

Cyclic Redundancy Check



- Cyclic Redundancy Check (CRC)
 - Error detection technique using a polynomial to verify data is not corrupted
 - Used in PCjr cartridges among many other places
 - Different algorithms
 - Direct
 - non-direct
 - Varying initial values
 - Reflected, Reversed (not covered here)
 - No error correction
- Reference:
 - A Painless Guide to CRC Error Detection Algorithms
 - Ross N. Williams, 19 August 1993
 - Search crc_v3.txt
 - CRC calculation (breitbandkatze.de)
 - <http://www.zorc.breitbandkatze.de/crc.html>
 - PCjr technical reference
 - A-106 (BIOS CRC routine for cartridges)

CRC Polynomial



- Defines how the particular CRC algorithm is calculated
- Not used the same way as “normal” polynomial arithmetic
- CRC uses polynomials and exclusive or (XOR, \oplus)
 - Removes the need to carry when “adding” polynomials
 - In this polynomial math, addition and subtraction give the same result
 - See “A Painless Guide to CRC Error Detection Algorithms” for more details

Polynomial Examples



- 4 bit CRC (1 nibble)
 - $x^4 + x + 1$, or
 - $x^4 + x + 1$
 - CRC order is 4
 - CRC representation:
 - 1 0011 – binary, or 0011
 - 0x1 0x3 – hex, or 0x3
 - Leading bit is often left off since it must be 1.
- 16 bit CRC (2 bytes)
 - $x^{16} + x^{12} + x^5 + 1$, or
 - $x^{16} + x^{12} + x^5 + 1$
 - CRC order is 16
 - CRC representation:
 - 0001 0000 0010 0001
 - 0x1021

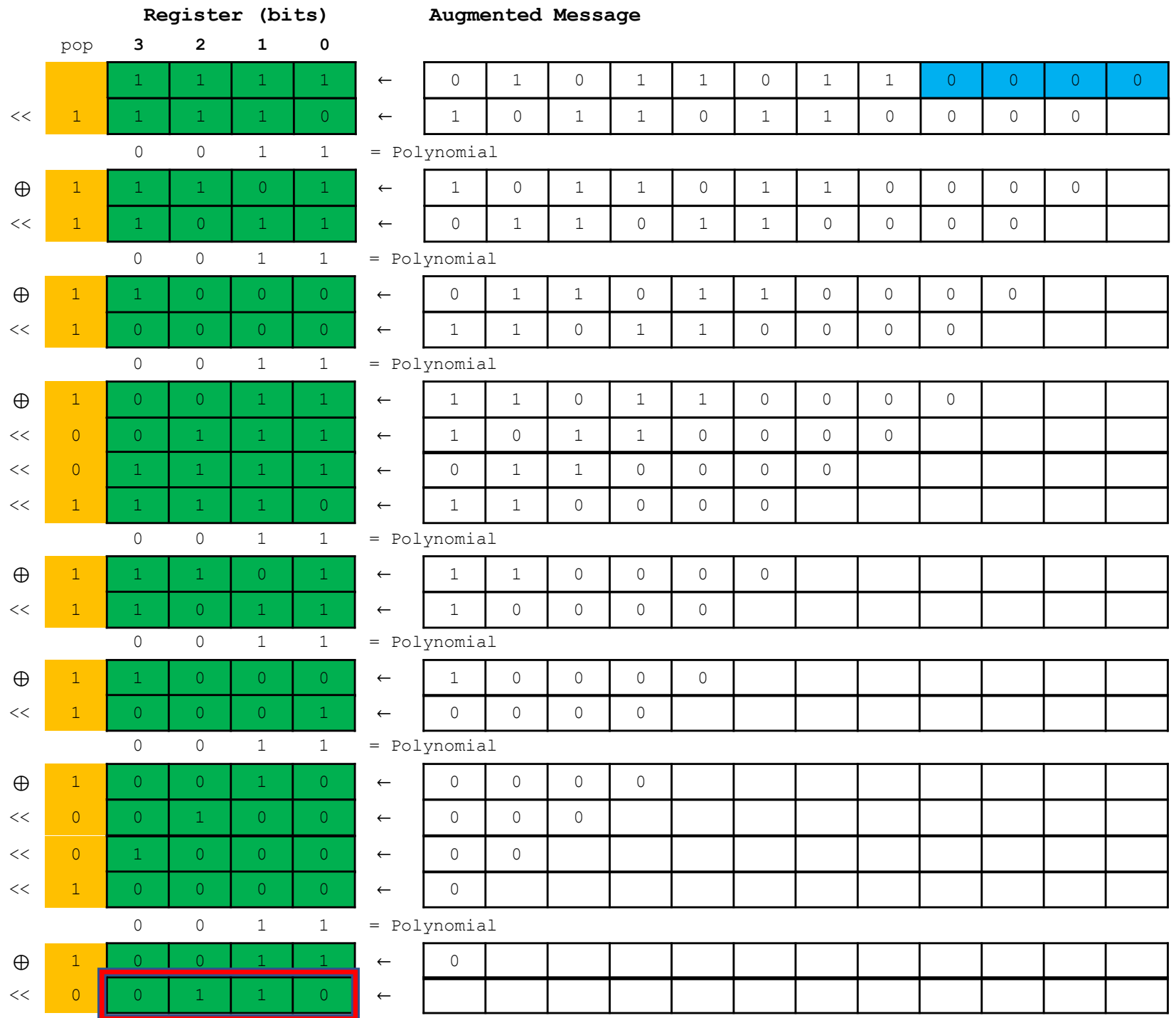


```

Poly = 10011 ) 1110 0111 0010 = Quotient
                1111 0101 1011 0000 = Augmented message
                1001 1
                -----
                110 11
                100 11
                -----
                10 000
                10 011
                -----
                0 0111
                0 0000
                -----
                0111 1
                0000 0
                -----
                111 10
                100 11
                -----
                11 011
                10 011
                -----
                1 0001
                1 0011
                -----
                0010 0
                0000 0
                -----
                010 00
                000 00
                -----
                10 000
                10 011
                -----
                0 0110
                0 0000
                -----
                0110 = CRC (Remainder)
```

