Kodowanie ASN.1

Grzegorz Danilewicz

O czym mówimy?

- Przykładowe reguły kodowania ze specyfikacji ASN.1 do reprezentacji bitowej

Podstawowe kodowanie BER

Podstawy BER

- Podstawowe reguły kodowania BER, to ogólne reguły kodowania dla ASN.1
- Pierwsze zasady kodowania dla ASN.1
- Proste, dość łatwe do wdrożenia
- Zorientowane na strukturę oktetową
- BER może zapewnić składnię transferu dla każdej wartości typu zdefiniowanego w ASN.1

Basic Encoding Rules

- Składnia abstrakcyjna jest konwertowana na składnię transferową przy użyciu reguły kodowania
- ASN.1 zawiera zestaw podstawowych zasad kodowania zdefiniowanych w ISO 8825/X.209
- X.209 jest nieefektywny, więc zdefiniowane są reguły kodowania upakowanego (Packed Encoding Rules), które dają bardziej zwarte rozwiązanie

Basic Encoding Rules

- Każdy element danych jest zakodowany tak, aby zawierał
  - identyfikator (znacznik lub typ)
  - długość wskazującą rozmiar pola danych
  - dane, które zawierają rzeczywistą zawartość obiektu
  - opcjonalną flagę końca zawartości, jeśli długość danych jest nieznana
- Wyrównywanie do granic oktetu

Podstawy kodowania BER

- Podstawowe zasady kodowania dla ASN.1 kodują każdą wartość ASN.1 w trzech częściach
- ILC częściej przedstawiane jako TLV

I(dentyfier)	L(ength)	C(ontents)
T(ype)	L(ength)	V(alue)

Kodowanie typu

	Typ							
bit	7	6	5	4	3	2	1	0
znaczenie	Klasa		P/C	Znacznik				

Kodowanie typu

Typ

bit	7	6	5	4	3	2	1	0
znaczenie	Klasa		P/C	Znacznik				

Klasa	7	6
Uniwersalna	0	0
Aplikacji	0	1
Specyficzny kontekst	1	0
Prywatna	1	1

P/C	6
0	Typ prosty
1	Kolekcja

Kodowanie znacznika

Typ

bit	7	6	5	4	3	2	1	0
znaczenie	Klasa		P/C	Znacznik				

Znacznik typu, kodowany binarnie (zależny od klasy)

Kodowanie znacznika

Typ

bit	7	6	5	4	3	2	1	0
znaczenie	Klasa		P/C	Znacznik				

Znacznik typu, kodowany binarnie (zależny od klasy)

Klasa	7	6
Uniwersalna	0	0

Typy wbudowane (wynikające ze specyfikacji ASN.1)

Kodowanie znacznika

Typ

bit	7	6	5	4	3	2	1	0
znaczenie	Klasa		P/C	Znacznik				

Znacznik typu, kodowany binarnie (zależny od klasy)

Klasa	7	6
Uniwersalna	0	0

Wbudowane typy proste

P/C	6
0	Typ prosty

Kodowanie znacznika dla wartości < 31

	Typ							
bit	7	6	5	4	3	2	1	0
znaczenie	Klasa		P/C	Znacznik				

Znacznik typu, kodowany binarnie (zależny od klasy)

Klasa	7	6	INTEGER				
Uniwersalna	0	0	0	0	0	1	0

P/C	6
0	Typ prosty

Kody dziesiętne znacznika wbudowanych typów prostych dla wartości < 31 (wybrane)

Znacznik typu klasy uniwersalnej	Znaczenie
0	Zarezerwowany dla BER
1	BOOLEAN
2	INTEGER
3	BIT STRING
4	OCTET STRING
5	NULL
6	OBJECT IDENTIFIER
9	REAL
10	ENUMERATED
12	UTF8String
18	NumericString
22	IA5String
27	GeneralString
29	CHARACTER STRING

Kody dziesiętne znacznika typów złożonych

Znacznik typu klasy uniwersalnej	Znaczenie
16	SEQUENCE, SEQUENCE OF
17	SET, SET OF

