PODSTAWY TELEINFORMATYKI

Ćwiczenie 2a

Detekcja i korekcja błędów w transmisji danych

Dla zapewnienia szybszej i sprawiedliwej obsługi wszystkich komputerów połączonych w sieć, większość sieci komputerowych dzieli dane na małe bloki nazywane pakietami. Aby rozróżnić ogólne pojęcie pakietu i specyficzne formy pakietów używane w konkretnej technice sieciowej, pakiety konkretnego formatu nazywamy ramkami. Każda technika sieciowa szczegółowo definiuje format i rozmiar ramek, jakich mogą używać nadawcy. W formacie ramki, który może być używany w sieciach zorientowanych na przesyłanie znaków (np.: łącza szeregowe RS-232) używa się zarejestrowanych znaków do oznaczenia początku i końca każdej ramki. Przesyłanie znaku początku i znaku końca stanowi zaletę wtedy, gdy bierze się pod uwagę możliwość dużych opóźnień i zawodnych komputerów. Dzieje się tak, ponieważ transmisja asynchroniczna pozwala na wystąpienie dowolnie długich odstępów czasu między kolejnymi znakami. Poza tym zawodny komputer może się zawiesić, a po restarcie kontynuować transmisję. Oznaczenie zarówno początku, jak i końca ramki zapewnia odporność na tego typu problemy.

Urządzenia elektroniczne i linie transmisyjne są podatne na zakłócenia elektromagnetyczne, dlatego dane przesyłane przez sieć mogą ulegać uszkodzeniom lub ginąć. Mała zmiana sygnału elektrycznego może spowodować błędną interpretację przez odbiorcę jednego lub więcej bitów danych. Zakłócenia mogą całkowicie zniszczyć sygnał, co może spowodować, że odbiorca wcale nie zauważy tego sygnału. Możliwy jest również odwrotny przypadek polegający na tym, że nadawca nie wysyła żadnego sygnału, a odbiorca otrzymuje jakiś sygnał. Takie problemy, które mogą powodować pojawienie się przypadkowych informacji, gubienie i zmianę danych, nazywają się błędami transmisji.

Wykrywanie oraz korekcja błędów jest głównym zadaniem protokołu łącza danych. Ciąg kontrolny umożliwia wykrywanie pewnych błędów w bloku danych, podobnie jak kontrola parzystości w przypadku transmisji asynchronicznej znaków. Po wykryciu błędów protokół powinien zdecydować albo o odrzuceniu danych, albo o skorygowaniu błędu. Istnieją dwa sposoby korekcji błędów: retransmisja danych ARQ (ang. Automatic Repear reQuest) lub z wykorzystaniem bezpośredniej korekcji błędów FEC (ang. Forward Error Control). Detekcja oraz korekcja błędów jest możliwa wyłącznie wtedy, jeżeli do przekazywanej wiadomości wprowadzono pewną redundancję (korekcja błędów wymaga większej redundancji aniżeli sama ich detekcja). Wynika stąd, że bezpośrednia korekcja błędów jest mniej efektywna od retransmisji danych. Ponadto algorytm bezpośredniej korekcji błędów nie jest w stanie skorygować wszystkich stwierdzonych błędów poprawnie, a więc stopa błędów w danych odzyskanych metodą FEC i tak przewyższa stopę błędów dla schematu ARQ. Ogólnie rzecz biorąc, automatyczna retransmisja ARQ jest preferowanym sposobem korekcji błędów w transmisji danych. Niekiedy jednak schemat ARQ nie jest zalecany, na przykład w przypadku łączności z

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

sondą kosmiczną realizowanej w trybie simpleksowym, a wtedy jedyną możliwością jest bezpośrednia korekcja błędów.

Detekcja i korekcja błędów.
Kontrola parzystości.

Parzystość umożliwia wykrywanie pewnych błędów, które mogą pojawić się w wiadomości przekazywanej od nadajnika do odbiornika. W celu skorygowania błędu zniekształcony znak (7 bitów kodu ASCII oraz bit parzystości) musi być przesłany raz jeszcze. Kontrola parzystości jest bardzo prostym sposobem wykrywania błędów, a pojedynczy bit parzystości jest wystarczający przy detekcji błędów w wiadomościach o długości nie przekraczającej ok. 12 bitów. Przy dłuższych wiadomościach każdy jej bajt jest zabezpieczany bitem parzystości. Przy dłuższych wiadomościach stosuje się jednak bardziej efektywne zasady kodowania detekcyjnego, dostosowane bardziej do ich charakteru. Wszystkie te schematy opierają się na dodawaniu do przekazywanej wiadomości redundancji umożliwiającej po stronie odbiorczej rozróżnienie wiadomości uszkodzonych od poprawnych. Bez uzupełniającej redundancji każda wiadomość jest poprawna i nie ma sposobu na rozróżnienie wiadomości uszkodzonych od poprawnych.

Kontrola parzystości polega na dodaniu do przekazywanej wiadomości tzw. bitu parzystości w taki sposób, że przekazywana wiadomość (łącznie z dodanym bitem parzystości) zawiera albo parzystą, albo nieparzystą liczbę jedynek, w zależności od tego, czy jest stosowana normalna czy negatywna kontrola parzystości. Rzecz jasna, odbiornik informacji musi pracować z takim samym schematem parzystości. Rozważmy dla przykładu transmisję znaku ASCII „J" przy zastosowaniu negatywnej kontroli parzystości. Znak „J" jest kodowany sekwencją bitów „1001010", a więc bit parzystości musi być wyzerowany, aby łączna liczba jedynek była nieparzysta. Bit parzystości jest dołączany jako najbardziej znaczący bit słowa ośmiobitowego.

