ARITMÉTICA DIGITAL
CAPITULO 3

Anotações do material suplementar (apresentações PPT) ao Livro do Hennessy e Patterson e do material do Prof. Celso Alberto Saibel Santos (DI/CT).

Aritmética

- O que foi visto:
  - Visão geral de Sistemas Computacionais
  - Abstrações:
    Arquitetura do Conjunto de Instruções (ISA)
    Ling. Assembly e Ling. de Máquina
  - Conjunto de Instruções MIPS (tipo R, I, J)
  - Código compilado

Números

- Bits são apenas bits (sem significado inerente)
  - Convenções definem relações entre bits e valores
- Números Binários (base 2)
  0000 0001 0010 0011 0100 0101 0110 0111 1000 1001...
  decimal: 0...2^N-1
- É claro, existem outras complicações:
  - números são finitos (*overflow*)
  - frações e números reais
  - números negativos
- Como representar números negativos?
  i.e., qual o padrão de bits representam tais números?

Possíveis Representações

- Sinal de Magnitude

000 = +0
001 = +1
010 = +2
011 = +3
100 = -0
101 = -1
110 = -2
111 = -3

Complemento de 1

000 = +0
001 = +1
010 = +2
011 = +3
100 = -3
101 = -2
110 = -1
111 = -0

Complemento de 2

000 = +0
001 = +1
010 = +2
011 = +3
100 = -4
101 = -3
110 = -2
111 = -1

- Questões:
  - Balanceamento, representações do zero, facilidade de operação, facilidade de obtenção dos valores
- Qual delas é a melhor? Por quê?

MIPS

- Números de 32 bit com sinal (em complemento de DOIS):

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{DOIS} = 0_{DEZ}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{DOIS} = +\ 1_{DEZ}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{DOIS} = +\ 2_{DEZ}$
...
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{DOIS} = +\ 2147483646_{DEZ}$ ← ***maxint***
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{DOIS} = \mathbf{+\ 2147483647_{DEZ}}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{DOIS} = \mathbf{-\ 2147483648_{DEZ}}$ ← ***minint***
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{DOIS} = -\ 2147483647_{DEZ}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{DOIS} = -\ 2147483646_{DEZ}$
...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{DOIS} = -\ 3_{DEZ}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{DOIS} = -\ 2_{DEZ}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{DOIS} = -\ 1_{DEZ}$

Complemento de 2

- Colocar um número em complemento de 2 = “inverter” todos os bits e “somar 1” ao valor
- Converter números de N bits em números “com mais” de N bits:
  - Operando imediato MIPS de 16 bits pode ser convertido para 32 bits para operações aritméticas
  - copiar o bit mais significante (o de sinal) nos outros bits

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

4 bits -> 8 bits
```

Adição & Subtração

- Exatamente como se aprende soma e subtração na escola (vai 1)

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

- Operações usando complemento de 2 são simples
  - Ex: subtração a partir da adição de número negativo em complemento de 2

```
  0111          0111
- 0101        + 1011
```

- Overflow (resultado grande demais para uma palavra “finita”):
  - Ex: somar dois número de N-bits nem sempre resulta num número de N-bits

```
  0111
+ 0001
```

Note que o valor com overflow pode gerar dúvidas:
Aqui, ele não indica um “vai-1”, mas um “overflow”.
Monitorar Overflow: monitorar o bit mais significativo nas operações

Detecção de Overflow

- Não há *overflow* quando se soma um positivo com um negativo;
- Não há *overflow* se os sinais são os mesmos numa subtração;
- Um *overflow* ocorre quando o valor afeta o sinal do resultado:
  - quando a soma de 2 positivos dá um negativo;
  - ou, a soma de 2 negativos dá um positivo;
  - ou, subtraindo um negativo de um positivo e ter um negativo;
  - ou, subtraindo um positivo de um negativo e ter um positivo.

Operation	Operand A	Operand B	Result indicating overflow
$A + B$	≥ 0	≥ 0	< 0
$A + B$	< 0	< 0	≥ 0
$A - B$	≥ 0	< 0	< 0
$A - B$	< 0	≥ 0	≥ 0

- Nas operações A + B e A – B, seria possível ocorrer overflow se B=0? E se A=0?

Efeitos do Overflow

- Ocorre uma exceção (interrupção)
  - Fluxo de controle “salta” para um endereço pré-definido para tratamento da exceção de *overflow*
  - O endereço “interrompido” é armazenado para um possível retormada do processamento
- Nem sempre se quer detectar *overflow*
  - Instruções MIPS adicionais:
    - `addu, addiu, subu`

      Obs: `addiu` *guarda a extensão-de-sinal!* `sltu, sltiu` *comparações sem sinal*

MULTIPLICAÇÃO BINÁRIA

Multiplicação Binária

