Fundamentos de Arquitetura de Computadores

Tiago Alves

Faculdade UnB Gama
Universidade de Brasília

Adição e subtração

Exatamente como base decimal (emprestar/vai 1s) descartando o carry out:

0011	0111	1100	0111	0110	0010
+ 0010	+1010	+1111	- 0110	- 0101	-0100
0101	10001	11011	0001	0001	1110

Facilidade de operações do complemento de dois: subtração usando adição para números negativos.

Overflow (resultado muito grande para a word finita do computador): somar dois números de n bits pode produzir um número de n+1 bits.

0101 = +5	1001 = -7	0110 = 6
+ 0100 = +4	+ 1010 = -6	- 1000 = -8
1001 ≠ 9	10011 ≠ -13	1110 ≠ 14

Note que o termo overflow não significa que um carry simplesmente transbordou (m de bits do resultado $>$ n bits das parcelas): indica que o resultado não "cabe" na faixa dinâmica de n bits!!!

Detectando Overflows

Não há overflow ao se:

- somar um número positivo com um negativo e
- subtrair operandos com sinais iguais.

O overflow ocorre quando o valor afeta o sinal: (obs. $A > 0$ e $B > 0$)

- somar dois positivos produz um negativo: $A + B < 0$
- somar dois negativos produz um positivo: $-A + (-B) > 0$
- subtrair um negativo de um positivo e obtenha um negativo: $A - (-B) < 0$
- subtrair um positivo de um negativo e obtenha um positivo: $-A - B > 0$

OU Se, na soma:

- os MSB dos operandos forem diferentes: Não há overflow.
- os MSB dos operandos forem iguais e o MSB do resultado for diferente dos operandos: Há overflow.

OU Se, na soma, o carry in do MSB for diferente do carry out: Há Overflow.

Efeitos do Overflow

Uma **exceção** (**interrupção interna**) ocorre:

- O controle salta para um endereço predefinido para exceção
- O endereço interrompido é salvo para uma possível retomada

Detalhes baseados na linguagem/sistema de software

- Nem sempre desejamos detectar overflow. Novas instruções MIPS: `addu`, `addiu`, `subu`
- Nota: `addiu` ainda com extensão de sinal
- Nota: `sltu`, `sltiu` para comparações sem sinal,

Principais Arquiteturas Aritméticas

Pilha:

- As operações são sempre realizadas com os argumentos na pilha, e o resultado é também armazenado na pilha. (HP, Forth)

Acumulador:

- As operações são feitas sobre registradores (incluindo A) e o resultado armazenado em um registrador especial chamado Acumulador (A).(Z80, 8051, alguns DSPs de Ponto Fixo)

Registrador-Registrador:

- As operações são feitas sobre registradores e o resultado é armazenado em qualquer registrador. (MIPS)

Registrador-Memória:

- As operações aritméticas buscam um dos argumentos na memória e armazenam o resultado em um registrador. (x86)

ULA: Unidade Lógica e Aritmética

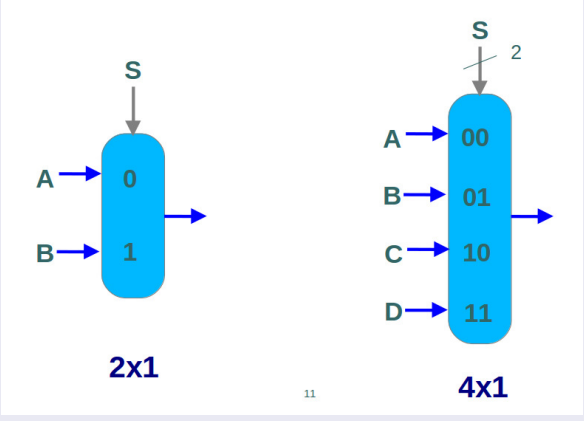
O MIPS tem uma ULA de 32 btis

Para ilustrar sua construção, segue a descrição de uma simples ULA com as seguintes operações:

- Soma e subtração em complemento de 2
- Operações lógicas `and`, `or` e `nor`
- Instrução `slt`
- Detecção de overflow
- Detecção de igualdade (comparação).

