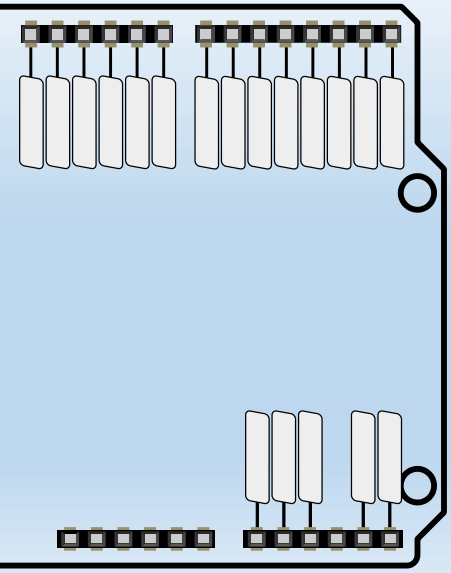
Variáveis

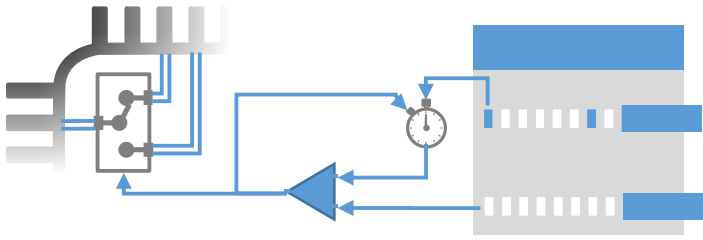


A constante de um homem é a variável de outro.

Alan Perlis

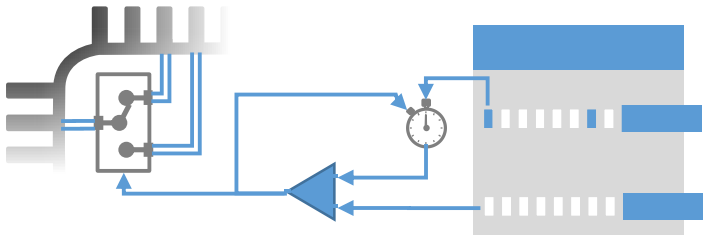
Variáveis

- Uma variável em linguagem C possui:
  - Um tipo que indica o tamanho
  - Um nome para referenciar o conteúdo
  - Um espaço reservado na memória para armazenar seu valor
- É um espaço de memória contém um valor o qual pode ser alterado ao longo do tempo.



Variáveis

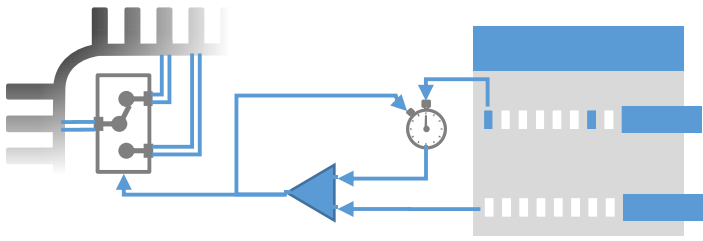
- Para nomes de variáveis podem ser usados quantos caracteres forem desejados contanto que o primeiro caracter seja uma letra ou sublinhado.
- A linguagem C faz distinção entre maiúsculas e minúsculas. matrix e MaTrIx são variáveis distintas
- É comum utilizar apenas minúsculas para nomes de variáveis e apenas MAIÚSCULAS para constantes
- Uma variável não pode ter o mesmo nome de uma palavra chave em linguagem C



