1P2H Digital Electronics 1 – Solutions

Mark Cannon Trinity Term 2026

1. Switching circuits

In circuit (a), the lamp is lit if $A = 1$ and ($B = 1$ or $C = 0$), i.e.

$$a.(b + \bar{c}) = f_a.$$

In circuit (b), the lamp is NOT lit if $A = 0$ or ($B = 0$ and $C = 1$), i.e.

$$\bar{a} + \bar{b}.c = \bar{f}_b.$$

By De Morgan's law

$$\bar{f}_b = \bar{a} + \bar{b}.c = \overline{a.(b + \bar{c})} = \bar{f}_a \quad \Longleftrightarrow \quad f_a = f_b.$$

2. Boolean algebra

(a).

a	b	c	d	f	a	b	c	d	f	a	b	c	d	f	a	b	c	d	f
0	0	0	0	1	0	1	0	0	0	1	0	0	0	1	1	1	0	0	1
0	0	0	1	0	0	1	0	1	0	1	0	0	1	0	1	1	0	1	1
0	0	1	0	1	0	1	1	0	0	1	0	1	0	1	1	1	1	0	1
0	0	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1

(b). Reduced SOP form by algebra:

$$\begin{aligned}
f &= ab(a\bar{c} + c\bar{d}) + cd + \bar{b}\bar{d} \\
&= ab(\bar{c} + c\bar{d}) + cd + \bar{b}\bar{d} && \text{(since } a.a = a\text{)} \\
&= ab(\bar{c} + \bar{d}) + cd + \bar{b}\bar{d} && \text{(race-hazard)} \\
&= ab\overline{(cd)} + cd + \bar{b}\bar{d} && \text{(De Morgan)} \\
&= ab + cd + \bar{b}\bar{d}. && \text{(race-hazard)}
\end{aligned}$$

Canonical SOP form can be obtained from the truth table by selecting entries when $f = 1$. This gives 11 minterms.

Alternatively, we have 5 terms when $f = 0$:

$$\bar{f} = \bar{a}\bar{b}\bar{c}d + \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d + \bar{a}bc\bar{d} + a\bar{b}\bar{c}d$$

Applying De Morgan's law twice gives

$$f = (a + b + c + \bar{d}).(a + \bar{b} + c + d).(a + \bar{b} + c + \bar{d})$$
$$.(a + \bar{b} + \bar{c} + d).(\bar{a} + b + c + \bar{d})$$

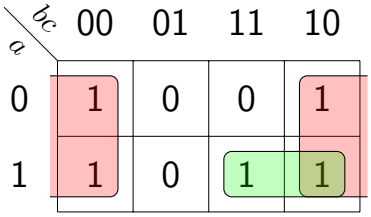
(c). Write out truth table and compare with table for f in part (a). If they are the same then equivalence proved.

3. Karnaugh maps

(a).

a	b	c	$a(b+\bar{c})$	$b\bar{c}$	$\bar{a}\bar{c}$	f_1
0	0	0	0	0	1	1
0	0	1	0	0	0	0
0	1	0	0	1	1	1
0	1	1	0	0	0	0
1	0	0	1	0	0	1
1	0	1	0	0	0	0
1	1	0	1	1	0	1
1	1	1	1	0	0	1

$\Longrightarrow$

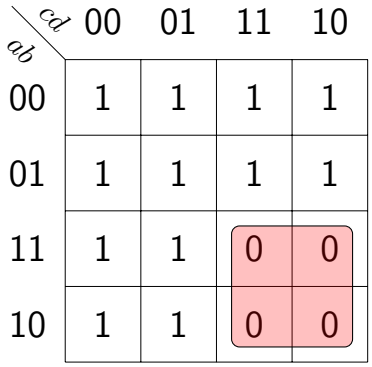


$f_1 = a.b + \bar{c}$

(b).

a	b	c	d	$\bar{a}b$	$\bar{a}\bar{b}\bar{d}$	$\bar{c}$	$\bar{a}cd$	f_2
0	0	0	0	0	1	1	0	1
0	0	0	1	0	0	1	0	1
0	0	1	0	0	1	0	0	1
0	0	1	1	0	0	0	1	1
0	1	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0	1
0	1	1	0	1	0	0	0	1
0	1	1	1	1	0	0	1	1
1	0	0	0	0	0	1	0	1
1	0	0	1	0	0	1	0	1
1	0	1	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0
1	1	0	0	0	0	1	0	1
1	1	0	1	0	0	1	0	1
1	1	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0

