Chapter 5: Number Representations and Arithmetic Circuits

- Digital circuits often need to perform arithmetic functions on numbers.
- Numbers are represented by sets of bits e.g. decimal 5 is equal to binary 101.

5.1 Positional Number Representation

- 5.1.1 Unsigned numbers
  - Decimal (base-10 or radix-10) (Afr: grondtal)
  - Binary (base-2)
  - LSB: least significant bit
  - MSB: most significant bit
  - Nibble = 4 bits
  - Byte = 8 bits
  - Word = 16 or 32 bits

5.1.2 Conversion between decimal and binary numbers

- Binary to decimal: Add each bit multiplied with its positional weight
- Decimal to binary: see figure 5.1

Convert $(857)_{10}$

			Remainder	
$857 \div 2$	=	428	1	LSB
$428 \div 2$	=	214	0	
$214 \div 2$	=	107	0	
$107 \div 2$	=	53	1	
$53 \div 2$	=	26	1	
$26 \div 2$	=	13	0	
$13 \div 2$	=	6	1	
$6 \div 2$	=	3	0	
$3 \div 2$	=	1	1	
$1 \div 2$	=	0	1	MSB

Result is $(1101011001)_2$

 Figure 5.1 Conversion from decimal to binary

5.1.3 Octal and Hexadecimal representations

- See Table 5.1

Decimal	Binary	Octal	Hexadecimal
00	00000	00	00
01	00001	01	01
02	00010	02	02
03	00011	03	03
04	00100	04	04
05	00101	05	05
06	00110	06	06
07	00111	07	07
08	01000	10	08
09	01001	11	09
10	01010	12	0A
11	01011	13	0B
12	01100	14	0C
13	01101	15	0D
14	01110	16	0E
15	01111	17	0F
16	10000	20	10
17	10001	21	11
18	10010	22	12

 Table 5.1 Numbers in different systems

5.2 Addition of Unsigned Numbers

- Half adder
- See figure 5.2

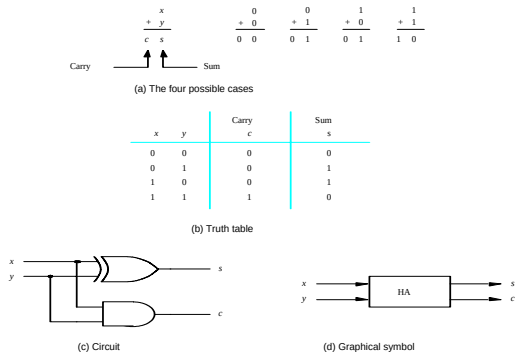


Figure5.2 Half-adder

Exclusive-OR gates

- F = x ⊕ y = x'y + xy'
- Exclusive-NOR (XNOR) (coincidence operator):
- F = (x ⊕ y)' = (x'y + xy')'
  = x (circle with dot inside) y

