ICM – Computer Science Major and M1 Cyber Physical and Social Systems
Data Interoperability and Semantics

Exam "PT1H30M"^^xsd:duration

Some mistakes that should not have been made:

* One byte is two hex digits. not one!
* Two hex digits is 8 bits. not one!
* One byte is 8 bits. not 4, not 5!

Some reminders I should have to make:

* "byte"@en = "octet"@en = octet"@fr . "bit"@en = "bit"@fr
* Take the time to read the exam and the appendices. They're here for a purpose !
* You can use this ASCII table for direct dec-hex conversion up to 127 !

Some strange ideas you had:

* I would never ask you to convert a long string into ASCII. This is a dumb exercise that doesn't solicit your neurons. I expected you to convert only one letter: 'W'.

Part 1 (5 points) Comparison of main data formats (target: 20min for this exercise)

On one large full-page table, shortly describe or compare each data formats from List 1 in terms of the features and criteria from List 2.

If I ask a large table: give me a large table. Not a paragraph for each data format.

Feature/Criteria	CSV	JSON	XML	YAML
Simplicity	++	+	-	+
Human-Readability	+	+	-	++
Interoperability	+ (different separators, charset, etc.)	++ (largely supported)	++ (same)	- (recent)
File Size (++ is lightweight)	-- (duplicate values)	+	-	++
Data Type Support	-- (just text)	+ (number string boolean null)	++ (XML Schema datatypes)	+ (more than in JSON)
Metadata Support	-- (apart from column names if header row)	- (as additional key-value pairs)	+ (XML header, comments, element attributes,..)	+ (same as JSON + comments, explicit datatypes, …)
Structural Complexity	--	+	++	+ (same as JSON + anchors/references)
Query	--	+ (JSONQuery)	++ (XQuery)	+ YAML is superset of JSON. JSONQuery can run on “some” YAML documents)
Schema Validation	--	+ (JSON Schema)	++ (DTD, XMLSchema, …)	+ similar to above
Extensibility	--	+ (not really. unless accepted by the schema)	++ (the X in XML stands for “Extensible”. XML	+ (same as JSON)

Feature/Criteria	CSV	JSON	XML	YAML
			namespaces are for that)	

Part 2 (/20 points) ASN.1

With most important parts highlighted in yellow

ASN.1 is a standard *interface description language* for defining data structures that can be serialized and deserialized in a cross-platform way. It provides a formal way to describe data and enables interoperability across different systems by ensuring consistent data encoding and decoding. While ASN.1 is now 40 years old, it is still broadly used in telecommunication and computer networking, such as for 5G mobile phone communications, LDAP directories, Securing HTTP communications with TLS (X.509) Certificates, Intelligent Transport Systems, and the Interledger Protocol for digital payments.

Protocol developers define data structures in ASN.1 *modules*, which are generally a section of a broader standards document written in the ASN.1 language. Here are some common ASN.1 base data types.

- BOOLEAN [tag number 01_{10}]: value can be TRUE or FALSE
- INTEGER [tag: 02_{10}]: a signed integer. A valid range can be specified with the notation (min..max)
- BIT STRING [tag number 03_{10}]: used for bit arrays, where each bit has an individual meaning.
- ENUMERATED [tag number 10_{10}]: a list of named items.
- SEQUENCE [tag number 16_{10}]: a collection of items to group together.
- CHOICE [n/a]: one of the items can be present at a time.
- IA5String [tag number 22_{10}]: a printable ASCII string.

Below is an ASN.1 module definition, adapted from the ETSI Intelligent Transport Systems (ITS) Common Data Dictionary definition https://forge.etsi.org/rep/ITS/asn1/cdd_ts102894_2.

