COMP2611: Computer Organization

Sequential logic and Data representation

Overview

- You will learn the following in this tutorial:
  - clock, simple memory elements (SR latch, D latch, D flip flop) and register file,
  - data representations (IEEE 754 single/double precision floating point numbers, ASCII encoding).

Sequential logic and data representation

Sequential logic

- clock
- Memory elements: SR latch, D latch, D flip flop, register file

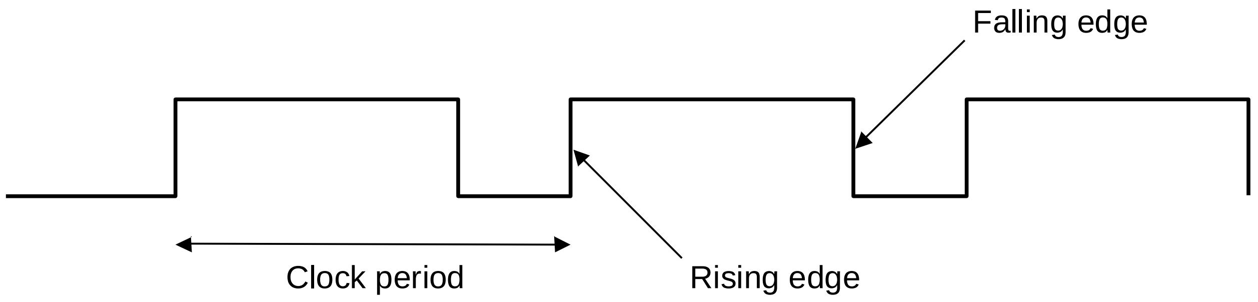
Data representation

- base conversions, two’s complement
- IEEE 754 floating point format,
- ASCII representation of characters

Exercises

Clock

- A clock acts as a global signal that gives all the components in the system an indication of time.
- Clock is used in sequential logic to decide and co-ordinate state updates.
- Just to remind you, a clock signal has three important key words that you need to know:



Sequential logic and data representation

Sequential logic

- clock
- Memory elements: S-R latch, D latch, D flip flop, register file

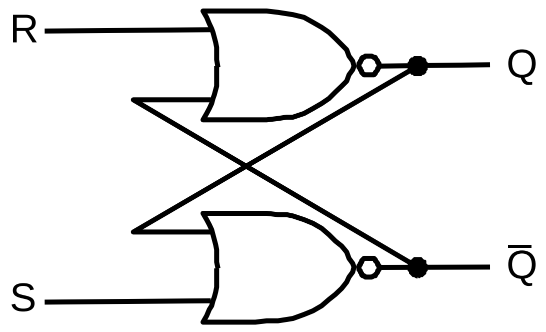
Data representation

- base conversions, two’s complement
- IEEE 754 floating point format,
- ASCII representation of characters

Exercises

The simplest memory element : S-R latch

- A S-R latch (Set-Reset latch) shown below is the simplest memory element.

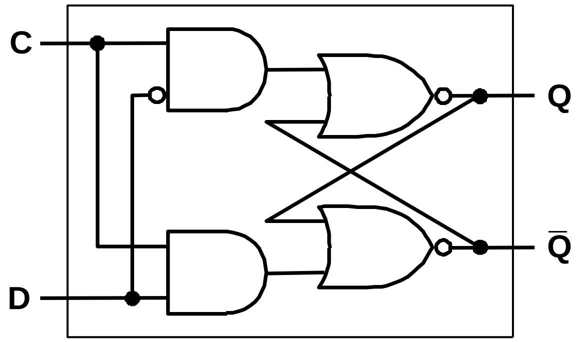


S	R	Action on Q
0	0	Nothing changed
0	1	Q=0
1	0	Q=1
1	1	forbidden

- Question: How does this circuit "stores" information? What is the size of the information being stored?

A clocked memory element – D latch

- The value stored in a D-latch can be updated iff the clock is asserted (i.e. C=1). It is a level Triggering device.

