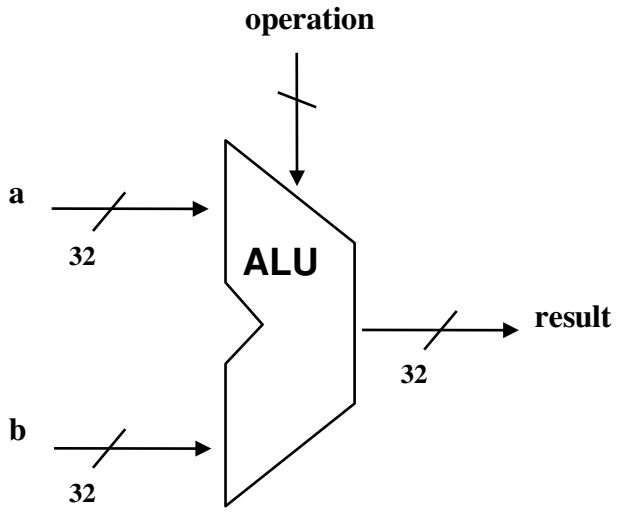
Aritmética Computacional

Capítulo 4

Aritmética

- Já vimos
  - Desempenho (segundos, ciclos, instruções)
  - Abstrações:
    Arquitetura de Conjunto de Instruções
    Linguagem de Montagem e Linguagem de Máquina
- Agora, veremos:
  - Implementação da arquitetura



Números

- **Bits são apenas bits**
  — convenções definem relacionamento entre bits e números
- **Números binários (base 2)**
  0000 0001 0010 0011 0100 0101 0110 0111 1000 1001...
  decimal: $0...2^n-1$
- **Com bases diferentes de 10, tem-se uma complicação adicional:**
  números são finitos -> tratamento de overflow
  frações e números reais -> ponto-flutuante
  números negativos
  e.g., lembrem-sem que não há a instrução subi no MIPS pois, addi pode adicionar um número negativo
- **Como representamos números negativos?**
  i.e., que padrões de bits representarão quais números?

Representações possíveis

Sign Magnitude:	One's Complement	Two's Complement
000 = +0	000 = +0	000 = +0
001 = +1	001 = +1	001 = +1
010 = +2	010 = +2	010 = +2
011 = +3	011 = +3	011 = +3
100 = -0	100 = -3	100 = -4
101 = -1	101 = -2	101 = -3
110 = -2	110 = -1	110 = -2
111 = -3	111 = -0	111 = -1

- Diferenças entre: numero de zeros, facilidade das operações

MIPS

- **Números de 32 bits com sinais:**

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = +\ 1_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = +\ 2_{ten}$
...
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = +\ 2{,}147{,}483{,}646_{ten}$ — *maxint*
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = +\ 2{,}147{,}483{,}647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -\ 2{,}147{,}483{,}648_{ten}$ — *minint*
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -\ 2{,}147{,}483{,}647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -\ 2{,}147{,}483{,}646_{ten}$
...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = -\ 3_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -\ 2_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -\ 1_{ten}$

Operações em complemento de 2

- **Negando um número em complemento de 2: inverter todos os bits e adicionar 1**
  - **Lembre-se que: "negar" e "inverter" são bem diferentes!**
- **Converter números de n bits em números com mais que n bits:**
  - **Imediato de 16 bits do MIPS é convertido para 32 bits por aritmética**
  - **Copiar os bits mais significativos (o bit de sinal) para a parte mais significativo do destino**

```
0010  -> 0000 0010
1010  -> 1111 1010
```

  - **"extensão do sinal" (lbu vs. lb)**

Novas instruções

- **instruções “unsigned”: (exemplo de aplicação, cálculo de memória)**
- **sltu $t1, $t2, $t3 # diferença é “sem sinal”**
- **slti e sltiu # envolve imediato, com ou sem sinal**