*Note: "**ego**" is how we name the vehicle on which the communicating ITS system is deployed. "**alter**" is another vehicle that is observed by "ego", or that communicates with "ego".*

```
ETSI-ITS-CDD DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
EgoData ::= SEQUENCE { -- invented for the exam -
    id                IA5String,
    energyStorage     EnergyStorageType,
    speed             SpeedValue,
    driveDirection    DriveDirection,
    lanePosition      LanePositionOptions
}

AlterData ::= SEQUENCE { -- invented --
    id                IA5String OPTIONAL,
    message           IA5String OPTIONAL,
    safeDistance      SafeDistanceIndicator,
    speed             SpeedValue,
    driveDirection    DriveDirection,
    lanePosition      LanePositionOptions
}

EnergyStorageType ::= BIT STRING { -- real def --
    hydrogenStorage       (0),
    electricEnergyStorage (1),
    liquidPropaneGas      (2),
    compressedNaturalGas  (3),
    diesel                (4),
    gasoline              (5),
    ammonia               (6)
}(SIZE(7))

SafeDistanceIndicator::= BOOLEAN -- real def --

SpeedValue ::= INTEGER { -- unit is 0,01 m/s --
    standstill  (0),
    outOfRange  (16382),
    unavailable (16383)
} (0..16383)


DriveDirection ::= ENUMERATED { -- real def --
    forward     (0),
    backward    (1),
    unavailable (2)
}

LanePositionOptions ::= CHOICE { -- real def --
    simplelanePosition      LanePosition,
    simpleLaneType          LaneType,
    detailedlanePosition    LanePositionAndType,
    ...
}

LanePositionAndType::= SEQUENCE { -- real def --
    transversalPosition LanePosition,
    laneType            LaneType DEFAULT traffic,
    ...
}

LanePosition ::= INTEGER { -- real def --
    offTheRoad          (-1),
    innerHardShoulder   (0),
    outerHardShoulder   (14)
} (-1..14)

LaneType::= INTEGER{ -- simplified: only some
values --
       traffic              (0),
```

```
        pedestrian          (12),                   }(0..31)
        parking             (17),                   END
        emergency           (18)
        ...
```

Because ASN.1 is both human-readable and machine-readable, an ASN.1 compiler can compile modules into libraries of code, *codecs*, that decode or encode the data structures.

ASN.1 defines different *encoding rules* that specify how to represent a data structure as bytes. Basic Encoding Rules (BER) is the oldest one, Packed Encoding Rules (PER) is the most compact. XML Encoding Rule (XER) is based on XML. JSON encoding rules (JER) is the easiest to start playing with ASN.1 and to debug applications.

In this part we will consider two messages, one about "ego", one about "alter":
Message 1: *"Ego is a Highway Grass Cutting Machine with id AB-123-CD. It uses and stores diesel, liquid propane gas, and electricity. It drives forward on the outer hard shoulder at a speed of 10.8 km/h."*
Message 2: *"Alter EF-456-GH is moving forward at 144 km/h on traffic lane number 3, not respecting safe distances"*

Note: Check out the appendices A-E, as they are all important to answer the questions in this part.
That sentence was really important. In an exam like this, you should first have a quick look at the appendices and identify what will be important.

Question 1. (1 pt) Justify that the encoded value for speed 10.8 km/h is integer 300.
Text useful in yellow in definition of `SpeedValue` : value is encoded in 0.01 m/s.
You should know, or be able to find that, 1m/s = 3.6 km/h.

10.8 km/h = 3 m/s = 300 x 0.01 m/s

Question 2. (2 pts)
How IEEE 754 floating point number would encode the value 144.0 ?
You can use this ASCII table for direct dec-hex conversion up to 127 !

