O conjunto de instruções

Rodolfo Azevedo

MC404 - Organização Básica de Computadores e Linguagem de Montagem

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

Revisitando as instruções do processador

- Vocês já viram essas instruções/variações em sala ou no laboratório
- A meta dos slides abaixo é servir de referência geral mas você também pode utilizar o Reference Card disponível na página da disciplina que contém todas as instruções

Instruções de Load

Instrução	Formato	Descrição
LB rd, rs1, imm	I	Load Byte
LH rd, rs1, imm	I	Load Halfword
LW rd, rs1, imm	I	Load Word
LBU rd, rs1, imm	I	Load Byte Unsigned
LHU rd, rs1, imm	I	Load Halfword Unsigned

Instruções de Store

Instrução	Formato	Descrição
SB rs2, rs1, imm	S	Store Byte
SH rs2, rs1, imm	S	Store Halfword
SW rs2, rs1, imm	S	Store Word

Instruções Aritméticas

Instrução	Formato	Descrição
ADD rd, rs1, rs2	R	Add
ADDI rd, rs1, imm	I	Add Immediate
SUB rd, rs1, rs2	R	Subtract
LUI rd, imm	U	Load Upper Immediate
AUIPC rd, imm	U	Add Upper Immediate to PC

Instruções Lógicas

Instrução	Formato	Descrição
XOR rd, rs1, rs2	R	Exclusive OR
XORI rd, rs1, imm	I	Exclusive OR Immediate
OR rd, rs1, rs2	R	OR
ORI rd, rs1, imm	I	OR Immediate
AND rd, rs1, rs2	R	AND
ANDI rd, rs1, imm	I	AND Immediate

Instruções de Deslocamento

Instrução	Formato	Descrição
SLL rd, rs1, rs2	R	Shift Left Logical
SLLI rd, rs1, imm	I	Shift Left Logical Immediate
SRL rd, rs1, rs2	R	Shift Right Logical
SRLI rd, rs1, imm	I	Shift Right Logical Immediate
SRA rd, rs1, rs2	R	Shift Right Arithmetic
SRAI rd, rs1, imm	I	Shift Right Arithmetic Immediate

Instruções de Comparação

Instrução	Formato	Descrição
SLT rd, rs1, rs2	R	Set Less Than
SLTI rd, rs1, imm	I	Set Less Than Immediate
SLTU rd, rs1, rs2	R	Set Less Than Unsigned
SLTIU rd, rs1, imm	I	Set Less Than Immediate Unsigned

Instruções de Salto Condicionais

Instrução	Formato	Descrição
BEQ rs1, rs2, imm	SB	Branch if Equal
BNE rs1, rs2, imm	SB	Branch if Not Equal
BLT rs1, rs2, imm	SB	Branch if Less Than
BGE rs1, rs2, imm	SB	Branch if Greater Than or Equal
BLTU rs1, rs2, imm	SB	Branch if Less Than Unsigned
BGEU rs1, rs2, imm	SB	Branch if Greater Than or Equal Unsigned

Instruções de Salto Incondicional

Instrução	Formato	Descrição
JAL rd, imm	UJ	Jump and Link
JALR rd, rs1, imm	I	Jump and Link Register

Instruções de Sistema

Instrução	Formato	Descrição
ECALL	I	Environment Call
EBREAK	I	Environment Break

Multiplicação e Divisão (não funciona nesse simulador)

Instrução	Formato	Descrição
MUL rd, rs1, rs2	R	Multiply
DIV rd, rs1, rs2	R	Divide
DIVU rd, rs1, rs2	R	Divide Unsigned
REM rd, rs1, rs2	R	Remainder
REMU rd, rs1, rs2	R	Remainder Unsigned

Mais exercícios

Implemente as funções abaixo utilizando as convenções do RISC-V para chamada de funções:

```
int strlen(const char *str);
char *strcpy(char *destination, const char *source);
int strcmp(const char *str1, const char *str2);
char *strcat(char *destination, const char *source);
```

É sempre importante lembrar que as convenções precisam ser seguidas!

int strlen(const char *str)

int strlen(const char *str)

```
strlen:
    addi sp, sp, -4
    sw s0, sp, 0
    addi s0, zero, 0
strlen_loop:
    lbu t0, a0, 0
    beq t0, zero, strlen_end
    addi s0, s0, 1
    addi a0, a0, 1
    j strlen_loop
strlen_end:
    mv a0, s0
    lw s0, s0, 0
    addi sp, sp, 4
    ret
```

int strlen(const char *str) - implementação alternativa

```
strlen:
    addi t0, zero, 0
strlen_loop:
    lbu t0, a0, 0
    beq t1, zero, strlen_end
    addi t0, t0, 1
    addi a0, a0, 1
    j strlen_loop
strlen_end:
    mv a0, t0
    ret
```

char *strcpy(char *destination, const char *source)

char *strcpy(char *destination, const char *source)

```
strcpy:
    lbu t0, a1, 0
    beq t0, zero, strcpy_end
    sbu t0, a0, 0
    addi a0, a0, 1
    addi a1, a1, 1
    j strcpy
strcpy_end:
    ret
```

int strcmp(const char *str1, const char *str2)

int strcmp(const char *str1, const char *str2)

```
strcmp:
    lbu t0, a0, 0
    lbu t1, a1, 0
    bne t0, t1, strcmp_cmp
    beq t0, zero, strcmp_neg
    beq t1, zero, strcmp_pos
    addi a0, a0, 1
    addi a1, a1, 1
    j strcmp
strcmp_cmp:
    sub a0, t0, t1
    j strcmp_end
strcmp_neg:
    addi a0, zero, -1
    j strcmp_end
strcmp_pos:
    addi a0, zero, 1
strcmp_end:
    ret
```

char *strcat(char *destination, const char *source)

char *strcat(char *destination, const char *source)

```
strcat:
    addi t0, a0, 0
strcat_loop:
    lbu t1, a0, 0
    beq t1, zero, strcat_copy
    addi a0, a0, 1
    j strcat_loop
strcat_copy:
    lbu 1, a1, 0
    sbu t2, a0, 0
    beq t1, zero, strcat_end
    addi a0, a0, 1
    addi a1, a1, 1
    j strcat_copy
strcat_end:
    addi a0, t0, 0
    ret
```