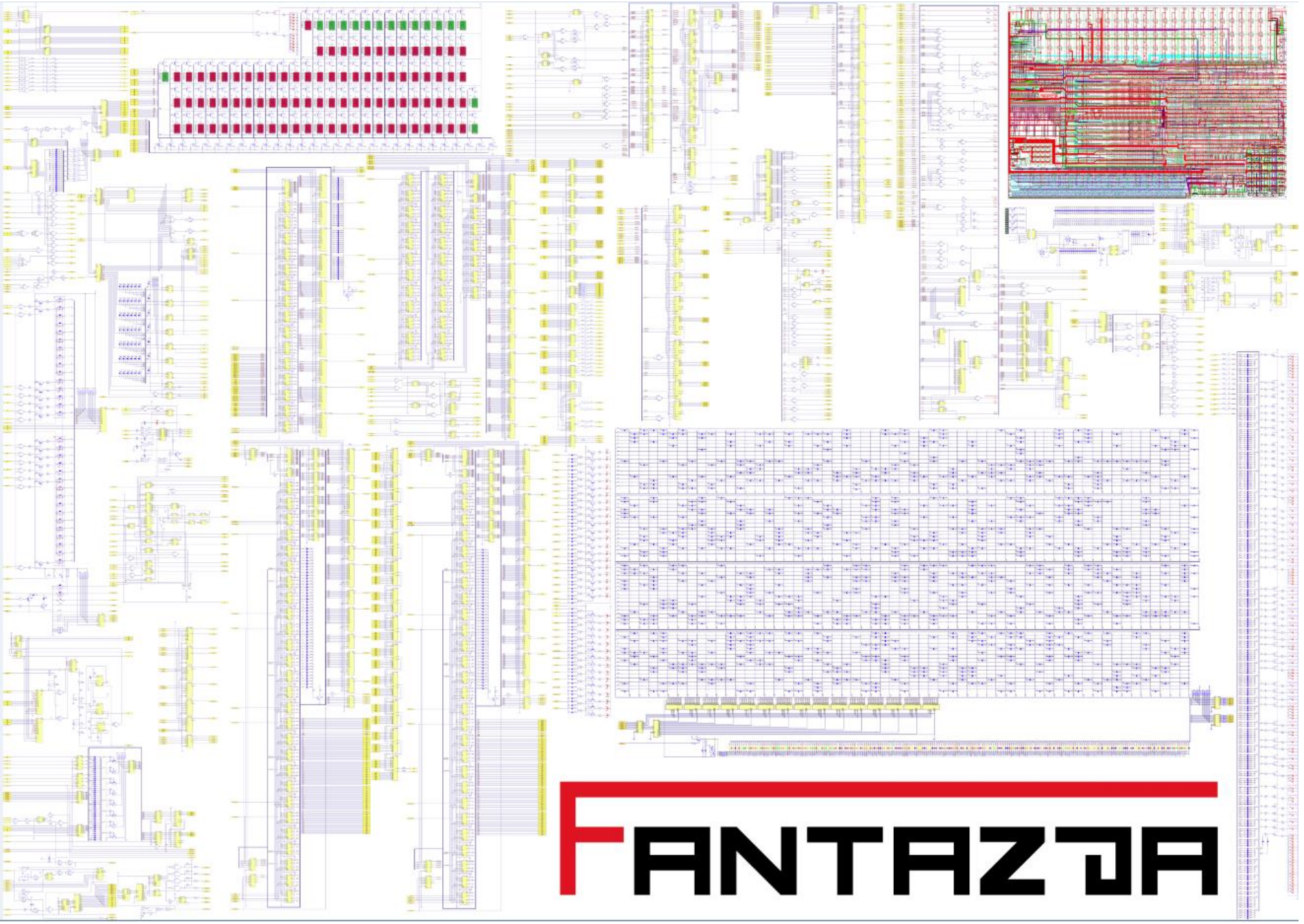
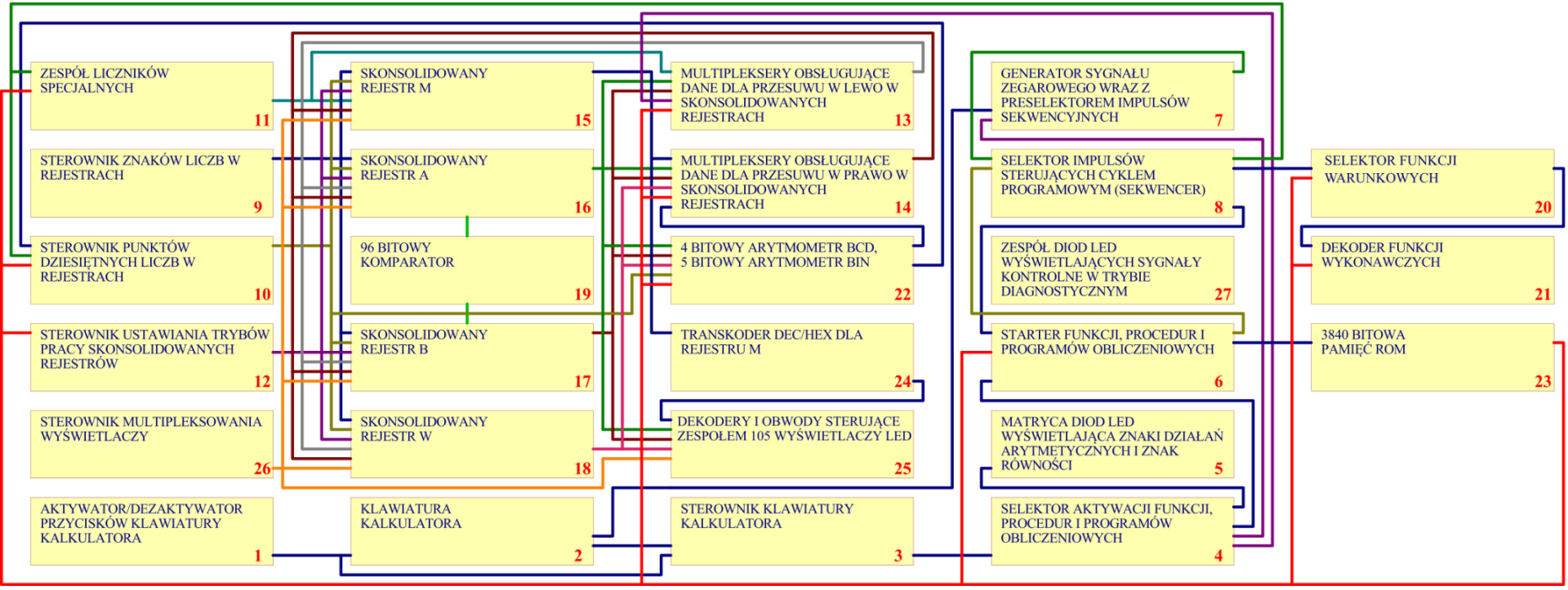
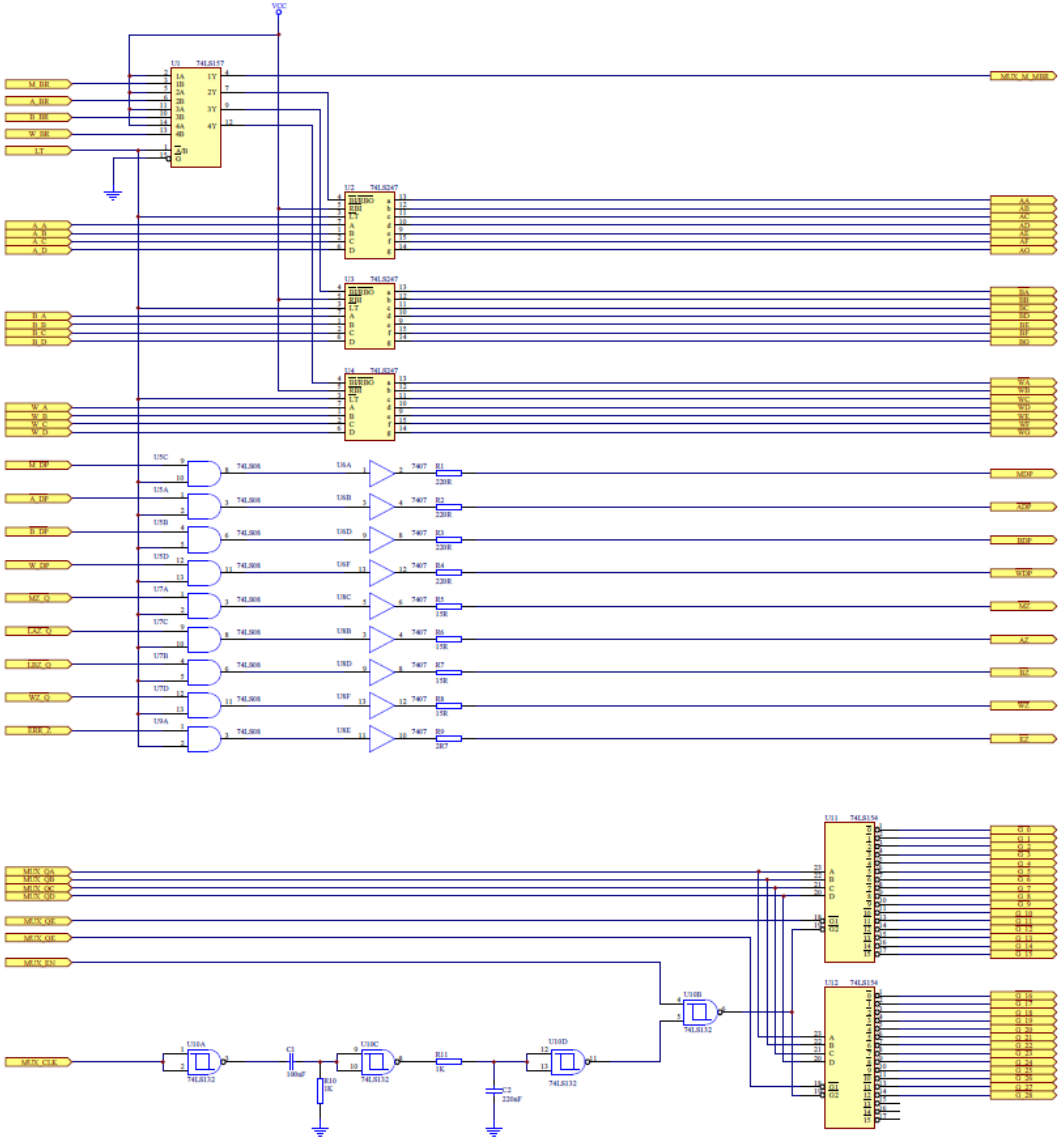
KALKULATOR NA UKŁADACH TTL Z TRYBEM DIAGNOSTYCZNYM


FANTAZJA

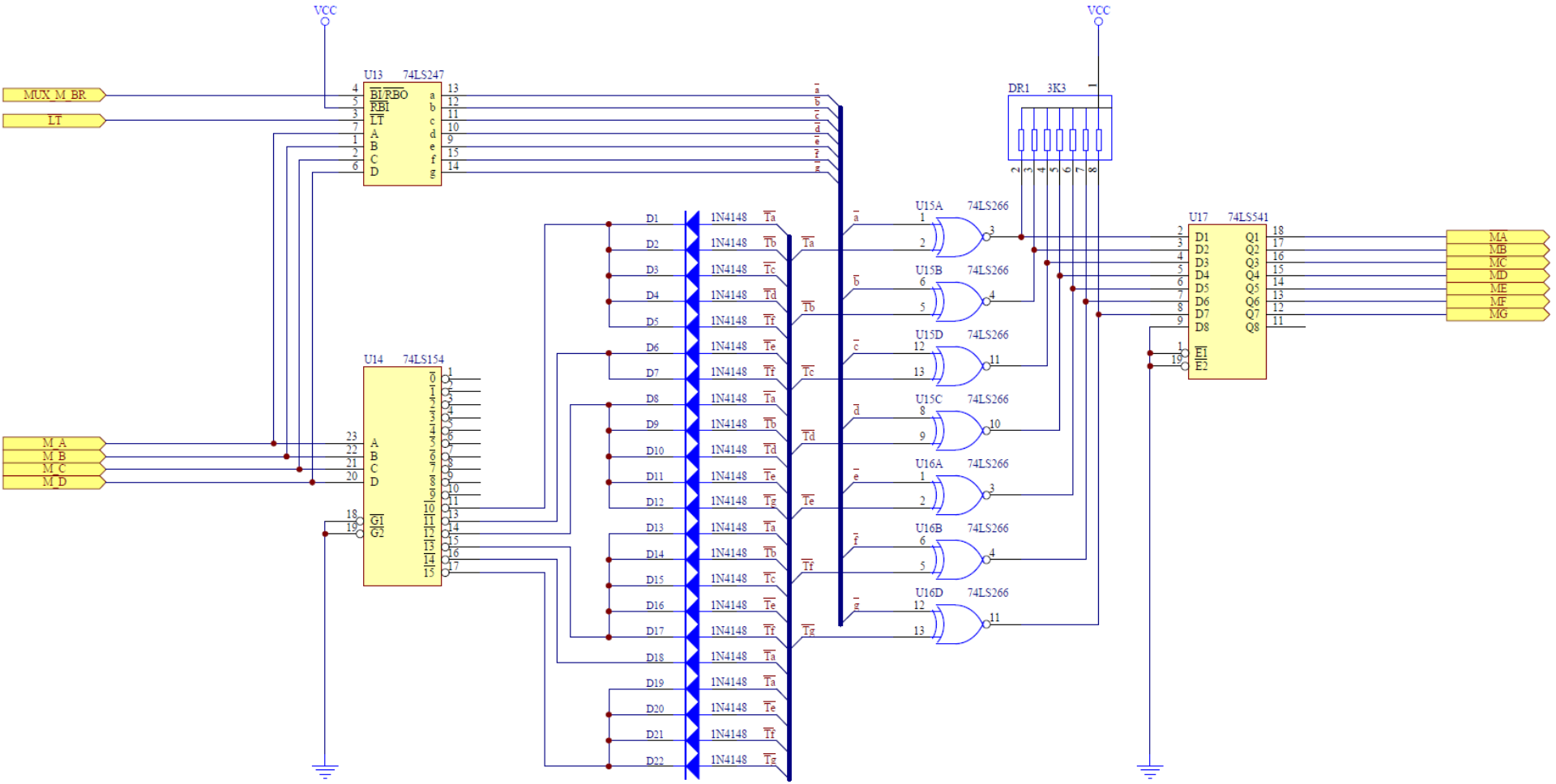
SCHEMAT BLOKOWY URZĄDZENIA



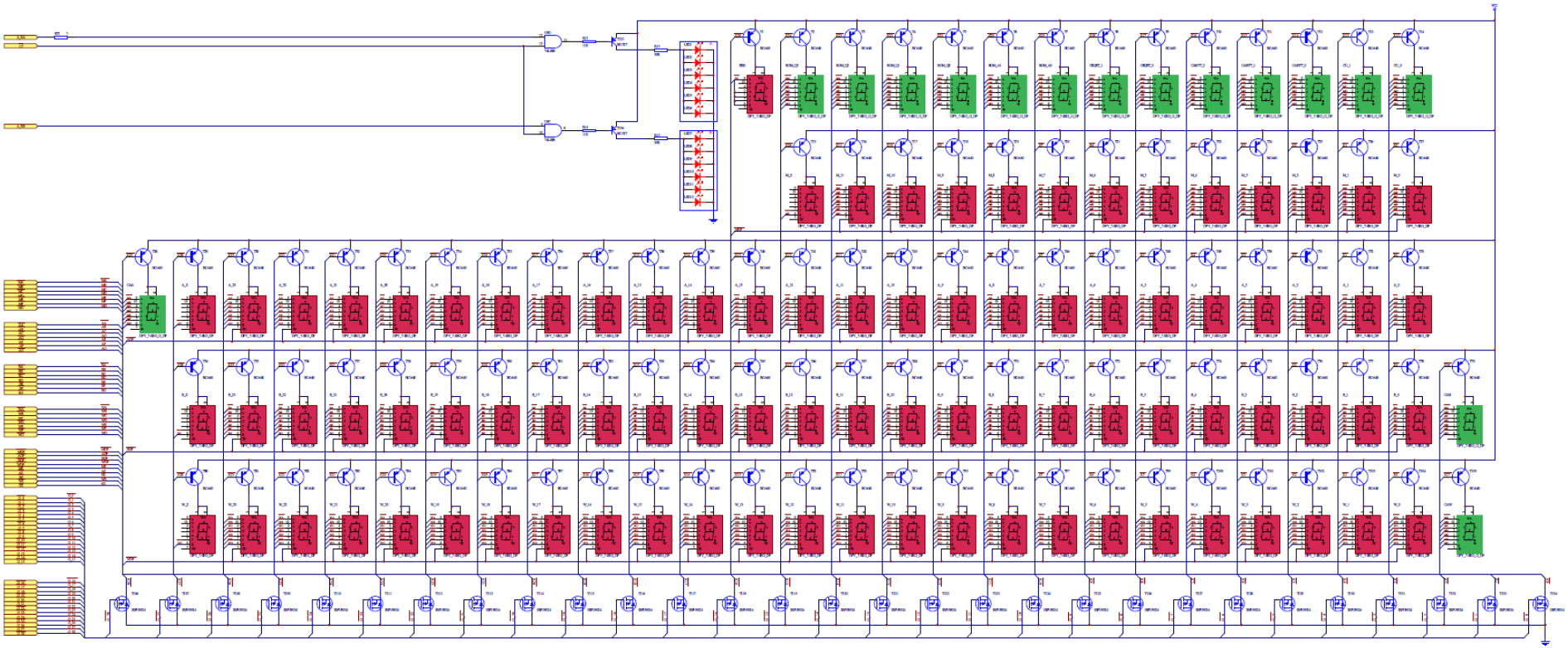
1.DEKODERY I OBWODY STERUJĄCE ZESPOŁEM WYŚWIETLACZY



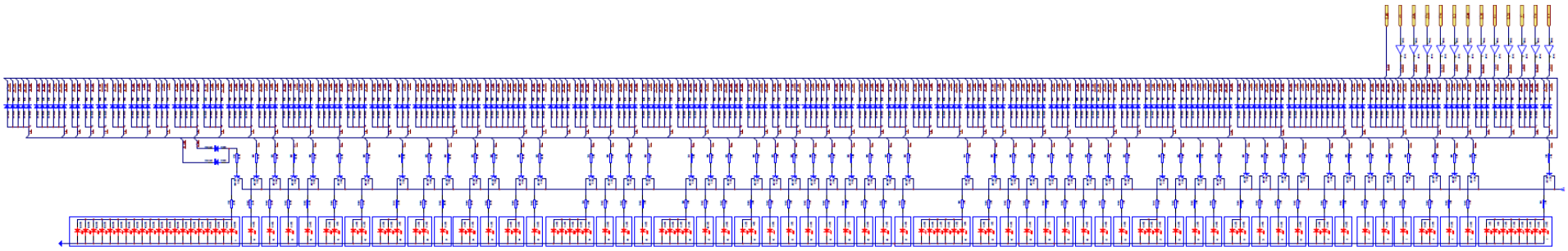
2.TRANSKODER DEC/HEX DLA WYŚWIETLACZY REJESTRU M



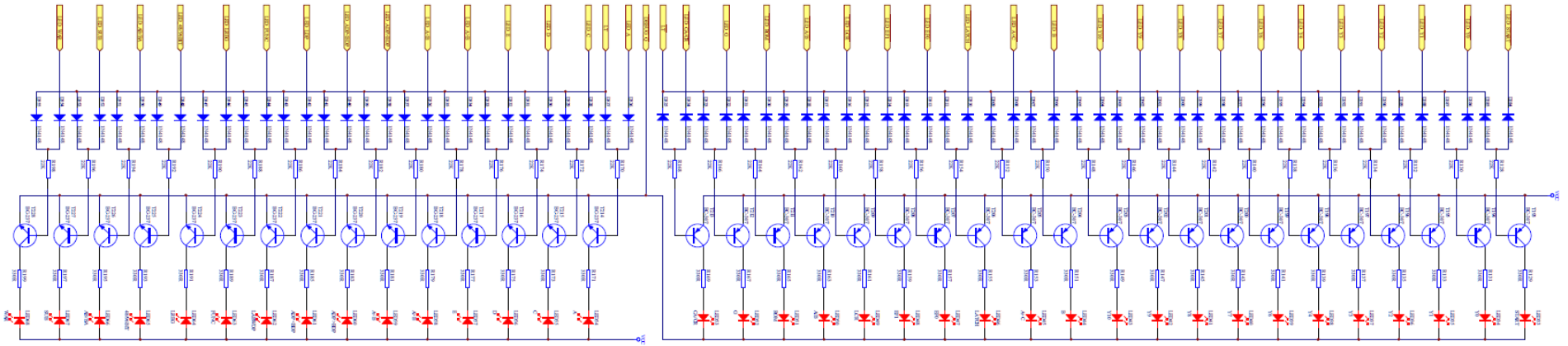
3.ZESPÓŁ 105 WYŚWIETLACZY SIEDMIOSEGMENTOWYCH WRAZ Z DIODAMI LED WYBORU AKTYWNEGO REJESTRU