P	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0

Kontrola parzystości umożliwia detekcje dowolnej nieparzystej liczby błędów (w tym przekłamanie dokładnie jednego bitu). W powyższym przykładzie załóżmy, że bit 2 został przekłamany – do odbiornika dotarła wiadomość w poniższej postaci:

P	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0

W otrzymanej sekwencji łączna liczba jedynek jest parzysta, a to oznacza przekłamanie otrzymanej sekwencji. Odbiornik nie jest w stanie określić, który bit błędnie odebrano, a więc transmisja całego znaku musi zostać powtórzona. Transmisja znaku musi być również powtórzona

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

wtedy, gdy sekwencję kodującą znak odebrano bezbłędnie, a przekłamaniu uległ wyłącznie bit parzystości. Należy również zauważyć, że przekłamanie jakiejkolwiek parzystej liczby błędów nie zostanie wykryte, gdyż parzysta liczba błędów nie zmienia parzystości lub nieparzystości łącznej liczby jedynek w słowie. Pojedyncza kontrola parzystości umożliwia jedynie detekcję dowolnej nieparzystej liczby błędów. Jeżeli przekłamanie jednego bitu, przekłamanie dwóch bitów itp. uznamy za błędy różnych typów, to pojedyncza kontrola parzystości wykrywa tylko połowę wszystkich typów błędów. Jeżeli jednak rozważymy liczbę, a nie typ powstałych błędów, to stwierdzimy, że kontrola parzystości umożliwia wykrycie wszystkich przypadkowych, pojawiających się niezależnie błędów, ponieważ w praktyce prawdopodobieństwo pojedynczego błędu, aczkolwiek bardzo małe, jest i tak znacznie większe od prawdopodobieństwa jednoczesnego wystąpienia dwóch błędów. Aby to dokładniej zobrazować załóżmy, że prawdopodobieństwo przekłamania dowolnego bitu (p) wynosi $l0^{-6}$, a wtedy dla n = 1000 bitów prawdopodobieństwo przekłamania jednego bitu wynosi ok. 10^{-3}, dwóch bitów ok. 10^{-6}, co wynika z przybliżonego wzoru na prawdopodobieństwo *m* błędów w ciągu *n* bitów (obowiązującego przy niskiej bitowej stopie błędów) $\approx (np)^m$.

Niestety, istnieje jeszcze inna klasa błędów, tzw. błędy seryjne, kiedy pojawianie się błędów nie jest niezależne. Seria błędów rozpoczyna się pierwszym przekłamanym bitem, a kończy ostatnim przekłamanym. Liczba błędów w serii może być równie dobrze parzysta, jak i nieparzysta, a więc kontrola parzystości będzie w stanie wykryć wyłącznie połowę tych błędów. Oznacza to, że przy dłuższych wiadomościach kontrola parzystości funkcjonuje prawidłowo, ale nie jest efektywna. Ulepszony schemat detekcji błędów dla długich wiadomości stanowi kontrola za pomocą redundancji cyklicznej CRC (ang. *Cyclic Redundancy Check)*. Parzystość jest granicznym przypadkiem (dla bloku będącego pojedynczym bitem) kontroli za pomocą redundancji cyklicznej CRC.

Sekwencja 7 bitów generuje 128 kombinacji kodowych. Ponieważ bit parzystości jest 8 bitem, to liczba możliwych kombinacji wzrasta do 256, ale tylko 128 z nich jest poprawnych. Pozostałe 128 kombinacji stanowi redundancję wprowadzoną przez bit parzystości. Jeżeli odbiornik otrzymuje niepoprawne słowo kodowe, musi przyjąć, że w czasie transmisji wystąpił błąd. Jeżeli odbiornik odbiera ważne słowo, musi założyć, że jest to słowo poprawne (nawet gdy nadajnik wysłał inną kombinację). Oznacza to, że nie można być absolutnie pewnym, że odebrana wiadomość jest poprawna (z kolei, jeżeli w wiadomości pojawił się błąd, jest ona na pewno niepoprawna). Ta obserwacja obowiązuje dla dowolnego schematu wykrywania błędów, a więc można wywnioskować, że transmisja danych nigdy nie jest w 100% pewnym przedsięwzięciem. Tym niemniej, jeżeli transmisja krótkich wiadomości jest zabezpieczona bitem parzystości, a dłuższych przy pomocy bardziej skomplikowanego schematu to prawdopodobieństwo niewykrycia błędu jest stosunkowa niewielkie.

Istnieją dwa rodzaje kontroli parzystości – sprawdza się parzystość (ang. even parity) lub nieparzystość (ang. odd parity). Nadawca i odbiorca muszą uzgodnić którego rodzaju używają. W obu przypadkach wyliczenie bitu parzystości jest bardzo proste. I kiedy dla przypadku badania

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

parzystości bit parzystości ustawia się tak by łączna liczba jedynek była parzysta, to podczas badania nieparzystości łączna liczba jedynek ma być nieparzysta. Po odebraniu znaku odbiorca zlicza bity o wartości „1" dla uzyskania bitu parzystości. Jeśli wszystkie bity zostały przesłane poprawnie, to wynik obliczenia będzie taki sam jak bit parzystości wysłany przez nadawcę. Jeśli którykolwiek pojedynczy bit zostanie przesłany niepoprawnie to wynik będzie różny.

Kod Hamminga korygujący błąd pojedynczy.

Kod Hamminga (korygujący błąd pojedynczy) wykrywa dowolny błąd pojedynczy oraz wskazuje jego pozycję, co umożliwia stosowną korekcję (zmiana bitu na przeciwny). Kod Hamminga jest przykładem kodowania FEC. Bity kontrolne są umieszczane w tych pozycjach wiadomości, których numery wyrażają się przez potęgę liczby 2. Jeżeli wiadomość jest zbudowana z 10 bitów $m_a = m_1m_2...m_{10}$, to bity kontrolne c_j pojawiają się wtedy, gdy a w m_a, jest potęgą liczby dwa.

m_{10}	m_9	m_8	m_7	m_6	m_5	m_4	m_3	m_2	m_1
d_6	d_5	c_4	d_4	d_3	d_2	c_3	d_1	c_2	c_1

Odbiornik wykorzystuje niepoprawne bity kontrolne do wyznaczenia pozycji przekłamanego bitu, na przykład, jeżeli niepoprawne są bity c4, c2 oraz c1, to Przekłamany bit występuje na pozycji m_7 (7 = 4 + 4 + 2 + 1).

