Booth's Algorithm

- There is more than one way to do multiplication
- An ALU can add or subtract
  - We can have: 6 = -2 +8
- A multiplication (A*B) can be performed as
  - A*(C-D); where B=C-D
  - Assume A=2 and B=6 (C=8 and D=2)

    2*(8-2)=12

Booth's Algorithm (cont'd)

```
         0 0 1 0 two
         0 1 1 0 two
   -------------------------
         0 0 0 0   shift (0)
       0 0 1 0     add   (1)
     0 0 1 0       add   (1)
   0 0 0 0         shift (0)
---------------------------------
   0 0 0 1 1 0 0 two
```

Booth's Algorithm (cont'd)

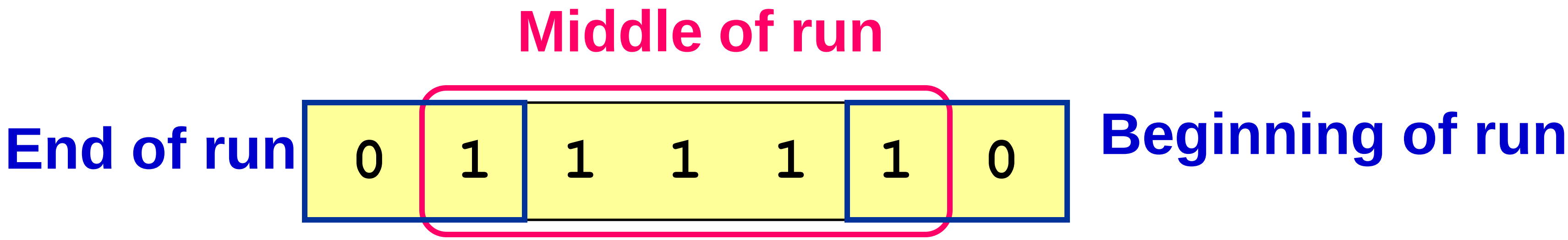
```
          0 0 1 0  two
          0 1 1 0  two
-------------------------
          0 0 0 0   shift (0)
        0 0 1 0     sub   (first 1)
      0 0 0 0       shift (string of 1s)
    0 0 1 0         add   (last 1 in string)
-------------------------
    0 0 0 1 1 0 0  two
```

Booth's Algorithm (cont'd)

```
        0 0 1 0 two
        0 1 1 0 two
   ----------------------
        0 0 0 0   shift (0)
  1 1 1 1 1 0     sub   (first 1)
    0 0 0 0       shift (string of 1s)
   0 0 1 0        add   (last 1 in string)
   ----------------------
  0 0 0 1 1 0 0 two
```

Two's complement

Booth's Algorithm (cont'd)



Using Booth's algorithm

Check current bit and previous bit

- 0 0 Middle of string of 0s, NO ARITHMETIC OPERATION
- 0 1 End of string of 1s, ADD THE MULTIPLICAND
- 1 0 Beginning of string of 1s, SUBTRACT THE MULTIPLICAND
- 1 1 Middle of string of 1s, NO ARITHMETIC OPERATION

Shift product to right (1 bit)