```
    1011    Multiplicando     (11 dec)
   x1101    Multiplicador     (13 dec)
  ------
    1011
   0000         Produtos parciais
  1011
 1011
--------
10001111    Produto           (143 dec)
```

OBS:
1. Resultado possui o dobro do tamanho em bits
2. Funciona apenas com operandos inteiros positivos

Simplificação

```
    1011    Multiplicando
  x 1101    Multiplicador
  ------
```

Procedimento básico:

- Examinar multiplicador da direita para esquerda
- Deslocar multiplicando 1 bit à esquerda a cada passo
- Simplificação: a cada passo somar o multiplicando aos produtos parciais, se multiplicador = 1

Multiplicação

- Obviamente, muito mais complexa que a adição!
  - Realizada através de “deslocamentos” (shift) e “somas” (add)
- Leva mais tempo e ocupa mais área de chip
- Veremos 3 possíveis versões de solução para a multiplicação...

```
    0010   multiplicando
  x 1011   multiplicador
```

- Números Negativos (nas soluções vistas)
  - Converte e multiplica
  - Solução simples, mas pouco eficiente (tempo e hardware)

MULTIPLICAÇÃO BINÁRIA – VERSÃO 1

Multiplicação Inteiros – V1

- Registrador para acumular o produto, inicializado com 0

```
    1000 (multiplicando)
   x1001 (multiplicador)
--------
00000000 (produto parcial)
```

OBS: O valor zero é representado com o dobro de bits do multiplicador e do multiplicando (ex: multiplicação de valores de 8 bits gera produto 16 bits)

Multiplicação Inteiros – V1

- Bit Multiplicador = 1 → adicionar multiplicando ao produto parcial

```
    1000 (multiplicando)
   x1001 (multiplicador)
--------
00000000
 +  1000
--------
00001000 (novo produto parcial)
```

Multiplicação Inteiros – V1

- Deslocar o multiplicando um bit à esquerda

```
   10000  (multiplicando)
 x 1001   (multiplicador)
--------
00000000
+ 1000
--------
00001000 (produto parcial)
```

Multiplicação Inteiros – V1

- Bit Multiplicador = 0 → não fazer nada

```
    10000 (multiplicando)
  x 1001 (multiplicador)
--------
00000000
+   1000
--------
00001000 (produto parcial)
```

Multiplicação Inteiros – V1

- Deslocar o multiplicando um bit à esquerda

```
  100000 (multiplicando)
  x 1001  (multiplicador)
00000000
+   1000
00001000 (produto parcial)
```

Multiplicação Inteiros – V1

- Bit Multiplicador = 0 → não fazer nada