$144_{10} = 128 + 16 = 1001\ 0000_{\ 2} = 2^7 \times 1.001 = (-1)^0 \times 2^{(134-127)} \times 1.001$
S = 0, E = $134_{10} = 128 + 6 = 1000\ 0110_{\ 2}$. , M=00100….
Thus 144_{10} = 0 1000 0110 00100…$_{IEEE754}$
= 0100 0011 0001 00…$_{IEEE754}$
= **43 10 00 00** $_{16}$

Find an IEEE 754 floating point number that approximates 10.8 at ±0.05

10.8 ±0.05 = $2^{-1} \times 21.6$ ±0.1 = $2^{-2} \times 43.2$ ±0.2 . Thus we'll encode $2^{-2} \times 43$ (= 10.75)
$2^{-2} \times 43_{10} = 2^{-2} \times 2B_{16}$ (ascii table) = $2^{-2} \times 0010\ 1011 = 2^{-2} \times 2^5 \times 1.01011 = 2^3 \times 1.01011 = (-1)^0 \times 2^{(130-127)} \times 1.01011$
S = 0, E = 130 = 128 + 2 = 1000 0010 $_2$ M = 0101100….
Thus 10.75_{10} = 0 1000 0010 0101100…
= 0100 0001 0010 1100 …
= $412C0000_{16}$

Question 3. (4 pts) Write a document that could be a plausible JER encoding of Message 1 about "ego"

official JER encoding:

```
{
   "id": "AB-123-CD",
   "energyStorage": "58",   # here 68₁₆ is   0110100(0),  where the ones refer to elements 1, 2, 4 of the bit string
   "speed": 300,   # encoding of 10.8 km/h in 0.01 m/s
   "driveDirection": 0,  # alias for "forward"
   "lanePosition": {
      "simplelanePosition": 14 # alias for "outerHardShoulder"
   }
}
```

I would give all points to:

```
{
   "id": "AB-123-CD",
   "energyStorage": ["electricEnergyStorage", "liquidPropaneGas", "diesel" ],
   "speed": 300,
   "driveDirection": "forward",
   "lanePosition": 14
}
```

* different names for keys are fine, as long as I -human- can understand them and it remains plausible.
* alternative valid values:
   * for `"energyStorage"`: `[1, 2, 4]`, `[false, true, true, false, true, false, false ]` ...
   * for `"speed"`: `"10.4 km/h"`, `"10,4 km/h"`, `"300"`, ...
   * for `"driveDirection"`: `0`
   * for `"lanePosition"`: `"outer hard shoulder"`

* I expect a valid JSON document !
* DO NOT embed everything as the value of an object with a unique key such as `{ “EgoData” : {…..} }`
* DO NOT embed each key-value pair in a dedicated object: { {“id”: "AB-123-CD" } , {"speed": 300}, … }
* DO NOT add unneeded information such as `"type": "EgoData"` , or `"name": "Grass Cutting Machine"`, that's not part of the data definition.
* DO NOT invent "possible" or "used" carburant keys. That's not part of the data definition.
* Highway is not the brand of the machine. We’re talking about a machine like in this picture:

Question 4. (1 pt) BER-encode the BIT STRING. `diesel+liquidPropaneGas+electricEnergyStorage`

For this question you should read Appendix C and D. A bit string is like its name says it is. It’s an array of bits, where each bit has a meaning. If bit zero is set to 1, then it means the vehicle stores hydrogen. Etc.
I would never ask you to convert this long string to ASCII! This would be a dumb exercise that doesn't solicit your neurons. Some of you wasted their time.

We, sender, choose to encode it as *primitive*.
Identifier is just the tag value: 00 for the class, 0 for the P/C bit, then 00011 for the Tag value part → identifier is 03_{16}

Content: we have seven bits to encode. bits number 1, 2, 4, are set to one. the others are set to zero.
We will need 1 bit for padding to round up to a multiple of 8 bits (one byte).
The initial octet of the content will thus be 01_{16} (the number of padding bits)
Then spec says that the bits in the bitstring value are commencing with bit 8. So the only subsequent octet will be:
0110100 + 0 (padding) = 68_{16}
Finally, we know that the content length is of two bytes, so Length octet will be 02_{16}
In the end, the answer is **03 02 01 68_{16}**

Question 5 (1 pt) I injected two syntax errors in this document. For each error, give the line number and explain it.