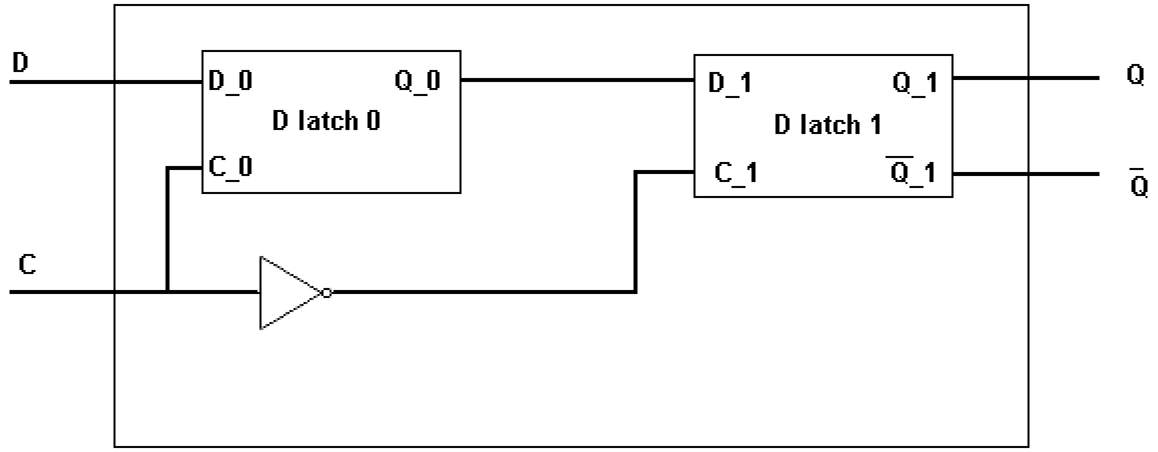


C (Clock)	D	Action on Q
0	0	Nothing changed
0	1	Nothing changed
1	0	Q=0
1	1	Q=1

- Note the S-R latch on the right part of the D-latch circuit.
- Question: From the circuit, argue whether it is possible to do update when the clock is not asserted?

D Flip-Flop

- A D flip-flop can be updated only on a falling/rising clock edge(edge Triggering device).
- There are many ways to create a D flip flop, the figure below (from the lecture note) shows a D flip flop created from two D latches.



- This flip flop can be updated in a falling edge or in a rising edge?
- Question: Without adding new hardware, how to modify the device so that it can only be updated on rising edges (if this is not already the case)?

Register file

- A register file is a piece of hardware that allows reading from and writing to the desired registers. The following figure shows a sample register file.

