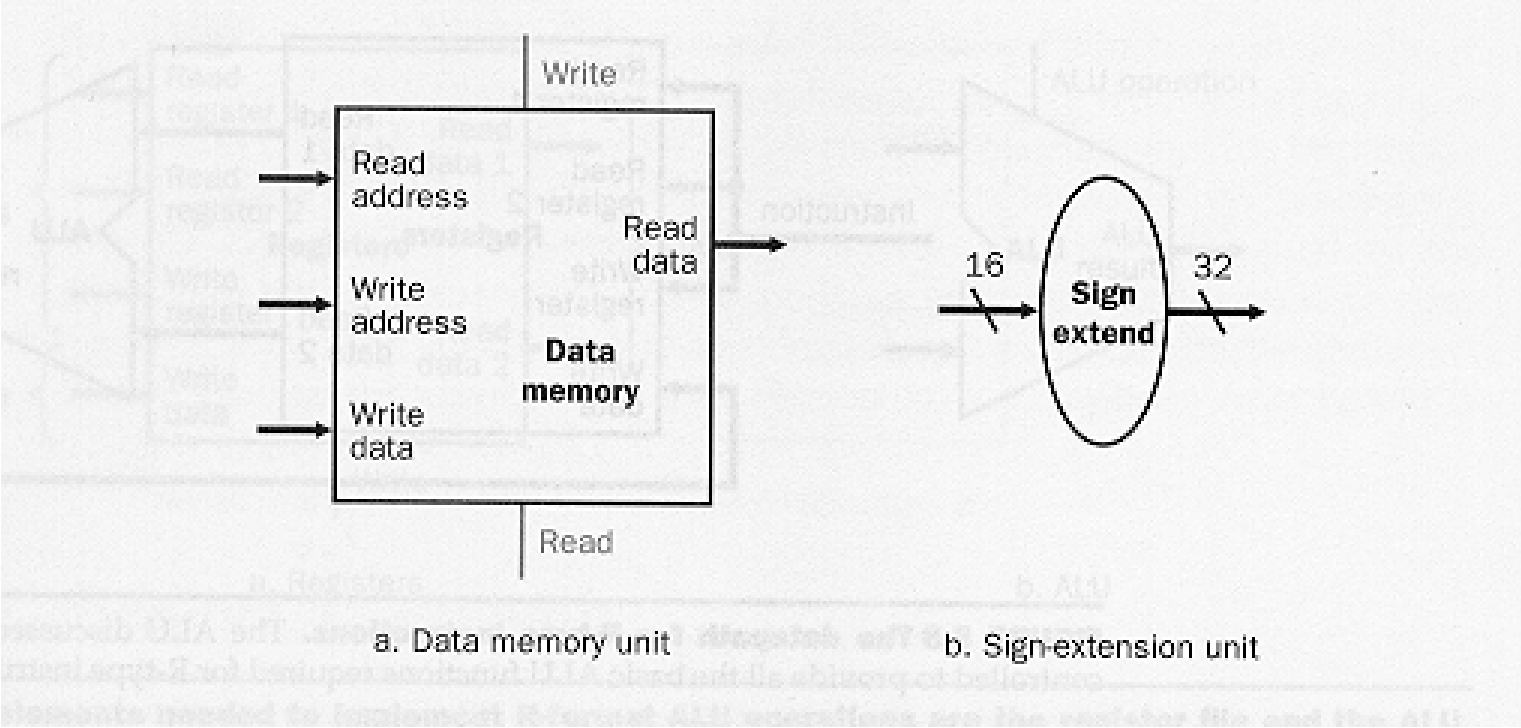
Instruções de Acesso à Memória e de Desvio Condicional

Prof. Alberto F. De Souza
LCAD/DI/UFES
sp1@lcad.inf.ufes.br

lcad
LABORATÓRIO DE COMPUTAÇÃO DE ALTO DESEMPENHO

Instruções de Acesso à Memória

- Para implementar as instruções de leitura e escrita na memória, precisamos dos componentes abaixo:

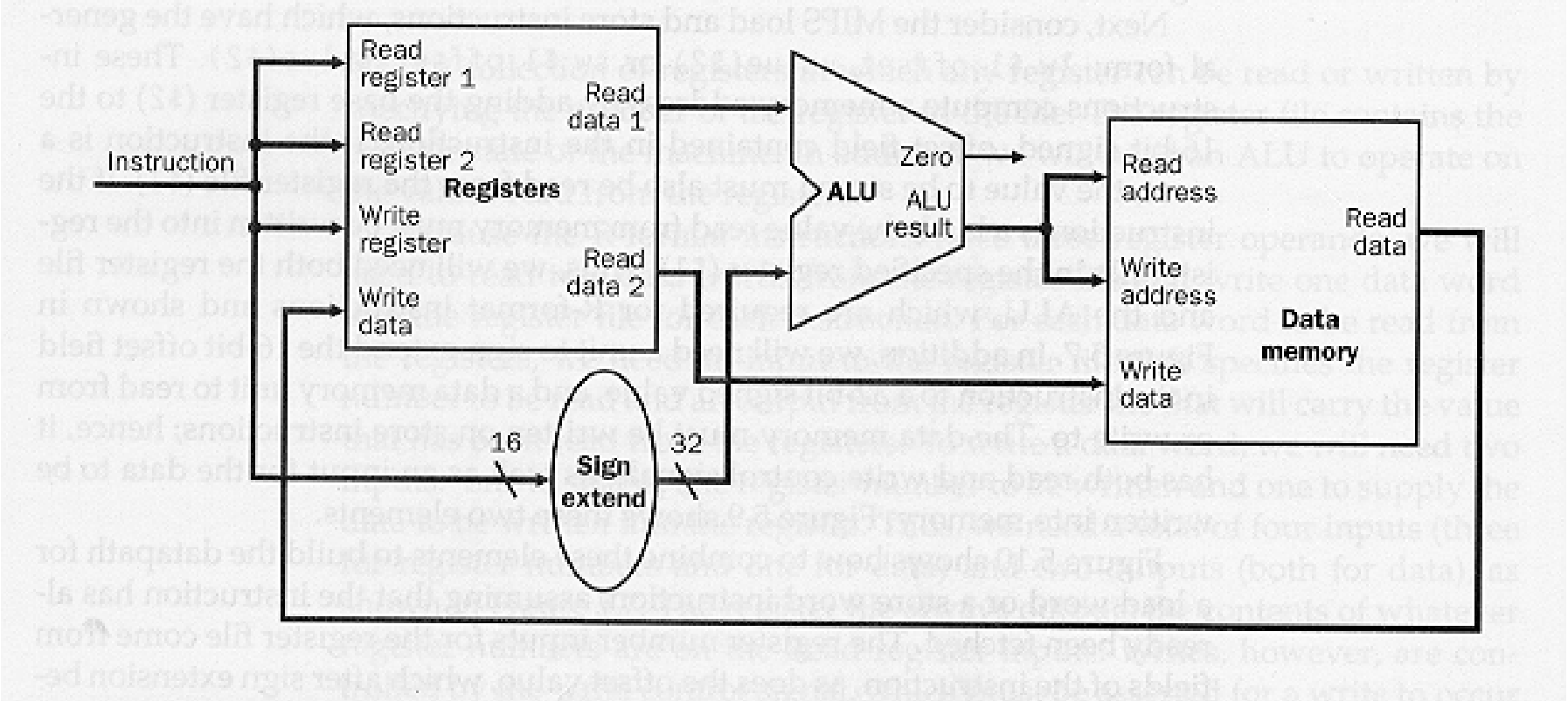


a. Data memory unit

b. Sign-extension unit

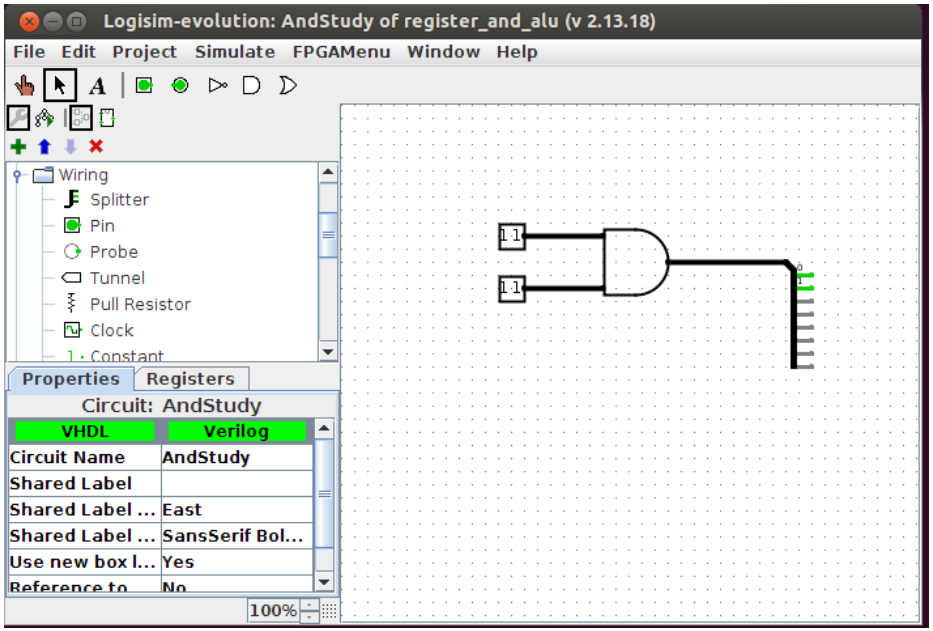
Instruções de Acesso à Memória

- Eles podem ser organizados como abaixo