Elementos Básicos

Multiplexador:



Elementos Básicos

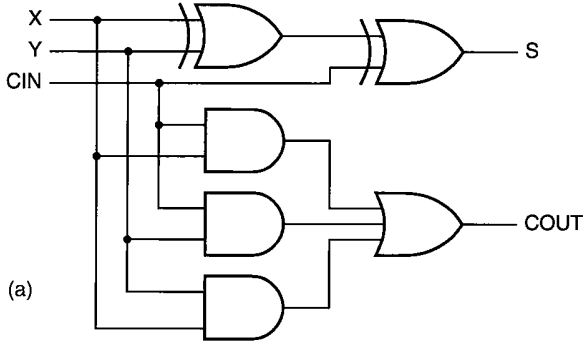
Somador Total

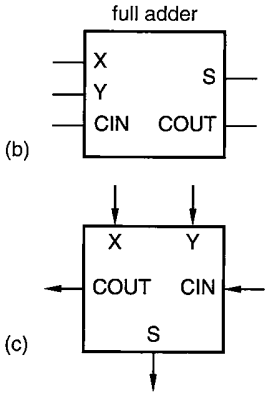
O circuito para somar três bits (ou somar duas palavras de 1 bit mais um *carry*) é chamado de **somador total** (*full adder*). Este circuito soma três bits, A, B e C_{in}, e gera uma soma de dois bits (de 0 a 3).

$\mathbf{C_{in}}$	$\mathbf{A}$	$\mathbf{B}$	$\mathbf{C_{out}}$	$\mathbf{S}$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

O que gera as equações da soma total:

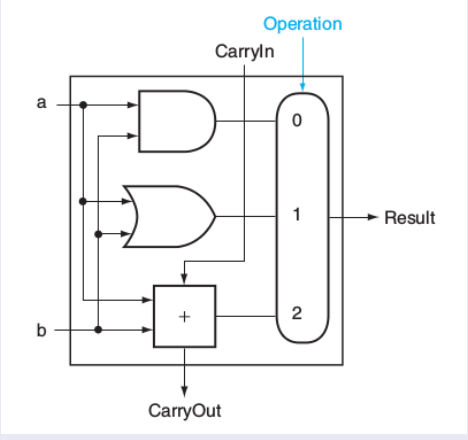
$$\mathbf{C_{out}} = \mathbf{A} \cdot \mathbf{B} + \mathbf{A} \cdot \mathbf{C_{in}} + \mathbf{B} \cdot \mathbf{C_{in}}$$
$$\mathbf{S} = \mathbf{A} \oplus \mathbf{B} \oplus \mathbf{C_{in}}$$


X
Y
CIN
S
COUT
(a)

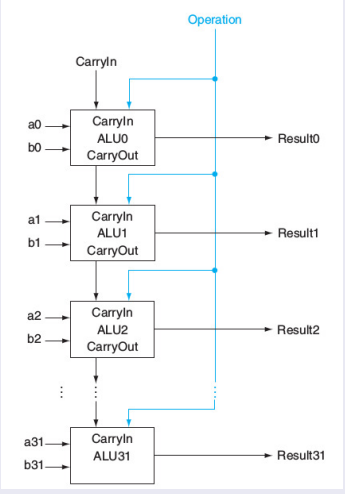

full adder
X
Y
S
CIN
COUT
(b)
X
Y
COUT
CIN
S
(c)

ULA de 1 bit

Operações de soma, and, or:

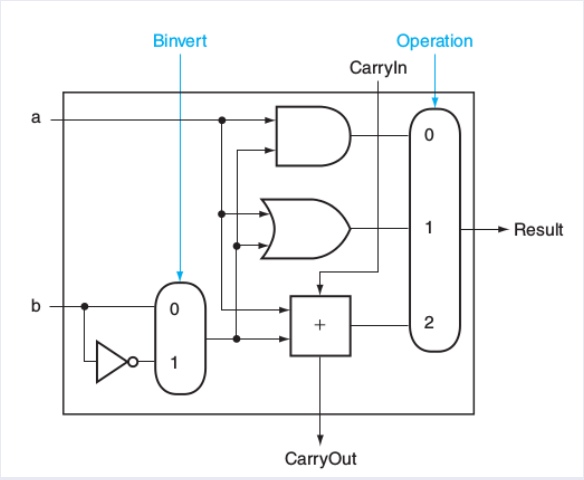


ULA de 32 bits



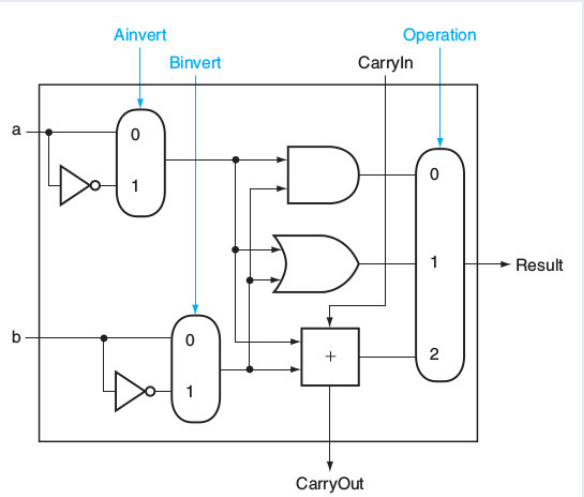
ULA de 1 bit: incluindo subtração

$$a - b = a + (\overline{b} + 1)$$



ULA de 1 bit: incluindo subtração e nor

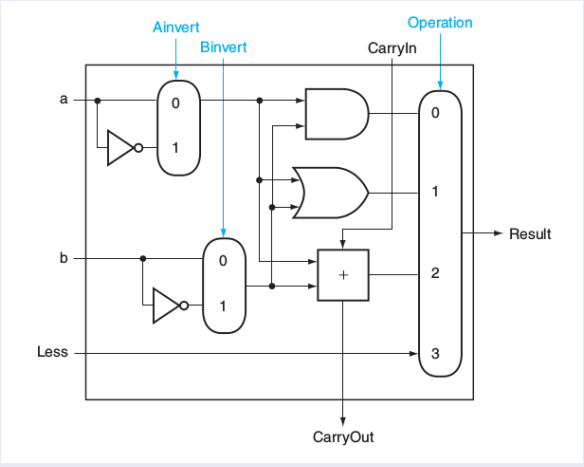
$$\overline{(a+b)} = \overline{a}.\overline{b}$$



Instrução slt

- A instrução `slt` gera saída 1, se rs $<$ rt, e 0, caso contrário. Assim, todos os bits, com exceção do bit menos significativo, são fixados em zero.
- O bit menos significativo depende do resultado da comparação.
- Inclui-se a entrada Less.

ULA de 1 bit: incluindo subtração, nor e slt

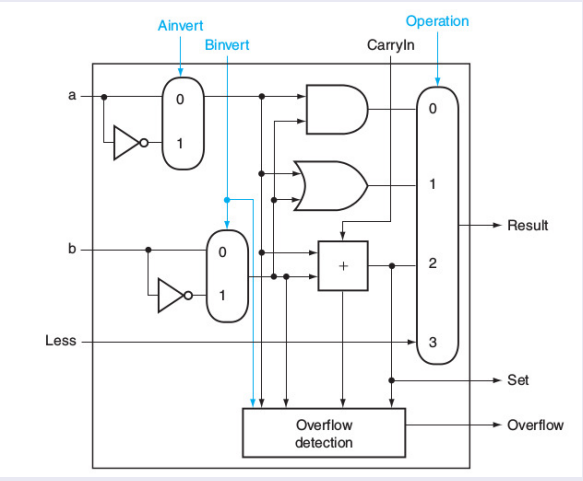


Calculando slt

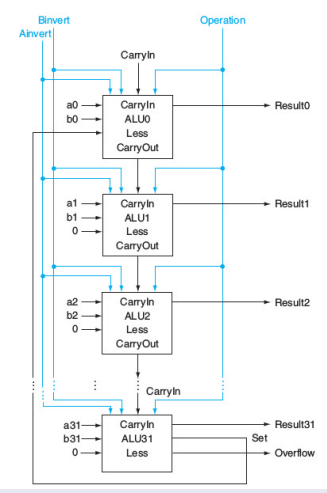
- O bit menos significativo da ULA deve ser 1 se rs $<$ rt
- Calcula-se este valor subtraindo rs de rt e tomando-se o bit mais significativo (Sinal):

  se $rs - rt < 0$, $rs < rt$.
- Utiliza-se o próprio subtrator da ULA para obter este valor, modificando-se o último estágio