Example:

- Non-direct method (using augmentation)
- CRC-4-ITU
 - 4-bit CRC ($x^4 + x + 1$)
 - Polynomial $0x13$ (1 0011), width 4
 - Message: 0101 1011 (ASCII '[')
 - Augmented message: 0101 1011 0000
 - Initial Value: $0xF$ (1111)



Straightforward CRC implementation
(AKA non-direct)

Same algorithm as previous slide

Initial value 1111 (0xF)
augmentation bits 0000 (0x0)

4 bit register

Message byte = 01011011 (ASCII '[')

Method:

1. Shift bits left by one
2. If popped bit is 1, xor (\oplus) register with polynomial
3. Continue until augmented message is processed
4. Continue with next message byte as appropriate

Final register is CRC



Table-Driven Method

- For a given byte [0,255] we can precompute a CRC table value
- Loop through all byte values [0,255] and compute a table of 256 values of CRC values

- So for a 16-bit polynomial, this is:

```
for (i = 0 ; i < 256 ; ++i )
{
    crc[i] = i;
    crc[i] <<= 8;

    for (j = 0 ; j < 8 ; ++j ) // iterate bit by bit
    {
        bool highbit = crc[i] && 0x80; // 0x80 = 1000 0000
        crc[i] <<= 1;
        if (highbit)
        {
            crc[i] ^= polynomial;
        }
    }
}
```

- Same calculation as before, only here we look to store the precomputed CRCs for all possible messages ahead of time.

Table-Driven Method

- E.g. compute table entry for 91
- polynomial = 0001 0000 0010 0001 = 0x1021 (16-bit poly)
- index = 0101 1011 (0x5B, 91)
- table[91]:

pop

crctable @ index 91 [0x5B] 01011011

	0	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0
<< 0	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0
<< 1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0
⊕	0	1	1	1	1	1	0	0	0	0	1	0	0	0	0	1
<< 0	1	1	1	1	1	0	0	0	0	1	0	0	0	0	1	0
<< 1	1	1	1	1	0	0	0	0	1	0	0	0	0	1	0	0
⊕	1	1	1	0	0	0	0	0	1	0	1	0	0	1	0	1
<< 1	1	1	0	0	0	0	0	1	0	1	0	0	1	0	1	0
⊕	1	1	0	1	0	0	0	1	0	1	1	0	1	0	1	1
<< 1	1	0	1	0	0	0	1	0	1	1	0	1	0	1	1	0
⊕	1	0	1	1	0	0	1	0	1	1	1	1	0	1	1	1
<< 1	0	1	1	0	0	1	0	1	1	1	1	0	1	1	1	0
⊕	0	1	1	1	0	1	0	1	1	1	0	0	1	1	1	1
<< 0	1	1	1	0	1	0	1	1	1	0	0	1	1	1	1	0

crctable[0x5B] = 1110 1011 1001 1110 (i.e. 0xEB9E)





Table-Driven Method (non-direct)

- Number of augmentation bytes = bytes of poly (e.g. 2 for a 16 bit poly)
- Working 1 byte at a time now instead of 1 bit << 8
- Method:
 1. Shift register left one byte
 2. Used rotated out value as index of table lookup
 3. Xor table value into register
 4. Continue

		Register (bytes)		Augmented Message		
		1	0			
pop						
<<	1111 1111 0xFF	1111 1111 0xFF	1111 1111 0xFF	← 0101 1011 0x5B	0000 0000 0x00	0000 0000 0x00
		1111 1111 0xFF	0101 1011 0x5B	← 0000 0000 0x00	0000 0000 0x00	
		0001 1110 0x1E	1111 0000 0xF0	= table[0xFF]		
⊕	1110 0001 0xE1	1110 0001 0xE1	1010 1011 0xAB	← 0000 0000 0x00	0000 0000 0x00	
		1010 1011 0xAB	0000 0000 0x00	← 0000 0000 0x00		
		1110 1101 0xED	0000 1111 0x0F	= table[0xE1]		
⊕	0100 0110 0x46	0100 0110 0x46	0000 1111 0x0F	← 0000 0000 0x00		
		0000 1111 0x0F	0000 0000 0x00	←		
		0010 1000 0x28	0000 0010 0x02	= table[0x46]		
⊕		0010 0111 0x27	0000 0010 0x02	←		

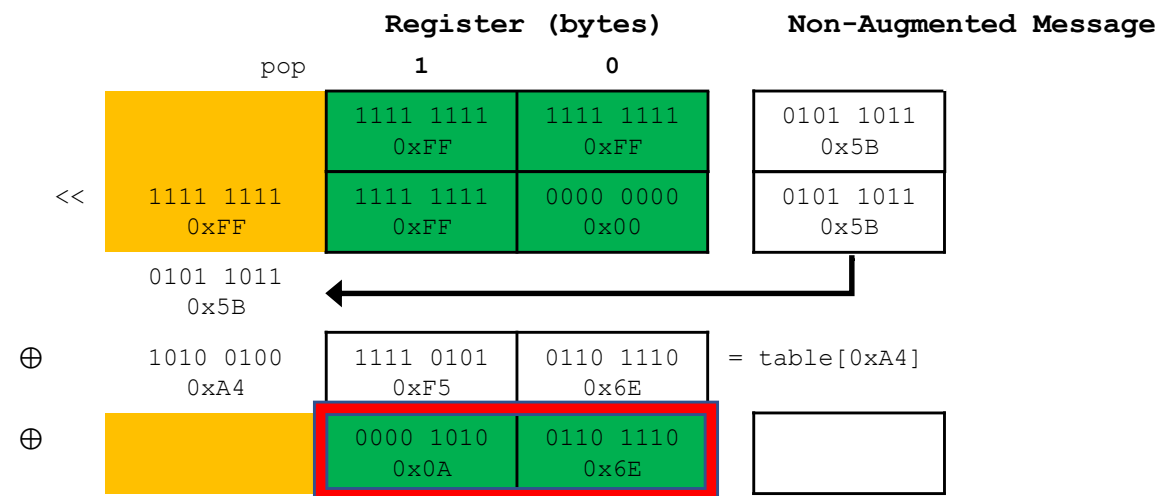
Direct Table Algorithm



- AKA A Slightly Mangled Table-Driven Implementation
 - A Painless Guide to CRC Error Detection Algorithms
- No augmentation bytes are used



