
# RISC-V: The Free and Open RISC Instruction Set Architecture

Rodolfo Azevedo

MC404 – Organização Básica de Computadores e Linguagem de Montagem

http://www.ic.unicamp.br/~rodolfo/mc404

# Variáveis Globais

- Variáveis globais ficam na área de dados

```
.section .data
altura:
   .word 200
largura:
   .word 17
```

# Constantes

- Constantes ficam na área de dados

```
.section .rodata
altura:
   .word 200
largura:
   .word 17
```

# Variáveis Locais

- Variáveis locais são alocadas na pilha, de forma similar ao utilizado para armazenar registradores

```
addi sp, sp, -16
sw ra, 12(sp)
sw s0, 8(sp)
```

- As variáveis altura e largura ficariam nos outros 8 bytes

# Struct

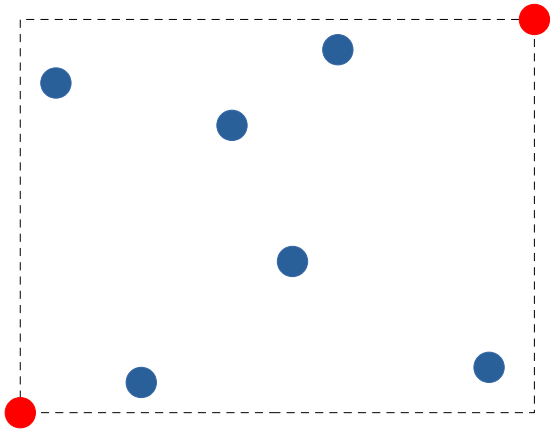
- É importante considerar o tamanho total de cada estrutura de dados
- Reservar o espaço de memória tipicamente na ordem em que os campos aparecem
- É comum fazer padding dos dados
  - Alinhar o tamanho para múltiplo de 4 bytes para usar instruções de palavras

# Exemplo

```
struct Pessoa {
   char nome[20];
   char sobrenome[20];
   int idade;
   float altura;
}
```

# Exemplo Completo

- Escreva um código que recebe um vetor de pontos e encontra o menor retângulo que contém todos eles. Todas as coordenadas são números inteiros e não podem existir pontos nas bordas.



# int main()

```
struct ponto {int x, y;};
struct ponto v[8] = …;
main() {
  struct ponto inf_esq, sup_dir;

  EncontraCantos(v, 8, &inf_esq, &sup_dir);
}
```

# void EncontraCantos(ponto *v, int tamanho, ponto *inf_esq, ponto *sup_dir)

```
*inf_esq = *v;
*sup_dir = *v;
tamanho --;
v ++;
while (tamanho) {
  AjustaInfEsq(v, inf_esq);
  AjustaSupDir(v, sup_dir);
  tamanho –;
  v ++;
}
inf_esq → x--; inf_esq → y --;
sup_dir → x++; sup_dir → y ++;
```

# AjustaInfEsq(ponto *p, ponto *inf_esq)

```
if p→x < inf_esq→x
   inf_esq→x = p→x;
if p→y < inf_esq→y
   inf_esq→y = p→y;
```

# AjustaSupDir(ponto *p, ponto *sup_dir)

```
if p→x > sup_dir→x
   sup_dir→x = p→x;
if p→y > sup_dir→y
   sup_dir→y = p→y;
```

# Exceções e Interrupções

- Eventos que podem causar a transferência da execução para outra parte do código, tipicamente para o Sistema Operacional
- Exceções
  - Causas internas ao core
  - Divisão por zero, falha de página, etc
- Interrupções
  - Causas externas ao core
  - Movimento do mouse, tecla digitada, dados prontos da rede ou disco

# Comunicação com periféricos

- Cada periférico possui um ou mais endereços de memória
- O algoritmo básico de leitura de um periférico é fazer polling:

```
if (tem_algo_para_ler()){
   Leia();
   Processe();
}
```

# Utilizando interrupções

- Forma alternativa de tratamento onde o periférico avisa o processador quando tem algo para tratar, chamando uma rotina de interrupção
- Existem múltiplas alternativas de interrupções para implementar, as variações são relacionadas à quantidade de trabalho de software e hardware

# Distribuição de trabalho entre SW e HW

- Totalmente em SW
  - Uma única rotina é chamada para qualquer evento externo e deve consultar todos os periféricos para descobrir o que aconteceu
- Híbrido
  - Uma única rotina é chamada para qualquer evento externo e recebe um registrador indicando o causador da interrupção
- Auxiliada por HW
  - Uma rotina diferente é chamada para cada evento externo facilitando a forma como o software é escrito

# Tipos de rotinas

- Endereço único
  - Único tratador cujo endereço fica num registrador especial
  - A cada evento externo, o PC atual é salvo e alterado para esse valor especial
- Tratador individualizado
  - Único endereço base que fica num registrador especial
  - Os endereços das rotinas de tratamento podem ser encontrados interpretando esse registrador como um vetor de endereços

# Registradores Especiais do Processador

- São registradores além dos 32 que já trabalhamos
- Guardam informações sobre o processador e controlam operações extras
- Também chamados de CSR → Control and Status Registers
- 3 instruções especiais
  - csrr a0, mscratch
  - csrw mscratch, a0
  - csrrw a0, mscratch, a0