Kodowanie znacznika dla wartości ≥ 31

	Typ							
bit	7	6	5	4	3	2	1	0
znaczenie	Klasa		P/C	Znacznik				
Typ	0	0	0	1	1	1	1	1
II oktet	1	Bity długiego znacznika (kolejne)						
	1							
	1							
ostatni oktet	0							

Kody dziesiętne znacznika wbudowanych typów prostych dla wartości ≥ 31 (wybrane)

Znacznik typu klasy uniwersalnej	Znaczenie
31	DATE
32	TIME-OF-DAY
33	DATE-TIME
34	DURATION

Ale:

Znacznik typu klasy uniwersalnej	Znaczenie
14	TIME

Przykład kodowania DATE [UNIVERSAL 31]

- Definicja typu ASN.1

Date ::= DATE

- Nadanie wartości zmiennej nowego typu

myDay Date ::= "2022-06-30"

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	0	0							Uniwersalny

Przykład kodowania DATE [UNIVERSAL 31]

- Definicja typu ASN.1

Date ::= DATE

- Nadanie wartości zmiennej nowego typu

myDay Date ::= "2022-06-30"

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	0	0	0						Typ prosty

Przykład kodowania DATE [UNIVERSAL 31]

- Definicja typu ASN.1

Date ::= DATE

- Nadanie wartości zmiennej nowego typu

myDay Date ::= "2022-06-30"

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	0	0	0	1	1	1	1	1	≥ 31

Przykład kodowania DATE [UNIVERSAL 31]

- Definicja typu ASN.1

Date ::= DATE

- Nadanie wartości zmiennej nowego typu

myDay Date ::= "2022-06-30"

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	0	0	0	1	1	1	1	1	1FH
Ostatni oktet znacznika	0								

Przykład kodowania DATE [UNIVERSAL 31]

- Definicja typu ASN.1

Date ::= DATE

- Nadanie wartości zmiennej nowego typu

myDay Date ::= "2022-06-30"

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	0	0	0	1	1	1	1	1	1FH
Ostatni oktet znacznika	0	0	0	1	1	1	1	1	Znacznik = 31

Przykład kodowania DATE [UNIVERSAL 31]

- Definicja typu ASN.1

Date ::= DATE

- Nadanie wartości zmiennej nowego typu

myDay Date ::= "2022-06-30"

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	0	0	0	1	1	1	1	1	1FH
Ostatni oktet znacznika	0	0	0	1	1	1	1	1	1FH

Przykład kodowania DATE [UNIVERSAL 31]

- Definicja typu ASN.1

Date ::= DATE

- Nadanie wartości zmiennej nowego typu

myDay Date ::= "2022-06-30"

- Początek zakodowanej informacji (2 oktety typu)

1F 1F

Podstawy kodowania BER

- Podstawowe zasady kodowania dla ASN.1 kodują każdą wartość ASN.1 w trzech częściach
- ILC częściej przedstawiane jako TLV

I(dentyfier)	L(ength)	C(ontents)
T(ype)	L(ength)	V(alue)

Kodowanie długości

- Forma krótka, gdy długość jest mniejsza od 128 oktetów
- Forma długa, gdy długość mieści się w granicach $< 128, 2^{1008})$ oktetów
- Długość nieskończona (nieokreślona)

Forma krótka pola długości

Oktet	7	6	5	4	3	2	1	0
Długość zawartości	0	Długość pola V						
Zawartość lub wartość	0 < V < 128 oktetów							

Forma długa pola długości

Oktet	7	6	5	4	3	2	1	0
Długość wskaźnika długości	1	K – długość pola L						
Długość zawartości	Pole L 1. K oktetów 2. Koduje ile oktetów ma V							
Zawartość lub wartość	V							

Przykład dla zawartości równej 128 oktetów:
1. 1000 0001 – 81H
2. 1000 0000 – 80H

Długość nieskończona

Oktet	7	6	5	4	3	2	1	0
Długość wskaźnika długości	1	0	0	0	0	0	0	0
Zawartość lub wartość	(nieokreślona długość)							
Jawny wskaźnik końca zawartości	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