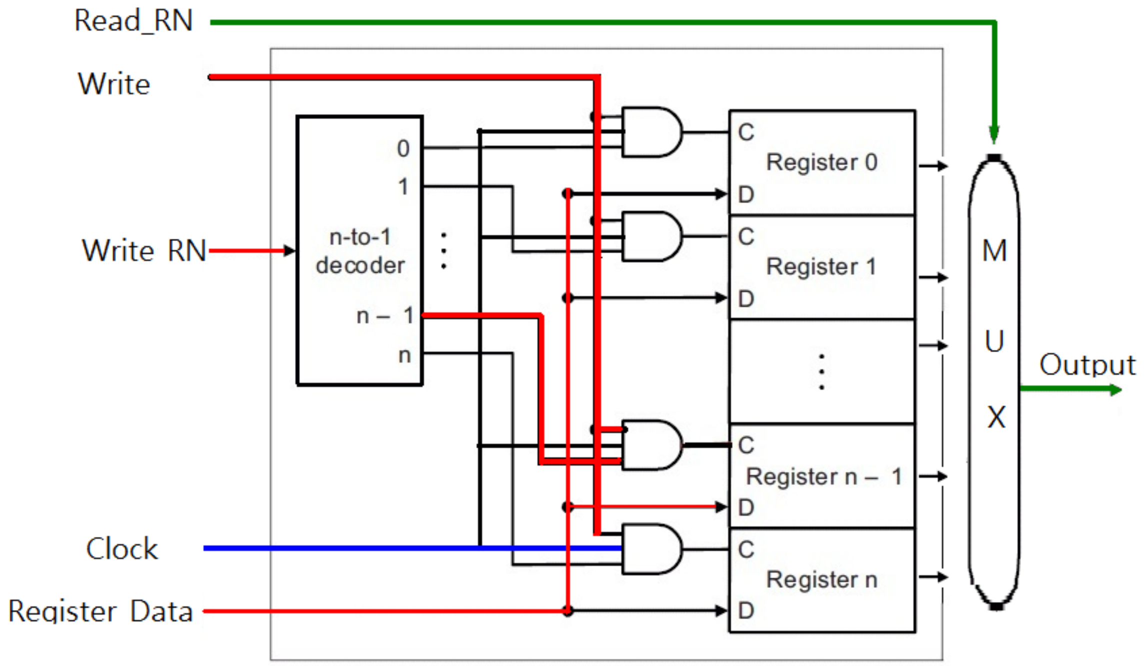
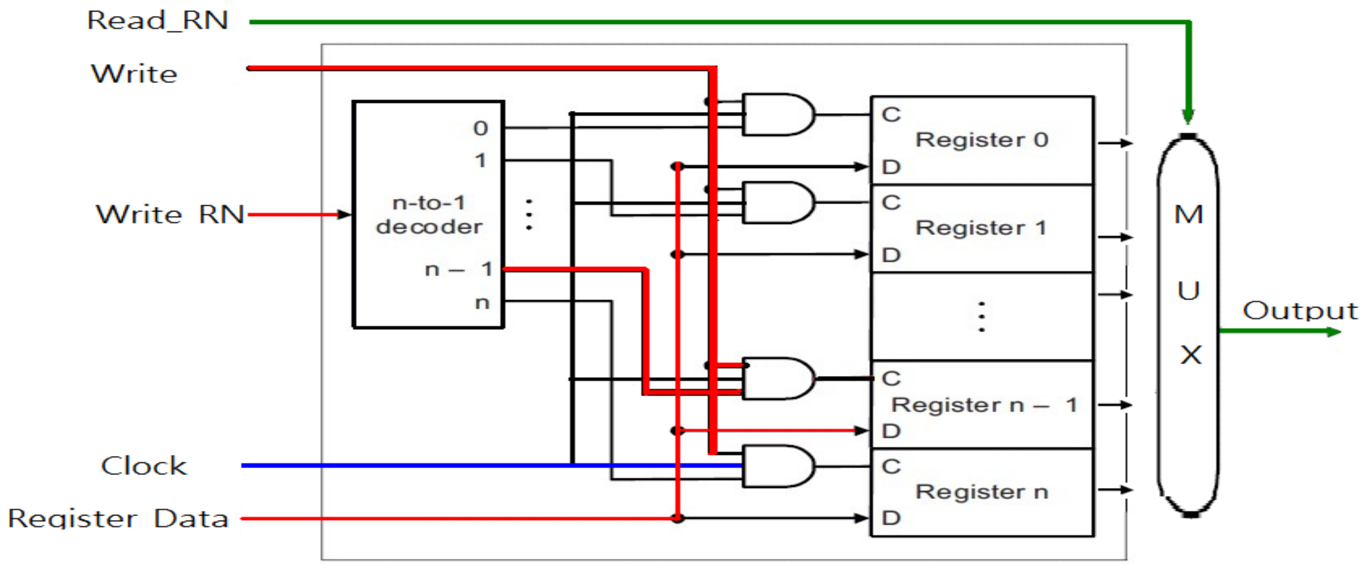


Figure 1

- How do you read from a register (what are the inputs)?
- How do you write to a register (what are the inputs)?

How to write to a register

- How do you write to a register (what are the inputs)?



Sequential logic and data representation

Sequential logic
- clock
- Memory elements: SR latch, D latch, D flip flop, register file

Data representation
- base conversions, two's complement
- IEEE 754 floating point format,
- ASCII representation of characters

Exercises

Conversion of bases

- Example : Convert $.625_{(10)}$ to the binary format:

  0.625 x 2 LHS of decimal pt=1 RHS =0.25
  0.25 x 2 LHS of decimal pt=0 RHS =0.5
  0.5 x 2 LHS of decimal pt=1 RHS = 0 done!