- Shift register left one byte (no augmentation) (direct)
 1. Xor rotated out value with next message byte
 2. Use above for table lookup
 3. Xor table value into register
 4. Continue
- Example:

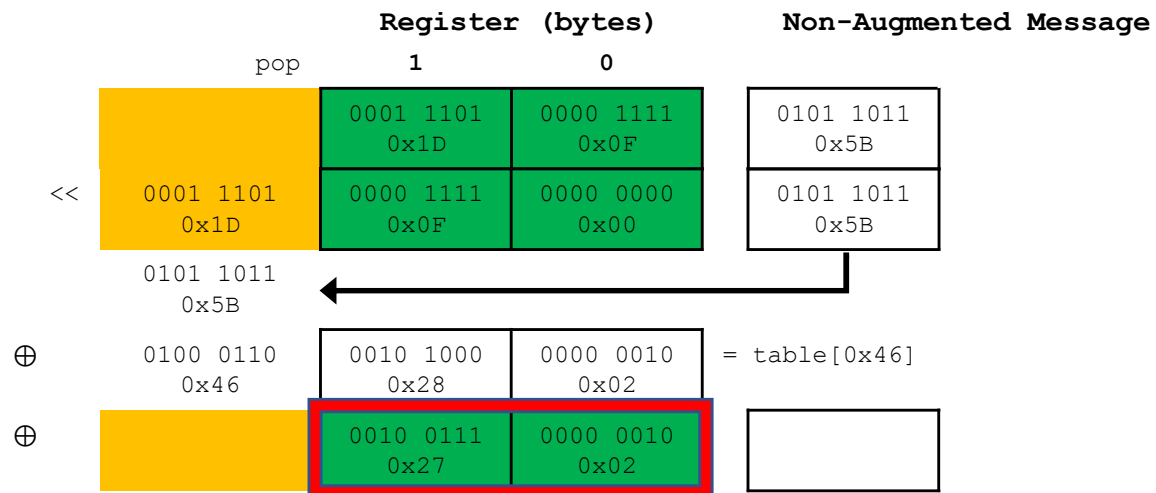


But wait!
non-direct method gave us:

0010 0111 0x27	0000 0010 0x02
-------------------	-------------------



- What if we change the initial value for the direct method?



- We can get the same CRC result as the non-direct method without using augmentation!

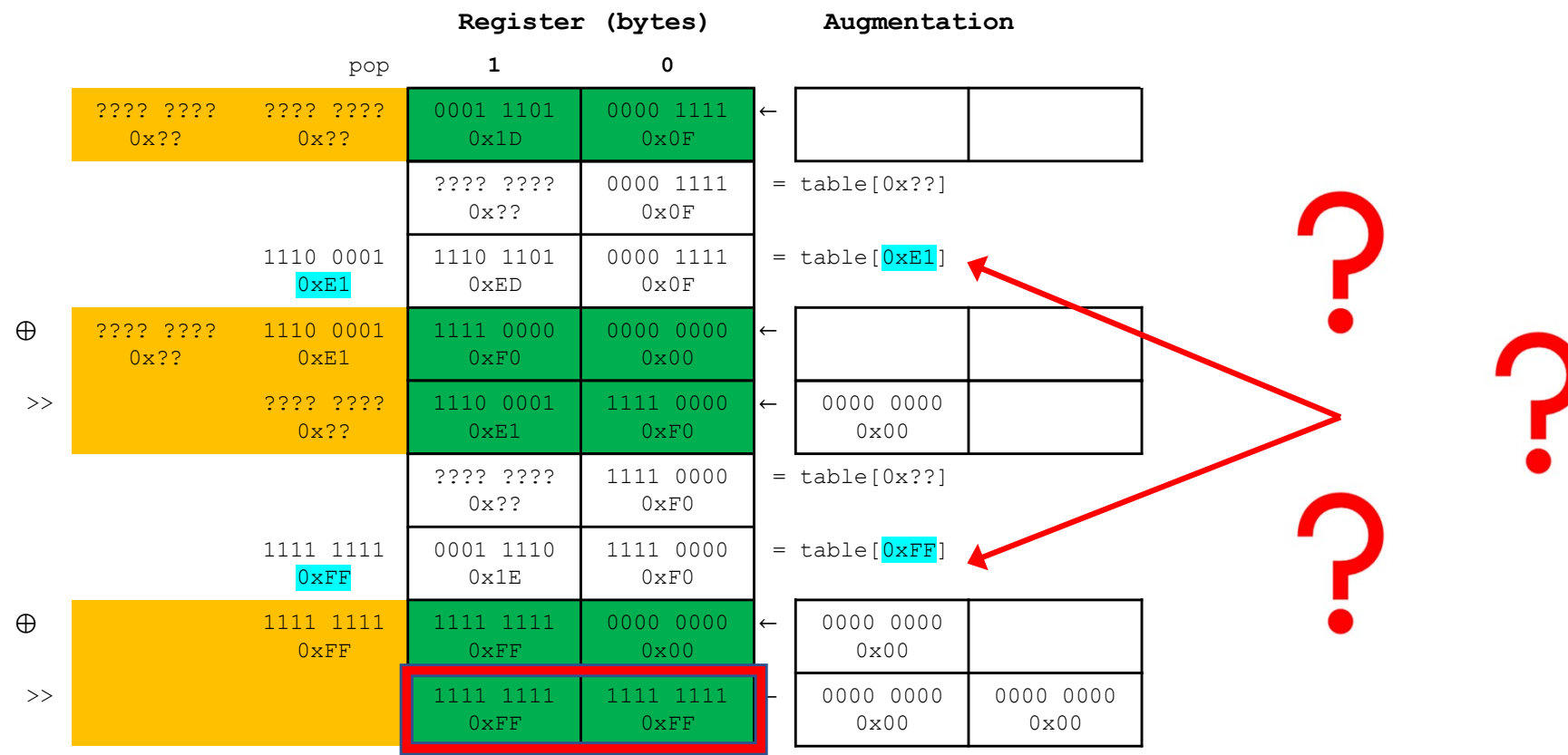


- Determining equivalent direct initial value based on the non-direct initial value
 - Using same polynomial for both methods
 - 1. Perform the non-direct Table method on the non-direct initial value