Variáveis

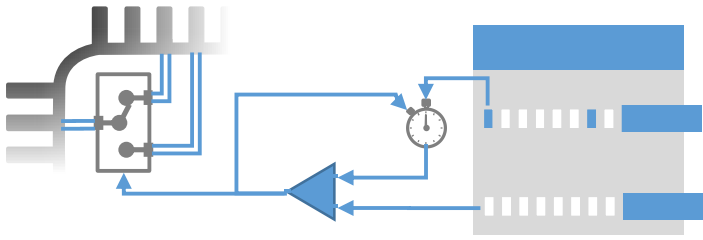
- Consistem num tipo seguindo de nome da variável
- Devem ser declaradas antes de iniciar a codificação
- Variáveis do mesmo tipo podem ser declaradas separadas por vírgula

```
void main(void){
        //Declaracao das variaveis
        int numFuncionarios;
        float salarioMinimo, bonificacao;
        double imposto, descontoEmFolha;
}//end main
```



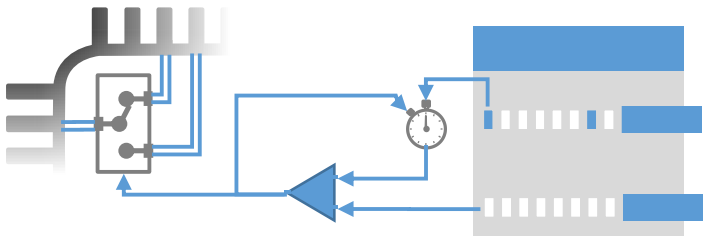
Variáveis

- Toda informação que é inserida num computador é armazenada em formato binário;
- Cada tipo de informação contém uma quantidade diferente de bits;
- Cada bit pode representar informações diferentes, mesmo dentro de uma mesma variável.



Tipos básicos da Linguagem C

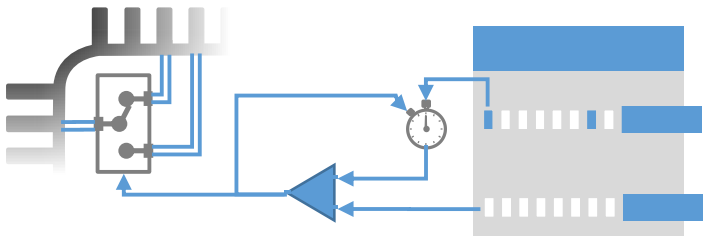
Tipo	Bits	Bytes	Faixa de valores
char	8	1	-128 à 127
int	16	2	-32.768 à 32.767
float	32	4	-3.4×10^{-38} à 3.4×10^{38};
double	64	8	-3.4×10^{-308} à 3.4×10^{-308}



Atribuição de valores

- Inicializar uma variável é atribuir um valor a esta no momento de sua declaração

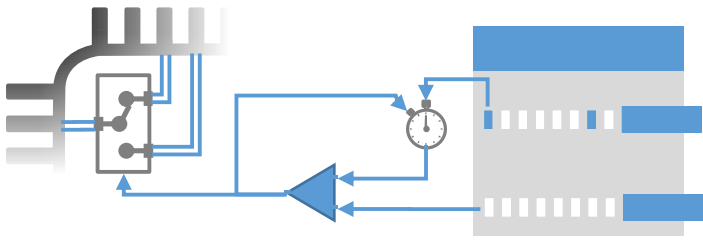
```
void main(void){
        int numFuncionarios = 2;
        float salarioMinimo = 510.0;
        double imposto = 0.25, descontoEmFolha = 151.97;
}//end main
```



Atribuição de valores

- Para armazenar textos utilizamos um vetor de caracteres
- A inicialização deste vetor pode ser feita utilizando o texto entre aspas duplas

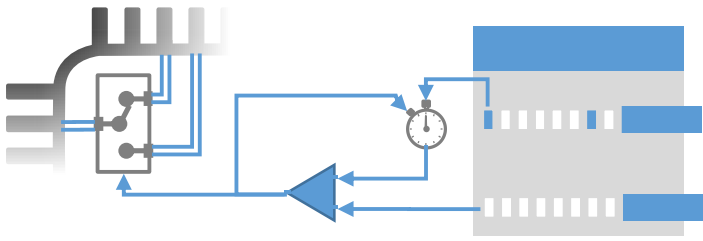
```
void main(void){
        char nome[10] = "Jose";
}//end main
```