The XML document below is the XER encoding of Message 2 about "alter":

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <AlterData>
3      <id>EF-456-GH</id>
4      <safeDistance>
5          <false /> # as strange as it looks, this is the actual XER encoding (test with py asn1tools)
6      </safeDistance>
7      <speed>4000<speed>  Line 7: tag speed is not closed.
8      <driveDirection>
9          <forward /> # perfectly fine: tag with no content, ends with  />
10     </driveDirection>
11     <lanePosition>
12         <detailedlanePosition>
13             <transversalPosition>3</transversalPotion> Line 13: typo in the closing tag
14             <laneType>traffic</laneType>
15         </detailedlanePosition>
16     </lanePosition>
17 </AlterData>
```

Question 6. (1 pt) What would be the BER encoding of
(a) positive integer 4000 ?
I know $1024 = 2^{10}$, so $4000<2^{12}$. I should be fine with only 2 bytes! and I know that it will look like 0000 1xxx xxxx xxxx
$4000 = 2000 \times 2 = 1000 \times 2^2 = 500 \times 2^3 = 250 \times 2^4 = 125 \times 2^5$.
stop here: I know from the ASCII Table that $125_{10} = 7D_{16} = 111\ 1101_{2}$
Multiply that by 2^5 is like **left-shifting** five times: 111 1101 **00000**
I rearrange bits and add zeros in most significant bit positions: $0000\ 1111\ 1010\ 0000_{2}$ = **$0F\ A0_{16}$**
as we want the BER encoding, we prepend this content with tag (02) and length (02).
Thus the final response is **$02\ 02\ 0F\ A0_{16}$**

Brute force approach to compute content is:
4000 – (reminder **0**) 2000 – (**0**) 1000 – (**0**) 500 – (**0**) 250 – (**0**) 125 – (**1**) 62 – (**0**) 31 – (**1**) 15 – (**1**) 7 – (**1**) 3 – (**1**) **1**
and I read in reverse order the digits that are underlined: $1111\ 1010\ 0000_2 = FA0_{16}$ That's two bytes: 0F A0. Not three!

(b) negative integer -120 ?
we note that -128 <= -120 <= 127, so we're fine with one byte! and I know that it will look like 1xxx xxxx (negative)
Different approaches here.
I may know that 1000 0000 = -128, and I just need to add eight to that: $1000\ 1000 = 88_{16}$
Brute force approach is:
1. encode 120 (still, I'm working smart, I'm using the ASCII table) $78_{16} = 0111\ 1000_{16}$
2. NOT that: 1000 0111
3. add one: 1000 1000 = **88_{16}**

as we want the BER encoding, we prepend this content with tag (02) and length (01).
Thus the final response is **$02\ 01\ 88_{16}$**

Question 7. (1 pt) Justify that the maximal encodable length in BER is 2^{1008}
one crucial information was lacking in the appendix here, my bad: In the Long form, bits cannot all be set to one.
If Length octet is FF, then we're using another type of encoding: "indefinite length". For example using indefinite length, a IA5String would end only when encountering two null bytes: 0x00 0x00.
On the seven bits we have to encode the length, we can encode up to integer 127. Minus the special case FF, which stands for "indefinite length", maximal length is then encoded on 126 octets. So $2^{(8x126)} = 2^{1008}$