		Register (bytes)		Augmentation	
		1	0		
<<	pop 1111 1111 0xFF	1111 1111 0xFF	1111 1111 0xFF	← 0000 0000 0x00	0000 0000 0x00
		1111 1111 0xFF	0000 0000 0x00	← 0000 0000 0x00	
		0001 1110 0x1E	1111 0000 0xF0	= table[0xFF]	
⊕	<< 1110 0001 0xE1	1110 0001 0xE1	1111 0000 0xF0	← 0000 0000 0x00	
		1111 0000 0xF0	0000 0000 0x00	←	
		1110 1101 0xED	0000 1111 0x0F	= table[0xE1]	
⊕		0001 1101 0x1D	0000 1111 0x0F		



- Determining equivalent non-direct initial value based on the direct initial value
 - Using same polynomial for both methods
 - 1. Perform a reversed non-direct method on the direct initial value
 - 1. Reverse calculations to determine what would give us the 0x00 augmentation bytes



Finding a table entry by the last 8 bits only



- XOR = \oplus
- $\text{table}[x \oplus y] = \text{table}[x] \oplus \text{table}[y]$
 - If x and y have the same length
- polynomial = 0001 0000 0010 0001 = 0x1021 (16-bit poly) [IMPORTANT: this example is specific to this polynomial!]

index =

A	B	C	D	E	F	G	H
---	---	---	---	---	---	---	---

- $\text{table}[\text{index}] = [\text{A} \ \& \ \text{table}[128]] \oplus [\text{B} \ \& \ \text{table}[64]] \oplus [\text{C} \ \& \ \text{table}[32]] \oplus [\text{D} \ \& \ \text{table}[16]] \oplus$
 $[\text{E} \ \& \ \text{table}[8]] \oplus [\text{F} \ \& \ \text{table}[4]] \oplus [\text{G} \ \& \ \text{table}[2]] \oplus [\text{H} \ \& \ \text{table}[1]]$

H & table[1] =

0	0	0	H	0	0	0	0	0	0	H	0	0	0	0	H
0	0	G	0	0	0	0	0	0	G	0	0	0	0	G	0
0	F	0	0	0	0	0	0	F	0	0	0	0	F	0	0
E	0	0	0	0	0	0	E	0	0	0	0	E	0	0	0
0	0	0	D	0	0	D	0	0	0	D	D	0	0	0	D
0	0	C	0	0	C	0	0	0	C	C	0	0	0	C	0
0	B	0	0	B	0	0	0	B	B	0	0	0	B	0	0
A	0	0	A	0	0	0	A	A	0	0	0	A	0	0	0

= H & poly

G & table[2] =

= G & poly<<1

F & table[4] =

= F & poly<<2

E & table[8] =

= E & poly<<3

D & table[16] =

= D & (poly<<4 \oplus poly)

C & table[32]

= C & (poly<<4 \oplus poly)<<1

B & table[64]

= B & (poly<<4 \oplus poly)<<2

A & table[128]

= A & (poly<<4 \oplus poly)<<3

table[index]=

0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H
E	F	G	H	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	A	B	C	D
0	0	0	0	0	0	0	A	B	C	D	0	0	0	0	0
A	B	C	D	0	0	0	0	0	0	0	0	0	0	0	0

\oplus

\oplus

\oplus

\oplus

\oplus

Finding a table entry by the last 8 bits only



table[index]=

0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0	⊕
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H	⊕
E	F	G	H	0	0	0	0	0	0	0	0	0	0	0	0	⊕
0	0	0	0	0	0	0	0	0	0	0	0	A	B	C	D	⊕
0	0	0	0	0	0	0	A	B	C	D	0	0	0	0	0	⊕
A	B	C	D	0	0	0	0	0	0	0	0	0	0	0	0	

So if we need a table entry ending in, e.g. 0000 1111 (0x0F) as shown in an earlier example (again assuming polynomial is 0x1021)

?	?	?	?	?	?	?	?	0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Therefore the following must be true

$D = 0$

$H \oplus D = H \oplus 0 = 1$, therefore $H = 1$

$H \oplus C \oplus D = 1 \oplus C \oplus 0 = 0$ therefore $C = 1$

$G = 0$

$B = 1$

$F = 0$

$A = 1$

$E = 0$

So the table index which satisfies this is ABCDEFGH = 11100001 = 0xE1

1	1	1	0	1	1	0	1	0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

NOTE! We have enough information to compute the table entry without needing to know what the index into the table was.

The table entry is unique for each combination of the final 8 bits. Only one solution exists!

What does the PCjr do?



- PCjr
 - CRC-16 (CRC-CCITT)
 - Used by the PCjr cartridge CRC routine
 - Cartridge routine is looking for CRC of cartridge data to result in 0x0000
 - PCjr Technical Reference page A-106
 - Polynomial
 - $x^{16} + x^{12} + x^5 + 1$ (0x1021 – 0001 0000 0010 0001)
 - Initial Value: 0xFFFF
 - Direct method



PCjr implementation in BIOS

A-106: CRC_CHECK (0xFE71)