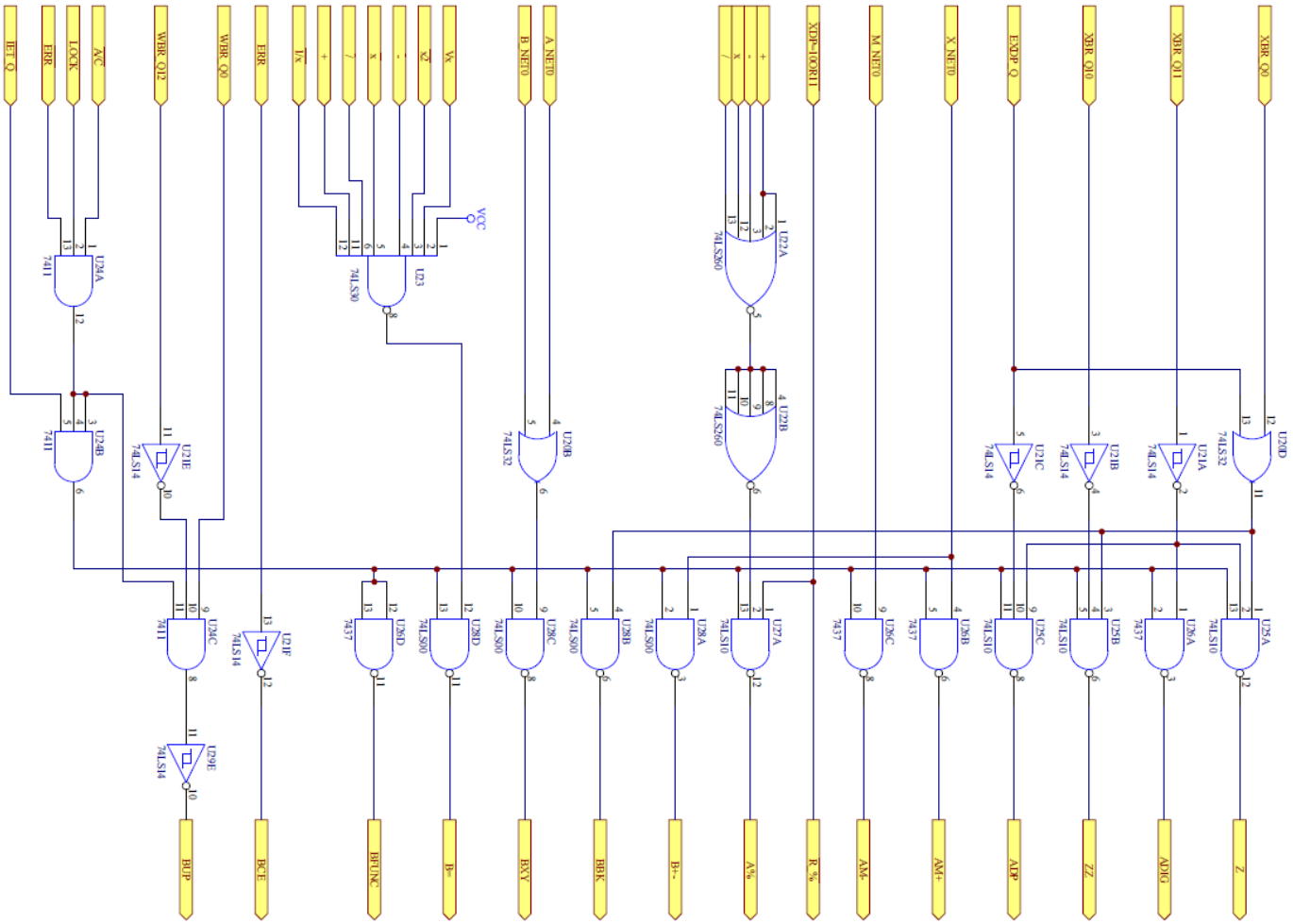
4.MATRYCA DIOD LED WYŚWIETLAJĄCA ZNAKI DZIAŁAŃ ARYTMETYCZNYCH I ZNAK RÓWNOŚCI



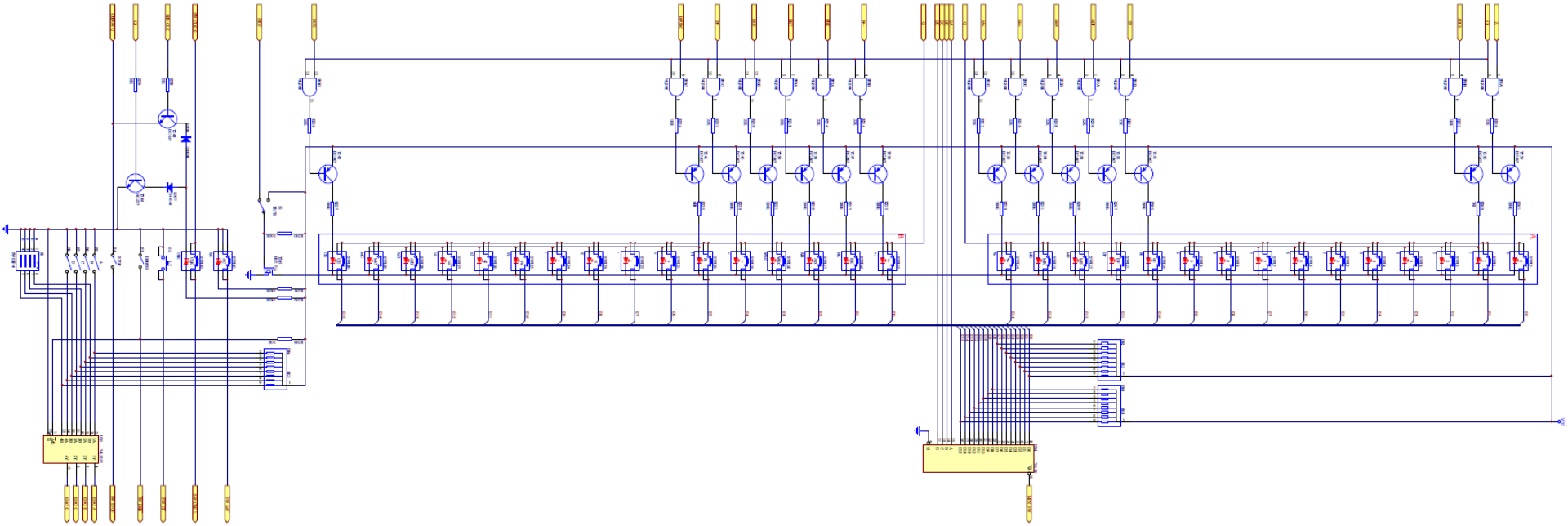
5.ZESPÓŁ DIOD LED WYŚWIETLAJĄCYCH SYGNAŁY KONTROLNE W TRYBIE KROKOWYM



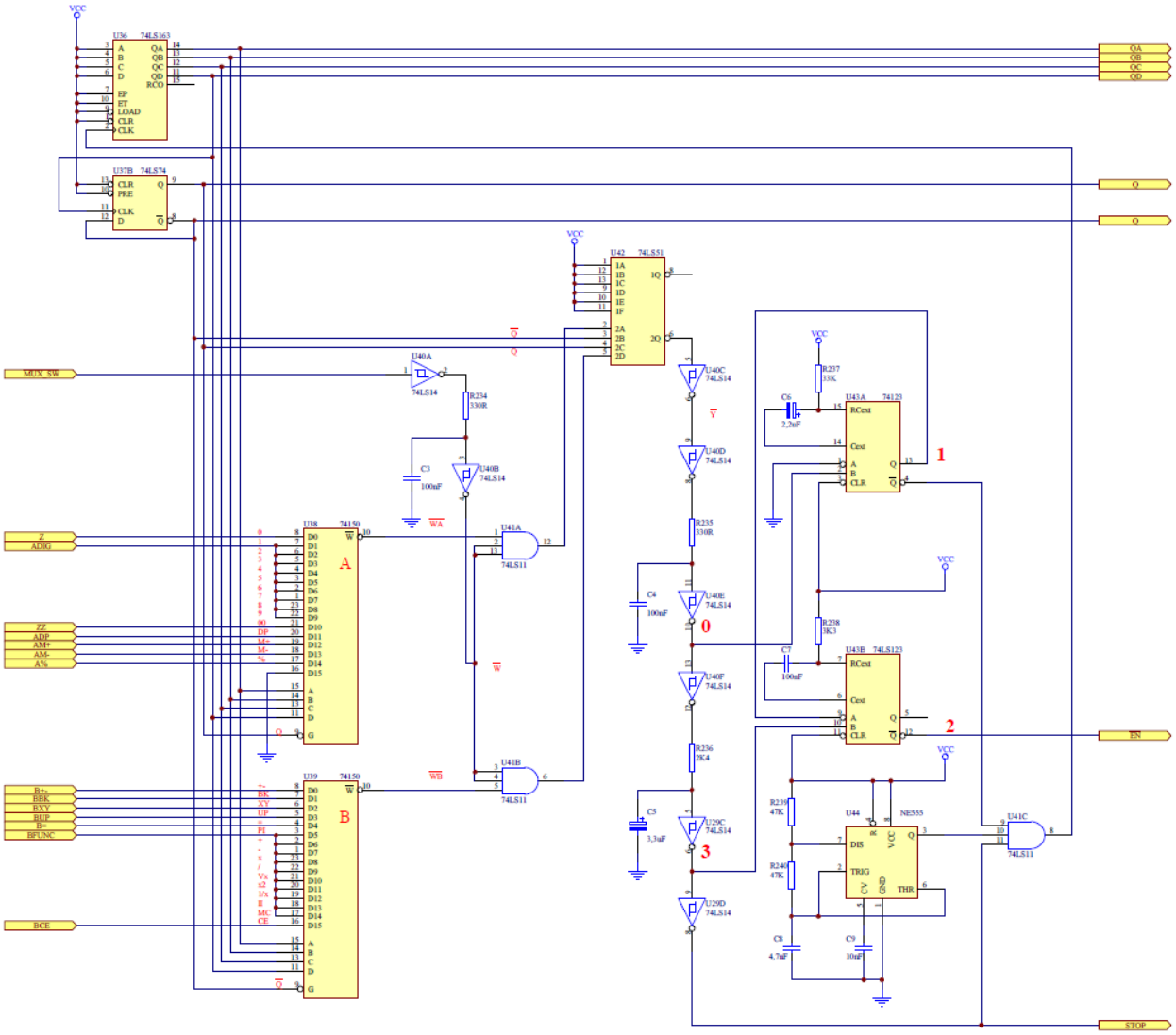
6.AKTYWATOR/DEZAKTYWATOR PRZYCISKÓW KLAWIATURY KALKULATORA



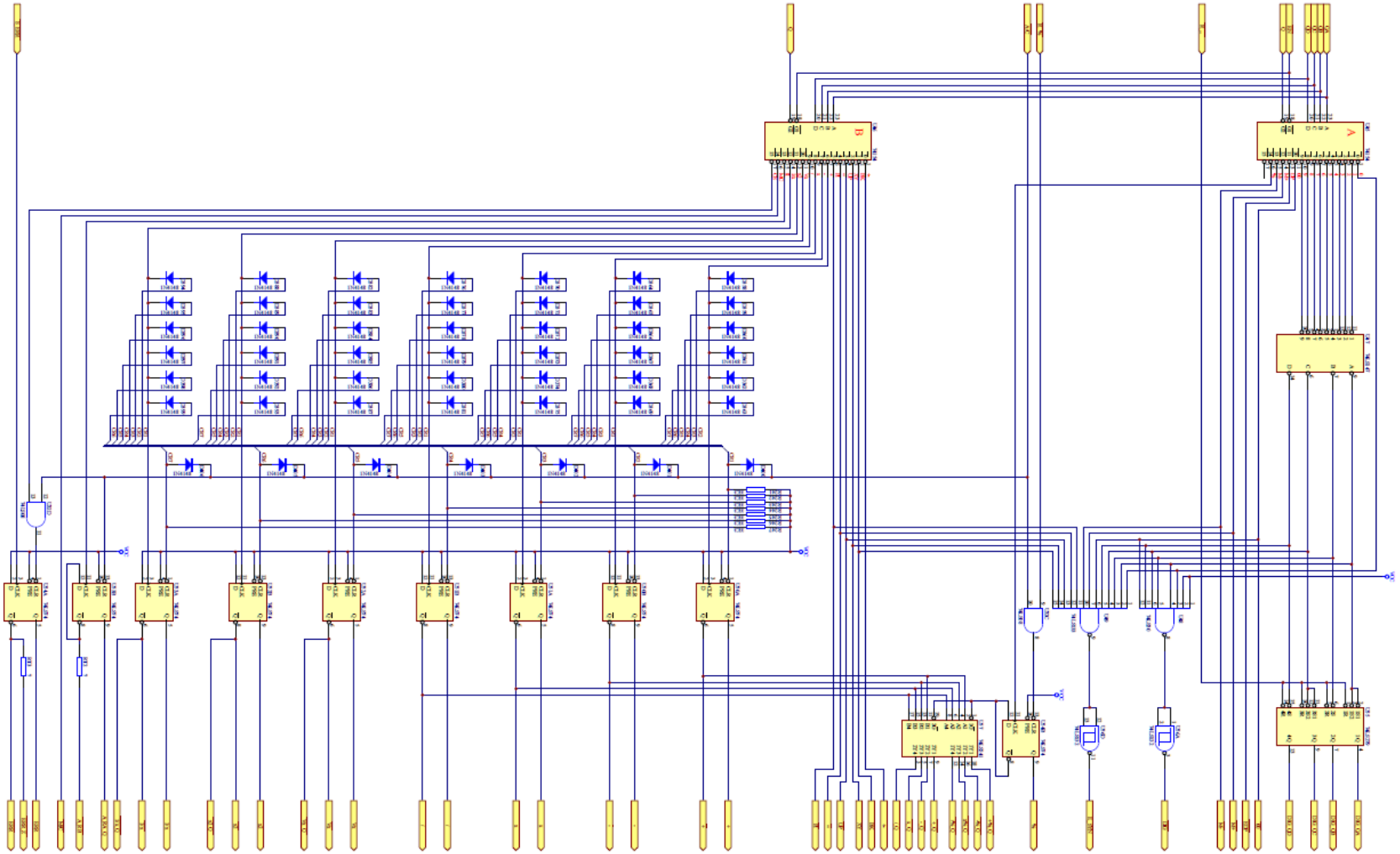
7.KLAWIATURA KALKULATORA



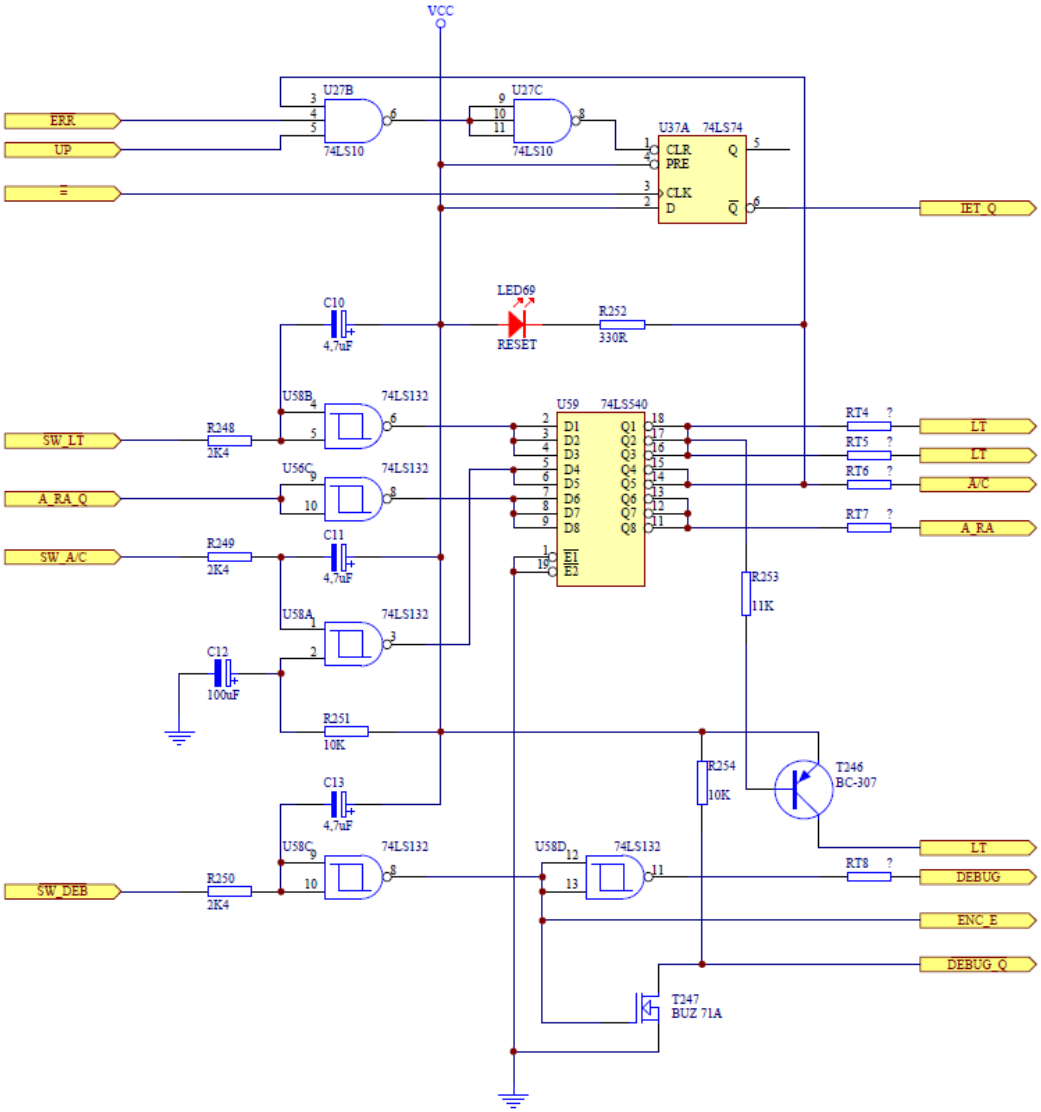
8.STEROWNIK KLAWIATURY KALKULATORA



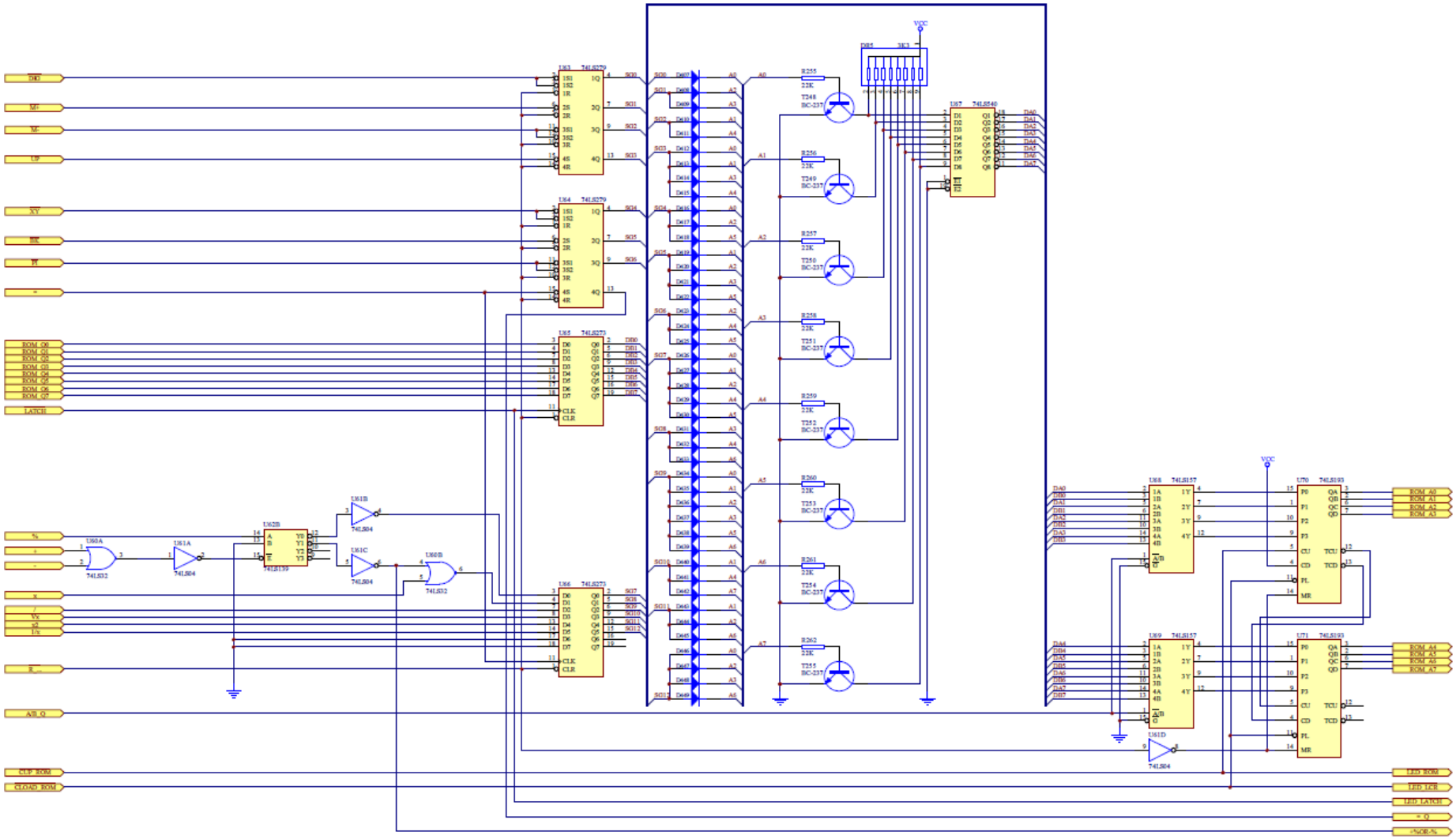
9.SELEKTOR AKTYWACJI FUNKCJI, PRECEDUR I PROGRAMÓW OBLICZENIOWYCH



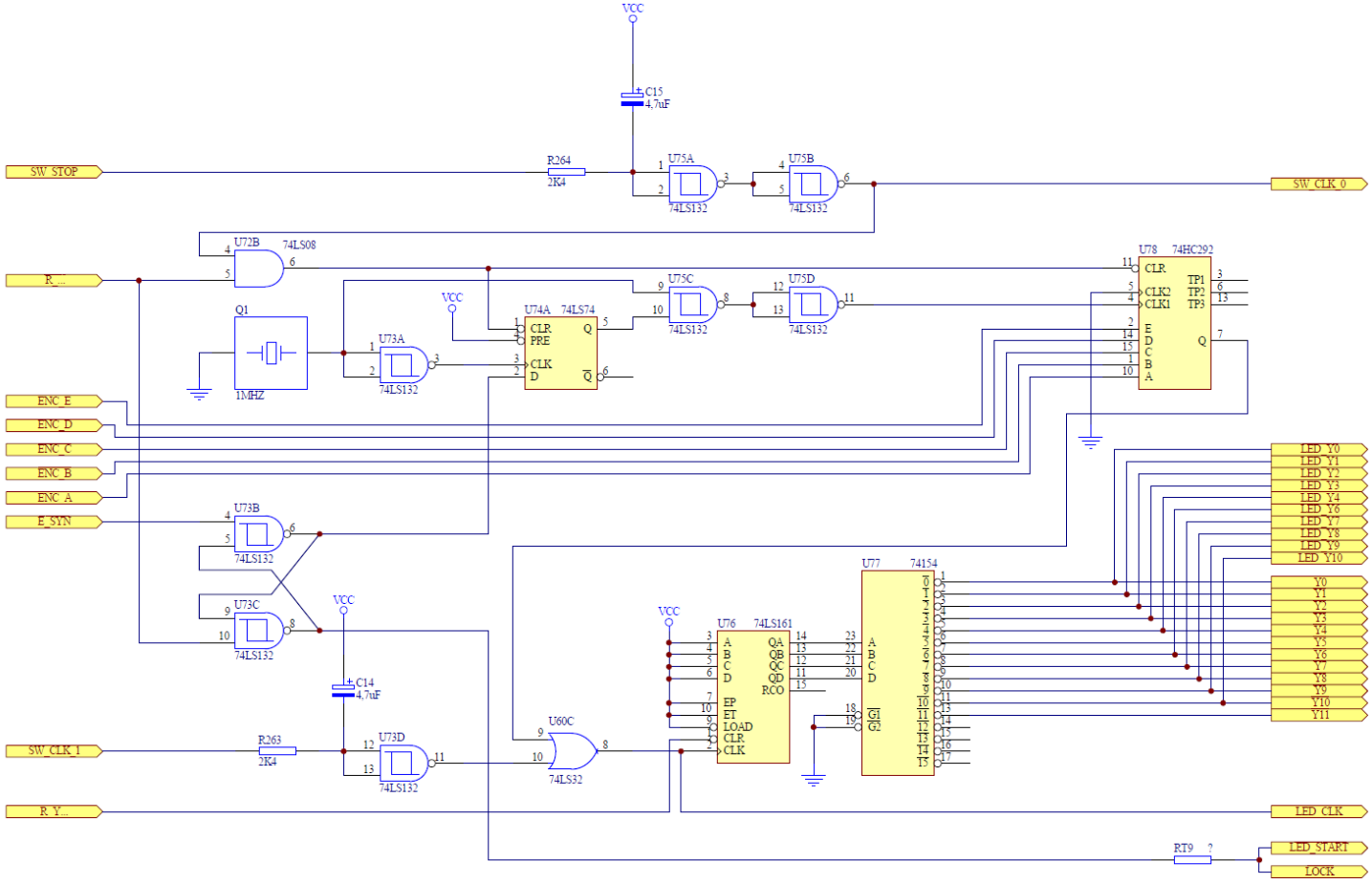
10. BUFOR SYGNAŁÓW NADRZĘDNYCH



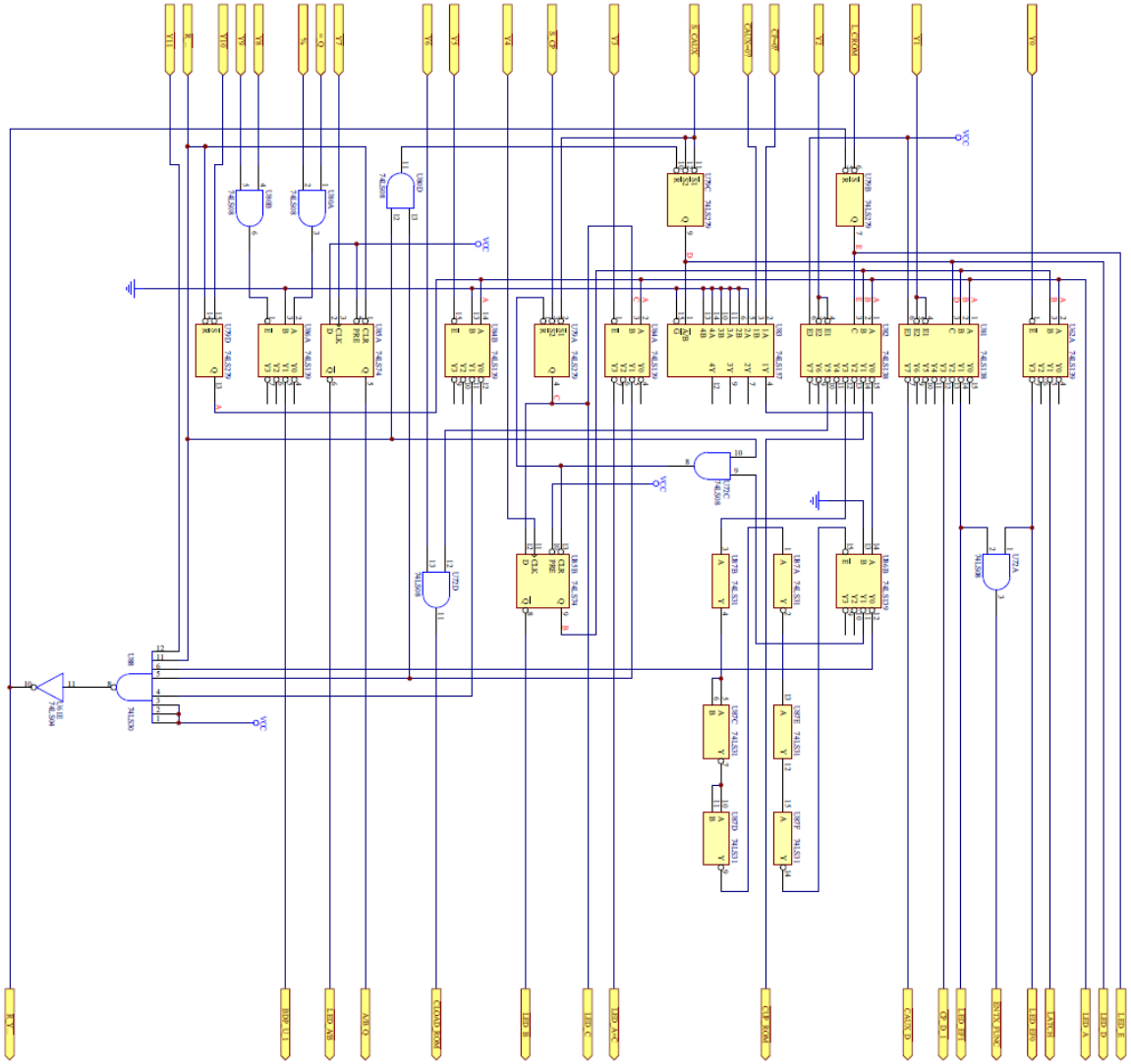
11.STARTER FUNKCJI, PROCEDUR I PROGRAMÓW OBLICZENIOWYCH



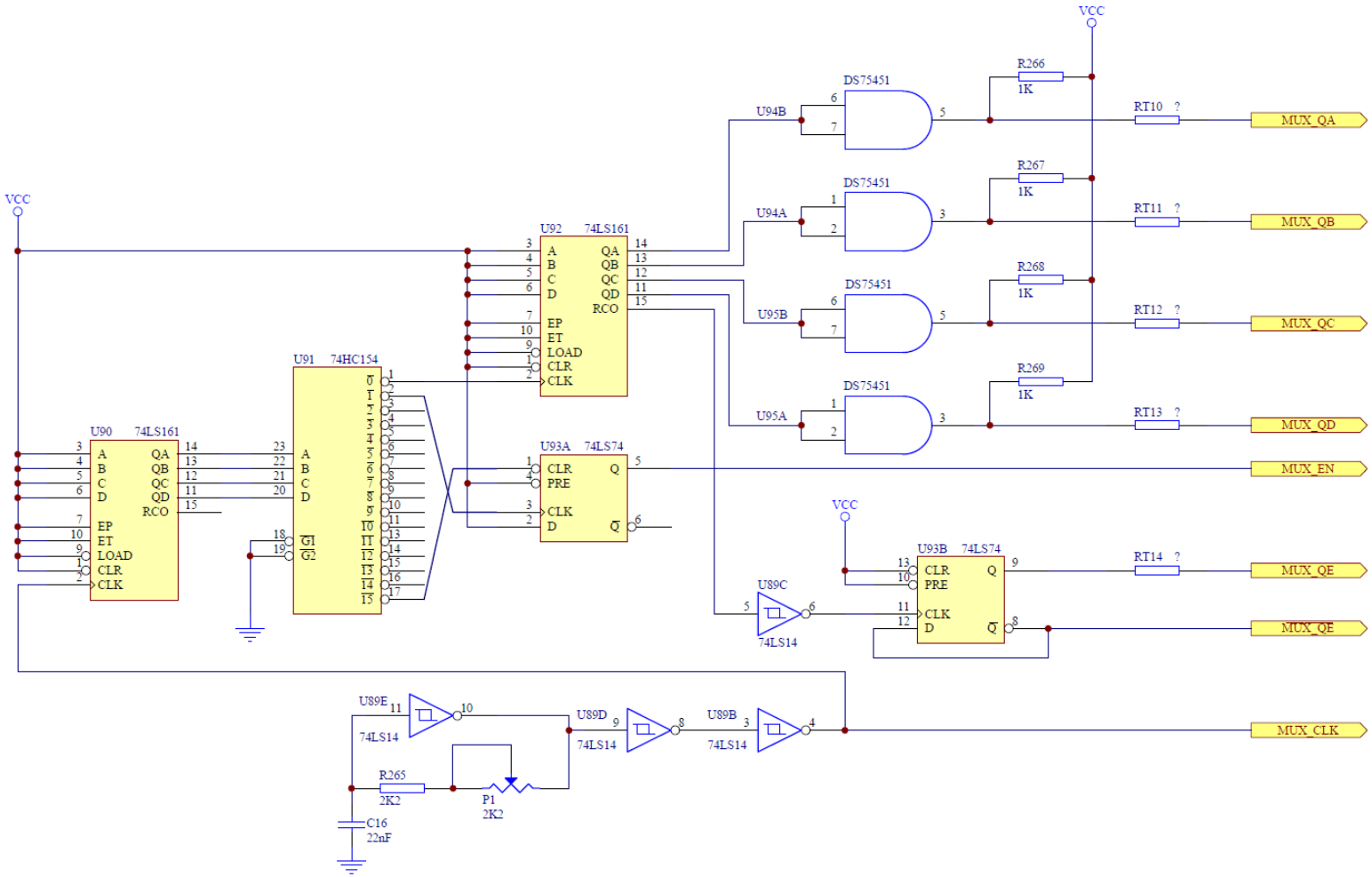
12.GENERATOR SYGNAŁU ZEGAROWEGO WRAZ Z PRESELEKTOREM IMPULSÓW SEKWENCYJNYCH



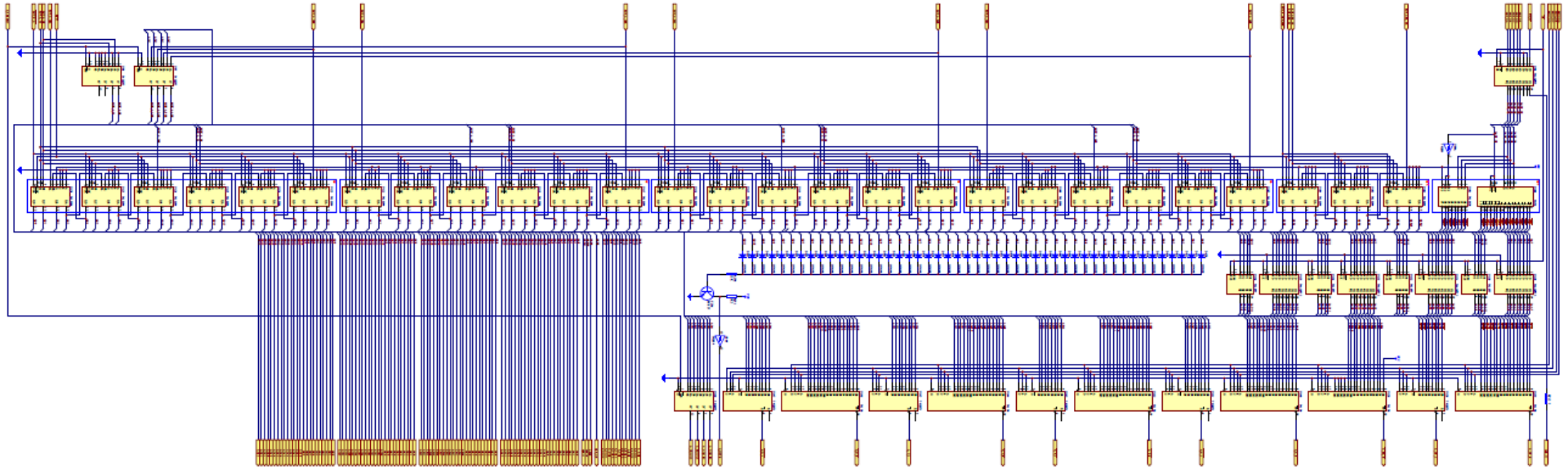
13.SELEKTOR IMPULSÓW STERUJĄCYCH CYKLEM PROGRAMOWYM (SEKWENCER)



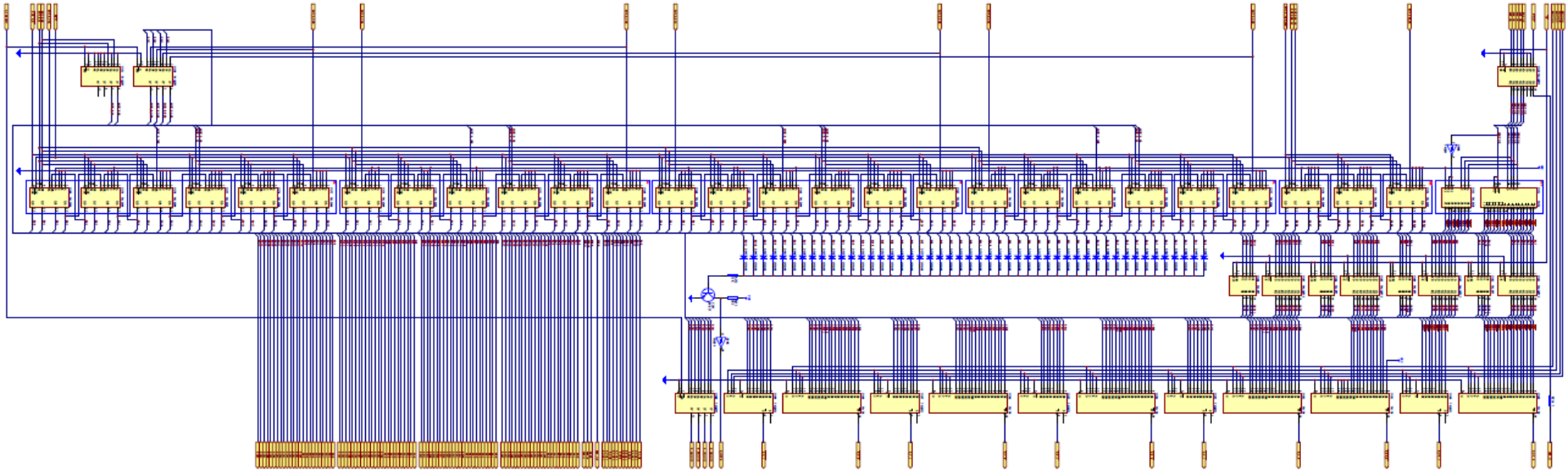
14.STEROWNIK MULTIPLEKSOWANIA WYŚWIETLACZY



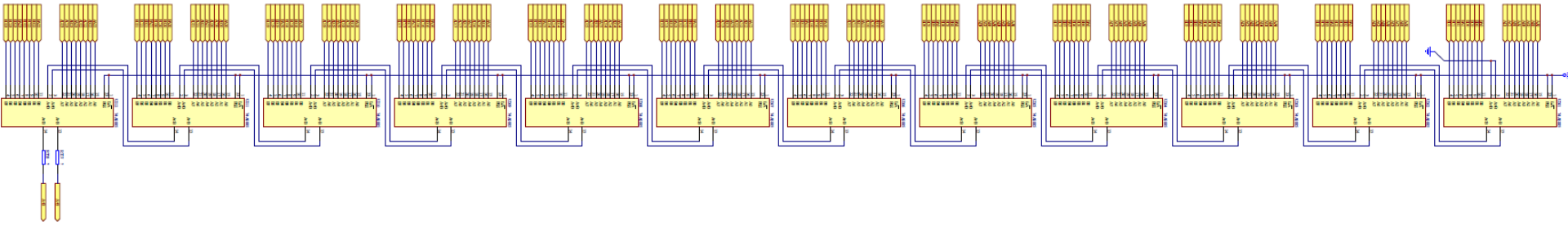
15.ZESPÓŁ REJESTRÓW I DEKODERÓW PREZENTUJĄCYCH WARTOŚĆ LICZBY „A”



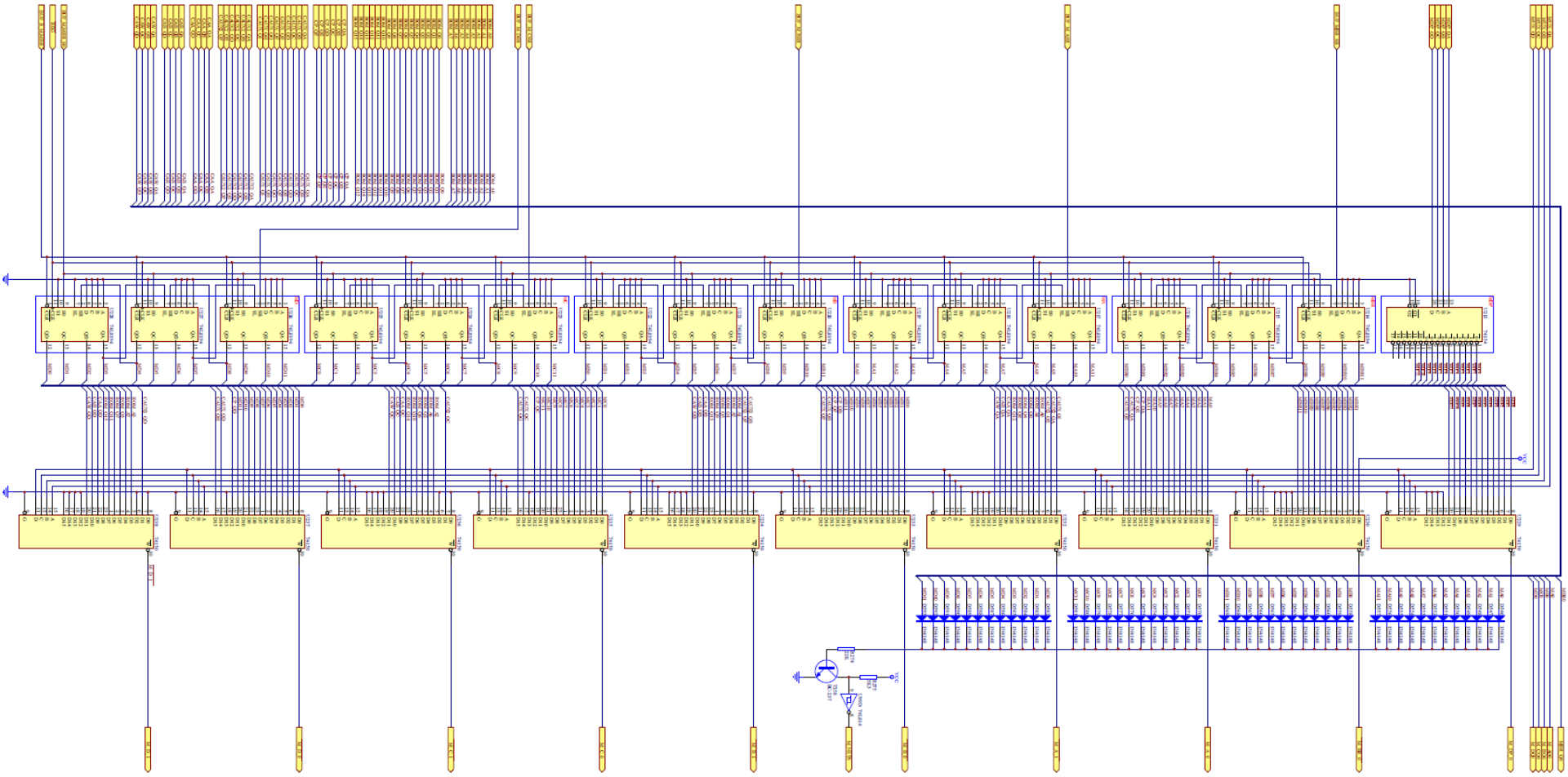
16.ZESPÓŁ REJESTRÓW I DEKODERÓW PREZENTUJĄCYCH WARTOŚĆ LICZBY „B”



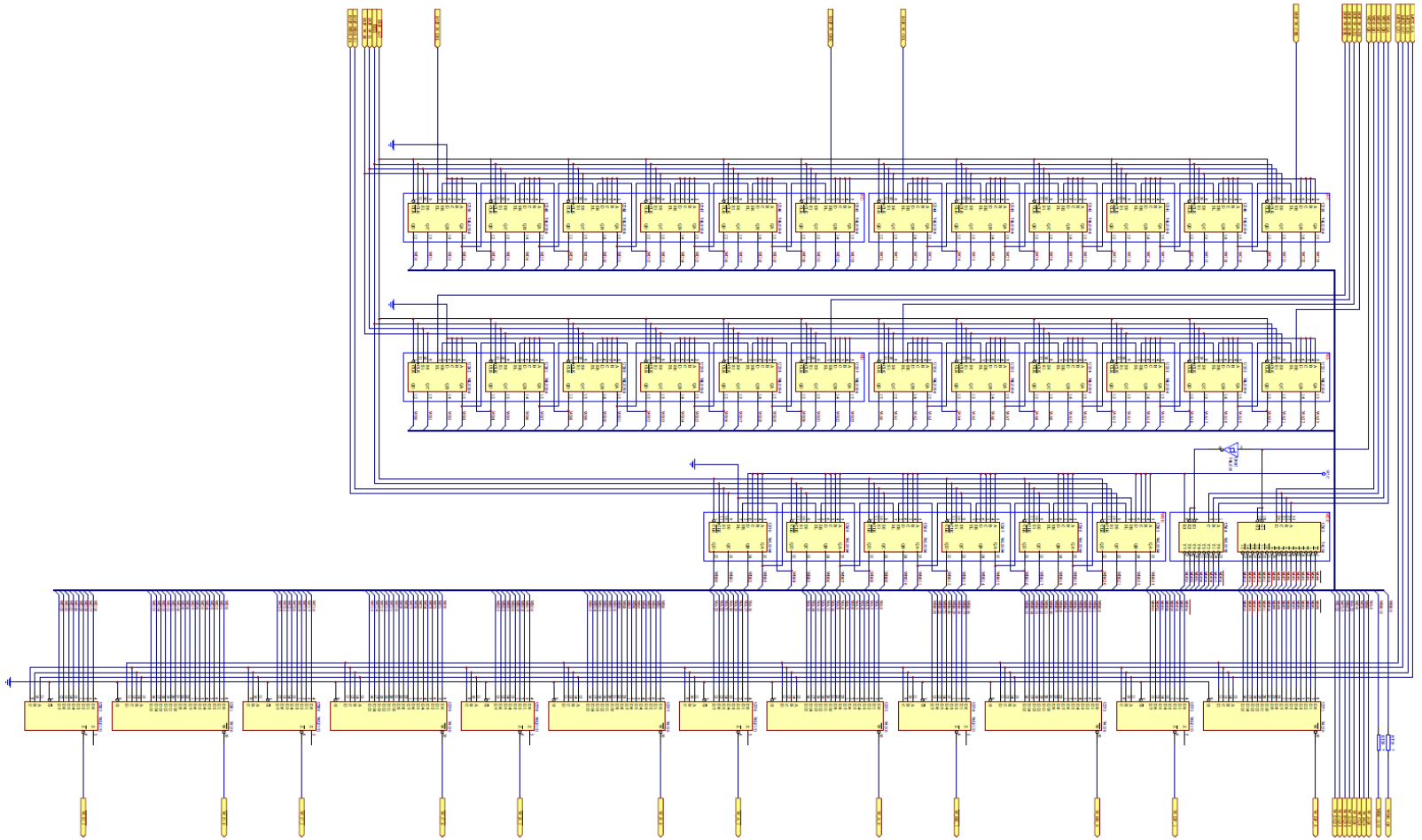
17.96 BITOWY KOMPARATOR



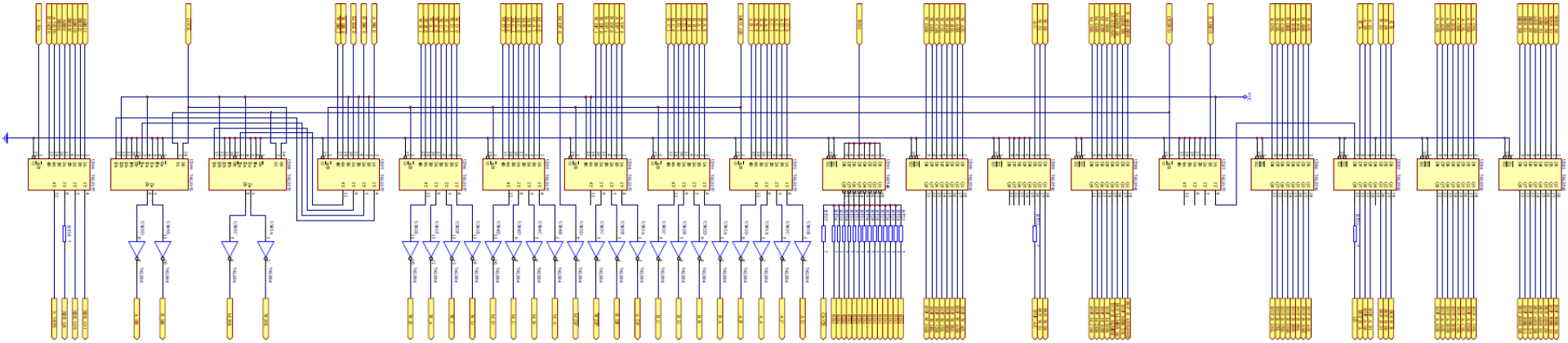
18.ZESPÓŁ REJESTRÓW I DEKODERÓW PREZENTUJĄCYCH WARTOŚĆ LICZBY „M”



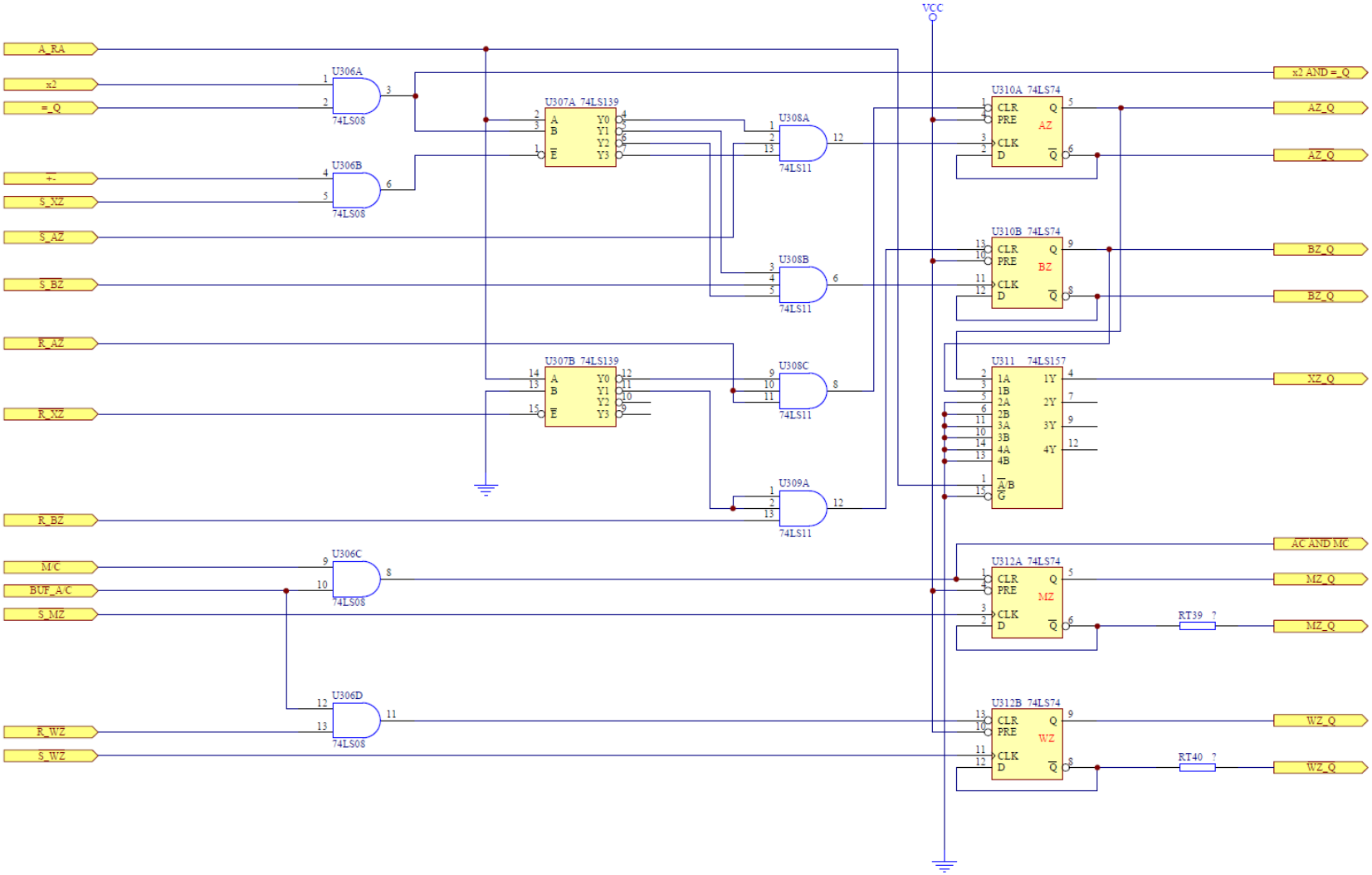
19.ZESPÓŁ REJESTRÓW I DEKODERÓW PREZENTUJĄCYCH WARTOŚĆ LICZBY „W”



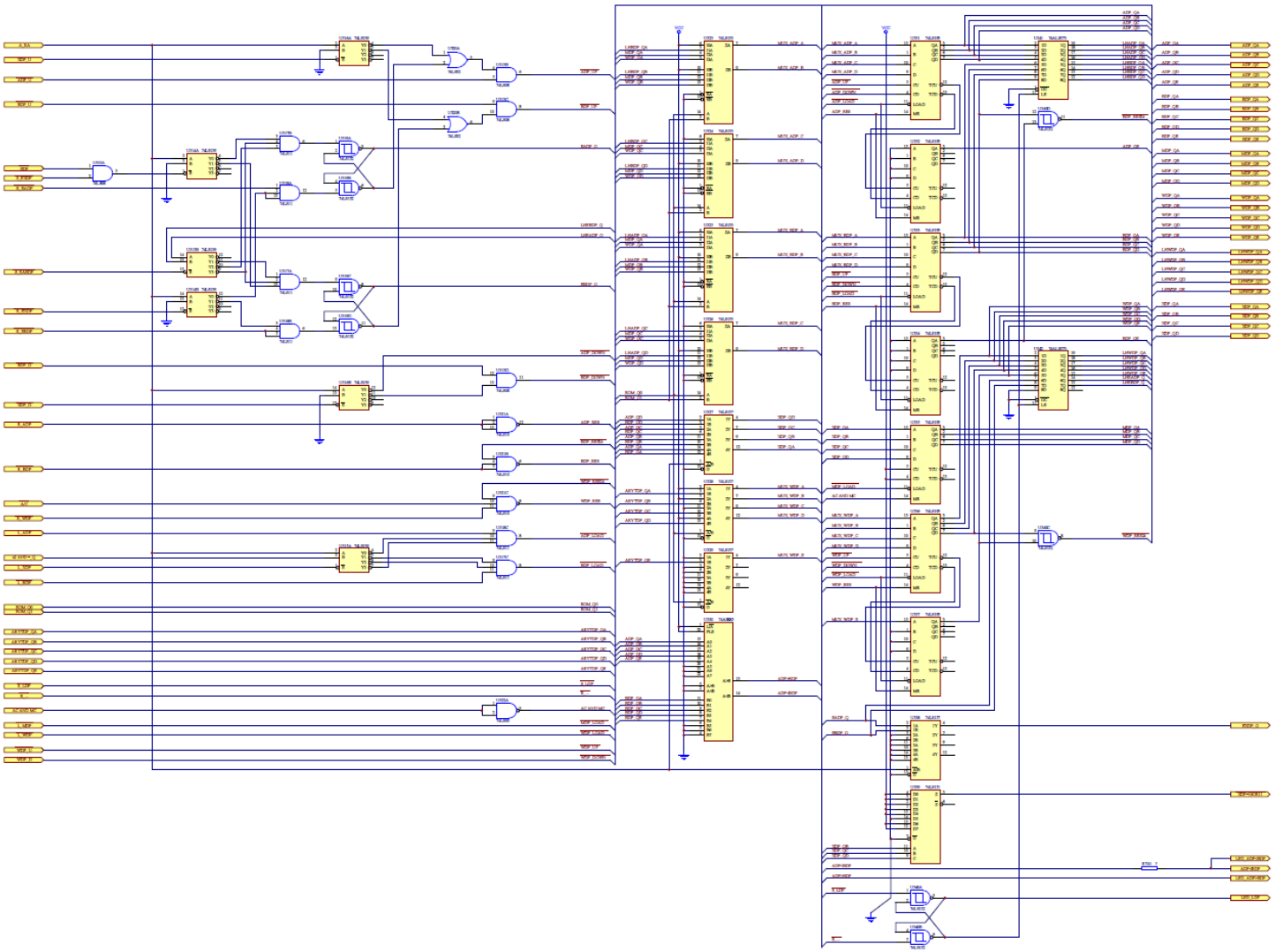
20. BUFORY SYGNAŁÓW WEJŚCIOWYCH DLA ZESPOŁÓW REJESTRÓW, DEKODERÓW ORAZ MULTIPLEKSERY STERUJĄCE ZESPOŁEM WYŚWIETLACZY



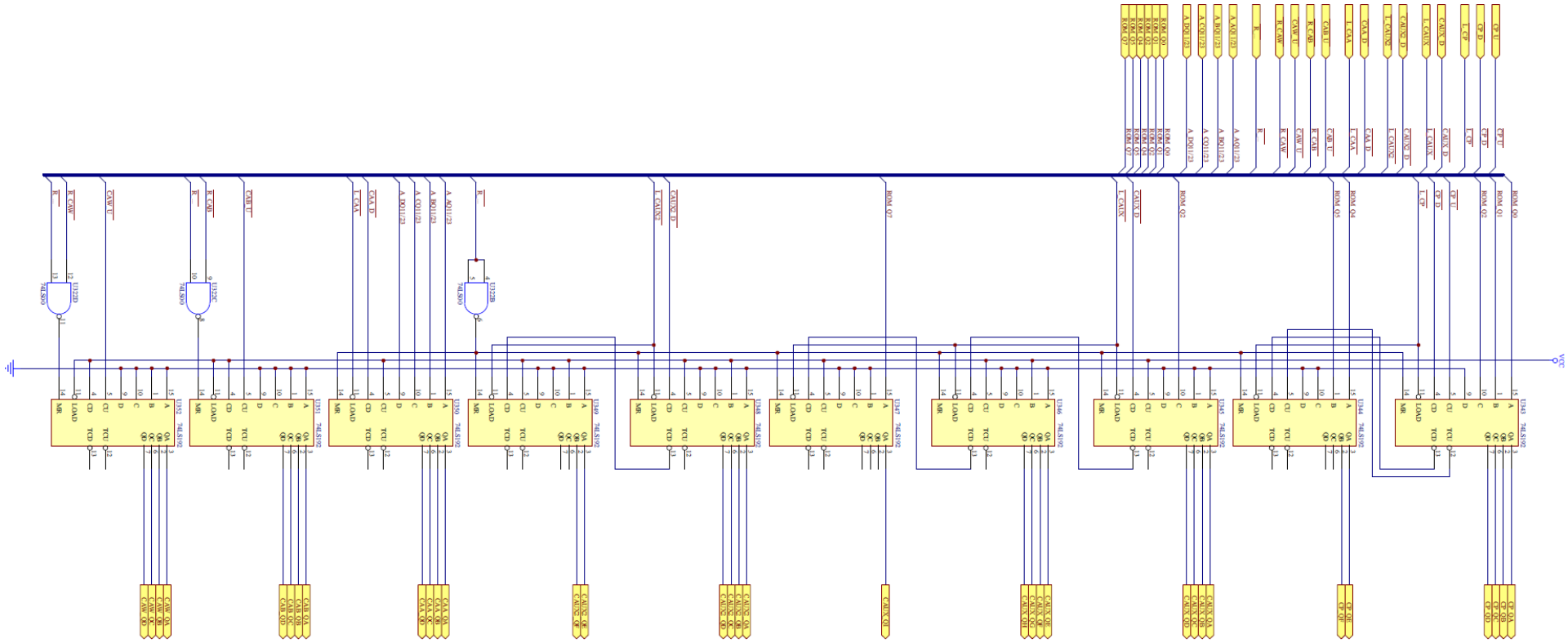
21.STEROWNIK ZNAKÓW WARTOŚCI LICZB



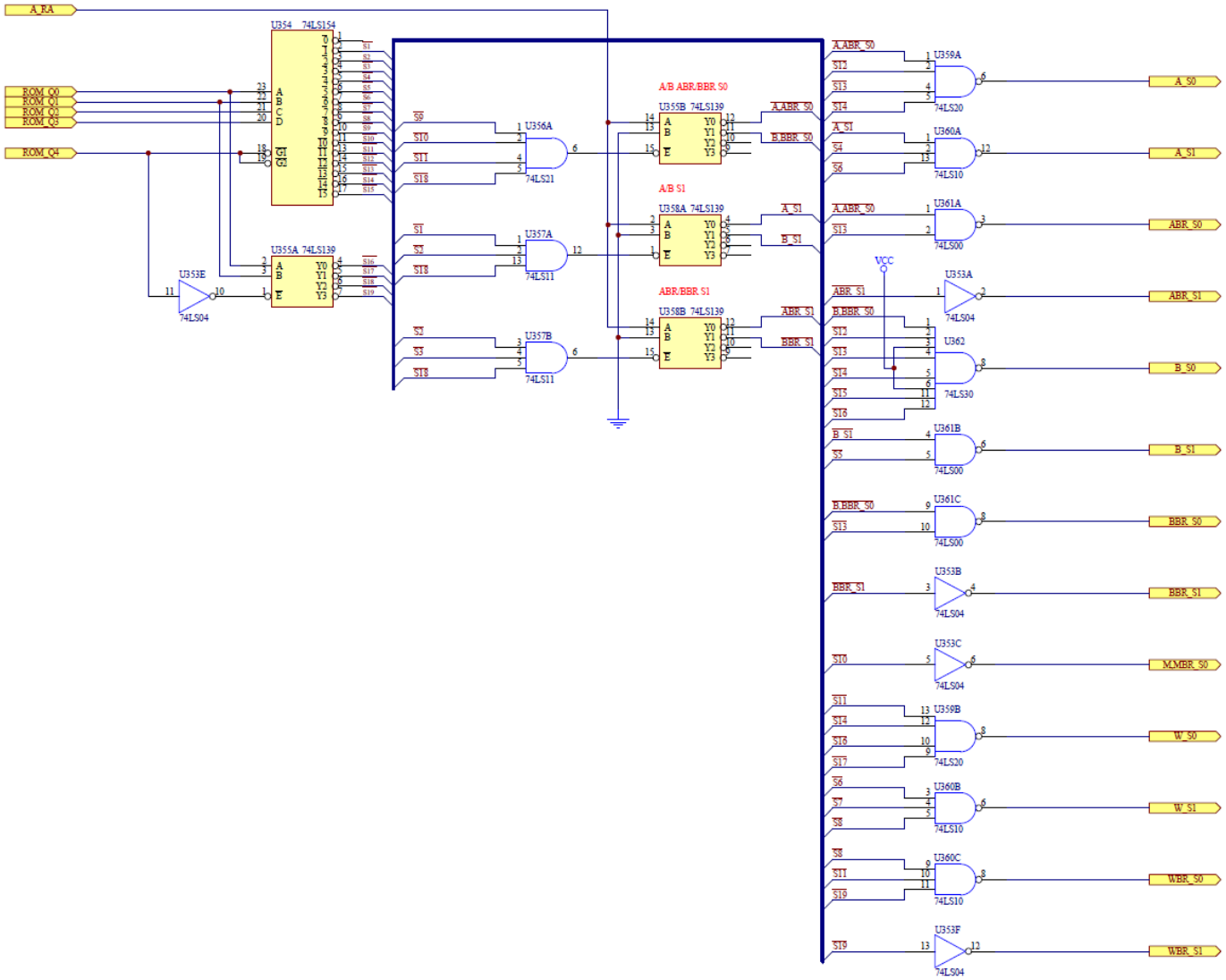
22.STEROWNIK ZESPOŁÓW LICZNIKÓW PUNKTÓW DZIESIĘTNYCH