$\Longrightarrow$



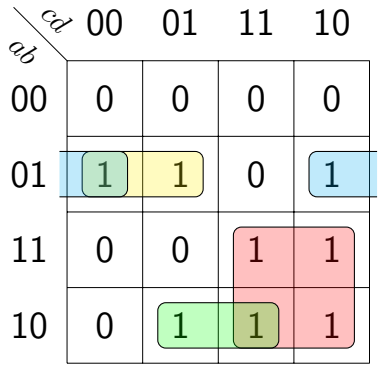
$\bar{f}_2 = a.c$ or $f_2 = \bar{a} + \bar{c}$

1P2H/3

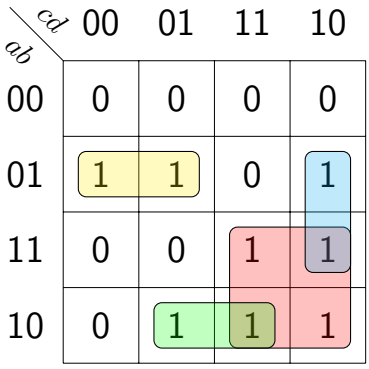
(c).

a	b	c	d	$(a \oplus b).(c \oplus d)$	abc	$\bar{a}b\bar{c}\bar{d}$	$a\bar{b}cd$	f_3
0	0	0	0	$0.0 = 0$	0	0	0	0
0	0	0	1	$0.1 = 0$	0	0	0	0
0	0	1	0	$0.1 = 0$	0	0	0	0
0	0	1	1	$0.0 = 0$	0	0	0	0
0	1	0	0	$1.0 = 0$	0	1	0	1
0	1	0	1	$1.1 = 1$	0	0	0	1
0	1	1	0	$1.1 = 1$	0	0	0	1
0	1	1	1	$1.0 = 0$	0	0	0	0
1	0	0	0	$1.0 = 0$	0	0	0	0
1	0	0	1	$1.1 = 1$	0	0	0	1
1	0	1	0	$1.1 = 1$	0	0	0	1
1	0	1	1	$1.0 = 0$	0	0	1	1
1	1	0	0	$0.0 = 0$	0	0	0	0
1	1	0	1	$0.1 = 0$	0	0	0	0
1	1	1	0	$0.1 = 0$	1	0	0	1
1	1	1	1	$0.0 = 0$	1	0	0	1

$\Longrightarrow$



OR



$$f_3 = a.c + \bar{a}.b\bar{c} + a.\bar{b}.d + \bar{a}.b.\bar{d} \qquad f_3 = a.c + \bar{a}.b\bar{c} + a.\bar{b}.d + b.c.\bar{d}$$

(d).

a	b	c	d	ab or $d\bar{c}$	$\bar{b}\bar{d}$ or $\bar{a}\bar{d}\bar{c}$	f_3
0	0	0	0	0	1	0
0	0	0	1	1	0	1
0	0	1	0	0	1	0
0	0	1	1	0	0	X
0	1	0	0	0	1	0
0	1	0	1	1	0	1
0	1	1	0	0	0	X
0	1	1	1	0	0	X
1	0	0	0	0	1	0
1	0	0	1	1	0	1
1	0	1	0	0	1	0
1	0	1	1	0	0	X
1	1	0	0	1	0	1
1	1	0	1	1	0	1
1	1	1	0	1	0	1
1	1	1	1	1	0	1

$\Longrightarrow$

ab \ cd	00	01	11	10
00	1	1	X	0
01	0	1	X	X
11	1	1	1	1
10	0	1	X	0

$$\bar{f}_4 = a.c + d$$

(e). The worst Karnaugh map is the one that doesn't allow any adjacent 1s to be amalgamated. This gives a checkerboard pattern:

ab \ cd	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

The negation of this function is just as bad. The corresponding Boolean function has the property that a change in the value of just one of the variables always changes the value of the result. The function is therefore often used as a parity check on the validity of the bits in a byte. This is the same property enjoyed by the exclusive-or function. The worst SOP Karnaugh map has the very neat expression:

