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

Revisitando instruções lógicas

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

Revisitando instruções de deslocamento

Instrução	Formato	Uso	
Shift Left	R	SLL	rd, rs1, rs2
Shift Left Immediate	I	SLLI	rd, rs1, shamt
Shift Right	R	SRL	rd, rs1, rs2
Shift Right Immediate	I	SRLI	rd, rs1, shamt
Shift Right Arithmetic	R	SRA	rd, rs1, rs2
Shift Right Arith Imm	I	SRAI	rd, rs1, shamt

Revisitando instruções aritméticas

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

Exemplos

Recaptulando

- Setar bit (transformar em 1) → OR
- Zerar bit (transformar em 0) → AND
- Selecionar bits → AND
- Inverter bits → XOR
- Utilize as instruções de deslocamento para colocar os valores nos bits corretos

Representação de Caracteres

- Codificações de caracteres
  - ASCII, ISO 8859-1 (Latin-1), UTF
- Codificações de strings
  - Terminando com caracter \0
  - Tamanho + string
- Armazenamento em memória
- Operações comuns com strings
  - strlen, strcpy, strcmp, strcat

ASCII

- Somente os 7 bits menos significativos

0	00	NUL	28	1C	FS	56	38	8	84	54	T	112	70	p
1	01	SOH	29	1D	GS	57	39	9	85	55	U	113	71	q
2	02	STX	30	1E	RS	58	3A	:	86	56	V	114	72	r
3	03	ETX	31	1F	US	59	3B	;	87	57	W	115	73	s
4	04	EOT	32	20	space	60	3C	<	88	58	X	116	74	t
5	05	ENQ	33	21	!	61	3D	=	89	59	Y	117	75	u
6	06	ACK	34	22	"	62	3E	>	90	5A	Z	118	76	v
7	07	BEL	35	23	#	63	3F	?	91	5B	[	119	77	w
8	08	BS	36	24	$	64	40	@	92	5C	\	120	78	x
9	09	HT	37	25	%	65	41	A	93	5D	]	121	79	y
10	0A	LF	38	26	&	66	42	B	94	5E	^	122	7A	z
11	0B	VT	39	27	'	67	43	C	95	5F	_	123	7B	{
12	0C	FF	40	28	(	68	44	D	96	60	`	124	7C	|
13	0D	CR	41	29	)	69	45	E	97	61	a	125	7D	}
14	0E	SO	42	2A	*	70	46	F	98	62	b	126	7E	~
15	0F	SI	43	2B	+	71	47	G	99	63	c	127	7F	DEL
16	10	DLE	44	2C	,	72	48	H	100	64	d			
17	11	DC1	45	2D	-	73	49	I	101	65	e			
18	12	DC2	46	2E	.	74	4A	J	102	66	f			
19	13	DC3	47	2F	/	75	4B	K	103	67	g			
20	14	DC4	48	30	0	76	4C	L	104	68	h			
21	15	NAK	49	31	1	77	4D	M	105	69	i			
22	16	SYN	50	32	2	78	4E	N	106	6A	j			
23	17	ETB	51	33	3	79	4F	O	107	6B	k			
24	18	CAN	52	34	4	80	50	P	108	6C	l			
25	19	EM	53	35	5	81	51	Q	109	6D	m			
26	1A	SUB	54	36	6	82	52	R	110	6E	n			
27	1B	ESC	55	37	7	83	53	S	111	6F	o			

Outros formatos

- ISO 8859-1 ou Latin-1
  - Codifica os caracteres de 128 até 255
- UTF
  - Cada caracter é representado em um codepoint
  - Codepoints são representados em memória de acordo com a codificação UTF selecionada
  - UTF-8, UTF-16, …
- https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know-about-unicode-and-character-sets-no-excuses

Codificando strings

- Formato mais comum
  - Toda string termina com um \0
  - “Ola\0”
- Formato alternativo
  - Toda string tem seu tamanho no primeiro byte
  - 3“Ola”
- Como elas são representadas em memória?

