CS330 - Computer Organization and Assembly Language Programming

Lecture 7

-Integer Arithmetic-

Professor : Mahmut Unan – UAB CS

Agenda

- Truncating
- Unsigned Addition/ Negation
- Two’s Complement Addition / Negation
- Unsigned Multiplication
- Signed Multiplication
- Booth’s Algorithm

Truncating Numbers

- Reduce the number of bits representing the number
- Truncating *w-bit* number to a *k* bit number, we drop the high order *w-k* bits
  - Can alter its value
    - A form of overflow

Summary: Expanding, Truncating: Basic Rules

- **Expanding (e.g., short int to int)**
  - Unsigned: zeros added
  - Signed: sign extension
  - Both yield expected result

- **Truncating (e.g., unsigned to unsigned short)**
  - Unsigned/signed: bits are truncated
  - Result reinterpreted
  - Unsigned: mod operation
  - Signed: similar to mod
  - For small numbers yields expected behavior

Why should I use unsigned?

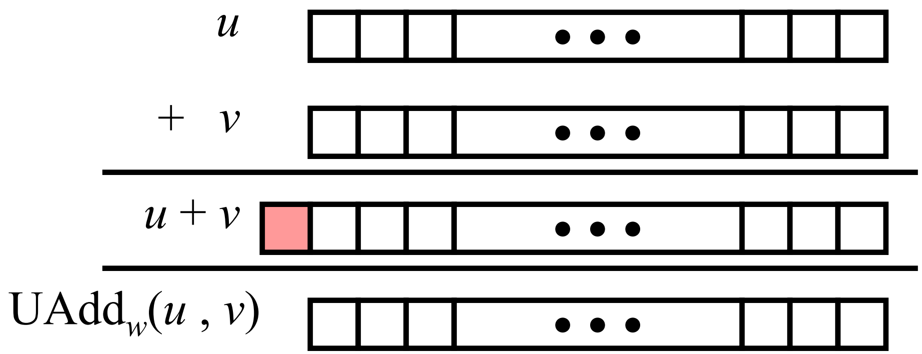
- Don't use just because number nonzero
  - C compilers on some machines generate less efficient code
  - Easy to make mistakes (e.g., casting)
  - Few languages other than C supports unsigned integers
- Do use when need extra bit's worth of range
  - Working right up to limit of word size

Unsigned Addition

Operands: *w* bits

True Sum: *w*+1 bits

Discard Carry: *w* bits



- Standard Addition Function
  - Ignores carry output
- Implements Modular Arithmetic

  $s \quad = \quad \text{UAdd}_w(u\ ,\ v) \quad = \quad u + v \mod 2^w$

Unsigned Addition Example

- 9+12 → 4 bit representation

1001 -> 9

0100 -> 4

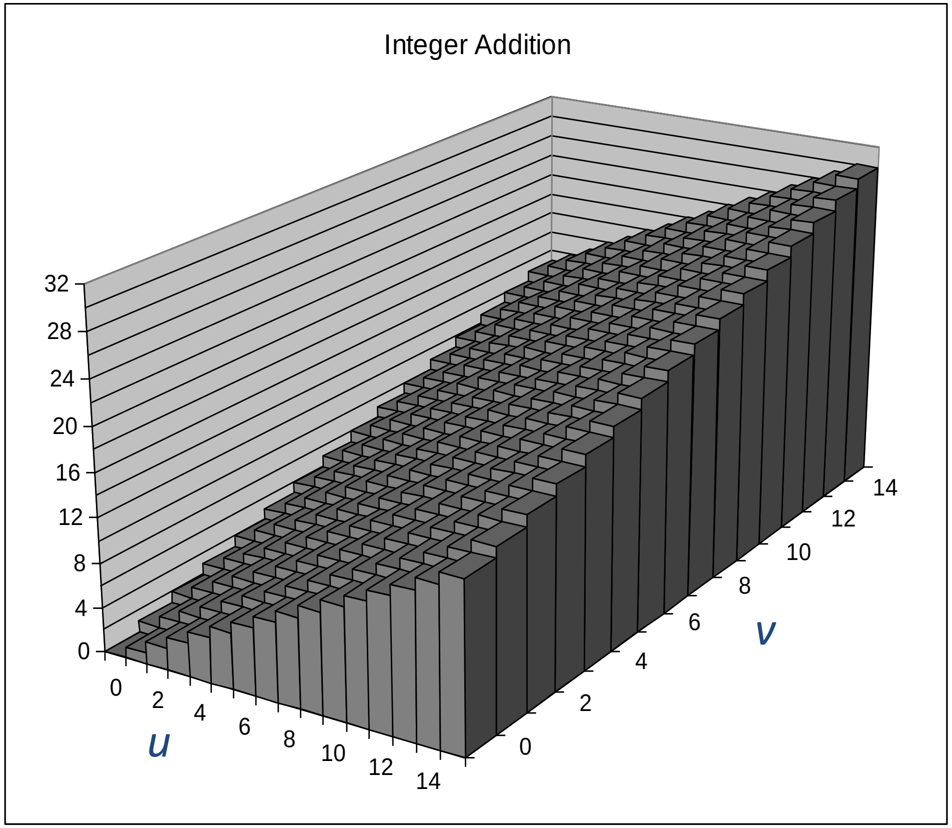
1101

- 13+13 → 4 bit representation
- 1101
- 1101
- 11010

Visualizing (Mathematical) Integer Addition

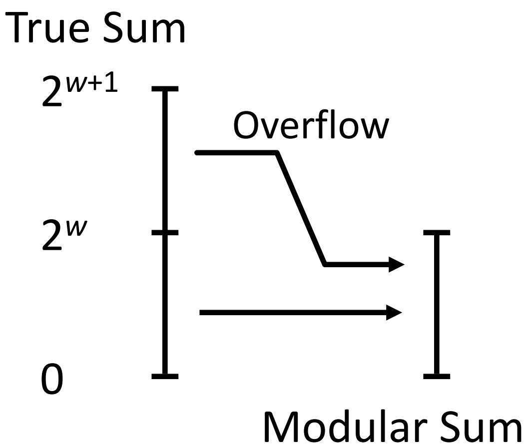
- Integer Addition
  - 4-bit integers *u*, *v*
  - Compute true sum $\text{Add}_4(u\ ,\ v)$
  - Values increase linearly with *u* and *v*
  - Forms planar surface