I awarded points to all students that noticed 1008=8*126 (they were on the right track)

Question 8. (1 pt) Assume we want to set the `message` IA5String in Message 2 about "alter" as follows:

"Warning: Automated grass cutter ahead. Maintain safe distance. Speed reduced. Hazardous debris possible. Stay alert for sudden stops and avoid lane changes near the vehicle. Thank you for your cooperation."

This message has a total of 205 characters. Give the most significant four bytes of that message encoded using BER

I would never ask you to convert this long string to ASCII! This would be a dumb exercise that doesn't solicit your neurons. Some of you wasted their time.

IA5String is primitive. So identifier is 00 for the class, 0 for the P/C bit, then 22_{10} for the Tag part. So identifier is 16_{16}

Content will of length 205 (above 127), and will start with ASCII character 57_{16}.

We need long form length encoding, for length 205, which can be encoded on one byte.

So length of length octet is 1000 0001 = 81_{16}

and encoded length is 205 = 128+77 = 80_{16} + $4D_{16}$ (being smart and using the ASCII table) = CD_{16}

Final answer is: **16 81 CD 57** $_{16}$

Question 9. (1 pt) Justify that the identifier octet for a SEQUENCE needs to be 30_{16}

Appendix E, section 8.9.1, says it's constructed. So identifier octet is: 00 for the class, 1 for P/C, then $16_{10}=10_{16}$

Final answer is: 0011 0000 = **30_{16}**

Question 10. (4 pts) Determine the BER-encoding for Message 2 about "alter". Justify step by step.

I want the same data as in the XML document above.

We use the same structure as in the example of Appendix E, and we compute the length at the end.

Sequence	Length	Contents
30_{16}	??	
	id	
	IA5String	Length Contents
	16_{16}	09_{16} "EF-456-GH"
	safeDistance	
	Boolean	Length Contents
	01_{16}	01_{16} 00_{16} (for false, see appendix)
	speed (answer to question 6a)	
	Integer	Length Content
	02	02 0F A0
	driveDirection (ENUMERATED, so value is encoded as its associated INTEGER value)	
	Enumerated	Length Content
	$0A_{16}$	01_{16} 01_{16}
	lanePosition (CHOICE: we use the simpleLanePosition here, so we encode it as an INTEGER)	
	02	01 03

Finally, we can determine the total length of the SEQUENCE content: 24 bytes. So the length octet ?? is 18_{16}

Question 11. (1 pt) What can go wrong with id and message being both OPTIONAL in the definition of `AlterData` ? Suggest additional encoding rules involving bits 8 and 7 of the identifier octet to avoid this issue.

OPTIONAL means it may be sent, it may be not sent. If we receive only one IA5String in the beginning of the sequence, we wouldn't be able to tell if encodes the `id`, or the `message` field.

The actual way BER deals with this is by defining class `b"10"` "*Context-specific*", to disambiguate the elements in a SEQUENCE or CHOICE tag. With the "AUTOMATIC TAGS" option in the preamble of the ETSI-ITS-CDD DEFINITION, it means that:

```
AlterData ::= SEQUENCE { -- invented --
   id                IA5String OPTIONAL,          -- class 0b10, tag 0. So identifier 0x80
   message           IA5String OPTIONAL,          -- class 0b10, tag 1. So identifier 0x81
   safeDistance      SafeDistanceIndicator,       -- class 0b10, tag 2. So identifier 0x82
   speed             SpeedValue,                  -- class 0b10, tag 3. So identifier 0x83
   driveDirection    DriveDirection,              -- class 0b10, tag 4. So identifier 0x84
   lanePosition      LanePositionOptions          -- class 0b10, tag 5. So identifier 0x85
}
```

Question 12 (2 pts) The most compact ASN.1 encoding rules are the Packed Encoding Rules (PER). This is the one commonly used in 3GPP cellular technologies such as UMTS (3G), LTE (4G), or 5G.
Give (4 maximum) concrete ideas for how PER greatly improves compaction with respect to BER.

There are obvious optimization techniques. Look we need 3 bytes to encode a stupid Boolean in BER!

I expected four ideas out of:

- more concise representation of lengths

- do not write tag and length when known
- no padding for each byte
- an integer constrained between 0 and 28 can be encoded in just five bits
- ascii character encoded on 7 bits not on 8

The last one is not implemented by PER.

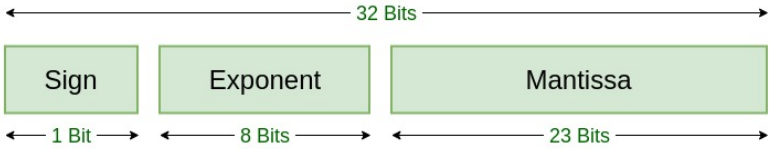
chatGPT's answer (modified with more context) is as follows:

The Packed Encoding Rules (PER) for ASN.1 offer a more compact encoding compared to the Basic Encoding Rules (BER) due to the following core principles:

- Elimination of Tagging Overhead: In PER, tags (identifying the type of data) are often omitted, as the structure of the data is known from the schema. BER includes tags for each element, which increases the size of the encoded data.
- Efficient Length Encoding: While BER explicitly encodes the length of every element, PER uses more efficient methods, such as omitting length fields when the size is fixed or known in advance. This reduces unnecessary length information.
Here; DriveDirection and LaneType never require more than one byte.
- Optimized Representation of Values: PER uses minimal representation for the values. For example, small integers or constrained data types are encoded using fewer bits, unlike BER, which tends to encode values using a fixed-size representation regardless of constraints.
Here, SafeDitanceIndicator is encoded on 1 bit, DriveDirection on 2 bits, LaneType on 5 bits.
- No Padding or Indefinite Lengths: PER avoids padding and indefinite length encodings, both of which are possible in BER. This helps in minimizing the size by only using the exact number of bits required to encode the data.
Here, the BIT STRING EnergyStorageType will not send the byte 0x01 that announces the padding of 1 bit, and the padding bit will not be sent.

These optimizations make PER a more compact alternative to BER, especially for constrained environments where efficiency is crucial.

Appendix A: Single precision IEEE 754 floating-point standard



$$-1^{s} \times 2^{(\exp-127)} \times 1.frac$$

Appendix B: ASCII Table

You can use this ASCII table for direct dec-hex conversion up to 127 !

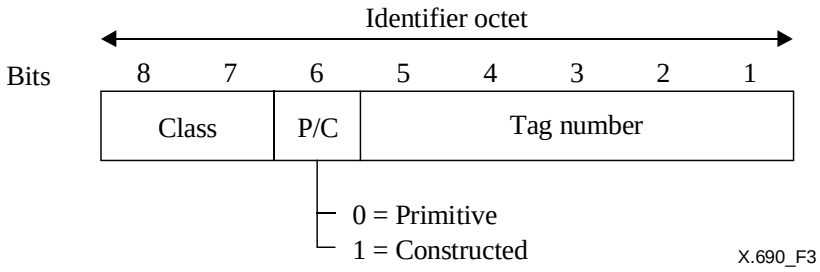
Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Appendix C: Introduction to ASN.1 Basic Encoding Rules

With important points highlighted

The Basic Encoding Rules (BER) uses a Tag-Length-Value (TLV) format for encoding all information. The tag indicates the **identifier** of the data that follows, the length indicates the total length of value (in bytes), and the value represents the actual data **contents**. Each value may consist of one or more TLV-encoded values, each with its own tag, length, and value.

Identifier octet (simplified)
The identifier octet encodes the ASN.1 tag of the type of the data value as follows:



In the context of this exam, we assume that bits 8 and 7 are always set to 0.
bit 6 shall be a 0 if the data contents is primitive, or 1 if it is constructed.
bits 5 to 1 encode the number of the tag as a binary integer with bit 5 as the most significant bit.

Encoding Lengths
Length is always specified in octets, and includes only the octets of the actual value (the contents). It does not include the lengths of the identifier or of the length field itself. We consider two ways to encode lengths:

Short form: for lengths between 0 and 127, the one-octet short form can be used. In the encoding below, bit 8 of the length octet is set to 0, and the length is encoded as an unsigned binary value in the octet's rightmost seven bits. 	**Long form**: for lengths between 0 and 2^{1008} octets, the long form can be used. It starts with an octet that contains the length of the length, followed by the actual length of the encoded value. For example, if the first octet of the length contains the value 4, the actual length of the contents is contained in the next four octets. 

Encoding of a boolean value: The encoding of a boolean value shall be primitive. The contents octets shall consist of a single octet. 0 if the value is FALSE, non-zero if the value is TRUE

Encoding of an integer value: The encoding of an integer value shall be primitive.
An integer value is encoded as a two's complement binary number on the smallest possible number of octets.

Encoding of a bit string: See appendix D "BER encoding of a bitstring value"

Encoding of an enumerated value
The encoding of an enumerated value shall be that of the integer value with which it is associated.

Encoding of a sequence value: See appendix E "BER encoding of a sequence value"

Encoding of a choice value
The encoding of a choice value shall be the same as the encoding of a value of the chosen type.

NOTE 1 – The encoding may be primitive or constructed depending on the chosen type.
NOTE 2 – The tag used in the identifier octets is the tag of the chosen type, as specified in the ASN.1 definition of the choice type.

Appendix D: BER encoding of a bitstring value

With important points highlighted

This is an excerpt of Rec. ITU-T X690 (pp. 8-9), available online at https://www.itu.int/rec/T-REC-X.690

8.6 Encoding of a bitstring value

8.6.1 The encoding of a bitstring value shall be either primitive or constructed at the option of the sender.

NOTE – Where it is necessary to transfer part of a bit string before the entire bitstring is available, the constructed encoding is used.

8.6.2 The contents octets for the primitive encoding shall contain an initial octet followed by zero, one or more subsequent octets.

8.6.2.1 The bits in the bitstring value, commencing with the leading bit and proceeding to the trailing bit, shall be placed in bits 8 to 1 of the first subsequent octet, followed by bits 8 to 1 of the second subsequent octet, followed by bits 8 to 1 of each octet in turn, followed by as many bits as are needed of the final subsequent octet, commencing with bit 8.

NOTE – The terms "leading bit" and "trailing bit" are defined in Rec. ITU-T X.680 | ISO/IEC 8824-1 as follows: *The first bit in a bit string is called the leading bit. The final bit in a bit string is called the trailing bit.*

8.6.2.2 The initial octet shall encode, as an unsigned binary integer with bit 1 as the least significant bit, the number of unused bits in the final subsequent octet. The number shall be in the range zero to seven.

8.6.2.3 If the bitstring is empty, there shall be no subsequent octets, and the initial octet shall be zero.

[…]

8.6.4.2 Example

If of type **`BIT STRING`**, the value **`'0A3B5F291CD'H`** can be encoded as shown below. In this example, the bit string is represented as a primitive:

BitString	Length	Contents
03_{16}	07_{16}	$040A3B5F291CD0_{16}$

Appendix E: BER encoding of a sequence value

With important points highlighted

This is an excerpt of Rec. ITU-T X690 (pp. 10), available online at https://www.itu.int/rec/T-REC-X.690

8.9 Encoding of a sequence value

8.9.1 The encoding of a sequence value shall be constructed.

8.9.2 The contents octets shall consist of the complete encoding of one data value from each of the types listed in the ASN.1 definition of the sequence type, in the order of their appearance in the definition, unless the type was referenced with the keyword OPTIONAL or the keyword DEFAULT.

8.9.3 The encoding of a data value may, but need not, be present for a type which was referenced with the keyword OPTIONAL or the keyword DEFAULT. If present, it shall appear in the encoding at the point corresponding to the appearance of the type in the ASN.1 definition.

EXAMPLE

If of type:

`SEQUENCE {name IA5String, ok BOOLEAN}`

the value:

`{name "Smith", ok TRUE}`

can be encoded as:

Sequence	Length	Contents		
30_{16}	$0A_{16}$			
		IA5String	Length	Contents
		16_{16}	05_{16}	"Smith"
		Boolean	Length	Contents
		01_{16}	01_{16}	FF_{16}