Booth's algorithm example

2 X **-3** **0010** x **1101**

Multiplicand Multiplier

Current bit Previous bit

product

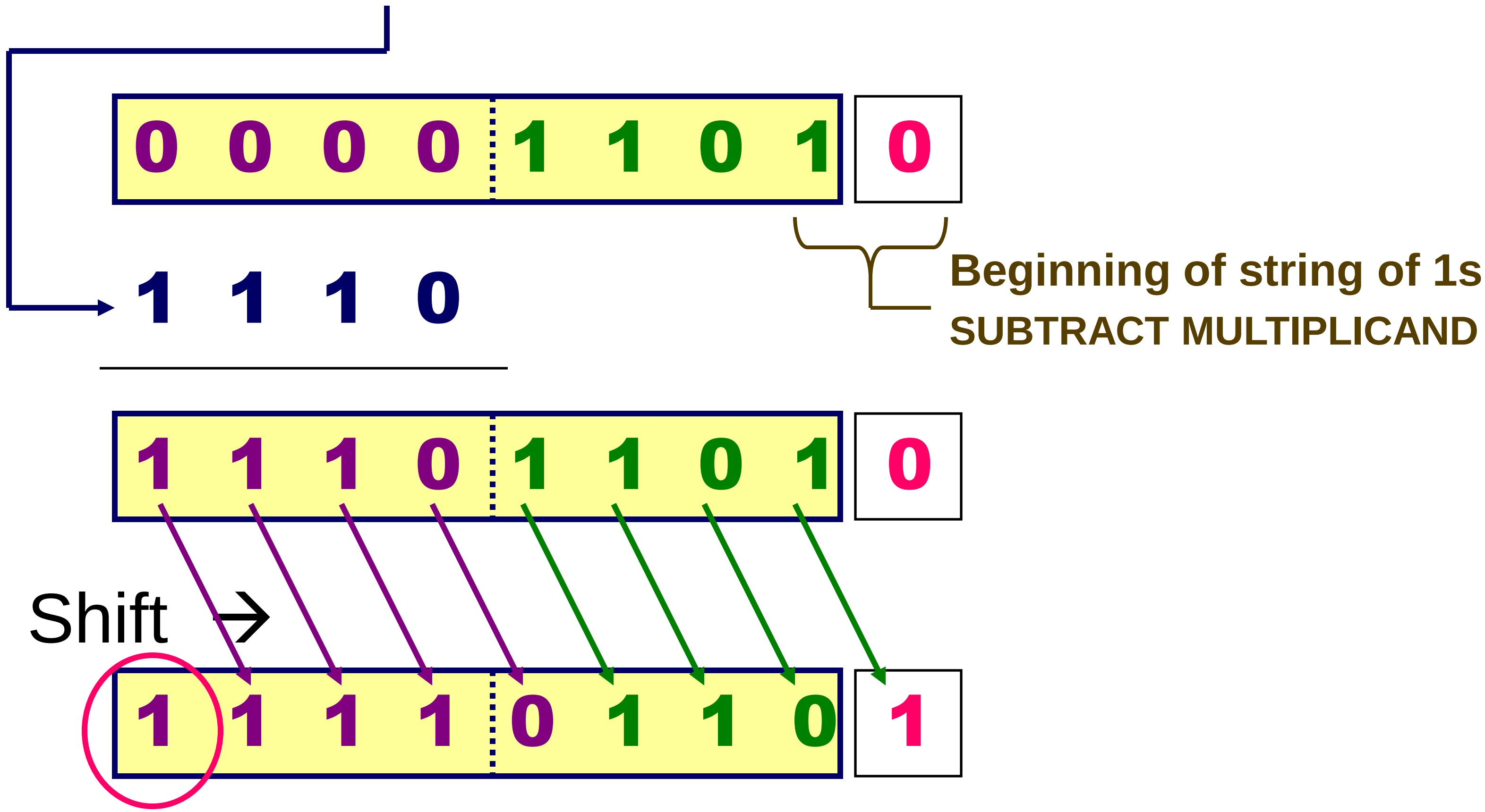
0	0	0	0	1	1	0	1	0

multiplier

Booth’s algorithm example

2 X -3 **0010** x **1101**

Multiplicand Multiplier



Booth's algorithm example

2 X -3 **0010** x **1101**

Multiplicand Multiplier

1	1	1	1	0	1	1	0	1

End of string of 1s

ADD MULTIPLICAND

0 0 1 0

0	0	0	1	0	1	1	0	1

Shift →

0	0	0	0	1	0	1	1	0

Booth's algorithm example

2 X -3 **0010** x **1101**

Multiplicand Multiplier

0	0	0	0	1	0	1	1	0

1 1 1 0

Beginning of string of 1s

SUBSTRACT MULTIPLICAND

1	1	1	0	1	0	1	1	0

Shift →

1	1	1	1	0	1	0	1	1

Booth's algorithm example

2 X -3 **0010** x **1101**

Multiplicand Multiplier

1	1	1	1	0	1	0	1	1

Middle of string of 1s

NO ARITHMETIC OPERATION

Shift → (last shift)

1	1	1	1	1	0	1	0	1

Booth's algorithm example

1 1 1 1 1 0 1 0

Two's complement

0 0 0 0 0 1 0 1

1

---

0 0 0 0 0 1 1 0

8-bit multiplication

Multiplicand: 35 → 0 0 1 0 0 0 1 1

Multiplier: 79 → 0 1 0 0 1 1 1 1

0 0 0 0 0 0 0 0	0 1 0 0 1 1 1 1	0
1 1 0 1 1 1 0 1		
1 1 0 1 1 1 0 1	0 1 0 0 1 1 1 1	0

Multiplicand: 35 → 0 0 1 0 0 0 1 1

Multiplier: 79 → 0 1 0 0 1 1 1 1

(1) Shift →

1 1 1 0 1 1 1 0	1 0 1 0 0 1 1 1	1

(2) Shift →

1 1 1 1 0 1 1 1	0 1 0 1 0 0 1 1	1

(3) Shift →

1 1 1 1 1 0 1 1	1 0 1 0 1 0 0 1	1

(4) Shift →

1 1 1 1 1 1 0 1	1 1 0 1 0 1 0 0	1

Multiplicand: 35 → 0 0 1 0 0 0 1 1

Multiplier: 79 → 0 1 0 0 1 1 1 1

1	1	1	1	1	1	0	1	1	1	0	1	0	1	0	0	1

0 0 1 0 0 0 1 1

0	0	1	0	0	0	0	0	1	1	0	1	0	1	0	0	1

(5) shift →

0	0	0	1	0	0	0	0	0	1	1	0	1	0	1	0	0

(6) shift →

0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	1	0

Multiplicand: 35 → 0 0 1 0 0 0 1 1
Multiplier: 79 → 0 1 0 0 1 1 1 1

0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	1	0

1 1 0 1 1 1 0 1

1	1	1	0	0	1	0	1	0	0	1	1	0	1	0	1	0

(7) shift →

1	1	1	1	0	0	1	0	1	0	0	1	1	0	1	0	1

Multiplicand: 35 → 0 0 1 0 0 0 1 1

Multiplier: 79 → 0 1 0 0 1 1 1 1

1	1	1	1	0	0	1	0	1	0	0	1	1	0	1	0	1

0 0 1 0 0 0 1 1

0	0	0	1	0	1	0	1	1	0	0	1	1	0	1	0	1

(8) shift →

0	0	0	0	1	0	1	0	1	1	0	0	1	1	0	1	0

2048 + 512 + 128 + 64 + 8 + 4 + 1 = 2765