04)

1a Geração – Válvulas

2a Geração - Transistores

3a Geração – Circuitos Integrados

4a Geração – Circuitos VLSI

07)

A Arquitetura de von Neumann (de John von Neumann, pronunciado Nóimánn) é uma arquitetura de computador que se caracteriza pela possibilidade de uma máquina digital armazenar seus programas no mesmo espaço de memória que os dados, podendo assim

21)

John von Neumann propôs que as instruções, lidas na época por cartões perfurados, fossem gravadas na memória do computador; o que faria sua execução e leitura mais rápidas, uma vez que se davam eletronicamente.

A Arquitetura de von Neumann (de [John von Neumann](https://pt.wikipedia.org/wiki/John_von_Neumann), pronunciado Nóimánn) é uma arquitetura de computador que se caracteriza pela possibilidade de uma máquina digital armazenar seus [programas](https://pt.wikipedia.org/wiki/Programa_de_computador) no mesmo espaço de [memória](https://pt.wikipedia.org/wiki/Mem%C3%B3ria_(computador)) que os dados, podendo assim manipular tais programas. Esta arquitetura é um projeto modelo de um computador digital de programa armazenado que utiliza uma unidade de processamento ([CPU](https://pt.wikipedia.org/wiki/CPU)) e uma de armazenamento ("memória") para comportar, respectivamente, instruções e dados.

A máquina proposta por Von Neumann reúne os seguintes componentes:

1. Uma [memória](https://pt.wikipedia.org/wiki/Mem%C3%B3ria)
2. Uma [unidade aritmética e lógica](https://pt.wikipedia.org/wiki/Unidade_aritm%C3%A9tica_e_l%C3%B3gica) (ALU)
3. Uma [unidade central de processamento](https://pt.wikipedia.org/wiki/Unidade_central_de_processamento) (CPU), composta por diversos [registradores](https://pt.wikipedia.org/wiki/Registrador), e
4. Uma Unidade de Controle (CU), cuja função é a mesma da tabela de controle da [Máquina de Turing universal](https://pt.wikipedia.org/wiki/M%C3%A1quina_de_Turing_universal): buscar um programa na memória, instrução por instrução, e executá-lo sobre os [dados](https://pt.wikipedia.org/wiki/Dados) de [entrada](https://pt.wikipedia.org/wiki/Entrada/sa%C3%ADda).

Todos os elementos dessa arquitetura são alinhados da estrutura hardware do CPU, assim o sistema pode realizar todas as suas atividades sem apresentar erros no desempenho. Von Neumann é continuamente influenciados pela evolução tecnológica, tendo peças mais modernas inseridas. Cada um dos elementos apresentados é realizado à custa de componentes físicos independentes, cuja implementação tem variado ao longo do tempo, consoante a evolução das tecnologias de fabricação.

27)

Aritméticas ℕ e ℤ , lógicas, controle de fluxo, aritméticas ℝ, acesso à memória, traps, co-processador.

37)

R opcode (6) rs (5) rt (5) rd (5) shamt (5) funct (6)

I opcode (6) rs (5) rt (5) imediato (16)

J opcode (6) endereço (26

|  |  |  |
| --- | --- | --- |
| **Nome** | **Sintaxe da Instrução** | **Significado** |
| Add | add $1,$v0,$3 | $1 = $v0 + $3 (signed) | R |
| Subtract | sub $1,$v0,$3 | $1 = $v0 – $3 (signed) | R |
| Add immediate | addi $1,$v0,CONST | $1 = $v0 + CONST (signed) | I |
| Multiply | mult $1,$v0 | LO = (($1 \* $v0) << 32) >> 32; HI = ($1 \* $v0) >> 32; | R |
| Divide | div $1, $v0 | LO = $1 / $v0     HI = $1 % $v0 | R |
| Load word | lw $1,CONST($v0) | $1 = Memory[$v0 + CONST] | I |
| Load halfword | lh $1,CONST($v0) | $1 = Memory[$v0 + CONST] (signed) | I |
| Load byte | lb $1,CONST($v0) | $1 = Memory[$v0 + CONST] (signed) | I |
| Store word | sw $1,CONST($v0) | Memory[$v0 + CONST] = $1 | I |
| Store half | sh $1,CONST($v0) | Memory[$v0 + CONST] = $1 | I |
| Store byte | sb $1,CONST($v0) | Memory[$v0 + CONST] = $1 | I |
| And | and $1,$v0,$3 | $1 = $v0 & $3 | R |
| And immediate | andi $1,$v0,CONST | $1 = $v0 & CONST | I |
| Or | or $1,$v0,$3 | $1 = $v0 | $3 | R |
| Or immediate | ori $1,$v0,CONST | $1 = $v0 | CONST | I |
| Exclusive or | xor $1,$v0,$3 | $1 = $v0 ^ $3 | R |
| Nor | nor $1,$v0,$3 | $1 = ~($v0 | $3) | R |
| Set on less than | slt $1,$v0,$3 | $1 = ($v0 < $3) | R |
| Set on less than immediate | slti $1,$v0,CONST | $1 = ($v0 < CONST) | I |
| Branch on equal | beq $1,$v0,CONST | if ($1 == $v0) go to PC+4+CONST | I |
| Branch on not equal | bne $1,$v0,CONST | if ($1 != $v0) go to PC+4+CONST | I |
| Jump | j CONST | goto address CONST | J |
| Jump register | jr $1 | goto address $1 | R |
| Jump and link | jal CONST | $31 = PC + 4; goto CONST | J |

