SSC0902
Organização e Arquitetura de Computadores

4ª e 5ª Aulas – Arquiteturas CISC e RISC

Arquitetura e Assembly da arquitetura RISC-V

Profa. Sarita Mazzini Bruschi
sarita@icmc.usp.br

Livro texto: Guia Prático RISC-V
David Patterson & Andrew Waterman

Arquitetura CISC

- CISC – *Complex Instruction Set Computer*
  - Computadores complexos devido a:
    - Instruções complexas que demandam um número grande de ciclos para serem executadas
    - Dezenas de modos de endereçamento
    - Instruções de tamanhos variados
    - Referência a operandos na memória principal
  - Questionamentos quanto à necessidade de certas instruções
    - Levantamentos mostram as instruções mais utilizadas nos programas

Arquitetura CISC

- Estudos de Knuth, Wortman, Tanenbaum e Patterson em várias linguagens com relação a porcentagem de comandos

Comando	Fortran	C	Pascal
atribuicao :=	51%	38%	45%
if	10	43	29
call	5	12	15
loop	9	3	5
goto	9	3	0
outros	16	1	6

Arquitetura CISC

- Portanto:
  - Nas arquiteturas CISC fica mais difícil implementar o pipeline
  - A taxa média de execução das instruções por ciclo tende a ser bem menor do que 1 IPC (*instruction per cycle*)
  - A unidade de controle é microprogramada
  - Instrução complexa significa um maior tempo para decodificar e executar, muitas das quais são raramente usadas
- Surgiu então a arquitetura RISC

Arquitetura RISC

- RISC – *Reduced Instruction Set Computer*
- Características:
  - Instruções mais simples, demandando um número fixo de ciclos de máquinas para sua execução
  - Uso de poucos e simples modos de endereçamento
  - Poucos formatos das instruções
  - Apenas instruções de load/store referenciam operandos na memória principal
  - Cada fase de processamento da instrução tem a duração fixa igual a um ciclo de máquina

Arquitetura RISC

- Portanto:
  - Implementadas com o uso do pipeline
    - Formato fixo das instruções facilita o pipeline
  - As instruções são executadas na sua maioria em apenas um ciclo de máquina
  - A unidade de controle é em geral *hardwired*
    - Não há microprograma para interpretar as instruções
  - Arquitetura orientada a registrador
    - Todas as operações aritméticas são realizadas entre registradores
    - Define-se um grande conjunto de registradores

Arquitetura RISC

- Primeiros computadores RISC:
  - IBM 801 (1980)
    - É o antecessor do IBM PC/RT (RISC Tecnology)
  - Berkeley RISC I e RISC II (1980 e 1981)
    - Projetado por Patterson e Séquin
    - Inspirou o projeto do processador SPARC, da SUN Microsystem
  - Stanford MIPS (1981)
    - Projetado por Hennessy
    - Originou a MIPS Computer Systems

Arquitetura RISC-V

- RISC-V (RISC-five)
  - Arquitetura de conjunto de instruções universal
  - Arquitetura aberta
  - Vários requisitos:
    - Atender todos os tamanhos de processadores
    - Funcionar bem em uma grande variedade de software e ling. de programação
    - Acomodar tecnologias de implementação
    - Ser eficiente para todo tipo de microarquitetura (organização)
    - Ser estável (ISA base não deve mudar)