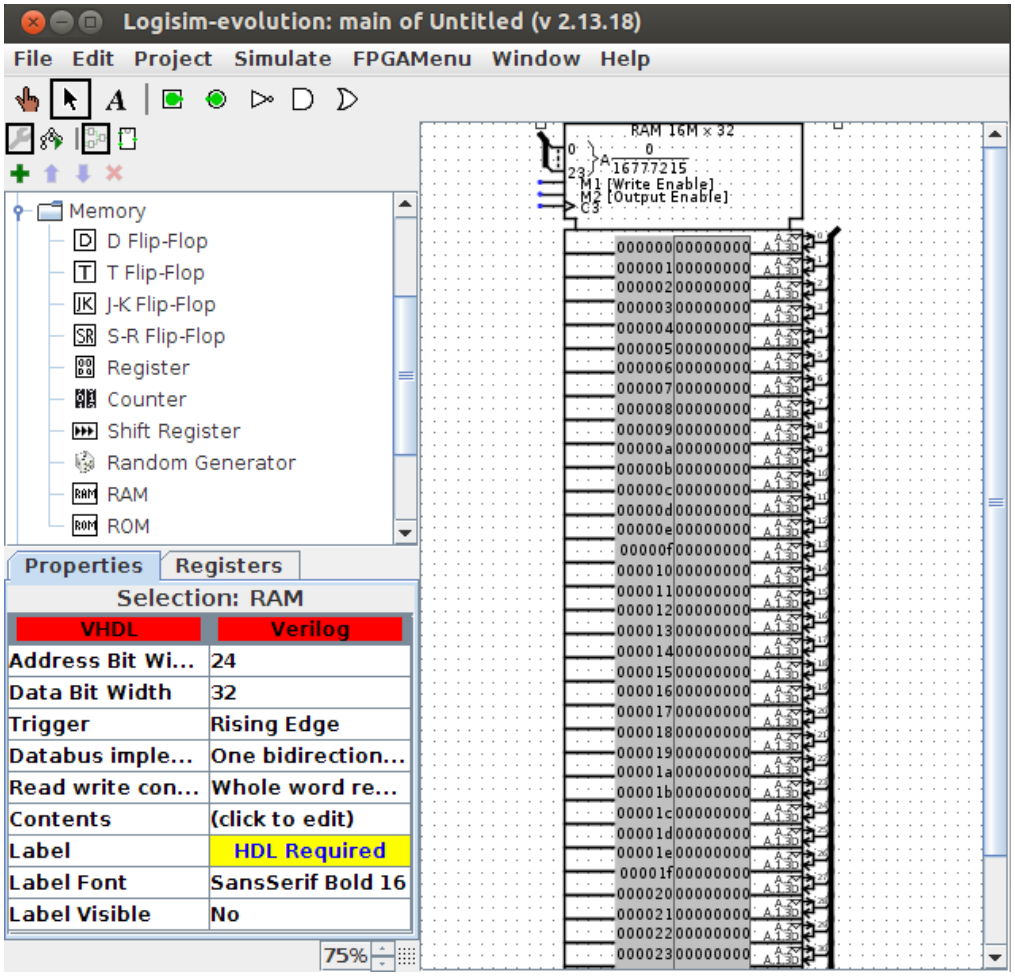
Unidade de Extensão de Sinal

- Para implementar a unidade extensão de sinal, podemos usar *splitters*
- A ferramenta Distribuidor (*Splitter*) da biblioteca Base () lhe permitirá fazer isso