Atribuição de valores

- A alteração do valor de uma variável no meio do programa chamada atribuição
- O operador de atribuição é o sinal de igual =

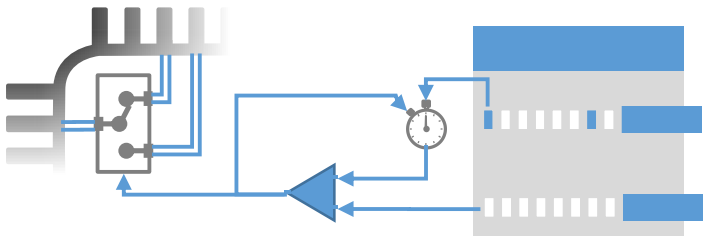
```
void main(void){
        float valor;
        valor = 1234.56;
        //aumentando o valor em 25%
        valor = valor * 1.25;
        return 0;
}//end main
```



Alteração de tipo

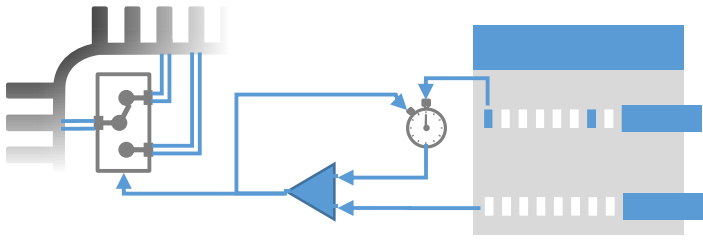
- typecast é a operação de mudança de tipo de um valor
- Para realizar um typecast basta colocar o tipo desejado entre parênteses na frente do valor a ser convertido

```
void main(void){
        int valor;
        float imposto;
        valor = 300;
        imposto = (int) imposto * 0.257;
        //Valor = 300 e imposto = 77
        //por causa da conversão perde precisão
}//end main
```



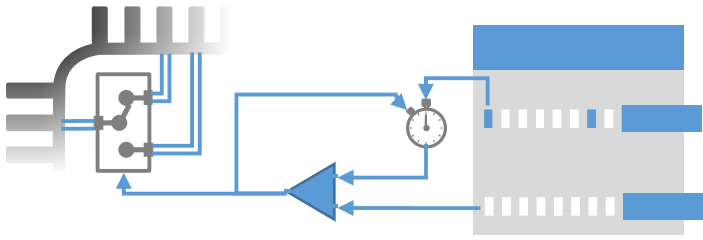
Alteração de tipo

- Ao realizar typecast tenha cuidado para não perder informação. No caso anterior o valor acaba sendo truncado pois, uma variável do tipo int não possue virgula



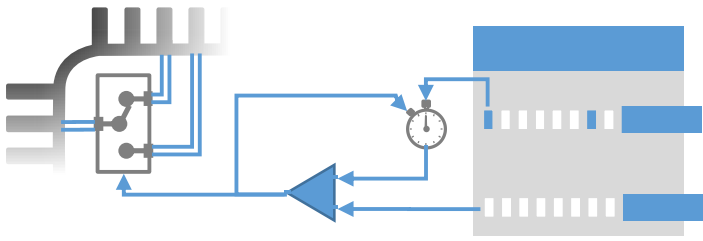
Modificador de tamanho

- Para obtermos um tamanho de variável diferente dos tamanhos padrões podemos utilizar dois modificadores: long e short
  - Uma variável do tipo long deve ser de tamanho MAIOR ou IGUAL a variável to tipo basico modificado;
  - Uma variável do tipo short deve ser de tamanho MENOR ou IGUAL a variável to tipo basico modificado
- Exemplos:
  - short int - 2 bytes: de -32.768 à 32.767;
  - int - 2 bytes: de -32.768 à 32.767;
  - long int - 4 bytes: de -2.147.483.648 à 2.147.483.647;