- **Exemplo: supor $s0 = FF FF FF FF e $s1 = 00 00 00 01**

slt $t0, $s0, $s1
como $s0 < 0 e $s1 > 0 ⇒ $s0<$s1 ⇒ $t0 = 1

sltu $t0, $s0, $s1
como $s0 e $s1 não tem sinal ⇒ $s0>$s1 ⇒ $t0 = 0

Cuidados com extensão 16 bits

- beq $s0, $s1, label # salta para PC + label se teste OK
- label tem 16 bits e PC possui 32 bits
  - estender de 16 para 32 bits antes da operação aritmética (PC+label)
- se label > 0
  - preencher com zeros à esquerda
- se label < 0 CUIDADO
  - preencher com 1´s à esquerda
  - verificar
- por este motivo operação é chamada de
  - EXTENSÃO DE SINAL

Adição e Subtração

- **Da mesma forma que na escola (vai/empresta 1s)**

```
   0111          0111          0110
 + 0110        - 0110        - 0101
```

- **Operações em complemento de 2 é fácil**
  - **Subtração é feita usando adição de números negativos**

```
   0111
 + 1010
```

- **Overflow (resultado é maior do que a capacidade do meio que armazena):**
  - **e.g., adicionando dois números de n-bits não produz um número de n bits**

```
   0111
 + 0001
   1000
```

Detectando Overflow

- **Não há overflow quando adiciona-se um número positivo e um número negativo**
- **Não há overflow quando os sinais são os mesmos em uma subtração**
- **CONDIÇÕES DE OVERFLOW**

op	A	B	resultado
A+B	+	+	-
A+B	-	-	+
A-B	+	-	-
A-B	-	+	+

Em hardware, comparar o "vai-um" e o "vem-um" com relação ao bit de sinal

Efeitos do overflow

- **Uma exceção (interrupção no programa) ocorre**
  - **O controle salta para endereço predefinido para tratar a exceção**
  - **Registrador EPC — *EXCEPTION PROGRAM COUNTER***
  - **Endereço interrompido é salvo para possível retorno**
    - mfc0 (move from system control): copia endereço do EPC para qualquer registrador
- **Nem sempre é desejável tratar de *overflows***
  — Instruções do MIPS: `addu, addiu, subu`

  note: `addiu` ***ainda estende o sinal!***
  note: `sltu, sltiu` ***para comparações sem sinal!***

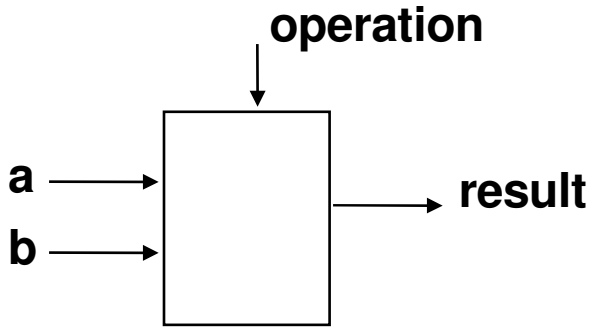
Instruções

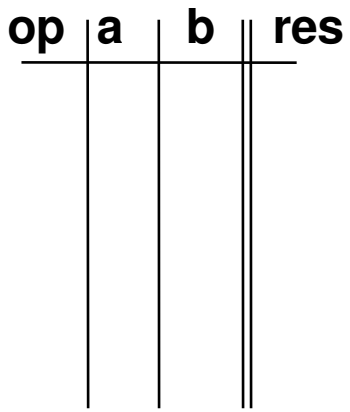
add	add	R
add immediate	addi	I
add unsigned	addu	R
add immediate unsigned	addiu	I
subtract	sub	R
subtract unsigned	subu	R
and	and	R
and immediate	andi	I
or	or	R
or immediate	ori	I
shift left logical	sll	R
shift right logical	srl	R
load upper immediate	lui	I
load word	lw	I
store word	sw	I
load byte unsigned	lbu	I
store byte	sb	I
branch on equal	beq	I
branch on not equal	bne	I
jump	j	J
jump and link	jal	J
jump register	jr	R
set less than	slt	R
set less than immediate	slti	I
set less than unsigned	sltu	R
set less than immediate unsigned	sltiu	I