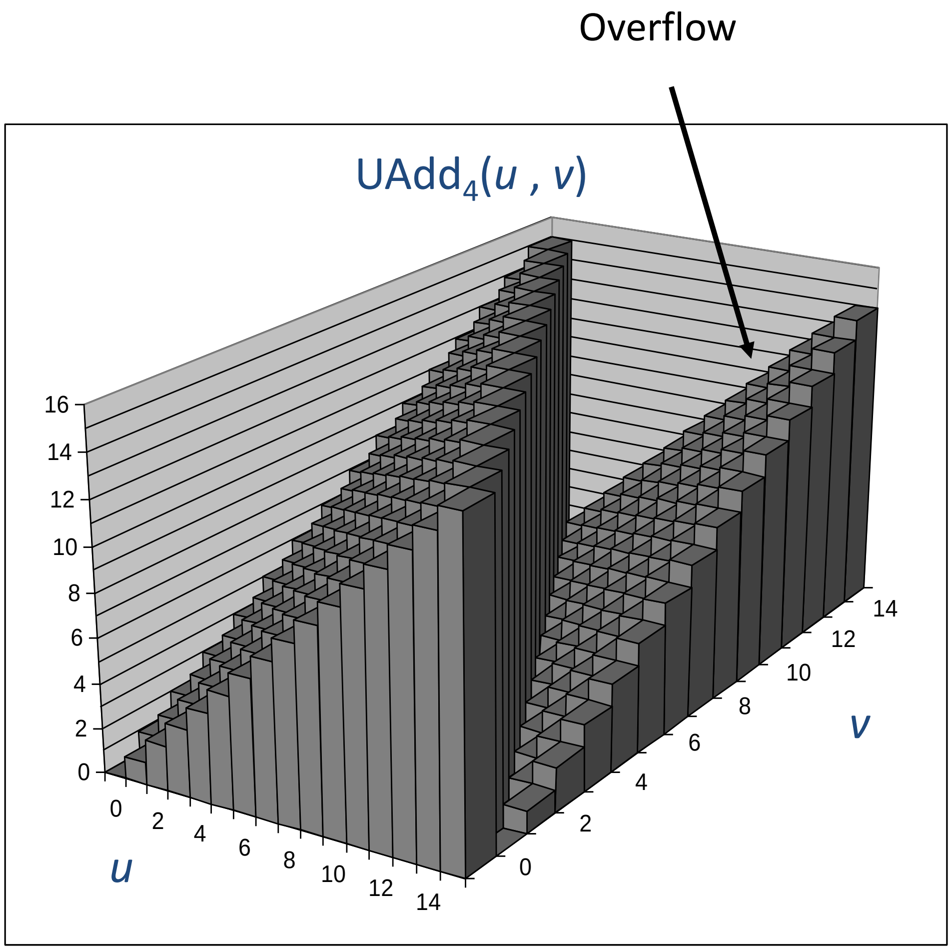
$\text{Add}_4(u\ ,\ v)$



Visualizing Unsigned Addition

- Wraps Around
  - If true sum ≥ 2^w
  - At most once



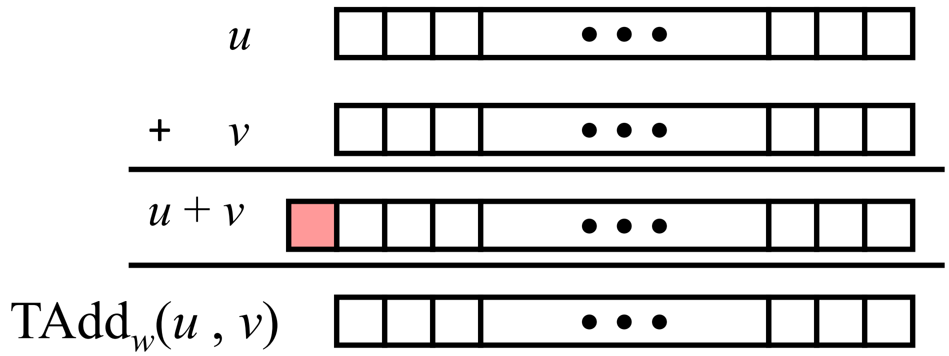


Two’s Complement Addition

Operands: *w* bits

True Sum: *w*+1 bits

Discard Carry: *w* bits



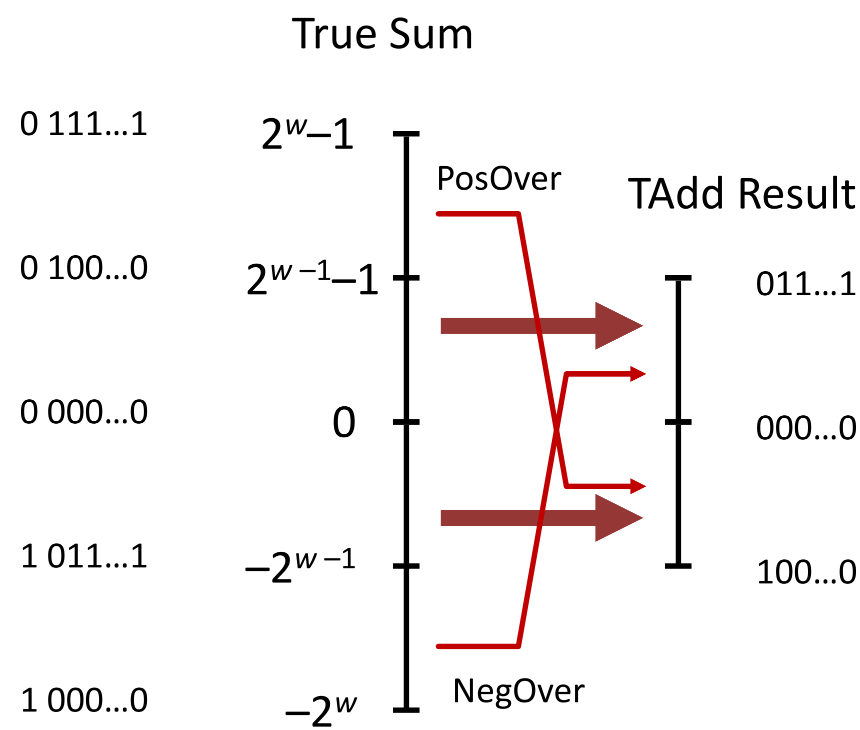
- TAdd and UAdd have Identical Bit-Level Behavior
  - Signed vs. unsigned addition in C:

```
int s, t, u, v;
s = (int) ((unsigned) u + (unsigned) v);
t = u + v
```