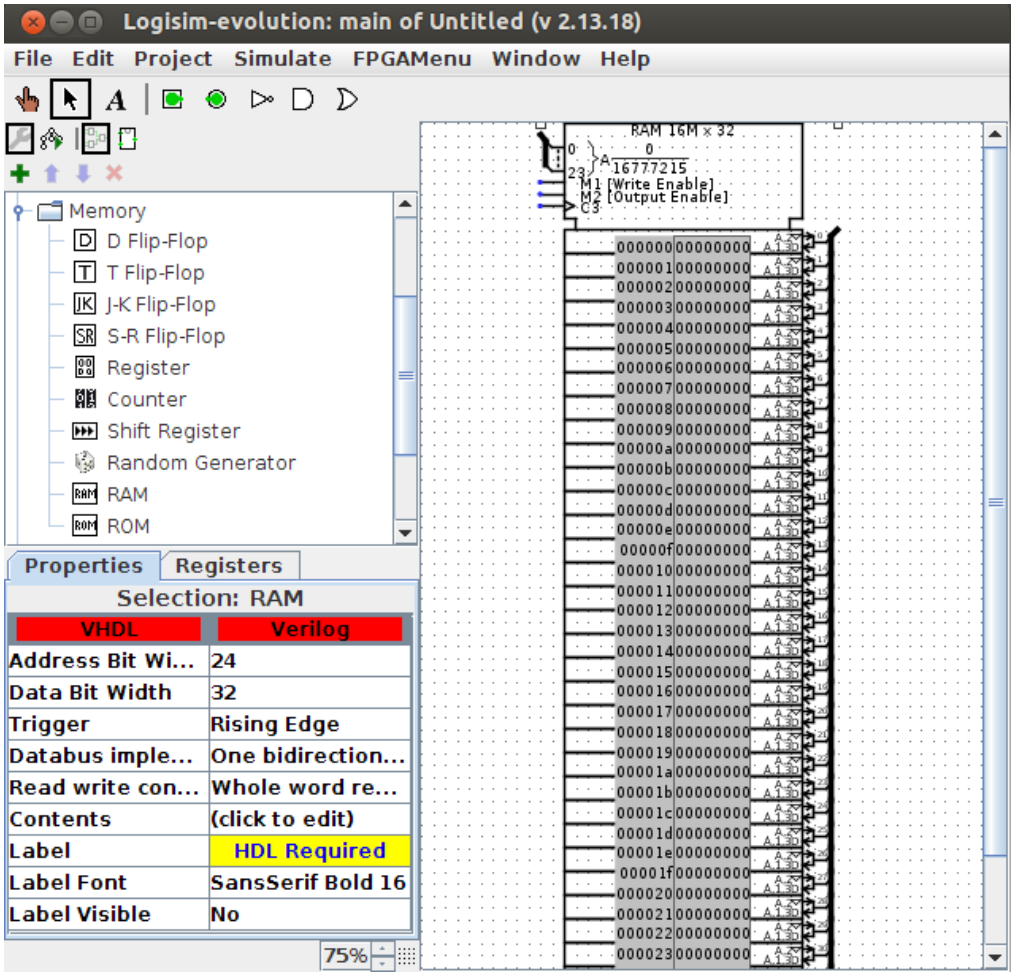
Memória de Dados

- Para implementar a memória de dados, usamos a RAM da biblioteca Memory
- Para ativar a escrita, usamos o sinal Write Enable; e a leitura, Output Enable