Address	Label	Instruction	Comment
FE71	CRC_CHECK	PROC NEAR	
		ASSUME DS:NOTHING	
FE71		MOV BX,CX	Save Count
FE73		MOV DX,0FFFFH	Init Encode Register
FE76		CLD	Set Dir flag to increment
FE77		XOR AH,AH	Init work reg high
FE79		MOV CL,4	Set rotate count
FE7B	CRC_1	LODSB	Get a byte
FE7C		XOR DH,AL	Form Aj + Cj + 1
FE7E		MOV AL,DH	
FE80		ROL AX,CL	Shift work reg back 4
FE82		XOR DX,AX	Add into result reg
FE84		ROL AX,1	Shift work reg back 1
FE86		XCHG DH,DL	Swap partial sum into result reg
FE88		XOR DX,AX	Add work reg into results
FE8A		ROR AX,CL	Shift work reg over 4
FE8C		AND AL,11100000B	Clear off (EFGH)
FE8E		XOR DX,AX	Add (ABCD) into results
FE90		ROR AX,1	Shift work reg on over (AH = 0 for next pass)
FE92		XOR DH,AL	Add (ABCD into results low)
FE94		DEC BX	Decrement count
FE95	JNZ CRC_1		Loop till count = 0000
FE97		OR DX,DX	DX S/B = 0000 if O.K.
FE99		RET	Return to caller
FE9A	CRC_CHECK	ENDP	

```
Union Register
{
    uint16_t x; // ax,bx,etc.
    struct
    {
        uint8_t l; // al,bl,etc.
        uint8_t h; // ah,bh, etc.
    };
};
```

```
Register cx = number_of_bytes;
char cartridge_data[number_of_bytes];

Register bx = cx; // MOV BX,CX
Register dx = 0xFFFF; // MOV DX, 0FFFFH
Register ax;
ax.h = 0; // XOR AH,AH

for (; bx.x < cx.x ; --bx.x ) // DEC BX, JNZ CRC_1
{
    ax.l = cartridge_data[i]; // LODSB
    dx.h ^= ax.l; // XOR DH,AL
    ax.l = dx.h; // MOV AL,DH
    ax.x = ax.x << 4; // ROL AX,CL (CL=4)
    dx.x ^= ax.x; // XOR DX,AX
    ax.x = ax.x << 1; // ROL AX,1
    std::swap(dx.h,dx.l); // XCHG DH,DL
    dx.x ^= ax.x; // XOR DX,AX
    ax.x = ax.x >> 4; // ROR AX,CL (CL=4)
    ax.l &= 0xE0; // AND AL,11100000
    dx.x ^= ax.x; // XOR DX,AX
    ax.x = ax.x >> 1; // ROR AX,1
    dx.h ^= ax.l; // XOR DH,AL
}

// OR DX,DX - sets appropriate flags based on value of DX.
```

What is the jr CRC routine doing?



- No table?
- Only loops once for each byte
- At first, it does not look like any of the methods discussed
- PCjr is performing direct table method
 - No table is created
 - saves memory having to precompute 256 x 2 byte table values at the expense of compute time

Address	Label	Instruction	Comment
FE71	CRC_CHECK	PROC NEAR	
		ASSUME DS:NOTHING	
FE71		MOV BX,CX	Save Count
FE73		MOV DX,0FFFFH	Init Encode Register
FE76		CLD	Set Dir flag to increment
FE77		XOR AH,AH	Init work reg high
FE79		MOV CL,4	Set rotate count
FE7B	CRC_1	LODSB	Get a byte
FE7C		XOR DH,AL	Form $A_j + C_j + 1$
FE7E		MOV AL,DH	
FE80		ROL AX,CL	Shift work reg back 4
FE82		XOR DX,AX	Add into result reg
FE84		ROL AX,1	Shift work reg back 1
FE86		XCHG DH,DL	Swap partial sum into result reg
FE88		XOR DX,AX	Add work reg into results
FE8A		ROR AX,CL	Shift work reg over 4
FE8C		AND AL,11100000B	Clear off (EFGH)
FE8E		XOR DX,AX	Add (ABCD) into results
FE90		ROR AX,1	Shift work reg on over (AH = 0 for next pass)
FE92		XOR DH,AL	Add (ABCD into results low)
FE94		DEC BX	Decrement count
FE95		JNZ CRC_1	Loop till count = 0000
FE97		OR DX,DX	DX S/B = 0000 if O.K.
FE99		RET	Return to caller
FE9A	CRC_CHECK	ENDP	



- FE79
 - DX = 0xFFFF (initial value – first pass only)

AX	0	0	0	0	0	0	0	0	0	?	?	?	?	?	?	?
DX	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

- FE7B – LODSB
 - AX = next byte from cartridge

AX	0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H
DX	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

- FE7C – XOR DH,AL

AX	0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H
----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

DX	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	A	B	C	D	E	F	G	H	0	0	0	0	0	0	0	0

- FE7E – MOV AL,DH

AX	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H

DX	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	A	B	C	D	E	F	G	H	0	0	0	0	0	0	0	0

Address	Label	Instruction	Comment
FE71	CRC_CHECK	PROC NEAR	
		ASSUME DS:NOTHING	
FE71		MOV BX,CX	Save Count
FE73		MOV DX,0FFFFH	Init Encode Register
FE76		CLD	Set Dir flag to increment
FE77		XOR AH,AH	Init work reg high
FE79		MOV CL,4	Set rotate count
FE7B	CRC_1	LODSB	Get a byte
FE7C		XOR DH,AL	Form A _j + C _j + 1
FE7E		MOV AL,DH	
FE80		ROL AX,CL	Shift work reg back 4
FE82		XOR DX,AX	Add into result reg
FE84		ROL AX,1	Shift work reg back 1
FE86		XCHG DH,DL	Swap partial sum into result reg
FE88		XOR DX,AX	Add work reg into results
FE8A		ROR AX,CL	Shift work reg over 4
FE8C		AND AL,11100000B	Clear off (EFGH)
FE8E		XOR DX,AX	Add (ABCD) into results
FE90		ROR AX,1	Shift work reg on over (AH = 0 for next pass)
FE92		XOR DH,AL	Add (ABCD into results low)
FE94		DEC BX	Decrement count
FE95		JNZ CRC_1	Loop till count = 0000
FE97		OR DX,DX	DX S/B = 0000 if O.K.
FE99		RET	Return to caller
FE9A	CRC_CHECK	ENDP	

AX

0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H

⊕

DX

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A	B	C	D	E	F	G	H	0	0	0	0	0	0	0	0