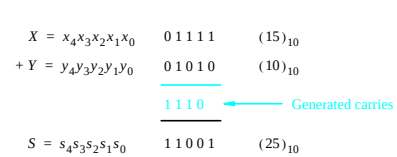


Figure 5.3 An example of addition

Full adder

- Same as half-adder but with carry-input added

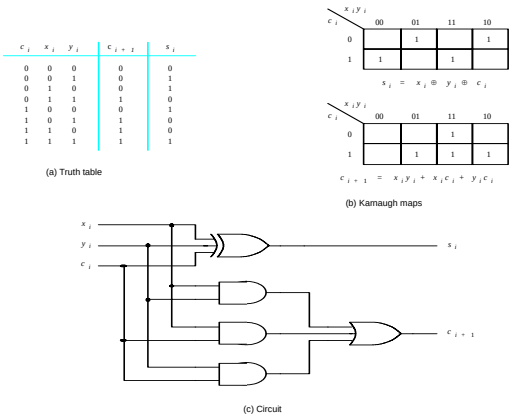


Figure5.4 Full-adder

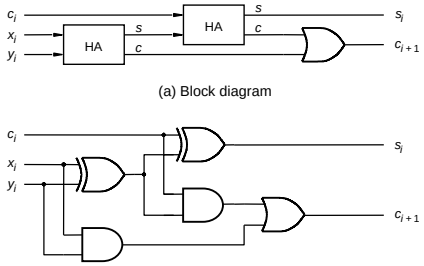


 Figure 5.5 A decomposed implementation of the full-adder circuit

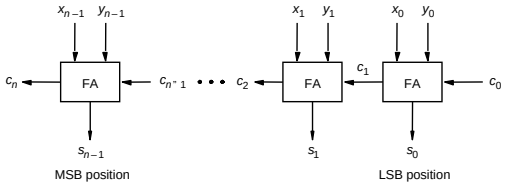


 Figure 5.6 An *n*-bit ripple-carry adder

5.3 Example

- Multiplier (see figure 5.7)

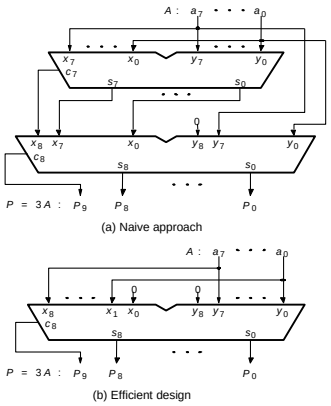


 Figure5.7 Circuit that multiplies an 8-bit unsigned number by 3

5.3 Signed Numbers

- 5.3.1 Negative numbers
  - Sign bit
  - Integer representation (see figure 5.8)

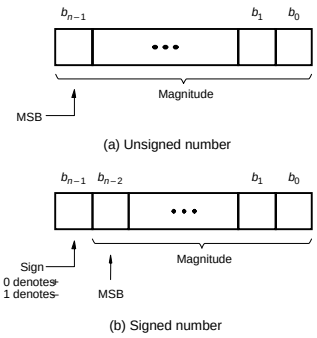


 Figure 5.8 Formats for representation of integers

Formats (see Table 5.2)

- Sign-and-magnitude
- 1's complement
- 2's complement

$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
0111	+7	+7	+7
0110	+6	+6	+6
0101	+5	+5	+5
0100	+4	+4	+4
0011	+3	+3	+3
0010	+2	+2	+2
0001	+1	+1	+1
0000	+0	+0	+0
1000	−0	−7	−8
1001	−1	−6	−7
1010	−2	−5	−6
1011	−3	−4	−5
1100	−4	−3	−4
1101	−5	−2	−3
1110	−6	−1	−2
1111	−7	−0	−1

 Table 5.2 Interpretation of four-bit signed integers

Addition

- 1's complement addition (figure 5.9)
- 2's complement addition (figure 5.10)
- 2's complement subtraction (figure 5.11)