Jeżeli długość słowa kodowego jest równa *n*, a długość bloku danych *k*, to sprawność kodu wynosi E = *k/n*. Kody tego typu są nazywane kodami blokowymi *(n*, k); przedstawiony wcześniej 10-bitowy kod był kodem blokowym (10, 6) o sprawności 60%. Bity kontrolne zajmują pozycje o numerach będących potęgami liczby 2, a to powoduje, że sprawność kodu wykazuje maksima dla długości wiadomości $n=2^j-1$ (dla $j \geq 2$, gdzie j jest liczbą bitów kontrolnych), a więc dla *n* = 3, 7, 15,... Ponieważ $k = n - j$, to sprawność kodu wynosi $E = 1 - j/n$ (E = 33,3%, 57,1%, 73,3%,...). Podstawową wadą opisanego kodu oraz innych kodów FEC jest to, że ich sprawność dla krótkich wiadomości jest niewielka, natomiast dla wiadomości długich prawdopodobieństwo błędów wielokrotnych wzrasta nadmiernie.

Z podanych rozważań wynika, że minimalna liczba bitów kontrolnych dla 4-bitowych danych źródłowych wynosi 3, co tworzy kod blokowy (7, 4). W celu udowodnienia tego wyniku rozpatrzmy przypadek ogólny. Źródło generuje 2^k poprawnych słów k-bitowych. Dla kodu korygującego pojedyncze błędy o długości n każde poprawne słowo kodowe musi się różnić w jednej pozycji od n niepoprawnych słów kodowych, a więc musi istnieć co najmniej $(n+1)2^k$ słów kodowych. Ponieważ jednak wszystkich możliwych słów kodowych jest 2^n, to:

$$(n+1)2^k \leq 2^n$$

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

Dla j bajtów kontrolnych mamy n = k+j, a więc dla kodu korygującego pojedyncze błędy spełniona jest nierówność:

$(k + j + 1) \leq 2^j$

Ponownie dla k=4 otrzymujemy j>=3.

Najlepszym sposobem wytłumaczenia zasady działania kodu jest odpowiednio dobrany przykład. Rozważając ponownie kod blokowy (7, 4); nadajnik przekazuje sekwencję 1011. Blok danych zawiera „1" w pozycji m_7, m_5 oraz m_3, a „0" w pozycji m_6. Cyfry dwójkowe odpowiadające wartościom a w m_a są dodawane (gdy dane są równe „1") w arytmetyce modulo-2 (z pominięciem przeniesienia).

1 1 1

1 0 1

0 1 1 +

---

0 0 1

W wyniku dodawania otrzymuje się wartości trzech bitów kontrolnych c_3 =0, c_2 =0, oraz c_1 =1, a wiec nadajnik nadaje następującą sekwencję kodowa:

m_7	m_6	m_5	M_4	m_3	m_2	m_1
d_4	d_3	d_2	c_3	d_1	c_2	c_1
1	0	1	0	1	0	1

W odbiorniku cyfry binarne odpowiadające pozycjom sekwencji zawierającym „1” są ponownie sumowane z wykorzystaniem arytmetyki modulo-2:

1 1 1

1 0 1

0 1 1

0 0 1 +

---

0 0 0

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

W wyniku otrzymuje się sekwencje samych zer 000, co sygnalizuje brak przekłamań.

Zakładając, że w trakcie transmisji zaistniał pojedynczy błąd, bit m_3 równy „1” został odebrany jako „0”. W wyniku realizacji tej samej procedury w odbiorniku otrzymuje się:

1	1	1
1	0	1
0	0	1 +
0	1	1

Otrzymana sekwencja wskazuje, że bit m_3 został przekłamany, co pozwala go skorygować (korekcja pojedynczych błędów). Procedura pozwala również korygować przekłamania bitów kontrolnych.

Stwierdzono wcześniej, że tak naprawdę to błędy seryjne, a nie pojedyncze przekłamania, stanowią problem w transmisji danych. Przeplatanie jest modyfikacją kodu Hamminga korygującego pojedynczy błąd zabezpieczający transmisję przed błędami seryjnymi. Zakładając, że kod zaprojektowano tak, aby korygować serię nie więcej niż osiem błędów; najpierw nadawane są wszystkie bity m_7, a następnie wszystkie bity m_6, itd. Dwa bity należące do tego samego bloku są rozdzielane przynajmniej ośmioma innymi bitami, a więc seria ośmiu błędów będzie wpływać na jeden bit w dowolnym słowie kodowym, a to pozwala korygować serię nie więcej niż ośmiu błędów.

Wykrywanie błędów za pomocą CRC.

Sieć może wykrywać więcej błędów bez zwiększania ilości informacji dodatkowych przesyłanych z każdym pakietem poprzez zastosowanie metody zwanej cykliczną kontrolą redundancji (ang. Cyclic Redundancy Check – CRC). Umożliwia ona wykrycie większej liczby błędów niż metoda sum kontrolnych. Suma kontrolna CRC jest bardzo powszechnym algorytmem liczenia sum kontrolnych dla bloku danych. Od stosowanej czasami sumy danych (suma wszystkich bajtów lub słów bloku sumowanych danych) różni ten algorytm jego znacznie większa niezawodność wykrywania błędów. CRC jest odporny na zmianę kolejności bajtów, nie wykrywaną przez sumę. Podstawowym elementem algorytmu jest wielomian generacyjny (generator). Rozmiar sumy kontrolnej zależy od stopnia tego wielomianu.