$$a \oplus b \oplus c \oplus d$$

4. MOSFET operation and CMOS

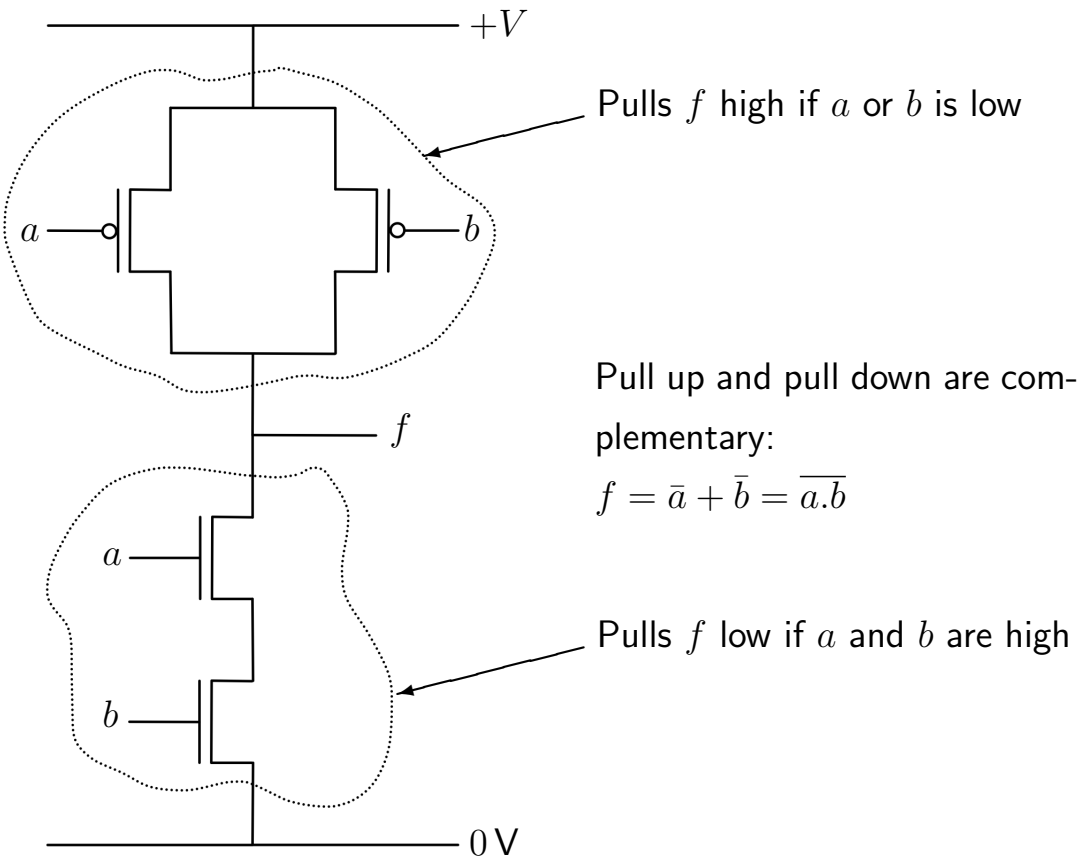
(a). Positive voltage on Gate draws electrons between Drain and Source to form a continuous n-region.

(b). The first stage on left provides $\bar{b}$.