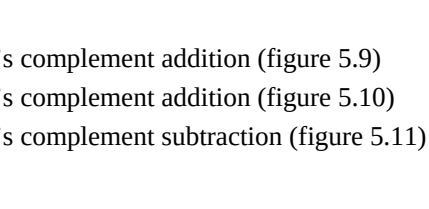


 Figure 5.9 Examples of 1's complement addition

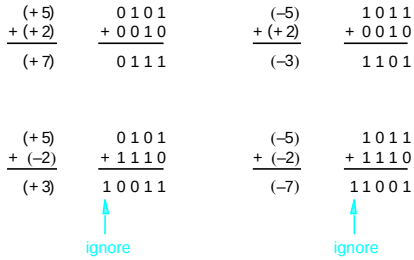


 Figure 5.10 Examples of 2's complement addition

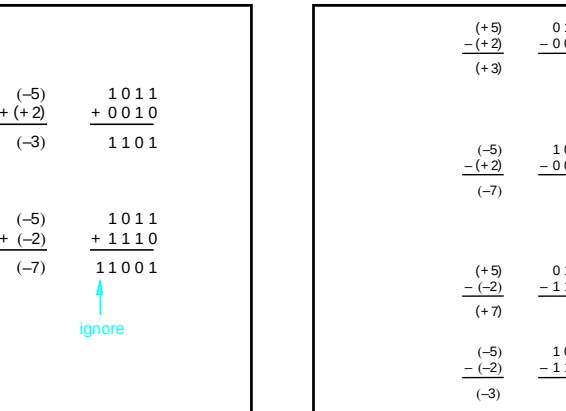


 Figure 5.11 Examples of 2's complement subtraction

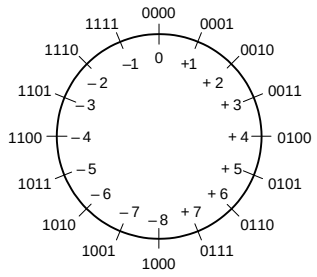


 Figure 5.12 Graphical interpretation of four-bit 2's complement numbers

5.3.3 Adder and subtractor unit

- See figure 5.13

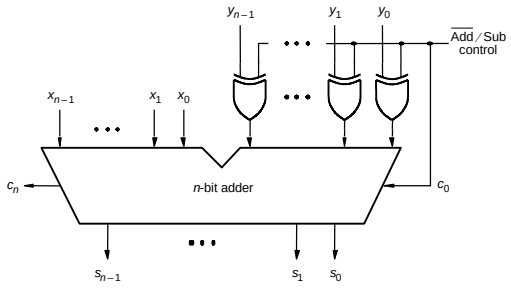


 Figure 5.13 Adder/subtractor unit

5.3.4 Radix-Complement Schemes

- Radix 2:
- 1's complement of P is K = $(2^n - 1) - P$
- 2's complement of P is K = $2^n - P$

- Also applicable to other radix values e.g. 10
  - (See 5.3.4 for examples)

5.3.5 Arithmetic overflow

- When the result of addition or subtraction exceeds the range -2^{n-1} to $2^{n-1} - 1$, then an arithmetic overflow (Afr: rekenkundige oorvloei) occurs.
- Overflow of the magnitude bits, together with the sign bit, should be taken into account.
- See figure 5.14 for more details.

(+7)	0 1 1 1	(–7)	1 0 0 1
+ (+2)	+ 0 0 1 0	+ (+2)	+ 0 0 1 0
(+9)	1 0 0 1	(–5)	1 0 1 1
	$c_4 = 0$		$c_4 = 0$
	$c_3 = 1$		$c_3 = 0$
(+7)	0 1 1 1	(–7)	1 0 0 1
+ (–2)	+ 1 1 1 0	+ (–2)	+ 1 1 1 0
(+5)	1 0 1 0 1	(–9)	1 0 1 1 1
	$c_4 = 1$		$c_4 = 1$
	$c_3 = 1$		$c_3 = 0$

 Figure 5.14 Examples of determination of overflow

5.3.6 Performance issues

- Price/performance ratio (Afr: Prys/werk-verrigting verhouding) is important in a design.
- Speed performance of a circuit is domianted by the *critical path* delay.

5.4 Fast Adders

- Main disadvantage of the ripple adder is the long delay of the carry that propagates through all the bits.
- Carry-Lookahead Adder reduces this delay.

5.4.1 Carry-Lookahead Adder

- From 5.4b:
  - $c_{i+1} = x_i y_i + x_i c_i + y_i c_i = x_i y_i + (x_i + y_i)c_i$
  - $c_{i+1} = g_i + p_i c_i$
  - Where $g_i = x_i y_i$ and $p_i = x_i + y_i$
  - g = generate, p = propagate

- Expanding these equations gives:
- $c_{i+1} = g_i + p_i(g_{i-1} + p_{i-1}c_{i-1})$
  $= g_i + p_i g_{i-1} + p_i p_{i-1} c_{i-1})$
- This expression implies a two level AND-OR circuit for a fast determination of c_{i+1}
- An adder based on this expression is called a *carry-lookahead adder*.