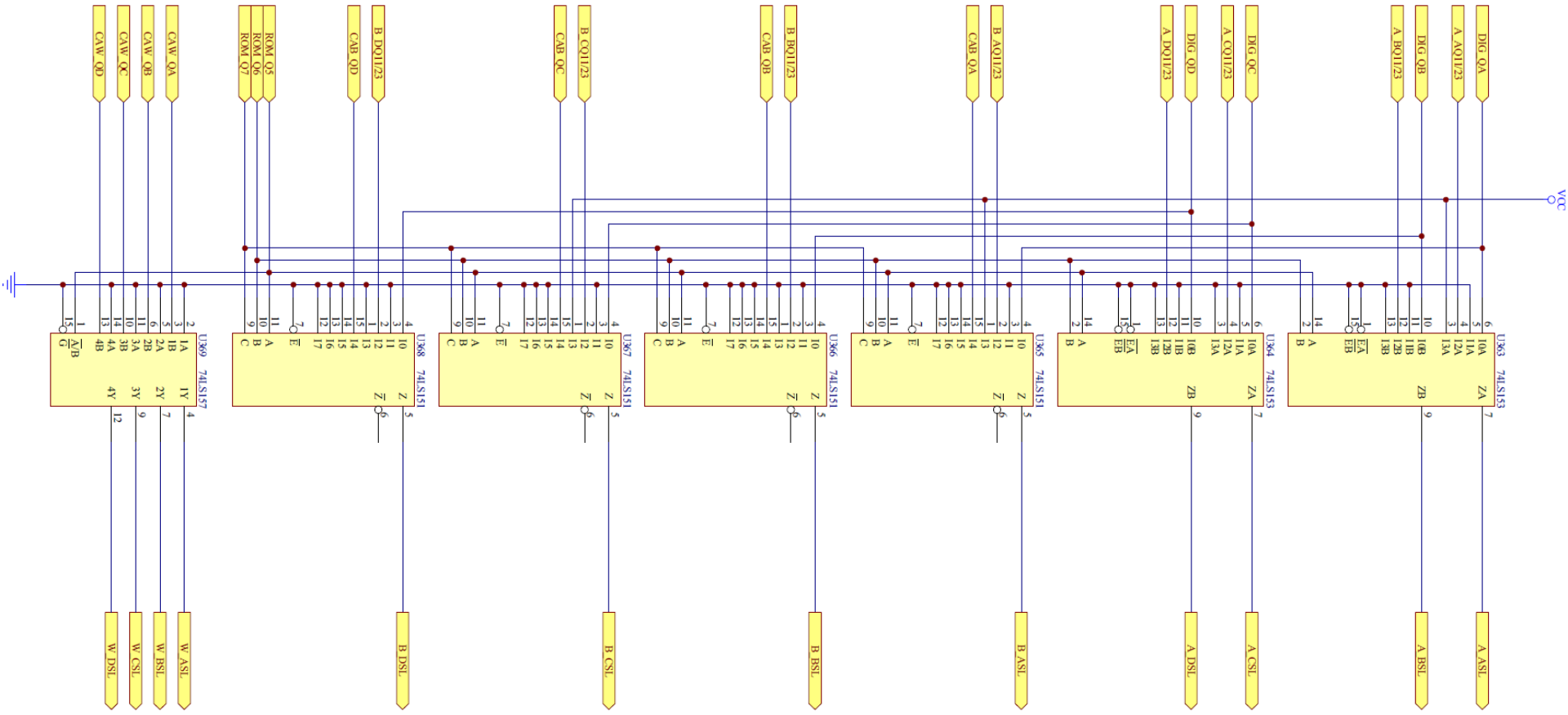
23.ZESPÓŁ LICZNIKÓW SPECJALNYCH



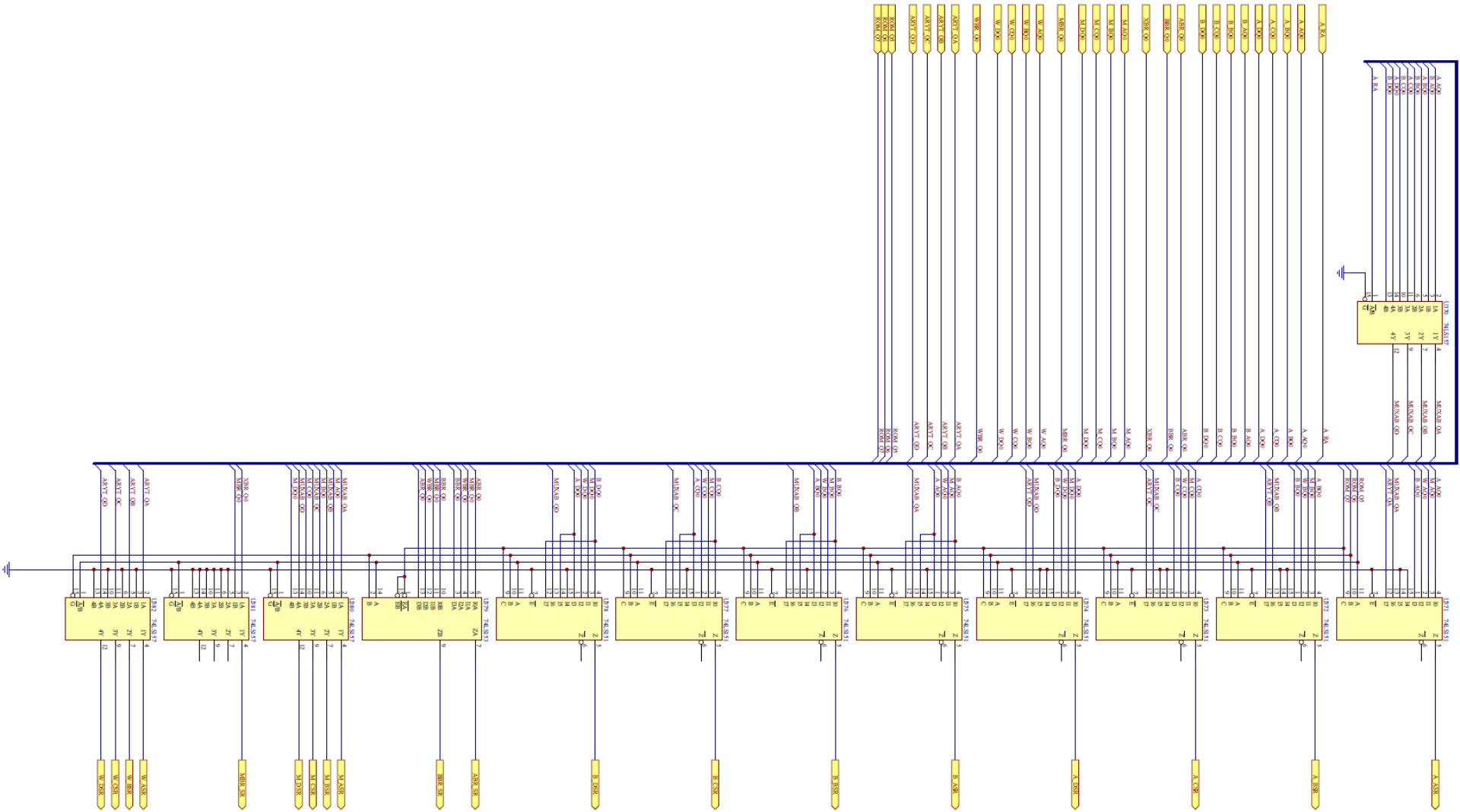
24.STEROWNIK USTAWIANIA TRYBÓW PRACY DLA ZESPOŁU REJESTRÓW



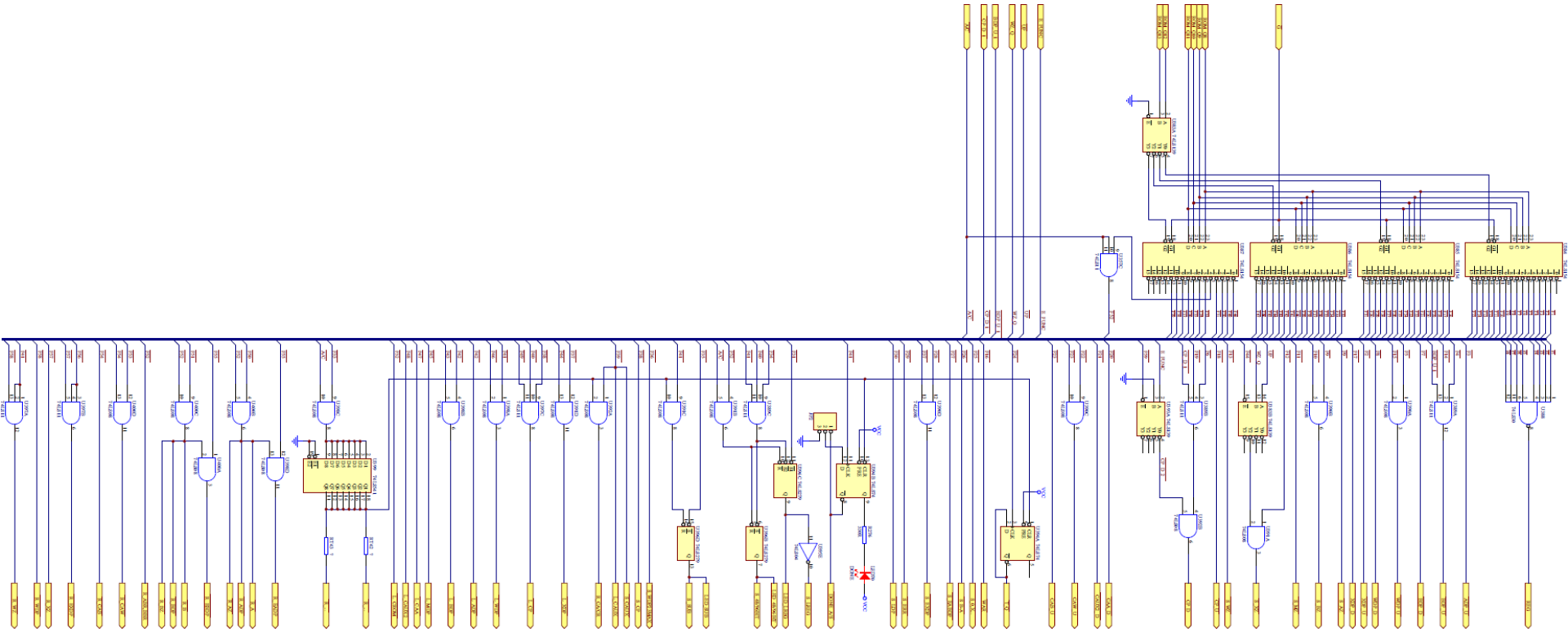
25.MULTIPLEKSERY OBSŁUGUJĄCE DANE DLA PRZESUWU W LEWO W ZESPOŁACH REJESTRÓW



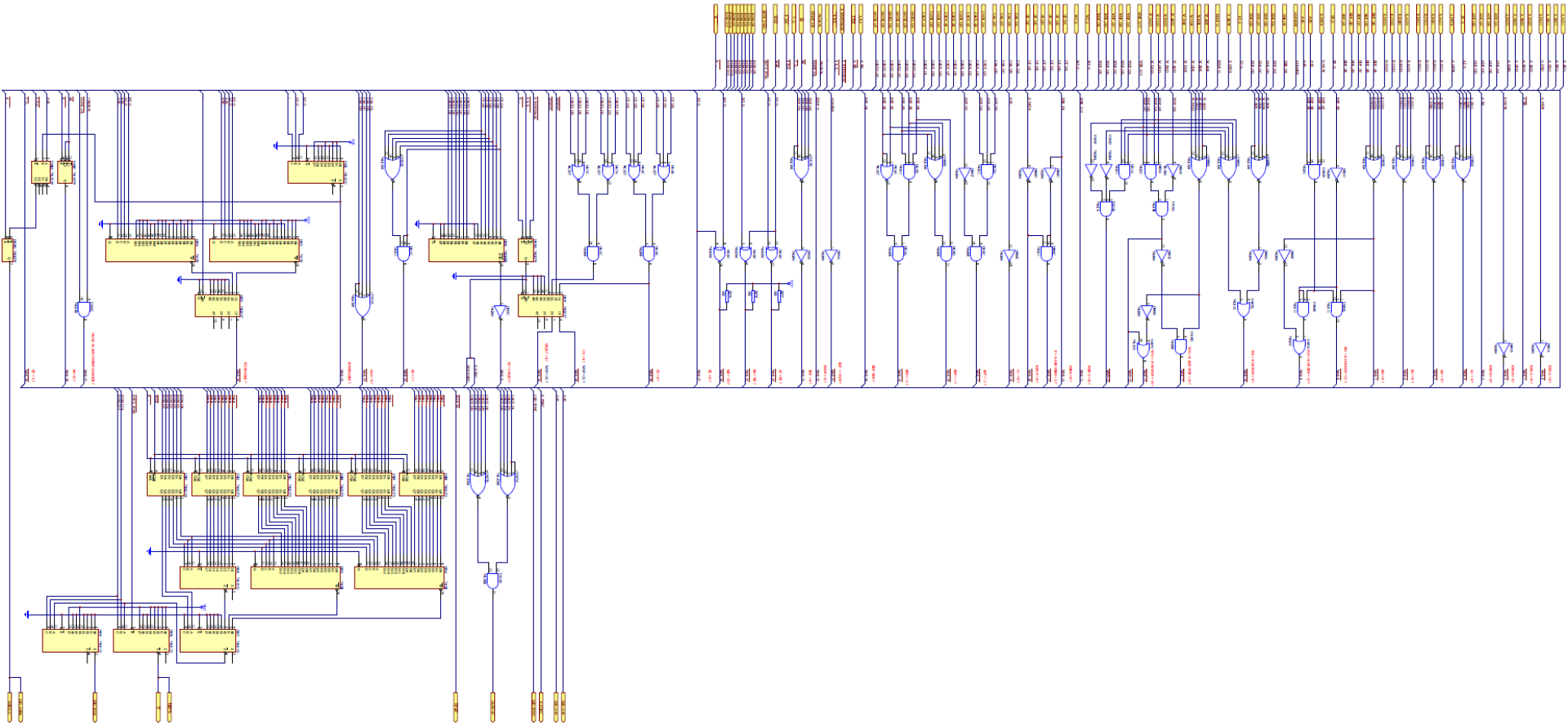
26.MULTIPLEKSERY OBSŁUGUJĄCE DANE DLA PRZESUWU W PRAWO W ZESPOŁACH REJESTRÓW



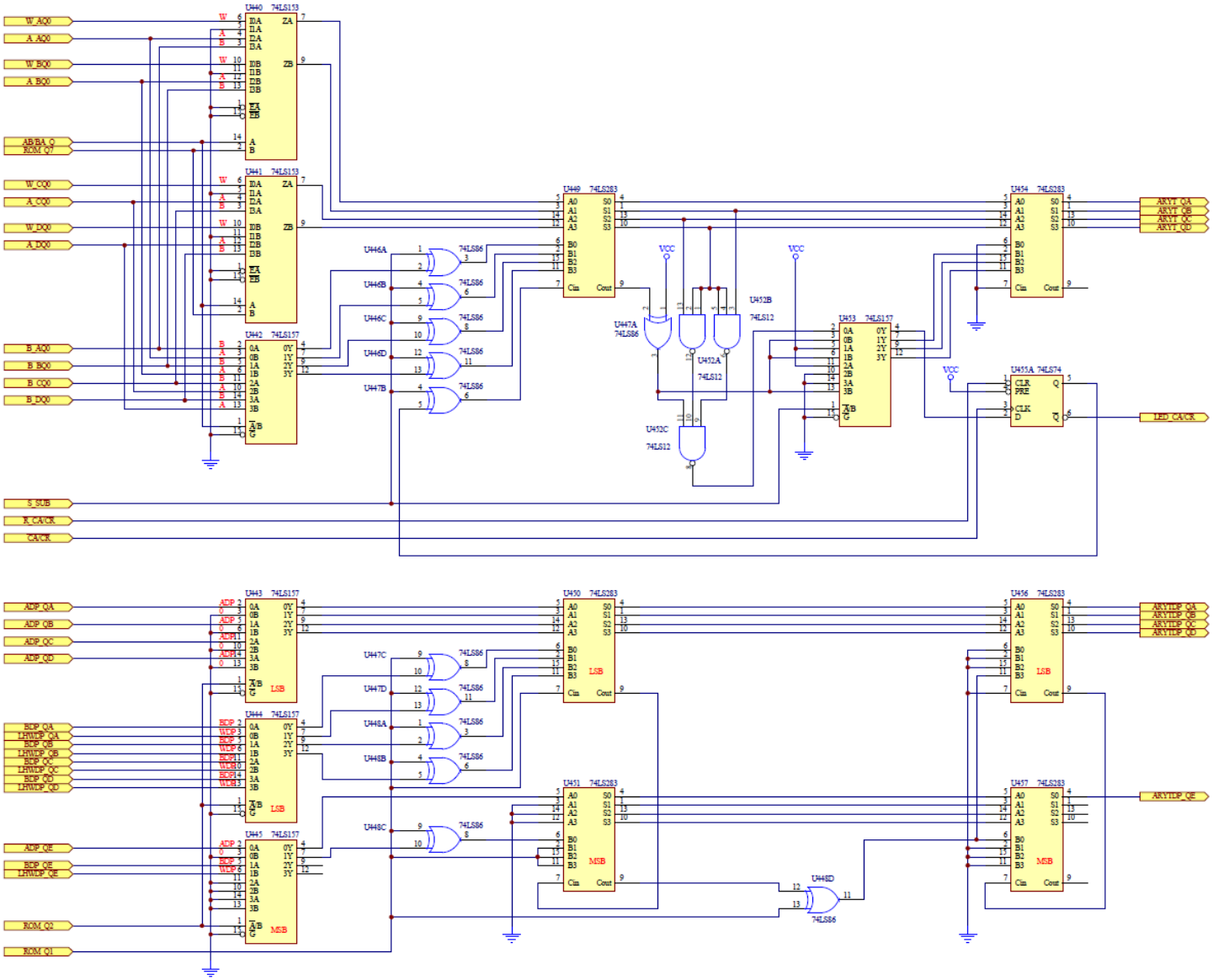
27.DEKODER FUNKCJI WYKONAWCZYCH



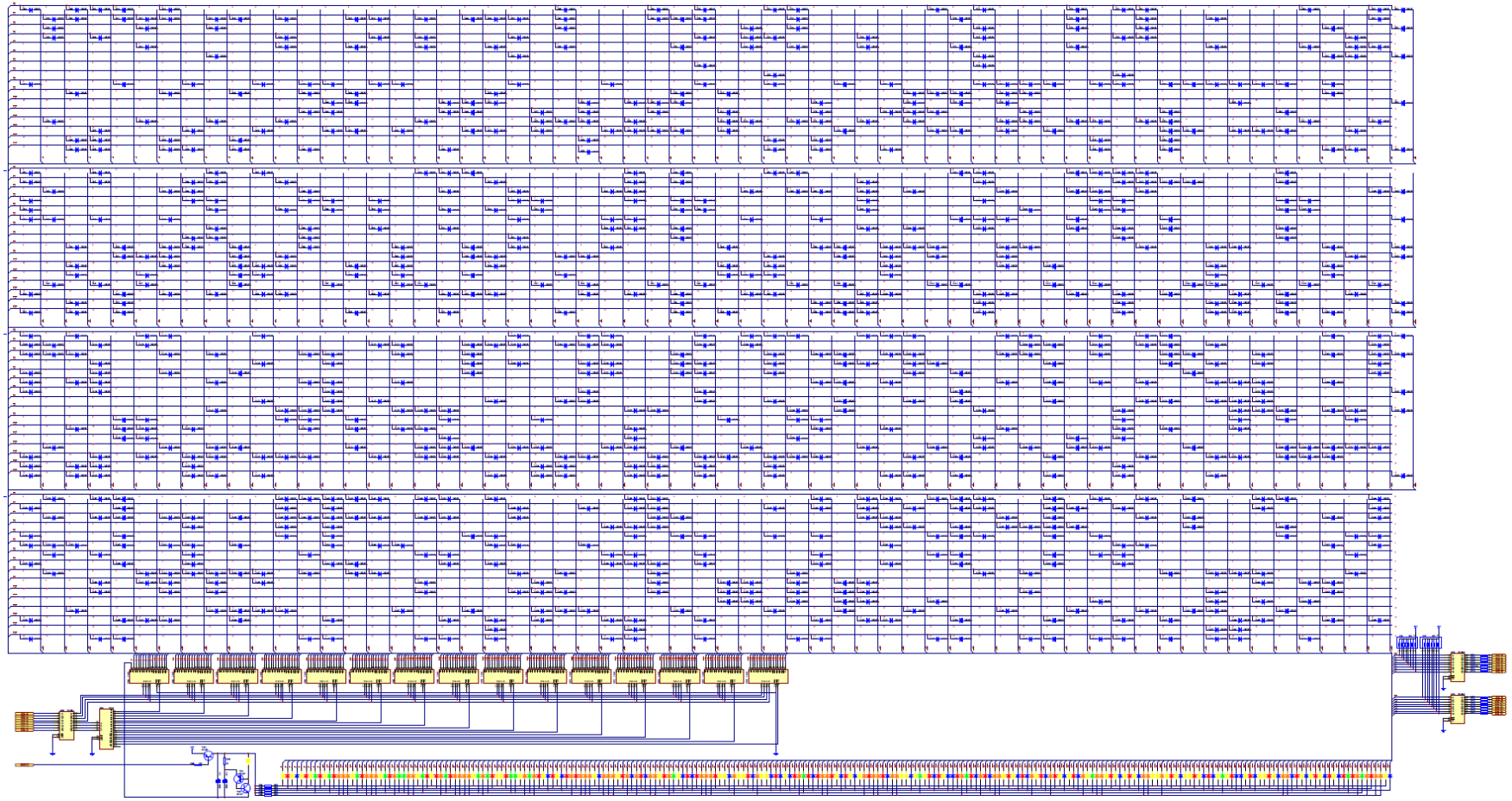
28.SELEKTOR FUNKCJI WARUNKOWYCH



29.4 BITOWY ARYTMOMETR BCD, 5 BITOWY ARYTMOMETR BIN



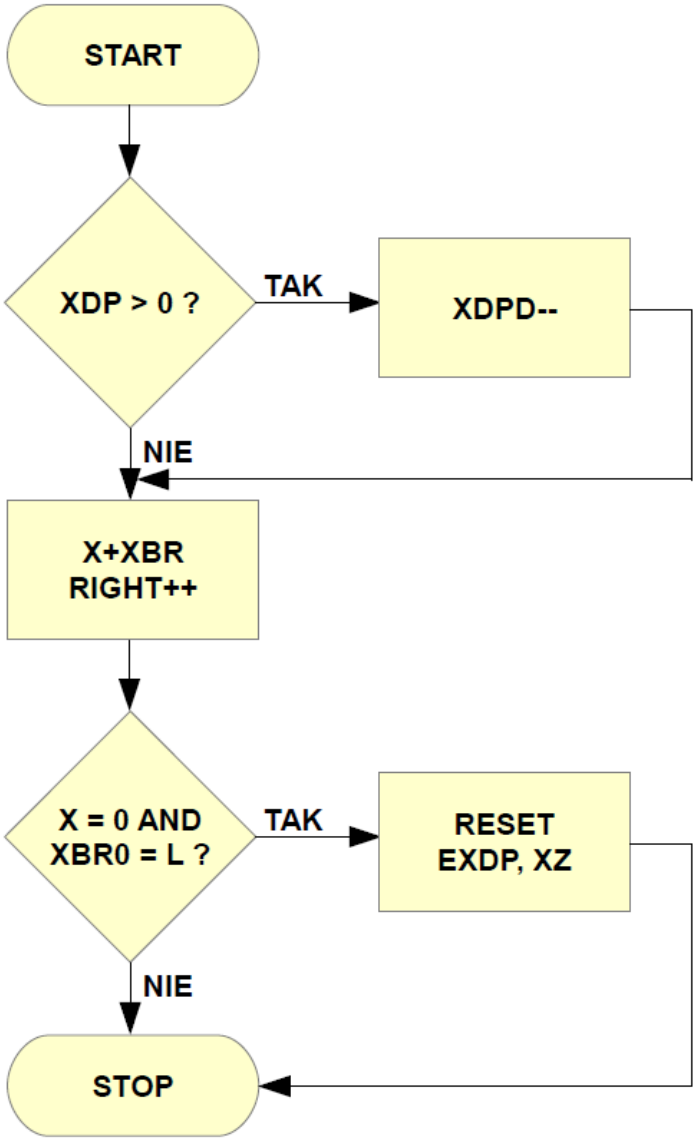
30.3840 BITOWA PAMIĘĆ ROM



PRZYKŁAD PROGRAMÓW ALGORYTMÓW

0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	XLEFT++	A,B: S0=L S1=H SL=DIG SR=X	1	CLK	A,B,M,W,ABR,BBR,MBR,WBR	1	
0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	XBR0 = H ?	FUNKCJA WARUNKOWA	14	CLK	WAR	16	
1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	1	XBRLEFT++ XDPU++	ABR,BBR: S0=L S1=H SL=H SR=X	3	CLK	A,B,M,W,ABR,BBR,MBR,WBR, XDP_U	2	
1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	SKOK	FUNKCJA SKOKU	9	LOAD	CROM	32	
0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	XBRLEFT++	ABR,BBR: S0=L S1=H SL=H SR=X	3	CLK	A,B,M,W,ABR,BBR,MBR,WBR	1	
0	0	0	1	0	0	0	0	0	0	0	1	0	1	1	0	EXDP = H ?	FUNKCJA WARUNKOWA	22	CLK	WAR	16	DIG + 00
1	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	XBRLEFT++ XDPU++	ABR,BBR: S0=L S1=H SL=H SR=X	3	CLK	A,B,M,W,ABR,BBR,MBR,WBR, XDP_U	2	
1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	RESET MASTER	STOP	0	RES	ALL	33	
0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	00 = H ?	FUNKCJA WARUNKOWA	38	CLK	WAR	16	
1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	X+XBRLEFT++ XDPU++	A,B,ABR,BBR: S0=L S1=H SR=X; A,B: SL=DIG; ABR,BBR: SL=H	2	CLK	A,B,M,W,ABR,BBR,MBR,WBR, XDP_U	2	
0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	RESET MASTER	STOP	0	RES	ALL	33	
0	0	0	1	0	0	0	0	0	0	0	1	1	0	1	0	MZ = XZ ?	FUNKCJA WARUNKOWA	26	CLK	WAR	16	
1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	LOAD MZ	FUNKCJA ŁADUJĄCA	0	CLK	MZ	11	
0	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	LOAD XDP TO MDP	FUNKCJA ŁADUJĄCA	0	LOAD	MDP	45	M+
0	0	1	0	0	1	1	0	0	0	0	1	0	0	1	0	LOAD CP=12,SCP	FUNKCJA ŁADUJĄCA	0	SET,LOAD	CP	38	
0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	X+XBR+M+MBR RIGHT++	A,B,ABR,BBR,M,MBR: S0=H S1=L SL=X; A,B,ABR,BBR: SR=Q0; M,MBR: SR=A,ABRQ0 OR B,BBRQ0	4	CLK	A,B,M,W,ABR,BBR,MBR,WBR	1	
0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	RESET MASTER	STOP	0	RES	ALL	33	
0	0	0	1	0	0	0	0	0	0	0	1	1	0	1	0	MZ = XZ ?	FUNKCJA WARUNKOWA	26	CLK	WAR	16	
1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	LOAD XZ	FUNKCJA ŁADUJĄCA	0	CLK	XZ	12	
0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0	LOAD MDP TO XDP	FUNKCJA ŁADUJĄCA	2	LOAD	XDP	44	
0	0	1	0	0	1	1	0	0	0	0	1	0	0	1	0	LOAD CP=12,SCP	FUNKCJA ŁADUJĄCA	0	SET,LOAD	CP	38	
0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	0	X+XBR+M+MBR RIGHT++ [2]	A,B,ABR,BBR,M,MBR: S0=H S1=L SL=X; A,ABR: SR=M,MBRQ0 OR B,BBR: SR=M,MBRQ0; M,MBR: SR=Q0	5	CLK	A,B,M,W,ABR,BBR,MBR,WBR	1	M-
0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	RESET EXDP	FUNKCJA RESETUJĄCA	0	RES	EXDP	56	
0	0	0	1	0	0	0	0	0	0	0	1	0	1	1	1	XDP > 0 ?	FUNKCJA WARUNKOWA	23	CLK	WAR	16	
1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	LOAD EXDP	FUNKCJA ŁADUJĄCA	0	SET	EXDP	28	
0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	RESET MASTER	STOP	0	RES	ALL	33	
0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	RESET A,B,ABR,BBR,EADP,EBDP,ADP,BDP,AZ,BZ,LREG,SET48/96BIT	FUNKCJA RESETUJĄCA	0	RES	A,B,ABR,BBR,ADP,BDP,EADP,EBDP,LATCH REG,48/96BIT,AZ,BZ	52	
0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	1	LOAD WDP TO XDP,XZ	FUNKCJA ŁADUJĄCA	3	LOAD	XDP,XZ	44	
0	0	1	0	0	1	1	0	0	0	0	1	0	0	1	0	LOAD CP=12,SCP	FUNKCJA ŁADUJĄCA	0	SET,LOAD	CP	38	
0	0	0	0	0	0	0	1	0	1	0	0	1	0	1	1	X+XBR+W+WBR RIGHT++	A,B,ABR,BBR,W,WBR: S0=H S1=L SL=X; A,ABR: SR=W,WBRQ0 OR B,BBR: SR=W,WBRQ0; W,WBR: SR=L	6	CLK	A,B,M,W,ABR,BBR,MBR,WBR	1	
0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	RESET WDP,WZ	FUNKCJA RESETUJĄCA	0	RES	WDP,WZ	58	UP
0	0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	REG X = 0 ?	FUNKCJA WARUNKOWA	28	CLK	WAR	16	
1	1	0	0	0	0	0	1	1	0	0	0	1	0	0	1	X+XBR RIGHT++	A,B,ABR,BBR: S0=H S1=L SL=X SR=0	8	CLK	A,B,M,W,ABR,BBR,MBR,WBR	1	
0	0	0	1	0	0	0	0	0	0	0	1	0	1	1	1	XDP > 0 ?	FUNKCJA WARUNKOWA	23	CLK	WAR	16	
1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	LOAD EXDP	FUNKCJA ŁADUJĄCA	0	SET	EXDP	28	
0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	RESET MASTER	STOP	0	RES	ALL	33	
0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	AZ = BZ ?	FUNKCJA WARUNKOWA	24	CLK	WAR	16	
1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	LOAD AZ AND BZ	FUNKCJA ŁADUJĄCA	0	CLK	AZ,BZ	9	
0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	LOAD LATCH DP	FUNKCJA ŁADUJĄCA	0	SET	LDP	30	
0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	1	LOAD ADP AND BDP	FUNKCJA ŁADUJĄCA	1	LOAD	ADP,BDP	42	
0	0	1	0	0	1	1	0	0	0	0	1	0	0	1	0	LOAD CP=12,SCP	FUNKCJA ŁADUJĄCA	0	SET,LOAD	CP	38	XY
0	0	0	0	0	0	0	1	0	1	1	0	1	1	0	1	A+ABR+B+BBR RIGHT++	A,B,ABR,BBR: S0=H S1=L SL=X; A,ABR: SR=B,BBRQ0; B,BBR: SR=A,ABRQ0	7	CLK	A,B,M,W,ABR,BBR,MBR,WBR	1	
0	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0	RESET EABDP	FUNKCJA RESETUJĄCA	0	RES	EABDP	55	
0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	LOAD EABDP	FUNKCJA ŁADUJĄCA	0	SET	EABDP	27	
0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	RESET MASTER	STOP	0	RES	ALL	33	
0	0	0	1	0	0	0	0	0	0	0	1	0	1	1	1	XDP > 0 ?	FUNKCJA WARUNKOWA	23	CLK	WAR	16	
1	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	XDPD--	INKREMENTACJA XDPD	0	CLK	XDP_D	17	
0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	1	X+XBR RIGHT++	A,B,ABR,BBR: S0=H S1=L SL=X SR=0	8	CLK	A,B,M,W,ABR,BBR,MBR,WBR	1	BK
0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	1	X = 0 AND XBR0 = L ?	FUNKCJA WARUNKOWA	15	CLK	WAR	16	
1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	RESET EXDP,XZ	FUNKCJA RESETUJĄCA	0	RES	EXDP,XZ	57	
0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	RESET MASTER	STOP	0	RES	ALL	33	
0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	X+XBR LOAD	A,B,ABR,BBR: S0=H S1=H SL=X SR=X	9	CLK	A,B,M,W,ABR,BBR,MBR,WBR	1	
0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	LOAD EXDP,XDP	FUNKCJA ŁADUJĄCA	0	SET,LOAD	EXDP,XDP	37	PI
0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	RESET MASTER	STOP	0	RES	ALL	33	
0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	RES SUB,WZ; LOAD D_A,S,LREG,SET 48/96BIT,WDP=11	FUNKCJA ŁADUJĄCA	0	RES,SET,LOAD	SUB,WZ,DONE_A/S,LREG,48/96BIT,WDP	41	
0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	ADP = 11 ?	FUNKCJA WARUNKOWA	17	CLK	WAR	16	
1	0	0	0	0	0	1	1	0	0	1	0	0	1	0	0	ALEFT++ ADPU++	A: S0=L S1=H SL=L SR=X	16	CLK	A,B,M,W,ABR,BBR,MBR,WBR,ADP_U	3	
1	0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	SKOK	FUNKCJA SKOKU	56	LOAD	CROM	32	
0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	1	BDP = 11 ?	FUNKCJA WARUNKOWA	19	CLK	WAR	16	
1	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	BLEFT++ BDPU++	B: S0=L S1=H SL=L SR=X	20	CLK	A,B,M,W,ABR,BBR,MBR,WBR,BDP_U	4	
1	0	1	0	0	0	0	0	0	0	1	1	1	0	1	1	SKOK	FUNKCJA SKOKU	59	LOAD	CROM	32	
0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	WZ ENABLE ?	FUNKCJA WARUNKOWA	36	CLK	WAR	16	ADD / SUB
1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	LOAD WZ	FUNKCJA ŁADUJĄCA	0	CLK	WZ	13	
0	0	0	1	0	0	0	0	0	0	1	0	0	1	1	1	SUB ENABLE ?	FUNKCJA WARUNKOWA	35	CLK	WAR	16	

DIAGRAM ALGORYTMU BACKSPACE



ALGORYTM BK

FUNKCJA:

Algorytm cofa zapisaną zawartość aktywnego rejestru: A lub B o jedno miejsce.

Algorytm składa się z 6 instrukcji.

Warunki nadrzędne:

Aby można było wykonać operację **BK** z klawiatury kalkulatora musi zostać spełniona następująca zależność:
gdy aktywny rejestr wartości jest równy zero musi być odblokowany punkt dziesiętny aktywnego zespołu rejestrów, przy zawartości rejestru wartości większej od zera nie jest istotne czy punkt dziesiętny jest odblokowany czy zablokowany.

ADRES BIN	Q15	Q14	Q13	Q12	Q11	Q10	Q9	Q8	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0	INSTRUKCJA
46	0	0	0	1	0	0	0	0	0	0	0	1	0	1	1	1	XDP > 0 ?
47	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	XDPD--
48	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	1	X+XBR RIGHT++
49	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	X = 0 AND XBR0 = L ?
50	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	RESET EXDP,XZ
51	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	RESET MASTER

Prześledźmy działanie algorytmu na poniższych przykładach:

Czarna czcionka dotyczy: wygaszonych cyfr w rejestrach, zablokowanych punktów dziesiętnych, nieaktywnego zespołu rejestrów, znaku „minus", oraz wartości zmiennych, które nie są aktywne w rozpatrywanym przykładzie.
Niebieska czcionka obrazuje pozycję neutralną (zawsze wyświetlane zero na najmłodszych wyświetlaczach rejestrów: A i B).
Czerwona czcionka dotyczy: podświetlonych cyfr w rejestrach, odblokowanych punktów dziesiętnych, aktywnego zespołu rejestrów, znaku „minus"oraz wartości zmiennych, które są aktywne w rozpatrywanym przykładzie.

Wartości początkowe:

1

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A	–	0	0	0	1	5	2	6	9	6	9	5	5.	0
B		0	0	0	0	0	0	0	0	0	0	0	0.	0

2

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A		0	0	5	6	8	8	8	5	5	5.	6	6	1
B		0	0	0	0	0	0	0	0	0	0	0	0.	1

3

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A	–	0	0	0	0	0	0	0	0	0	0	0	6.	1
B	–	0	0	2	5	6	8	9	9	9	9	0	0.	0

4

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A		0	0	0	0	0	0	0	0	0	0	0	0.	0
B	–	0	0	0	0	0	0	0	0	0	0	0	3.	0

5

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A		5	8	6.	3	2	5	0	5	8	6	2	2	1
B		0	0	0	0	5	5	6	9.	6	6	6	3	1

6

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A		0	0	0	0	0	0	0	0	0	0	0	2.	0
B	–	0	0	0	0	0	0	0	0	0	0	1.	3	1

(46) XDP > 0 ?

Pierwszą instrukcją algorytmu jest instrukcja warunkowa, która bada czy aktywny punkt dziesiętny znajduje na pozycji większej niż zero. Gdy warunek jest prawdą świadczy to o tym iż aktywny punkt dziesiętny znajduje się na pozycji większej niż zero. Konsekwencją tego jest wykonanie kolejnej instrukcji:

(47) XDPD--

Instrukcja ta aktywuje funkcję **XDP_D**, która dekrementuje o jeden wartość aktywnego punktu dziesiętnego.

Wartości po wykonaniu instrukcji, dla przykładów spełniających warunek pozytywnie:

5

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A		5	8	6	3.	2	5	0	5	8	6	2	2	1
B		0	0	0	0	5	5	6	9.	6	6	6	3	1

6

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A		0	0	0	0	0	0	0	0	0	0	0	2.	0
B	–	0	0	0	0	0	0	0	0	0	0	1	3.	1

Gdy warunek jest fałszem powyższa instrukcja nie jest w ogóle przetwarzana.

(48) X+XBR RIGHT++

Instrukcja ta powoduje przesunięcie w prawo o jedną pozycję: zawartości aktywnego rejestru wartości i aktywnego rejestru podświetlania.

A,B,ABR,BBR: S0=H S1=L SL=X SR=0

Aktywny rejestr wartości oraz podświetlania zostaje ustawiony wg powyższej zależności: wybrany tryb przesuwu danych w prawo poprzez odpowiedni stan logiczny wejść ustawiania: S0 i S1, wejścia szeregowe dla przesuwu danych w prawo znajdują się w niskim stanie logicznym (dla aktywnego rejestru zawartości odzwierciedleniem tego stanu rzeczy jest cyfra zero), stan logiczny wejść szeregowych dla przesuwu danych w lewo jest nieistotny.

A,B,M,W,ABR,BBR,MBR,WBR

Zdekodowana instrukcja wywoła funkcję REG, narastające zbocze tego sygnału spowoduje przesunięcie danych w prawo w aktywnym rejestrze wartości oraz przesunięcie statusu podświetlania w aktywnym rejestrze podświetlania.