Arquitetura RISC-V

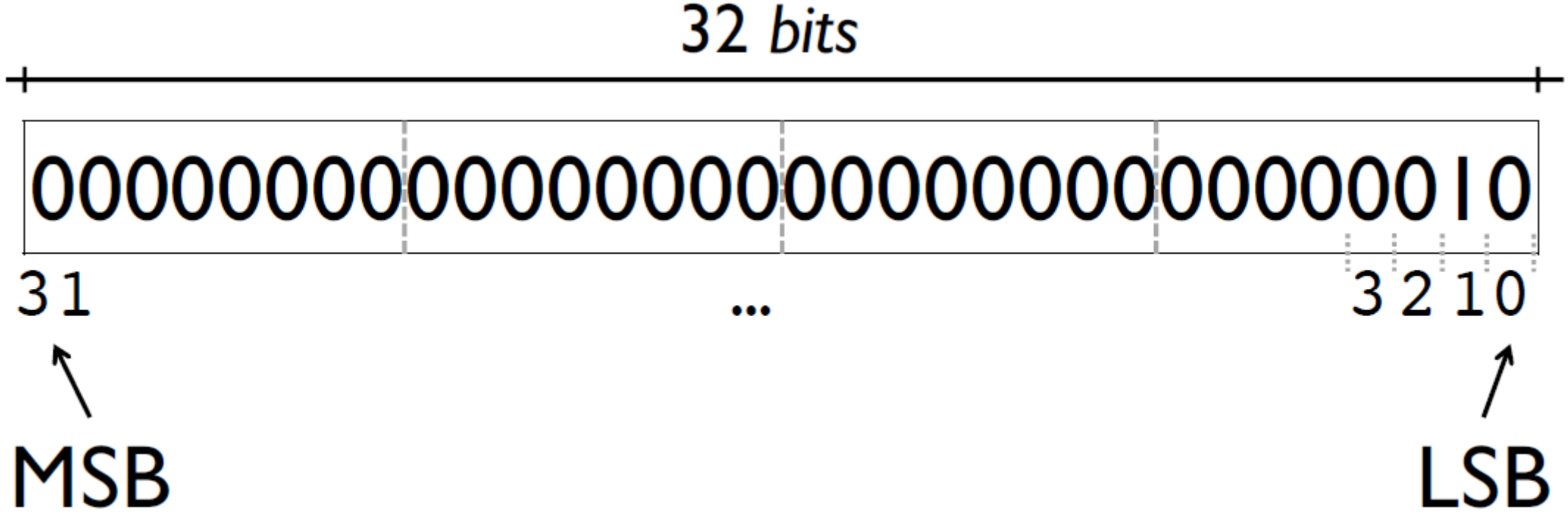
- Mantida atualmente pela Fundação RISC-V
  - www.riscv.org
  - Fundação aberta e sem fins lucrativos
  - Mais de 325 empresas parceiras!

>$50B		>$5B, <$50B		>$0.5B, <$5B	
Google	USA	BAE Systems	UK	AMD	USA
Huawei	China	MediaTek	Taiwan	Andes Technology	China
IBM	USA	Micron Tech.	USA	C-SKY Microsystems	China
Microsoft	USA	Nvidia	USA	Integrated Device Tech.	USA
Samsung	Korea	NXP Semi.	Netherlands	Mellanox Technology	Israel
		Qualcomm	USA	Microsemi Corp.	USA
		Western Digital	USA		

Arquitetura RISC-V

- Características da Arquitetura RISC-V
  - Arquitetura de 32 bits (existem arquiteturas mais recentes de 64 bits)
  - Possui 32 registradores de propósito geral
  - Possui 32 registradores para ponto flutuante

Registradores do RISC-V



Registradores do RISC-V

Registrador	Nome ABI	Descrição	Preservado em toda a chamada?
x0	zero	Hard-wired zero	—
x1	ra	Endereço de retorno	Não
x2	sp	Ponteiro de pilha	Sim
x3	gp	Ponteiro global	—
x4	tp	Ponteiro de Thread	—
x5	t0	Registrador de link temporário/alternativo	Não
x6–7	t1–2	Temporários	Não
x8	s0/fp	Registrador salvo/Ponteiro de quadro	Sim
x9	s1	Registrador salvo	Sim
x10–11	a0–1	Argumentos de função / valores de retorno	Não
x12–17	a2–7	Argumentos de função	Não
x18–27	s2–11	Registradores salvos	Sim
x28–31	t3–6	Temporários	Não
f0–7	ft0–7	Temporários FP	Não
f8–9	fs0–1	Registradores salvos FP	Sim
f10–11	fa0–1	Argumentos e valores de retorno FP	Não
f12–17	fa2–7	Argumentos FP	Não
f18–27	fs2–11	Registradores salvos FP	Sim
f28–31	ft8–11	Temporários FP	Não

Arquitetura RISC-V

- Possui memória endereçada a byte
  - Menor unidade endereçável é 1 byte
- Outros tipos de dados:
  - halfword: 2 bytes
  - word: 4 bytes
  - float: 4 bytes
  - double: 8 bytes

Arquitetura RISC-V

- Diversos conjuntos de instruções:
  - RV32I: Conjunto base de 32 bits com instruções para operações com números inteiros.
  - RV32M: Instruções de multiplicação e divisão
  - RV32F e RV32D: Instruções de ponto-flutuante
  - RV32A: Instruções atômicas
  - RV32C: Instruções compactas, de 16 bits
  - RV32V: Instruções vetoriais (SIMD)

Arquitetura RISC-V

- Arquitetura Load/Store: Os valores têm que ser carregados nos registradores antes de realizar as operações.
  - Não há instruções que operam diretamente em valores na memória!