multiply	mult	R
multiply unsigned	multu	R
divide	div	R
divide unsigned	divu	R
move from Hi	mfhi	R
move from Lo	mflo	R
move from system control (EPC)	mfc0	R
fp add single	add.s	R
fp add doublr	add.d	R
fp subtract single	sub.s	R
fp subtract double	sub.d	R
fp multiply single	mul.s	R
fp multiply double	mul.d	R
fp divide single	div.s	R
fp divide double	div.d	R
load word to fp single	lwc1	I
store word to fp single	swc1	I
branch on fp true	bclt	I
branch on fp false	bclf	I
fp compare single (x= eq,neq,lt,le,gt,ge)	c.x.s	R
fp compare double (x= eq,neq,lt,le,gt,ge)	c.x.d	R

Uma ULA (unidade de lógica e aritmética)

- **Vamos construir uma ULA para suportar as instruções andi e ori**
  - **Construiremos uma ALU de 1 bit, e usaremos 32 delas**

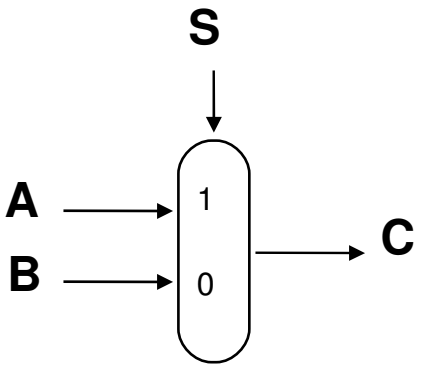




- **Implementação possível (soma-de-produtos):**

Multiplexador

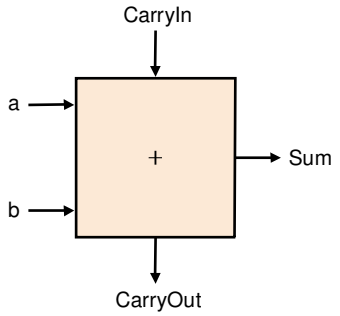
- **Selecionar uma das entradas para ser a saída, baseando-se no sinal de controle**



- **Vamos construir a ALU usando o MUX!**

Diferentes implementações

- **Projeto de Hw não é fácil decidir a melhor forma para construir algo**
  - **Não deseja-se muitas entradas para uma única porta**
  - **Não deseja-se "passar" por muitas portas**
  - **Compreender bem o projeto é extremamente importante**
- **Olhando a ULA de 1 bit para soma**