  - Will give `s == t`

TAdd Overflow

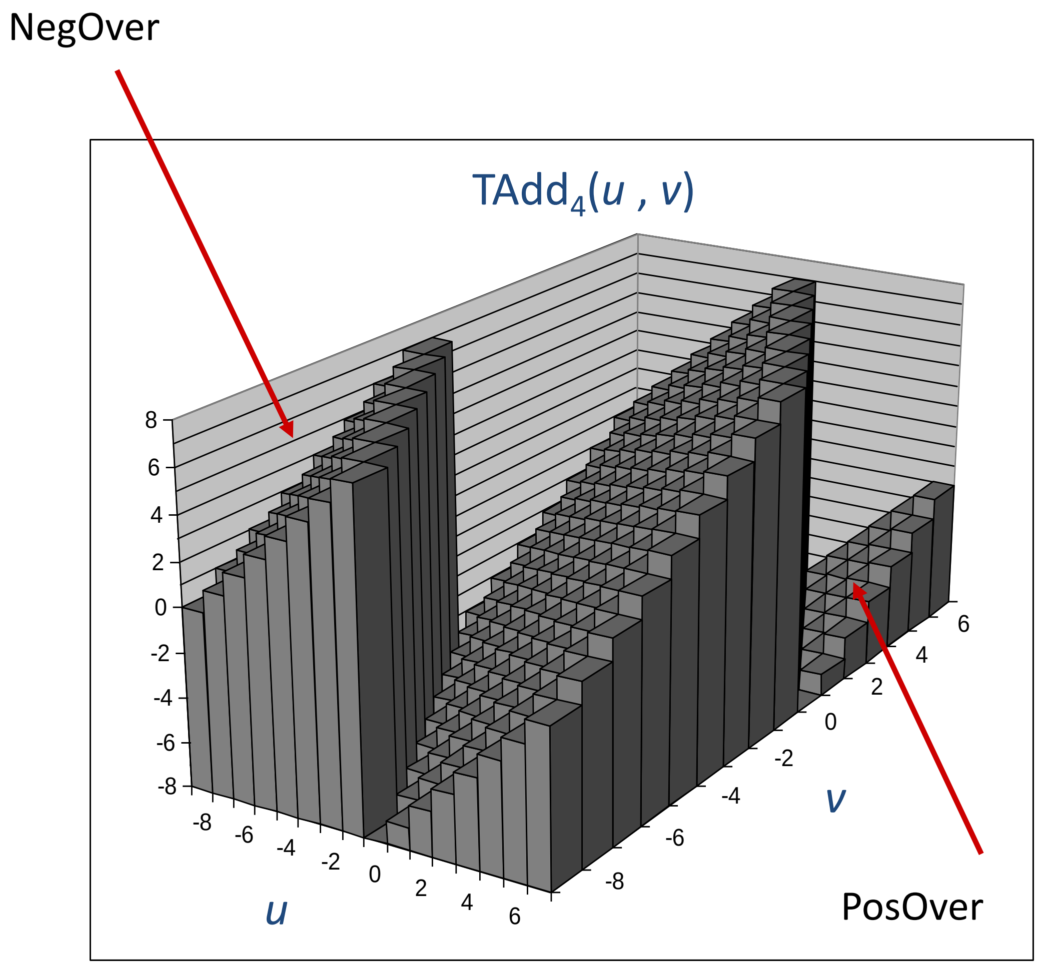
- Functionality
  - True sum requires *w*+1 bits
  - Drop off MSB
  - Treat remaining bits as 2's comp. integer



$$TAdd_w(u,v) = \begin{cases} u+v+2^{w-1} & u+v < TMin_w \quad \textbf{(NegOver)} \\ u+v & TMin_w \le u+v \le TMax_w \\ u+v-2^{w-1} & TMax_w < u+v \quad \textbf{(PosOver)} \end{cases}$$

Visualizing 2's Complement Addition

- Values
  - 4-bit two's comp.
  - Range from -8 to +7
- Wraps Around
  - If sum $\geq 2^{w-1}$
    - Becomes negative
    - At most once
  - If sum $< -2^{w-1}$
    - Becomes positive
    - At most once



1001 = −7 +0101 = 5 1110 = −2 (a) (–7) + (+5)	1100 = −4 +0100 = 4 10000 = 0 (b) (–4) + (+4)
0011 = 3 +0100 = 4 0111 = 7 (c) (+3) + (+4)	1100 = −4 +1111 = −1 11011 = −5 (d) (–4) + (–1)
0101 = 5 +0100 = 4 1001 = Overflow (e) (+5) + (+4)	1001 = −7 +1010 = −6 10011 = Overflow (f) (–7) + (–6)

Figure 10.3 Addition of Numbers in Twos Complement Representation

OVERFLOW RULE:

- If two numbers are added, and they are both positive or both negative, then overflow occurs if and only if the result has the opposite sign.

```
  0010 =  2
 +1001 = -7
  1011 = -5

(a) M = 2 = 0010
    S = 7 = 0111
   -S =     1001
```

```
  0101 =  5
 +1110 = -2
 10011 =  3

(b) M = 5 = 0101
    S = 2 = 0010
   -S =     1110
```

```
  1011 = -5
 +1110 = -2
 11001 = -7

(c) M =-5 = 1011
    S = 2 = 0010
   -S =     1110
```

```
  0101 = 5
 +0010 = 2
  0111 = 7

(d) M = 5 = 0101
    S =-2 = 1110
   -S =     0010
```

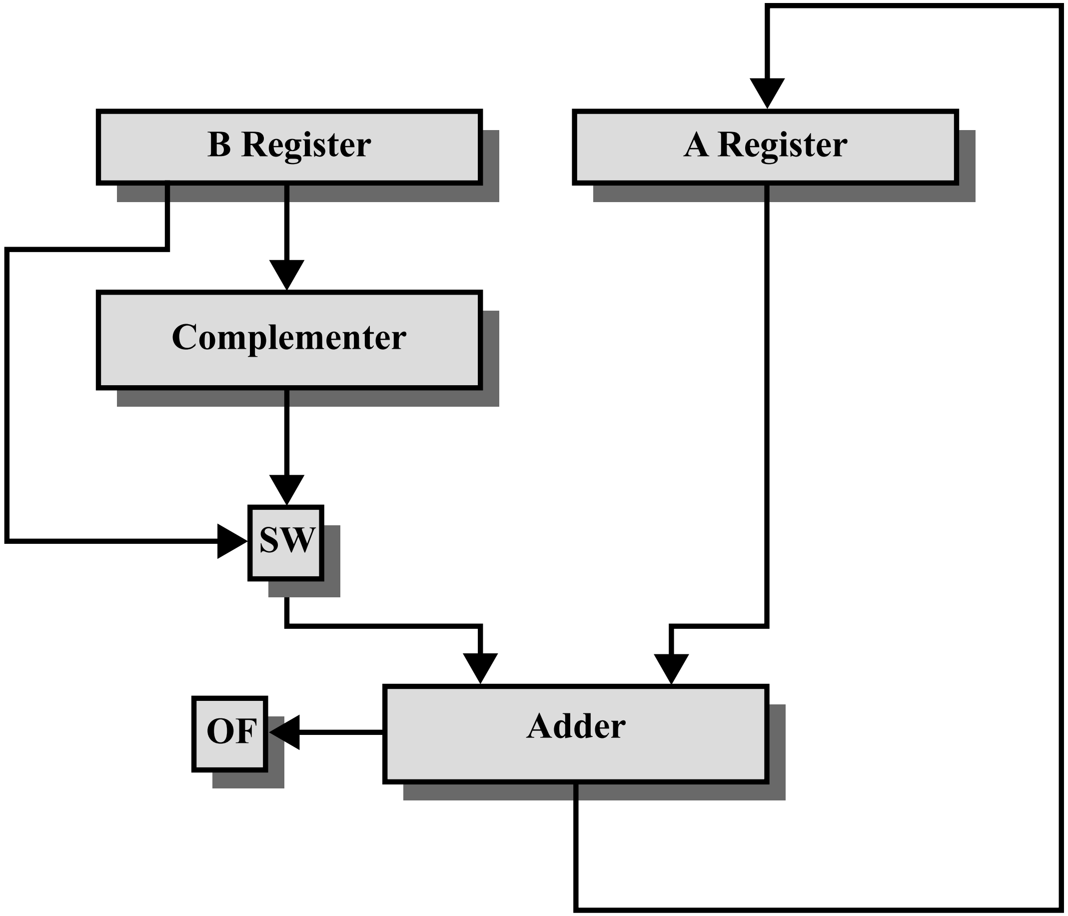
```
  0111 = 7
 +0111 = 7
  1110 = Overflow

(e) M =  7 = 0111
    S = -7 = 1001
   -S =      0111
```

```
  1010 = -6
 +1100 = -4
 10110 = Overflow

(f) M = -6 = 1010
    S =  4 = 0100
   -S =      1100
```

Figure 10.4 Subtraction of Numbers in Twos Complement Representation (M – S)

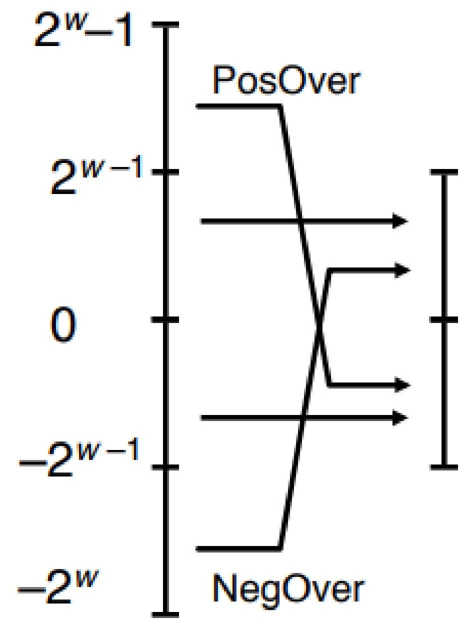


OF = overflow bit
SW = Switch (select addition or subtraction)

Figure 10.6 Block Diagram of Hardware for Addition and Subtraction

Detecting 2's comp. overflow

- Task
  - Given s = TAddw(u , v)
  - Determine if s = Addw(u , v)
  - Example
  - int s, u, v;
  - s = u + v;



- Claim
  - Overflow iff either:
    - $u, v < 0, s \geq 0$ (NegOver)
    - $u, v \geq 0, s < 0$ (PosOver)

```
ovf = (u<0 == v<0) && (u<0 != s<0);
```

Exercises

- Assume numbers are represented in 8 bit two's complement representation. Show the calculation of the followings;
  - **6+13**
  - 0000 0110
  - 0000 1101
  - 0001 0011

  - **12-5**
  - 0000 1100
  - 1111 1011
  - 10000 0111

  - **7-2**
  - 0000 0111
  - 1111 1110
  - 10000 0101
  - **65 - 33**

- $-39 + 92 = 53$:

$$
\begin{array}{ccccccccc}
\boxed{1} & 1 & & 1 & 1 & & & & \\
 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
+ & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
\hline
 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1
\end{array}
$$

Carryout without overflow. Sum is correct.