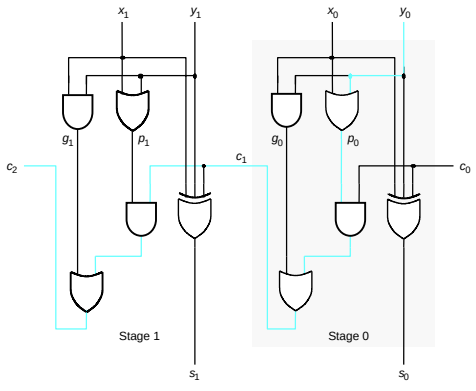


 Figure 5.15 A ripple-carry adder with generate/propagate signals

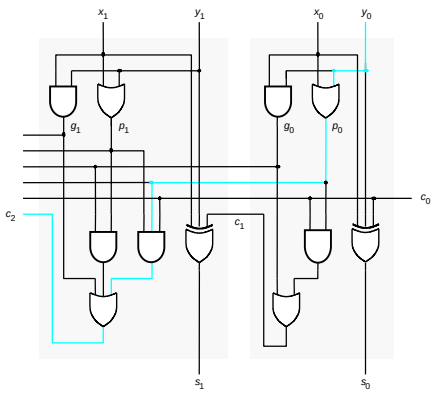


 Figure 5.16 The first two stages of a carry-lookahead adder

5.6 Multiplication

- Multiplication by 2 is the same as shifting to the left.
- Division by 2 is the same as shifting to the right.
- General Multiplication – the same as multiplication by hand (Afr: Lang vermenigvuldiging)

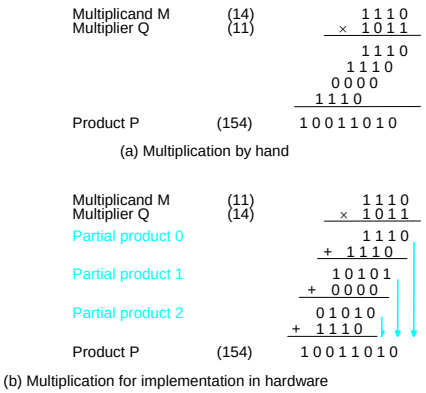


Figure5.32 Multiplication of unsigned numbers

5.6.2. Multiplication of Signed Numbers

- Sign extension
  - For positive numbers, add zeroes
  - For negative numbers, add ones

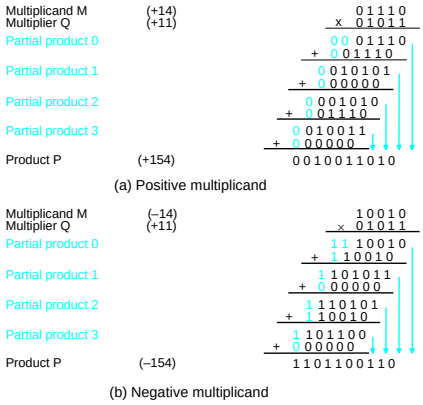


Figure5.34 Multiplication of signed numbers

5.7 Other Number Representations

- Fixed Point
  - B = $b_{n-1}b_{n-2}..b_1b_0.b_{-1}b_{-2}..b_{-k}$
  - Addition and Subtraction works as for integers.
  - Small range
- Floating Point
  - FN = Mantissa * $R^{exponent}$
  - IEEE Single and Double
  - Arithmetic operations complex

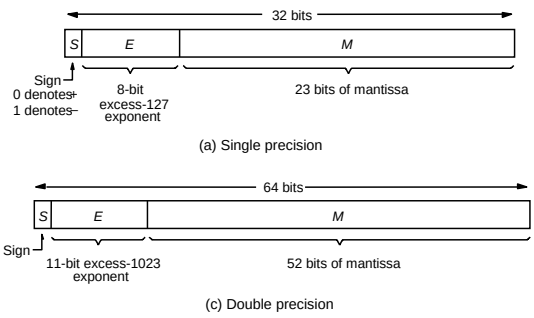


Figure 5.35 IEEE standard floating-point formats

5.7.3. Binary Coded Decimal

- Marriage between decimal and binary – Each decimal digit is represented by one nibble (4 bits).
- BCD uses 10 out of the possible 16 combinations of four bits. The remaining 6 are invalid.
- Pro – Convenient for characters display purposes
- Con – Complex circuits and under-utilised code space.