⊕

• FE80 – ROL AX,CL (CL=4)

AX

0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	A	B	C	D	E	F	G	H	0	0	0	0

⊕

DX

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A	B	C	D	E	F	G	H	0	0	0	0	0	0	0	0

⊕

• FE82 – XOR DX,AX

AX

0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	A	B	C	D	E	F	G	H	0	0	0	0

⊕

DX

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A	B	C	D	E	F	G	H	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	A	B	C	D	E	F	G	H	0	0	0	0

⊕

⊕

⊕

Address	Label	Instruction	Comment
FE71	CRC_CHECK	PROC NEAR	
		ASSUME DS:NOTHING	
FE71		MOV BX,CX	Save Count
FE73		MOV DX,0FFFFH	Init Encode Register
FE76		CLD	Set Dir flag to increment
FE77		XOR AH,AH	Init work reg high
FE79		MOV CL,4	Set rotate count
FE7B	CRC_1	LODSB	Get a byte
FE7C		XOR DH,AL	Form A _j + C _j + 1
FE7E		MOV AL,DH	
FE80		ROL AX,CL	Shift work reg back 4
FE82		XOR DX,AX	Add into result reg
FE84		ROL AX,1	Shift work reg back 1
FE86		XCHG DH,DL	Swap partial sum into result reg
FE88		XOR DX,AX	Add work reg into results
FE8A		ROR AX,CL	Shift work reg over 4
FE8C		AND AL,11100000B	Clear off (EFGH)
FE8E		XOR DX,AX	Add (ABCD) into results
FE90		ROR AX,1	Shift work reg on over (AH = 0 for next pass)
FE92		XOR DH,AL	Add (ABCD into results low)
FE94		DEC BX	Decrement count
FE95		JNZ CRC_1	Loop till count = 0000
FE97		OR DX,DX	DX S/B = 0000 if O.K.
FE99		RET	Return to caller
FE9A	CRC_CHECK	ENDP	



AX

0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	A	B	C	D	E	F	G	H	0	0	0	0

⊕

DX

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A	B	C	D	E	F	G	H	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	A	B	C	D	E	F	G	H	0	0	0	0

⊕
⊕
⊕

• FE84 – ROL AX,1

AX

0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0

⊕

DX

1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1
A	B	C	D	E	F	G	H	0	0	0	0	0	0	0	0
0	0	0	0	A	B	C	D	E	F	G	H	0	0	0	0

⊕
⊕

• FE86 – XCHG DH,DL

AX

0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0

⊕

DX

0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H
E	F	G	H	0	0	0	0	0	0	0	0	A	B	C	D

⊕
⊕



Address	Label	Instruction	Comment
FE71	CRC_CHECK	PROC NEAR	
		ASSUME DS:NOTHING	
FE71		MOV BX,CX	Save Count
FE73		MOV DX,0FFFFH	Init Encode Register
FE76		CLD	Set Dir flag to increment
FE77		XOR AH,AH	Init work reg high
FE79		MOV CL,4	Set rotate count
FE7B	CRC_1	LODSB	Get a byte
FE7C		XOR DH,AL	Form A _j + C _j + 1
FE7E		MOV AL,DH	
FE80		ROL AX,CL	Shift work reg back 4
FE82		XOR DX,AX	Add into result reg
FE84		ROL AX,1	Shift work reg back 1
FE86		XCHG DH,DL	Swap partial sum into result reg
FE88		XOR DX,AX	Add work reg into results
FE8A		ROR AX,CL	Shift work reg over 4
FE8C		AND AL,11100000B	Clear off (EFGH)
FE8E		XOR DX,AX	Add (ABCD) into results
FE90		ROR AX,1	Shift work reg on over (AH = 0 for next pass)
FE92		XOR DH,AL	Add (ABCD into results low)
FE94		DEC BX	Decrement count
FE95		JNZ CRC_1	Loop till count = 0000
FE97		OR DX,DX	DX S/B = 0000 if O.K.
FE99		RET	Return to caller
FE9A	CRC_CHECK	ENDP	

AX

0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0

⊕

DX

0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H
E	F	G	H	0	0	0	0	0	0	0	0	A	B	C	D

⊕

⊕

• FE88 – XOR DX,AX

AX

0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0

⊕

DX

0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H
E	F	G	H	0	0	0	0	0	0	0	0	A	B	C	D
0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0

⊕

⊕

⊕

⊕

• FE8A – ROR AX,CL (CL=4)

AX

0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	A	B	C	D	E	F	G	H	0

⊕

DX

0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H
E	F	G	H	0	0	0	0	0	0	0	0	A	B	C	D
0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0

⊕

⊕

⊕

Address	Label	Instruction	Comment
FE71	CRC_CHECK	PROC NEAR	
		ASSUME DS:NOTHING	
FE71		MOV BX,CX	Save Count
FE73		MOV DX,0FFFFH	Init Encode Register
FE76		CLD	Set Dir flag to increment
FE77		XOR AH,AH	Init work reg high
FE79		MOV CL,4	Set rotate count
FE7B	CRC_1	LODSB	Get a byte
FE7C		XOR DH,AL	Form A _j + C _j + 1
FE7E		MOV AL,DH	
FE80		ROL AX,CL	Shift work reg back 4
FE82		XOR DX,AX	Add into result reg
FE84		ROL AX,1	Shift work reg back 1
FE86		XCHG DH,DL	Swap partial sum into result reg
FE88		XOR DX,AX	Add work reg into results
FE8A		ROR AX,CL	Shift work reg over 4
FE8C		AND AL,11100000B	Clear off (EFGH)
FE8E		XOR DX,AX	Add (ABCD) into results
FE90		ROR AX,1	Shift work reg on over (AH = 0 for next pass)
FE92		XOR DH,AL	Add (ABCD into results low)
FE94		DEC BX	Decrement count
FE95		JNZ CRC_1	Loop till count = 0000
FE97		OR DX,DX	DX S/B = 0000 if O.K.
FE99		RET	Return to caller
FE9A	CRC_CHECK	ENDP	