- -19 + -7 = -26:

$$
\begin{array}{ccccccccc}
\boxed{1} & 1 & 1 & 1 & 1 & & & 1 & \\
 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
+ & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\
\hline
 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0
\end{array}
$$

Carryout without overflow. Sum is correct.

- 44 + 45 = 89:

$$
\begin{array}{ccccccccc}
 & & 1 & & 1 & 1 & & & \\
 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
+ & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
\hline
 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1
\end{array}
$$

No overflow nor carryout.

- 104 + 45 = 149:

$$
\begin{array}{cccccccccc}
 & 1 & 1 & & 1 & & & & \\
 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
+ & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
\hline
 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1
\end{array}
$$

Overflow, no carryout. Sum is not correct.

- -103 + -69 = -172:

$$
\begin{array}{ccccccccc}
\boxed{1} & & 1 & 1 & 1 & & 1 & 1 & \\
 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
+ & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
\hline
 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0
\end{array}
$$

Overflow, with incidental carryout. Sum is not correct.

http://sandbox.mc.edu/~bennet/cs110/tc/add.html

- $127 + 1 = 128$:

$$
\begin{array}{ccccccccc}
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \\
 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
+ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

Overflow, no carryout. Sum is not correct.

Multiplication

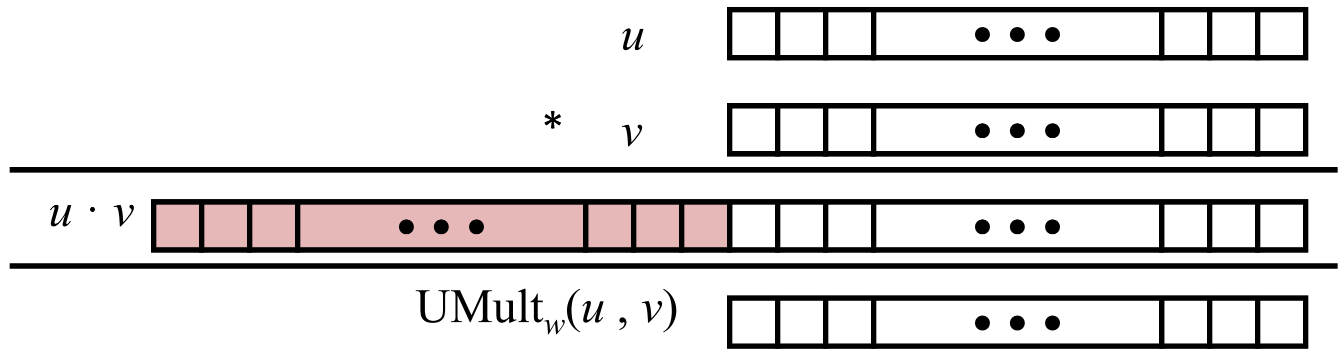
- Goal: Computing Product of w-bit numbers x, y
  - Either signed or unsigned
- But, exact results can be bigger than w bits
  - Unsigned: up to $2w$ bits
    - Result range: $0 \leq x * y \leq (2^{w} - 1)^{2} = 2^{2w} - 2^{w+1} + 1$
  - Two's complement min (negative): Up to $2w$-1 bits
    - Result range: $x * y \geq (-2^{w-1})*(2^{w-1}-1) = -2^{2w-2} + 2^{w-1}$
  - Two's complement max (positive): Up to $2w$ bits, but only for $(TMin_{w})^{2}$
    - Result range: $x * y \leq (-2^{w-1})^{2} = 2^{2w-2}$
- So, maintaining exact results…
  - would need to keep expanding word size with each product computed
  - is done in software, if needed
    - e.g., by "arbitrary precision" arithmetic packages

Unsigned Multiplication in C

Operands: w bits

True Product: $2*w$ bits

Discard w bits: w bits



- Standard Multiplication Function
  - Ignores high order w bits
- Implements Modular Arithmetic

  $\mathrm{UMult}_w(u, v) = u \cdot v \bmod 2^w$

Unsigned Binary Multiplication

1011	**Multiplicand (11)**
×1101	**Multiplier (13)**
1011	
0000	**Partial products**
1011	
1011	
10001111	**Product (143)**