Przykład:
1. 1000 0000 – 80H
...
N – 1 0000 0000 – 00H
N 0000 0000 – 00H

Kodowanie typów złożonych

- Typy zawarte w typach są wysyłane w części zawartości wiadomości
- Wskaźnik długości zawartości typu zawierającego obejmuje wszystkie oktety typu zawartego

Typ A	Długość zawartości A	Typ B	Długość zawartości B	Zawartość B
		Zawartość A		

Przykłady

```
MyModule DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
  B ::= [APPLICATION 0] EXPLICIT BOOLEAN
  b B ::= FALSE

  i INTEGER ::= -1
  j INTEGER ::= 255

  S ::= SEQUENCE {
    a INTEGER,
    b OCTET STRING
  }
  s S ::= { a 4, b 'ABCD'H }
END
```


Przykłady

```
B ::= [APPLICATION 0] EXPLICIT BOOLEAN
b B ::= FALSE
```

T	L	V		
		T	L	V
60H	03H	01H	01H	00H
0110 0000	0000 0011	0000 0001	0000 0001	0000 0000

Przykłady

B ::= [APPLICATION 0] EXPLICIT BOOLEAN

b B ::= FALSE

T	L	V		
		T	L	V
60H	03H	01H	01H	00H
0110 0000	0000 0011	0000 0001	0000 0001	0000 0000

Klasa: Aplikacja

Przykłady

B ::= [APPLICATION 0] EXPLICIT BOOLEAN

b B ::= FALSE

T	L	V		
		T	L	V
60H	03H	01H	01H	00H
0110 0000	0000 0011	0000 0001	0000 0001	0000 0000

Klasa: Aplikacja
P/C: aplikacja musi być typu złożonego

Przykłady

B ::= [APPLICATION 0] EXPLICIT BOOLEAN

b B ::= FALSE

T	L	V		
		T	L	V
60H	03H	01H	01H	00H
0110 0000	0000 0011	0000 0001	0000 0001	0000 0000

Klasa: Aplikacja
P/C: aplikacja musi być typu złożonego
Znacznik: o

Przykłady

B ::= [APPLICATION 0] EXPLICIT BOOLEAN

b B ::= FALSE

T	L	V		
		T	L	V
60H	03H	01H	01H	00H
0110 0000	0000 0011	0000 0001	0000 0001	0000 0000

Długość zawartości typu B

Przykłady

```
B ::= [APPLICATION 0] EXPLICIT BOOLEAN
b B ::= FALSE
```

T	L	V		
		T	L	V
60H	03H	01H	01H	00H
0110 0000	0000 0011	0000 0001	0000 0001	0000 0000

Zawartość typu B:
[UNIVERSAL 1], boolean

Przykłady

```
B ::= [APPLICATION 0] EXPLICIT BOOLEAN
b B ::= FALSE
```

T	L	V		
		T	L	V
60H	03H	01H	01H	00H
0110 0000	0000 0011	0000 0001	0000 0001	0000 0000

Zawartość typu B:
[UNIVERSAL 1], boolean
Długość zawartości typu boolean

Przykłady

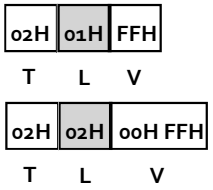
```
B ::= [APPLICATION 0] EXPLICIT BOOLEAN
b B ::= FALSE
```

T	L	V		
		T	L	V
60H	03H	01H	01H	00H
0110 0000	0000 0011	0000 0001	0000 0001	0000 0000

Zawartość typu B:
[UNIVERSAL 1], boolean
Długość zawartości typu boolean
Zawartość – FALSE (00H)