```
  100000 (multiplicando)
   x1001 (multiplicador)
--------
00000000
+   1000
--------
00001000 (produto parcial)
```

Multiplicação Inteiros – V1

- Deslocar o multiplicando um bit à esquerda

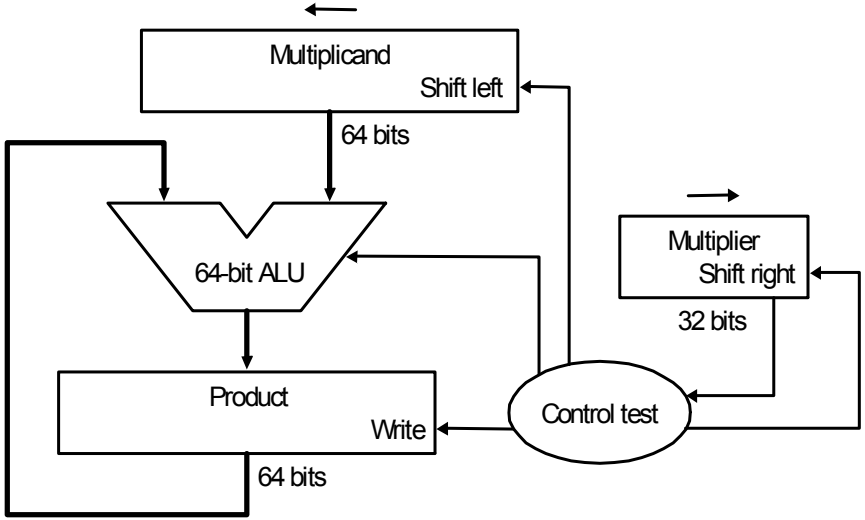
```
  1000000 (multiplicando)
   x 1001 (multiplicador)
 --------
 00000000
+    1000
 --------
 00001000 (produto parcial)
```

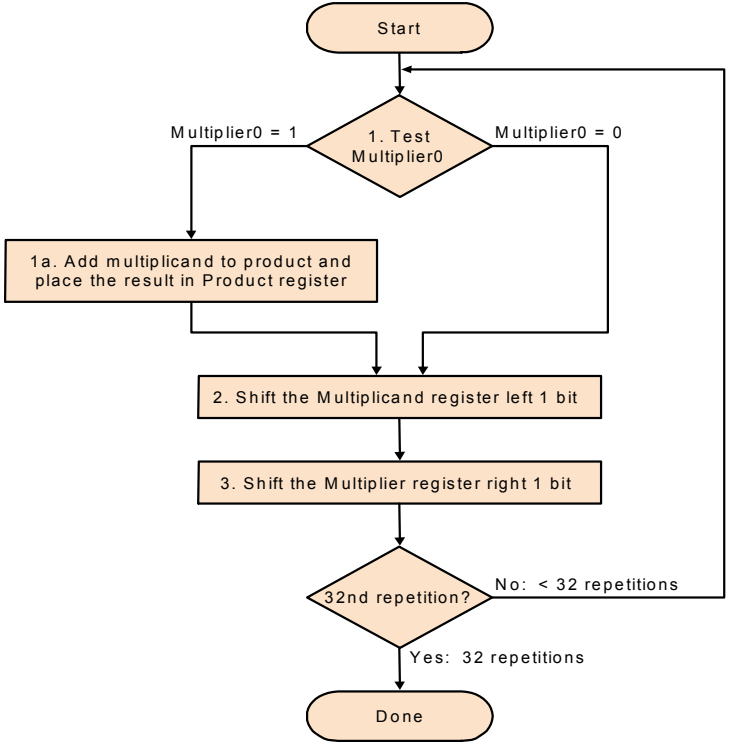
Multiplicação Inteiros – V1

- Bit Multiplicador = 1 → adicionar multiplicando ao produto parcial

```
 1000000 (multiplicando)
   x1001 (multiplicador)
--------
00000000
 +  1000
--------
00001000
+ 1000000
--------
01001000 (produto final)
```

Implementação “sequencial”





MULTIPLICAÇÃO BINÁRIA – VERSÃO 2

Problemas na Versão 1

- A metade dos 64-bits do registrador-multiplicando é 0
- Observe que pelo menos a metade do somador de 64-bits estará somando 0’s!
- Uma solução: deslocar o produto à direita ao invés do multiplicando à esquerda;
- Apenas a metade esquerda do registrador-produto é somada ao multiplicando;
- Versão 2 é uma otimização da versão anterior.

Multiplicação Inteiros – V2

- Uso de um registrador para acumular o produto, inicializado com 0

```
    1000 (multiplicando)
   x1001 (multiplicador)
--------
00000000 (produto parcial, inicializado com zero)
```

Multiplicação Inteiros – V2

- Bit Multiplicador = 1 → adicionar multiplicando ao produto parcial e deslocar o produto parcial à direita

```
    1000 (multiplicando)
   x1001 (multiplicador)
--------
00000000
+1000
--------
10000000 (produto parcial)

01000000 (novo produto parcial)
```

Multiplicação Inteiros – V2

- Bit Multiplicador = 0

→ desloca produto parcial à direita

```
    1000  (multiplicando)
   x1001  (multiplicador)
01000000  (produto parcial)
 01000000 (novo produto parcial)
```

Multiplicação Inteiros – V2

- Bit Multiplicador = 0 → desloca produto parcial à direita