Assembly RISC-V

- Para programar na arquitetura RISC-V é necessário saber o Assembly RISC-V
  - Estrutura do código
    - Rótulos e Diretivas
    - Chamada ao sistema para fazer E/S - ecall
  - Instruções básicas
  - Ferramenta de simulação da arquitetura
    - RARS (RISC-V Assembler and Runtime Simulator)
  - Exemplo “Hello World”

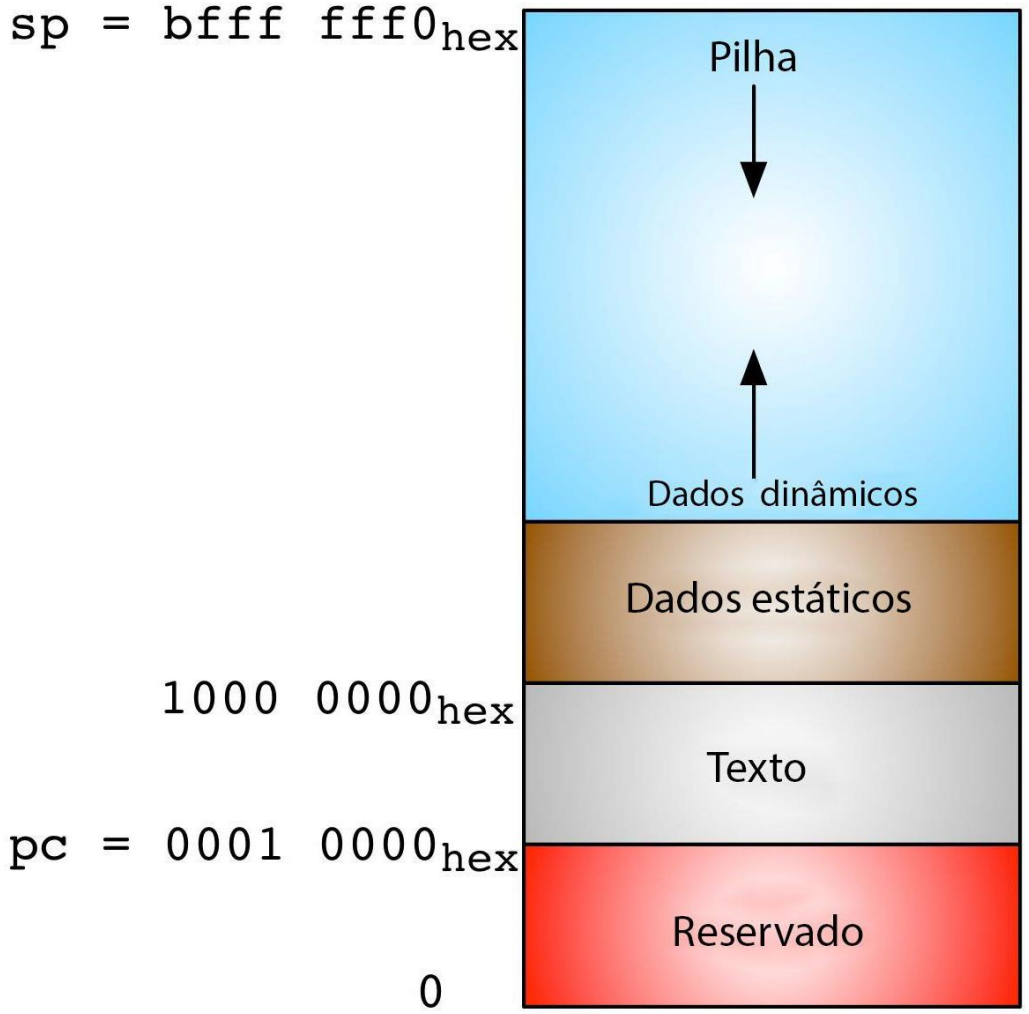
Estrutura de um programa assembly

- Segmento de texto (código)
  - endereço 0x00010000
- Segmento de dados
  - endereço 0x10000000

```
.data     # diretiva que indica o início do segmento de dados
# variáveis estáticas

.text     # diretiva que indica o início do segmento de texto
# código fonte
```