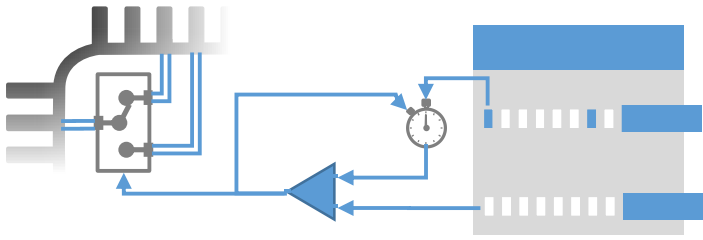
Modificador de tamanho

Tipo	Bytes	Excursão máxima
unsigned char	1	0 à 255
signed char	1	-128 à 127
unsigned int	2	0 à 65.535
signed int	2	-32.768 à 32.767
long int	4	-2.147.483.648 à 2.147.483.647
unsigned long int	4	0 à 4.294.967.295
short int	2	-32.768 à 32.767



Modificador de sinal

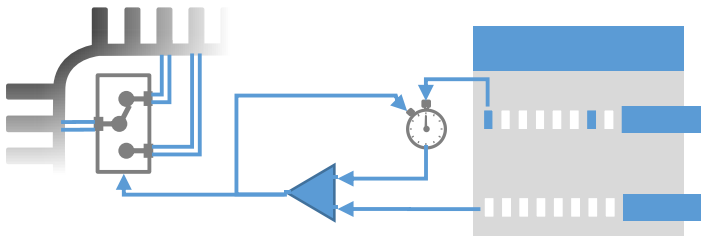
- Todos os tipos básicos apresentados possuem sinal
  - Se não for necessário o uso de sinal é possível utilizar o modificador unsigned
- As variáveis conseguem, com o mesmo espaço, armazenar um valor mais alto
  - Para garantir que aquela variável tem sinal utilizamos o modificador signed
- Exemplos:
  - signed int - 2 bytes: de -32.768 à 32.767;
  - int - 2 bytes: de -32.768 à 32.767;
  - unsigned int - 2 bytes: de 0 à 65.535;



Modificadores de Acesso

- Os modificadores de acesso modificam o comportamento de leitura/escrita da variável

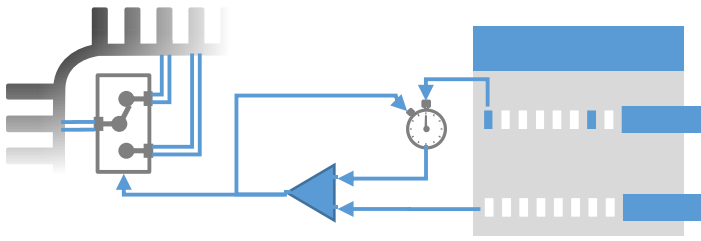
```
void main(void){
        unsigned char X;
        while (X!=X);
}
```



Modificadores de Acesso

- Compilando o código anterior temos em assembly:

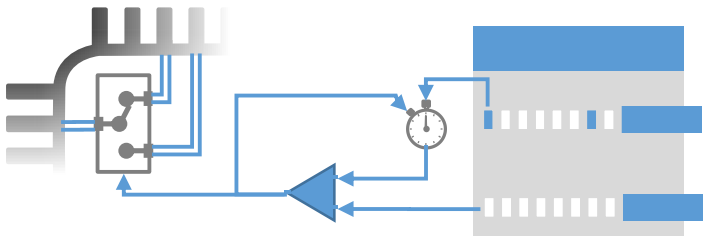
```
// Starting pCode block
S_Teste__main code
_main:
.line 19 // Teste.c while (X!=X);

RETURN
```



