Universidade Federal de Pelotas
Centro de Desenvolvimento Tecnológico
Bacharelado em Ciência da Computação
Engenharia de Computação

Arquitetura e Organização de Computadores I

Prática

Aula 7

Assembly Estendido, Strings e Vetores

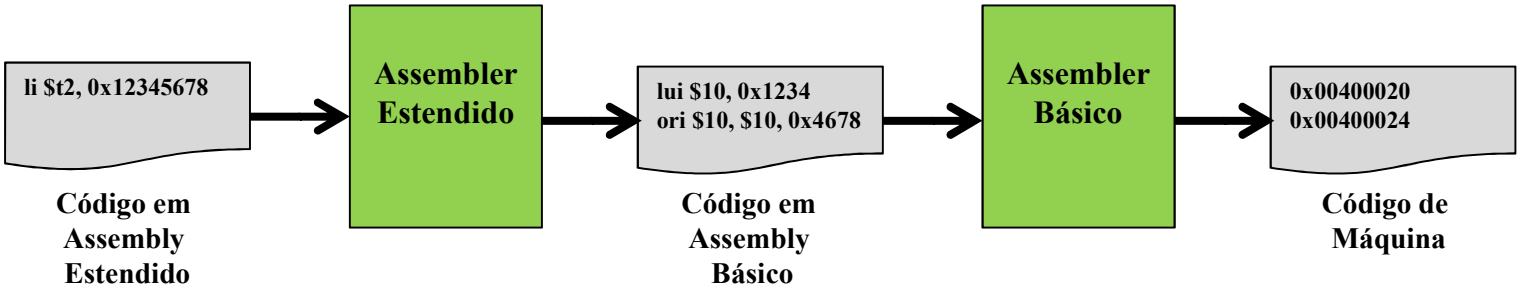
Prof. Guilherme Corrêa
gcorrea@inf.ufpel.edu.br

Prof. Bruno Zatt

Assembly Estendido

- Acrescenta instruções ao assembly básico;
- Instruções denominadas **pseudo-instruções**;
- Tem o objetivo de **simplificar** a vida do programador, estendendo as potencialidades da máquina original;
- Pode ser visto como uma **máquina virtual**, com um conjunto de instruções mais completo.

Assembly Estendido



- **Pseudo-instruções** são traduzidas pelo assembler estendido em uma ou mais **instruções assembly**;
- Instruções assembly têm **equivalência direta** para instruções binárias, que podem ser executadas diretamente pelo processador;
- Pseudo-instruções precisam ser **traduzidas** para assembly.

Assembly Estendido

Pseudo-instrução `move`

- Atribui o valor de um registrador a outro

  Exemplo: `move $t0, $s0`

  Atribui (copia) o valor de **$s0** para **$t0**.

  É traduzido para o assembly básico como:

  `addu $8, $0, $16`

Assembly Estendido

Pseudo-instrução `li`

- Carrega no registrador destino o valor inteiro positivo ou negativo de até 32 bits

Exemplo: `li $t0, 125`

Atribui o valor **125** para **$t0**.

Pode ser traduzido para o assembly básico de diferentes formas, dependendo do valor:

Pseudo-Intrução	Tradução
`li $t0,125`	`ori $8,$0,125`
`li $t1,-39`	`addiu $9,$0,-39`
`li $t2,0x12345678`	`lui $10,0x1234` `ori $10,$10,0x4678`

Assembly Estendido

Pseudo-instruções para acesso à memória

`lw rd, endereco1`	`# carrega rd com um valor` `# armazenado em endereco1`
`la rd, endereco2`	`# carrega rd com o endereço` `# de memoria de endereco2`
`sw rd, endereco3`	`# escreve conteúdo de rd no` `# endereço simbolico endereco3`

Exemplo:

```
        .text
        lw $t0, endereco1
        la $t1, endereco2
        sw $t2, endereco3

        .data
        endereco1: .word 5
        endereco2: .word 10
        endereco3: .word
```

Assembly Estendido

Registrador *Assembler Temporary* ($at)

Registrador	Nome	Uso (convenção)
$0	$zero	Zero
$1	**$at**	***Assembler Temporary***
$2, $3	$v0, $v1	Valor de retorno de subrotina
$4 – $7	$a0 – $a3	Argumentos de subrotina
$8 – $15	$t0 – $t7	Temporários (locais à função)
$16 – $23	$s0 – $s7	Salvos (não alterados na função)
$24, $25	$t8, $t9	Temporários
$26, $27	$k0, $k1	Kernel (reservado para SO)
$28	$gp	*Global Pointer*
$29	$sp	*Stack Pointer*
$30	$fp	*Frame Pointer*
$31	$ra	Endereço de Retorno

Assembly Estendido

Registrador *Assembler Temporary* ($at)

- Usado quando uma pseudo-instrução requer valores **intermediários** em sua tradução;
- No **MARS**, o registrador **$1** (ou **$at**) **não deve ser usado** pelo programador quando se usam pseudo-instruções.