AX

0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	⊕
0	0	0	0	0	0	0	A	B	C	D	E	F	G	H	0	

DX

0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	⊕
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H	⊕
E	F	G	H	0	0	0	0	0	0	0	0	A	B	C	D	⊕
0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0	



• FE8C – AND AL,11100000B

AX

0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	⊕
0	0	0	0	0	0	0	A	B	C	D	0	0	0	0	0	

DX

0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	⊕
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H	⊕
E	F	G	H	0	0	0	0	0	0	0	0	A	B	C	D	⊕
0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0	

• FE8E – XOR DX,AX

AX

0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	⊕
0	0	0	0	0	0	0	A	B	C	D	0	0	0	0	0	

DX

0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	⊕
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H	⊕
E	F	G	H	0	0	0	0	0	0	0	0	A	B	C	D	⊕
0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0	⊕
0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	⊕
0	0	0	0	0	0	0	A	B	C	D	0	0	0	0	0	

Address	Label	Instruction	Comment
FE71	CRC_CHECK	PROC NEAR	
		ASSUME DS:NOTHING	
FE71		MOV BX,CX	Save Count
FE73		MOV DX,0FFFFH	Init Encode Register
FE76		CLD	Set Dir flag to increment
FE77		XOR AH,AH	Init work reg high
FE79		MOV CL,4	Set rotate count
FE7B	CRC_1	LODSB	Get a byte
FE7C		XOR DH,AL	Form Aj + Cj + 1
FE7E		MOV AL,DH	
FE80		ROL AX,CL	Shift work reg back 4
FE82		XOR DX,AX	Add into result reg
FE84		ROL AX,1	Shift work reg back 1
FE86		XCHG DH,DL	Swap partial sum into result reg
FE88		XOR DX,AX	Add work reg into results
FE8A		ROR AX,CL	Shift work reg over 4
FE8C		AND AL,11100000B	Clear off (EFGH)
FE8E		XOR DX,AX	Add (ABCD) into results
FE90		ROR AX,1	Shift work reg on over (AH = 0 for next pass)
FE92		XOR DH,AL	Add (ABCD into results low)
FE94		DEC BX	Decrement count
FE95		JNZ CRC_1	Loop till count = 0000
FE97		OR DX,DX	DX S/B = 0000 if O.K.
FE99		RET	Return to caller
FE9A	CRC_CHECK	ENDP	

AX

0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	A	B	C	D	0	0	0	0

⊕

DX

0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H
E	F	G	H	0	0	0	0	0	0	0	0	A	B	C	D
0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	A	B	C	D	0	0	0	0

⊕

⊕

⊕

⊕

⊕

• FE90 – ROR AX,1

AX

0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	A	B	C	D	0	0	0	0

⊕

DX

0	0	0	1	0	0	0	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H
E	F	G	H	0	0	0	0	0	0	0	0	A	B	C	D
0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0
0	0	0	0	0	0	0	0	A	B	C	D	0	0	0	0

⊕

⊕

⊕

⊕

Address	Label	Instruction	Comment
FE71	CRC_CHECK	PROC NEAR	
		ASSUME DS:NOTHING	
FE71		MOV BX,CX	Save Count
FE73		MOV DX,0FFFFH	Init Encode Register
FE76		CLD	Set Dir flag to increment
FE77		XOR AH,AH	Init work reg high
FE79		MOV CL,4	Set rotate count
FE7B	CRC_1	LODSB	Get a byte
FE7C		XOR DH,AL	Form Aj + Cj + 1
FE7E		MOV AL,DH	
FE80		ROL AX,CL	Shift work reg back 4
FE82		XOR DX,AX	Add into result reg
FE84		ROL AX,1	Shift work reg back 1
FE86		XCHG DH,DL	Swap partial sum into result reg
FE88		XOR DX,AX	Add work reg into results
FE8A		ROR AX,CL	Shift work reg over 4
FE8C		AND AL,11100000B	Clear off (EFGH)
FE8E		XOR DX,AX	Add (ABCD) into results
FE90		ROR AX,1	Shift work reg on over (AH = 0 for next pass)
FE92		XOR DH,AL	Add (ABCD into results low)
FE94		DEC BX	Decrement count
FE95		JNZ CRC_1	Loop till count = 0000
FE97		OR DX,DX	DX S/B = 0000 if O.K.
FE99		RET	Return to caller
FE9A	CRC_CHECK	ENDP	



AX

0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	A	B	C	D	0	0	0	0

⊕

DX

0	0	0	1	0	0	0	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H
E	F	G	H	0	0	0	0	0	0	0	0	A	B	C	D
0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0
0	0	0	0	0	0	0	A	B	C	D	0	0	0	0	0

⊕

⊕

⊕

⊕

• FE92 – XOR DH,AL

AX

0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	A	B	C	D	0	0	0	0

⊕

DX

0	0	0	1	0	0	0	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H
E	F	G	H	0	0	0	0	0	0	0	0	A	B	C	D
0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0
0	0	0	0	0	0	0	A	B	C	D	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
A	B	C	D	0	0	0	0	0	0	0	0	0	0	0	0

⊕

⊕

⊕

⊕

⊕

⊕

Address	Label	Instruction	Comment
FE71	CRC_CHECK	PROC NEAR	
		ASSUME DS:NOTHING	
FE71		MOV BX,CX	Save Count
FE73		MOV DX,0FFFFH	Init Encode Register
FE76		CLD	Set Dir flag to increment
FE77		XOR AH,AH	Init work reg high
FE79		MOV CL,4	Set rotate count
FE7B	CRC_1	LODSB	Get a byte
FE7C		XOR DH,AL	Form A _j + C _j + 1
FE7E		MOV AL,DH	
FE80		ROL AX,CL	Shift work reg back 4
FE82		XOR DX,AX	Add into result reg
FE84		ROL AX,1	Shift work reg back 1
FE86		XCHG DH,DL	Swap partial sum into result reg
FE88		XOR DX,AX	Add work reg into results
FE8A		ROR AX,CL	Shift work reg over 4
FE8C		AND AL,11100000B	Clear off (EFGH)
FE8E		XOR DX,AX	Add (ABCD) into results
FE90		ROR AX,1	Shift work reg on over (AH = 0 for next pass)
FE92		XOR DH,AL	Add (ABCD into results low)
FE94		DEC BX	Decrement count
FE95		JNZ CRC_1	Loop till count = 0000
FE97		OR DX,DX	DX S/B = 0000 if O.K.
FE99		RET	Return to caller
FE9A	CRC_CHECK	ENDP	



