

🔒

raqueleucaria / FAC

Private

<> Code

🔍 Issues

🔗 Pull requests

⚙️ Zenhub

▶️ Actions

📁 Projects

📖 Wiki

🛡️ Security


📈 Insights

⚙️ Settings

🔗 main

...

FAC / Aulas / 6\_Representacao.md

 raqueleucaria Aulas+teste

🕒 History

👤 1 contributor

☰

170 lines (125 sloc) | 5.06 KB

...

# Aula 6 - Representação em linguagem de máquina

(02/12)

- Formato das instruções (R e I)
- Instruções lógicas e de deslocamento

## Representação em linguagem de máquina

- Todas as instruções são traduzidas para binário pelo montador (assembler).
- Os códigos binários gerados são chamados de linguagem/código de máquina.
- Instruções MIPS --> codificadas como palavras de 32 bits
- A conversão é pautada em 3 formatos de representação: tipo R, I, J

## Formato R

- Formato mais padrão
- A instrução do tipo R são representadas num binário de 32 bits segregdos pela seguinte forma:

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- **op**: Código da operação (opcode) - Identifica a instrução do tipo I
  - **rs**: Número do 1º registragor de origem
  - **rt**: Número do 2º registragor de origem
  - **rd**: Número do registragor de destino
  - **shamt**: Tamanho do deslocamento - Quantidade de shift (00000 for now)
  - **funct**: Código da função (complementa o opcode) - Diz qual a operação que vai ser executada
- São instruções do tipo R: Aritméticas, lógicas e deslocamentos.

Ex: add St0, \$s1, \$s2

Sign in now to use ZenHub

special	\$s1	\$s2	\$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

= 00000010001100100100100000000100000 = 02324020<sub>16</sub>

- Modelo de computação atual: Um dado e um programa tem a mesma forma

## Obs: PROVA

- Por que os registradores estão com 5 bits?
- POIS, a arquitetura cabe 32 bits =  $2^5$

## Hexadecimal

### ■ Base 16

- Representação compacta para binários
- 4 bits por dígito hexadecimal

0	0000	4	0100	8	1000	c	1100
1	0001	5	0101	9	1001	d	1101
2	0010	6	0110	a	1010	e	1110
3	0011	7	0111	b	1011	f	1111

### ■ Exemplo: eca8 6420

- Hexadecimal
  - 1110 1100 1010 1000 0110 0100 0010 0000

## Formato I

- São representadas num binário de 32 bits da seguinte forma:

op	rs	rt	const. ou address
6bits	5 bits	5 bits	16 bits

- **op**: Código da operação (opcode)
- **rs**: Registrador de origem
- **rt**: Registrador de destino (ou origem para sw)
- Constante:  $-2^{15}$  to  $+2^{15}-1$
- Endereç: offset/deslocamento adicionado ao endereço base em rs
- São instruções do tipo I: Imediatas e de acesso à memória
- **OBS**: A capacidade máxima de uma constante é de  $-2^{15}$  a  $2^{15} - 1$

```

sw $t0 0($s0)
  |  |  |
  rt |  rt
  const.

```

## PROVA

- Montador: pega a instrução e converte para binário -> reversível (1:1)
- COMPILAÇÃO NÃO É REVERSÍVEL (1:N) -> várias formas de compilar

- Vetor x programa (na memória) -> iguais
  - Programa é um vetor de instruções
  - na memória as instruções e os dados são salvos igualmente
  - é possível fazer operações com programas

## Operações lógicas

São insstruções para manipulação de bits

Operation	C	Java	MIPS
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bitwise AND	&	&	and, andi
Bitwise OR	"	"	"
Bitwise NOT	~	~	nor

- Úteis para inserir ou extrair bits numa palavra

## Operações de shift (deslocamento)

- Do tipo R
- Formas muito rápidas de fazer mult ou div por 2
- exceção ao Tipo i, por usar uma constante, mas a constante entra no shamt
- **shamt**: Quantas posições deslocar
- Shift à esquerda (**sll** - shift left logical)
  - Desloca à esquerda e preenche com zero a direita --> multiplica por 2
  - sll *i* bits multiplica por  $2^i$
- Shift à direita (**srl** - shift right logical)
  - Desloca à direita e preenche com zero a esquerda --> divide por 2
  - srl *i* bits divide por  $2^i$  (APENAS SEM SINAL)
- Exemplo:

## Operações AND

- Útil para usar como máscara -> extrair bits de uma palavra
  - Seleciona alguns bits define os demais como zero
  - Extrair o bit mais significativo
  - AND: vai bit a bit o que for 1 e 1 da resultado 1

Exemplo:

```
0010 1010 | -> e
1011 0110 |
-----
0010 0010
```

and \$t0, \$t1, \$t2 # armazena em t0 o mais significativo

## Operações OR

- Útil para incluir bits numa palavra
  - Seleciona alguns bits define os demais como zero
  - Exemplo: Extrair o bit mais significativo
  - AND: vai bit a bit o que for 1 e 1 da resultado 1
- Definir como 1 um bit zero de uma palavra

Exemplo:

0010 1010 |-> ou 1011 0110 |

1011 1110

or \$t0, \$t1, \$t2 # armazena em t0 o mais significativo

## Operações NOT

- Útil para inverter os bits numa palavra
  - Muda 0 para 1 e 1 para 0
  - Não existe o NOT, possui o NOR
  - MIPS possui a instrução tipo R NOR
    - $a \text{ NOR } b == \text{NOT}(a \text{ OR } b)$
  - Para fazer o NOT basta apenas fazer o nor com zero

nor \$t0, \$t1, \$zero

## Importante

- 1 Uma instrução também é uma palavra.
- 2 Por serem palavras os dados (arquitetura de Von Neumann). Isso é chamado de programa armazenado.

Observação: Uma palavra pode representar inteiros de  $-2^{31}$  a  $2^{31} - 1$ , ou -2.147.483.648 a 2.147.483.647. O campo contante, de  $-2^{15}$  a  $2^{15} - 1$ , ou -32.768 a 32.767

Para representarmos o 0 tiramos 1 na parte positiva, por isso do "-1".