Przykłady

```
i INTEGER ::= -1
j INTEGER ::= 255
```

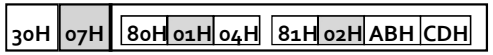


[UNIVERSAL 2], integer, -1

[UNIVERSAL 2], integer, 255

Przykłady – kodowana długość

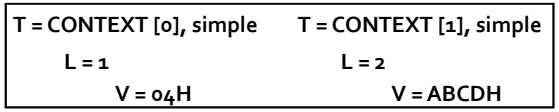
```
S ::= SEQUENCE {
    a INTEGER,
    b OCTET STRING
}
s S ::= { a 4, b 'ABCD'H }
```



T L V

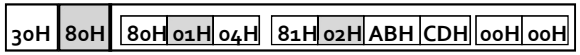
T = [UNIVERSAL 16], structured, sequence

L = 7



Przykłady – długość nieskończona

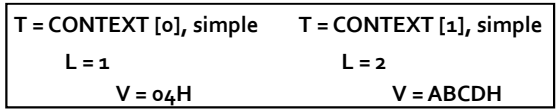
```
S ::= SEQUENCE {
    a INTEGER,
    b OCTET STRING
}
s S ::= { a 4, b 'ABCD'H }
```



T L V

T = [UNIVERSAL 16], structured, sequence

L = 7



Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```


Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	1	0							Kontekst

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	**1**	**0**	**1**						**Typ złożony**

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**Znacznik = 1**

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**A1H**

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Długość	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**09H**

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	0	0							Uniwersalna

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	0	0	0						Typ prosty

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	0	0	0	0	0	0	1	0	INTEGER

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09
02

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	0	0	0	0	0	0	1	0	02H

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09
02 01

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Długość	0	0	0	0	0	0	0	1	**01H**

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09
02 01 30

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Wartość	0	0	1	1	0	0	0	0	**30H = 48**

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09
02 01 30

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	0	0							**Uniwersalna**

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09
02 01 30

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	0	0	0						**Typ prosty**

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09
02 01 30

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	0	0	0	0	0	1	0	0	OCTET STRING

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09
02 01 30
04

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Typ	0	0	0	0	0	1	0	0	04H

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09
02 01 30
04 04

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Długość	0	0	0	0	0	1	0	0	04H

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09
02 01 30
04 04 44 58 2D 37

Oktet	7	6	5	4	3	2	1	0	Kodowanie
Wartość	0	1	0	0	0	1	0	0	44H
	0	1	0	1	1	0	0	0	58H
	0	0	1	0	1	1	0	1	2DH
	0	0	1	1	0	1	1	1	37H

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09
02 01 30 długość 3
04 04 44 58 2D 37 długość 6

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09
02 01 30 długość 1
04 04 44 58 2D 37

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09
02 01 30
04 04 44 58 2D 37 długość 4

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod:
A1 09
02 01 30
04 04 44 58 2D 37

		labID			currentProject		
T = A1	L = 09	T = 02	L = 01	V = 30	T = 04	L = 04	V = 44 58 2D 37
		Typ złożony – Division (zawartość 9 oktetów)					

Przykład CHOICE

```
M DEFINITIONS IMPLICIT TAGS ::= BEGIN
Division ::= CHOICE {
  manufacturing [0] IMPLICIT SEQUENCE {
    plantID      INTEGER,
    majorProduct OCTET STRING
  },

  r-and-d       [1] IMPLICIT SEQUENCE {
    labID          INTEGER,
    currentProject OCTET STRING
  },

  unassigned    [2] IMPLICIT  NULL
}
END
```

```
value Division ::=
  r-and-d :
  {
    labID 48,
    currentProject '44582D37'H
  }
```

Kod heksadecymalny dla ciągu bitów wysyłanego i odbieranego do/z łącza:

A1 09 02 01 30 04 04 44 58 2D 37

Zapis wartości typu REAL

- Zapis wartości rzeczywistych w ASN.1

  { mantysa, baza, cecha }

  $$\textbf{wartość} = \textbf{mantysa} \times \textbf{baza}^{\textbf{cecha}}$$

- Przykład zapisu wartości 10.0 w ASN.1

  ten REAL ::= { 10, 2, 0 }

  $$\text{ten} = 10 \times 2^0$$

Zapis wartości typu REAL

- Zasady
  - mantysa i cecha mogą przyjmować dowolną (dodatnią bądź ujemną) wartość całkowitą
  - baza może przyjmować wartość 2, 4 lub 16
  - każda kombinacja tych wartości jest dozwolona
- Zapis z bazą 10 jest możliwy w notacji ASN.1, ale wtedy zapis wartości traktowany jest jak łańcuch znaków