CRC umożliwia wykrywanie szerszej klasy błędów niż sumy kontrolne. Szczególnie ważne jest wykrywanie dwóch konkretnych kategorii typowych błędów. Pierwszym z nich może

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

być awaria sprzętu często powoduje uszkadzanie określonej grupy bitów. Na przykład w urządzeniu znakowego wejścia/wyjścia może to oznaczać ustawianie pierwszych dwóch bitów każdego znaku na zero. Takie błędy są czasami nazywane błędami pionowymi**,** gdyż występują w pionowych kolumnach (gdy dane zapiszemy wierszami). Drugim CRC pozwala na szczególnie skuteczne wykrywanie błędów związanych ze zmianą wartości niewielkiej liczby bliskich bitów. Takie błędy są nazywane błędami grupowymi (ang. *burst errors).* Wykrywanie błędów grupowych jest bardzo ważne, gdyż odpowiadają one za znaczną część wszystkich błędów transmisji w sieciach komputerowych. Zakłócenia elektryczne, spowodowane na przykład przez burzę, powodują często błędy tego właśnie typu. Podobne skutki mają zakłócenia powodowane przez silniki elektryczne i inne urządzenia tego typu.

Metoda cyklicznej kontroli redundancji:

W nadajniku przekazywane dane są przetwarzane w celu wyznaczenia sekwencji bitów tworzących redundancję cykliczną CRC; sekwencja CRC jest dodawana do danych użytkowych przed ich transmisją. W odbiorniku realizowany jest taki sam algorytm dla całego bloku danych, z uwzględnieniem sekwencji kontrolnej. Wynikiem tej procedury powinna być określona z góry sekwencja (same zera), co sygnalizuje poprawną transmisję.

Najlepiej wytłumaczyć zasadę działania redundancji cyklicznej poprzez analizę odpowiednio dobranego przykładu. Dane oraz blok kontrolny tworzą sekwencję cyfr binarnych, a więc liczbę w zapisie dwójkowym. Algorytm redundancji cyklicznej działa na tych właśnie liczbach z wykorzystaniem zasad arytmetyki modulo-2. Na początku rozważań, dla większej ich przejrzystości, należy posługiwać się notacją dziesiętną (pewne rzeczy będą dzięki temu bardziej zrozumiałe).

Przykład:

Zakładając, że nadajnik zamierza przekazać sekwencję binarną 14562_{10}; algorytm redundancji cyklicznej korzysta z „klucza" równego 13. Algorytm jest realizowany w nadajniku w trzech fazach:

1) zera (dwa w tym przypadku) są dodawane do danych użytkowych - liczba dziesiętna reprezentująca dane użytkowe jest mnożona przez 100, 14562 x 100 = 1456200,
2) rezultat otrzymany w kroku (1) jest dzielony przez wartość klucza (w tym przypadku 13) w celu określenia całkowitoliczbowej reszty (r = 5),
3) wartość reszty wyznaczonej w kroku (2) jest odejmowana od wyniku kroku (1), a więc 1456200- 5 = 1456195.

Sekwencja wyznaczona w kroku 3) jest sekwencją przekazywaną, składającą się z danych użytkowych 14561 oraz dodawanej sekwencji kontrolnej 95. W odbiorniku otrzymana sekwencja 1456195 (przypadek transmisji bezbłędnej) jest dzielona przez wartość klucza nadajnika, a w wyniku otrzymujemy zero, co pozwala sekwencję 14561 + 1 uznać za prawidłową. Niezerowa

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

wartość reszty wyznaczonej w odbiorniku sygnalizuje błędy transmisji. Zauważmy, że błędy transmisji nie zostaną wykryte, jeżeli ich efektem jest liczba będąca wielokrotnością klucza, na przykład liczba 1456208 jest niepoprawna (dane odebrane to sekwencja 14563), aczkolwiek spełnia ona test dzielenia przez wartość klucza.

Rzeczywisty algorytm opiera się na arytmetyce modulo-2 oraz zapisie dwójkowym. Brak przeniesień jest zaletą arytmetyki modulo-2, co z kolei powoduje, że przekazywane dane nie są modyfikowane (w przykładzie sekwencją przekazywaną jest 14561, a nie 14562, co wynika z przeniesienia przy, odejmowaniu). Najważniejszą konsekwencją braku przeniesień w arytmetyce modulo-2 jest jednak szybkość obliczeń, ponieważ układy elektroniczne ją wykonujące mogą przetwarzać wszystkie bity w sposób równoległy, bez konieczności uwzględniania przeniesień. Jako układ elektroniczny wykorzystuje się w tym przypadku szeregowe rejestry przesuwne z bramkami XOR tworzącymi sprzężenie zwrotne pomiędzy wyjściem , a różnymi stopniami rejestru (układ taki jest również wykorzystywany do generacji pseudolosowych sekwencji binarnych). Oczywiście, nadajnik oraz odbiornik muszą pracować z tą samą wartością klucza. Znane są wartości kluczy standardowych, zapewniających optymalną pracę algorytmu; przykładowo w protokole bisync zastosowano klucz CRC-16, natomiast w protokole HDLC klucz CRC-ITU.

Wartość klucza CRC-ITU jest równa 1 0001 0000 0010 0001; jest to 17-bitowa liczba generująca 16-bitową sekwencję kontrolną, umożliwiającą : wykrywanie serii błędów o długości nieprzekraczającej 16 bitów. Standardowym zapisem takich liczb jest tak zwany wielomian generujący. Bity równe „1" są wskazywane przez odpowiednią potęgę liczby x; potęga informuje o położeniu bitu w kluczu. Klucz CRC-ITU jest określony wielomianem generującym:

$$x^{16} + x^{12} + x^5 + 1$$

Ostatnia „1" ($x^0 = 1$) w wielomianie generującym jest istotna, bowiem to ona jest odpowiedzialna za bit parzystości, co umożliwia detekcje wszystkich błędów nieparzystych. Wszystkie pozostałe bity sekwencji kontrolnej służą do wykrywania seryjnych błędów parzystych o długości nie przekraczającej 16 bitów (ograniczenie klucza).