Wartości po wykonaniu instrukcji:

1

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A	–	0	0	0	0	1	5	2	6	9	6	9	5.	0
B		0	0	0	0	0	0	0	0	0	0	0	0.	0

2

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A		0	0	5	6	8	8	8	5	5	5.	6	6	1
B		0	0	0	0	0	0	0	0	0	0	0	0.	1

3

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A	–	0	0	0	0	0	0	0	0	0	0	0	0.	1
B	–	0	0	2	5	6	8	9	9	9	9	0	0.	0

4

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A		0	0	0	0	0	0	0	0	0	0	0	0.	0
B	–	0	0	0	0	0	0	0	0	0	0	0	0.	0

5

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A		0	5	8	6.	3	2	5	0	5	8	6	2	1
B		0	0	0	0	5	5	6	9	6.	6	6	3	1

6

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A		0	0	0	0	0	0	0	0	0	0	0	2.	0
B	–	0	0	0	0	0	0	0	0	0	0	0	1.	1

(49) X = 0 AND XBR0 = L ?

Kolejną instrukcją algorytmu jest instrukcja warunkowa, która bada czy zawartość aktywnego rejestru wartości jest równa zero oraz czy status wyjścia Q0 aktywnego rejestru podświetlania znajduje się w stanie niskim. Gdy warunek jest prawdą świadczy to o tym iż, aktywny zespół rejestrów znajduje się n neutralnej pozycji. Konsekwencją tego jest wykonanie kolejnej instrukcji:

(50) RESET EXDP,XZ

Instrukcja ta aktywuje dwie funkcję: **R_EXDP** i **R_XZ**, które załączają ewentualną blokadę licznikowi aktywnego punktu dziesiętnego oraz resetują ewentualnie znak „minus" aktywnego zespołu rejestrów gdy ten jest aktywny.

Wartości po wykonaniu instrukcji, które są ostateczne:

2

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A		0	0	5	6	8	8	8	5	5	5.	6	6	1
B		0	0	0	0	0	0	0	0	0	0	0	0.	0

3

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A		0	0	0	0	0	0	0	0	0	0	0	0.	0
B	–	0	0	2	5	6	8	9	9	9	9	0	0.	0

4

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A		0	0	0	0	0	0	0	0	0	0	0	0.	0
B		0	0	0	0	0	0	0	0	0	0	0	0.	0

Gdy warunek jest fałszem powyższa instrukcja nie jest w ogóle przetwarzana i poniższe wartości są ostateczne:

1

REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A	–	0	0	0	0	1	5	2	6	9	6	9	5.	0
B		0	0	0	0	0	0	0	0	0	0	0	0.	0

5

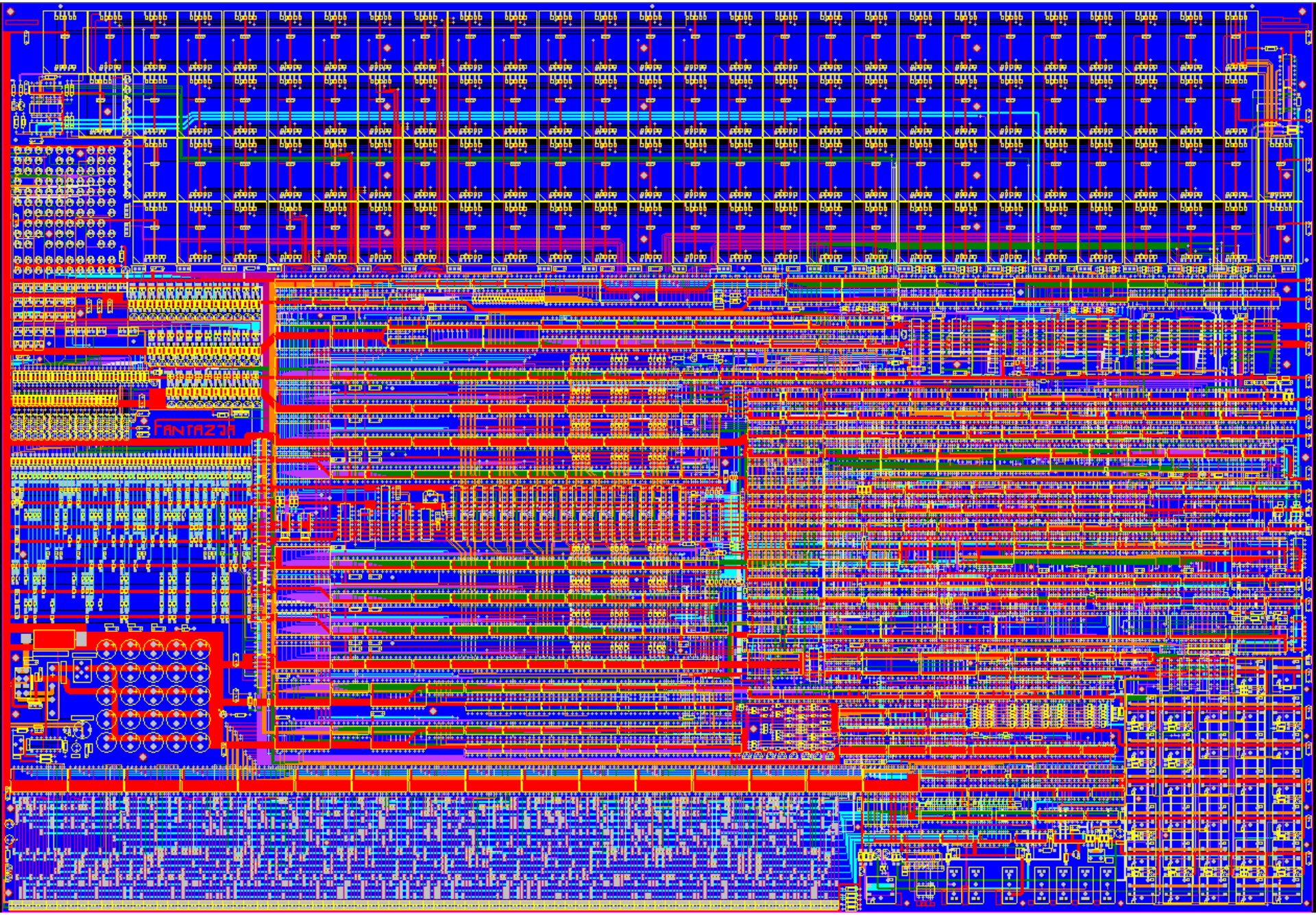
REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A		0	5	8	6.	3	2	5	0	5	8	6	2	1
B		0	0	0	0	5	5	6	9	6.	6	6	3	1

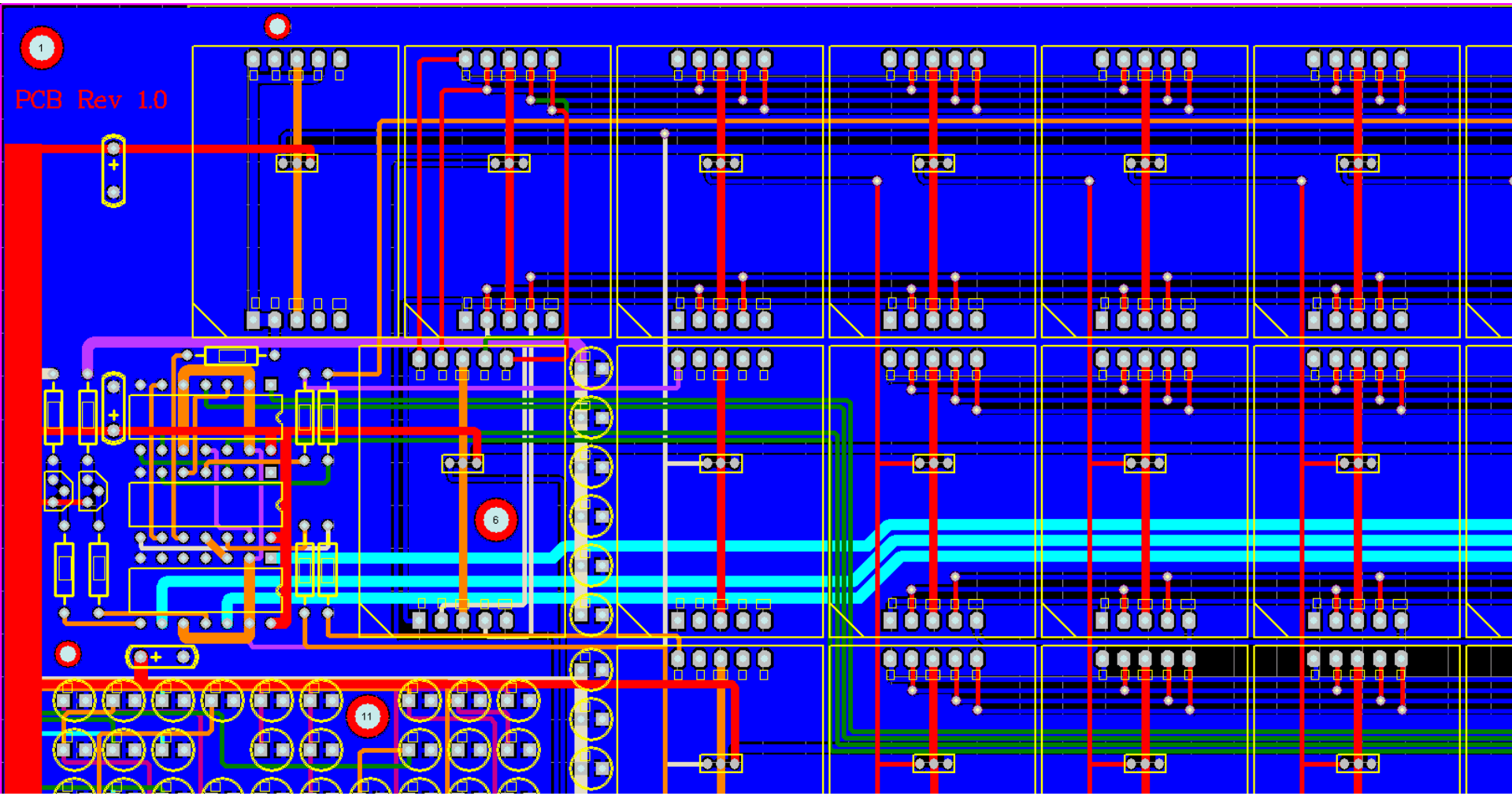
6

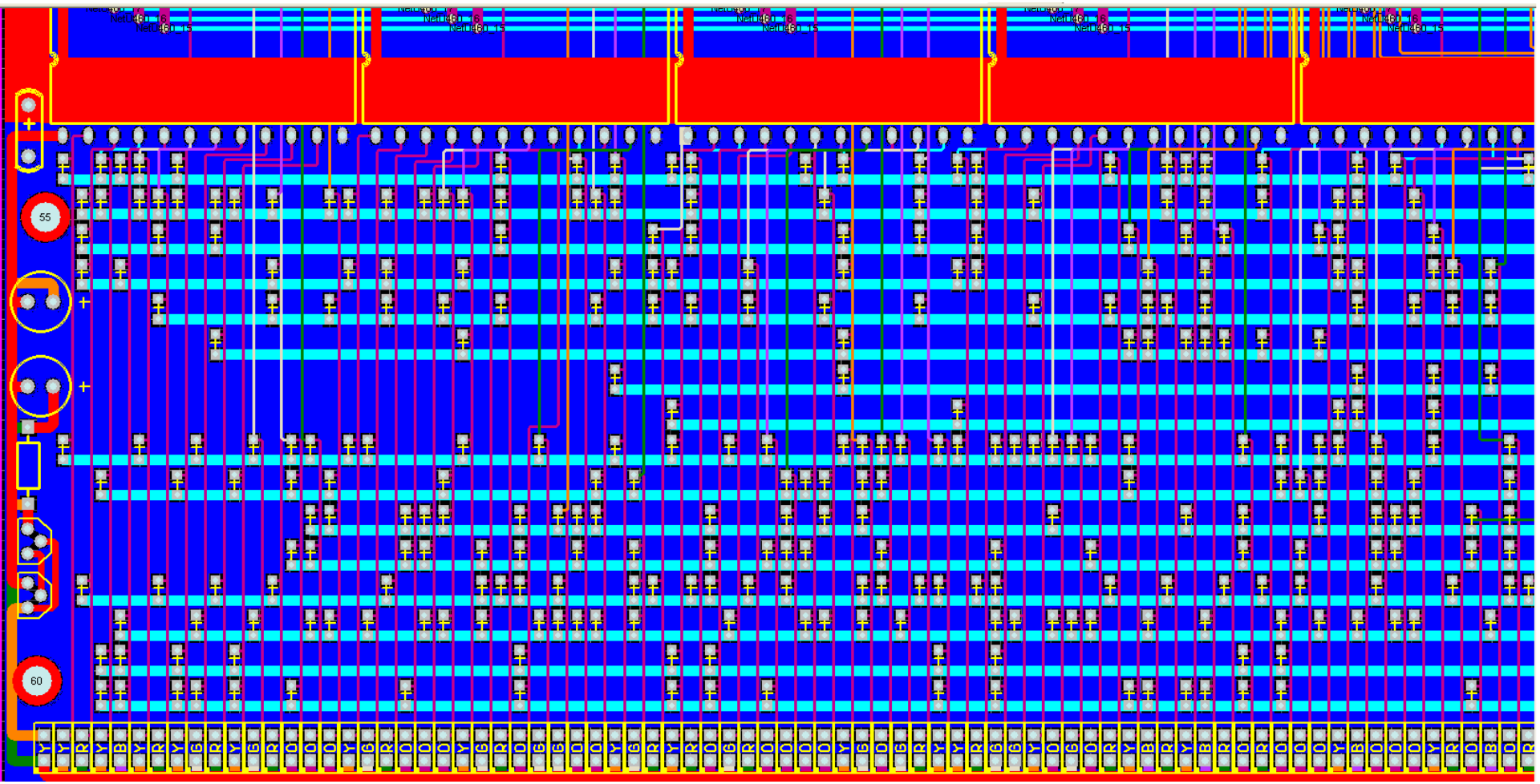
REJESTR	ZNAK	11	10	9	8	7	6	5	4	3	2	1	0	EXDP
A		0	0	0	0	0	0	0	0	0	0	0	2.	0
B	–	0	0	0	0	0	0	0	0	0	0	0	1.	1

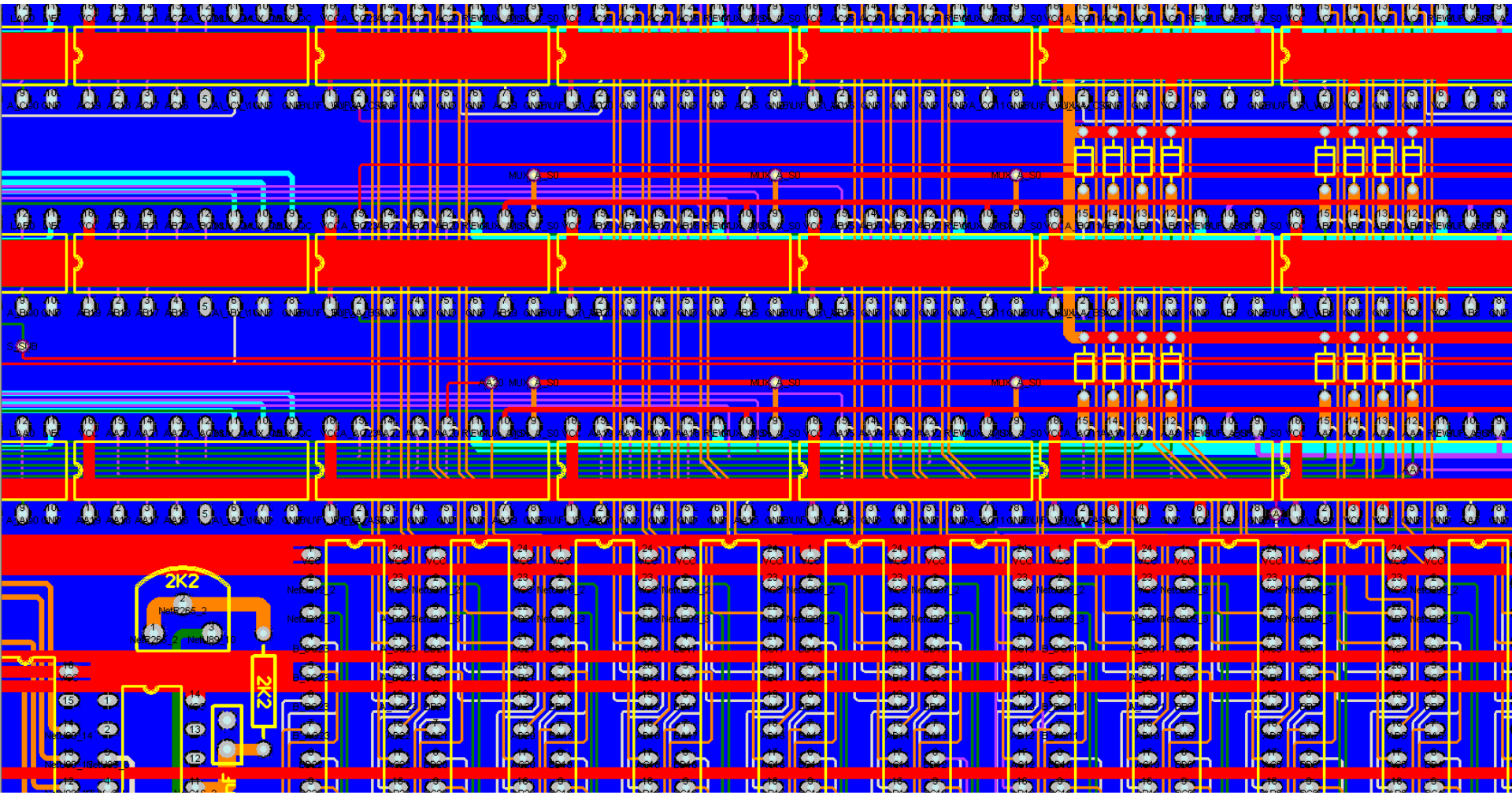
(51) RESET MASTER

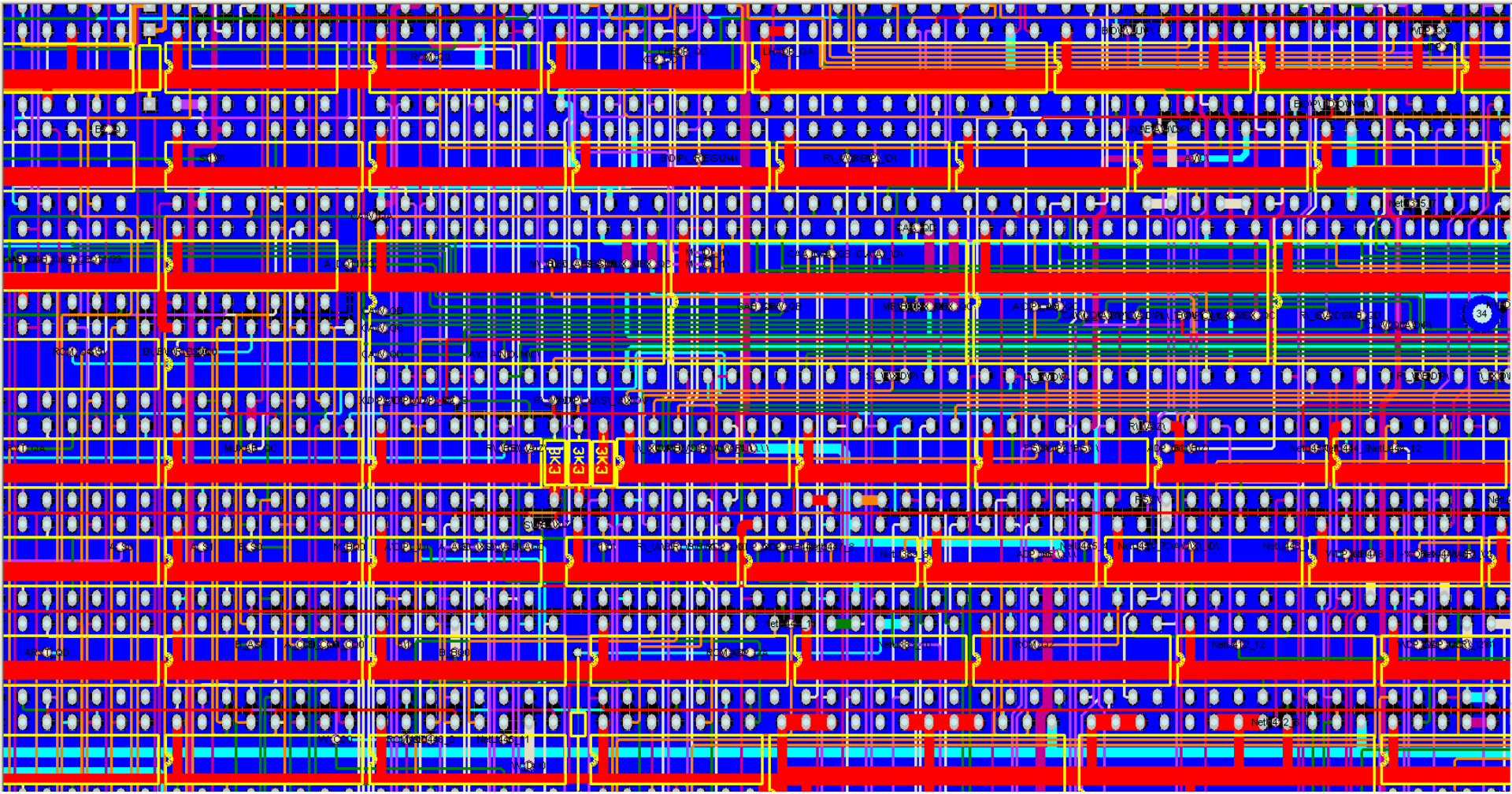
Instrukcja ta aktywuje funkcję R_..., która zakańcza algorytm (następuje resetowanie obwodów, które biorą udział w przetwarzaniu algorytmów, zostaje również ustawiony adres pamięci ROM na pozycję początkową).


FANTAZJA


PCB Rev 1.0


55
60


2K2
2K2


8k2
3k3
3k3

NOWY CEL

KOMPUTER NA UKŁADACH TTL

NOWY CEL

KOMPUTER NA UKŁADACH TTL

NOWY CEL

KOMPUTER NA UKŁADACH TTL

MMX™ Technology Overview

MMX™ Instruction Set Summary

intel

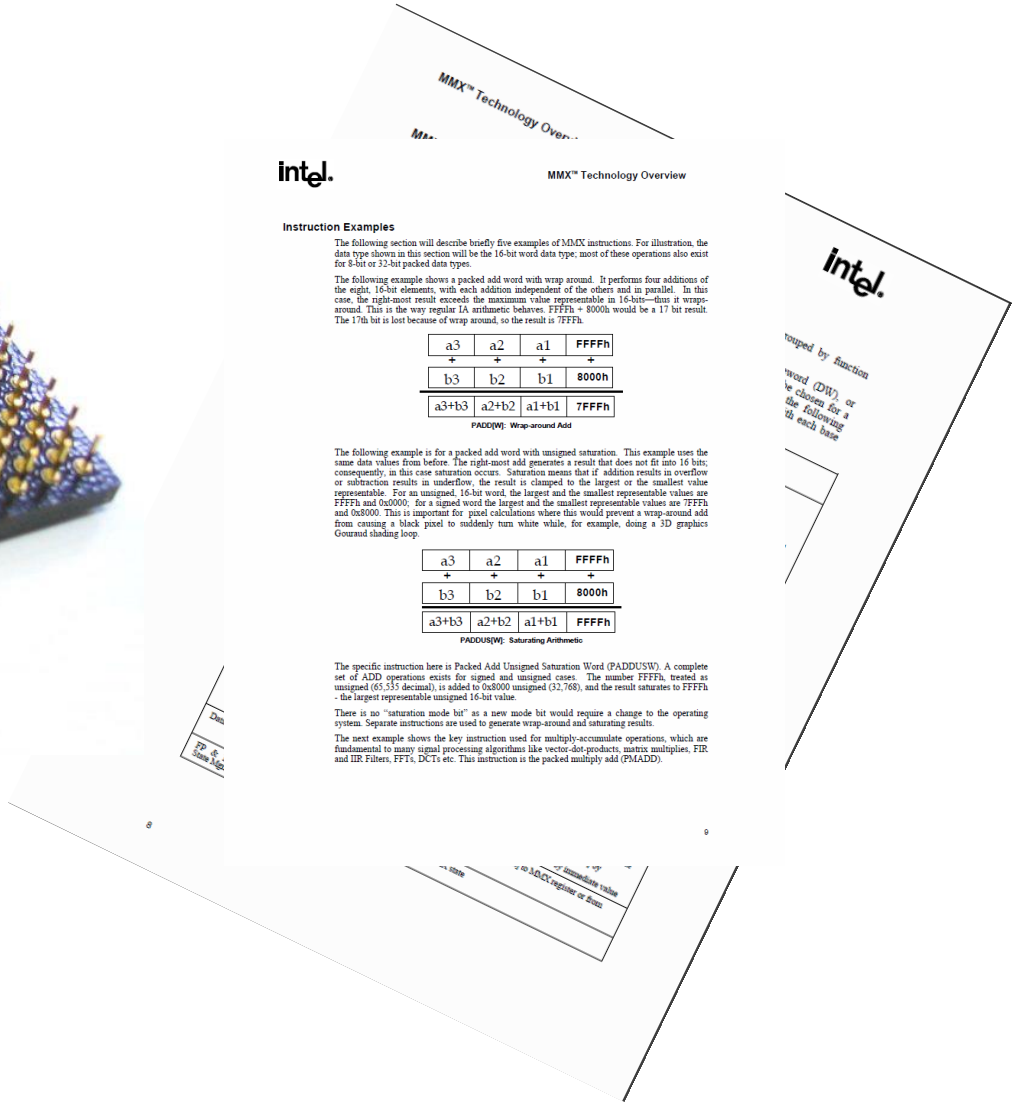
The instructions and corresponding mnemonics in the table below are grouped by function categories.

If an instruction supports multiple data types—byte (B), word (W), doubleword (DW), or quadword (QW), the datatypes are listed in brackets. Only one data type may be chosen for a given instruction. For example, the base mnemonic PADD (packed add) has the following variations: PADDB, PADDW, and PADDD. The number of opcodes associated with each base mnemonic is listed.

Category	Mnemonic	Number of Different Opcodes	Description
Arithmetic	PADD[B,W,D]	3	Add with wrap-around on [byte, word, doubleword]
	PADDS[B,W]	2	Add signed with saturation on [byte, word]
	PADDUS[B,W]	2	Add unsigned with saturation on [byte, word]
	PSUB[B,W,D]	3	Subtract with wrap-around on [byte, word, doubleword]
	PSUBS[B,W]	2	Subtract signed with saturation on [byte, word]
	PSUBUS[B,W]	2	Subtract unsigned with saturation on [byte, word]
	PMULHW	1	Packed multiply high on words
	PMULLW	1	Packed multiply low on words
	PMADDWD	1	Packed multiply on words and add resulting pairs
Comparison	PCMPEQ[B,W,D]	3	Packed compare for equality [byte, word, doubleword]
	PCMPGT[B,W,D]	3	Packed compare greater: data [byte, word, doubleword]
Conversion	PACKUSWB	1	Pack words into bytes (unsigned with saturation)
	PACKSS[WB,DW]	2	Pack [words into bytes, doublewords into words] (signed with saturation)
	PUNPCKH [BW,WD,DQ]	3	Unpack (interleave) high-order [bytes, words, doublewords] from MMX™ register
	PUNPCKL [BW,WD,DQ]	3	Unpack (interleave) low-order [bytes, words, doublewords] from MMX™ register
Logical	PAND	1	Bitwise AND
	PANDN	1	Bitwise AND NOT
	POR	1	Bitwise OR
	PXOR	1	Bitwise XOR
Shift	PSLL[W,D,Q]	6	Packed shift left logical [word, doubleword, quadword] by amount specified in MMX register or by immediate value
	PSRL[W,D,Q]	6	Packed shift right logical [word, doubleword, quadword] by amount specified in MMX register or by immediate value
	PSRA[W,D]	4	Packed shift right arithmetic [word, doubleword] by amount specified in MMX register or by immediate value
Data Transfer	MOV[D,Q]	4	Move [doubleword, quadword] to MMX register or from MMX register
FP & MMX State Clear	EMMS	1	Empty MMX state

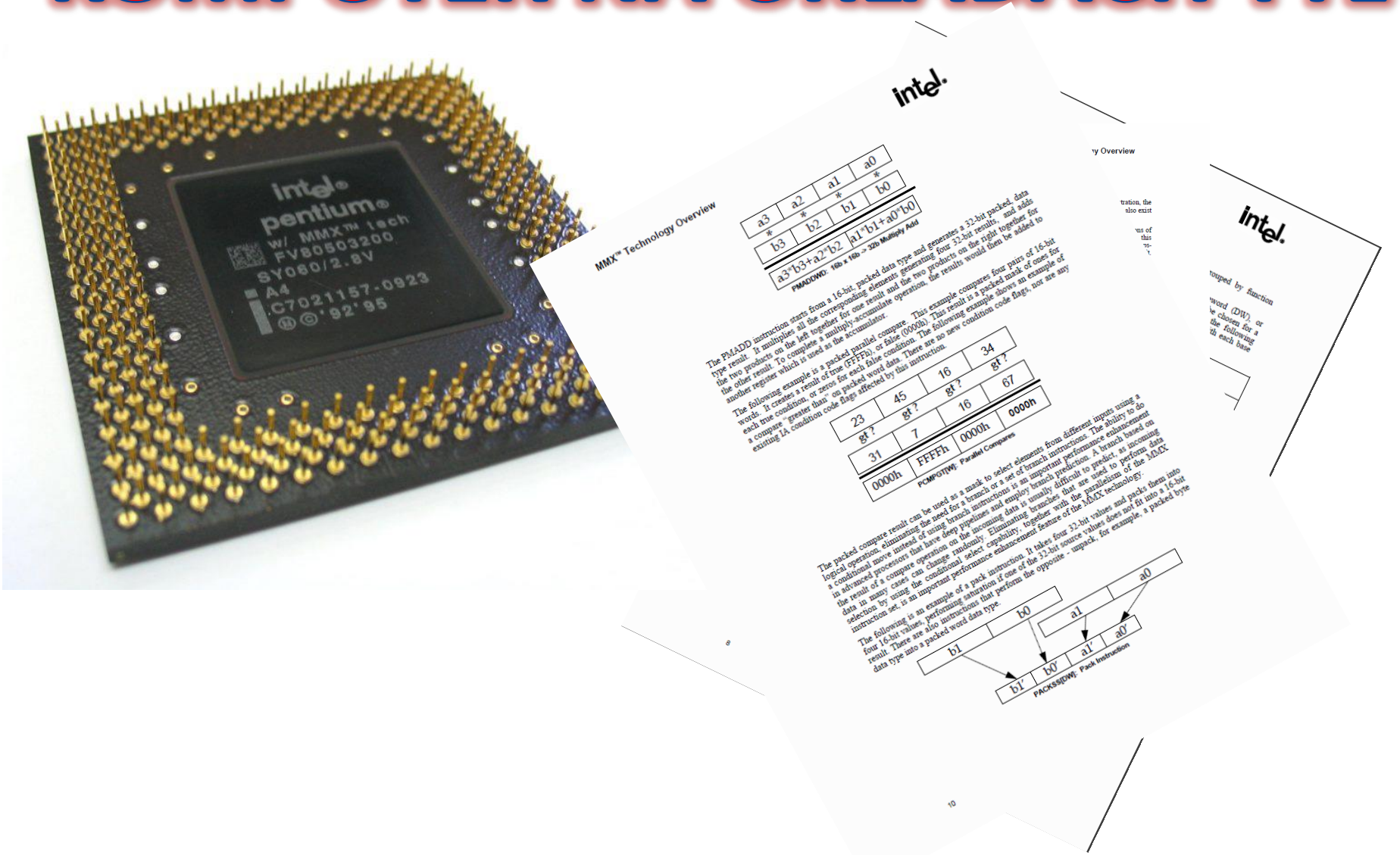
NOWY CEL

KOMPUTER NA UKŁADACH TTL



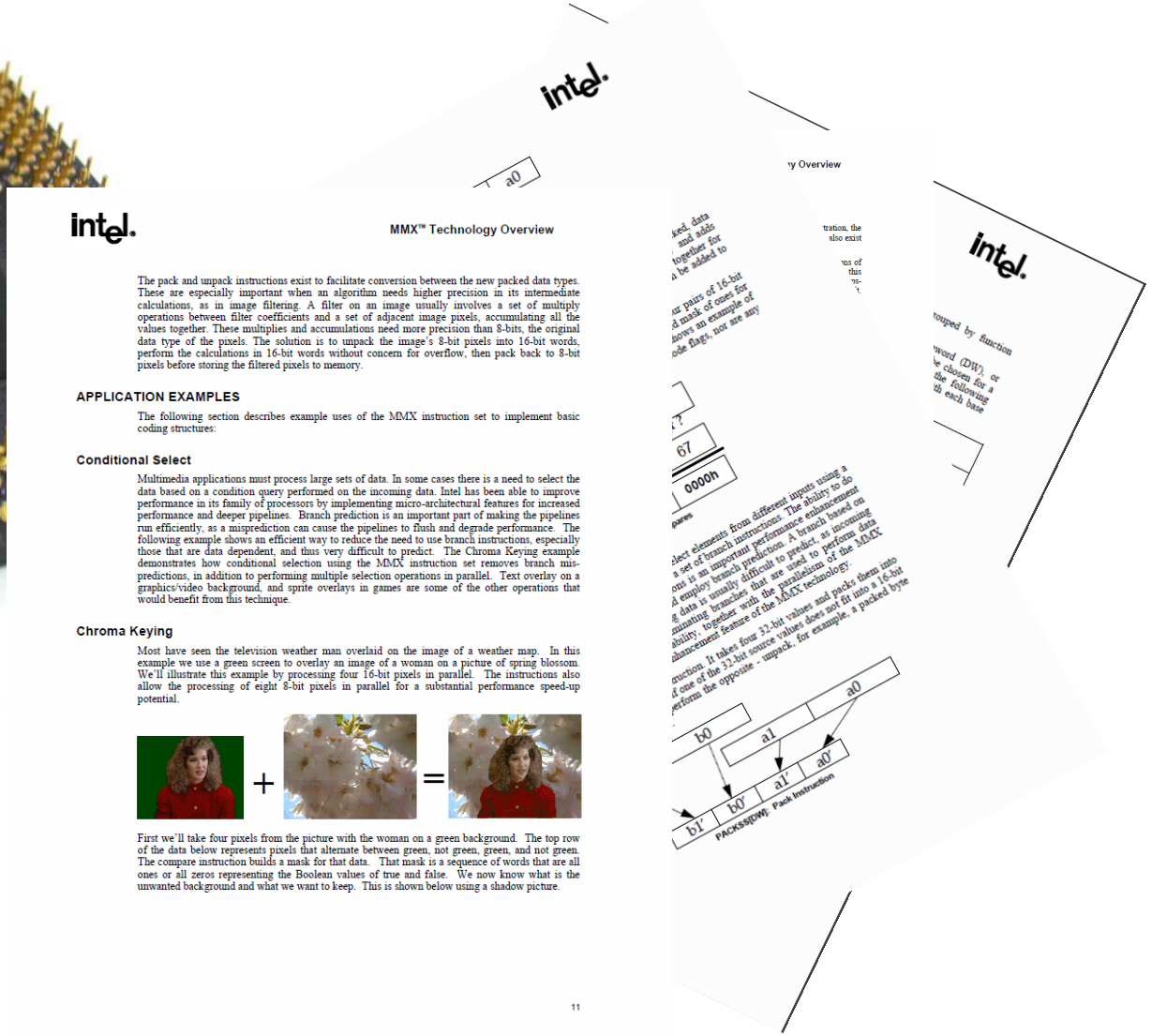
NOWY CEL

KOMPUTER NA UKŁADACH TTL



NOWY CEL

KOMPUTER NA UKŁADACH TTL



intel.

MMX™ Technology Overview

The pack and unpack instructions exist to facilitate conversion between the new packed data types. These are especially important when an algorithm needs higher precision in its intermediate calculations, as in image filtering. A filter on an image usually involves a set of multiply operations between filter coefficients and a set of adjacent image pixels, accumulating all the values together. These multiplies and accumulations need more precision than 8-bits, the original data type of the pixels. The solution is to unpack the image's 8-bit pixels into 16-bit words, perform the calculations in 16-bit words without concern for overflow, then pack back to 8-bit pixels before storing the filtered pixels to memory.

APPLICATION EXAMPLES

The following section describes example uses of the MMX instruction set to implement basic coding structures:

Conditional Select

Multimedia applications must process large sets of data. In some cases there is a need to select the data based on a condition query performed on the incoming data. Intel has been able to improve performance in its family of processors by implementing micro-architectural features for increased performance and deeper pipelines. Branch prediction is an important part of making the pipelines run efficiently, as a misprediction can cause the pipelines to flush and degrade performance. The following example shows an efficient way to reduce the need to use branch instructions, especially those that are data dependent, and thus very difficult to predict. The Chroma Keying example demonstrates how conditional selection using the MMX instruction set removes branch mis-predictions, in addition to performing multiple selection operations in parallel. Text overlay on a graphics/video background, and sprite overlays in games are some of the other operations that would benefit from this technique.

Chroma Keying

Most have seen the television weather man overlaid on the image of a weather map. In this example we use a green screen to overlay an image of a woman on a picture of spring blossom. We'll illustrate this example by processing four 16-bit pixels in parallel. The instructions also allow the processing of eight 8-bit pixels in parallel for a substantial performance speed-up potential.

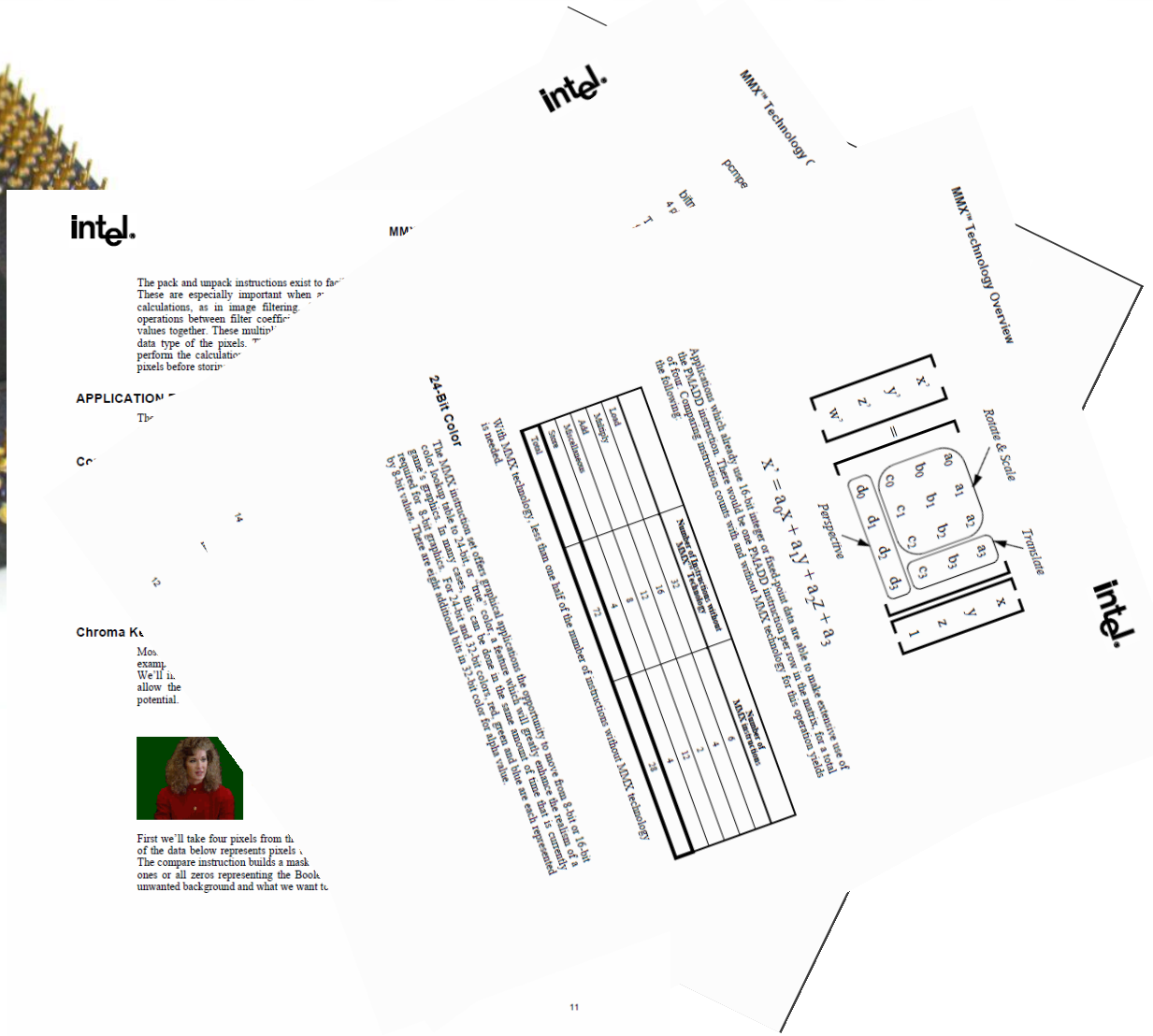
First we'll take four pixels from the picture with the woman on a green background. The top row of the data below represents pixels that alternate between green, not green, green, and not green. The compare instruction builds a mask for that data. That mask is a sequence of words that are all ones or all zeros representing the Boolean values of true and false. We now know what is the unwanted background and what we want to keep. This is shown below using a shadow picture.