• Cleanup

DX

1	1	1	0	0	0	0	1	1	1	1	1	0	0	0	0	⊕
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H	⊕
E	F	G	H	0	0	0	0	0	0	0	0	A	B	C	D	⊕
0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0	⊕
0	0	0	0	0	0	0	A	B	C	D	0	0	0	0	0	⊕
A	B	C	D	0	0	0	0	0	0	0	0	0	0	0	0	

table[index = ABCDEFGH bits] =

0	0	0	A	B	C	D	E	F	G	H	0	0	0	0	0	⊕
0	0	0	0	0	0	0	0	A	B	C	D	E	F	G	H	⊕
E	F	G	H	0	0	0	0	0	0	0	0	0	0	0	0	⊕
0	0	0	0	0	0	0	0	0	0	0	0	A	B	C	D	⊕
0	0	0	0	0	0	0	A	B	C	D	0	0	0	0	0	⊕
A	B	C	D	0	0	0	0	0	0	0	0	0	0	0	0	

DX

1	1	1	0	0	0	0	1	1	1	1	1	0	0	0	0	⊕
table[index = ABCDEFGH]																



Address	Label	Instruction	Comment
FE71	CRC_CHECK	PROC NEAR	
		ASSUME DS:NOTHING	
FE71		MOV BX,CX	Save Count
FE73		MOV DX,0FFFFH	Init Encode Register
FE76		CLD	Set Dir flag to increment
FE77		XOR AH,AH	Init work reg high
FE79		MOV CL,4	Set rotate count
FE7B	CRC_1	LODSB	Get a byte
FE7C		XOR DH,AL	Form A _j + C _j + 1
FE7E		MOV AL,DH	
FE80		ROL AX,CL	Shift work reg back 4
FE82		XOR DX,AX	Add into result reg
FE84		ROL AX,1	Shift work reg back 1
FE86		XCHG DH,DL	Swap partial sum into result reg
FE88		XOR DX,AX	Add work reg into results
FE8A		ROR AX,CL	Shift work reg over 4
FE8C		AND AL,11100000B	Clear off (EFGH)
FE8E		XOR DX,AX	Add (ABCD) into results
FE90		ROR AX,1	Shift work reg on over (AH = 0 for next pass)
FE92		XOR DH,AL	Add (ABCD into results low)
FE94		DEC BX	Decrement count
FE95		JNZ CRC_1	Loop till count = 0000
FE97		OR DX,DX	DX S/B = 0000 if O.K.
FE99		RET	Return to caller
FE9A	CRC_CHECK	ENDP	



- Direct method calls for table lookup of:
 - xor of rotated out value with non-augmented message

- Rotated out value = high bits of initial value **0xFF**
 - Non-augmented message = ABCDEFGH bits


- $\text{table}[\text{0xFF} \oplus \text{ABCDEFGH}] = \text{table}[\text{0xFF}] \oplus \text{table}[\text{ABCDEFGH}]$

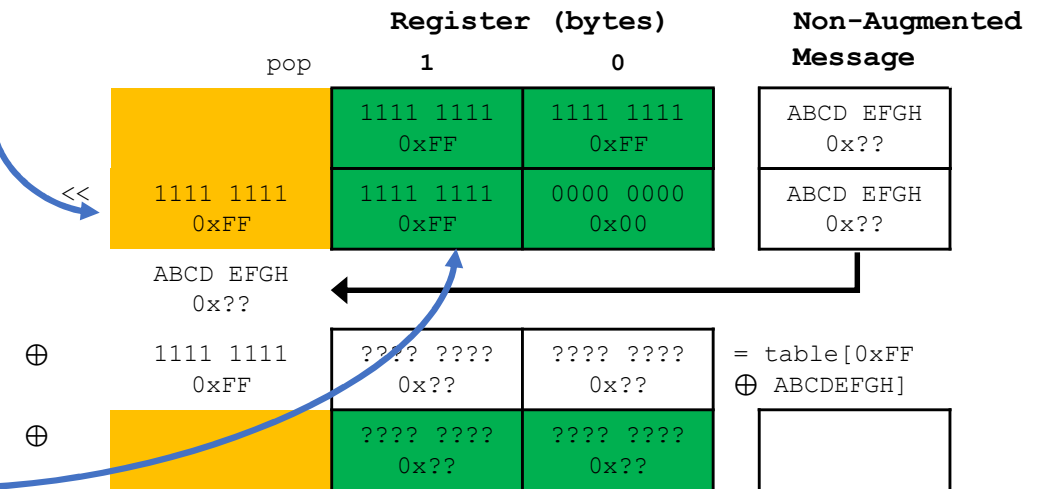
- Then the table value is xor with the register

- **0xFF00**
 - Remember that in direct method 0xFFFF was shifted left 1 byte
 - $\text{table}[\text{0xFF}] \oplus \text{table}[\text{ABCDEFGH}] \oplus \text{0xFF00} =$

0xFF00	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	⊕
table[0xFF]	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0	⊕
table[ABCDEFGH]																

- Matches DX from CRC calculation in BIOS

1	1	1	0	0	0	0	1	1	1	1	1	0	0	0	0	 \oplus
table[ABCDEFGH]																



Note: $\text{table}[\text{0xFF}] \oplus \text{0xFF00}$ here is only correct for the first iteration shown in this example. Subsequent iterations are similar but the starting value of DX (register) will of course be updated for the next message byte.

Thanks for Watching!



- More than you ever wanted to know about the CRC algorithm used on PCjr cartridges
- See my github for
 - A Painless Guide to CRC Error Detection Algorithms
 - C++ implementation
 - <https://github.com/guldmuddypaws>