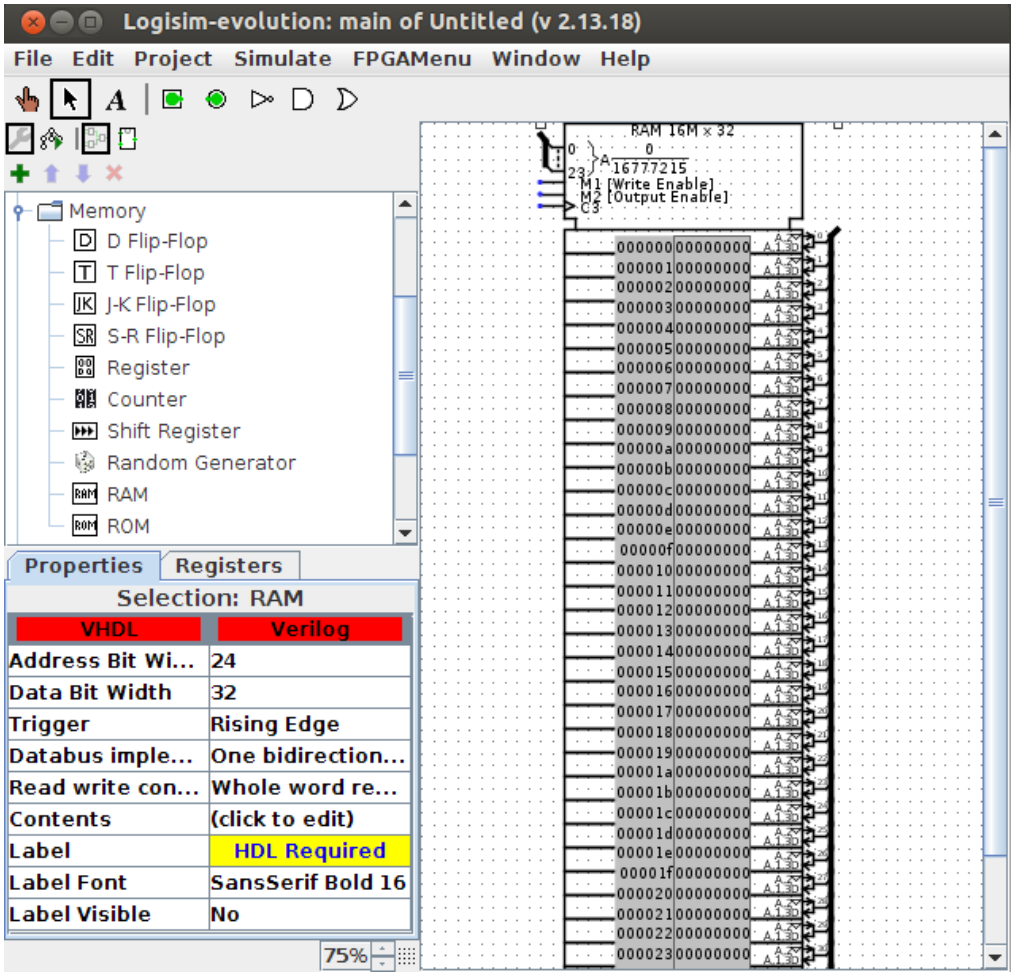
Memória de Dados

- Mas note que o componente RAM default possui apenas um barramento para endereço, e apenas um para leitura e escrita
- Assim, mude suas propriedades para que tenha barramentos de leitura e escrita separados (o barramento de endereços é comum à leitura e à escrita)



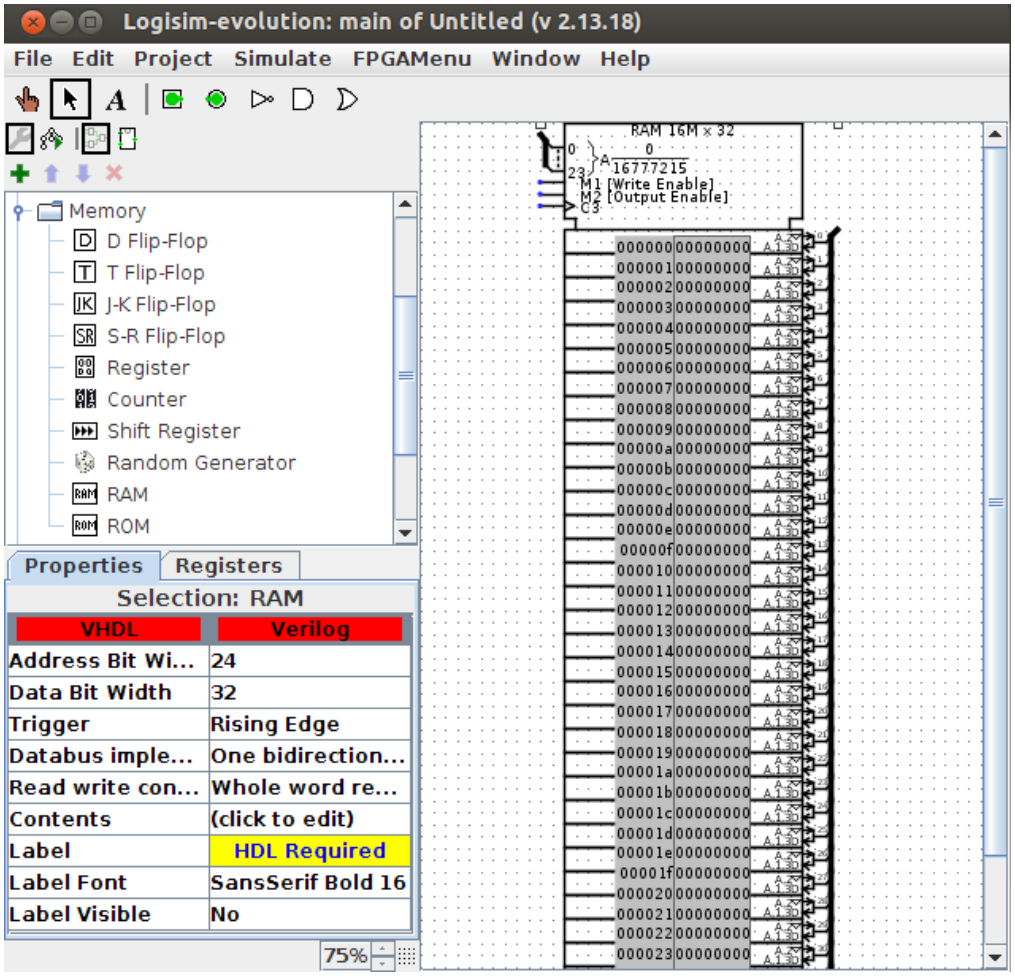
Memória de Dados

- O menor elemento de memória da MIPS ISA é o byte
- Assim, a memória deve permitir o acesso a bytes
- Ela também deve permitir o acesso à half words e words
- Deste modo, ela deve ser composta de quatro unidades de memória de um byte de largura