11

NOWY CEL

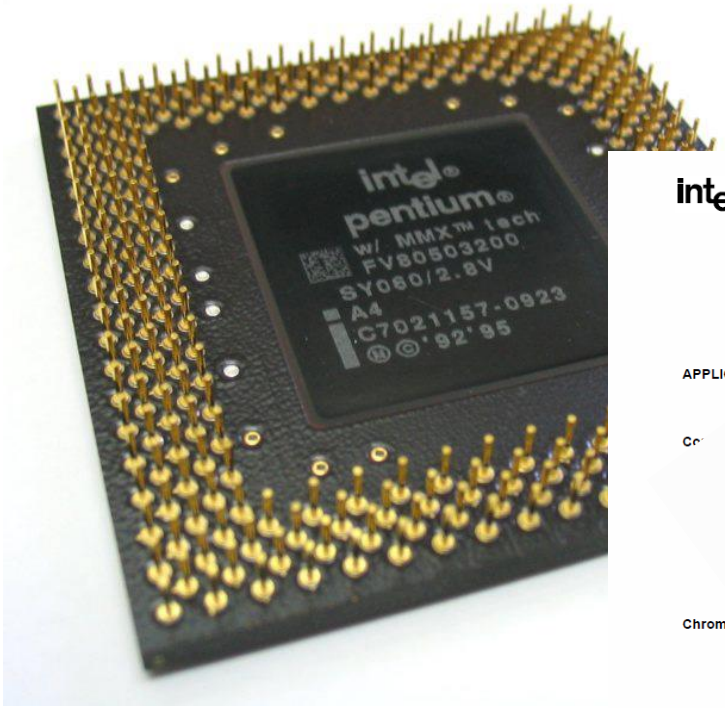
KOMPUTER NA UKŁADACH TTL

NOWY CEL
KOMPUTER NA UKŁADACH TTL



NOWY CEL

KOMPUTER NA UKŁADACH TTL



intel.

MM

The pack and unpack instructions exist to fa
These are especially important when
calculations, as in image filtering.
operations between filter coeffi
values together. These multip
data type of the pixels.
perform the calculatio
pixels before storin

APPLICATION

Th

Co

Chroma K

Mos
examp
We'll u
allow the
potential.

First we'll take four pixels from th
of the data below represents pixels
The compare instruction builds a mask
ones or all zeros representing the Book
unwanted background and what we want t

11

intel.

MMX™ Technology Overview

Image compositing and alpha blending are operations that can be performed on 24-bit color images.

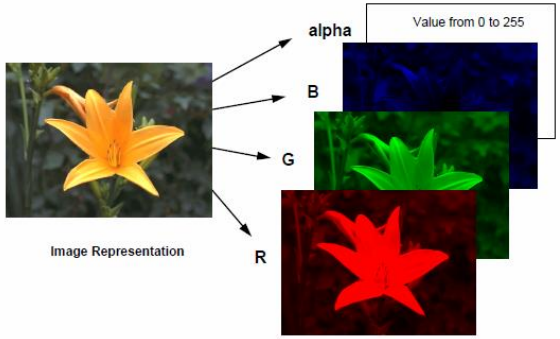


Image Dissolve Using Alpha Blending

This example shows how the MMX instruction set will speed up image compositing. In this example, a flower will dissolve into a swan. The screen starts with a picture of the flower. As the flower gradually fades away, the swan gradually appears.

The math for the dissolve is a straight-forward function. Alpha determines the intensity of the flower. At full intensity, the flower's 8-bit alpha value is FFH, or 255. By plugging 255 in the dissolve equation, each flower pixel is 100 percent and each swan pixel is 0 percent. The equation below calculates each pixel:

Result_pixel = Flower_pixel * (alpha/255) + Swan_pixel * [1 - (alpha/255)]

Illustrated below are the flower and swan when alpha = 230:

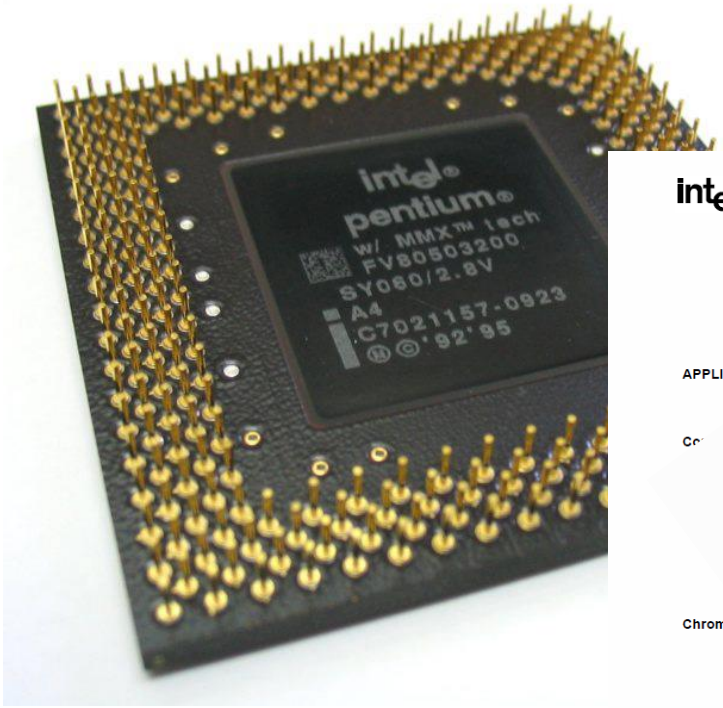
When the alpha value is 230, the resulting picture is 90 percent flower and 10 percent swan. On close examination, some of the swan image appears in the picture to the right of the equal sign.

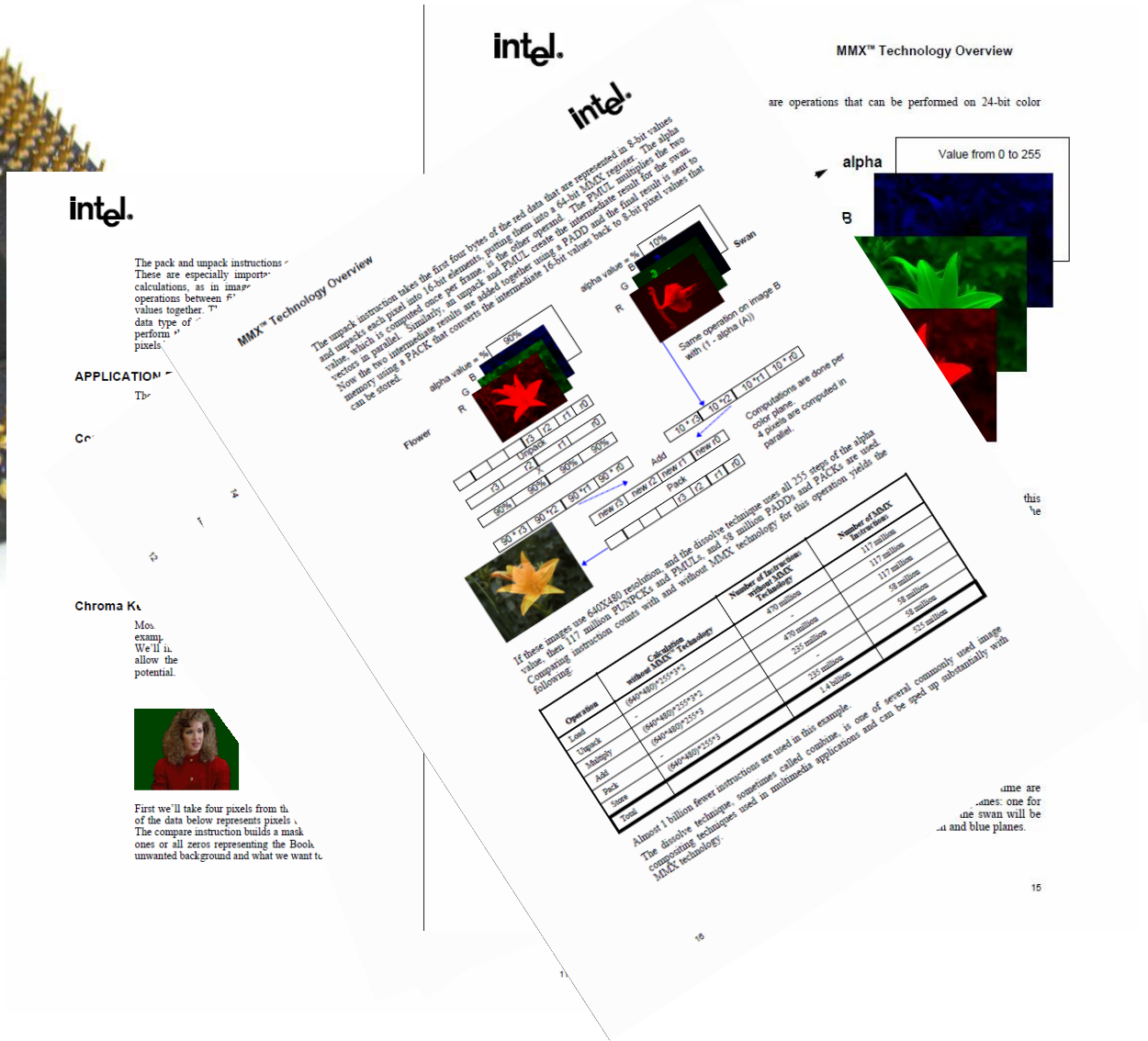
This example assumes that the 24-bit color data is organized so that four pixels at a time are processed from one color plane, that is, the image is separated into individual color planes: one for red, one for green, one for blue. The first four red values from the flower and the swan will be processed first. After finishing the red plane, the processing moves to the green and blue planes.

15

NOWY CEL

KOMPUTER NA UKŁADACH TTL





NOWY CEL

KOMPUTER NA UKŁADACH TTL

intel.

MMX™ Technology Overview

Combine	Dissolve: Fade in, fade out effect A * alpha(A) + B * (1 - alpha(A))
A over B	Transparent Image placed on background A + (B * (a - alpha(A)))
A in B	Image A only where B has Opacity A * alpha (B)
A out B	Image A only where B has transparency A * (1 - alpha(B))
A top B	(A in B) over B (A * alpha(B)) + (B * (1 - alpha(A)))
A XOR B	(B * (1 - alpha(A))) + (A * (1 - alpha(B)))

Alpha blending is a technique used by game developers that is similar to image compositing. Alpha blending allows race cars to drive realistically through fog or smoke, allows a more realistic view of fish in water, or a rabbit in a translucent tube. In these examples, the alpha values wouldn't necessarily be the same for the whole frame, but the basic concept remains the same.

SUMMARY

MMX technology brings more power to multimedia and communication applications. MMX technology adds new data types and instructions that can process data in parallel. MMX technology is fully compatible with existing operating systems and application software.

MMX technology brings a step improvement to the PC platform and enables new applications and usage of PCs. It helps establish a new paradigm in the industry with the PC as an improved communications and multimedia device. Systems enabled with MMX technology will ramp in high volume in 1997 as Intel incorporates the technology in multiple processor generations.

RELATED DOCUMENTATION

Refer to the following documentation for more information on MMX technology.

- *Intel Architecture MMX™ Technology Developers' Manual* (Order Number 243013)
- *Intel Architecture MMX™ Technology Programmer's Reference Manual* (Order Number 243007)

Refer to Intel's corporate website for the latest information on related documentation:

http://www.intel.com/

17

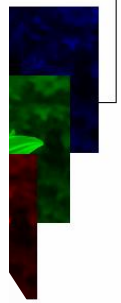
Almost 1 billion fewer instructions are u... sometimes called c...
The dissolve technique,
compositing techniques used in multimedia appli...
MMX technology.

15

Chroma K...

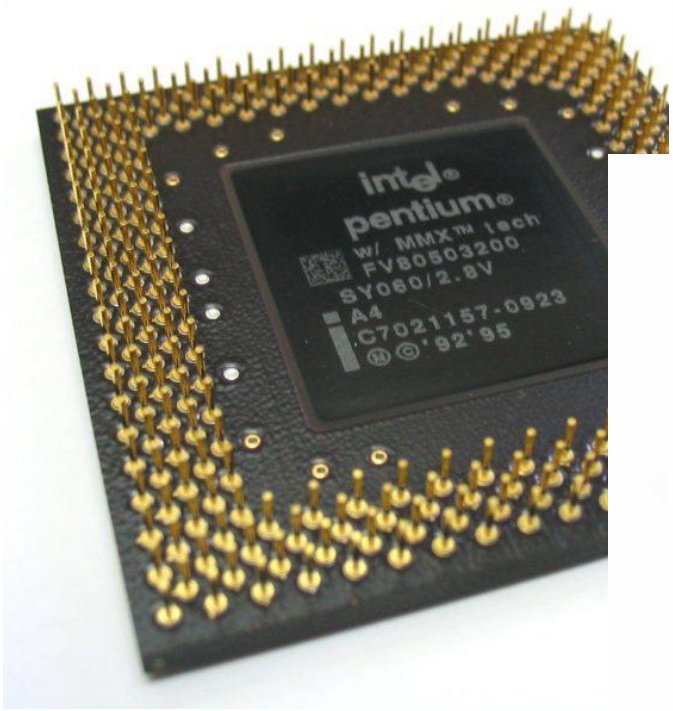
First we'll take four pixels from th...
of the data below represents pixels ...
The compare instruction builds a mask ...
ones or all zeros representing the Book ...
unwanted background and what we want t...

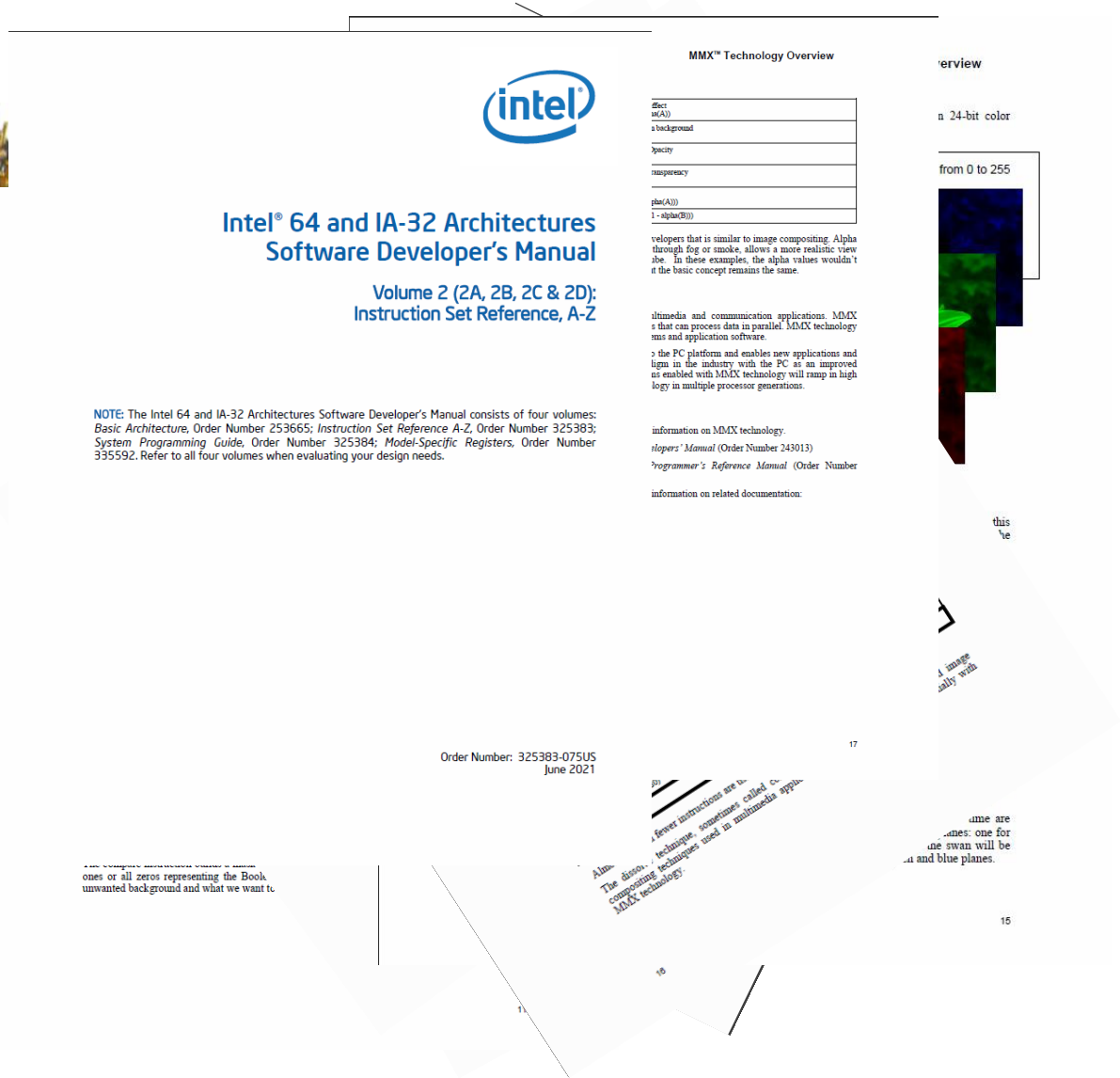
from 0 to 255



NOWY CEL

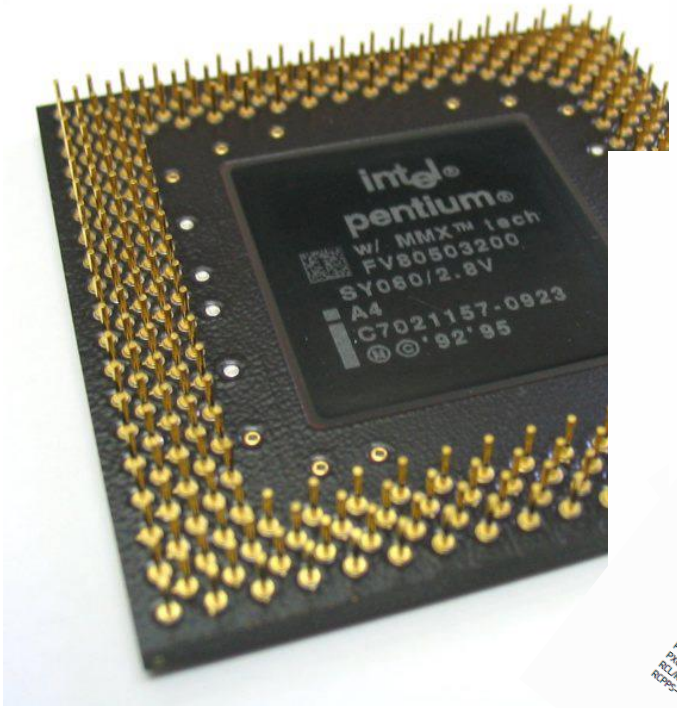
KOMPUTER NA UKŁADACH TTL

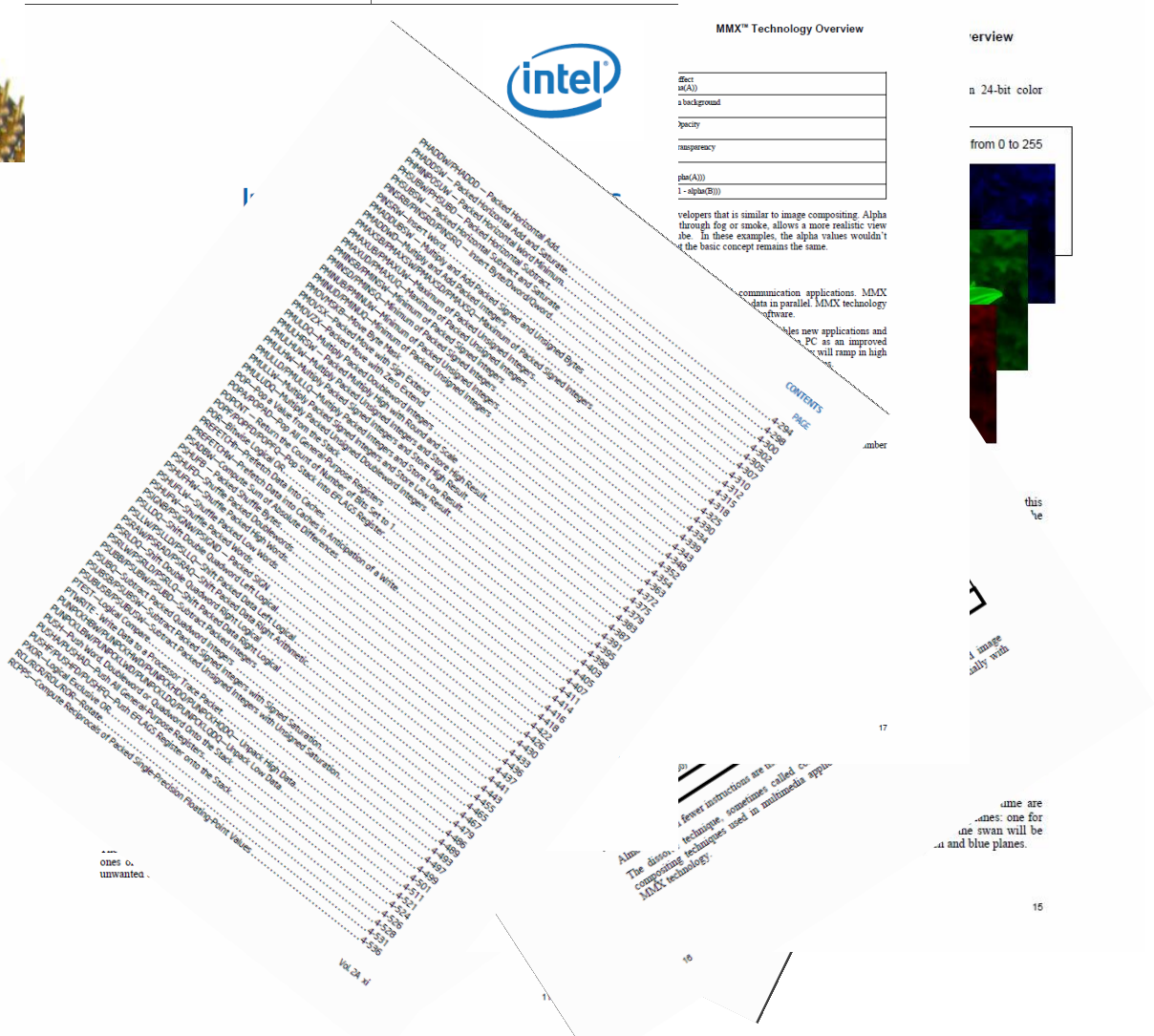




NOWY CEL

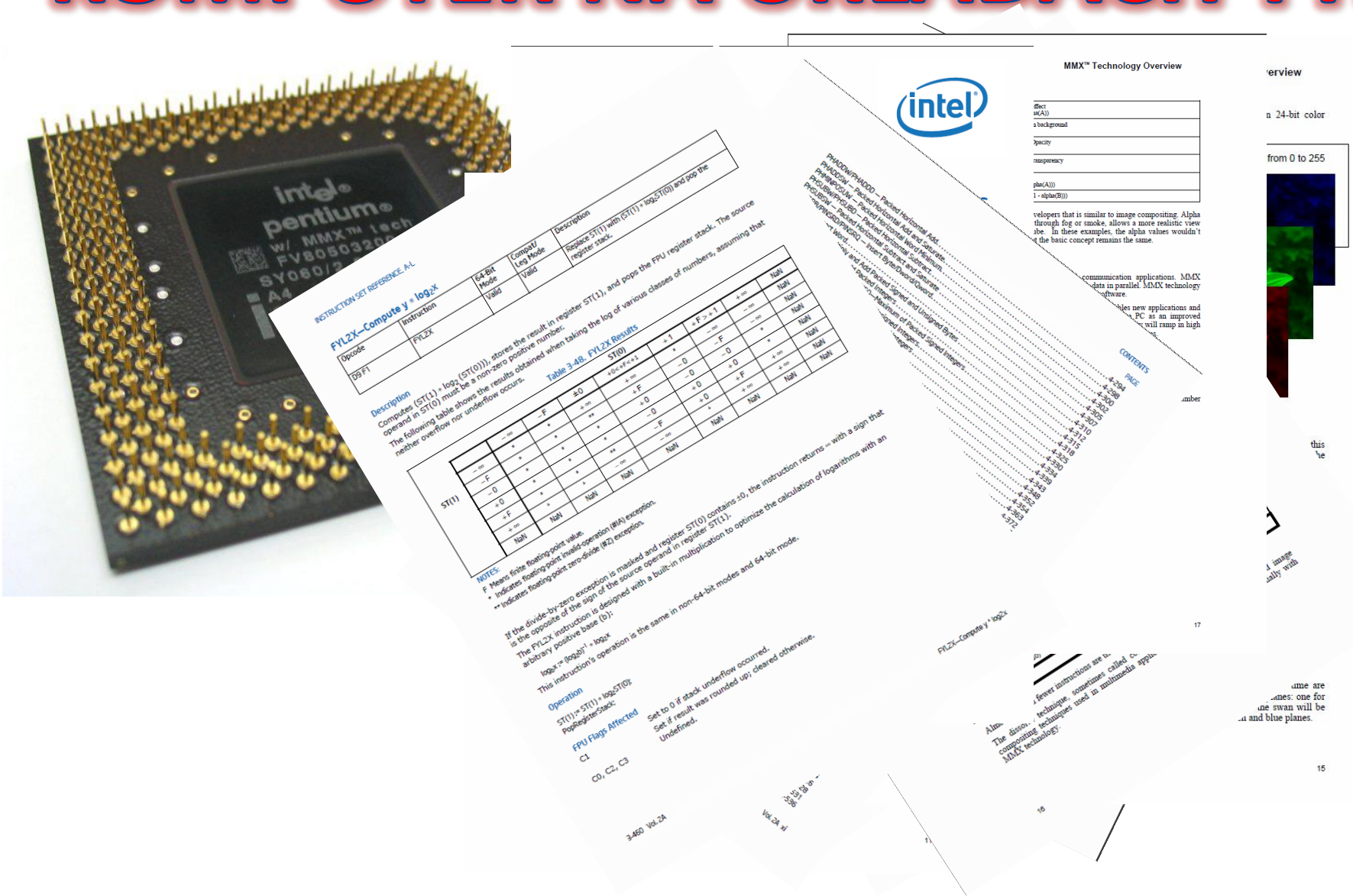
KOMPUTER NA UKŁADACH TTL





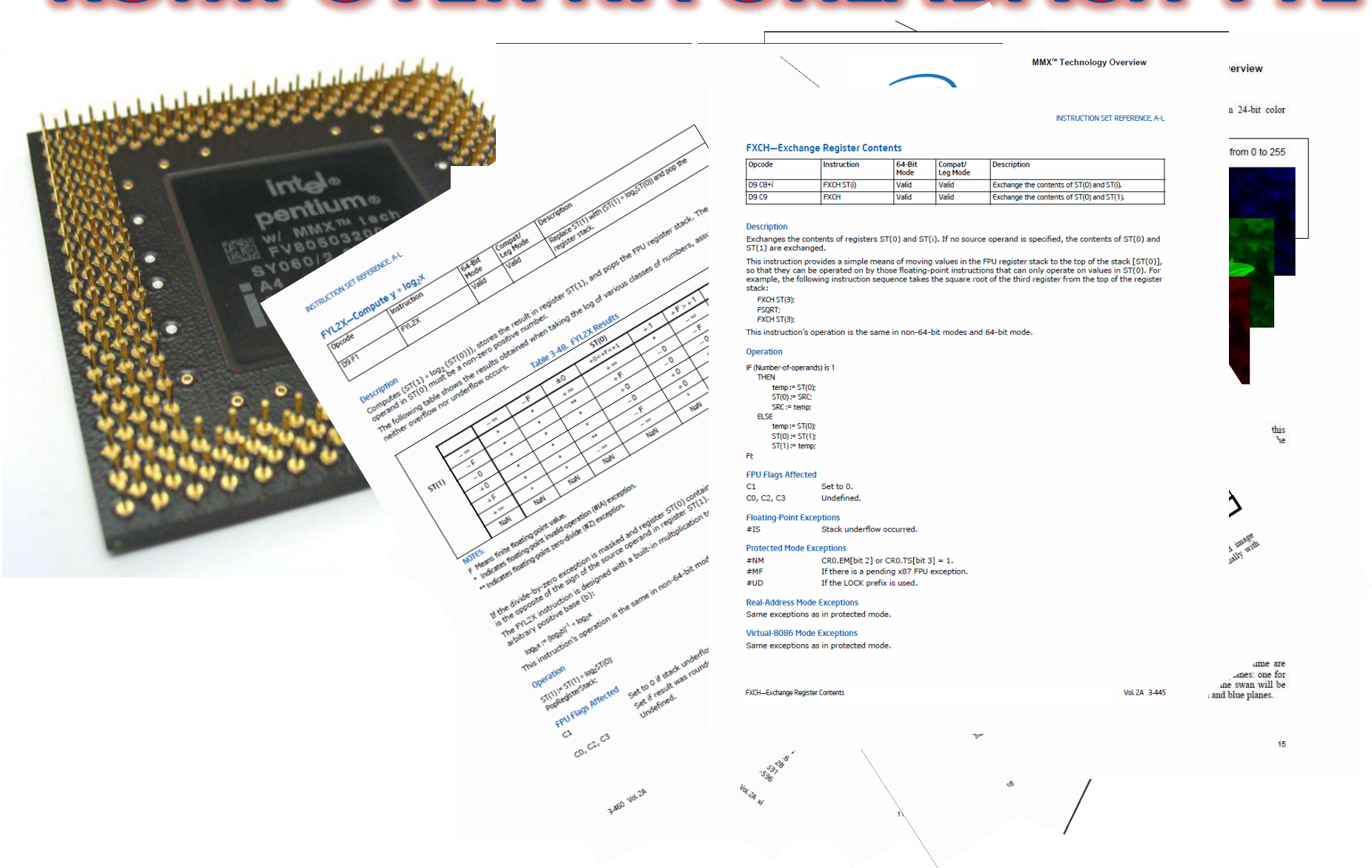
NOWY CEL

KOMPUTER NA UKŁADACH TTL



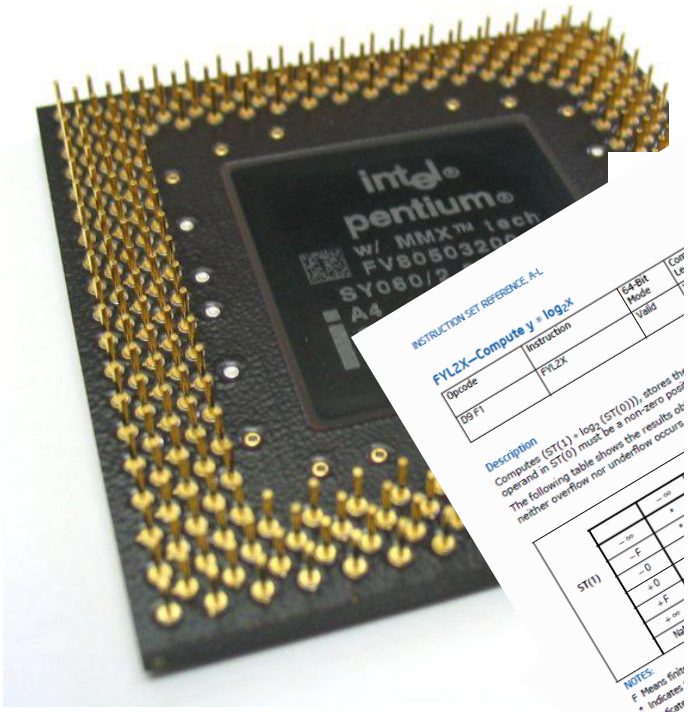
NOWY CEL

KOMPUTER NA UKŁADACH TTL



NOWY CEL

KOMPUTER NA UKŁADACH TTL



INSTRUCTION SET REFERENCE, A-L

FSUBR/FSUBRP/FISUBR—Reverse Subtract

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
D8 /5	FSUBR *m32fp*	Valid	Valid	Subtract ST(0) from *m32fp* and store result in ST(0).
DC /5	FSUBR *m64fp*	Valid	Valid	Subtract ST(0) from *m64fp* and store result in ST(0).
D8 E8+*i*	FSUBR ST(0), ST(*i*)	Valid	Valid	Subtract ST(0) from ST(*i*) and store result in ST(0).
DC E0+*i*	FSUBR ST(*i*), ST(0)	Valid	Valid	Subtract ST(*i*) from ST(0) and store result in ST(*i*).
DE E0+*i*	FSUBRP ST(*i*), ST(0)	Valid	Valid	Subtract ST(*i*) from ST(0), store result in ST(*i*), and pop register stack.
DE E1	FSUBRP	Valid	Valid	Subtract ST(1) from ST(0), store result in ST(1), and pop register stack.
DA /5	FISUBR *m32int*	Valid	Valid	Subtract ST(0) from *m32int* and store result in ST(0).
DE /5	FISUBR *m16int*	Valid	Valid	Subtract ST(0) from *m16int* and store result in ST(0).

Description

Subtracts the destination operand from the source operand and stores the difference in the destination location. The destination operand is always an FPU register; the source operand can be a register or a memory location. Source operands in memory can be in single-precision or double-precision floating-point format or in word or doubleword integer format.

These instructions perform the reverse operations of the FSUB, FSUBP, and FISUB instructions. They are provided to support more efficient coding.

The no-operand version of the instruction subtracts the contents of the ST(1) register from the ST(0) register and stores the result in ST(1). The one-operand version subtracts the contents of the ST(0) register from the contents of a memory location (either a floating-point or an integer value) and stores the result in ST(0). The two-operand version, subtracts the contents of the ST(i) register from the ST(0) register or vice versa.

The FSUBRP instructions perform the additional operation of popping the FPU register stack following the subtraction. To pop the register stack, the processor marks the ST(0) register as empty and increments the stack pointer (TOP) by 1. The no-operand version of the floating-point reverse subtract instructions always results in the register stack being popped. In some assemblers, the mnemonic for this instruction is FSUBR rather than FSUBRP.

The FISUBR instructions convert an integer source operand to double extended-precision floating-point format before performing the subtraction.

The following table shows the results obtained when subtracting various classes of numbers from one another, assuming that neither overflow nor underflow occurs. Here, the DEST value is subtracted from the SRC value (SRC – DEST = result).

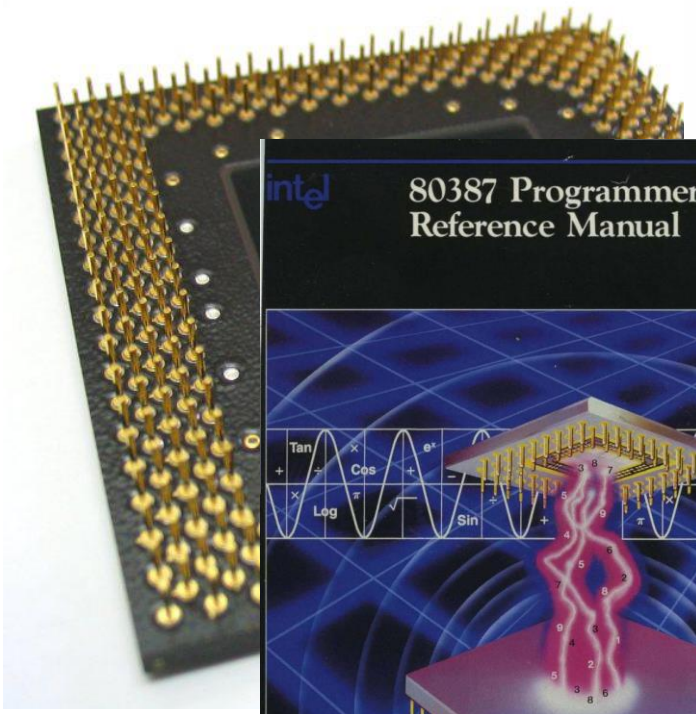
When the difference between two operands of like sign is 0, the result is +0, except for the round toward –∞ mode, in which case the result is –0. This instruction also guarantees that +0 – (–0) = +0, and that –0 – (+0) = –0. When the source operand is an integer 0, it is treated as a +0.

When one operand is ∞, the result is ∞ of the expected sign. If both operands are ∞ of the same sign, an invalid-operation exception is generated.

NOWY CEL

KOMPUTER NA UKŁADACH TTL

NOWY CEL
KOMPUTER NA UKŁADACH TTL



INSTRUCTION SET REFERENCE, A-L

FSUBR/FSUBRP/FISUBR—Reverse Subtract

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
		Valid	Valid	Subtract ST(0) from *m32fp* and store result in ST(0).
		Valid	Valid	Subtract ST(0) from *m64fp* and store result in ST(0).
		Valid	Valid	Subtract ST(0) from ST(i) and store result in ST(0).
		Valid	Valid	Subtract ST(i) from ST(0) and store result in ST(i).
		Valid	Valid	Subtract ST(i) from ST(0), store result in ST(i), and pop register stack.
		Valid	Valid	Subtract ST(1) from ST(0), store result in ST(1), and pop register stack.
		Valid	Valid	Subtr...
		Valid	Valid	

ology Overview

INSTRUCTION SET REFERENCE, A-L

erview

n 24-bit color

from 0 to 255

FSQRT—Square Root

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
D9 FA	FSQRT	Valid	Valid	Computes square root of ST(0) and stores the result in ST(0).

Description

Computes the square root of the source value in the ST(0) register and stores the result in ST(0).

The following table shows the results obtained when taking the square root of various classes of numbers, assuming that neither overflow nor underflow occurs.

Table 3-37. FSQRT Results

SRC (ST(0))	DEST (ST(0))
-∞	*
-F	*
-0	-0
+0	+0
+F	+F
+∞	+∞
NaN	NaN

NOTES:
F Means finite floating-point value.
* Indicates floating-point invalid-arithmetic-operand (#IA) exception.

Operation

This instruction's operation is the same in non-64-bit modes and 64-bit mode.

ST(0) := SquareRoot(ST(0));

FPU Flags Affected

C1
Set to 0 if stack underflow occurred.
Set if result was rounded up; cleared otherwise.
Undefined.

C0, C2, C3

Floating Point Exceptions

Stack underflow occurred.
Source operand is an SNaN value or unsupported format.
Source operand is a negative value (except for -0).
Value cannot be represented exactly in destination format.

#IS
#IA

Protected Mode Exceptions

#D
#P

CR0.EM[bit 2] or CR0.TS[bit 3] = 1.
If there is a pending x87 FPU exception.
If the LOCK prefix is used.

#NM
#MF
#UD

Address Mode Exceptions

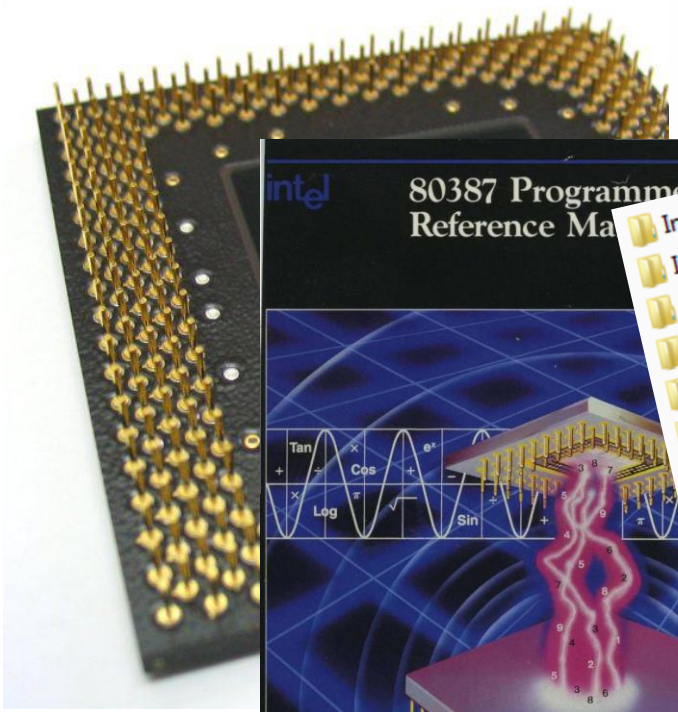
...s in protected mode.

...ame are
...anes: one for
...he swan will be
... and blue planes.