Jako przykład redundancji cyklicznej obliczanej w dwójkowej arytmetyce modulo-2 rozważmy kontrolę parzystości, której odpowiada wielomian generujący $x^1 + 1$; binarna sekwencja nadawana jest równa 11001.

W tym celu zmodyfikujmy opisany wcześniej algorytm obliczania CRC dla zapisu dziesiętnego:

1) ponieważ długość klucza wynosi 2, sekwencja kontrolna będzie zawierać jeden bit, a więc do sekwencji danych należy dodać jedno „0" tworząc sekwencję 110010,
2) sekwencję 1 10010 dzieli się przez wartość klucza w celu wyznaczenia reszty (w tym przypadku 1) w arytmetyce modulo-2:

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

```
                        1   0   0   0   1
      ---------------------------------
1   1 ) 1   1   0   0   1   0
        1   1
        -------
            0   0
            0   0
            -----
                0   0
                0   0
                -----
                    0   1
                    0   0
                    -----
                        1   0
                        1   1
                        -----
                R=          1
```

3) odejmuje się wartość reszty od sekwencji 110010, ponieważ jednak odejmowanie modulo-2 jest równoważne dodawaniu modulo-2, to przekazywana sekwencja jest po prostu pierwotną sekwencją danych z dodaną resztą 110011.

```
1   1   0   0   1   0
                1   -
-------------------
1   1   0   0   1   1
```

W rozważanym przypadku bit kontrolny jest bitem parzystości normalnej. Nazwa redundancja cykliczna pochodzi stąd, że w wyniku cyklicznego przesunięcia bitów otrzymuje się poprawną sekwencję kodową. Stwierdzenie to jest ewidentne dla kontroli parzystości, na przykład przesuwając jeden bit w prawo otrzymuje się sekwencję 111001, nadal spełniającą test kontroli parzystości.

Algorytm wyznaczania redundancji cyklicznej występuje niekiedy w postaci zmodyfikowanej, jak na przykład ma to miejsce w protokole HDLC [ISO/IEC 3309]:

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

- szeregowy rejestr przesuwny wykorzystywany do generacji sekwencji kontrolnej FCS zawiera wyłącznie „1”;
- o przekazywanych danych dodawane jest uzupełnienie reszty do jedności;
- szeregowy rejestr przesuwny wykorzystywany do sprawdzania sekwencji kontrolnej FCS zawiera również wyłącznie „1”;
- wartość reszty sygnalizującej brak błędów transmisji jest równa 0001 1101 0000 1111.

Wielomiany do wyliczania sum kontrolnych.

Podstawowym elementem algorytmu jest wielomian generacyjny (generator). Rozmiar sumy kontrolnej zależy od stopnia tego wielomianu. Oto kilka popularnych wielomianów:

- CRC –12:

$X^{12} + X^{11} + X^3 + X^2 + X + 1$

- CRC – 16:

$X^{16} + X^{15} + X^2 + 1$

- CRC –16 REVERSE:

$X^{16} + X^{14} + X + 1$

- CRC – 32:

$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + x + 1$

(wielomian ten stosowany jest w sieciach lokalnych Ethernet oraz Token Ring.)

- SDLC (IBM,CCITT):

$X^{16} + X^{12} + X^5 + 1$

- SDLC REVERSE:

$X^{16} + X^{11} + X^4 + 1$

- CRC- ITU:

$X^{16} + X^{12} + X^5 + 1$

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

- Wielomian stosowany w sieciach ATM oraz DQDB:

$X^8 + X^2 + X + 1$

Podane klucze są w stanie wykrywać błędy seryjne o długości nieprzekraczającej długości sekwencji kontrolnej. Klucze te umożliwiają wszystkich błędów podwójnych, niekoniecznie seryjnych oraz także wielu błędów seryjnych o długości przekraczającej długość gwarantowaną.

O skuteczności stosowania powyższych wielomianów przekonuje prawdopodobieństwo wykrycia błędu przy użyciu CRC-16:

- 100 proc. w przypadku błędów pojedynczych, podwójnych, seryjnych (seria nie dłuższa niż 16 bitów), z nieparzystą liczbą bitów,
- 99,997 proc. dla błędów seryjnych 17-bitowych,
- 99,998 proc. dla błędów seryjnych 18-bitowych i dłuższych.

Dla kodu CRC -32 wartości te są następujące :

- wszystkie błędy seryjne, o długości nie przekraczającej 32 bitów są wykrywane;
- prawdopodobieństwo nie wykrycia błędu seryjnego 33-bitowego wynosi 2^{-31}
- prawdopodobieństwo nie wykrycia błędu seryjnego 34-bitowego lub dłuższego wynosi 2^{-32}.

Sprzętowa implementacja wykrywania błędów za pomocą CRC.

Sprzęt do wyliczania CRC ma dwa proste bloki funkcjonalne:

- rejestr przesuwający;
- bramka XOR (alternatywa wykluczająca);