Exemplo:

Pseudo-Intrução	Tradução
`la $t0, endereco`	`lui $1,0x00001001 #base` `ori $8,$1,0x00000008 #offset`
`lw $t0, endereco`	`lui $1,0x00001001 #base` `lw $8,0x00000008($1) #offset`

Assembly Estendido

O que faz?

```
##
## soma dois valores (val2 + val3)
##
   .text
   .globl main
main:
   la $t0,val2                # $t0 = endereço de val2
   lw $t1,0($t0)              # $t1 = val2
   lw $t2,4($t0)              # $t2 = val3
   addu $t1,$t1,$t2           # $t1 = val2 + val3
.data
   val0: .word 0
   val1: .word 1
   val2: .word 2
   val3: .word 3
   val4: .word 4
   val5: .word 5
```

Strings

- As **strings** são escritas em memória, em **bytes contíguos**;
- Cada caractere é representado em um byte;
- Endereços são **alinhados por byte**;
- Codificação **ASCII**;
- Definimos as strings no código assembly **entre aspas** após **.asciiz** na seção de dados da memória.

Exemplo:

```
.data
texto: .asciiz "Albus Dumbledore"
```

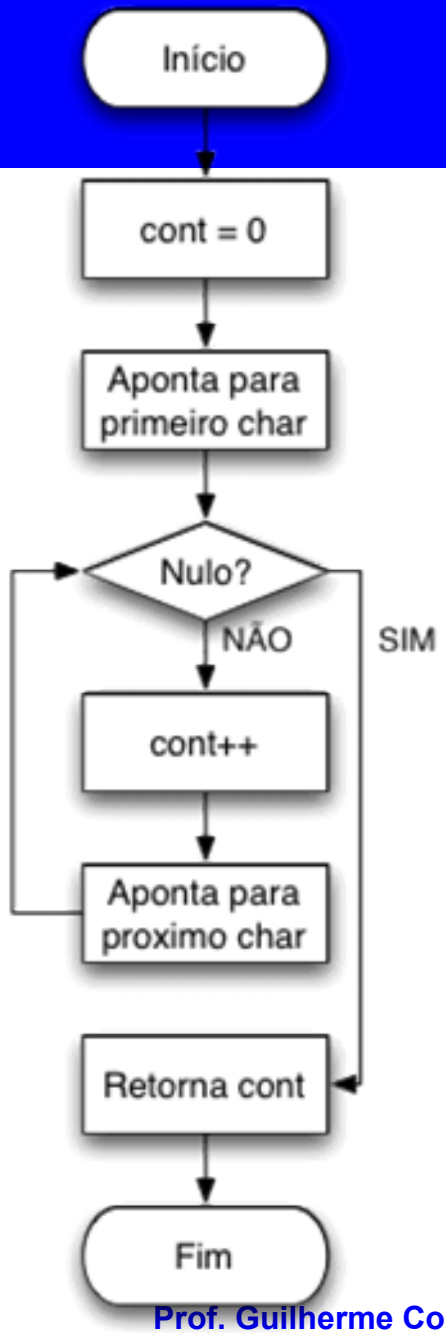
Tabela ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Strings

Exemplo: função strlen()

```
## strlen.asm
##
## Conta os caracteres de uma string
##
## Registradores:
##  $8 -- cont
##  $9 -- ponteiro para o char
## $10 -- caracter atual (no byte menos significativo)

        .text
        .globl  main

# Inicialização
main:   ori     $8,$0,0         #  cont = 0
        lui     $9,0x1000       #  ponteiro para primeiro char

# enquanto ch != null faça
loop:   lbu     $10,0($9)       # carrega o char
        sll     $0,$0,0         # delay slot

        beq     $10,$0,fim      # Nulo?
        sll     $0,$0,0         # delay slot

        addi    $8,$8,1         # cont++
        addi    $9,$9,1         # ponteiro++

        j       loop
        sll     $0,$0,0         # delay slot

# fim (resultado em $8)
fim:    sll     $0,$0,0

        .data
string: .asciiz  "Ser ou nao ser?"
```



Vetores

- O tamanho do vetor é definido por um inteiro (*word*);
- O vetor é inicializado com valores separados por vírgulas;
- Os endereços são alinhados por *words*.

Exemplo:

```
.data
  tamanho:  .word   12
  vetor:    .word   4,-2,33,52,1,17,11,7,90,-7,8,-13
```

Vetores

- **Exemplo:**

  Soma valores positivos e negativos de um vetor em dois acumuladores diferentes.