Decimal digit	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Table 5.3 Binary-coded decimal digits

BCD Addition

- Add each digit (4 bits, or nibble) as ordinary binary. Let Z = X + Y with X and Y being BCD digits.
- If Z <= 9 then no adjustment is needed.
- If Z > 9 then 6 must be added to correct the digit and a carry must be brought over to the next BCD digit.


```
X      0111    7
+ Y  + 0101  + 5
-----------------
Z      1100   12
     + 0110
     ------
carry→ 10010
       ~~~
       S = 2

X      1000    8
+ Y  + 1001  + 9
-----------------
Z      10001  17
     + 0110
     ------
carry→ 10111
       ~~~
       S = 7
```

Figure 5.36 Addition of BCD digits

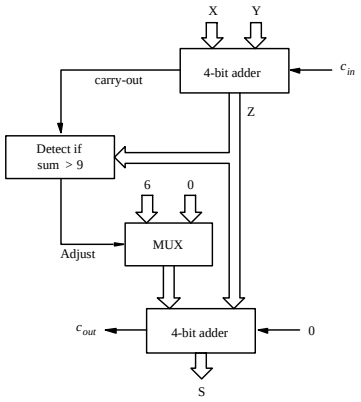


Figure 5.37 Block diagram for a one-digit BCD adder

5.8. ASCII Character Code

- American Standard Code for Information Interchange
- Allows different computer systems to exchange text documents.
- Standard specifies 7 bits – the 8th is often used for parity.

Bit positions	Bit positions 654							
3210	000	001	010	011	100	101	110	111
0000	NUL	DLE	SPACE	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

NUL	Null/Idle	SI	Shift in
SOH	Start of header	DLE	Data link escape
STX	Start of text	DC1-DC4	Device control
ETX	End of text	NAK	Negative acknowledgement
EOT	End of transmitted text	SYN	Synchronous idle
ENQ	Enquiry	ETB	End of transmitted block
ACK	Acknowledgement	CAN	Cancel (error in data)
BEL	Audible signal	EM	End of medium
BS	Back space	SUB	Special sequence
HT	Horizontal tab	ESC	Escape
LF	Line feed	FS	File separator
VT	Vertical tab	GS	Group separator
FF	Form feed	RS	Record separator
CR	Carriage return	US	Unit separator
SO	Shift out	DEL	Delete/Idle

Bit positions of node format — 6 5 4 3 2 1 0

Table 5.4 The seven-bit ASCII code

Parity

- Simple error checking scheme.
- One bit – the parity bit – is added and is set or cleared so that the total number of set bits in the byte is either even (even parity) or odd (odd parity)
- Use XOR function to create and check parity.

Chapter 6
Combinational-circuit Building Blocks

- Logic circuits are used to construct building blocks for larger designs.

6.1 Multiplexers

- A building block with a number of ***data inputs***, multiplexed to one ***data output***, under control of one or more ***select inputs***.

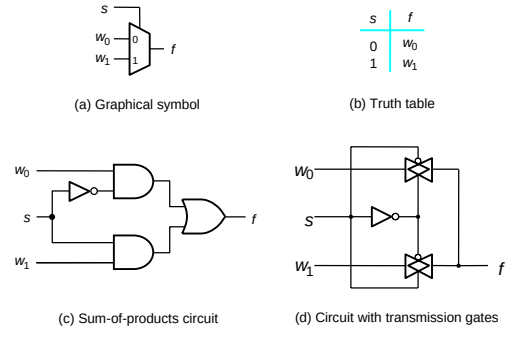


Figure 6.1 A 2-to-1 multiplexer

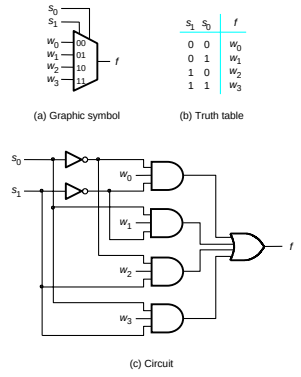
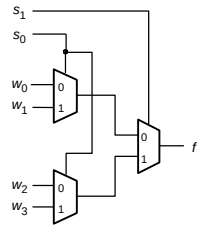
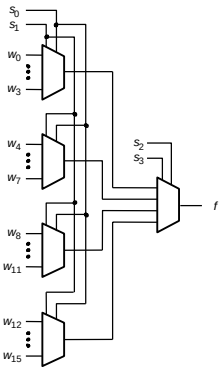
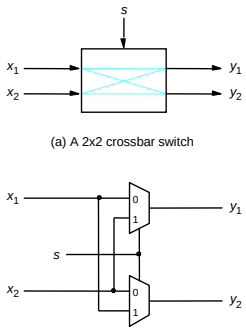