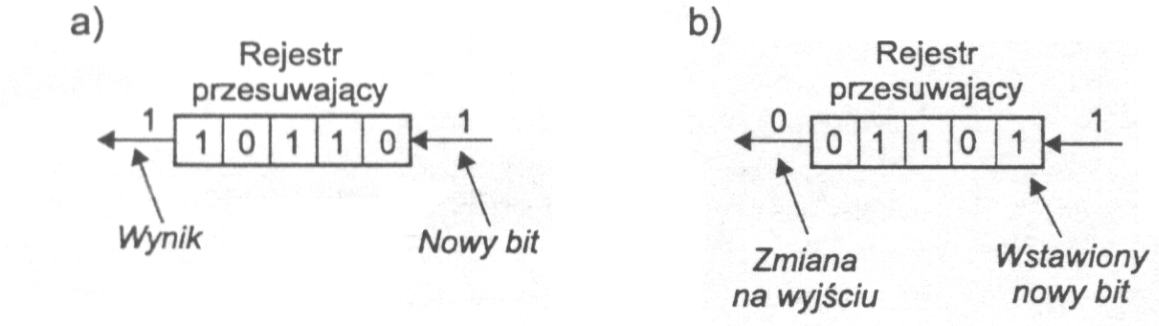
Wyliczanie funkcji XOR

A	B	Wynik
0	0	0
0	1	1
1	0	1
1	1	0

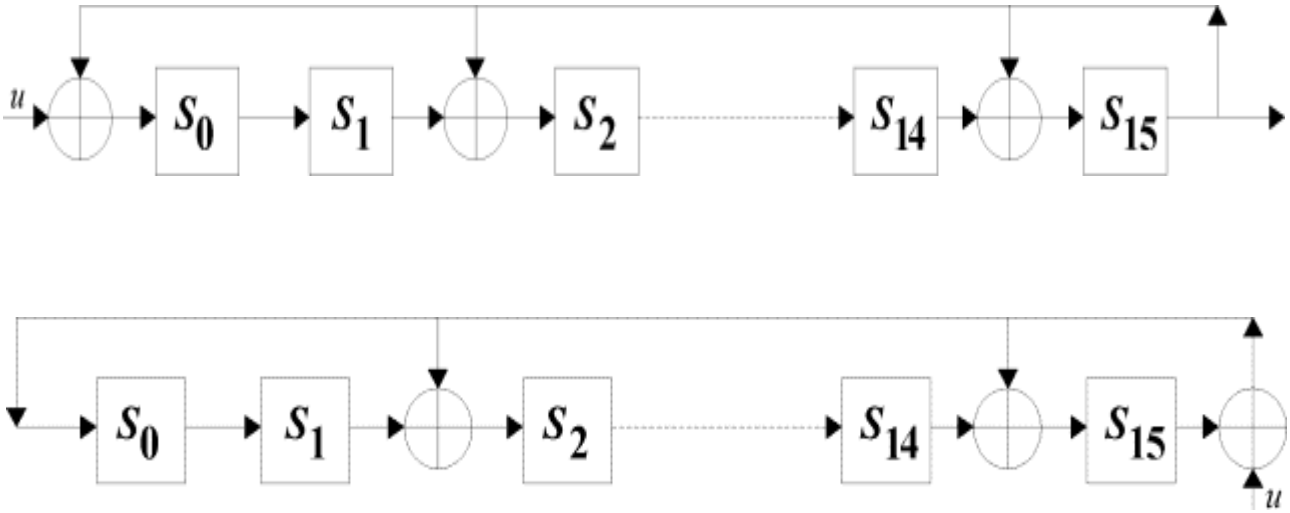
Drugie urządzenie potrzebne do wyliczania CRC to rejestr przesuwający. Bity są w nim przesuwane po kolei od prawej do lewej. Rejestr taki przechowuje określoną liczbę bitów (np. 16). Gdy jest pełny to wstawienie do niego kolejnego bitu zmusza do usunięcia jednego z dotychczasowych. Zakłada się, że na wyjściu rejestru przesuwającego pojawia się wartość skrajnie lewego bitu. Gdy ten bit się zmienia, wartość na wyjściu też się zmienia.

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

Rejestr przesuwający umożliwia wykonywanie dwóch operacji: inicjowania i przesunięcia. Inicjowanie polega na ustawieniu wszystkich bitów w rejestrze na „0”. Na wyjściu rejestru jest wtedy „0”. Przesunięcie polega na natychmiastowym przesunięciu wszystkich bitów w rejestrze o jeden w lewo, wstawiając na skrajnie prawej pozycji bitu z wejścia i ustawienie wyjścia na wartość skrajnie lewego bitu.



Rys. 1. Rysunek przedstawia rejestr przesuwający przed (a) i po (b) operacji przesunięcia.



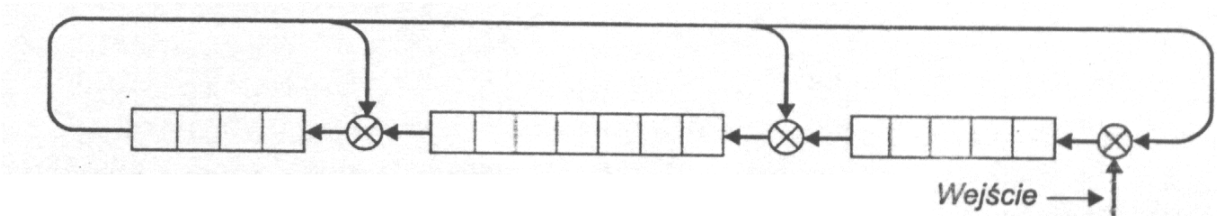
Rys. 2 Sprzętowe liczenie CRC

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

Najpopularniejsze są dwa schematy sprzętowego liczenia CRC, które są przedstawione na rys.2. Pierwszy z prezentowanych "układ z opóźnieniem" symuluje dzielenie pisemne danych (dla których liczy się sumę kontrolną) przez wielomian generacyjny. Na rysunku 1 dzielnikiem jest kod CRC-16. operacje wykonywane według schematu z Rys. 1. Kolejne pozycje dwójkowe danych (dzielnej) umieszczane są w najmłodszej pozycji rejestru, a cały rejestr jest cyklicznie przesuwany w kierunku jego najstarszej pozycji; bit opuszczający rejestr przesuwu z najstarszej pozycji ma wpływ na wykonanie operacji XOR na rejestrze: jeżeli ma wartość 1, wówczas operacja ta jest wykonywana na bitach zerowym, drugim, piętnastym i szesnastym, który opuścił już rejestr; jeżeli opisany bit ma wartość 0, operacja XOR jest pomijana. W obydwu przypadkach wraca się na początek algorytmu. Po wprowadzeniu do rejestru przesuwu ostatniego bitu danych, wykonuje się powyższe operacje tyle razy, ile wynosi stopień wielomianu generacyjnego (w tym przypadku 16), z tym, że teraz umieszcza się w najmłodszej pozycji wartość 0 - jest to równoznaczne z wydłużeniem danych. Po wykonaniu tych operacji wynik CRC znajdzie się w rejestrze przesuwu. W przypadku pominięcia etapu końcowego, ostatnich 16 bitów danych będzie "zaszytych" wewnątrz CRC, co w rezultacie oznacza, że liczenie nie zostało dokończone. Tytułowe opóźnienie w pierwszym schemacie wynika właśnie z konieczności wydłużenia danych.