Figure 10.7 Multiplication of Unsigned Binary Integers

Unsigned Multiplication

- 7 * 4

7 = 0111

4 = 0100

Unsigned Multiplication

- 7 * 4

7 = 0111

4 = 0100

```
   0111
   0100
x_______
   0000
```

Unsigned Multiplication

- 7 * 4

7 = 0111

4 = 0100

```
   0111
   0100
x_______
   0000
  0000
```

Unsigned Multiplication

- 7 * 4

7 = 0111

4 = 0100

```
     0111
     0100
x__________
     0000
    0000
   0111
```

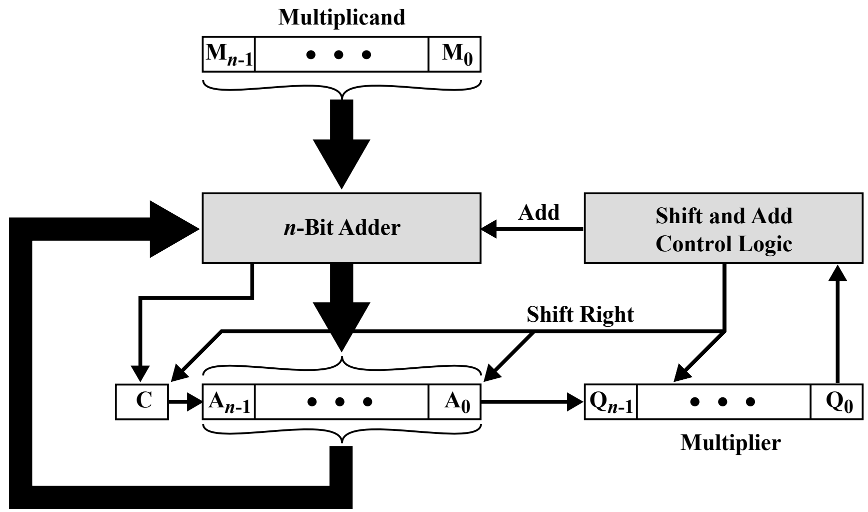
Unsigned Multiplication

- 7 * 4

7 = 0111

4 = 0100

```
      0111
      0100
 x_________
      0000
     0000
    0111
+  0000
___________
   0011100
   = 28
```



(a) Block Diagram

C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

(b) Example from Figure 9.7 (product in A, Q)

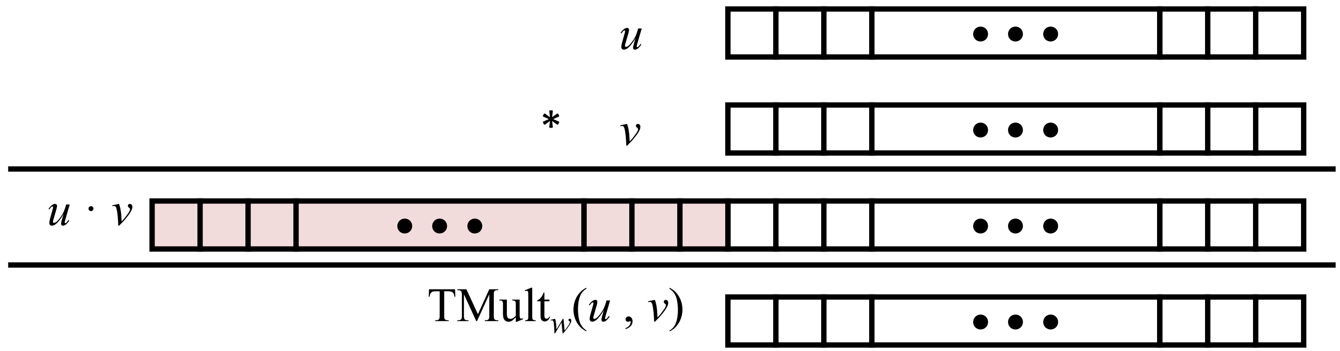
Figure 10.8 Hardware Implementation of Unsigned Binary Multiplication

Signed Multiplication in C

Operands: *w* bits

True Product: 2**w* bits

Discard *w* bits: *w* bits



- Standard Multiplication Function
  - Ignores high order *w* bits
  - Some of which are different for signed vs. unsigned multiplication
  - Lower bits are the same

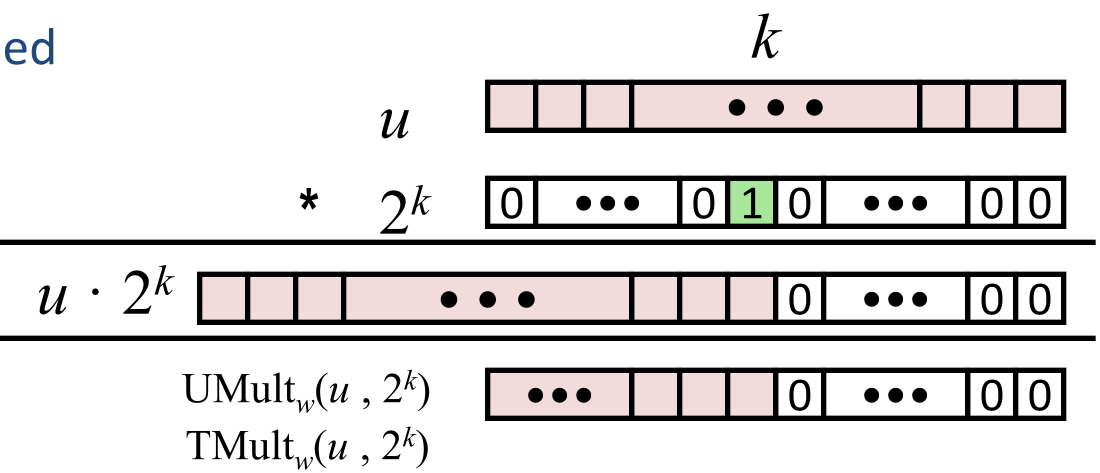
Power-of-2 Multiply with Shift

- **Operation**
  - **`u << k`** gives **`u * 2`**^{k}
  - Both signed and unsigned