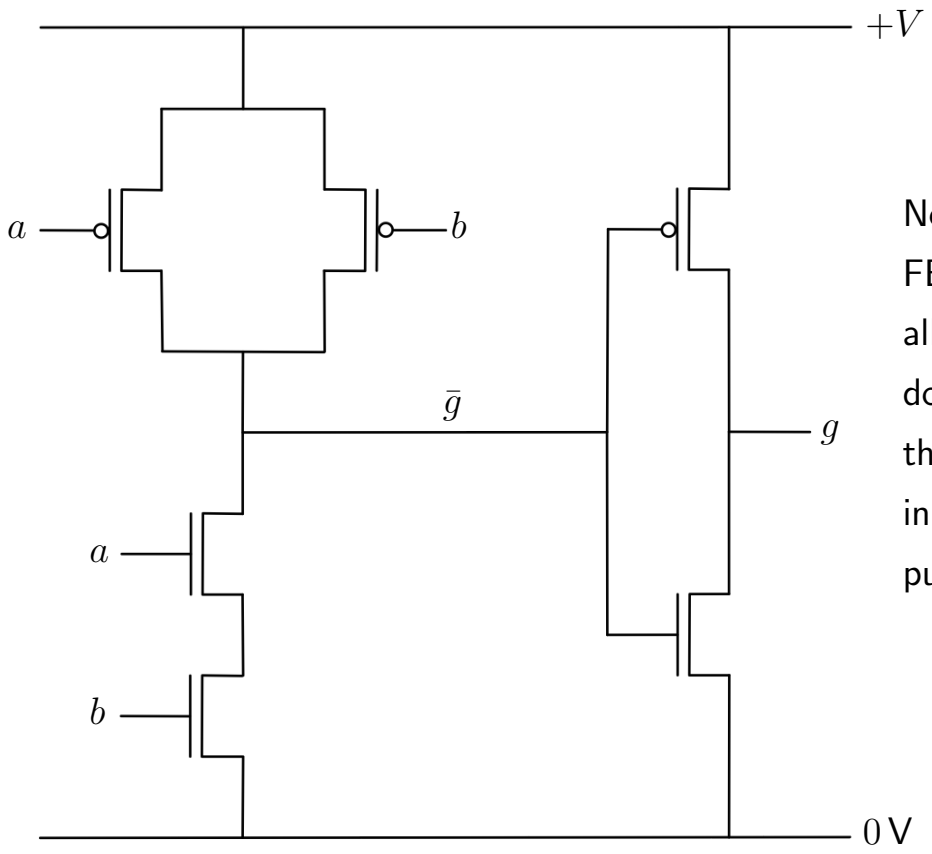
Consider n-channel devices at bottom half of second stage. The output f will be at $0\,\mathrm{V}$ if ($\bar{b} = 5\,\mathrm{V}$ or a $= 5\,\mathrm{V}$) and $c = 5\,\mathrm{V}$. So

$$\bar{f} = (a + \bar{b}).c \qquad \text{or} \qquad f = \bar{a}.b + \bar{c}$$

(c). $f = \overline{a.b}$ (NAND) – consider conditions on a and b for $f = 0$ and $f = 1$:



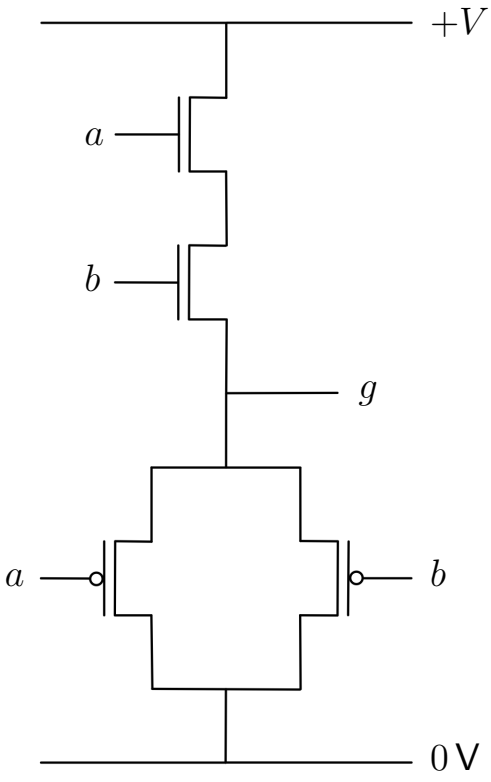
For $g = a.b$ (AND), use $\bar{g} = \overline{a.b}$, i.e. NAND + NOT:



Note that if a FET is in parallel for the pull down side, then it must be in series on the pull up side.

Note also that we CANNOT do this:

because the n-channel devices won't work for pull up and p-channel devices can't pull down.