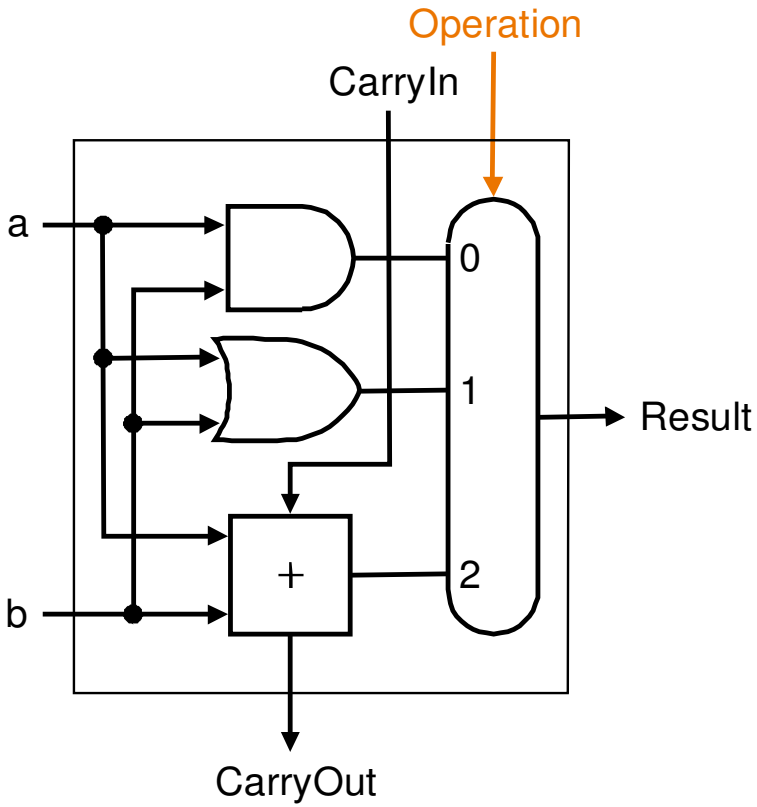


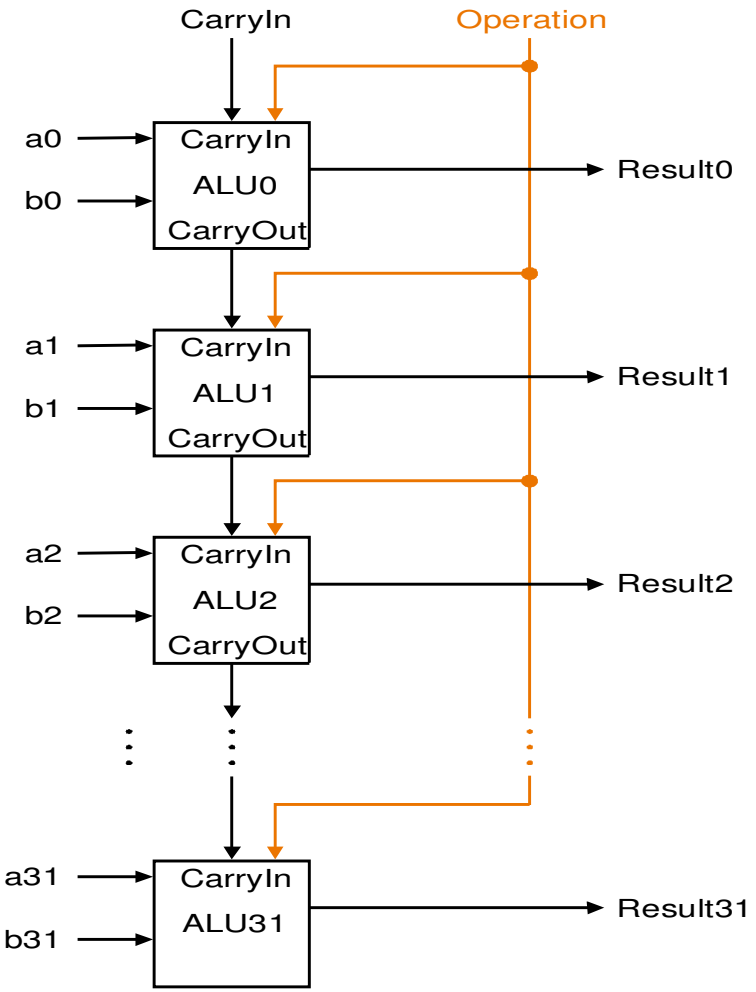
$c_{out} = a\ b + a\ c_{in} + b\ c_{in}$

$sum = a \text{ xor } b \text{ xor } c_{in}$

- **Como construir uma ULA de 1-bit para add, and, e or?**
- **Como construir uma ULA de 32 bits?**

Construindo uma ULA de 32 bits





E quanto a subtração (a – b) ?