```
   1000 (multiplicando)
  x1001 (multiplicador)
--------
 00010000 (novo produto parcial)
```

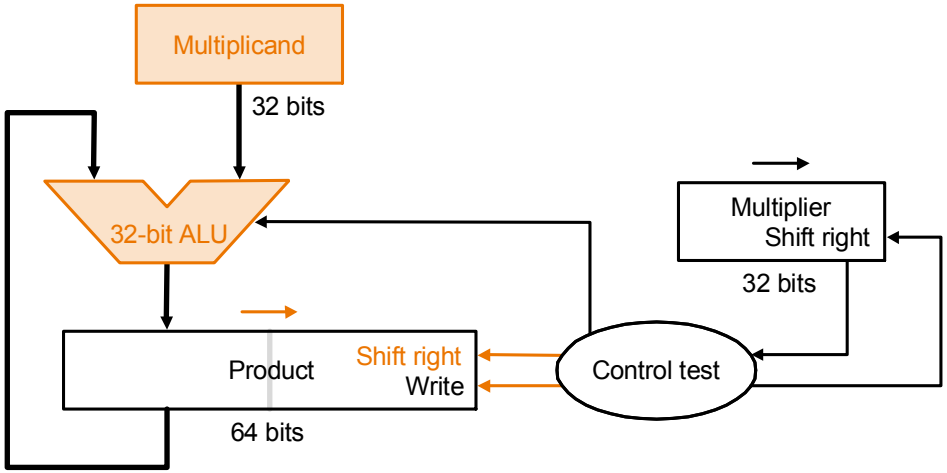
Multiplicação Inteiros – V2

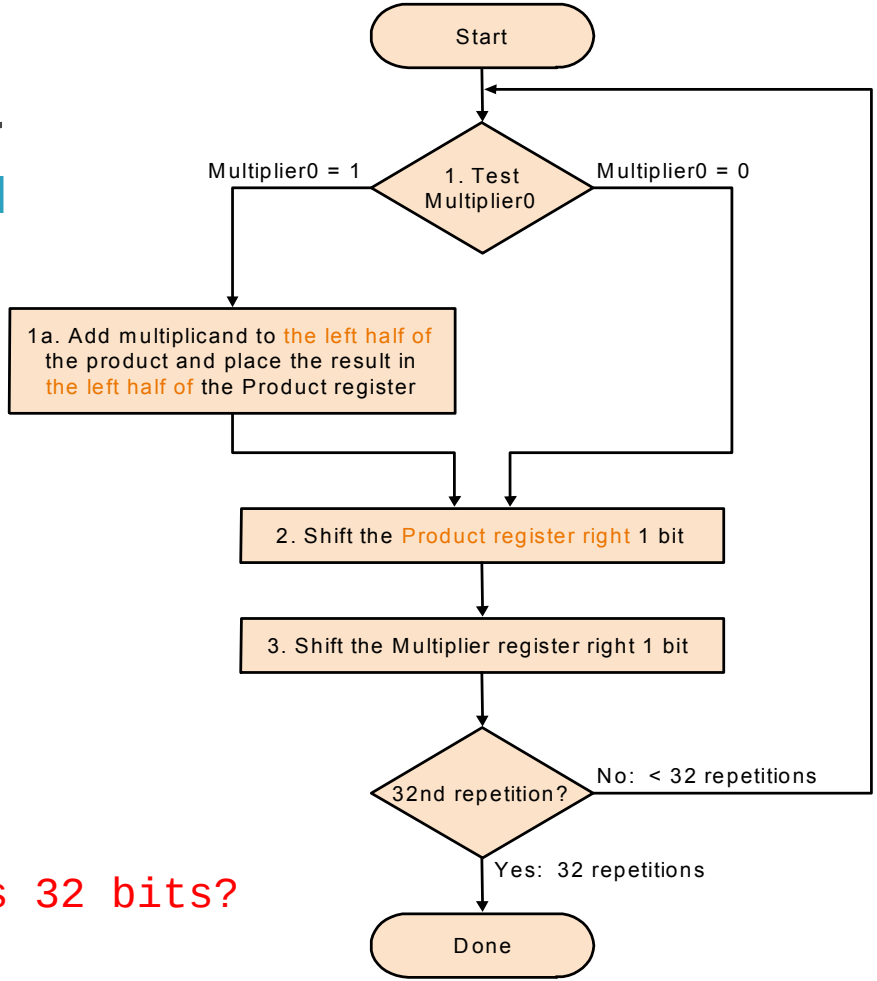
- Bit Multiplicador = 1 → adicionar multiplicando ao produto parcial e deslocar o produto parcial à direita

