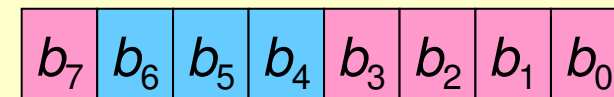
0		0	→	0
1		0	→	1
0		1	→	1
1		1	→	1



| (OU binário)



=



Representação Binária

- TrocaBit01

Representação Binária

Fim