ULA de 1 bit (último estágio): detector de overflow



ULA de 32 bits revisada



Teste de igualdade

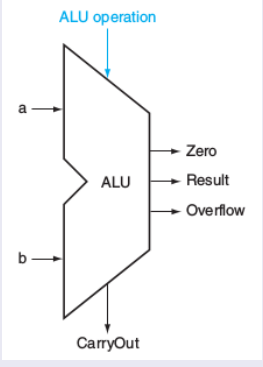
As instruções bne e beq testam se dois valores são iguais ou não

Para testar igualdade subtrai-se os dois operandos e testa-se se o resultado é zero:

- $A == B$ se $A - B = 0$
- Teste: operação `NOR` entre os bits do resultado.
- Igual = `not(R0 or R1 or ... or R31)`

ULA de 32 bits com comparação

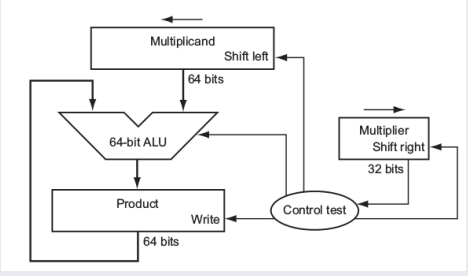
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

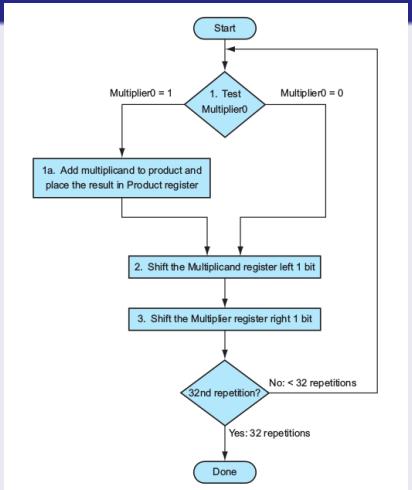


Multiplicação

- Mais complexa do que a adição: realizada através de deslocamento e adição.
- Requer mais tempo (de computação) e mais área de chip.
- Veremos apenas três versões.
- Números negativos: converta para sem sinal, multiplique, defina sinal do resultado.

Multiplicação: primeira versão

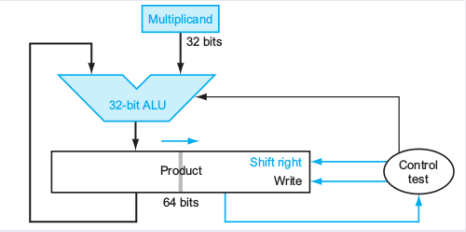


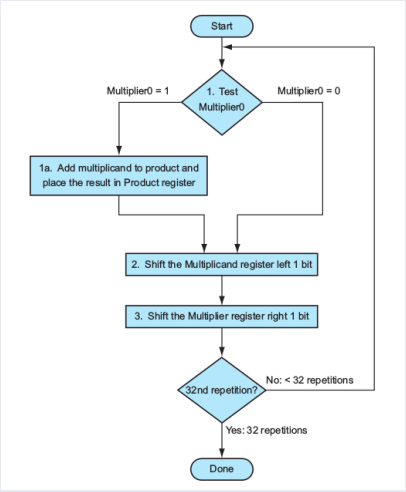


Double and add: O multiplicando de 32 bits é disposto no lado direito do registrador Multiplicand e deslocado para esquerda de 1 bit a cada passo da multiplicação. O multiplicador é deslocado na direção oposta a cada passo. O algoritmo inicia com o produto inicializado em 0. Control decide quando deslocar Multiplicand e Multiplier e quando escrever o valor no registrador Product.

Multiplicação: segunda versão

O multiplicador inicia na metade direita do produto.