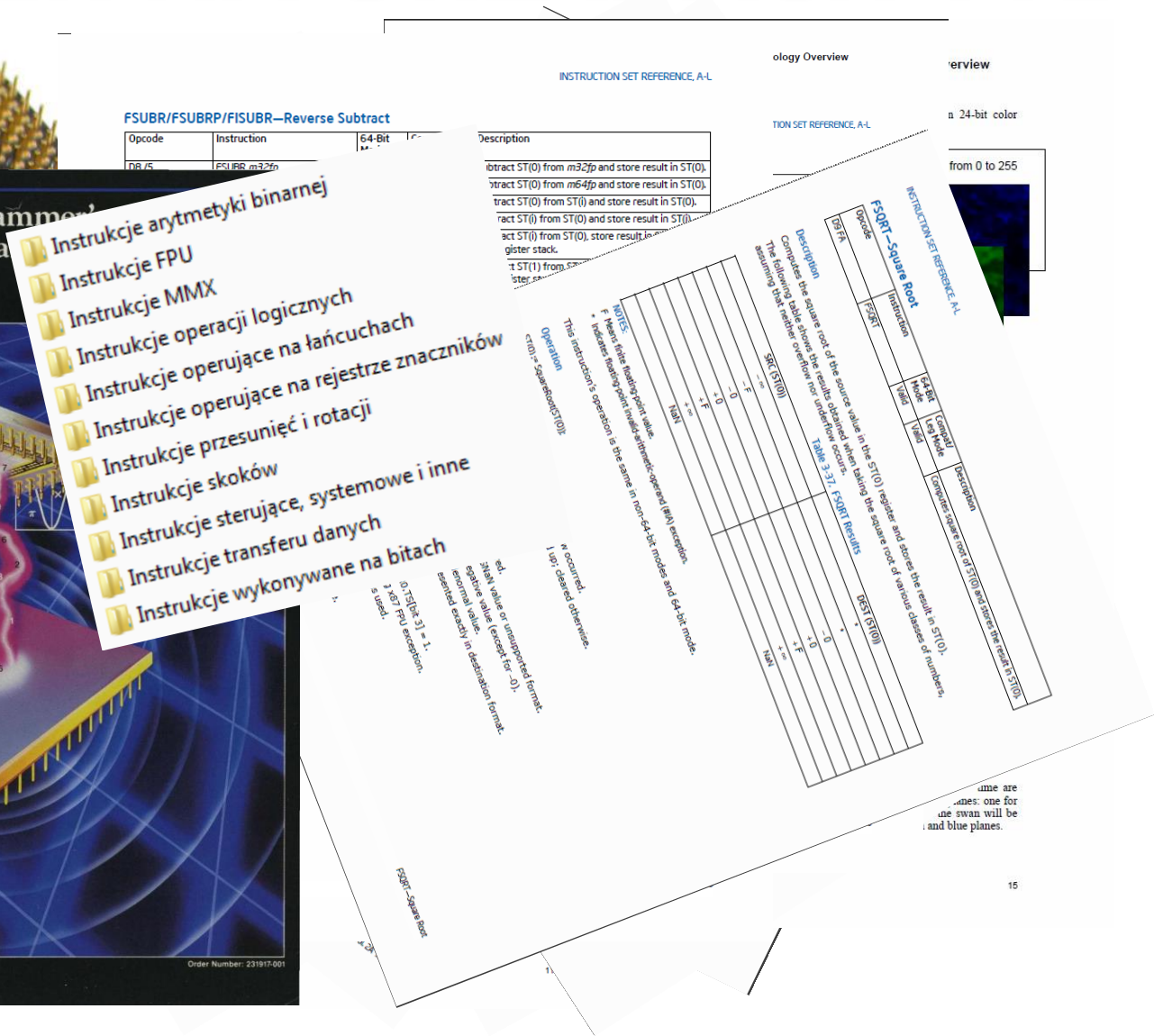
FSQRT—Square Root

15

NOWY CEL
KOMPUTER NA UKŁADACH TTL



- Instrukcje arytmetyki binarnej
- Instrukcje FPU
- Instrukcje MMX
- Instrukcje operacji logicznych
- Instrukcje operujące na łańcuchach
- Instrukcje operujące na rejestrze znaczników
- Instrukcje przesunięć i rotacji
- Instrukcje skoków
- Instrukcje sterujące, systemowe i inne
- Instrukcje transferu danych
- Instrukcje wykonywane na bitach



NOWY CEL

KOMPUTER NA UKŁADACH TTL

INSTRUCTION SET REFERENCE, A-L

ology Overview

erview

n 24-bit color

from 0 to 255

TION SET REFERENCE, A-L

FSUBR/FSUBRP/FISUBR—Reverse Subtract

Opcode	Instruction	64-Bit	Description
			ubtract ST(0) from *m32fp* and store result in ST(0).
			btract ST(0) from *m64fp* and store result in ST(0).
			tract ST(0) from ST(i) and store result in ST(0).
			ract ST(i) from ST(0) and store result in ST(i).

arytmetyki binarnej

ADD/ADDR/CADDcc/CADDRcc A,B	0	0115-0129	15	SUMOWANIE LICZB CAŁKOWITYCH	ARGUMENT A ⇋ B = ARGUMENT A + ARGUMENT B
ADC/ADCR/CADCcc/CADCRcc A,B	0	0130-0144	15	SUMOWANIE LICZB CAŁKOWITYCH Z PRZENIESIENIEM	ARGUMENT A ⇋ B = ARGUMENT A + ARGUMENT B + c
SUB/SUBR/CSUBcc/CSUBRcc A,B	0	0145-0159	15	ODEJMOWANIE LICZB CAŁKOWITYCH	ARGUMENT A ⇋ B = ARGUMENT A - ARGUMENT B
SBB/SBBR/CSBBcc/CSBBRcc A,B	0	0160-0174	15	ODEJMOWANIE LICZB CAŁKOWITYCH Z PRZENIESIENIEM	ARGUMENT A ⇋ B = ARGUMENT A - (ARGUMENT B + c)
RSUB/RSUBR/CRSUBcc/CRSUBRcc A,B	0	0175-0189	15	ODWROTNE ODEJMOWANIE LICZB CAŁKOWITYCH	ARGUMENT A ⇋ B = ARGUMENT B - ARGUMENT A
RSBB/RSBBR/CRSBBcc/CRSBBRcc A,B	0	0190-0204	15	ODWROTNE ODEJMOWANIE LICZB CAŁKOWITYCH Z PRZENIESIENIEM	ARGUMENT A ⇋ B = ARGUMENT B - (ARGUMENT A + c)
MUL/MULR/CMULcc/CMULRcc A,B	0	0205-0219	15	MNOŻENIE LICZB CAŁKOWITYCH BEZ ZNAKU	ARGUMENT A ⇋ B = ARGUMENT A × ARGUMENT B
IMUL/IMULR/CIMULcc/CIMULRcc A,B	0	0220-0234	15	MNOŻENIE LICZB CAŁKOWITYCH ZE ZNAKIEM	ARGUMENT A ⇋ B = ARGUMENT A × ARGUMENT B
DIV/DIVR/CDIVcc/CDIVRcc A,B	0	0235-0249	15	DZIELENIE LICZB CAŁKOWITYCH BEZ ZNAKU BEZ RESZTY	ARGUMENT A ⇋ B = ARGUMENT A / ARGUMENT B
IDIV/IDIVR/CIDIVcc/CIDIVRcc A,B	0	0250-0264	15	DZIELENIE LICZB CAŁKOWITYCH ZE ZNAKIEM BEZ RESZTY	ARGUMENT A ⇋ B = ARGUMENT A / ARGUMENT B
RDIV/RDIVR/CRDIVcc/CRDIVRcc A,B	0	0265-0279	15	ODWROTNE DZIELENIE LICZB CAŁKOWITYCH BEZ ZNAKU BEZ RESZTY	ARGUMENT A ⇋ B = ARGUMENT B / ARGUMENT A
RIDIV/RIDIVR/CRIDIVcc/CRIDIVRcc A,B	0	0280-0294	15	ODWROTNE DZIELENIE LICZB CAŁKOWITYCH ZE ZNAKIEM BEZ RESZTY	ARGUMENT A ⇋ B = ARGUMENT B / ARGUMENT A
INC/CINCcc A	0	0295-0300	6	ZWIĘKSZANIE ZAWARTOŚCI O JEDEN	ARGUMET A = ARGUMENT A+1
DEC/CDECcc A	0	0301-0306	6	ZMNIEJSZANIE ZAWARTOŚCI O JEDEN	ARGUMET A = ARGUMENT A-1
NEG/CNEGcc A	0	0307-0312	6	NEGACJA DO U2	ARGUMET A = 0-ARGUMENT A
CMP/RCMP/CCMPcc/CRCMPcc A,B	0	0313-0327	15	PORÓWNANIE PRZEZ ODEJMOWANIE	REJESTR FLAG = ARGUMENT A - ARGUMENT B ⇋ ARGUMENT B - ARGUMENT A
ADDE/ADDER/CADDEcc/CADDERcc A,B,C	1	0001-0017	17	SUMOWANIE LICZB CAŁKOWITYCH	ARGUMENT A = ARGUMENT B + ARGUMENT C ——— ARGUMENT A + ARGUMENT B = ARGUMENT C
ADCE/ADCER/CADCEcc/CADCERcc A,B,C	1	0018-0034	17	SUMOWANIE LICZB CAŁKOWITYCH Z PRZENIESIENIEM	ARGUMENT A = ARGUMENT B + ARGUMENT C + c ——— ARGUMENT A + ARGUMENT B + c = ARGUMENT C
SUBE/SUBER/CSUBEcc/CSUBERcc A,B,C	1	0035-0051	17	ODEJMOWANIE LICZB CAŁKOWITYCH	ARGUMENT A = ARGUMENT B - ARGUMENT C ——— ARGUMENT A - ARGUMENT B = ARGUMENT C
SBBE/SBBER/CSBBEcc/CSBBERcc A,B,C	1	0052-0068	17	ODEJMOWANIE LICZB CAŁKOWITYCH Z PRZENIESIENIEM	ARGUMENT A = ARGUMENT B - (ARGUMENT C + c) ——— ARGUMENT A - (ARGUMENT B + c) = ARGUMENT C
RSUBE/RSUBER/CRSUBEcc/CRSUBERcc A,B,C	1	0069-0085	17	ODWROTNE ODEJMOWANIE LICZB CAŁKOWITYCH	ARGUMENT A = ARGUMENT C - ARGUMENT B ——— ARGUMENT B - ARGUMENT A = ARGUMENT C
RSBBE/RSBBER/CRSBBEcc/CRSBBERcc A,B,C	1	0086-0102	17	ODWROTNE ODEJMOWANIE LICZB CAŁKOWITYCH Z PRZENIESIENIEM	ARGUMENT A = ARGUMENT C - (ARGUMENT B + c) ——— ARGUMENT B - (ARGUMENT A + c) = ARGUMENT C
MULE/MULER/CMULEcc/CMULERcc A,B,C,D	1	0103-0103	1	MNOŻENIE LICZB CAŁKOWITYCH BEZ ZNAKU	ARGUMENT ZESPOLONY AB = ARGUMENT C × ARGUMENT D ——— ARGUMENT A × ARGUMENT B = ARGUMENT ZESPOLONY CD
IMULE/IMULER/CIMULEcc/CIMULERcc A,B,C,D	1	0104-0104	1	MNOŻENIE LICZB CAŁKOWITYCH ZE ZNAKIEM	ARGUMENT ZESPOLONY AB = ARGUMENT C × ARGUMENT D ——— ARGUMENT A × ARGUMENT B = ARGUMENT ZESPOLONY CD
DIVE/DIVER/CDIVEcc/CDIVERcc A,B,C,D	1	0105-0105	1	DZIELENIE LICZB CAŁKOWITYCH BEZ ZNAKU Z RESZTĄ	ARGUMENT ZESPOLONY A,B = ARGUMENT C / ARGUMENT D ——— ARGUMENT A / ARGUMENT B = ARGUMENT ZESPOLONY C,D
IDIVE/IDIVER/CIDIVEcc/CIDIVERcc A,B,C,D	1	0106-0106	1	DZIELENIE LICZB CAŁKOWITYCH ZE ZNAKIEM Z RESZTĄ	ARGUMENT ZESPOLONY A,B = ARGUMENT C / ARGUMENT D ——— ARGUMENT A / ARGUMENT B = ARGUMENT ZESPOLONY C,D
RDIVE/RDIVER/CRDIVEcc/CRDIVERcc A,B,C,D	1	0107-0107	1	ODWROTNE DZIELENIE LICZB CAŁKOWITYCH BEZ ZNAKU Z RESZTĄ	ARGUMENT ZESPOLONY A,B = ARGUMENT D / ARGUMENT C ——— ARGUMENT B / ARGUMENT A = ARGUMENT ZESPOLONY C,D
RIDIVE/RIDIVER/CRIDIVEcc/CRIDIVERcc A,B,C,D	1	0108-0108	1	ODWROTNE DZIELENIE LICZB CAŁKOWITYCH ZE ZNAKIEM Z RESZTĄ	ARGUMENT ZESPOLONY A,B = ARGUMENT D / ARGUMENT C ——— ARGUMENT B / ARGUMENT A = ARGUMENT ZESPOLONY C,D

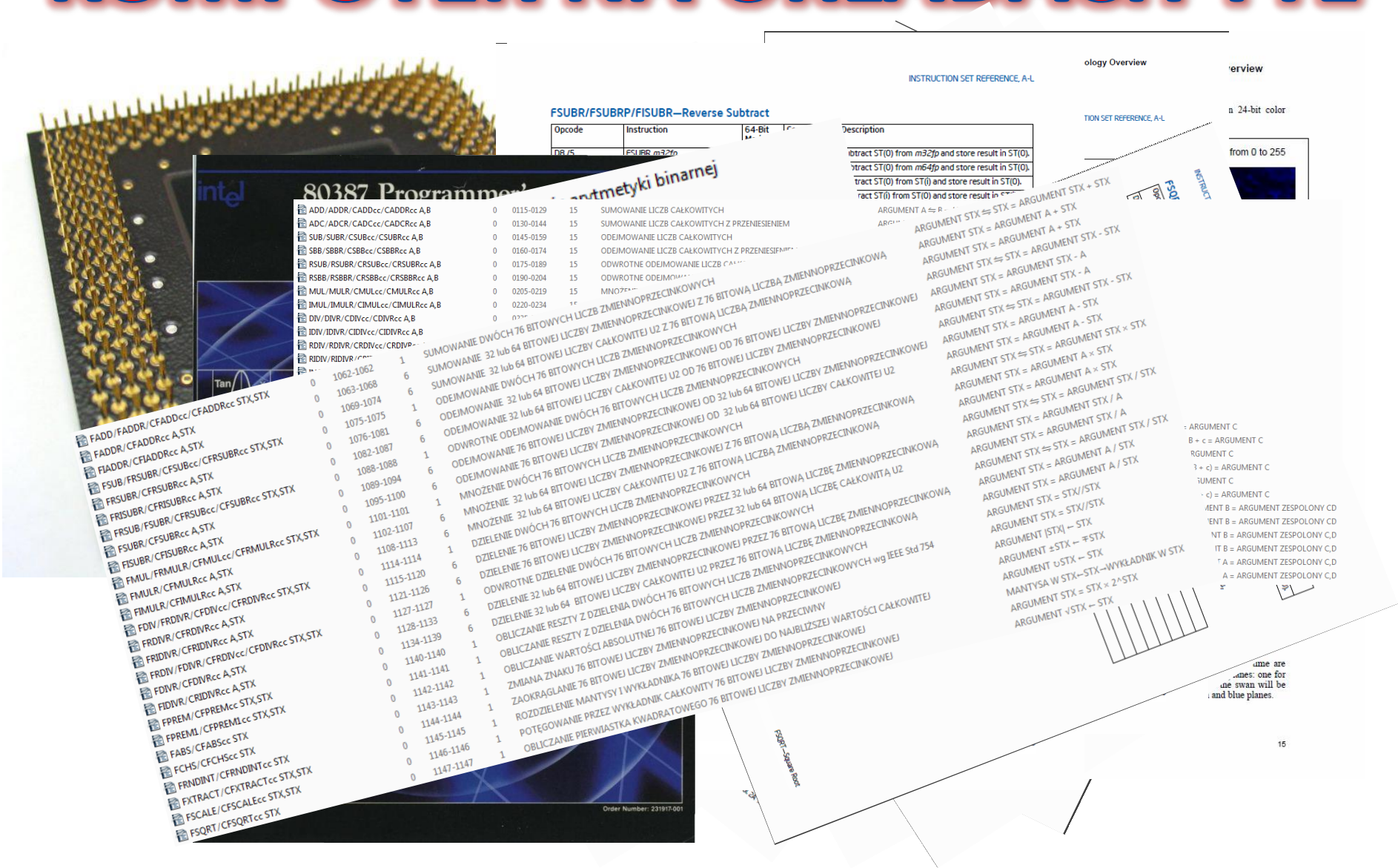
ame are
anes: one for
he swan will be
and blue planes.

FSQRT—Square Root

15

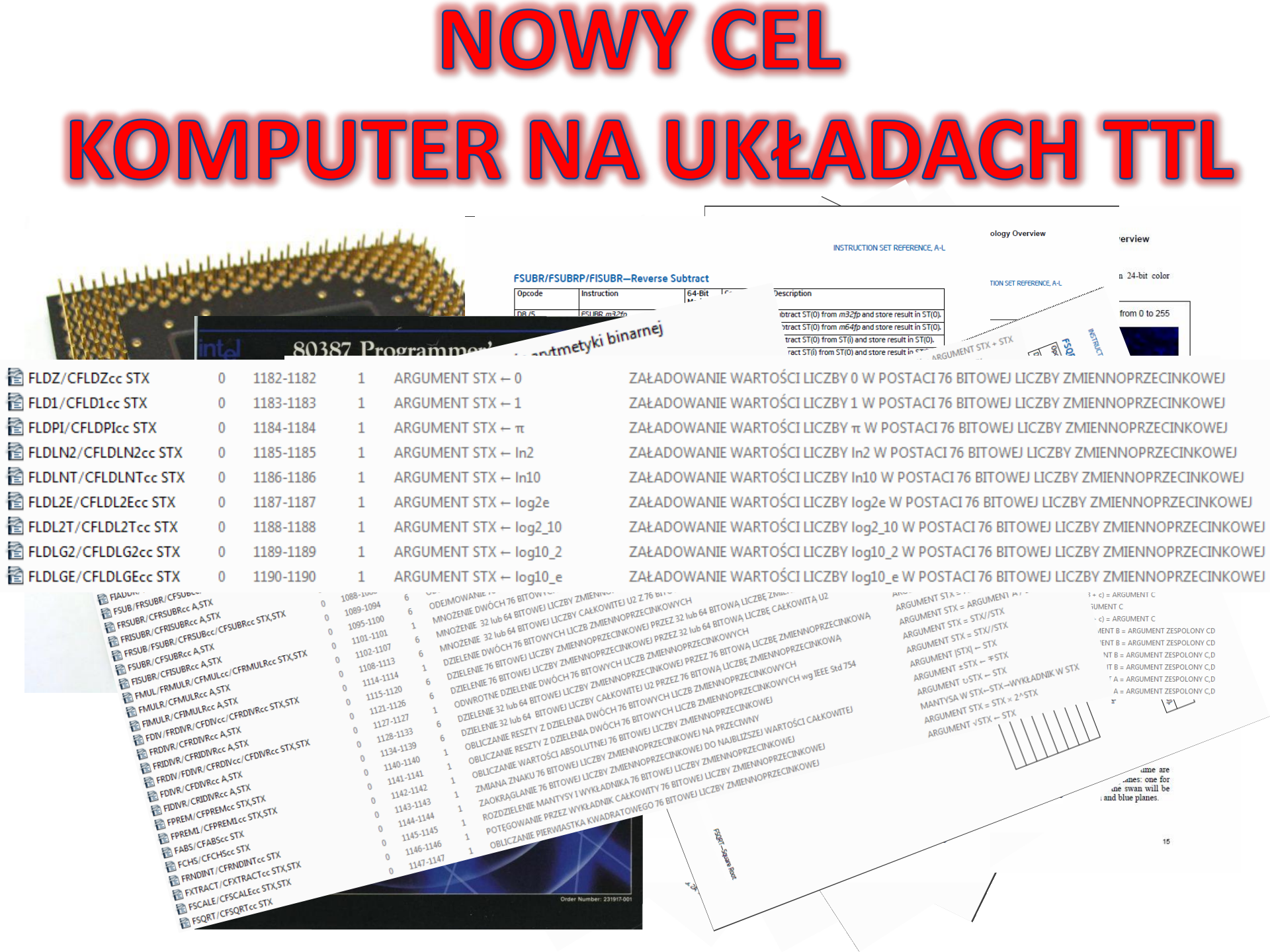
NOWY CEL

KOMPUTER NA UKŁADACH TTL



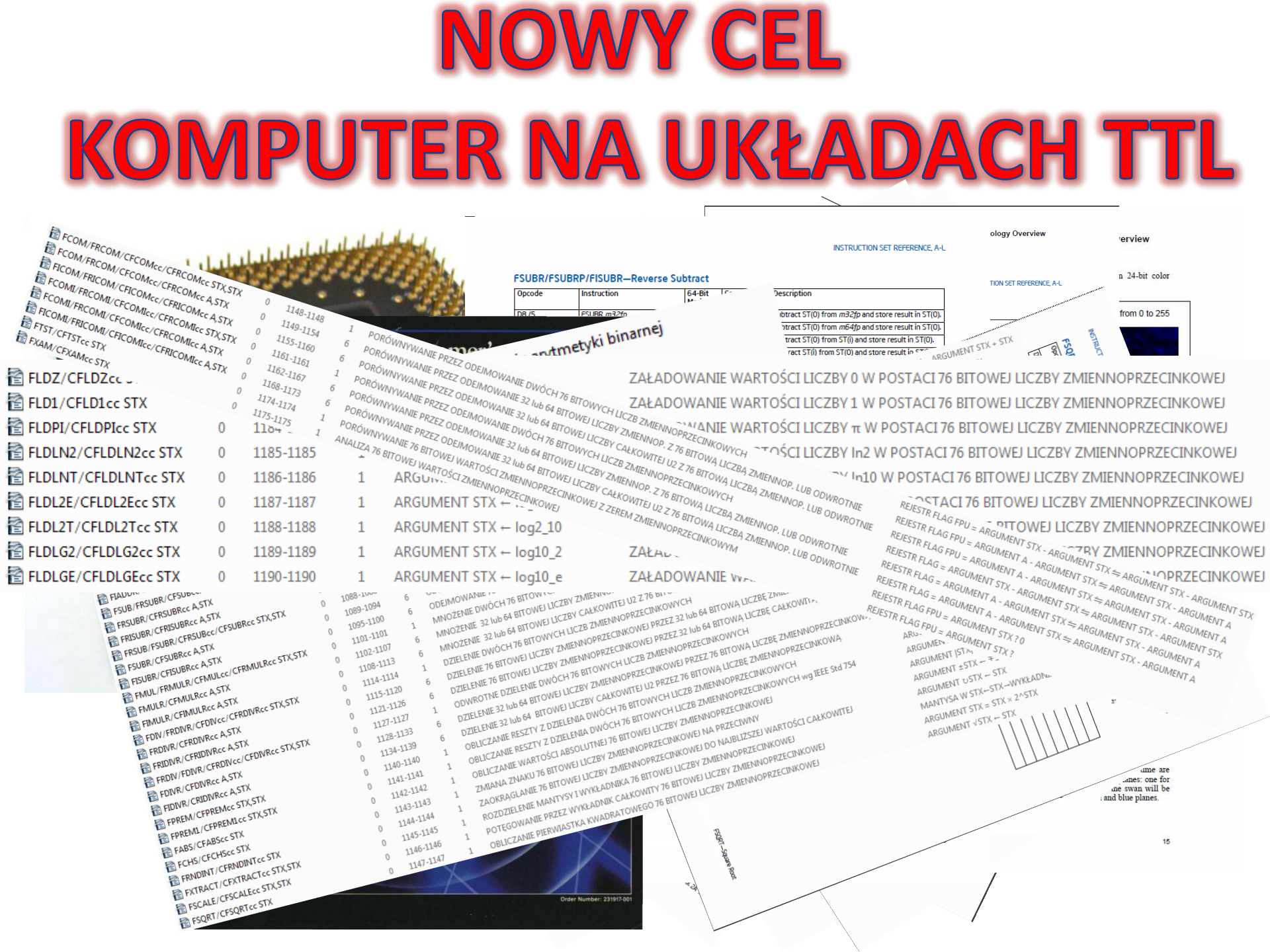
NOWY CEL

KOMPUTER NA UKŁADACH TTL



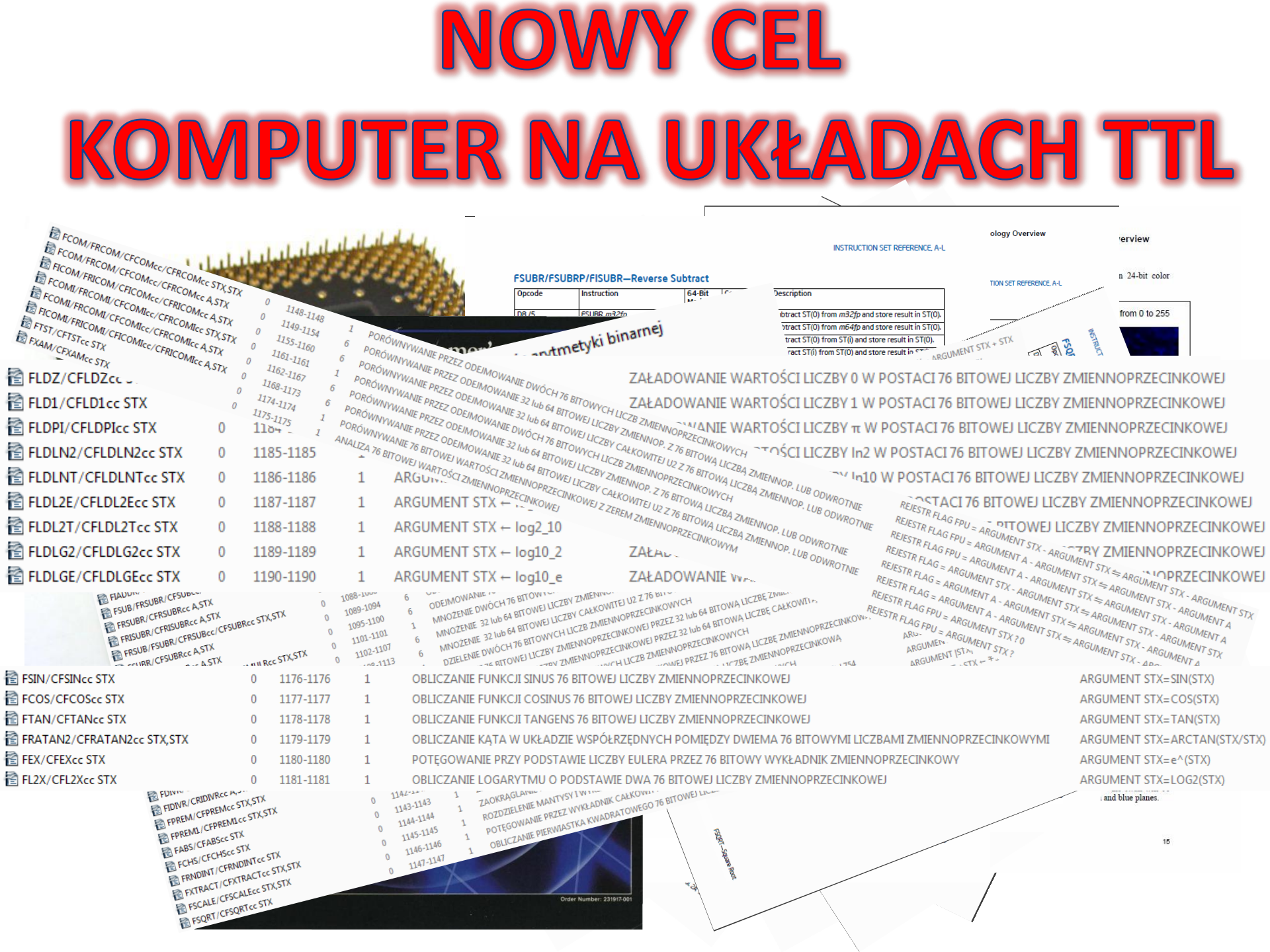
NOWY CEL

KOMPUTER NA UKŁADACH TTL



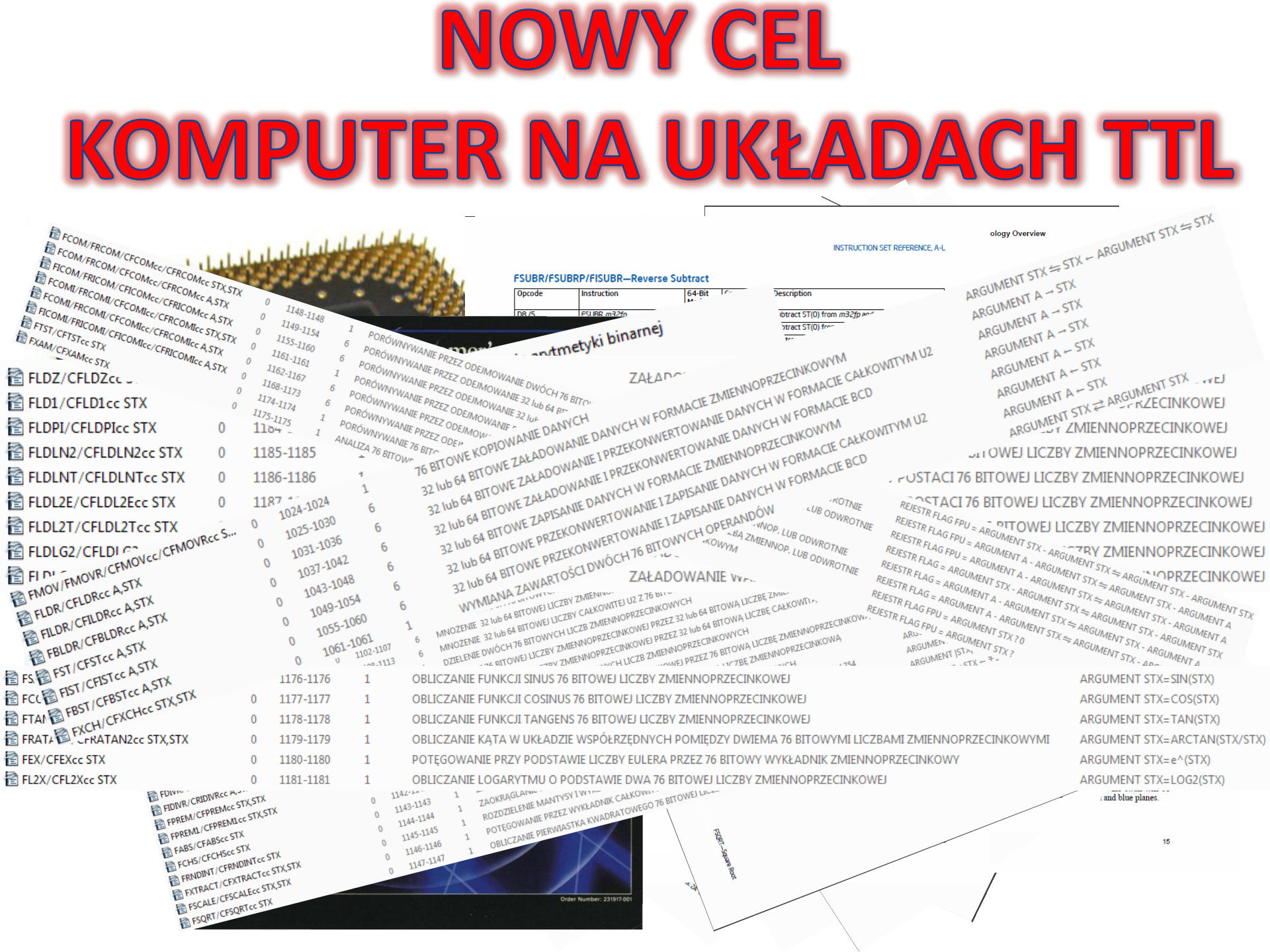
NOWY CEL

KOMPUTER NA UKŁADACH TTL



NOWY CEL

KOMPUTER NA UKŁADACH TTL



NOWY CEL

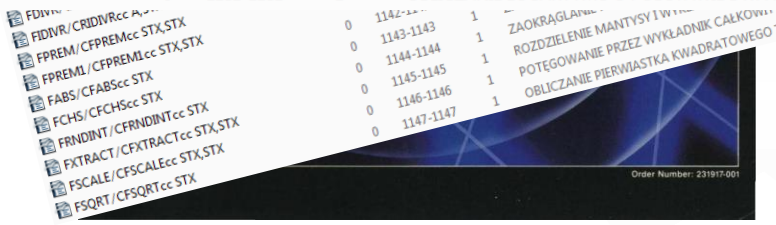
KOMPUTER NA UKŁADACH TTL

PADDB/PADDBR/CPADDBcc/CPADDBRcc MMX,MMX	0	0732-0732	1	DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU	MMX_B = MMX_B + MMX_B ——— MMX_B + MMX_B = MMX_B
PADDB/PADDBR/CPADDBcc/CPADDBRcc A,MMX	0	0733-0738	6	DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU	A_B = A_B + MMX_B ——— A_B + MMX_B = MMX_B
PADDW/PADDWR/CPADDWcc/CPADDWRcc MMX,MMX	0	0739-0739	1	DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWIJANIA WYNIKU	MMX_W = MMX_W + MMX_W ——— MMX_W + MMX_W = MMX_W
PADDW/PADDWR/CPADDWcc/CPADDWRcc A,MMX	0	0740-0745	6	DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWIJANIA WYNIKU	A_W = A_W + MMX_W ——— A_W + MMX_W = MMX_W
PADDD/PADDDR/CPADDDcc/CPADDDRcc MMX,MMX	0	0746-0746	1	DODAWANIE DWÓCH PODWÓJNYCH SŁÓW W TRYBIE ZAWIJANIA WYNIKU	MMX_D = MMX_D + MMX_D ——— MMX_D + MMX_D = MMX_D
PADDD/PADDDR/CPADDDcc/CPADDDRcc A,MMX	0	0747-0752	6	DODAWANIE DWÓCH PODWÓJNYCH SŁÓW W TRYBIE ZAWIJANIA WYNIKU	A_D = A_D + MMX_D ——— A_D + MMX_D = MMX_D
PADDSB/PADDSBR/CPADDSBcc/CPADDSBRcc MMX,MMX	0	0753-0753	1	DODAWANIE OŚMIU BAJTÓW ZE ZNAKIEM W TRYBIE NASYCANIA WYNIKU	MMX_B = MMX_B + MMX_B ——— MMX_B + MMX_B = MMX_B
PADDSB/PADDSBR/CPADDSBcc/CPADDSBRcc A,MMX	0	0754-0759	6	DODAWANIE OŚMIU BAJTÓW ZE ZNAKIEM W TRYBIE NASYCANIA WYNIKU	A_B = A_B + MMX_B ——— A_B + MMX_B = MMX_B
PADDSW/PADDSWR/CPADDSWcc/CPADDSWRcc MMX,MMX	0	0760-0760	1	DODAWANIE CZTERECH SŁÓW ZE ZNAKIEM W TRYBIE NASYCANIA WYNIKU	MMX_W = MMX_W + MMX_W ——— MMX_W + MMX_W = MMX_W
PADDSW/PADDSWR/CPADDSWcc/CPADDSWRcc A,MMX	0	0761-0766	6	DODAWANIE CZTERECH SŁÓW ZE ZNAKIEM W TRYBIE NASYCANIA WYNIKU	A_W = A_W + MMX_W ——— A_W + MMX_W = MMX_W
PADDUSB/PADDUSBR/CPADDUSBcc/CPADDUSBRcc MMX,MMX	0	0767-0767	1	DODAWANIE OŚMIU BAJTÓW BEZ ZNAKU W TRYBIE NASYCANIA WYNIKU	MMX_B = MMX_B + MMX_B ——— MMX_B + MMX_B = MMX_B
PADDUSB/PADDUSBR/CPADDUSBcc/CPADDUSBRcc A,MMX	0	0768-0773	6	DODAWANIE OŚMIU BAJTÓW BEZ ZNAKU W TRYBIE NASYCANIA WYNIKU	A_B = A_B + MMX_B ——— A_B + MMX_B = MMX_B
PADDUSW/PADDUSWR/CPADDUSWcc/CPADDUSWRcc MMX,MMX	0	0774-0774	1	DODAWANIE CZTERECH SŁÓW BEZ ZNAKU W TRYBIE NASYCANIA WYNIKU	MMX_W = MMX_W + MMX_W ——— MMX_W + MMX_W = MMX_W
PADDUSW/PADDUSWR/CPADDUSWcc/CPADDUSWRcc A,MMX	0	0775-0780	6	DODAWANIE CZTERECH SŁÓW BEZ ZNAKU W TRYBIE NASYCANIA WYNIKU	A_W = A_W + MMX_W ——— A_W + MMX_W = MMX_W
PSUBB/PRSUBBR/CPSUBBcc/CPRSUBBRcc MMX,MMX	0	0781-0781	1	ODEJMOWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU	MMX_B = MMX_B - MMX_B ——— MMX_B - MMX_B = MMX_B
PSUBB/PRSUBBR/CPSUBBcc/CPRSUBBRcc A,MMX	0	0782-0787	6	ODEJMOWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU	A_B = A_B - MMX_B ——— MMX_B - A_B = MMX_B
PSUBW/PRSUBWR/CPSUBWcc/CPRSUBWRcc MMX,MMX	0	0788-0788	1	ODEJMOWANIE CZTERECH SŁÓW W TRYBIE ZAWIJANIA WYNIKU	MMX_W = MMX_W - MMX_W ——— MMX_W - MMX_W = MMX_W
PSUBW/PRSUBWR/CPSUBWcc/CPRSUBWRcc A,MMX	0	0789-0794	6	ODEJMOWANIE CZTERECH SŁÓW W TRYBIE ZAWIJANIA WYNIKU	A_W = A_W - MMX_W ——— MMX_W - A_W = MMX_W
PSUBD/PRSUBDR/CPSUBDcc/CPRSUBDRcc MMX,MMX	0	0795-0795	1	ODEJMOWANIE DWÓCH PODWÓJNYCH SŁÓW W TRYBIE ZAWIJANIA WYNIKU	MMX_D = MMX_D - MMX_D ——— MMX_D - MMX_D = MMX_D
PSUBD/PRSUBDR/CPSUBDcc/CPRSUBDRcc A,MMX	0	0796-0801	6	ODEJMOWANIE DWÓCH PODWÓJNYCH SŁÓW W TRYBIE ZAWIJANIA WYNIKU	A_D = A_D - MMX_D ——— MMX_D - A_D = MMX_D
PSUBSB/PRSUBSBR/CPSUBSBcc/CPRSUBSBRcc MMX,MMX	0	0802-0802	1	ODEJMOWANIE OŚMIU BAJTÓW ZE ZNAKIEM W TRYBIE NASYCANIA WYNIKU	MMX_B = MMX_B - MMX_B ——— MMX_B - MMX_B = MMX_B
PSUBSB/PRSUBSBR/CPSUBSBcc/CPRSUBSBRcc A,MMX	0	0803-0808	6	ODEJMOWANIE OŚMIU BAJTÓW ZE ZNAKIEM W TRYBIE NASYCANIA WYNIKU	A_B = A_B - MMX_B ——— MMX_B - A_B = MMX_B
PSUBSW/PRSUBSWR/CPSUBSWcc/CPRSUBSWRcc MMX,MMX	0	0809-0809	1	ODEJMOWANIE CZTERECH SŁÓW ZE ZNAKIEM W TRYBIE NASYCANIA WYNIKU	MMX_W = MMX_W - MMX_W ——— MMX_W - MMX_W = MMX_W
PSUBSW/PRSUBSWR/CPSUBSWcc/CPRSUBSWRcc A,MMX	0	0810-0815	6	ODEJMOWANIE CZTERECH SŁÓW ZE ZNAKIEM W TRYBIE NASYCANIA WYNIKU	A_W = A_W - MMX_W ——— MMX_W - A_W = MMX_W
PSUBUSB/PRSUBUSBR/CPSUBUSBcc/CPRSUBUSBRcc MMX,MMX	0	0816-0816	1	ODEJMOWANIE OŚMIU BAJTÓW BEZ ZNAKU W TRYBIE NASYCANIA WYNIKU	MMX_B = MMX_B - MMX_B ——— MMX_B - MMX_B = MMX_B
PSUBUSB/PRSUBUSBR/CPSUBUSBcc/CPRSUBUSBRcc A,MMX	0	0817-0822	6	ODEJMOWANIE OŚMIU BAJTÓW BEZ ZNAKU W TRYBIE NASYCANIA WYNIKU	A_B = A_B - MMX_B ——— MMX_B - A_B = MMX_B
PSUBUSW/PRSUBUSWR/CPSUBUSWcc/CPRSUBUSWRcc MMX,MMX	0	0823-0823	1	ODEJMOWANIE CZTERECH SŁÓW BEZ ZNAKU W TRYBIE NASYCANIA WYNIKU	MMX_W = MMX_W - MMX_W ——— MMX_W - MMX_W = MMX_W
PSUBUSW/PRSUBUSWR/CPSUBUSWcc/CPRSUBUSWRcc A,MMX	0	0824-0829	6	ODEJMOWANIE CZTERECH SŁÓW BEZ ZNAKU W TRYBIE NASYCANIA WYNIKU	A_W = A_W - MMX_W ——— MMX_W - A_W = MMX_W
PMULHW/PMULHWR/CPMULHWcc/CPMULHWRcc MMX,MMX	0	0830-0830	1	MNOŻENIE CZTERECH SŁÓW ZE ZNAKIEM, ZAPISANIE STARSZEJ CZĘŚCI WYNIKU	MMX_HW = MMX_W x MMX_W ——— MMX_W x MMX_W = MMX_HW
PMULHW/PMULHWR/CPMULHWcc/CPMULHWRcc A,MMX	0	0831-0836	6	MNOŻENIE CZTERECH SŁÓW ZE ZNAKIEM, ZAPISANIE STARSZEJ CZĘŚCI WYNIKU	A_HW = A_W x MMX_W ——— A_W x MMX_W = MMX_HW
PMULHUW/PMULHUWR/CPMULHUWcc/CPMULHUWRcc MMX,MMX	0	0837-0837	1	MNOŻENIE CZTERECH SŁÓW BEZ ZNAKU, ZAPISANIE STARSZEJ CZĘŚCI WYNIKU	MMX_HW = MMX_W x MMX_W ——— MMX_W x MMX_W = MMX_HW
PMULHUW/PMULHUWR/CPMULHUWcc/CPMULHUWRcc A,MMX	0	0838-0843	6	MNOŻENIE CZTERECH SŁÓW BEZ ZNAKU, ZAPISANIE STARSZEJ CZĘŚCI WYNIKU	A_HW = A_W x MMX_W ——— A_W x MMX_W = MMX_HW
PMULLW/PMULLWR/CPMULLWcc/CPMULLWRcc MMX,MMX	0	0844-0844	1	MNOŻENIE CZTERECH SŁÓW ZE ZNAKIEM / BEZ ZNAKU, ZAPISANIE MŁODSZEJ CZĘŚCI WYNIKU	MMX_LW = MMX_W x MMX_W ——— MMX_W x MMX_W = MMX_LW
PMULLW/PMULLWR/CPMULLWcc/CPMULLWRcc A,MMX	0	0845-0850	6	MNOŻENIE CZTERECH SŁÓW ZE ZNAKIEM / BEZ ZNAKU, ZAPISANIE MŁODSZEJ CZĘŚCI WYNIKU	A_LW = A_W x MMX_W ——— A_W x MMX_W = MMX_LW
PMADDWD/PMADDWDR/CPMADDWDWcc/CPMADDWDRcc MMX,MMX	0	0851-0851	1	MNOŻENIE Z DODAWANIEM CZTERECH SŁÓW ZE ZNAKIEM	MMX_D = MMX_W x MMX_W + MMX_W x MMX_W——— MMX_W x MMX_W + MMX_W x MMX_W = MMX_D
PMADDWD/PMADDWDR/CPMADDWDWcc/CPMADDWDRcc A,MMX	0	0852-0857	6	MNOŻENIE Z DODAWANIEM CZTERECH SŁÓW ZE ZNAKIEM	A_D = A_W x MMX_W + A_W x MMX_W——— A_W x MMX_W + A_W x MMX_W = MMX_D
PIMULE/PIMULER/CPIMULEcc/CPIMULERcc MMX A,MMX B,MMX C,MMX D	1	0149-0149	1	64 BITOWE MNOŻENIE LICZB CAŁKOWITYCH ZE ZNAKIEM	ZESPOLONY MMX AB = MMX C × MMX D ——— MMX A × MMX B = ZESPOLONY MMX CD
PMULE/PMULER/CPMULEcc/CPMULERcc MMX A,MMX B,MMX C,MMX D	1	0150-0150	1	64 BITOWE MNOŻENIE LICZB CAŁKOWITYCH BEZ ZNAKU	ZESPOLONY MMX AB = MMX C × MMX D ——— MMX A × MMX B = ZESPOLONY MMX CD