Figure 6.2 A 4-to-1 multiplexer

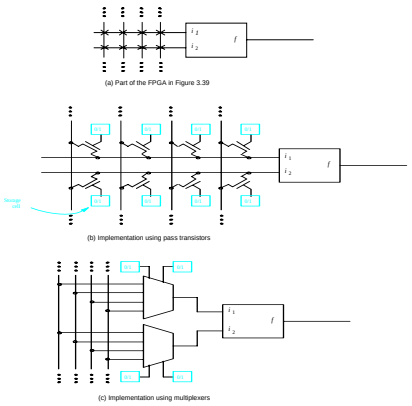
- For figure 6.2:
- $f = s_1's_0'w_0 + s_1's_0w_1 + s_1s_0'w_1 + s_1s_0w_2$
- The four combinations of s_1s_0 allow one of the four w's through to the output.





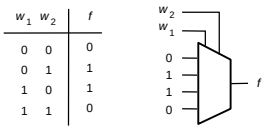


- Application of programmable switches and multiplexers in FPGA's.

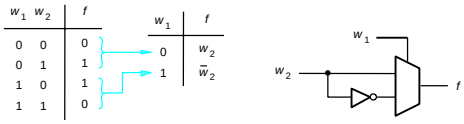


6.1.1 Synthesis of Logic Functions Using Multiplexers

- Multiplexers can be used to implement any logic function.
- Implementation not always optimal.
- Shannon's Expansion is used by some CAD tools.

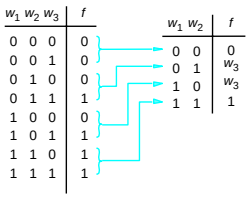


(a) Implementation using a 4-to-1 multiplexer

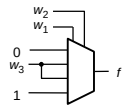


(b) Modified truth table (c) Circuit

 Figure 6.7 Synthesis of a logic function using multiplexers

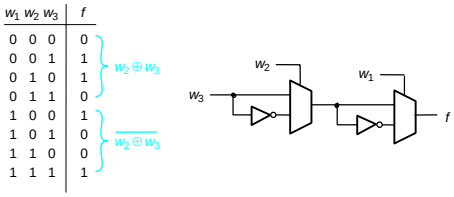


(a) Modified truth table



(b) Circuit

 Figure 6.8 Three-input majority function

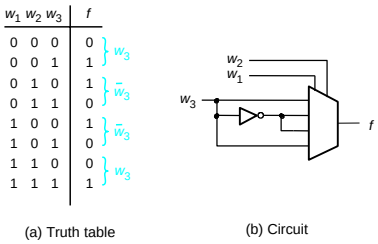


(a) Truth table (b) Circuit

 Figure 6.9 Three-input XOR function

6.1.2 Multiplexer Synthesis Using Shannon's Expansion

- Any Boolean function $f(w_1,\ldots,w_n)$ can be written in the form
- $f(w_1,w_2,\ldots,w_n) = w_1' \cdot f(0,w_2,\ldots,w_n) + w_1 \cdot f(1,w_2,\ldots,w_n)$
- $f = w_1' \cdot f_{w1'} + w_1 \cdot f_{w1}$
- where $f_{w1'}$ is called the ***cofactor*** (Afr: ko-faktor) of f with respect to w_1' and f_{w1} the cofactor of f with respect to w_1.
- See examples 6.5 and 6.6.