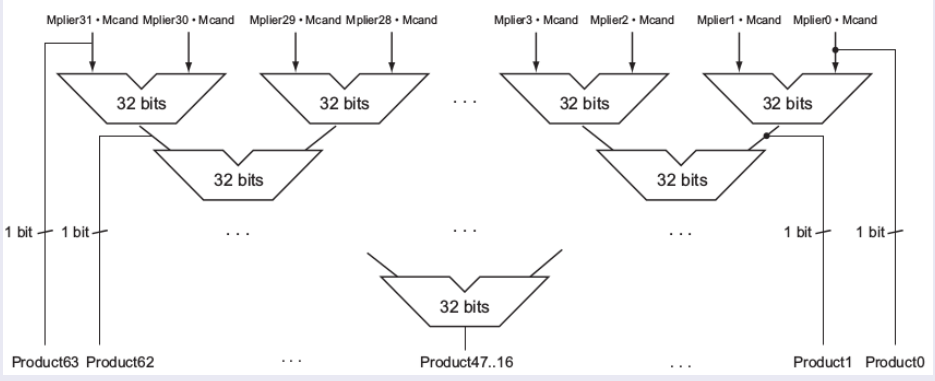




Quem movimenta para direita é o conteúdo de Product. O registrador Multiplier não é mais necessário, pois o valor é posto na porção da direita do registrador Product (note diferenças em destaque). Na prática, o registrador Product deve possuir 65 bits para armazenar o carry do somador.

Multiplicação: terceira versão

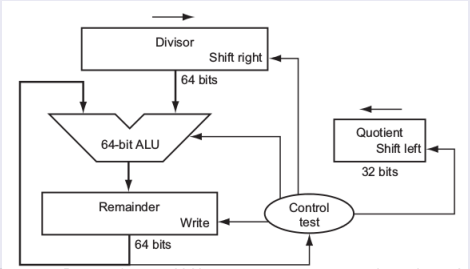
Loop unrolling. Maior área em chip

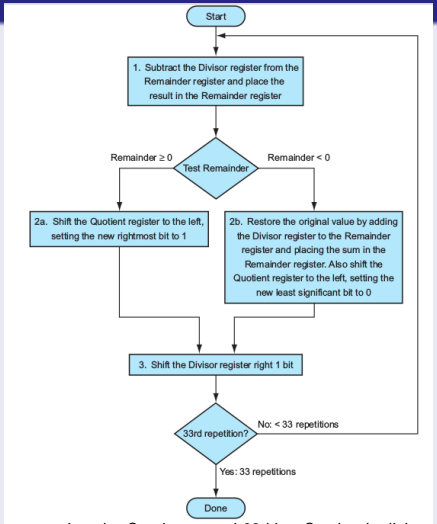


Divisão

- Operação recíproca da multiplicação
- Ainda menos frequente e mais peculiar
- dividendo $=$ divisor $\times$ quociente $+$ resto

Divisão Simples





Os registradores Divisor e Remainder e a ALU possuem comprimento de 64 bits. Apenas o registrador Quotient possui 32 bits. O valor do divisor de 32 bits é colocado na porção esquerda do registrador Divisor e deslocado para a direita 1 bit a cada iteração. O valor do resto é inicializado com o valor do dividendo. Control decide quando deslocar os registradores Divisor e Quotient e quando escrever o novo valor no registrador Remainder.

Multiplicação e Divisão com sinal

- Na multiplicação: verificar se os sinais do multiplicando e do multiplicador são iguais ou diferentes, definindo o sinal do produto.
- Na divisão: precisamos definir o sinal do quociente e do resto.
  - dividendo = divisor × quociente + resto
  - $\frac{7}{2} \rightarrow 7 = 3 \times 2 + 1$
  - $\frac{-7}{2} \rightarrow -7 = -3 \times 2 + (-1)$ ou $-7 = -4 \times 2 + 1$?

Regra:

- Quociente: Mesma regra da multiplicação
- Resto: Mesmo sinal do Dividendo.

Aritmética inteira no MIPS

Adição:

- `add, addi, addu, addiu`
- `addu $t0,$t1,$t2  # $t0=$t1+$t2  sem sinal/overflow`

Subtração:

- `sub, subu`
- `sub $t0,$t1,$t2  # $t0=$t1-$t2  com sinal/overflow`

Multiplicação:

- `mult, multu`
- `mult $t0,$t1  # {Hi,Lo}=$t0x$t1  Registradores Hi e Lo`

Divisão:

- `div, divu`
- `div $t0,$t1  # Lo=floor($t0/$t1)  Hi=$t0 % $t1 (Resto)`

Movimentação: Move From/To High/Low

- `mfhi, mflo, mthi, mtlo`
- `mfhi $t0  # $t0=Hi`
- `mtlo $t1  # Lo=$t1`