FLDZ/CFLDZcc STX	0	1177-1177	1	OBLICZANIE FUNKCJI COSINUS 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ	ARGUMENT STX=COS(STX)
FLD1/CFLD1cc STX	0	1178-1178	1	OBLICZANIE FUNKCJI TANGENS 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ	ARGUMENT STX=TAN(STX)
FLDPI/CFLDPIcc STX	0	1179-1179	1	OBLICZANIE KĄTA W UKŁADZIE WSPÓŁRZĘDNYCH POMIĘDZY DWIEMA 76 BITOWYMI LICZBAMI ZMIENNOPRZECINKOWYMI	ARGUMENT STX=ARCTAN(STX/STX)
FLDLN2/CFLDLN2cc STX	0	1180-1180	1	POTĘGOWANIE PRZY PODSTAWIE LICZBY EULERA PRZEZ 76 BITOWY WYKŁADNIK ZMIENNOPRZECINKOWY	ARGUMENT STX=e^(STX)
FLDLNT/CFLDLNTcc STX	0	1181-1181	1	OBLICZANIE LOGARYTMU O PODSTAWIE DWA 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ	ARGUMENT STX=LOG2(STX)
FLDL2E/CFLDL2Ecc STX					
FLDL2T/CFLDL2Tcc STX					
FMOV/FMOVR/CFMOVcc/CFMOVRcc					
FLDR/CFLDRcc A,STX					
FILDR/CFILDRcc A,STX					
FBLDR/CFBLDRcc A,STX					
FST/CFSTcc A,STX					
FIST/CFISTcc A,STX					
FBST/CFBSTcc A,STX					
FXCH/CFXCHcc STX,STX					
FRATAN/CFRATAN2cc STX,STX					
FEX/CFEXcc STX					
FL2X/CFL2Xcc STX					

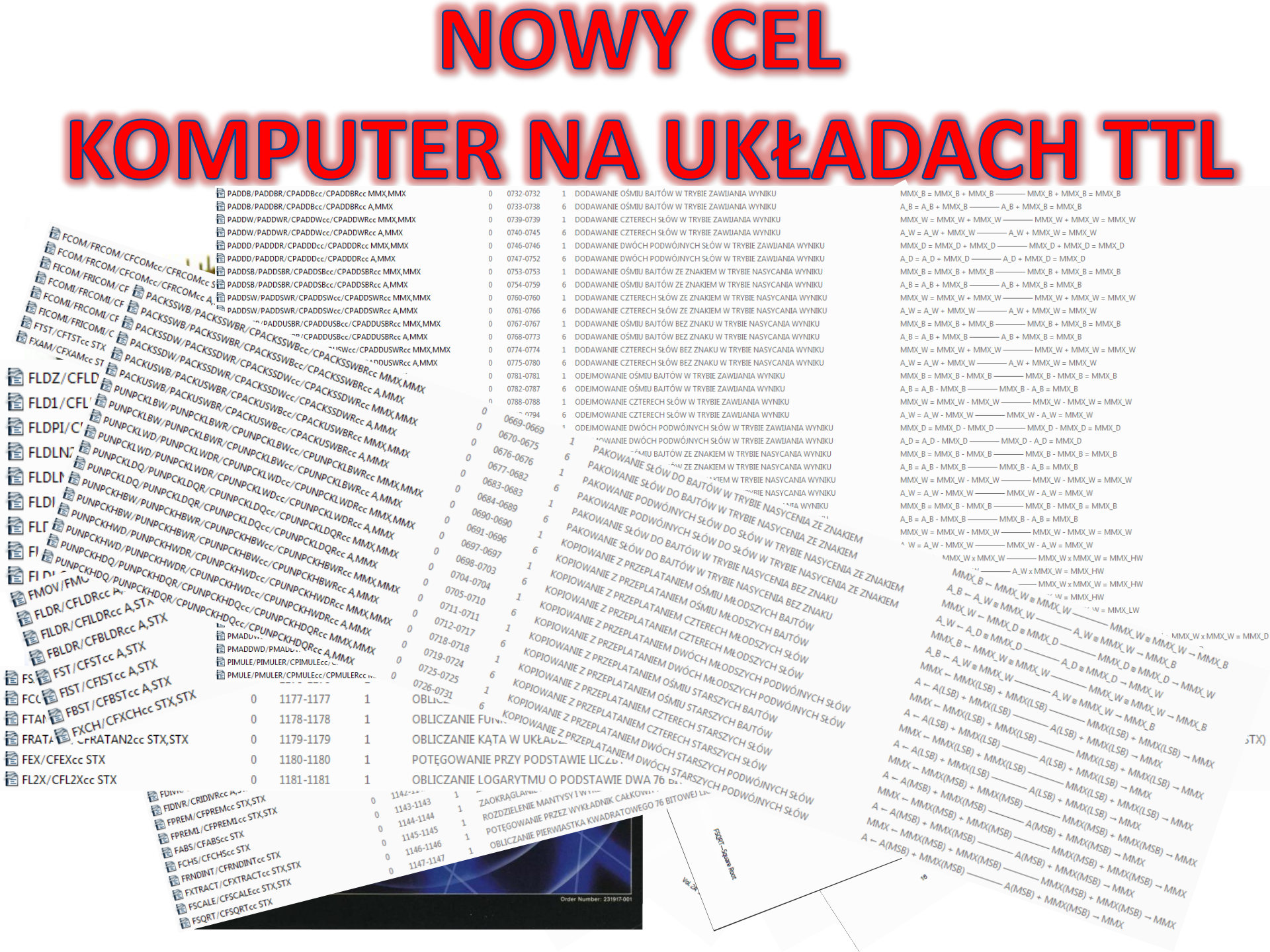
FCOM/FRCOM/CFCOMcc/CFRCOMcc STX
FCOM/FRCOM/CFCOMcc/CFRCOMcc A,STX
FICOM/FRICOM/CFICOMcc/CFRICOMcc STX
FICOM/FRICOM/CFICOMcc/CFRICOMcc A,STX
FCOMI/FRCOMI/CFCOMIcc/CFRCOMIcc STX
FICOMI/FRICOMI/CFICOMIcc/CFRICOMIcc A,STX
FTST/CFTSTcc STX
FXAM/CFXAMcc STX

FIDIVR/CRIDIVRcc A,STX
FPREM/CFPREMcc STX,STX
FPREM1/CFPREM1cc STX,STX
FABS/CFABScc STX
FCHS/CFCHScc STX
FRNDINT/CFRNDINTcc STX
FXTRACT/CFXTRACTcc STX,STX
FSCALE/CFSCALEcc STX,STX
FSQRT/CFSQRTcc STX



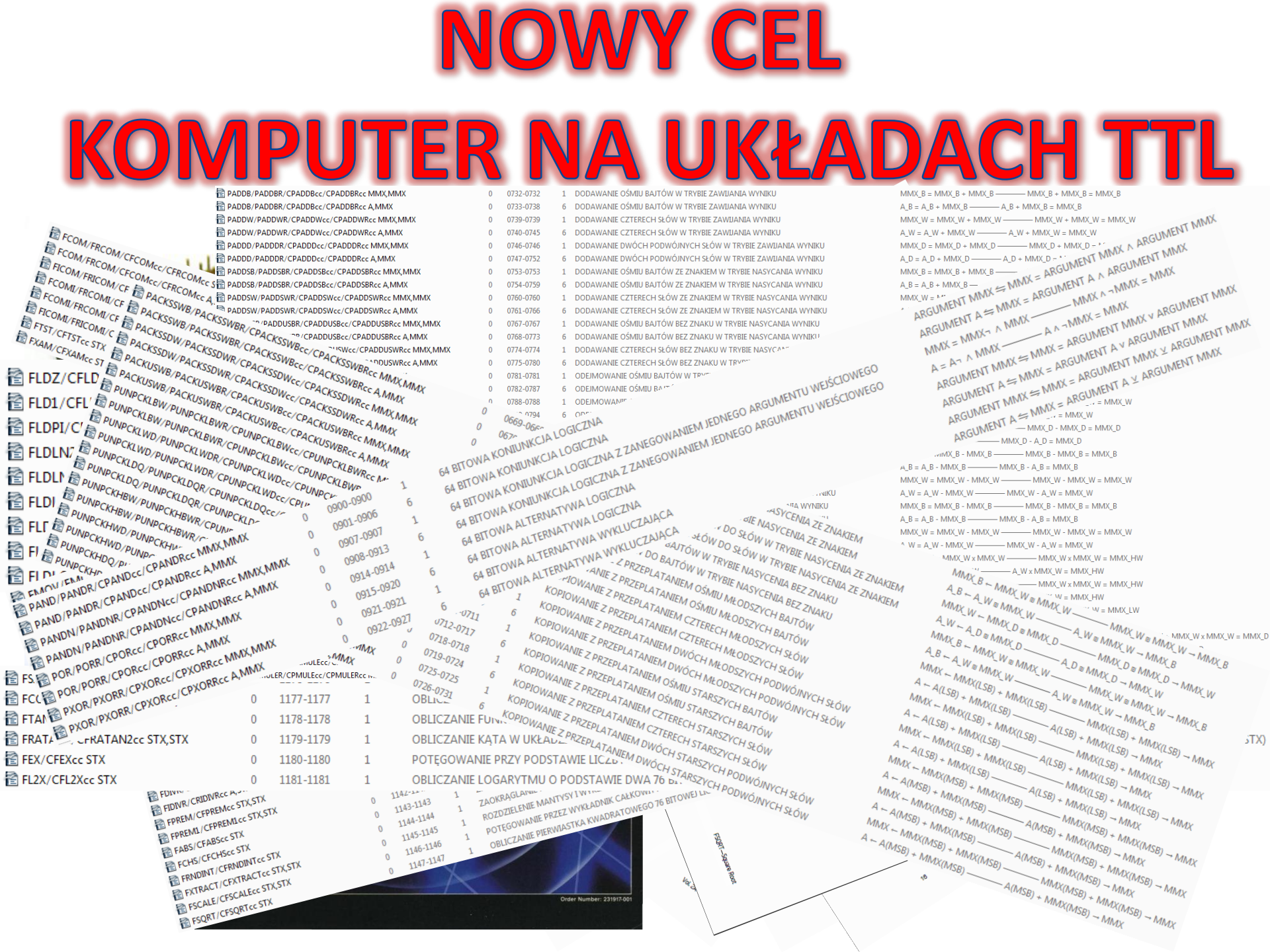
NOWY CEL

KOMPUTER NA UKŁADACH TTL



NOWY CEL

KOMPUTER NA UKŁADACH TTL



NOWY CEL

KOMPUTER NA UKŁADACH TTL

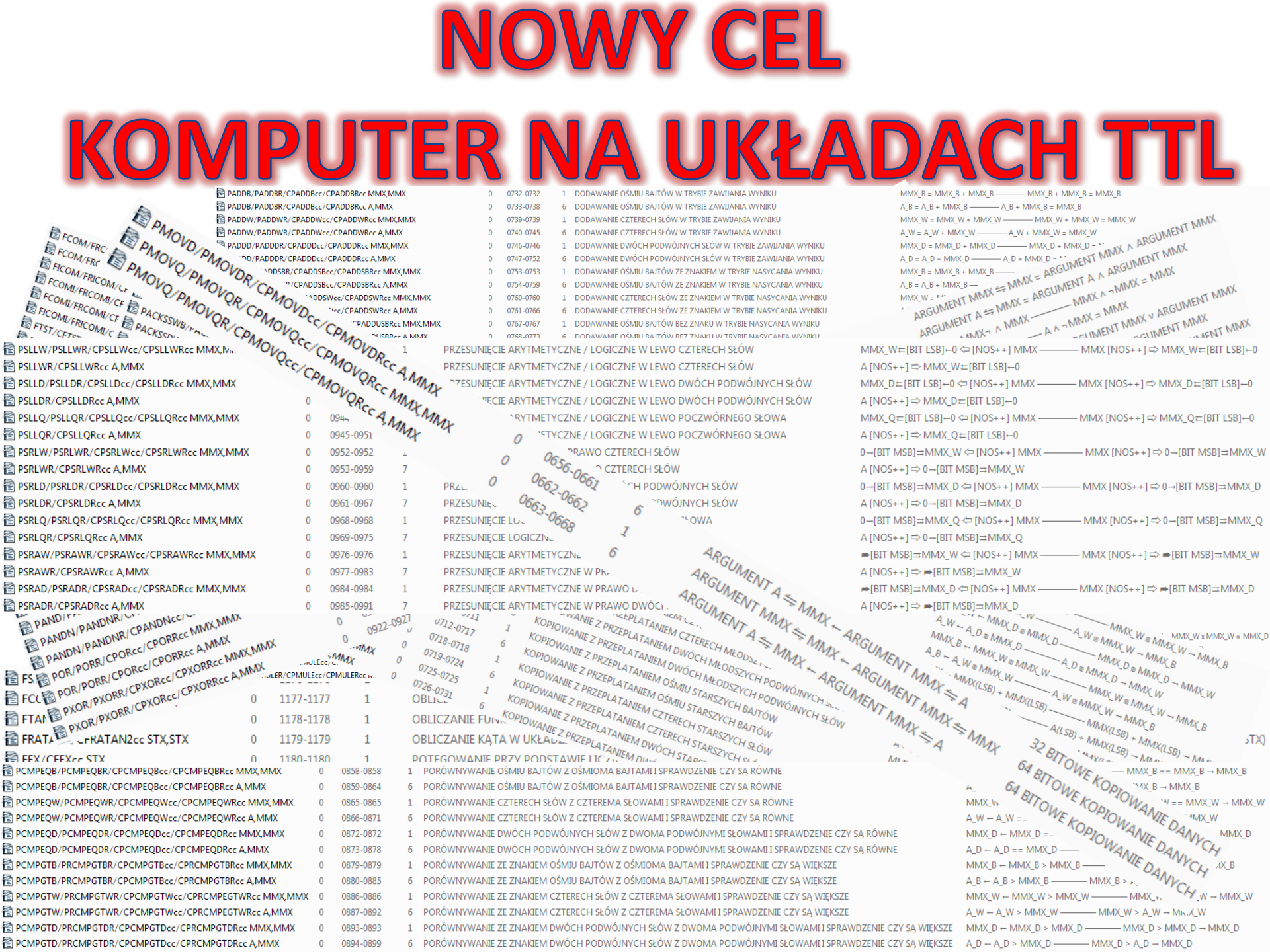
NOWY CEL

KOMPUTER NA UKŁADACH TTL

PADDB/PADDBR/CPADDBcc/CPADDBRcc MMX,MMX	0	0732-0732	1	DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU	MMX_B = MMX_B + MMX_B ——— MMX_B + MMX_B = MMX_B
PADDB/PADDBR/CPADDBcc/CPADDBRcc A,MMX	0	0733-0738	6	DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU	A_B = A_B + MMX_B ——— A_B + MMX_B = MMX_B
PADDW/PADDWR/CPADDWcc/CPADDWRcc MMX,MMX	0	0739-0739	1	DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWIJANIA WYNIKU	MMX_W = MMX_W + MMX_W ——— MMX_W + MMX_W = MMX_W
PADDW/PADDWR/CPADDWcc/CPADDWRcc A,MMX	0	0740-0745	6	DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWIJANIA WYNIKU	A_W = A_W + MMX_W ——— A_W + MMX_W = MMX_W
PADDD/PADDDR/CPADDDcc/CPADDDRcc MMX,MMX	0	0746-0746	1	DODAWANIE DWÓCH PODWÓJNYCH SŁÓW W TRYBIE ZAWIJANIA WYNIKU	MMX_D = MMX_D + MMX_D ——— MMX_D + MMX_D =
PADDD/PADDDR/CPADDDcc/CPADDDRcc A,MMX	0	0747-0752	6	DODAWANIE DWÓCH PODWÓJNYCH SŁÓW W TRYBIE ZAWIJANIA WYNIKU	A_D = A_D + MMX_D ——— A_D + MMX_D =
PADDSB/PADDSBR/CPADDSBcc/CPADDSBRcc MMX,MMX	0	0753-0753	1	DODAWANIE OŚMIU BAJTÓW ZE ZNAKIEM W TRYBIE NASYCANIA WYNIKU	MMX_B = MMX_B + MMX_B ———
PADDSB/PADDSBR/CPADDSBcc/CPADDSBRcc A,MMX	0	0754-0759	6	DODAWANIE OŚMIU BAJTÓW ZE ZNAKIEM W TRYBIE NASYCANIA WYNIKU	A_B = A_B + MMX_B —
PADDSW/PADDSWR/CPADDSWcc/CPADDSWRcc MMX,MMX	0	0760-0760	1	DODAWANIE CZTERECH SŁÓW ZE ZNAKIEM W TRYBIE NASYCANIA WYNIKU	MMX_W = M
PADDSW/PADDSWR/CPADDSWcc/CPADDSWRcc A,MMX	0	0761-0766	6	DODAWANIE CZTERECH SŁÓW ZE ZNAKIEM W TRYBIE NASYCANIA WYNIKU	
/PADDUSBR/CPADDUSBcc/CPADDUSBRcc MMX,MMX	0	0767-0767	1	DODAWANIE OŚMIU BAJTÓW BEZ ZNAKU W TRYBIE NASYCANIA WYNIKU	
/CPADDUSBRcc A,MMX	0	0768-0773	6	DODAWANIE OŚMIU BAJTÓW BEZ ZNAKU W TRYBIE NASYCANIA WYNIKU	
PSLLW/PSLLWR/CPSLLWcc/CPSLLWRcc MMX,MMX	0	0928-0928	1	PRZESUNIĘCIE ARYTMETYCZNE / LOGICZNE W LEWO CZTERECH SŁÓW	MMX_W⇐[BIT LSB]←0 ⇦ [NOS++] MMX ——— MMX [NOS++] ⇨ MMX_W⇐[BIT LSB]←0
PSLLWR/CPSLLWRcc A,MMX	0	0929-0935	7	PRZESUNIĘCIE ARYTMETYCZNE / LOGICZNE W LEWO CZTERECH SŁÓW	A [NOS++] ⇨ MMX_W⇐[BIT LSB]←0
PSLLD/PSLLDR/CPSLLDcc/CPSLLDRcc MMX,MMX	0	0936-0936	1	PRZESUNIĘCIE ARYTMETYCZNE / LOGICZNE W LEWO DWÓCH PODWÓJNYCH SŁÓW	MMX_D⇐[BIT LSB]←0 ⇦ [NOS++] MMX ——— MMX [NOS++] ⇨ MMX_D⇐[BIT LSB]←0
PSLLDR/CPSLLDRcc A,MMX	0	0937-0943	7	PRZESUNIĘCIE ARYTMETYCZNE / LOGICZNE W LEWO DWÓCH PODWÓJNYCH SŁÓW	A [NOS++] ⇨ MMX_D⇐[BIT LSB]←0
PSLLQ/PSLLQR/CPSLLQcc/CPSLLQRcc MMX,MMX	0	0944-0944	1	PRZESUNIĘCIE ARYTMETYCZNE / LOGICZNE W LEWO POCZWÓRNEGO SŁOWA	MMX_Q⇐[BIT LSB]←0 ⇦ [NOS++] MMX ——— MMX [NOS++] ⇨ MMX_Q⇐[BIT LSB]←0
PSLLQR/CPSLLQRcc A,MMX	0	0945-0951	7	PRZESUNIĘCIE ARYTMETYCZNE / LOGICZNE W LEWO POCZWÓRNEGO SŁOWA	A [NOS++] ⇨ MMX_Q⇐[BIT LSB]←0
PSRLW/PSRLWR/CPSRLWcc/CPSRLWRcc MMX,MMX	0	0952-0952	1	PRZESUNIĘCIE LOGICZNE W PRAWO CZTERECH SŁÓW	0→[BIT MSB]⇒MMX_W ⇦ [NOS++] MMX ——— MMX [NOS++] ⇨ 0→[BIT MSB]⇒MMX_W
PSRLWR/CPSRLWRcc A,MMX	0	0953-0959	7	PRZESUNIĘCIE LOGICZNE W PRAWO CZTERECH SŁÓW	A [NOS++] ⇨ 0→[BIT MSB]⇒MMX_W
PSRLD/PSRLDR/CPSRLDcc/CPSRLDRcc MMX,MMX	0	0960-0960	1	PRZESUNIĘCIE LOGICZNE W PRAWO DWÓCH PODWÓJNYCH SŁÓW	0→[BIT MSB]⇒MMX_D ⇦ [NOS++] MMX ——— MMX [NOS++] ⇨ 0→[BIT MSB]⇒MMX_D
PSRLDR/CPSRLDRcc A,MMX	0	0961-0967	7	PRZESUNIĘCIE LOGICZNE W PRAWO DWÓCH PODWÓJNYCH SŁÓW	A [NOS++] ⇨ 0→[BIT MSB]⇒MMX_D
PSRLQ/PSRLQR/CPSRLQcc/CPSRLQRcc MMX,MMX	0	0968-0968	1	PRZESUNIĘCIE LOGICZNE W PRAWO POCZWÓRNEGO SŁOWA	0→[BIT MSB]⇒MMX_Q ⇦ [NOS++] MMX ——— MMX [NOS++] ⇨ 0→[BIT MSB]⇒MMX_Q
PSRLQR/CPSRLQRcc A,MMX	0	0969-0975	7	PRZESUNIĘCIE LOGICZNE W PRAWO POCZWÓRNEGO SŁOWA	A [NOS++] ⇨ 0→[BIT MSB]⇒MMX_Q
PSRAW/PSRAWR/CPSRAWcc/CPSRAWRcc MMX,MMX	0	0976-0976	1	PRZESUNIĘCIE ARYTMETYCZNE W PRAWO CZTERECH SŁÓW	➡[BIT MSB]⇒MMX_W ⇦ [NOS++] MMX ——— MMX [NOS++] ⇨ ➡[BIT MSB]⇒MMX_W
PSRAWR/CPSRAWRcc A,MMX	0	0977-0983	7	PRZESUNIĘCIE ARYTMETYCZNE W PRAWO CZTERECH SŁÓW	A [NOS++] ⇨ ➡[BIT MSB]⇒MMX_W
PSRAD/PSRADR/CPSRADcc/CPSRADRcc MMX,MMX	0	0984-0984	1	PRZESUNIĘCIE ARYTMETYCZNE W PRAWO DWÓCH PODWÓJNYCH SŁÓW	➡[BIT MSB]⇒MMX_D ⇦ [NOS++] MMX ——— MMX [NOS++] ⇨ ➡[BIT MSB]⇒MMX_D
PSRADR/CPSRADRcc A,MMX	0	0985-0991	7	PRZESUNIĘCIE ARYTMETYCZNE W PRAWO DWÓCH PODWÓJNYCH SŁÓW	A [NOS++] ⇨ ➡[BIT MSB]⇒MMX_D
FS	0	1177-1177	1	OBLIC	
FTA	0	1178-1178	1	OBLICZANIE FUN	
FRATA	0	1179-1179	1	OBLICZANIE KĄTA W UKŁAD	
FEX/CFEXcc STX	0	1180-1180	1	POTĘGOWANIE PRZY PODSTAWIE LIC	
PCMPEQB/PCMPEQBR/CPCMPEQBcc/CPCMPEQBRcc MMX,MMX	0	0858-0858	1	PORÓWNYWANIE OŚMIU BAJTÓW Z OŚMIOMA BAJTAMI I SPRAWDZENIE CZY SĄ RÓWNE	MMX_B ← MMX_B == MMX_B ——— MMX_B == MMX_B → MMX_B
PCMPEQB/PCMPEQBR/CPCMPEQBcc/CPCMPEQBRcc A,MMX	0	0859-0864	6	PORÓWNYWANIE OŚMIU BAJTÓW Z OŚMIOMA BAJTAMI I SPRAWDZENIE CZY SĄ RÓWNE	A_B ← A_B == MMX_B ——— A_B == MMX_B → MMX_B
PCMPEQW/PCMPEQWR/CPCMPEQWcc/CPCMPEQWRcc MMX,MMX	0	0865-0865	1	PORÓWNYWANIE CZTERECH SŁÓW Z CZTEREMA SŁOWAMI I SPRAWDZENIE CZY SĄ RÓWNE	MMX_W ← MMX_W == MMX_W ——— MMX_W == MMX_W → MMX_W
PCMPEQW/PCMPEQWR/CPCMPEQWcc/CPCMPEQWRcc A,MMX	0	0866-0871	6	PORÓWNYWANIE CZTERECH SŁÓW Z CZTEREMA SŁOWAMI I SPRAWDZENIE CZY SĄ RÓWNE	A_W ← A_W == MMX_W ——— A_W == MMX_W → MMX_W
PCMPEQD/PCMPEQDR/CPCMPEQDcc/CPCMPEQDRcc MMX,MMX	0	0872-0872	1	PORÓWNYWANIE DWÓCH PODWÓJNYCH SŁÓW Z DWOMA PODWÓJNYMI SŁOWAMI I SPRAWDZENIE CZY SĄ RÓWNE	MMX_D ← MMX_D == MMX_D ——— MMX_D == MMX_D → MMX_D
PCMPEQD/PCMPEQDR/CPCMPEQDcc/CPCMPEQDRcc A,MMX	0	0873-0878	6	PORÓWNYWANIE DWÓCH PODWÓJNYCH SŁÓW Z DWOMA PODWÓJNYMI SŁOWAMI I SPRAWDZENIE CZY SĄ RÓWNE	A_D ← A_D == MMX_D ——— A_D == MMX_D → MMX_D
PCMPGTB/PRCMPGTBR/CPCMPGTBcc/CPRCMPGTBRcc MMX,MMX	0	0879-0879	1	PORÓWNYWANIE ZE ZNAKIEM OŚMIU BAJTÓW Z OŚMIOMA BAJTAMI I SPRAWDZENIE CZY SĄ WIĘKSZE	MMX_B ← MMX_B > MMX_B ——— MMX_B > MMX_B → MMX_B
PCMPGTB/PRCMPGTBR/CPCMPGTBcc/CPRCMPGTBRcc A,MMX	0	0880-0885	6	PORÓWNYWANIE ZE ZNAKIEM OŚMIU BAJTÓW Z OŚMIOMA BAJTAMI I SPRAWDZENIE CZY SĄ WIĘKSZE	A_B ← A_B > MMX_B ——— MMX_B > A_B → MMX_B
PCMPGTW/PRCMPGTWR/CPCMPGTWcc/CPRCMPEGTWRcc MMX,MMX	0	0886-0886	1	PORÓWNYWANIE ZE ZNAKIEM CZTERECH SŁÓW Z CZTEREMA SŁOWAMI I SPRAWDZENIE CZY SĄ WIĘKSZE	MMX_W ← MMX_W > MMX_W ——— MMX_W > MMX_W → MMX_W
PCMPGTW/PRCMPGTWR/CPCMPGTWcc/CPRCMPEGTWRcc A,MMX	0	0887-0892	6	PORÓWNYWANIE ZE ZNAKIEM CZTERECH SŁÓW Z CZTEREMA SŁOWAMI I SPRAWDZENIE CZY SĄ WIĘKSZE	A_W ← A_W > MMX_W ——— MMX_W > A_W → MMX_W
PCMPGTD/PRCMPGTDR/CPCMPGTDcc/CPRCMPGTDRcc MMX,MMX	0	0893-0893	1	PORÓWNYWANIE ZE ZNAKIEM DWÓCH PODWÓJNYCH SŁÓW Z DWOMA PODWÓJNYMI SŁOWAMI I SPRAWDZENIE CZY SĄ WIĘKSZE	MMX_D ← MMX_D > MMX_D ——— MMX_D > MMX_D → MMX_D
PCMPGTD/PRCMPGTDR/CPCMPGTDcc/CPRCMPGTDRcc A,MMX	0	0894-0899	6	PORÓWNYWANIE ZE ZNAKIEM DWÓCH PODWÓJNYCH SŁÓW Z DWOMA PODWÓJNYMI SŁOWAMI I SPRAWDZENIE CZY SĄ WIĘKSZE	A_D ← A_D > MMX_D ——— MMX_D > A_D → MMX_D

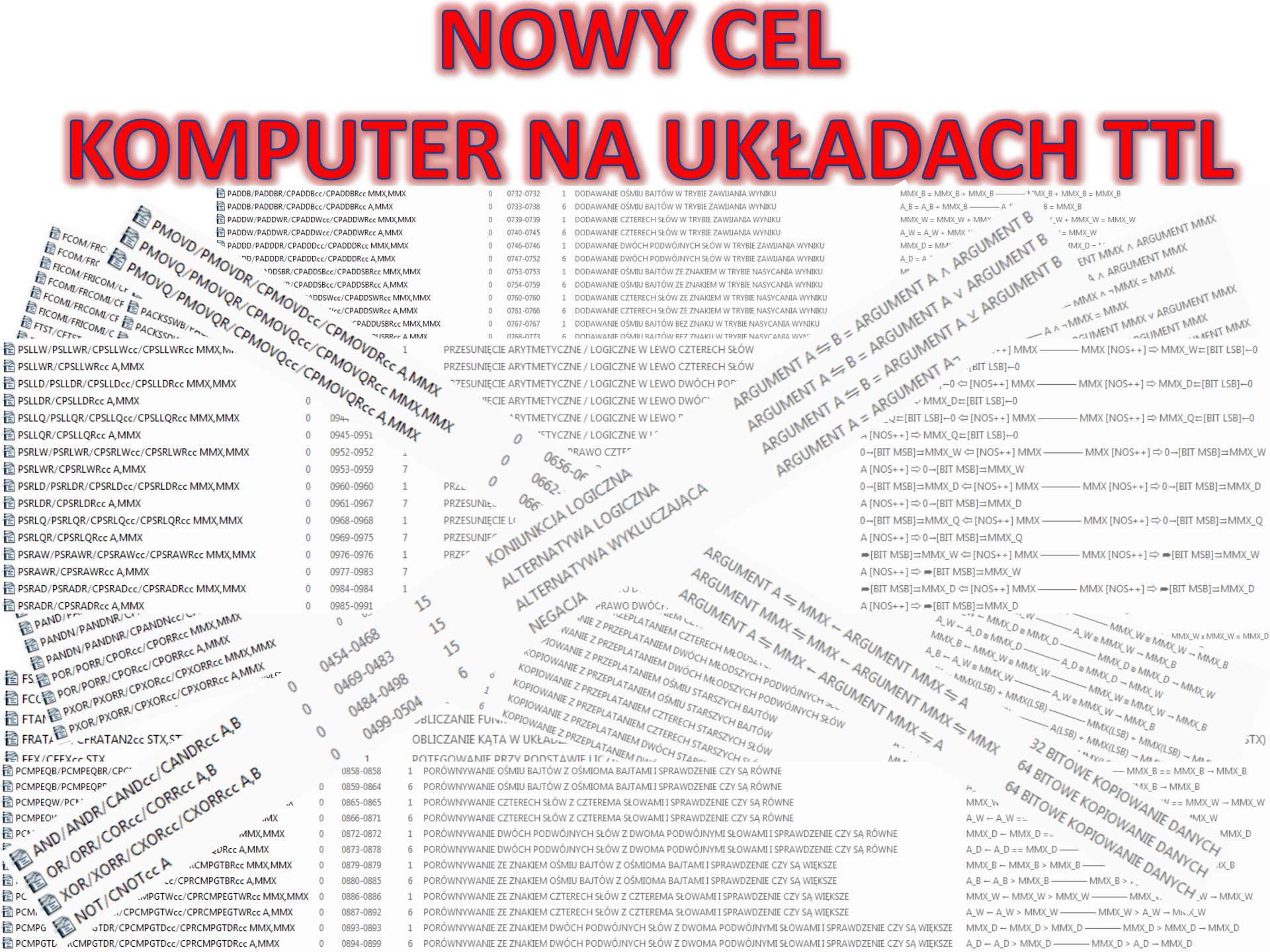
NOWY CEL

KOMPUTER NA UKŁADACH TTL



NOWY CEL

KOMPUTER NA UKŁADACH TTL



NOWY CEL

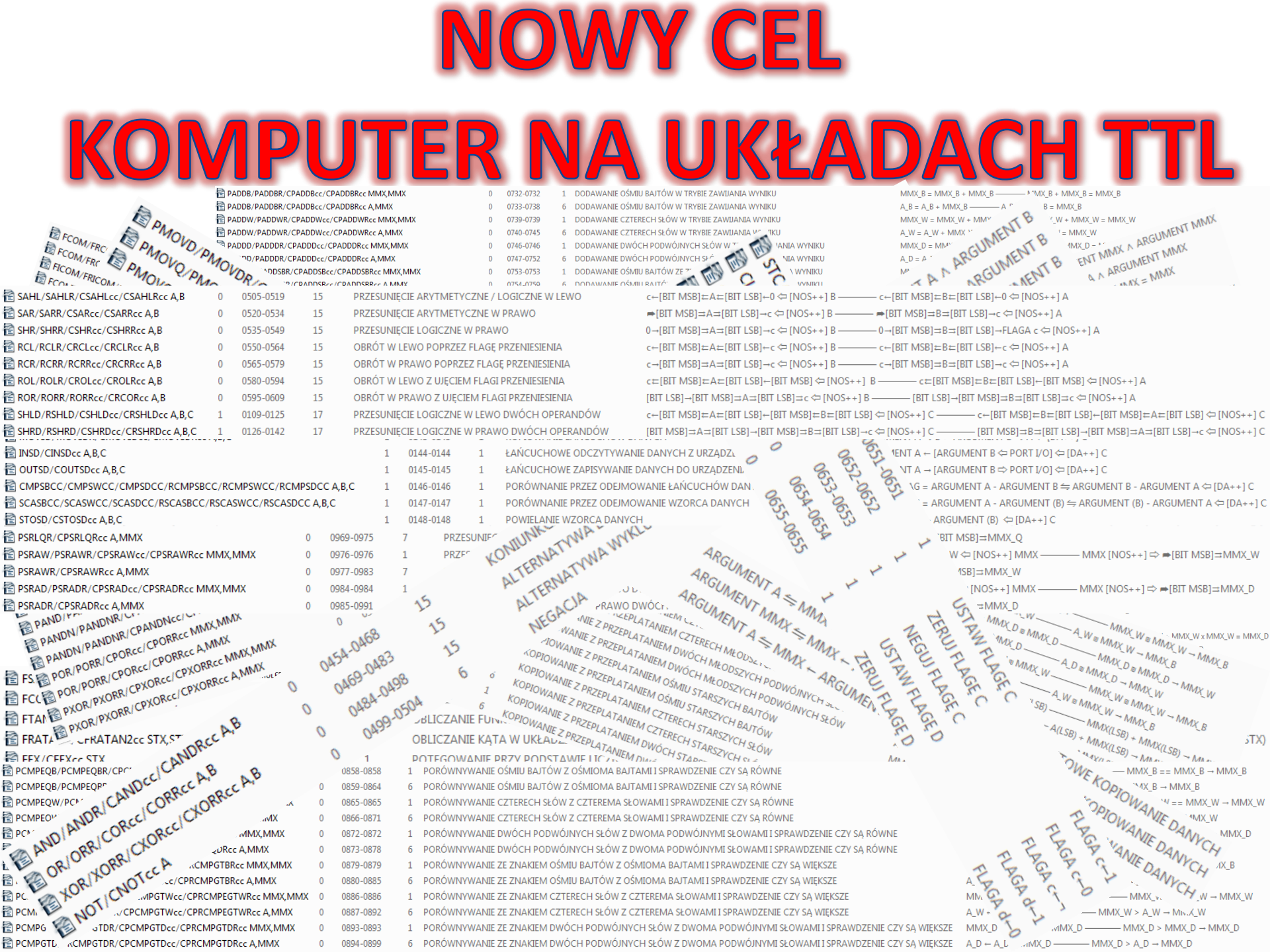
KOMPUTER NA UKŁADACH TTL

NOWY CEL

KOMPUTER NA UKŁADACH TTL

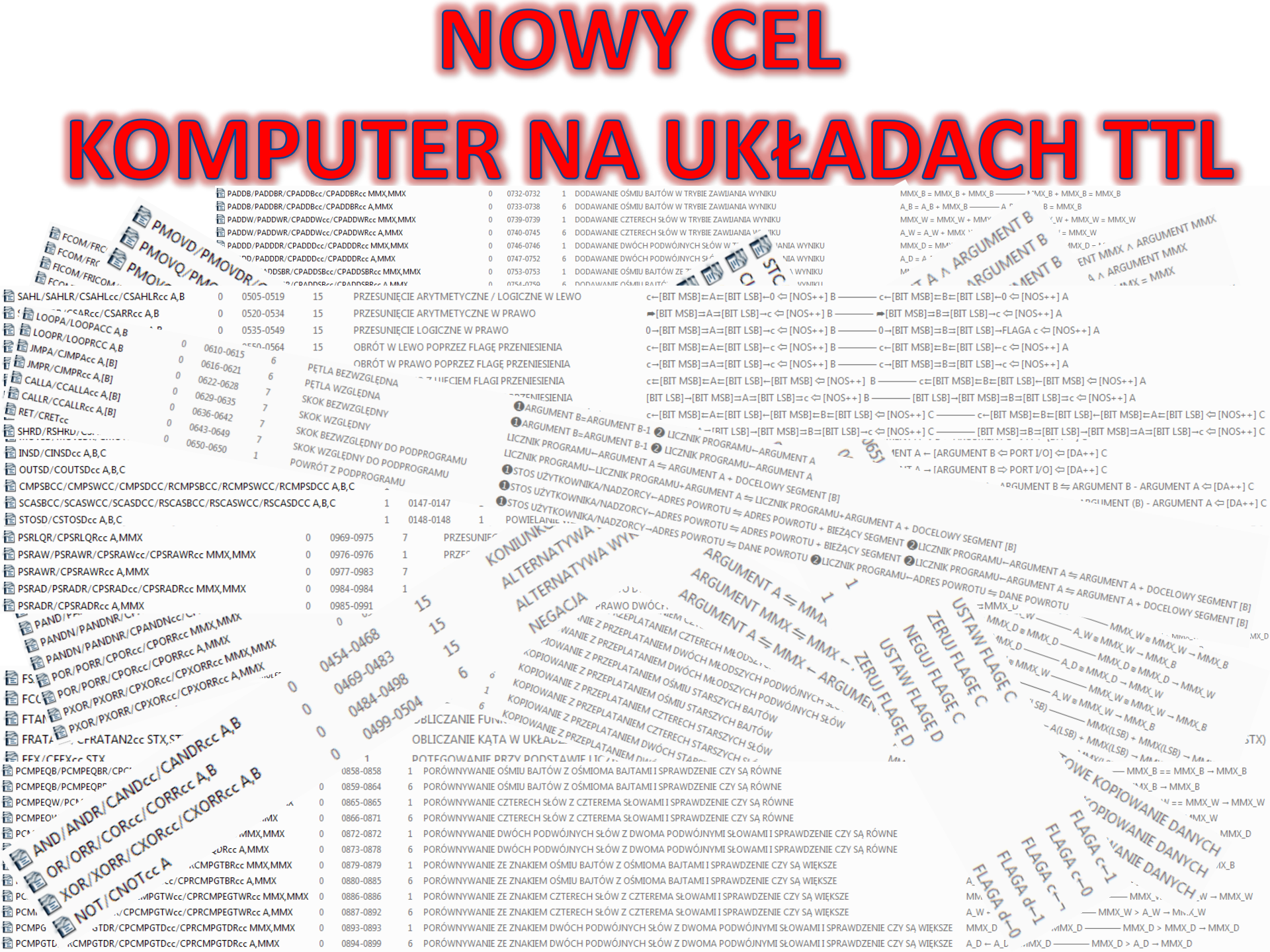
NOWY CEL

KOMPUTER NA UKŁADACH TTL



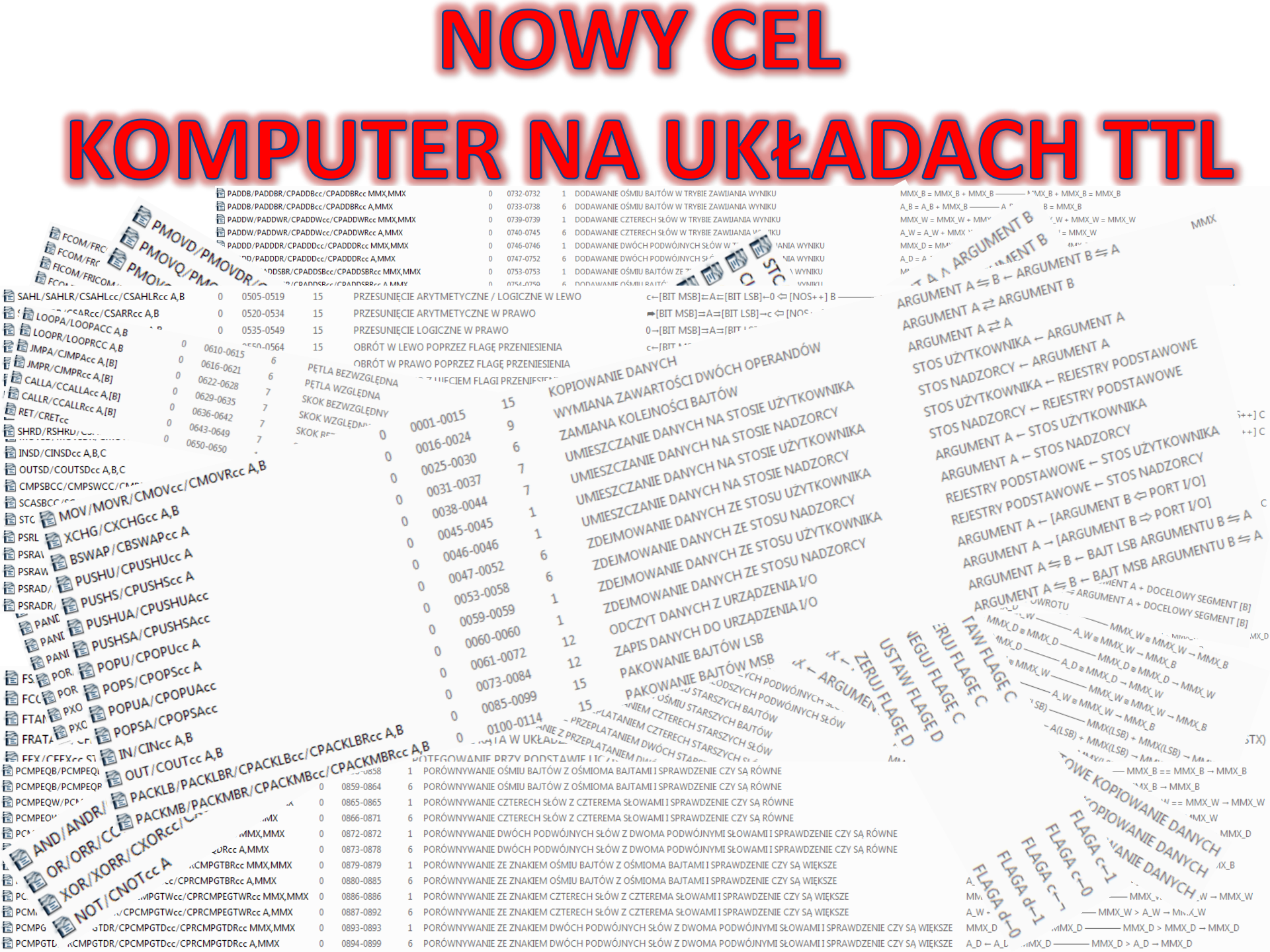
NOWY CEL

KOMPUTER NA UKŁADACH TTL



NOWY CEL

KOMPUTER NA UKŁADACH TTL



NOWY CEL

KOMPUTER NA UKŁADACH TTL

NOWY CEL

KOMPUTER NA UKŁADACH TTL

64 BITOWE KOPIOWANIE DANYCH

PMOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MMX,MMX

RELACJE: MMX←MMX MMX→MMX

Instrukcje działają w następujący sposób:
64 bitowa zawartość argumentu źródłowego **MMX** kopiowana jest do 64 bitowej zawartości argumentu docelowego **MMX**.
Operacje te odbywają się tradycyjnie lub po spełnieniu jednego z 16 argumentów funkcji warunkowej.
Dokładny opis zależności argumentów warunkowych wraz z praktycznymi przykładami można znaleźć w dokumencie: „Instrukcje warunkowe".