```
    1000 (multiplicando)
   x1001 (multiplicador)
  -------
  00010000
 +1000
  -------
  10010000 (novo produto parcial)

  01001000 (produto final)
```

Segunda Versã





Os registradores MIPS não têm apenas 32 bits?
Como faço as operações com 64 bits?

Observando a Versão 2

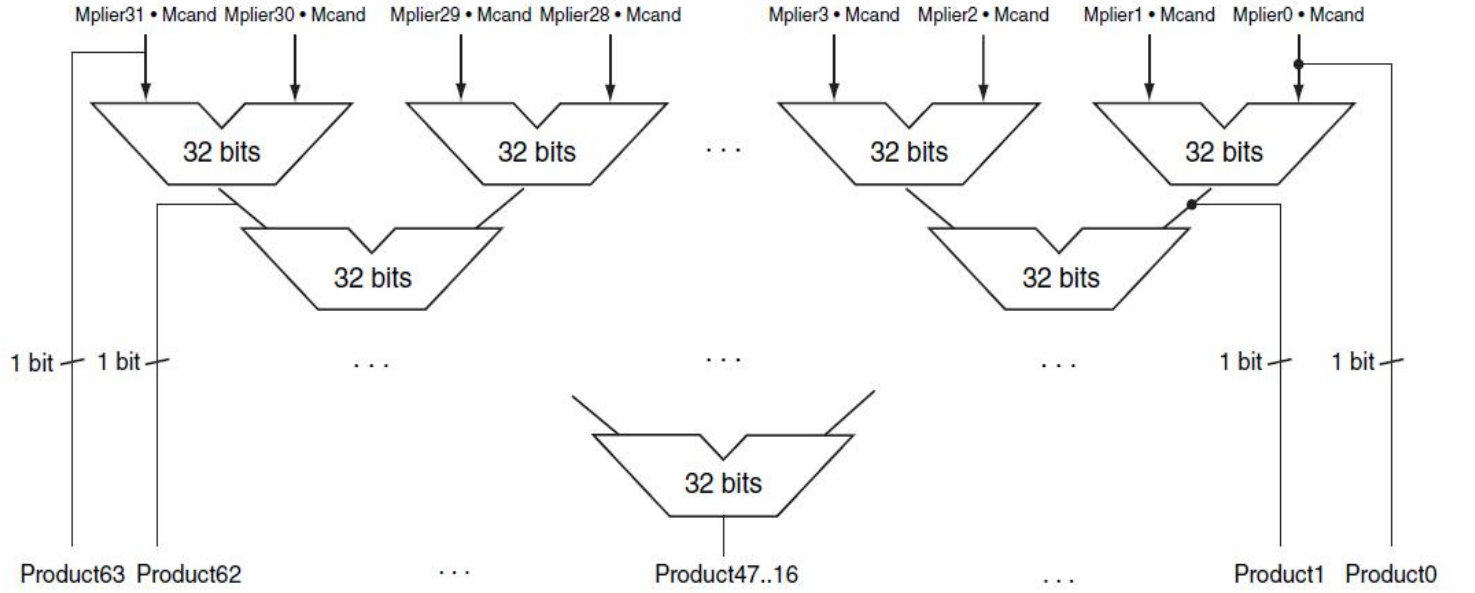


FIGURE 3.8 Fast multiplication hardware. Rather than use a single 32-bit adder 31 times, this hardware "unrolls the loop" to use 31 adders and then organizes them to minimize delay.

MULTIPLICAÇÃO BINÁRIA – VERSÃO 3

Versão 3

- A versão 2 é bastante eficiente com relação à minimização de somas e uso de deslocamentos;
- Uma última melhoria pode ser feita com o uso da metade direita do registrador-produto para o multiplicador;
  - Lembrem-se que o registrador-produto tem o dobro do tamanho dos registradores-operando;
- A versão 3 faz exatamente isto, otimizando a versão 2.

Terceira e Última Versão