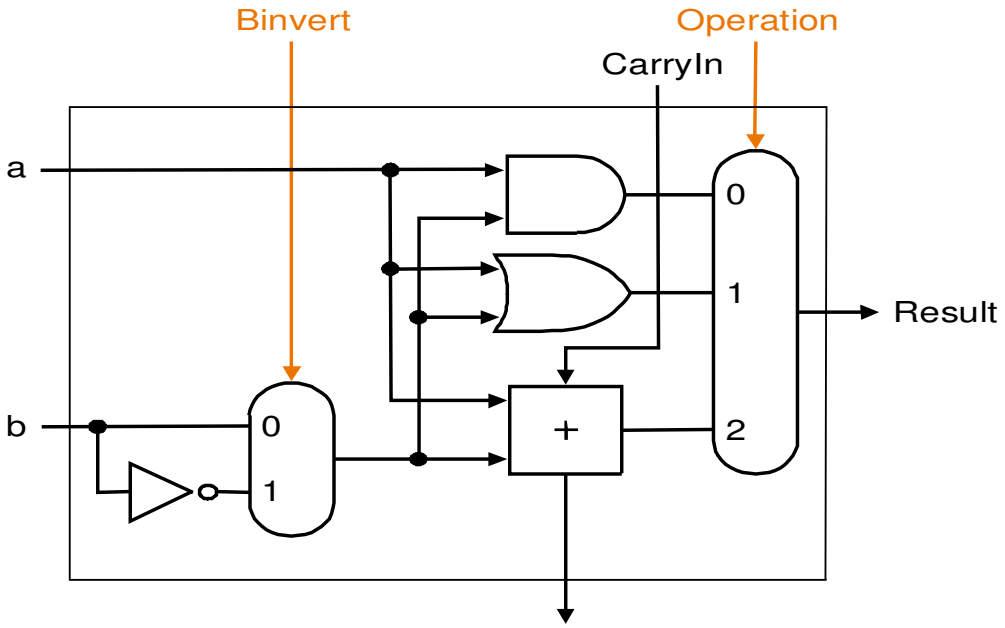
- **Abordagem de complemento de 2: apenas negar b e adicionar**