Modificadores de Acesso

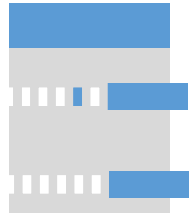
- Adicionando o modificador volatile a variável passa a ser interpretada como possível de alteração, por isso o compilador sempre realiza a leitura.

```
void main(void) {
        volatile unsigned char X;
        while (X!=X);
}
```



Modificadores de Acesso

```
S_Teste__main code // Starting pCode block
_main:
_00105_DS_:
.line 19 // Teste.c while (X != X);
MOVLW 0x83 //primeira parte do endereço
MOVWF r0x00
MOVLW 0x0f //segunda parte do endereço
MOVWF r0x01
MOVFF r0x00, FSR0L
MOVFF r0x01, FSR0H
MOVFF INDF0, r0x00 //realiza primeira leitura
MOVLW 0x83 //primeira parte do endereço
MOVWF r0x01
MOVLW 0x0f //segunda parte do endereço
MOVWF r0x02
MOVFF r0x01, FSR0L
MOVFF r0x02, FSR0H
MOVFF INDF0, r0x01 //realiza segunda leitura
MOVF r0x00, W
XORWF r0x01, W
BNZ _00105_DS_ //faz o teste para igualdade
RETURN
```

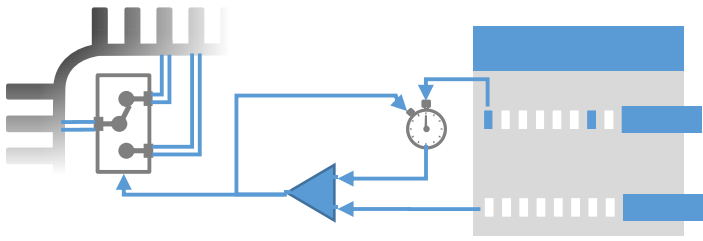


Modificadores de acesso

- Utilização de const:

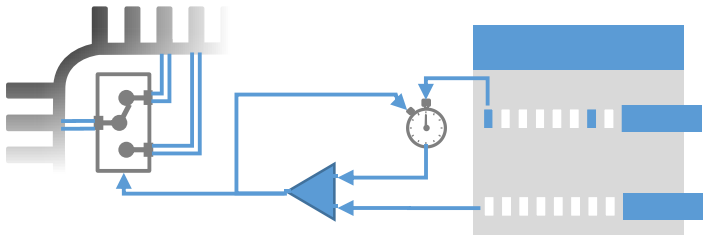
```
//inicio do programa
void main(void)  {
        constunsigned char X;
        X = 3;
}
```

- A compilação resulta em erro, já que a variável X não pode ser alterada.



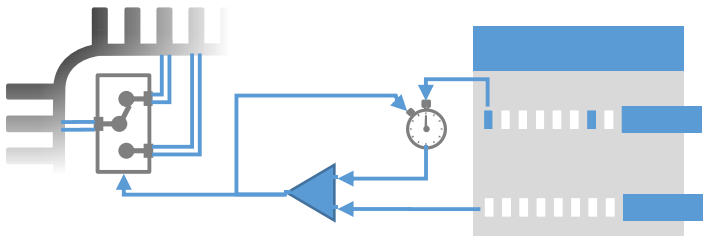
Modificador de persistência

- A persistência de uma variável pode ser assegurada com o modificador static
- Com ele, uma região de memória é reservada apenas para a variável definida
- Permite criar variáveis de armazenamento permanente entre chamadas de funções
- A memória alocada para a variável não pode ser utilizada em outro lugar



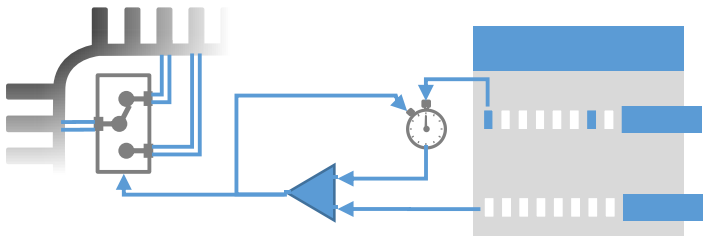
Modificador de persistência

```
//cria uma função de contagem
//como a variável é simples o valor se perde
int ContadorPersistente(int reseta){
        char variavel_persistente;
        if (reseta) {
                variavel_persistente = 0;
        }else{
                return (variavel_persistente++);
        }
        return -1;
}
```