  $0.625_{(10)} \Rightarrow 0.101_{(2)}$

  Most significant digit

  Least significant digit

- Example : Convert $101.101_{(2)}$ to the decimal format:

  $(1 \times 2^{2}) + (0 \times 2^{1}) + (1 \times 2^{0}) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) = 5.625_{(10)}$

  $101.101_{(2)} \Rightarrow 5.625_{(10)}$

Two’s complement representation

- Computers use two’s complement representation scheme for signed numbers.
- The positive numbers are represented as before. The negative numbers are converted as follows:
  - bit inverted the number to get its one’s complement,
  - add 1 to the one’s complement number to get the two’s complement.
- Bit 31 is the sign bit. (for +ve numbers bit 31 is 0, for –ve numbers bit 31 is 1)
- Example : -5 (0000 0000 0000 0000 0000 0000 0000 $0101_{(2)}$)
  - Its 1’s complement is

    1111 1111 1111 1111 1111 1111 1111 $1010_{(2)}$
  - After adding 1 the 2’s complement becomes

    1111 1111 1111 1111 1111 1111 1111 $1011_{(2)}$

Two’s complement representation

- To convert a two’s complement number back to its decimal form, one needs to do the follows:
  - Look at the sign bit, if sign bit is zero, convert the binary number directly to decimal format,
  - If the sign bit is one, invert the number and then add 1 to the inverted number. Convert the result to decimal format and put a –ve sign to the number
- Example : Convert the two’s complement number $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011_{(2)}$ to decimal format
  - Sign bit =1, invert all the bits we have

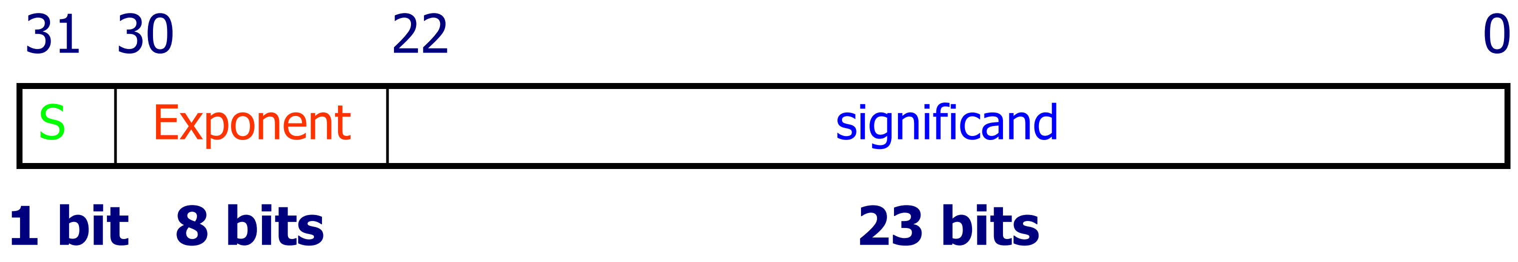
    $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0100_{(2)}$
  - Add 1 to the inverted number we have

    $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0101_{(2)}$
  - Convert the result to decimal format and add a –ve sign

    $-5_{(10)}$

The IEEE 754 single precision floating point format

- The IEEE 754 standard uses 32 bits to represent single precision floating point numbers.



- S : sign bit (0 positive, 1 negative),
- Exponent : 8-bit field, bias = 127,
- Significant : 23-bit field.

Exercise: Convert $-5.625_{(10)}$ to the single precision floating point format:

Convert to the single precision floating point format

- To convert a real number into the single precision format, three steps are involved:
  - 1. note the sign and convert the absolute value of number into binary format,
  - 2. normalize the number in 1,
  - 3. calculate the exponent value in binary format.
- Exercise:

Convert $-5.625_{(10)}$ to the single precision floating point format:

1. $5.625_{(10)} = 101.101_{(2)}$, sign bit =1
2. normalize $101.101 = 1.01101 \times 2^2$
3. exponent value = (bias +2)= (127+2) = $129_{(10)}$
   = $1000\ 0001_{(2)}$

The resulting single precision representation is

1 1000 0001 01101000000000000000000

↑ Implicit leading 1

Convert from the single precision floating point format

- To convert a number in single precision representation back to the decimal representation, one just needs to reverse the procedure:
  - 1. calculate the exponent value and convert it to decimal format,
  - 2. re-construct the value from the exponent and the significant,
  - 3. put the proper sign to the number by referring to the sign bit
- An example:

  Convert 1 1000 0001 01101000000000000000000 to the decimal representation.