Drugi schemat z Rys. 2 prezentuje nieco zmodernizowaną metodę, dającą oczywiście taki sam wynik końcowy, jak w wypadku zastosowania algorytmu pierwszego. Różnica polega na wykonaniu operacji XOR pomiędzy wprowadzanym bitem a najstarszą pozycją rejestru przesuwu. Jeżeli wynikiem tej operacji jest wartość 1, to przesuwa się dane w rejestrze i wykonuje operację XOR, w przeciwnym wypadku ograniczamy się wyłącznie do przesunięcia.

Kolejny rysunek (Rys.3) pokazuje, w jaki sposób można połączyć trzy rejestry przesuwające i trzy bramki XOR, aby wyliczyć 16 – bitową wartość CRC:



Rys. 3 Przedstawiony schemat odpowiada wielomianowi $X^{16} + X^{12} + X^5 + 1$

Z rysunku widać, że potrzebny układ składa się z trzech rejestrów przesuwających połączonych bramkami XOR. Dane z wyjścia pierwszej od lewej bramki XOR wędrują jednocześnie: na wejścia dwóch pozostałych bramek XOR oraz na wejście pierwszego rejestru przesuwającego. Na powyższym rysunku rejestry przesuwające są 4-, 7- i 5-bitowe. W celu

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

wyliczenia CRC wszystkie rejestry inicjuje się bitami zerowymi, a następnie wypełnia bitami komunikatu (przez powtarzanie operacji przesunięcia). Znaczy to, że każdy kolejny bit komunikatu jest podawany na wejście pierwszej od lewej bramki XOR (w miejscu oznaczonym jako wejście), a następnie wszystkie rejestry przesuwające otrzymują rozkaz przesunięcia. Ten proces jest powtarzany do chwili wstawienia do układu wszystkich bitów komunikatu wejściowego. Po zakończeniu tej operacji rejestry przesuwające zawierają 16-bitową wartość CRC danych wejściowych. Odbiorca używa identycznego układu do wyliczenia CRC odebranego komunikatu, po czym porównuje wyliczoną wartość z CRC przysłanym przez nadawcę.

Aby uprościć proces kontroli CRC, standardowe algorytmy obliczania CRC różnią się nieco od opisanego powyżej - przy wyliczaniu CRC nadawca dodaje na końcu komunikatu dodatkowe 16 bitów o wartości zero. Dodanie tych bitów powoduje, że uzyskujemy wartość odwrotną. Odbiorca zamiast wyliczania CRC i porównywania z oryginalną, która przybyła, oblicza CRC komunikatu wraz z wartością CRC dla niego. Jeśli wszystkie bity odebrano poprawnie, to wynikiem jest zero. Zaleta opisanej techniki polega na możliwości szybkiego sprzętowego porównania 16-bitowej liczby z zerem.

Intuicyjnie da się zauważyć dwa powody dla których CRC umożliwia wykrycie większej liczby błędów niż zwykła metoda sum kontrolnych. Po pierwsze wartość wejściowa jest przesuwana przez wszystkie trzy rejestry przesuwające, zmiana jednego bitu danych powoduje więc dużą zmianę wyliczonej wartości CRC. Po drugie w układzie zastosowano sprzężenie zwrotne, w którym wartość na wyjściu pierwszego od lewej strony rejestru przesuwającego wpływa na wynik na wynik na wyjściu pierwszej od lewej strony bramki XOR, wartość związana z pojedynczym bitem danych wejściowych jest więc przesuwana przez rejestry więcej niż raz.

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

Funkcje liczące CRC.

Funkcje liczącą CRC według algorytmu „z opóźnieniem”:

```
#define CRC_16      0x8005u
unsigned licz_crc(unsigned char *blok, unsigned dlugosc)
{       register unsigned long a=((unsigned long)*(char *)blok < 24) +
        ((unsigned long)*(((char *)blok)+1) < 16);
        unsigned i=2;
        register char j=8
        while(i<dlugosc+2)
        {       (unsigned)a= i<dlugosc ? (unsigned)blok[i] < 8 : 0;
                while(j--)
                {       if(a & 0x80lu<24)
                        {       a<=1;
                                a^= CRC_16<16lu;
                        }
                        else
                        a<=1;
                }
                i++;
        }
        return (unsigned)(a >> 16);
}
```

Widać na nim pewną modernizację polegającą na wprowadzeniu pierwszych dwóch bajtów danych bezpośrednio do rejestru przesuwu, oszczędza to owych dodatkowych cykli. Mankamentem tego rozwiązania jest możliwość liczenia CRC dla bloków o długości co najmniej dwóch lub więcej bajtów oraz problemy z liczeniem sumy kontrolnej dla danych, które są dzielone na części.