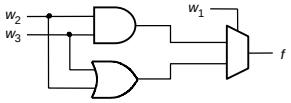
(a) Truth table (b) Circuit

 Figure 6.10 Three-input XOR function

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

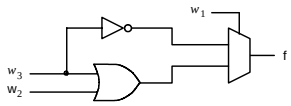
w_1	f
0	w_2w_3
1	$w_2 + w_3$

(b) Truth table

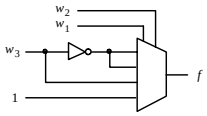


(b) Circuit

 Figure 6.11 Three-input majority function using a 2-to-1 MUX



(a) Using a 2-to-1 multiplexer



(b) Using a 4-to-1 multiplexer

 Figure 6.12 Example circuits

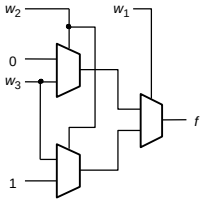
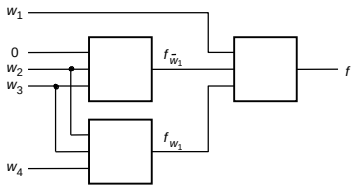
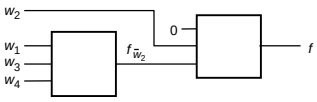


 Figure 6.13 Example circuit



(a) Using three 3-LUTs



(b) Using two 3-LUTs

 Figure 6.14 Example circuits

6.2 Decoders

- Decoders (Afr: dekodeerders) are used to decode encoded information.
- Binary decoder (Afr: binêre dekodeerder) has n inputs and 2^n outputs – see figure 6.15. Only one output is set (true, 1) at a time – ***one-hot encoded***. Example in figure 6.16.
- ***Decoder trees*** – see figures 6.17 and 6.18.

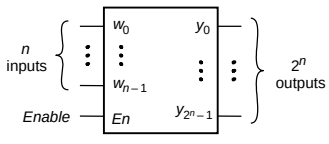
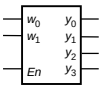


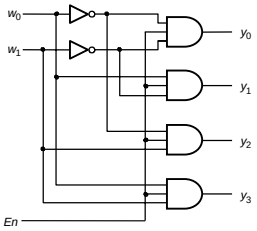
 Figure 6.15 An n-to-2^n decoder

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

(a) Truth table



(b) Graphic symbol



(c) Logic circuit

Figure 6.16 A 2-to-4 decoder

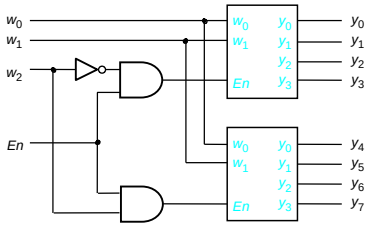


Figure 6.17 A 3-to-8 decoder using two 2-to-4 decoder

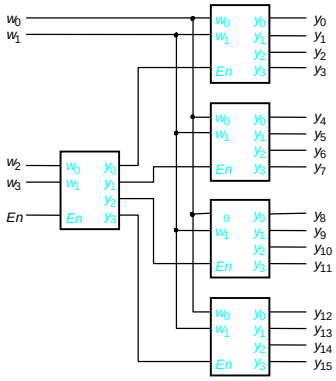


Figure 6.18 A 4-to-16 decoder built using a decoder tree

Decoder applications

- Used to select memory blocks in computer structures.
- Implement a multiplexer – see figure 6.19.

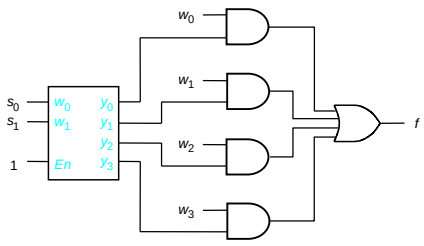
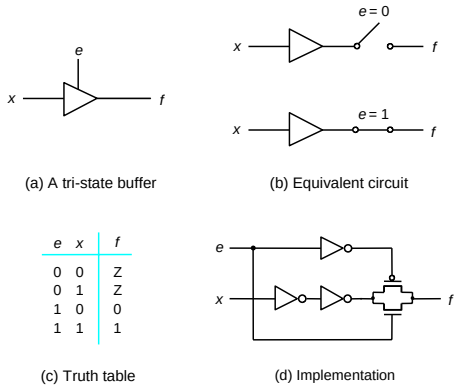
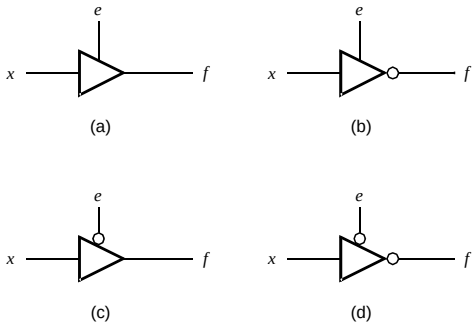