  $$a - b = a + (- b)$$

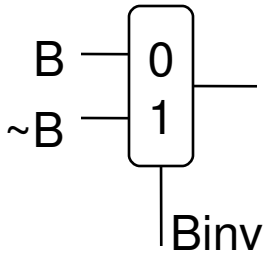
- **Como negar?**

  $$(- a) = comp_2(a) = comp_1(a) + 1$$

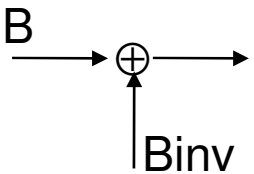
- **Uma solução direta:**

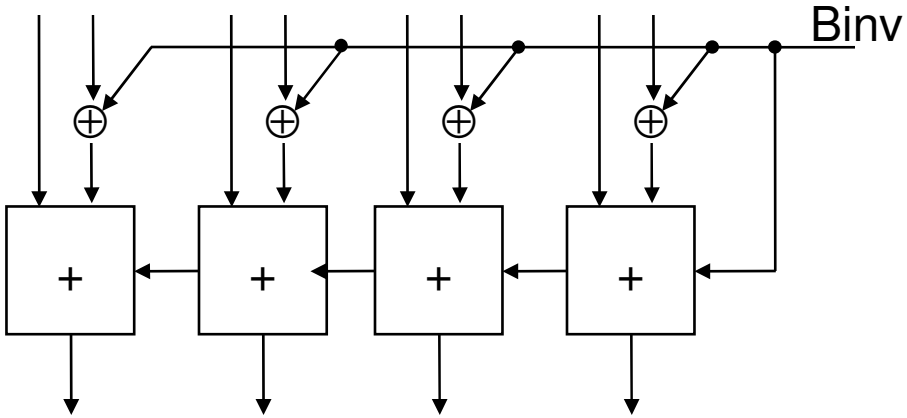


Subtratores



equivalente à

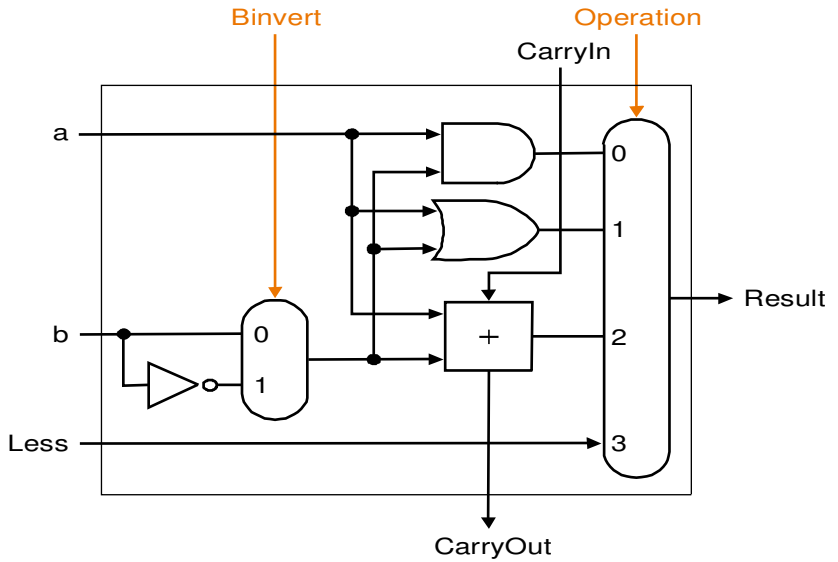




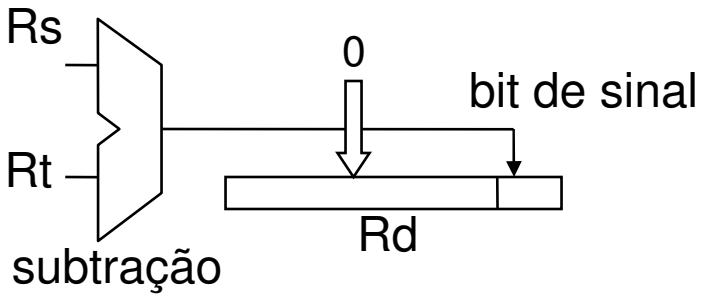
Construindo a ULA para o MIPS

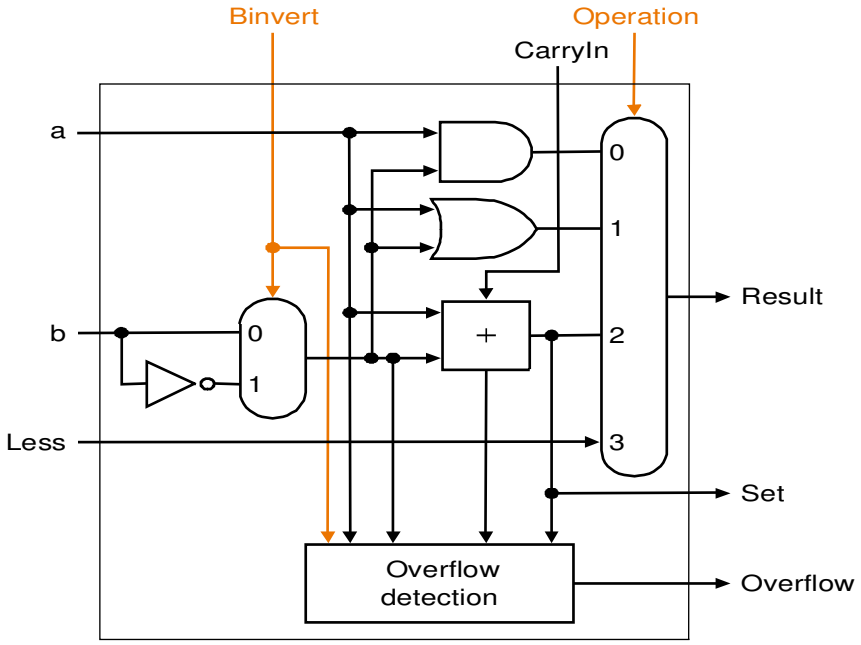
- Necessita suportar a instrução slt
  - Lembre-se que: slt é uma instrução aritmética
  - Produz 1 se rs < rt e 0 em caso contrário
  - Usar subtração: (a-b) < 0 implica que a < b
- Necessita suportar teste de igualdade (beq $t5, $t6, $t7)
  - Usar subtração: (a-b) = 0 implica que a = b