Zapis wartości typu REAL

- Dla kodowania mantysa jest dodatkowo dzielona na kolejne pola

  $$\textbf{mantysa} = \textbf{znak} \times \textbf{liczba} \times 2^{\textbf{skalowanie}}$$

- Zasady
  - znak może przyjmować wartość +1 lub -1
  - liczba jest dodatnią wartością całkowitą
  - skalowanie może przyjmować wartości 0, 1, 2 lub 3

Zapis wartości typu REAL

- Przykład

ten REAL ::= { 10, 2, 0 }

ten = 10×2^0 (mantysa = 10)

mantysa = $1 \times 10 \times 2^0$

(znak = +1, liczba = 10, skalowanie = 0)

Kodowanie wartości typu REAL w BER

1: T = 09H

2: L = (1 + e + n)

3:

Oktet informacyjny				
1	znak	baza	skalowanie	kodowanie cechy

4: 1 ..... e

(e oktetów dla kodowania wartości cechy; pierwszy oktet zawiera długość kodowanej wartości cechy, gdy kodowanie cechy = 11)

4+e: 1 ..... n

(n oktetów do kodowania wartości dodatniej liczby całkowitej tak żeby $\mathbf{mantysa} = \mathbf{znak} \times \mathbf{liczba} \times 2^{\mathbf{skalowanie}}$)

Kodowanie wartości typu REAL w BER

znak	
0	dla +1
1	dla -1

baza	
00	dla 2
01	dla 4
10	dla 16

skalowanie	
00	dla 0
01	dla 1
10	dla 2
11	dla 3

kodowanie cechy	
00	cecha kodowana w następnym oktecie
01	cecha kodowana w następnych dwóch oktetach
10	cecha kodowana w następnych trzech oktetach
11	kodowanie liczby oktetów cechy w następnym oktecie, a kodowanie cechy w następnych oktetach

Kodowanie wartości typu REAL w BER

1: T = 09H

2: L = 03H

3:

Oktet informacyjny				
1	0 (+1)	00 (2)	00 (0)	00 (cecha kodowana w następnym oktecie)

(80H)

4: Cecha = 00H

5: Liczba w mantysie = 0AH

ten REAL ::= { 10, 2, 0 }

Kod heksadecymalny dla ciągu bitów wysyłanego i odbieranego do/z łącza:

090380000A

BER – CER – DER

- BER posiada wiele opcji kodowania, np.
  - kodowanie o określonej długości a kodowanie o nieokreślonej długości
  - niesegmentowane wartości łańcuchowe a wartości łańcuchowe podzielone na segmenty
- Jedna wartość może mieć wiele zakodowanych postaci
- Nieodpowiednie dla np. sum kontrolnych czy podpisów cyfrowych
- Rozwiązanie: ograniczone reguły kodowania wywodzące się z BER
  - kanoniczne zasady kodowania (CER)
    - kodowanie zawsze o nieokreślonej długości
    - ciągi zakodowane w segmentach po 1000 oktetów
  - Wyróżnione reguły kodowania (DER)
    - zawsze kodowanie o określonej długości
    - brak segmentacji w wartościach ciągów

Podstawy PER

- Problemy BER
  - BER jest gadatliwy – ilość informacji sterujących jest ogromna w porównaniu z ilością rzeczywistych danych
  - BER nie jest dobry, jeśli zdolności przesyłowe są ograniczone
- Packed Encoding Rules (PER) minimalizują liczbę przesyłanych oktetów i bitów
- Bardziej złożone niż BER
- Zorientowane na kodowanie bitowe