Instruções de Memória

Instrução	Formato	Uso	
Load Byte	I	LB	rd, rs1, imm
Load Halfword	I	LH	rd, rs1, imm
Load Word	I	LW	rd, rs1, imm
Load Byte Unsigned	I	LBU	rd, rs1, imm
Load Half Unsigned	I	LHU	rd, rs1, imm
Store Byte	S	SB	rs1, rs2, imm
Store Halfword	S	SH	rs1, rs2, imm
Store Word	S	SW	rs1, rs2, imm

Operações Comuns com Strings

- 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);

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)

Primeiras chamadas de sistema

- Dependente da implementação do Sistema Operacional
- No nosso simulador
  - Coloque o número da systemcall no registrador t0
  - Coloque os argumentos em a0, a1, etc
  - Execute a instrução ecall
  - Se houver valor de retorno, estará em a0

Chamadas de Sistema

Syscall	**id (coloque em t0)**	**Descrição**
Imprime inteiro	1	Imprime o valor de a0 no console como inteiro
Imprime caracter	2	Imprime o valor de a0 no console como caracter
Imprime string	3	Imprime string a0 com tamanho a1 no console
Lê inteiro	4	Lê inteiro do console e retorna em a0
Lê caracter	5	Lê caracter do console e retorna o valor ASCII
Lê string	6	Lê string do tamanho solicitado em a1 e armazena no endereço indicado em a0
SBRK	7	Aloca a0 bytes de memória e retorna ponteiro para o bloco de memória. Desalocar memória com a0 negativo.

Exemplo (do simulador)

```
main:
    # let's start by loading an integer from the user
    # putting 4 in register t0 an calling ecall does that
    addi t0, zero, 4
    # after ecall is run, a box will pop up in the console
    # write a (small) number and press enter to continue
    ecall
    # let's print that integer multiplied by two
    slli a0, a0, 1
    addi t0, zero, 1
    ecall
    # now, let's try working with strings and memory
    # read a character
    addi t0, zero, 5
    ecall
    # print the same character and a newline
    addi t0, zero, 2
    ecall
    addi t0, zero, 2
    addi a0, zero, 13
    ecall
```

Imprimindo string

```
    .rodata
.HELLO:
    .word 0x4C4C4548
    .word 0x0000214F
    .text
    # print the string "HELLO!\n"
    addi t0, zero, 3          # this is the string printing syscall
    lui a0, %hi(.HELLO)       # this loads the top 20 bits
                              # of .HELLO address into a0
    addi a0, a0, %lo(.HELLO) # this loads the bottom 12 bits
    addi a1, zero, 7          # length of the string
    ecall
    # print characters '!', '\n', '-'
    addi t0, zero, 2
    addi a0, zero, 33
    ecall
    addi t0, zero, 2
    addi a0, zero, 13
    ecall
    addi t0, zero, 2
    addi a0, zero, 45
    ecall
```

Lendo string

```
# load a string of length 10
addi t0, zero, 6
addi a0, sp, -10
addi a1, zero, 10
ecall
# print the same string again
addi t0, zero, 3
addi a0, sp, -10  # address of the string start
addi a1, zero, 10  # length of the string
ecall
```

Alocação de Memória

```
# finally, let's try and allocate some memory with SBRK
addi t0, zero, 7      # SBRK syscall
addi a0, zero, 16     # allocate 4 words
ecall                 # heap memory (purple) should be
                      # 4 lines long in the memory pane
# try and deallocate some memory with SBRK
addi t0, zero, 7      # SBRK syscall
addi a0, zero, -16    # deallocate 4 words
ecall                 # heap memory (purple) should be
                      # 0 lines long in the memory pane
# allocate too many words
addi t0, zero, 7      # SBRK syscall
addi a0, zero, 10000  # allocate 2500 words
ecall                 # an error should pop up
```