Oferecendo suporte a slt

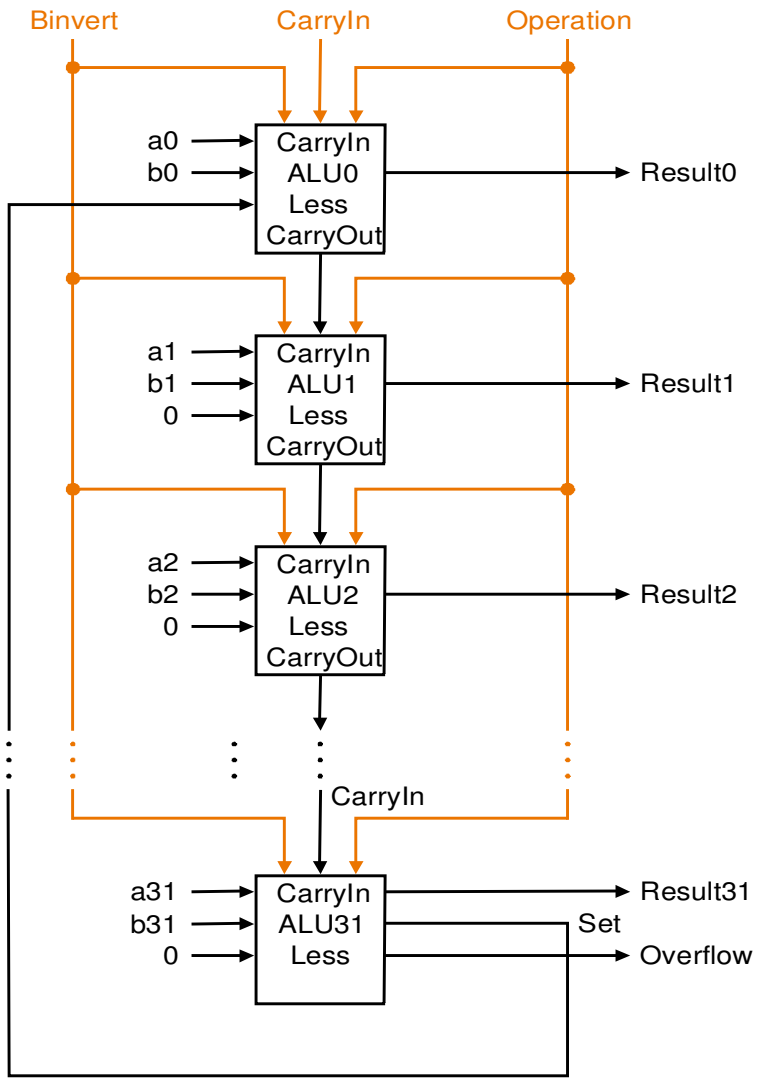


a.





b.