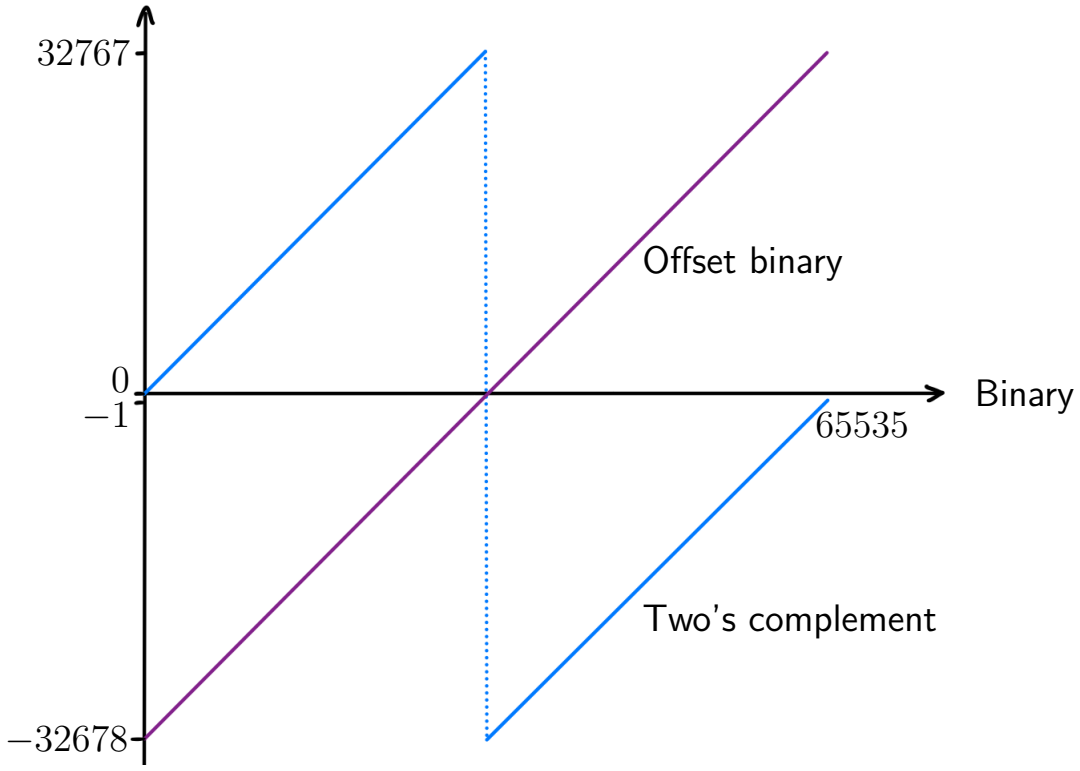
5. Binary codes

(a). Largest value $= 2^{16} - 1 = 65535$, smallest $= 0$.

(b). Offset binary: largest $= 2^{16} - 1 - 2^{15} = 32767 = \text{FFFF}_{\text{Hex}}$
smallest $= -2^{15} = -32768 = 0000_{\text{Hex}}$

(c). $-1 \rightarrow \text{FFFF}_{\text{Hex}}$

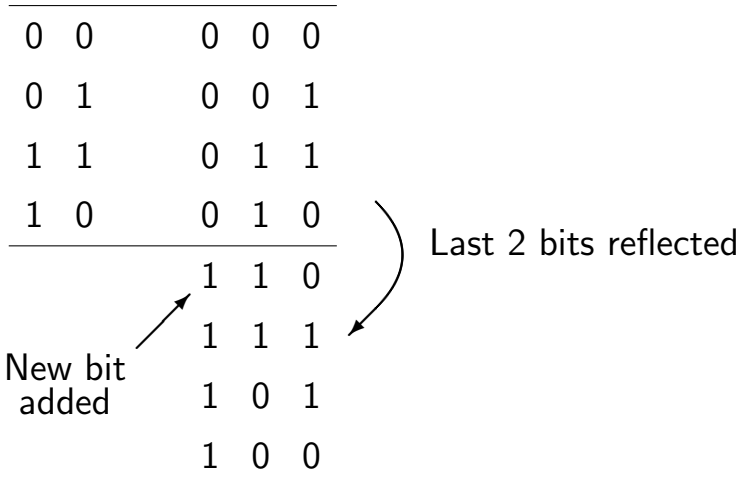
(d).



(e). BCD – Binary coded decimal, used for preserving the decimal representation of a number. Could be used, for example, to store telephone numbers in a directory.

Gray – unit distance code, only one bit changes between represented numbers. Ofte used when converting analogue to digital where bit changes arise from a physical process, i.e. an optical encoder.

(f). Gray code (reflected binary):



Application: shaft encoder

6. Two's complement

(a). $2^n - 1 = 1\,0\,0\,0\,0 \cdots 0$

$$\underline{\qquad\qquad -\ 1}$$

$$\underline{0\,1\,1\,1\,1 \cdots 1} \quad \Longrightarrow \quad 2^n - 1 \text{ has } n \text{ ones}$$

(b). $(2^8 - 1) - 01010101 \quad \Longrightarrow$

$$\begin{array}{rcc} & 1111 & 1111 \\ - & 0101 & 0101 \\ \hline & 1010 & 1010 \end{array}$$

(i) Since $2^n - 1$ has all ones, $2^n - 1 - b$ (where b is any binary number) cannot generate any carries between bit positions, so each bit subtraction is independent.