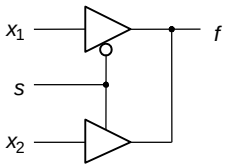
Figure 6.19 A 4-to-1 multiplexer built using a decoder

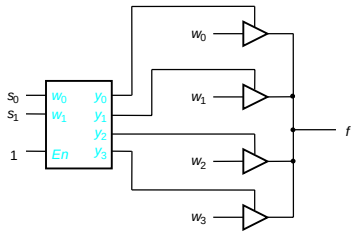
Three-state buffers (chapter 3)

- Three-state (3-state) buffers (Afr: drie-toestand-buffers)
- Output can be in one of three states i.e. 0, 1 or Z.
- Z = high impedance state (hi-Z).









6.2.1 Demultiplexers

- Opposite function of a multiplexer – n inputs and 2^n outputs.
- Transfers a source bit to one of many destinations.
- Often equivalent to a decoder – see figure 6.16.

Application of decoders

- Important application in computer memory blocks – see figure 6.21.
- Memory address, data input/output.
- ROM – read-only-memory
- PROM – programmable read-only-memory
- EPROM – erasable-programmable read-only-memory
- EEPROM – electrically-erasable-programmable read-only-memory

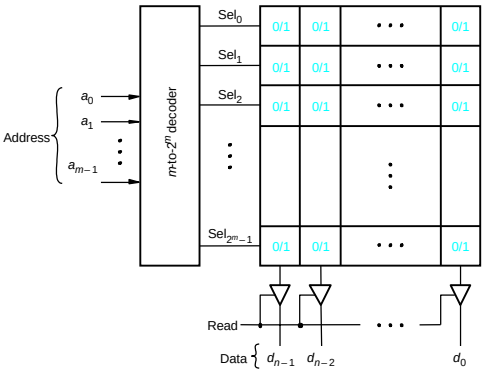


Figure 6.21 A 2^m x n read-only memory (ROM) block

6.3 Encoders

- 6.3.1 Binary encoders
  - 2^n-to-n binary encoder (figure 6.22, 6.23)
  - Used to compact data (information)
- 6.3.2 Priority encoders (figure 6.24)

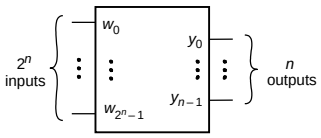
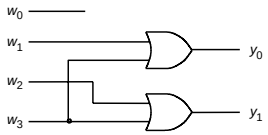


Figure 6.22 A 2^n-to-n binary encoder

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

(a) Truth table



(b) Circuit

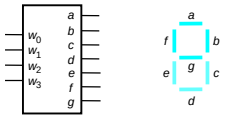
Figure 6.23 A 4-to-2 binary encoder

w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Figure 6.24 Truth table for a 4-to-2 priority encoder

6.4 Code Converters

- The preceding encoders and decoders are used often and are for specific codes.
- Any other code converter can be realized i.e. BCD-to-7-segment converter (figure 6.25)



(a) Code converter

(b) 7-segment display

w_3	w_2	w_1	w_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

(c) Truth table

Figure 6.25 A BCD-to-7-segment display code converter

6.5 Arithmetic Comparison Circuits

- Adders, subtracters and multipliers in previous chapters.
- Comparator (figure 6.26)

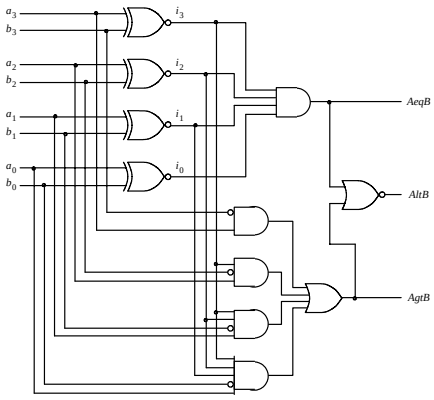


Figure 6.26 A four-bit comparator circuit