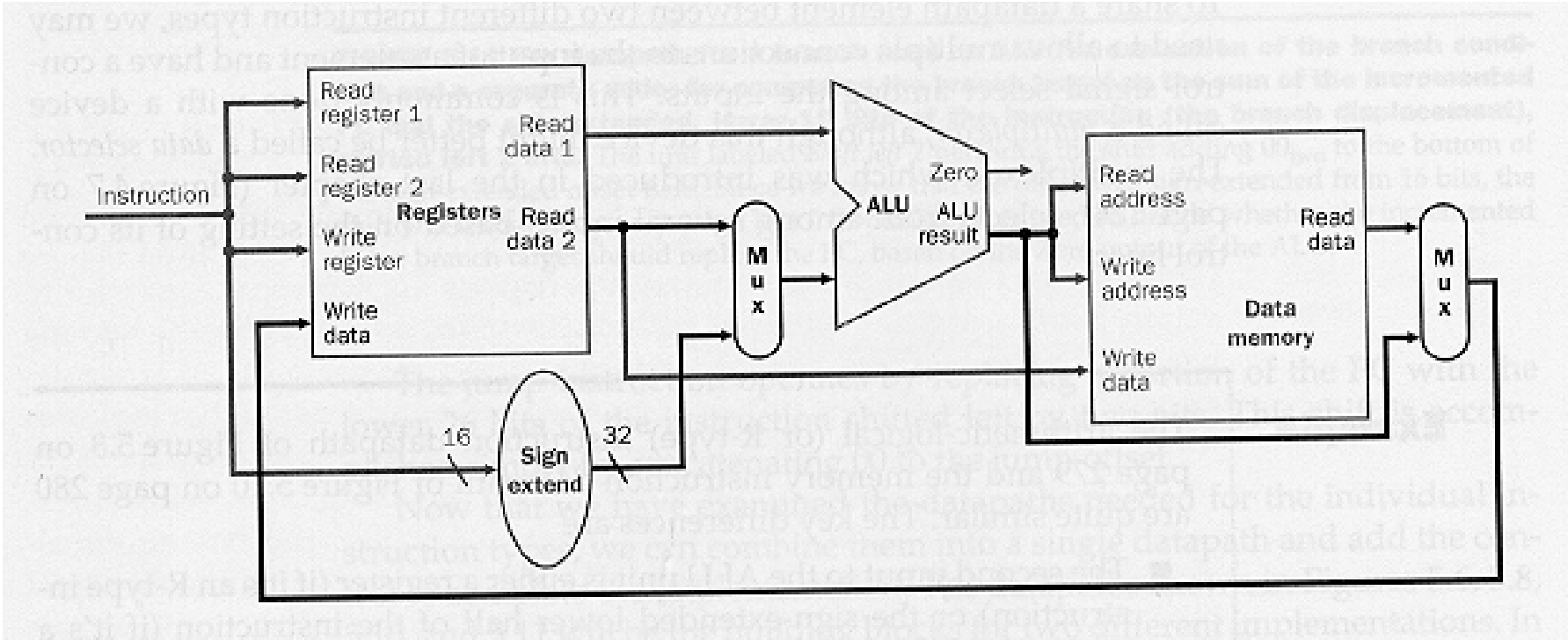
Memória de Dados

- Seu hardware deve permitir acessar para leitura ou escrita:
  - qualquer byte
  - qualquer half word cujo endereço tenha o último bit igual a zero
  - Qualquer word cujo endereço tenha os dois últimos bits iguais a zero
- Use multiplexadores e demultiplexadores para viabilizar estes padrões de acesso