  1. exponent = 1000 0001 = 129, => exponent = 129-127=2

  2. reconstruct from the exponent 1.01101 x 2^2 = $101.101_{(2)}$

     = $5.625_{(10)}$

  3. sign bit =1 => $-5.625_{(10)}$

Implicit leading 1

The IEEE 754 double precision floating point format

- The IEEE 754 standard uses 64 bits to represent double precision floating point numbers.

63	62	51	0
S	Exponent	significand	
1 bit	**11 bits**	**52 bits**	

- S : sign bit (0 positive, 1 negative),
- Exponent : 11-bit field, bias = 1023,
- Significant : 52-bit field.

Exercise: Convert -5.625$_{(10)}$ to the double precision floating point format:

Convert to the double precision floating point format

- The steps for converting a real number into the double precision format are identical to those of the single precision, except for two things :
  - 1. the exponent is 11-bit and the bias is now 1023 (instead of 127),
  - 2. the significant contains 52 bits (instead of 23 bits).
- Exercise :

  Convert $-5.625_{(10)}$ to the double precision floating point format:

  1. $5.625_{(10)} = 101.101_{(2)}$, sign bit =1
  2. normalize $101.101 = 1.01101 \times 2^{2}$
  3. exponent value = (bias+2)= (1023+2) = $1025_{(10)}$
     = $100\ 0000\ 0001_{(2)}$

  The resulting single precision representation is

  1 100 0000 0001 0110100000000000000000…0000

  1bit 11 bits 52 bits

  Implicit leading 1

Convert from the double precision floating point format

- To convert a number in double precision representation back to the decimal representation, follows a reverse procedure as below:
  - 1. calculate the exponent value and convert it to decimal format,
  - 2. re-construct the value from the exponent and the significant by adding the implicit leading 1,
  - 3. put the proper sign to the number by referring to the sign bit
- An example:

  Convert 1 100 0000 0001 01101000000000000000000…0000 to the decimal representation.

  1. exponent = 1000 0000 0001 = $1025_{(10)}$ => exponent = 1025-1023=2

  2. reconstruct from the exponent $1.01101 \times 2^2 = 101.101_{(2)}$

  $= 5.625_{(10)}$

  3. sign bit =1 => $-5.625_{(10)}$

IEEE 754 special cases

- IEEE 754 Single precision format:

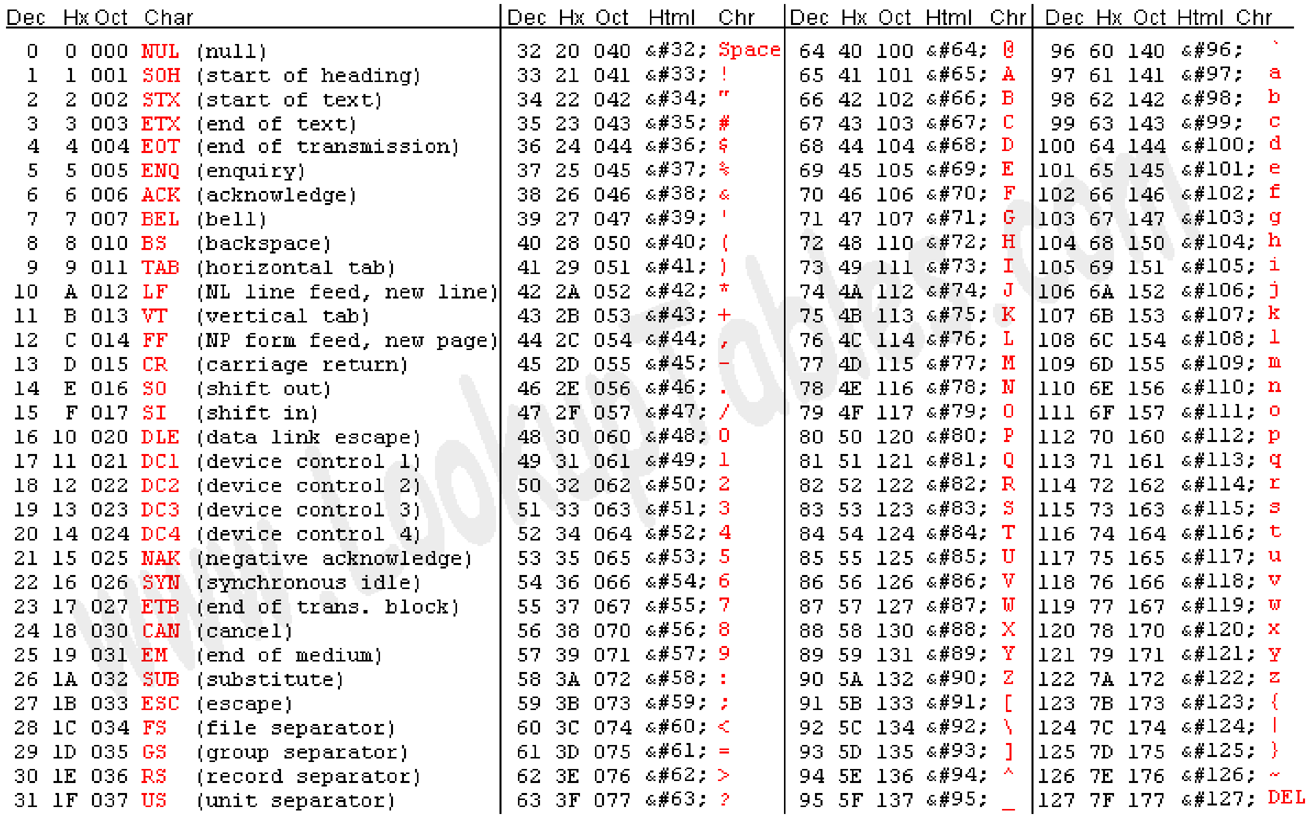
Exponent / Significand	0	**1 - 254**	255
0	0	$(-1)^S \times (1.F) \times (2)^{E-127}$	$(-1)^S \times (\infty)$
$\neq$ 0	$(-1)^S \times (0.F) \times (2)^{-126}$		non-numbers e.g. 0/0 , $\sqrt{-1}$

- IEEE 754 Double precision format:

Exponent / Significand	0	**1 - 2046**	2047
0	0	$(-1)^S \times (1.F) \times (2)^{E-1023}$	$(-1)^S \times (\infty)$
$\neq$ 0	$(-1)^S \times (0.F) \times (2)^{-1022}$		non-numbers e.g. 0/0 , $\sqrt{-1}$

Representing text– the ASCII codes

- The American Standard Code for Information Interchange (ASCII) is a character encoding scheme for encoding text.
- ASCII code uses 8 bits to represent one character.
- The list of the first 128 characters is listed in the table below



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Sequential logic and data representation

Sequential logic
- clock
- Memory elements: SR latch, D latch, D flip flop, register file

Data representation
- base conversions, two's complement
- IEEE 754 floating point format,
- ASCII representation of characters

Exercises

Exercises

Question 1: Given the bit pattern 1000 0000 0100 0110 0000 0000 0000 0000

- What is the value if this is a 2’s complement representation?
- What if the pattern is an unsigned integer?
- What if it is an IEEE single precision number?
- What if it represents 4 ASCII characters (assume bits 31-24, 23-16, 15-8, 7-0 store the characters, and ASCII value of 128 is the symbol ‘€’).

Exercises

Question 2: Assume the bit pattern 1001 1100 follows the IEEE-like floating point representation format

S	Exponent	significand
1 bit	**3 bits**	**4 bits**

- What is the bias of the exponent?
- What value the given pattern is representing?

Exercises

- Question 3: Convert the decimal real numbers given below into their Binary fractions in IEEE 754 single precisions, using either rounding to nearest (ties to even) or truncating when overflow occurs:
  a) 5.5
  b) 8.25
  c) 9.6

Conclusion

- Today we have reviewed:
  - clock, simple memory elements (S-R latch, D latch, D flip-flop) and register file,
  - simple base conversions, the IEEE 754 floating point format, and the ASCII character scheme.