Modificador de persistência

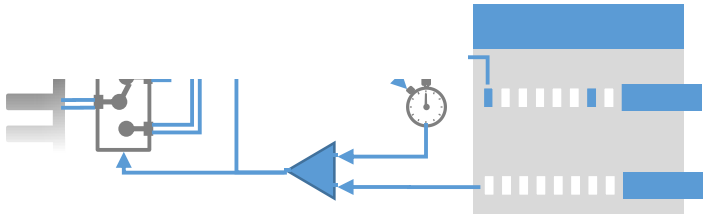
```
//cria um contador persistente que é
//incrementado a cada chamada de função
int ContadorPersistente(int reseta){
        static char variavel_persistente;
        if (reseta){
                variavel_persistente = 0;
        }else{
                return (variavel_persistente++);
        }
        return -1;
}
```



Operação com variáveis

- Quais os problemas de precisão que podem acontecer?

```
void main (void) {
        char var8;
        int var16;
        long int var32;
        float pont16;
        double pont32;
        var8 = var8 + var16;      // 1
        var8 = var8 + var8;       // 2
        var16 = var8 * var8;      // 3
        var32 = var32 / var16;    // 4
        var32 = pont32 * var32;   // 5
        pont16 = var8 / var16;    // 6
        pont16 = pont32 * var32; // 7
        pont16 = 40 / 80;         // 8
}
```

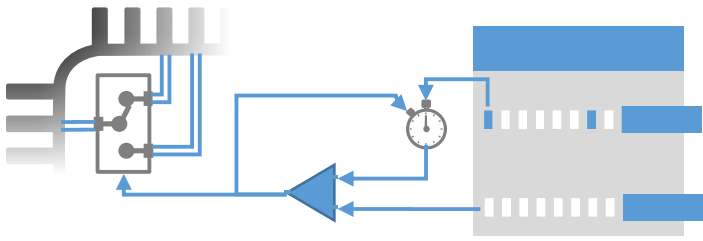


Operação com variáveis

- Quando resolvemos codificar uma informação num formato digital, e portanto finito, sempre existe perda de
  - informação
  - resolução
  - Amplitude
- Depois de quantas iterações os loops param?

```
float x = 0;
while (x != 4) {
        lcdChar('*');
        x += 0.4f;
}
```

```
char x = 0;
while (x < 200) {
        lcdChar('*');
        x++;
}
```

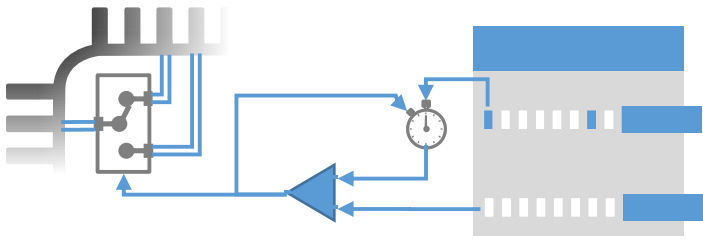


Ponteiros e Referência

Ponteiros

- São variáveis que guardam um endereço (localização) de memória
  - os tipos de valores colocados nos endereços de memória são definidos na declaração de um ponteiro
    - é esse tipo que indica ao compilador a quantidade de memória necessária para armazenar os valores
  - uma variável do tipo ponteiro aponta para uma variável de um determinado tipo (char, int, float, double, ...})
  - é necessário na declaração de um ponteiro, especificar para qual tipo de variável ele irá apontar
  - os ponteiros são declarados com um * antes do nome da variável