(ii) Also $1 - 0 = 1 = \bar{0}$ so $1 - b_n = \bar{b}_n$.
$1 - 1 = 0 = \bar{1}$

(i) and (ii) imply that $2^n - 1 - b = \bar{b}$

so $\underline{2^n - b = \bar{b} + 1}$

(c). Let $b = b_{n-1}b_{n-2}\ldots b_0$ be an n-bit binary number

then $2^n + b =$

$$\begin{array}{rcccc} 1 & 0 & 0 & 0 & \cdots 0 \\ +\ 0 & b_{n-1} & b_{n-2} & b_{n-3} & \cdots b_0 \\ \hline 1 & b_{n-1} & b_{n-2} & b_{n-3} & \cdots b_0 \end{array}$$

so the least significant n bits aren't affected by the addition.

(d). $2^n + (b - c) = b - c$ in bit positions $0 \to n - 1$

But $2^n + (b - c) = (2^n - c) + b$ because of (c)

$2^n - c$ creates the 2's complement of c, and hence the 2's complement of c can be used with addition to compute $b - c$.

(e). Hex $03 \to$ FD (2's complement)

So $\begin{array}{r} 4\,0 \\ -\,0\,3 \\ \hline \\ \hline \end{array}$ is equivalent to $\begin{array}{r} 4\ 0 \\ +\,\mathsf{F}\,\mathsf{D} \\ \hline 1\ 3\,\mathsf{D} \\ \hline \end{array}$ $\Longrightarrow$ $40_{\mathsf{Hex}} - 03_{\mathsf{Hex}} = 3\mathsf{D}_{\mathsf{Hex}}$

7. Adders and subtractors

(a). Carry out is generated when the addition of two n-bit numbers is too large to fit into n bits.

$C_{\mathsf{out}} = 1$ when $A_{n-1}A_{n-2}\cdots A_0 + B_{n-1}B_{n-2}\cdots B_0 \geq 2^n$.

(b). Overflow is generated when the result is of incorrect sign and is therefore relevant to signed arithmetic only.

i.e. $+$ve $+$ $+$ve $\to$ $-$ve $\Rightarrow$ overflow

$-$ve $+$ $-$ve $\to$ $+$ve $\Rightarrow$ overflow

(c). Ripple adder – it takes time for the effect of the carries to "ripple" along the full-adder circuits.

Look-ahead carry – compute the carry using faster logic, which is helpful if a carry dependent decision is required.

8. Fixed point arithmetic

(a). Smallest number $= 0000.0001 = 1/16$

(b). Largest number $= 1111.1111 = 2^4 - 1/16 = 255/16$

(c). Yes, simple binary addition.

(d). Consider the multiplication of two fixed point numbers $C = A \times B$

A has the form $A = a_i \mid a_f$, where $a_i =$ integer (4 bits)
$a_f =$ fraction (4 bits)

$$\Longrightarrow \quad A = a_i + a_f/2^4$$

$$\Longrightarrow \quad C = c_i + c_f/2^4 = a_i \times b_i + a_i \times b_f/2^4 + a_f \times b_f/2^8$$

Two potential problems: $c_i > 2^4 - 1 \;\Rightarrow$ overflow
$c_f > 1 \;\Rightarrow$ underflow

(i) OK, (ii) underflow, (iii) overflow.

Possible algorithm:

1). Treat both numbers as 8-bit integers: $A = \frac{1}{2^4}(2^4 \times a_i + a_f)$

2). Multiply the two 8-bit numbers to give

$A \times B = \frac{1}{2^8}\left[(2^4 \times a_i + a_f) \times (2^4 \times b_i + b_f)\right]$ (16-bit result)

3). Shift result right 4 places.

4). If rounding, then add the carry to the result of 3),

if truncating, discard carry.

5). If leading 8 bits are non-zero, then signal overflow,

else just discard the leading 8 bits in the 16-bit result,
and return the least significant 8 bits.

9. Floating point number representation

s	$E+63$	mantissa, f

$\implies \quad (-1)^s \times 2^E \times 1.f$

Largest positive number is

$$
\begin{aligned}
&0 \mid 111\ 1110 \mid 1111\ 1111 \\
&= 1 \times 2^{126-63} \times 1.1111\ 1111 \\
&= 1 \times 2^{63} \times \left(2 - \frac{1}{256}\right) \\
&= 1.841 \times 10^{19}
\end{aligned}
$$