Operands: *w* bits

True Product: *w+k* bits

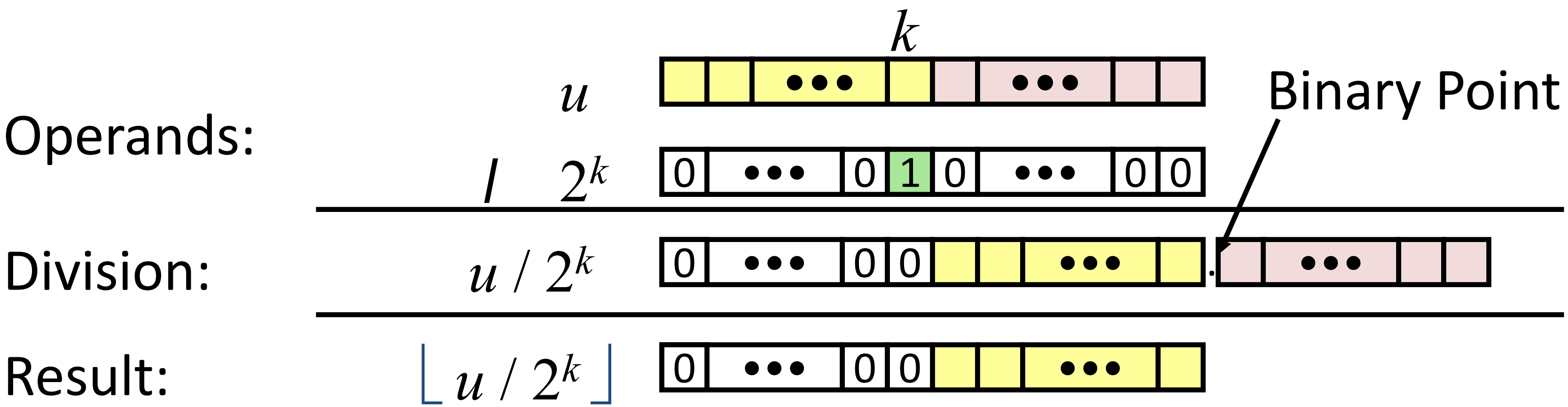
Discard *k* bits: *w* bits



- **Examples**
  - **`u << 3 == u * 8`**
  - **`(u << 5) - (u << 3) == u * 24`**
  - Most machines shift and add faster than multiply
    - Compiler generates this code automatically

Unsigned Power-of-2 Divide with Shift

- **Quotient of Unsigned by Power of 2**
  - `u >> k` gives $\lfloor \mathtt{u} / 2^k \rfloor$
  - Uses logical shift



	Division	Computed	Hex	Binary
`x`	15213	15213	`3B 6D`	`00111011 01101101`
`x >> 1`	7606.5	7606	`1D B6`	`00011101 10110110`
`x >> 4`	950.8125	950	`03 B6`	`00000011 10110110`
`x >> 8`	59.4257813	59	`00 3B`	`00000000 00111011`

Booth's Algorithm

- A Q Q_{-1} Action

----

00 → Shift (Arithmetic Right Shift)

01→ Add Numbers (A=A+M), Shift

10→ Subtract Numbers (A=A-M), Shift

11→ Shift (Arithmetic Right Shift)

Exercises

- 6*(-2)
- 5*(-4)
- 14*(-5)

6 * (-2) =

- A Q Q_{-1} Action

----

6 → 0110 → Multiplicand →M

6 * (-2) =

- A Q Q_{-1} Action

----

6 → 0110 → Multiplicand → M

1001

1

+__________

-M = 1010

6 * (-2) =

- A Q Q_{-1} Action

----

6 → 0110 → Multiplicand →M

1001

1

+__________

-M = 1010

2→ 0010 → Multiplier → Q

1101

1

+__________

1110

6 * (-2) =

- A Q Q_{-1} Action

----

6 → 0110 → Multiplicand → M

1001

1

+___________

-M = 1010

2→ 0010 → Multiplier → Q

1101

1

+___________

-2→1110

A =0000

$Q_{-1} = 0$

M=0110 -M = 1010 Q=1110 A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000	1110	0	00→ Shift(AR)
2				
3				
4				

M=0110 -M = 1010 Q=1110 A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000	1110	0	
2				
3				
4				

4 Bits→ 4Iterations
Result = 8 bits

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

$M=0110 \quad -M = 1010 \quad Q=1110 \quad A=0000 \quad Q_{-1}=0$

Step	A	Q	Q_{-1}	Action
1	0000 0000	1110 0111	0 0	00→ Shift(AR)
2				
3				
4				

Arithmetic Right Shift → Duplicate the left most bit

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=0110 -M = 1010 Q=1110 A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000 0000	1110 0111	0 0	00→ Shift(AR)
2	0000 1010 1101	0111 0111 0011	0 0 1	10 → Subtract (A=A-M) A = 0000 -M =1010 A-M=1010 Shift(AR)
3				
4				

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=0110 -M = 1010 Q=1110 A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000 0000	1110 0111	0 0	00→ Shift(AR)
2	0000 1010 1101	0111 0111 0011	0 0 1	10 → Subtract (A=A-M) A = 0000 -M =1010 A-M=1010 Shift(AR)
3	1101 1110	0011 1001	1 1	11→ Shift(AR)
4				

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=0110   -M = 1010 Q=1110   A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000 0000	1110 0111	0 0	00→ Shift(AR)
2	0000 1010 1101	0111 0111 0011	0 0 1	10 → Subtract (A=A-M) A = 0000 -M =1010 A-M=1010 Shift(AR)
3	1101 1110	0011 1001	1 1	11→ Shift(AR)
4	1110 1111	1001 0100	1 1	11→ Shift(AR)

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=0110 -M = 1010 Q=1110 A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000 0000	1110 0111	0 0	00→ Shift(AR)
2	0000 1010 1101	0111 0111 0011	0 0 1	10 → Subtract (A=A-M) A = 0000 -M =1010 A-M=1010 Shift(AR)
3	1101 1110	0011 1001	1 1	11→ Shift(AR)
4	1110 1111	1001 0100	1 1	11→ Shift(AR)

1111 0100 = -12

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

Example 2 → 5 * -4

- A Q Q_{-1} Action

----

5 → 0101 → Multiplicand → M

1010

1

+__________

-M = 1011

4→ 0100 → Multiplier → Q

1011

1

+__________

-4→1100

M=0101 -M = 1011 Q=1100 A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000	1110	0	00→ Shift(AR)
2				
3				
4				

4 Bits→ 4Iterations
Result = 8 bits

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=0101 -M = 1011 Q=1100 A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000 0000	1100 0110	0 0	00→ Shift(AR)
2				
3				
4				

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=0101   -M = 1011 Q=1100   A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000 0000	1100 0110	0 0	00→ Shift(AR)
2	0000 0000	0110 0011	0 0	00→ Shift(AR)
3				
4				