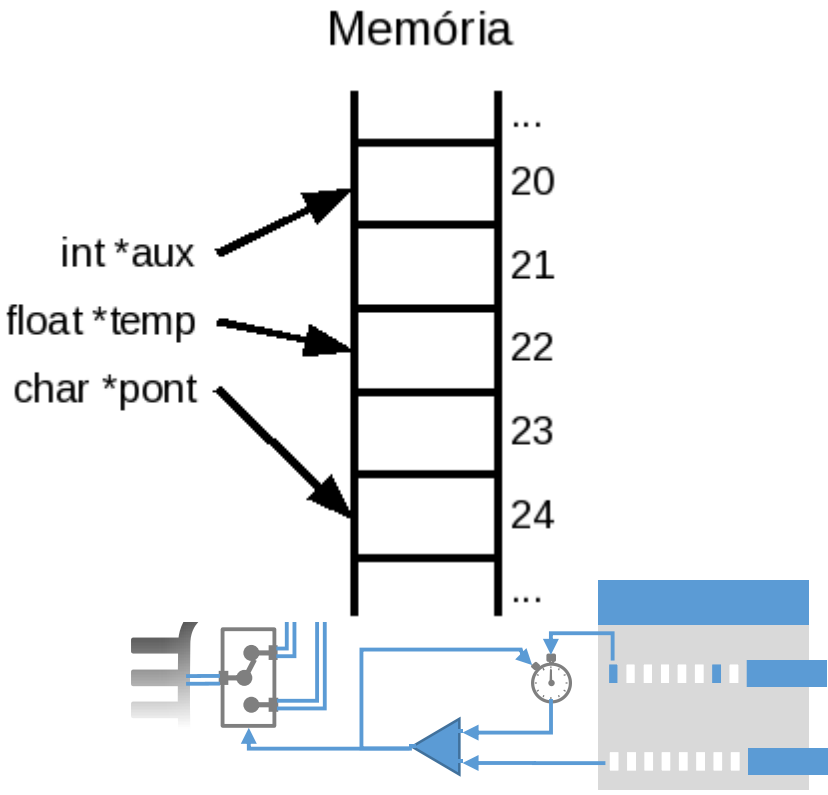
```
tipo *nomeVariavel;
```



Ponteiros

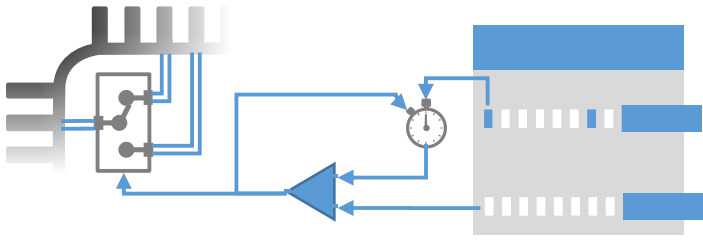
- aux, temp e pont são variáveis que armazenam endereços de memória e não valores do tipo int, float, char
- * é usado quando deseja-se acessar o valor que está na posição de memória e não ao endereço da memória

```
int *aux;
float *temp;
char *pont;
```



Os operadores e os ponteiros

- O operador & e o operador * são utilizados quando trabalhamos com ponteiros
- Operador &
  - obtém o endereço da variável escrita depois do operador (endereço de)
  - como os ponteiros também são variáveis eles ocupam memória
  - pode-se obter o endereço do ponteiro e ter ponteiros para ponteiros (múltiplos *)
- Operador *
  - o operador * faz o contrário do operador &
  - dado um ponteiro, o operador * acessa o conteúdo apontado por ele



Os operadores e os ponteiros

```
void main(void){
        int x=10;
        int *p1 = &x; //ponteiro para um inteiro

        lcdInit()
        *p1 = 20; //ou p1[0] = 20;

        lcdNumber(p1);
        lcdNumber(x);
        lcdNumber(*p1);
        lcdNumber(p1[0]);
        return 0;
} //endMain
```