Kolejne funkcje liczące sumy kontrolne 16- i 32-bitowe dla dowolnego generatora. Funkcje kontynuuj_crc i kontynuuj_crc32 służą do kontynuacji liczenia CRC dla danych, które z racji swoich rozmiarów muszą być podzielone na fragmenty. W takim przypadku sumę kontrolną pierwszego bloku danych liczy się za pomocą funkcji licz_crc lub licz_crc32, a wynik jego CRC umieszczamy jako pierwszy parametr wywołania odpowiedniej funkcji kontynuacyjnej. Z kolei jej wynik przekazuje się w kolejnym wywołaniu. Operacje te powtarzamy do momentu osiągnięcia końca danych. Zastosowanie schematu drugiego może wymagać dodatkowego uzasadnienia. Wynika ono z konieczności wydłużania danych na końcu o pozycje zerowe w algorytmie pierwszym, co utrudnia jego użycie do operowania na danych dzielonych na fragmenty. Aby zastosować schemat pierwszy, CRC pierwszego bloku danych musi być policzony bez wydłużania tego bloku o pozycje zerowe; dotyczy to również bloków znajdujących

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

się wewnątrz danych - dopiero w ostatnim fragmencie dopisuje się na jego końcu zera. Należy zatem napisać trzy funkcje.

```
/* Definicje wielomianów generacyjnych dla CRC 16 i 32 bit. */
#define crc_16       0x8005u
#define crc_16_rev   0x4003u
#define crc_SDLC     0x1021u  /* ( IBM, CCITT ) */
#define crc_SDLC_rev 0x0811u
#define crc_32       0x04c11db7lu  /* ( ETHERNET ) */
#define CRC_32 crc_32  /* Aktualnie używane generatory */
#define CRC_16 crc_16
unsigned licz_crc(unsigned char *blok, unsigned dlugosc)
{           register unsigned a=0, b, i=0;
            register char j;
            while(i<dlugosc)
            {           b=blok[i] < 8;
                        for(j=8;j--;)
                        {           if((a ^^ b) & 0x8000)
                                    {           a<=1;
                                                b<=1;
                                                a^^= CRC_16;
                                    }
                                    else
                                    {           a<=1;
                                                b<=1;
                                    }
                        }
                        i++;
            }
            return a;
}
```


Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

```
unsigned long licz_crc32(unsigned char *blok,unsigned dlugosc)
{       register unsigned long a=0;
        register unsigned char b, j;
        register unsigned i=0;
        while(i<dlugosc)
        {       b=blok[i];
                for(j=8;j--;)
                {       if((a & 0x80lu<24)>0 ^^ (b & 0x80)>0)
                        {       a<=1;
                                b<=1;
                                a^^= crc_32;
                        }
                        else
                        {       a<=1;
                                b<=1;
                        }
                }
                i++;
        }
        return a;
}
```

```
unsigned long kontynuuj_crc32(unsigned long crc,
unsigned char *blok, unsigned dlugosc)
{       register unsigned long a=crc;
        register unsigned char b,j;
        register unsigned i=0;
        while(i<dlugosc)
        {       b=blok[i];
                for(j=8;j--;)
                {       if((a & 0x80lu<24)>0 ^^ (b & 0x80)>0)
                        {       a<=1;
                                b<=1;
                                a^^= crc_32;
                        }
                        else
                        {       a<=1;
                                b<=1;
                        }
                }
                i++;
        }
        return a;
}
```


Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

Program posługujący się powyższymi funkcjami. Liczy CRC dla dowolnego pliku:

```
#include <stdio.h>
#include <dir.h>
#include <io.h>
#include <fcntl.h>
#include "wydruk2.c"

unsigned char bufor[20000];
main()
{       char wejscie[60];
        struct ffblk ffblk;
        int plik, bajtow_wczytanych;
        unsigned crc16=0;
        unsigned long crc32=0;
        printf("Podaj nazwe pliku wejsciowego :\n");
        scanf("%60s",wejscie);
        if(findfirst(wejscie,&ffblk,0))
        {       printf("\nPlik o podanej nazwie nie istnieje\n");
                return;
        }
        if((plik=open(wejscie,O_BINARY | O_RDONLY)) == -1)
        {       printf("Wystapil blad przy otwieraniu pliku\n");
                return;
        }
        if(bajtow_wczytanych=read(plik,bufor,20000))
        {       crc16=licz_crc(bufor,bajtow_wczytanych);
                crc32=licz_crc32(bufor, bajtow_wczytanych); }
                while(bajtow_wczytanych=read(plik,bufor,20000))
                {       crc16=kontynuuj_crc(crc16,bufor,bajtow_wczytanych);
                        crc32=kontynuuj_crc32(crc32,bufor,bajtow_wczytanych);
                }
                close(plik);
                printf("Wartosci CRC dla podanego pliku sa nastepujace :\n");
                printf("CRC-16 - %x\n",crc16);
                printf("CRC-32 - %lx\n",crc32);
                }
        }
return;
}
```


Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01

Polecania do realizacji na laboratorium.

Stworzyć program testujący i porównujący trzy algorytmy korekcji błędów sygnałów cyfrowych przesyłanych w sieciach teleinformatycznych.

Funkcje programu:

a) Możliwość podglądu sygnału wejściowego, nadmiarowego (dane kontrolne), zakłóconego, wyjściowego z poprawionymi i wykrytymi błędami (oznaczenie błędów), sygnał wyjściowy bez danych kontrolnych.
b) Możliwość ręcznego zakłócania poprawności sygnału wejściowego.
c) Generacja sygnału wejściowego (symulujący przesyłanie danych)
d) Generacja zakłóceń
e) Obliczanie błędów wykrytych, skorygowanych, oraz nie wykrytych
f) Wyliczenie ilości przesyłanych danych rzeczywistych i informacji nadmiarowych

W programie wykorzystać:

cykliczna kontrola redundancji (CRC): CRC 12, CRC 16, CRC 16 REVERSE, CRC 32, SDLC, SDLC REVERSE, CRC-ITU, ATM

kodowanie Hamminga,

kontorla parzystości

LITERATURA

1. Sieci LAN, MAN i WAN – protokoły komunikacyjne Józef Woźniak
2. Wprowadzenie do transmisji danych Andrew Simmonds
3. Sieci komputerowe i intersieci P. Comer

Projekt Absolwent na miarę czasu współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego;
Nr umowy UDA-POKL.04.01.01-00-421/10-01