Layout de memória usado pelo RISC-V



Rótulos

- Identificam uma linha no código para referência
- Úteis para :
  - desvios condicionais
    - estruturas condicionais e de repetição
  - desvios incondicionais
  - chamadas a procedimentos
    - uma variação de desvio incondicional
- sempre são seguidos de dois pontos

```
loop:           beq t0, t1, fim_loop
                # código interno ao loop
                j loop
fim_loop:       # primeira instrução fora do loop
```

Rótulos

```
                .data           # diretiva p/ início do seg de dados
vlr_inteiro:    .word 157
string:         .asciiz “Hello World"

                .text           # diretiva p/ início do segmento de texto
                .globl main     # diretiva p/ usar rotulo em outro programa
main:                           # rótulo para ponto de entrada no processo

loop:           beq t0, t1, fim_loop  # se t0=t1go fim_loop

                # código interno ao loop

                j loop                  # retorne p/ inicio do loop
fim_loop:       # primeira instrução fora do loop
```

Diretivas do Montador

- Determinam configurações ao código
  - sempre precedidas por ponto

.align n
.ascii str
.asciiz str
.byte b1, ..., bn
.data
.double d1, ..., dn
.float f1, ..., fn
.globl
.half h1, ..., hn
.space n
.text
.word w1, ..., wn

Diretivas do Montador

```
                .data                   # diretiva p/ início do seg de dados
                .align 0                # diretiva alinhar a memória em caractere
string:         .asciiz “Hello World”   # diretiva para definir uma string
                .align 2                # diretiva alinhar a memória em inteiro
vlr_inteiro:    .word 157               # diretiva para definir um inteiro
                .text                   # diretiva p/ início do seg. de texto
                .globl main             # diretiva p/ usar rotulo em outro prog.

main:           # rótulo para ponto de entrada no processo

                # código fonte
```

Chamadas ao sistema (ecall)

Name	Number	Description	Inputs	Ouputs
PrintInt	1	Prints an integer	a0 = integer to print	N/A
PrintFloat	2	Prints an floating point number	fa0 = float to print	N/A
PrintDouble	3	Prints a double precision floating point number	fa0 = double to print	N/A
PrintString	4	Prints a null-terminated string to the console	a0 = the address of the string	N/A
ReadInt	5	Read an int from input console	N/A	a0 = the int
ReadFloat	6	Read a float from input console	N/A	fa0 = the float
ReadDouble	7	Reads a double from input console	N/A	fa0 = the double
ReadString	8	Reads a string from the console	a0 = address of input buffer a1 = maximum number of characters to read	N/A
Sbrk	9	Allocate heap memory	a0 = amount of memory in bytes	a0 = address to the allocated block
Exit	10	Exits the program with code 0	N/A	N/A
PrintChar	11	Prints an ascii character	a0 = character to print (only lowest byte is considered)	N/A
ReadChar	12	Read a character from input console	N/A	a0 = the character
Exit2	93	Exits the program with a code	a0 = the number to exit with	N/A

Chamadas ao sistema (ecall)

```
                .data               # diretiva p/ início do seg de dados
                .align 2            # alinha a memória para armazenar inteiro
vlr_inteiro:    .word 157
                .align 0            # alinha a memória para armazenar caractere
string:         .asciiz “Hello World"

                .text               # diretiva p/ início do segmento de texto
                .globl main         # diretiva p/ usar rotulo em outro prog.
main:                               # rótulo para ponto de entrada no processo
                .align 2            # alinha a memória para armazenar as instruções de 32 bits
                addi a7, x0, 4      # Código do serviço 4 (impressão de string)
                la a0, string       # Enderço do 1o byte da string
                ecall               # Chamada linux

                addi a7, x0, 10     # código do serviço que encerra
                ecall               # chamada linux para terminar o programa
```