Podstawy PER

- Specjalistyczne kodowanie oparte na znajomości typów danych
- Generuje znaczniki tylko wtedy, gdy są potrzebne, aby zapobiec niejasności
- Generuje długości tylko wtedy, gdy rozmiar obiektu może się zmieniać
- Długości reprezentowane są w zwartej formie

Podstawy PER

- Kodowanie nie zawsze jest wyrównane do granic oktetów chyba że zastosowano wariant „wyrównany” (aligned) reguł
- Elementy opcjonalne w sekwencji są wskazywane przez listę flag jednobitowych umieszczonych na początku sekwencji

Podstawy PER

- Tylko istotne informacje są przesyłane przy użyciu najmniejszej potrzebnej liczby bitów
- Informacje kontrolne są wysyłane tylko w razie potrzeby
  - gdy obecne są opcjonalne komponenty
  - długości list
  - wybrane alternatywy z CHOICE
- Warianty z wyrównaniem do oktetu i bez
  - wyrównany – czasami dodawane są bity wypełniające, aby wartości zaczynały się od granicy oktetu
  - niewyrównany – brak bitów wypełniających (UPER)

Prosty przykład kodowania PER

- Jak kodować wartość całkowitą 5?

	BER	PER
INTEGER	02H 01H 05H	01H 05H
INTEGER (0..5)	02H 01H 05H	101B
INTEGER (0..10)	02H 01H 05H	0101B
INTEGER (5..10)	02H 01H 05H	000B

- Uwagi:
  - Bez identyfikatora typu
  - Długość kodowana tylko, gdy potrzeba
  - Kodowanie wartości używa najmniejszej koniecznej liczby bitów

Prosty przykład kodowania PER

- Jak kodować wartość całkowitą 5?

	BER	PER
INTEGER	02H 01H 05H	
INTEGER (0..5)	02H 01H 05H	
INTEGER (0..10)	02H 01H 05H	
INTEGER (5..10)	02H 01H 05H	000B

Ograniczanie typu nie ma dużego znaczenia w kodowaniu BER
Każdy przypadek wymaga 24 bitów

- Uwagi:
  - Bez identyfikatora typu
  - Długość kodowana tylko, gdy potrzeba
  - Kodowanie wartości używa najmniejszej koniecznej liczby bitów

Prosty przykład kodowania PER

- Jak kodować wartość całkowitą 5?

	BER	PER
INTEGER	02H 01H 05H	01H 05H
INTEGER (0..5)	02H 01H 05H	101B
INTEGER (0..10)	02H 01H 05H	0101B
INTEGER (5..10)	02H 01H 05H	000B

Ograniczanie typu ma realne znaczenia w kodowaniu PER

- Uwagi:
  - Bez identyfikatora typu
  - Długość kodowana tylko, g
  - Kodowanie wartości używa
  - koniecznej liczby bitów

Prosty przykład kodowania PER

- Jak kodować wartość całkowitą 5?

	BER	PER
INTEGER	02H 01H 05H	01H 05H
INTEGER (0..5)	02H 01H 05H	101B
INTEGER (0..10)	02H 01H 05H	0101B
INTEGER (5..10)	02H 01H 05H	000B

- Uwagi:
  - Bez identyfikatora typu
  - Długość kodowana tylko, g
  - Kodowanie wartości używa koniecznej liczby bitów

N wartości kodowanych na maks. $\lceil \log_2 N \rceil$ bitach

Prosty przykład kodowania PER

- Jak kodować wartość całkowitą 5?

	BER	PER
INTEGER	02H 01H 05H	01H 05H
INTEGER (0..5)	02H 01H 05H	101B
INTEGER (0..10)	02H 01H 05H	0101B
INTEGER (5..10)	02H 01H 05H	000B

- Uwagi:
  - Bez identyfikatora typu
  - Długość kodowana tylko, g
  - Kodowanie wartości używa koniecznej liczby bitów

Dodatkowo indeksowanie wartości z przedziału

Prosty przykład kodowania PER

- Jak kodować wartość całkowitą 5?