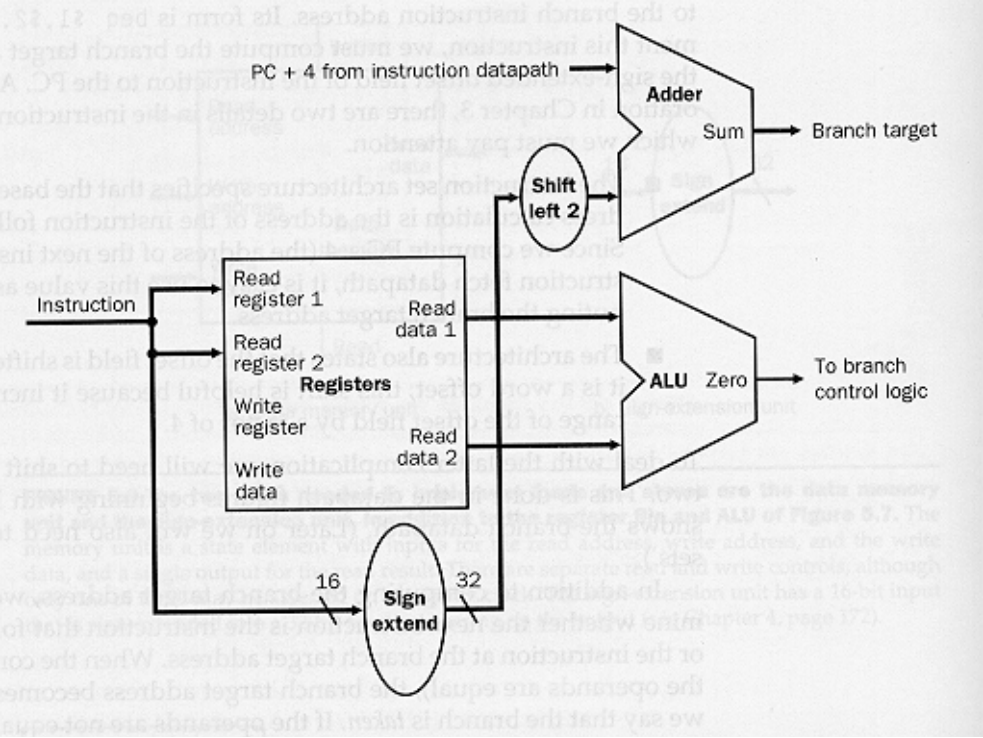
Instruções de Acesso à Memória

- Para juntar o circuito das instruções de acesso à memória ao circuito das instruções aritméticas, podemos empregar multiplexadores



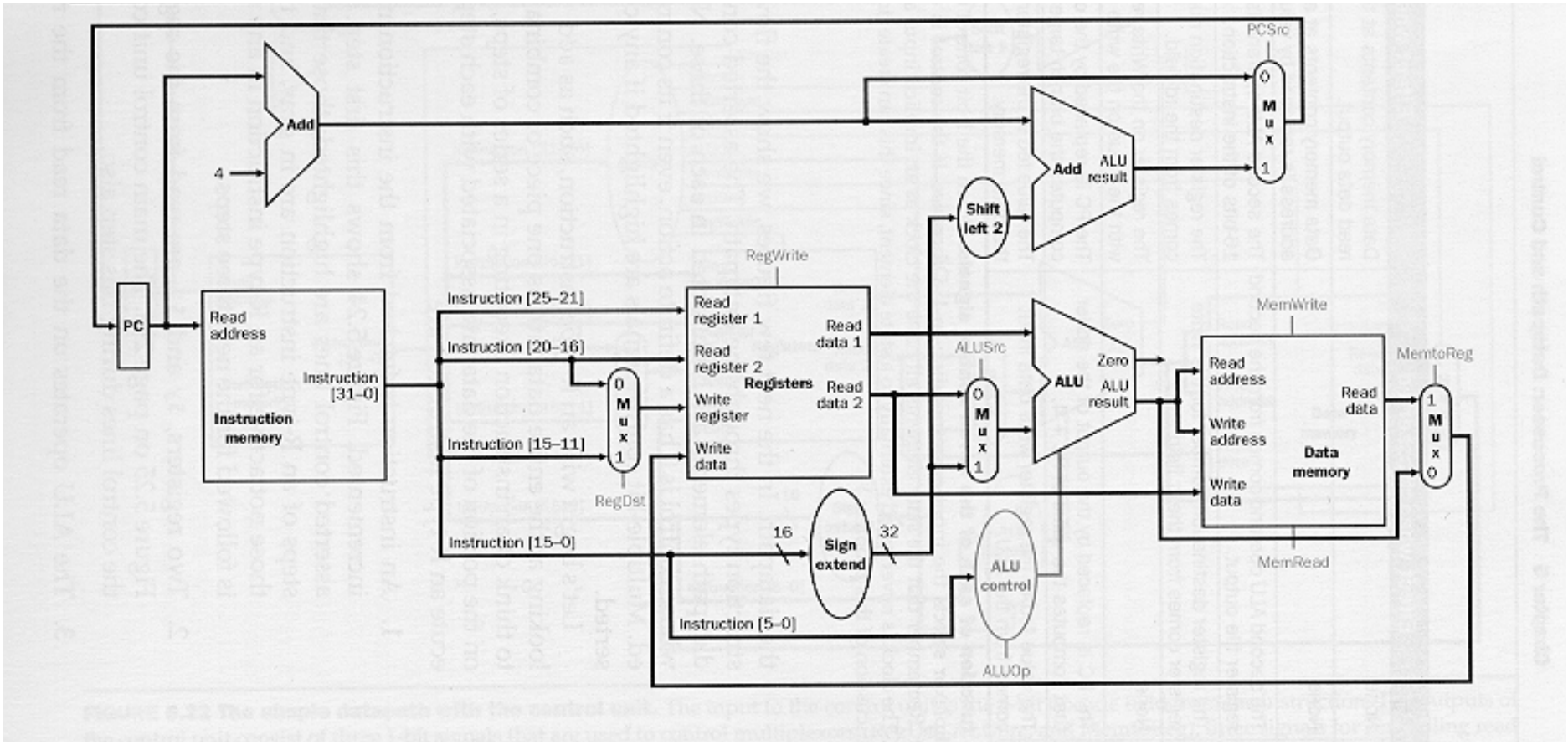
Instruções de Desvio Condicional

- Para implementar as instruções de desvio condicional, podemos usar o circuito ao lado
- A unidade *Shift left 2* pode ser implementada com *splitters* também