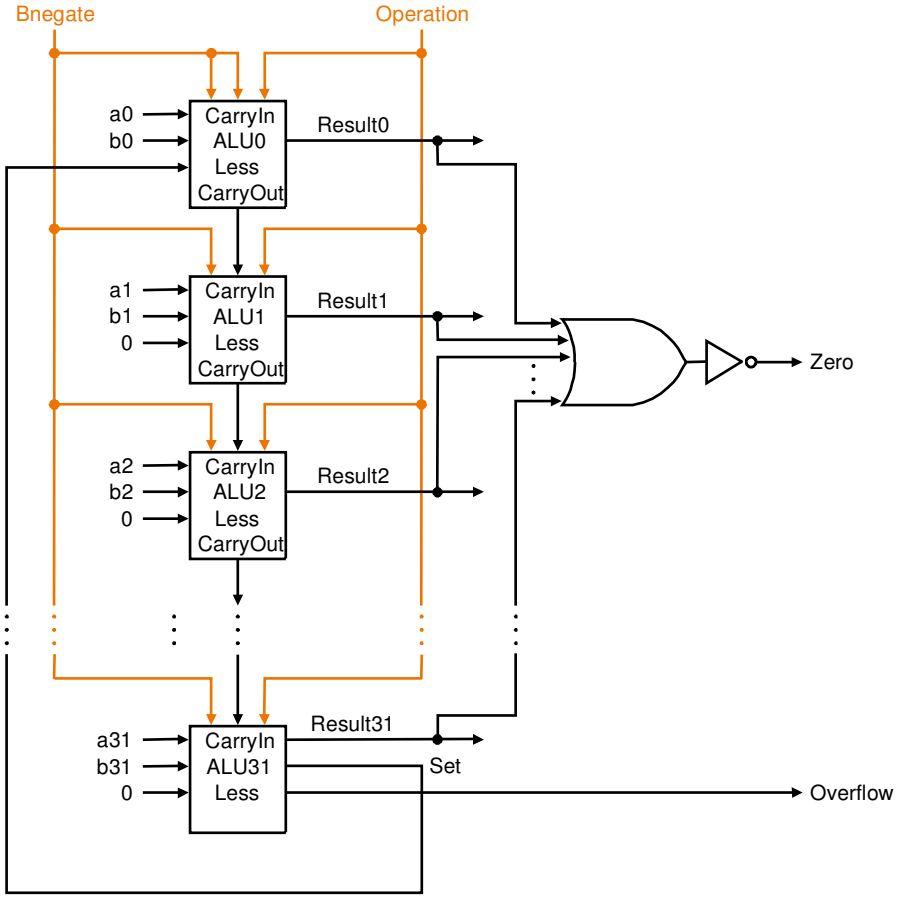


Teste para igualdade

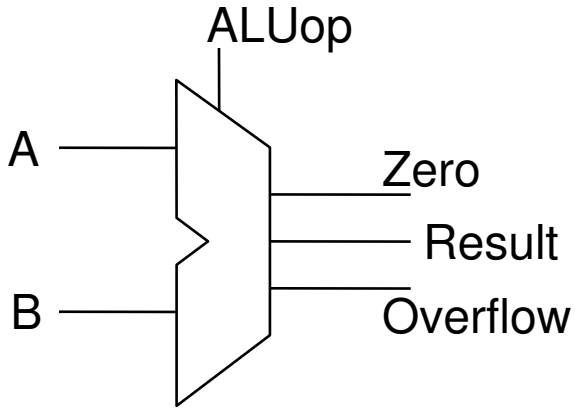
- **O controle é dado por:**

```
000 = and
001 = or
010 = add
110 = subtract
111 = slt
```

•*Note: zero é um 1 quando o resultado é zero!*



ULA



32 bits: A, B, resultado

1 bit: Zero, Overflow

3 bits: ALUop

ALUop Binv-OP	Instrução
0 00	and
0 01	or
0 10	add
1 10	sub
1 11	slt
1 10	beq

Conclusões

- **Pode-se construir uma ULA para suportar as instruções MIPS**
  - **Usar multiplexador para selecionar entrada desejada**
  - **Pode-se, de maneira eficiente, realizar subtração em complemento de 2**
  - **Pode-se replicar uma ULA de 1-bit para produzir uma ULA de 32-bits**
- **Pontos importantes sobre o hardware**
  - **Todas as abertas estão sempre "trabalhando"**
  - **A velocidade da porta é afetada pelo número de entradas da porta!**
  - **A velocidade do circuito é afetado diretamente pelo número de portas em série (portas no caminho crítico)**
- **Mudanças na organização das portas lógicas e dos circuitos podem melhorar o desempenho (similar ao adotar algoritmos melhores em software)**