	BER	PER	Zysk
INTEGER	02H 01H 05H	01H 05H	33%
INTEGER (0..5)	02H 01H 05H	101B	87,5%
INTEGER (0..10)	02H 01H 05H	0101B	83,3%
INTEGER (5..10)	02H 01H 05H	000B	87,5%

- Uwagi:
  - Bez identyfikatora typu
  - Długość kodowana tylko, g
  - Kodowanie wartości używa koniecznej liczby bitów

Dodatkowo indeksowanie wartości z przedziału

Przykład kodowania PER dla typu złożonego

- Kodowanie typu złożonego

```
S ::= SEQUENCE {        -- AUTOMATIC TAGS
  a INTEGER (1..5),
  b BOOLEAN OPTIONAL
}
s S ::= { a 4, b TRUE }
```

```
S SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 6
  a INTEGER: tag = [0] primitive; length = 1
    4
  b BOOLEAN: tag = [1] primitive; length = 1
    TRUE
```

Przykład kodowania PER dla typu złożonego

```
S SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 6
  a INTEGER: tag = [0] primitive; length = 1
    4
  b BOOLEAN: tag = [1] primitive; length = 1
    TRUE
```

BER, określona długość **PER**

30 06 80 01 04 81 01 FF B8

```
sequence, len = 6
  context, tag = 0 (integer), len = 1, v = 4
    context, tag = 1 (boolen), len = 1, v = <> 0
```


Przykład kodowania PER dla typu złożonego

```
S ::= SEQUENCE {              -- AUTOMATIC TAGS
  a INTEGER (1..5),
  b BOOLEAN OPTIONAL
}
s S ::= { a 4, b TRUE }
```

BER, określona długość **PER**

30 06 80 01 04 81 01 FF B8

1011 1000

Przykład kodowania PER dla typu złożonego

```
S ::= SEQUENCE {              -- AUTOMATIC TAGS
  a INTEGER (1..5),
  b BOOLEAN OPTIONAL
}
s S ::= { a 4, b TRUE }
```

BER, określona długość **PER**

30 06 80 01 04 81 01 FF B8

1011 1000

składowa b obecna

Przykład kodowania PER dla typu złożonego

```
S ::= SEQUENCE {              -- AUTOMATIC TAGS
  a INTEGER (1..5),
  b BOOLEAN OPTIONAL
}
s S ::= { a 4, b TRUE }
```

Ograniczenie wartości – struktura indeksowana od 0 do 4

BER, określona długość **PER**

30 06 80 01 04 81 01 FF B8

1011 1000

Przykład kodowania PER dla typu złożonego

```
S ::= SEQUENCE {              -- AUTOMATIC TAGS
  a INTEGER (1..5),
  b BOOLEAN OPTIONAL
}
s S ::= { a 4, b TRUE }
```

Ograniczenie wartości – struktura indeksowana od 0 do 4

BER, określona długość
30 06 80 01 04 81 01 FF

PER
B8
1011 1000

Obecna wartość spod indeksu 3 == 4

Przykład kodowania PER dla typu złożonego

```
S ::= SEQUENCE {              -- AUTOMATIC TAGS
  a INTEGER (1..5),
  b BOOLEAN OPTIONAL
}
s S ::= { a 4, b TRUE }
```

TRUE: <> 0

BER, określona długość
30 06 80 01 04 81 01 FF

PER
B8
1011 1000

Przykład kodowania PER dla typu złożonego

```
S ::= SEQUENCE {              -- AUTOMATIC TAGS
  a INTEGER (1..5),
  b BOOLEAN OPTIONAL
}
s S ::= { a 4, b TRUE }
```

Uzupełnienie do oktetu

BER, określona długość
30 06 80 01 04 81 01 FF

PER
B8
1011 1000

Przykład kodowania PER dla typu złożonego

```
S ::= SEQUENCE {              -- AUTOMATIC TAGS
  a INTEGER (1..5),
  b BOOLEAN OPTIONAL
}
s S ::= { a 4, b TRUE }
```

BER, określona długość
30 06 80 01 04 81 01 FF

PER
B8
1011 1000
zysk 7/8 = 87,5%