40)

Estes são registros especiais usados para armazenar resultado da multiplicação e divisão. Eles não são diretamente endereçável. Seu conteúdo é acessado com mfhi e mflo.

divu s,t lo <-- s div t ; hi <-- s mod t

multu s,t hi / lo < -- s \* t ;

mfhi $v0

mflo $v1

44)

|  |  |  |
| --- | --- | --- |
| And | and $1,$2,$3 | R |
| And immediate | andi $1,$2,CONST | I |
| Or | or $1,$2,$3 | R |
| Or immediate | ori $1,$2,CONST | I |
| Exclusive or | xor $1,$2,$3 | R |
| Nor | nor $1,$2,$3 | R |
| Set on less than | slt $1,$2,$3 | R |
| Set on less than immediate | slti $1,$2,CONST | I |

46)

add $s0,$zero,$zero

li $s0,0x00000000

add $s1,$zero,42

add $s2,$zero,10

add $s3,$zero,$s2

add $s4,$zero,$s4

add $t1,$s0,$s1

add $t2,$s2,$s3

add $t3,$t2,$s4

add $s5,$t1,$t3

mul $t4,$s0,$s1

mul $t5,$s2,$s3

mul $t6,$s4,$s5

add $t7,$t4,$t5

add $s6,$t6,$t7

lw $3,4($fp)

lw $2,8($fp)

teq $2,$0,7

div $0,$3,$2

mfhi $2

mflo $4

lw $3,4($fp)

lw $2,12($fp)

teq $2,$0,7

div $0,$3,$2

mfhi $2

mflo $2

subu $2,$4,$2

lw $4,4($fp)

lw $3,16($fp)

teq $3,$0,7

div $0,$4,$3

mfhi $3

mflo $3

addu $2,$2,$3

sw $2,20($fp)

add $s1,$s1,1

add $s2,$s2,42

mul $s3,$s3,2

add $s2,$zero,10

add $s0,$zero,0x00ff

add $s1,$zero,0x7777

54)

addiu $sp,$sp,-8

sw $fp,4($sp)

move $fp,$sp

sw $a0,8($fp)

# num>=0

lw $v0,8($fp)

bltz $v0,$L2

nop

#num <=127

lw $v0,8($fp)

slt $v0,$v0,128

beq $v0,$zero,$L2

nop

#return 1

li $v0,1 # 0x1

b $L3

nop

$L2: #return 0

move $v0,$zero

$L3:

move $sp,$fp

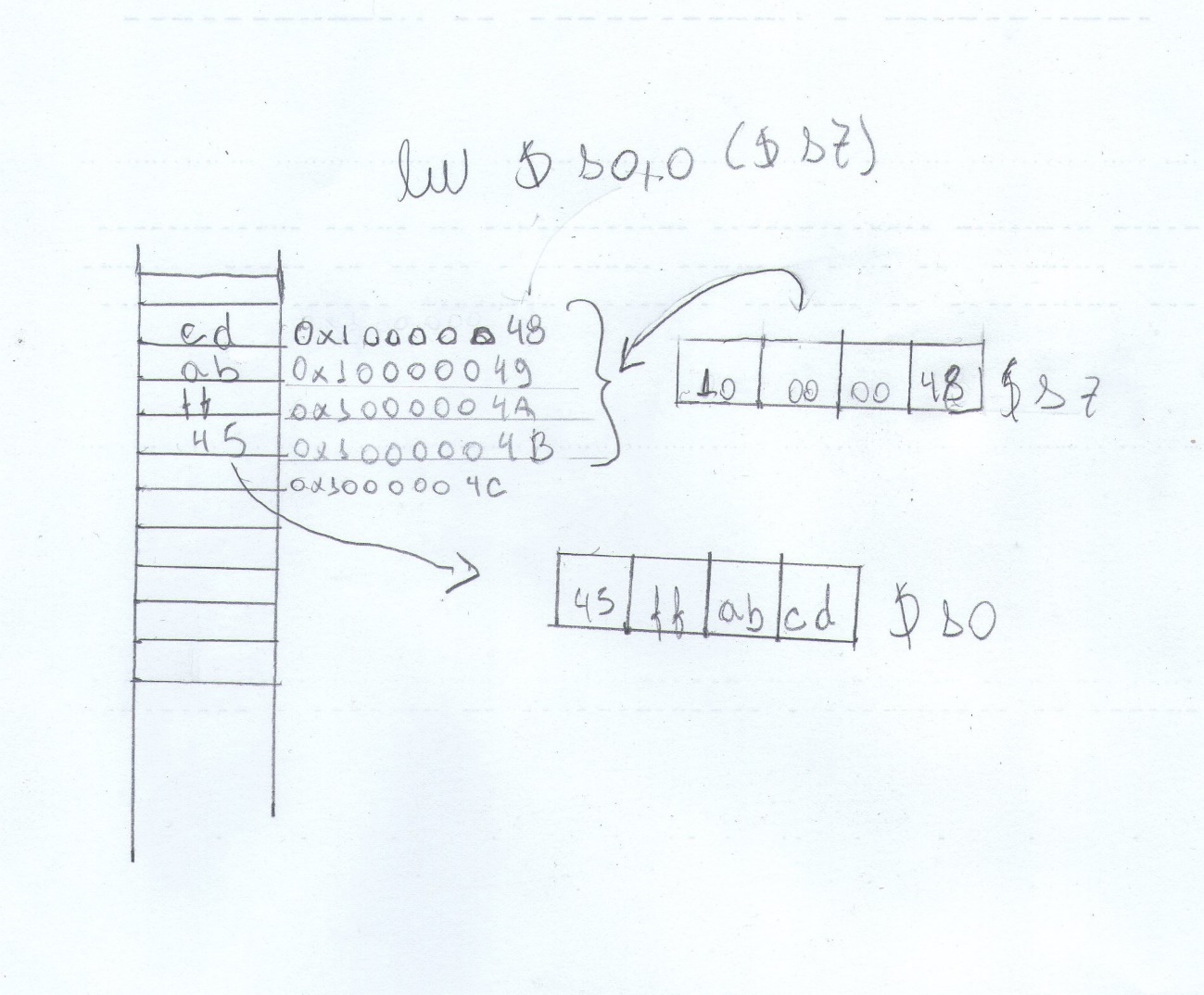
lw $fp,4($sp)

addiu $sp,$sp,8

j $31

nop

65)



69)

a)

sra $t1,$t0,31

xor $t0,$t0,$t1

sub $t0,$t0,$t1

b)

beq $zero,$zero,Label

c)

add $t1,$zero,42

beq $t0, $t1, LABEL

d)

div $s, $t

mflo $d

e)

lui $at, upper 16 bits

ori $t2, $at, lower 16 bits

f)

lui $at, upper 16 bits

ori $t2, $at, lower 16 bits

g)

add $rt,$rs,$zero

h)

sub $t1,$zero,$t0

bgez $t0,skip

sub $t1,$zero,$t0

skip:

i)

nor $rt, $rs, $zero

j)

div $s, $t

mfhi $d

k)

lui $at, upper 16 bits

ori $t2, $at, lower 16 bits

78)

.include "utils.asm"

.data

.align 4

pt: .word

str1: .asciiz "Digite os 2 nums"

.text

la $t0, str1

ImprimeString($t0)

LeInteiro

addu $s0, $zero, $v0 #a lido

LeInteiro

addu $s1, $zero, $v0 #b lido

M: add $s2, $zero, $zero # r = 0

add $s3, $zero, $s0 # i = a

FOR1: slt $t1, $zero, $s3 # 0 < i

beq $t1, $zero, SAI1

#----------------------

add $s2, $s2,$s1 #r+=b

#----------------------

addi $s3, $s3, -1 # i++

j FOR1

#-------------------------

SAI1: sw $t7, 0($s2)

done

###################################################################################################################################

la $t0, str1

ImprimeString($t0)

LeInteiro

addu $s4, $zero, $v0 #b lido

LeInteiro

addu $s5, $zero, $v0 #exp lido

add $s6, $zero, $s4 # r = b

add $s7, $zero, $s5 # i = exp

add $t5,$zero,1

FOR2: slt $t1, $s7, $t5 # 1 < i

beq $t1, $zero, SAI2

#----------------------

add $s0, $zero,$s6 #k=m(r,b)

add $s1, $zero, $s4

beq $t4, $zero, M

#----------------------

addi $s7, $s7, -1 # i--

j FOR2

#-------------------------

SAI2: sw $t7, 0($s2)

done

97)

75 segundos