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

$M=0101 \quad -M = 1011 \quad Q=1100 \quad A=0000 \quad Q_{-1}=0$

Step	A	Q	Q_{-1}	Action
1	0000 0000	1100 0110	0 0	00→ Shift(AR)
2	0000 0000	0110 0011	0 0	00→ Shift(AR)
3	0000 1011 1101	0011 0011 1001	0 0 1	10→ Subtract A=0000 -M= 1011 A=A-M → 1011 Shift(AR)
4				

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

$M=0101 \quad -M = 1011 \quad Q=1100 \quad A=0000 \quad Q_{-1} = 0$

Step	A	Q	Q_{-1}	Action
1	0000 0000	1100 0110	0 0	00→ Shift(AR)
2	0000 0000	0110 0011	0 0	00→ Shift(AR)
3	0000 1011 1101	0011 0011 1001	0 0 1	10→ Subtract A=0000 -M= 1011 A=A-M → 1011 Shift(AR)
4	1101 1110	1001 1100	1 1	11→ Shift(AR)

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

$M=0101 \quad -M = 1011 \quad Q=1100 \quad A=0000 \quad Q_{-1}=0$

Step	A	Q	Q_{-1}	Action
1	0000 0000	1100 0110	0 0	00→ Shift(AR)
2	0000 0000	0110 0011	0 0	00→ Shift(AR)
3	0000 1011 1101	0011 0011 1001	0 0 1	10→ Subtract A=0000 -M= 1011 A=A-M → 1011 Shift(AR)
4	1101 1110	1001 1100	1 1	11→ Shift(AR)

1110 1100 =-20

00	Arithmetic Right Shift
11	Arithmetic Right Shift
01	ADD (A=A+M)+ Shift
10	SUBTRACT (A=A-M)+Shift

Example 3 → 14 * -5

- A Q Q_{-1} Action

---

14 → 01110 → Multiplicand → M

5 Bits→ 5 Iterations
Result = 10 bits

Example 3 → 14 * -5

- A Q Q_{-1} Action

----

14 → 01110 → Multiplicand → M

10001

1

+__________

-M = 10010

5→ 00101 → Multiplier → Q

11010

1

+__________

-5→11011

M=01110 -M = 10010 Q=11011 A=00000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	00000	11011	0	
2				
3				
4				
5				

5 Bits→ 5 Iterations
Result = 10 bits

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=01110 -M = 10010 Q=11011 A=00000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	00000 10010 11001	11011 11011 01101	0 0 1	10→ SUBTRACT (A=A-M)+Shift A = 00000 -M = 10010 A-M= 10010 Shift (AR)
2				
3				
4				
5				

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=01110 -M = 10010 Q=11011 A=00000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	00000	11011	0	10→ SUBTRACT (A=A-M)+Shift A = 00000 -M = 10010
	10010	11011	0	A-M= 10010
	11001	01101	1	Shift (AR)
2	11001	01101	1	Shift (AR)
	11100	10110	1	
3				
4				
5				

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=01110 -M = 10010 Q=11011 A=00000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	00000 10010 11001	11011 11011 01101	0 0 1	10→ SUBTRACT (A=A-M)+Shift A = 00000 -M = 10010 A-M= 10010 Shift (AR)
2	11001 11100	01101 10110	1 1	Shift (AR)
3	11100 01010 00101	10110 10110 01011	1 1 0	01→ ADD (A=A+M)+ Shift A = 11100 M = 01110 Carry - ignored A+M= 01010 → Actually 101010 Shift (AR)
4				
5				

M=01110   -M = 10010 Q=11011   A=00000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	00000 10010 11001	11011 11011 01101	0 0 1	10→ SUBTRACT (A=A-M)+Shift A = 00000 -M = 10010 A-M= 10010 Shift (AR)
2	11001 11100	01101 10110	1 1	Shift (AR)
3	11100 01010 00101	10110 10110 01011	1 1 0	01→ ADD (A=A+M)+ Shift A = 11100 M = 01110 Carry - ignored A+M= 01010 → Actually 101010 Shift (AR)
4	00101 10111 11011	01011 01011 10101	0 0 1	10→ SUBTRACT (A=A-M)+Shift A = 00101 -M = 10010 A-M= 10111 Shift (AR)
5				

$M=01110 \quad -M = 10010 \quad Q=11011 \quad A=00000 \quad Q_{-1}=0$

Step	A	Q	Q_{-1}	Action
1	00000	11011	0	10→ SUBTRACT (A=A-M)+Shift
				A = 00000
				-M = 10010
	10010	11011	0	A-M= 10010
	11001	01101	1	Shift (AR)
2	11001	01101	1	Shift (AR)
	11100	10110	1	
3	11100	10110	1	01→ ADD (A=A+M)+ Shift
				A = 11100
				M = 01110 Carry - ignored
	01010	10110	1	A+M= 01010 → Actually 101010
	00101	01011	0	Shift (AR)
4	00101	01011	0	10→ SUBTRACT (A=A-M)+Shift
				A = 00101
				-M = 10010
	10111	01011	0	A-M= 10111
	11011	10101	1	Shift (AR)
5	11011	10101	1	Shift (AR)
	11101	11010	1	

M=01110 -M = 10010 Q=11011 A=00000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	00000	11011	0	10→ SUBTRACT (A=A-M)+Shift
				A = 00000
				-M = 10010
	10010	11011	0	A-M= 10010
	11001	01101	1	Shift (AR)
2	11001	01101	1	Shift (AR)
	11100	10110	1	
3	11100	10110	1	01→ ADD (A=A+M)+ Shift
				A = 11100
				M = 01110
	01010	10110	1	A+M= 01010 → Actually 101010
	00101	01011	0	Shift (AR)
4	00101	01011	0	10→ SUBTRACT (A=A-M)+Shift
				A = 00101
				-M = 10010
	10111	01011	0	A-M= 10111
	11011	10101	1	Shift (AR)
5	11011	10101	1	Shift (AR)
	11101	11010	1	

Carry - ignored

11101 11010 =-70