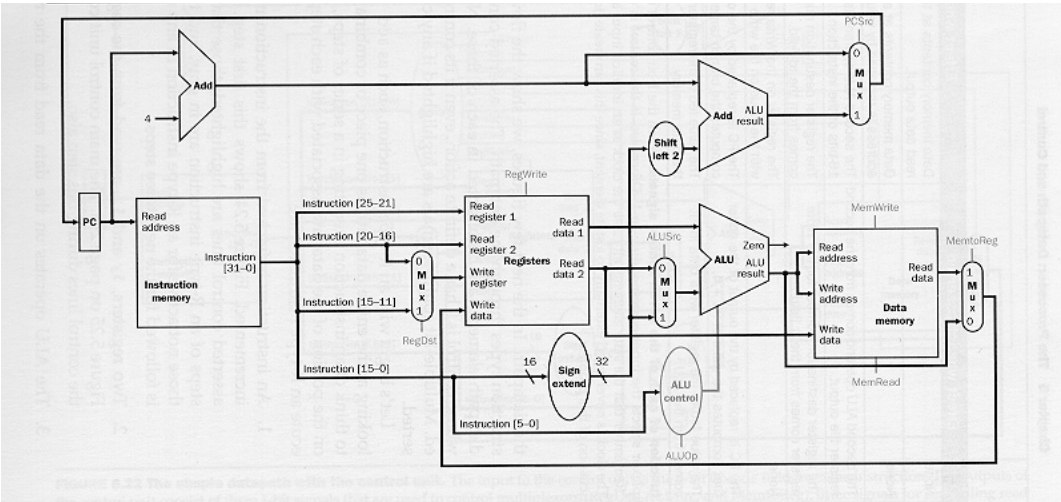
Juntando Todos os Circuitos

- Podemos juntar todos os circuitos que implementamos previamente e vimos hoje como abaixo:



Trabalho 03

- Implemente os circuitos para as instruções de acesso a memória lw, sw, lhu, sh, lbu e sb, e as instruções de desvio condicional beq e bne. Junte estes circuitos aos circuitos implementados previamente como na figura:



- Não é necessário implementar a unidade “ALU control” para o Trabalho 03
- Os trabalhos podem ser feitos em grupos de até 3 alunos e devem ser enviados para sp1@lcad.inf.ufes.br
- **O e-mail deve conter o nome completo dos alunos componentes do grupo**

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow detected
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three operands; overflow detected
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow detected
	add unsigned	addu $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow undetected
	subtract unsigned	subu $s1,$s2,$s3	$s1 = $s2 – $s3	Three operands; overflow undetected
	add immediate unsigned	addiu $s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow undetected
	move from coprocessor register	mfc0 $s1,$epc	$s1 = $epc	Copy Exception PC + special regs
	multiply	mult $s2,$s3	Hi, Lo = $s2 × $s3	64-bit signed product in Hi, Lo
	multiply unsigned	multu $s2,$s3	Hi, Lo = $s2 × $s3	64-bit unsigned product in Hi, Lo
	divide	div $s2,$s3	Lo = $s2 / $s3, Hi = $s2 mod $s3	Lo = quotient, Hi = remainder
	divide unsigned	divu $s2,$s3	Lo = $s2 / $s3, Hi = $s2 mod $s3	Unsigned quotient and remainder
	move from Hi	mfhi $s1	$s1 = Hi	Used to get copy of Hi
	move from Lo	mflo $s1	$s1 = Lo	Used to get copy of Lo
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Word from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Word from register to memory
	load half unsigned	lhu $s1,100($s2)	$s1 = Memory[$s2 + 100]	Halfword memory to register
	store half	sh $s1,100($s2)	Memory[$s2 + 100] = $s1	Halfword register to memory
	load byte unsigned	lbu $s1,100($s2)	$s1 = Memory[$s2 + 100]	Byte from memory to register
	store byte	sb $s1,100($s2)	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immediate	lui $s1,100	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 |$s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND with constant
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Condi-tional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; two's complement
	set less than immediate	slti $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1=0	Compare < constant; two's complement
	set less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1=0	Compare less than; natural numbers
	set less than immediate unsigned	sltiu $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; natural numbers
Uncondi-tional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call