Mnemonik instrukcji cc	Opis	Warunek	Zależność	Kod Funkcji warunkowej
==	Wykonanie, jeśli równe/jeśli zero **(argumenty uniwersalne)**	Z=1	A=B lub B=A A-B=0 lub B-A=0	0001
!=	Wykonanie, jeśli różne/jeśli różne od zera **(argumenty uniwersalne)**	Z=0	A≠B lub B≠A A-B≠0 lub B-A≠0	0010
>	Wykonanie, jeśli powyżej/jeśli nie poniżej i nie równe **(argumenty dotyczące liczb bez znaku)**	C=0 i Z=0	A>B lub B>A A⧹B lub B⧹A	0011
>=	Wykonanie, jeśli powyżej lub równe/jeśli nie poniżej **(argumenty dotyczące liczb bez znaku)**	C=0	A≥B lub B≥A A↚B lub B↚A	0100
<	Wykonanie, jeśli poniżej/jeśli nie powyżej i nie równe **(argumenty dotyczące liczb bez znaku)**	C=1	A<B lub B<A A↛B lub B↛A	0101
<=	Wykonanie, jeśli poniżej lub równe/jeśli nie powyżej **(argumenty dotyczące liczb bez znaku)**	Z=1 lub C=1	A≤B lub B≤A A↛B lub B↛A	0110
S>	Wykonanie, jeśli większe/jeśli nie mniejsze i nie równe **(argumenty dotyczące liczb ze znakiem)**	Z=0 i S=O	A>B lub B>A A⧹B lub B⧹A	0111
S>=	Wykonanie, jeśli większe lub równe/jeśli nie mniejsze **(argumenty dotyczące liczb ze znakiem)**	S=O	A≥B lub B≥A A↚B lub B↚A	1000
S<	Wykonanie, jeśli mniejsze/jeśli nie większe i nie równe **(argumenty dotyczące liczb ze znakiem)**	S≠O	A<B lub B<A A↛B lub B↛A	1001
S<=	Wykonanie, jeśli mniejsze lub równe/jeśli nie większe **(argumenty dotyczące liczb ze znakiem)**	Z=1 lub S≠O	A≤B lub B≤A A↛B lub B↛A	1010
C	Wykonanie, jeśli było przeniesienie	C=1		0101

NOWY CEL

KOMPUTER NA UKŁADACH TTL

64 BITOWE KOPIOWANIE DANYCH

PMOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MMX,MMX

RELACJE: MMX←MMX MMX→MMX

Instrukcje działają w następujący sposób:
64 bitowa zawartość argumentu źródłowego **MMX** kopiowana jest do 64 bitowej zawartości argumentu docelowego **MMX**.
Operacje te odbywają się tradycyjnie lub po spełnieniu jednego z 16 argumentów funkcji warunkowej.
Dokładny opis zależności argumentów warunkowych wraz z praktycznymi przykładami można znaleźć w dokumencie: „Instrukcje warunkowe".

Mnemonik instruk...
==
!=
>
>=
<
<=
S>
S>=
S<
S<=
C

64 BITOWE KOPIOWANIE DANYCH

1. PMOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MMX,MMX

UPRZYWILEJOWANIA:	KONTEKST OBSŁUGI PAMIĘCI OPERACYJNEJ:		
	BEZ STRONICOWANIA PAMIĘCI	STRONICOWANIE PAMIĘCI	STRONICOWANIE PAMIĘCI Z SEGMENTACJĄ
TRYB UŻYTKOWNIKA	✗	✓	✓
TRYB NADZORCY	✓	✓	✓

Instrukcja jest jednowierszowa (4 bajtowa).

64 bitowa zawartość rejestru źródłowego **MMX** zostaje skopiowana do 64 bitowej zawartości rejestru docelowego **MMX**.

MMX = MM0,MM1,MM2,MM3,MM4,MM5,MM6,MM7.

REJESTR MMX←REJESTR MMX
REJESTR MMX→REJESTR MMX

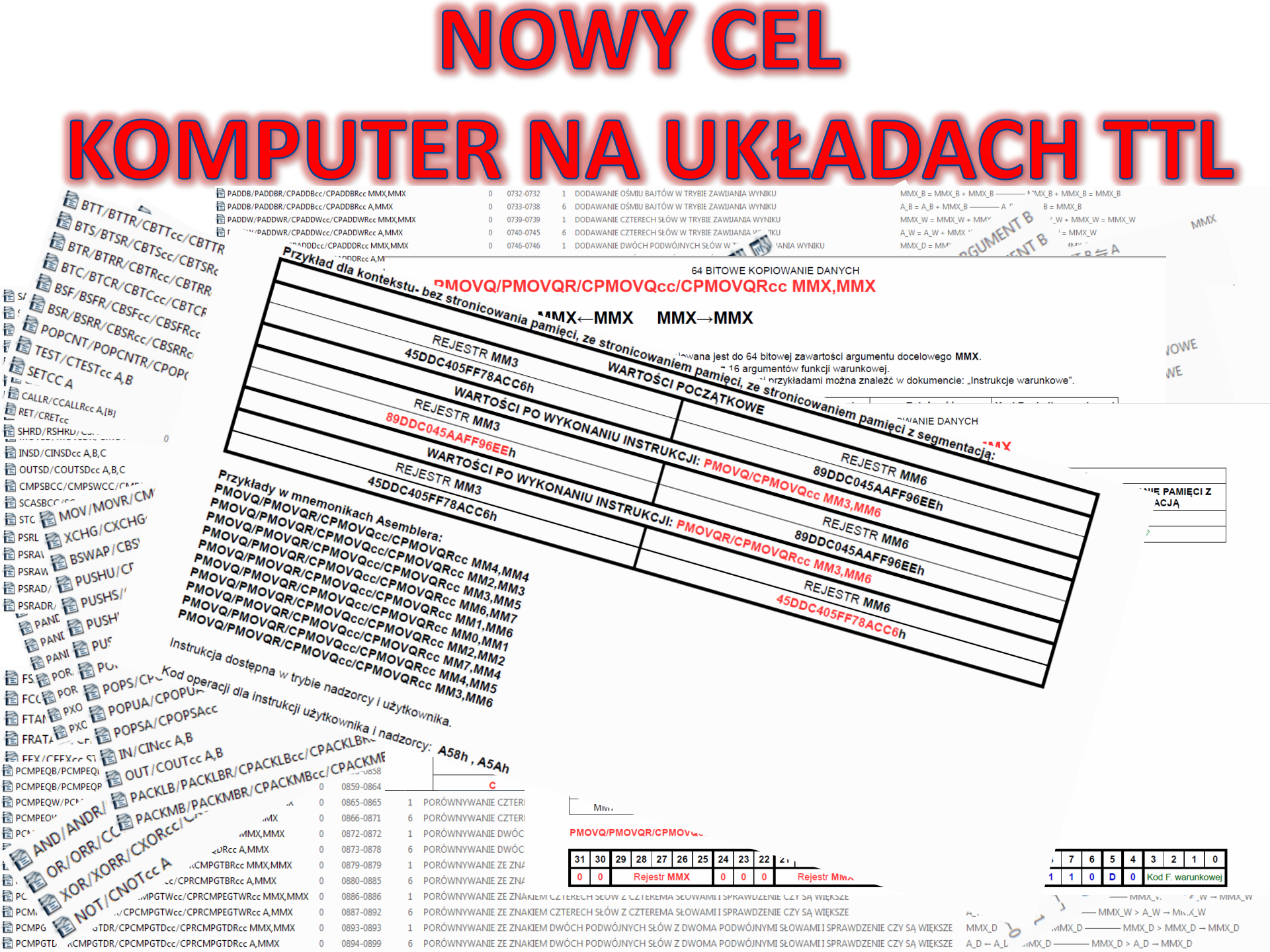
Rejestr **MMX**	Kod rejestru **MMX**	Rejestr **MMX**	Kod rejestru **MMX**
MM0	10111	MM0	10111
MM1	11000	MM1	11000
MM2	11001	MM2	11001
MM3	11010	MM3	11010
MM4	11011	MM4	11011
MM5	11100	MM5	11100
MM6	11101	MM6	11101
MM7	11110	MM7	11110

PMOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MMX,MMX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	Rejestr MMX					0	0	0	Rejestr MMX					0	1	0	1	0	0	1	0	1	1	0	D	0	Kod F. warunkowej			

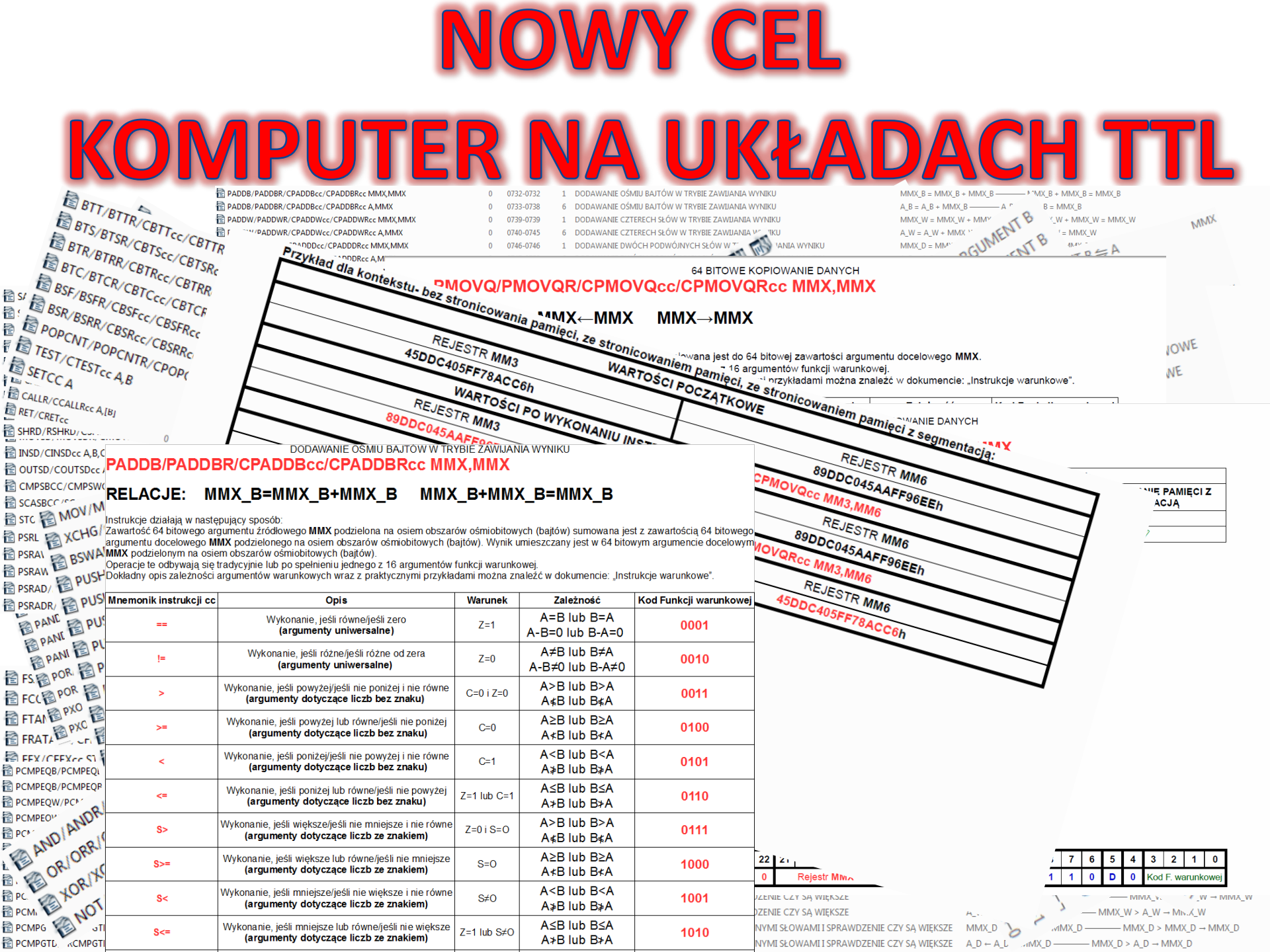
NOWY CEL

KOMPUTER NA UKŁADACH TTL



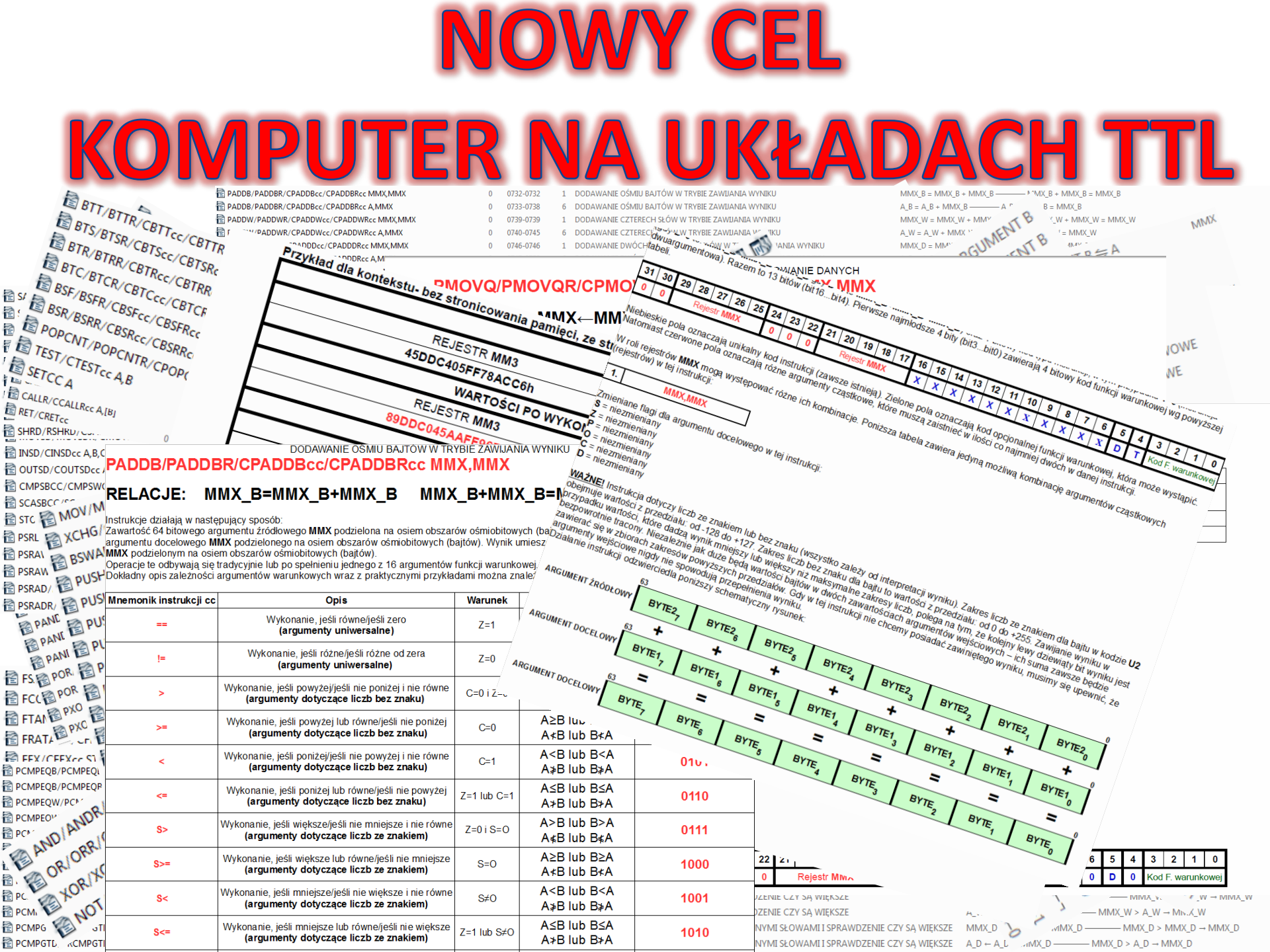
NOWY CEL

KOMPUTER NA UKŁADACH TTL



NOWY CEL

KOMPUTER NA UKŁADACH TTL



NOWY CEL

KOMPUTER NA UKŁADACH TTL

DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU

1. PADDB/PADDBR/CPADDBcc/CPADDBRcc MMX,MMX

UPRZYWILEJOWANIA:	KONTEKST OBSŁUGI PAMIĘCI OPERACYJNEJ:		
	BEZ STRONICOWANIA PAMIĘCI	STRONICOWANIE PAMIĘCI	STRONICOWANIE PAMIĘCI Z SEGMENTACJĄ
TRYB UŻYTKOWNIKA	✗	✓	✓
TRYB NADZORCY	✓	✓	✓

Instrukcja jest jednowierszowa (4 bajtowa).

Zawartość 64 bitowego rejestru źródłowego **MMX** podzielona na osiem obszarów ośmiobitowych (bajtów) sumowana jest z zawartością 64 bitowego rejestru docelowego **MMX** podzielonego na osiem obszarów ośmiobitowych (bajtów). Wynik umieszczany jest w 64 bitowym rejestrze docelowym **MMX** podzielonym na osiem obszarów ośmiobitowych (bajtów).

MMX = MM0,MM1,MM2,MM3,MM4,MM5,MM6,MM7.

REJESTR MMX_B=REJESTR MMX_B+REJESTR MMX_B
REJESTR MMX_B+REJESTR MMX_B=REJESTR MMX_B

Rejestr **MMX**	Kod rejestru **MMX**	Rejestr **MMX**	Kod rejestru **MMX**
MM0	10111	MM0	10111
MM1	11000	MM1	11000
MM2	11001	MM2	11001
MM3	11010	MM3	11010
MM4	11011	MM4	11011
MM5	11100	MM5	11100
MM6	11101	MM6	11101
MM7	11110	MM7	11110

PADDB/PADDBR/CPADDBcc/CPADDBRcc MMX,MMX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	Rejestr **MMX**					0	0	0	Rejestr **MMX**					0	1	0	1	1	0	1	1	1	0	0	D	0	Kod F. warunkowej			

najmłodsze 4 bity (bit3 ..bit0) zawierają 4 bitowy kod funkcji warunkowej wg powyższej

Zielone pola oznaczają kod opcjonalnej funkcji warunkowej, która moze wystąpić
re muszą zaistnieć w ilości co najmniej dwóch w danej instrukcji.

za tabela zawiera jedyną możliwą kombinację argumentów cząstkowych

erpretacji wyniku). Zakres liczb ze znakiem dla bajtu w kodzie **U2**
wartości z przedziału: od 0 do +255. Zawijanie wyniku w
liczb, polega na tym, że kolejny lewy dziewiąty bit wyniku jest
ach argumentów wejściowych – ich suma zawsze będzie
cemy posiadać zawiniętego wyniku, musimy się upewnić, że

<=	Wykonanie, jeśli poniżej lub równe/jeśli nie powyżej **(argumenty dotyczące liczb bez znaku)**	Z=1 lub C=1	A≤B lub B≤A A⊁B lub B⊁A	0110
S>	Wykonanie, jeśli większe/jeśli nie mniejsze i nie równe **(argumenty dotyczące liczb ze znakiem)**	Z=0 i S=O	A>B lub B>A A⊄B lub B⊄A	0111
S>=	Wykonanie, jeśli większe lub równe/jeśli nie mniejsze **(argumenty dotyczące liczb ze znakiem)**	S=O	A≥B lub B≥A A⊄B lub B⊄A	1000
S<	Wykonanie, jeśli mniejsze/jeśli nie większe i nie równe **(argumenty dotyczące liczb ze znakiem)**	S≠O	A<B lub B<A A⊁B lub B⊁A	1001
S<=	Wykonanie, jeśli mniejsze lub równe/jeśli nie większe **(argumenty dotyczące liczb ze znakiem)**	Z=1 lub S≠O	A≤B lub B≤A A⊁B lub B⊁A	1010

NOWY CEL

KOMPUTER NA UKŁADACH TTL

DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU

1. PADDB/PADDBR/CPADDBcc/CPADDBRcc MMX,MMX

UPRZYWILEJOWANIA:	KONTEKST OBSŁUGI PAMIĘCI OPE…	
	BEZ STRONICOWANIA PAMIĘCI	STRONICOWANI… ANIE PAMIĘCI Z …TACJĄ
TRYB UŻYTKOWNIKA	✗	
TRYB NADZORCY	✓	

Instrukcja jest jednowierszowa (4 bajtowa).

Zawartość 64 bitowego rejestru źródłowego **MMX** podzielona … wego rejestru docelowego **MMX** podzielonego na osiem obsza… …n **MMX** podzielonym na osiem obszarów ośmiobitowych (ba…

MMX = MM0,MM1,MM2,MM3,MM4,MM5,MM6,MM7

REJESTR MMX_B=REJESTR M…
REJESTR MMX_B+REJESTR MM…

Rejestr **MMX**	Kod rejestru **MMX**	Rejestr **MMX**	Kod
MM0	10111	MM0	
MM1	11000	MM1	11…
MM2	11001	MM2	110…
MM3	11010	MM3	11010
MM4	11011	MM4	11011
MM5	11100	MM5	11100
MM6	11101	MM6	11101
MM7	11110	MM7	11110

PADDB/PADDBR/CPADDBcc/CPADDBRcc MMX,MMX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14
0	0	Rejestr MMX					0	0	0	Rejestr MMX					0	1	0

	(argumenty dotyczące liczb bez znaku)		A⩾B lub B…
<=	Wykonanie, jeśli poniżej lub równe/jeśli nie powyżej (argumenty dotyczące liczb bez znaku)	Z=1 lub C=1	A≤B lub B≤… A⩾B lub B⩾A
S>	Wykonanie, jeśli większe/jeśli nie mniejsze i nie równe (argumenty dotyczące liczb ze znakiem)	Z=0 i S=O	A>B lub B>A A⩽B lub B⩽A
S>=	Wykonanie, jeśli większe lub równe/jeśli nie mniejsze (argumenty dotyczące liczb ze znakiem)	S=O	A≥B lub B≥A A⩽B lub B⩽A
S<	Wykonanie, jeśli mniejsze/jeśli nie większe i nie równe (argumenty dotyczące liczb ze znakiem)	S≠O	A<B lub B<A A⩾B lub B⩾A
S<=	Wykonanie, jeśli mniejsze lub równe/jeśli nie większe (argumenty dotyczące liczb ze znakiem)	Z=1 lub S≠O	A≤B lub B≤A A⩾B lub B⩾A

Przykłady w mnemonikach Asemblera:
PADDB/PADDBR/CPADDBcc/CPADDBRcc MM0,MM1
PADDB/PADDBR/CPADDBcc/CPADDBRcc MM2,MM3
PADDB/PADDBR/CPADDBcc/CPADDBRcc MM4,MM5
PADDB/PADDBR/CPADDBcc/CPADDBRcc MM6,MM7
45 DD C4 05 FF 78 AC C6h
CE BA 84 4A A9 77 42 B4h

Kod operacji dla instrukcji użytkownika i nadzorcy
Instrukcja dostępna w trybie nadzorcy i użytkownika
B70h - B72h

PADDB/PADDBR/CPADDBcc/CPADDBRcc MMX,MMX

Przykład dla kontekstu: bez stronicowania pamięci, ze stronicowaniem pamięci z segmentacją:
REJESTR MM3
WARTOŚCI PO WYKONANIU INSTRUKCJI: PADDB/PADDBR/CPADDBcc/CPADDBRcc MM3,MM6
REJESTR MM6
WARTOŚCI POCZĄTKOWE

Kod F. warunkowej

najmłodsze 4 bity (bit3…bit0) zawierają 4 bitowy kod funkcji warunkowej wg powyższej

Zielone pola oznaczają kod opcjonalnej funkcji warunkowej, która może wystąpić
re muszą zaistnieć w ilości co najmniej dwóch w danej instrukcji.
za tabela zawiera jedyną możliwą kombinację argumentów cząstkowych

akres liczb ze znakiem dla bajtu w kodzie **U2**
od 0 do +255. Zawijanie wyniku w
kolejny lewy dziewiąty bit wyniku jest
ych – ich suma zawsze będzie
wyniku, musimy się upewnić, że

BYTE2 BYTE1 BYTE0

6	5	4	3	2	1	0
0	D	0	Kod F. warunkowej			

MMX_B = MMX_B + MMX_B ——— MMX_B + MMX_B = MMX_B
A_B = A_B + MMX_B ——— A… B = MMX_B
MMX_W = MMX_W + MMX… …_W + MMX_W = MMX_W
A_W = A_W + MMX… …= MMX_W
MMX_D = MMX…

MMX_W > A_W → MMX_W
MMX_D ——— MMX_D > MMX_D → MMX_D
A_D ← A_L… MMX_D ——— MMX_D > A_D → MMX_D

…NYMI SŁOWAMI I SPRAWDZENIE CZY SĄ WIĘKSZE

NOWY CEL

KOMPUTER NA UKŁADACH TTL

DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU

1. PADDB/PADDBR/CPADDBcc/CPADDBRcc MMX,MMX

SUMOWANIE DWÓCH 76 BITOWYCH LICZB ZMIENNOPRZECINKOWYCH

FADD/FADDR/CFADDcc/CFADDRcc STX,STX

RELACJE: STX=STX+STX STX+STX=STX

Instrukcje działają w następujący sposób:
76 bitowa zawartość argumentu źródłowego **STX** sumowana jest z 76 bitową zawartością argumentu docelowego **STX**. Wynik umieszczany jest w 76 bitowym argumencie docelowym **STX**.
Operacje te odbywają się tradycyjnie lub po spełnieniu jednego z 16 argumentów zmiennoprzecinkowej funkcji warunkowej.
Dokładny opis zależności argumentów warunkowych wraz z praktycznymi przykładami można znaleźć w dokumencie: „Instrukcje warunkowe".

Mnemonik instrukcji cc	Opis	Flagi warunku: C3	C2	C1	C0	Zależność	Kod Funkcji warunkowej
==	Wykonanie, jeśli równe	0	0	0	1	A=B lub B=A	0001
>	Wykonanie, jeśli większe	0	0	1	0	A>B lub B>A	0010
<	Wykonanie, jeśli mniejsze	0	0	1	1	A<B lub B<A	0011
+SN	Wykonanie, jeśli +SNaN	0	1	0	0	A=+SNaN	0100
-SN	Wykonanie, jeśli -SNaN	0	1	0	1	A=-SNaN	0101
+QN	Wykonanie, jeśli +QNaN	0	1	1	0	A=+QNaN	0110
-QN	Wykonanie, jeśli -QNaN	0	1	1	1	A=-QNaN	0111
+I	Wykonanie, jeśli dodatnia nieskończona liczba	1	0	0	0	A=+ ∞	1000
-I	Wykonanie, jeśli ujemna nieskończona liczba	1	0	0	1	A=- ∞	1001
+N	Wykonanie, jeśli dodatnia znormalizowana liczba	1	0	1	0	A=+N	1010
-N	Wykonanie, jeśli ujemna znormalizowana liczba	1	0	1	1	A=-N	1011
+D	Wykonanie, jeśli dodatnia zdenormalizowana liczba	1	1	0	0	A=+D	1100
-D	Wykonanie, jeśli ujemna zdenormalizowana liczba	1	1	0	1	A=-D	1101
+Z	Wykonanie, jeśli dodatnie zero	1	1	1	0	A=+0	1110
-Z	Wykonanie, jeśli ujemne zero	1	1	1	1	A=-0	1111

Kodem operacji dla instrukcji standardowej jest: 11 bitowy kod instrukcji, 1 bitowy kod kierunku dla argumentów wykonywanej instrukcji (gdy **D=0** to: **STX=STX+STX** gdy **D=1** to **STX+STX=STX**) oraz 1 bitowy kod typu instrukcji, w tym przypadku **T=0** (instrukcja dwuargumentowa). Razem to 13 bitów (bit16...bit4). Pierwsze najmłodsze 4 bity (bit3...bit0) zawierają 4 bitowy kod funkcji warunkowej, który w tym przypadku jest nieaktywny i musi zawierać cztery logiczne zera!

<=	Wykonanie, jeśli poniżej lub równe/jeśli nie powyżej **(argumenty dotyczące liczb bez znaku)**	Z=1 lub C=1	A≤B lub B≤A A≯B lub B≯A
S>	Wykonanie, jeśli większe/jeśli nie mniejsze i nie równe **(argumenty dotyczące liczb ze znakiem)**	Z=0 i S=O	A>B lub B>A A≤B lub B≤A
S>=	Wykonanie, jeśli większe lub równe/jeśli nie mniejsze **(argumenty dotyczące liczb ze znakiem)**	S=O	A≥B lub B≥A A<B lub B<A
S<	Wykonanie, jeśli mniejsze/jeśli nie większe i nie równe **(argumenty dotyczące liczb ze znakiem)**	S≠O	A<B lub B<A A≱B lub B≱A
S<=	Wykonanie, jeśli mniejsze lub równe/jeśli nie większe **(argumenty dotyczące liczb ze znakiem)**	Z=1 lub S≠O	A≤B lub B≤A A≯B lub B≯A

najmłodsze 4 bity (bit3...bit0) zawierają 4 bitowy kod funkcji warunkowej wg powyższej

Zielone pola oznaczają kod opcjonalnej funkcji warunkowej, która może wystąpić
re muszą zaistnieć w ilości co najmniej dwóch w danej instrukcji.

za tabela zawiera jedyną możliwą kombinację argumentów cząstkowych

Kod F. warunkowej

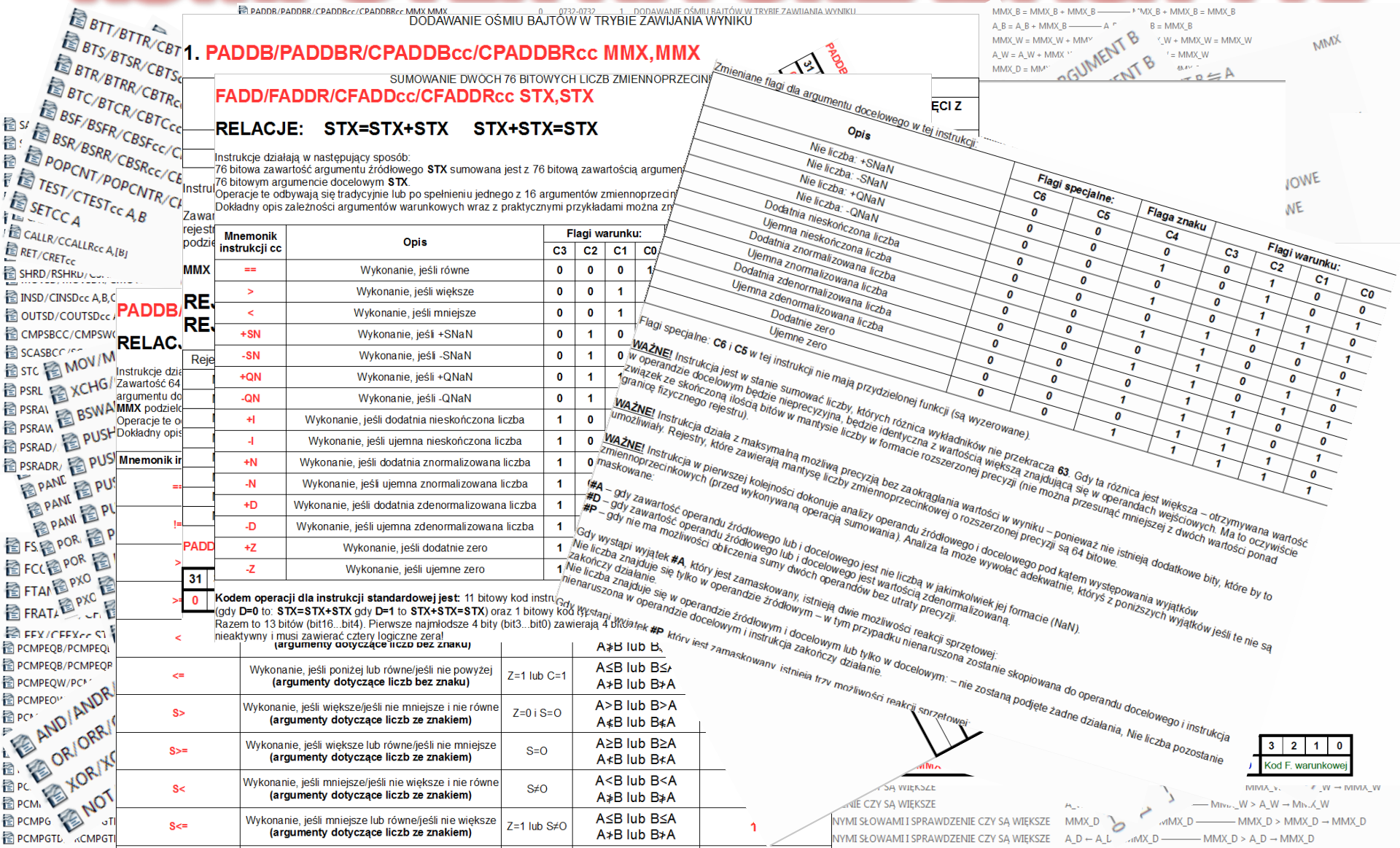
REJESTR MM6

BYTE2 BYTE1 BYTE0

MMX_B = MMX_B + MMX_B ——— MMX_B + MMX_B = MMX_B
A_B = A_B + MMX_B ——— A_B = MMX_B
MMX_W = MMX_W + MMX ——— MMX_W + MMX_W = MMX_W
A_W = A_W + MMX ——— MMX_W
MMX_D = MMX ——— MMX_W > A_W → MMX_W
MMX_D ——— MMX_D > MMX_D → MMX_D
A_D ← A_D ——— MMX_D > A_D → MMX_D

NOWY CEL

KOMPUTER NA UKŁADACH TTL



DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU

1. PADDB/PADDBR/CPADDBcc/CPADDBRcc MMX,MMX

SUMOWANIE DWÓCH 76 BITOWYCH LICZB ZMIENNOPRZECIN

FADD/FADDR/CFADDcc/CFADDRcc STX,STX

RELACJE: STX=STX+STX STX+STX=STX

Instrukcje działają w następujący sposób:
76 bitowa zawartość argumentu źródłowego **STX** sumowana jest z 76 bitową zawartością argumen
76 bitowym argumencie docelowym **STX**.
Operacje te odbywają się tradycyjnie lub po spełnieniu jednego z 16 argumentów zmiennoprzecin
Dokładny opis zależności argumentów warunkowych wraz z praktycznymi przykładami można zn

Mnemonik instrukcji cc	Opis	Flagi warunku: C3	C2	C1	C0
==	Wykonanie, jeśli równe	0	0	0	1
>	Wykonanie, jeśli większe	0	0	1	
<	Wykonanie, jeśli mniejsze	0	0	1	
+SN	Wykonanie, jeśli +SNaN	0	1	0	
-SN	Wykonanie, jeśli -SNaN	0	1	0	
+QN	Wykonanie, jeśli +QNaN	0	1		
-QN	Wykonanie, jeśli -QNaN	0	1		
+I	Wykonanie, jeśli dodatnia nieskończona liczba	1	0		
-I	Wykonanie, jeśli ujemna nieskończona liczba	1	0		
+N	Wykonanie, jeśli dodatnia znormalizowana liczba	1	0		
-N	Wykonanie, jeśli ujemna znormalizowana liczba	1			
+D	Wykonanie, jeśli dodatnia zdenormalizowana liczba	1			
-D	Wykonanie, jeśli ujemna zdenormalizowana liczba	1			
+Z	Wykonanie, jeśli dodatnie zero	1			
-Z	Wykonanie, jeśli ujemne zero	1			

Kodem operacji dla instrukcji standardowej jest: 11 bitowy kod instru
(gdy **D=0** to: **STX=STX+STX** gdy **D=1** to **STX+STX=STX**) oraz 1 bitowy kod
Razem to 13 bitów (bit16...bit4). Pierwsze najmłodsze 4 bity (bit3...bit0) zawierają 4 bitowe
nieaktywny i musi zawierać cztery logiczne zera!

Mnemonik instrukcji cc	Opis	Flagi warunku: C3	C2	C1	C0
<=	Wykonanie, jeśli poniżej lub równe/jeśli nie powyżej (argumenty dotyczące liczb bez znaku)	Z=1 lub C=1			A≤B lub B≤A lub A≯B lub B≯A
S>	Wykonanie, jeśli większe/jeśli nie mniejsze i nie równe (argumenty dotyczące liczb ze znakiem)	Z=0 i S=O			A>B lub B>A lub A≤B lub B≤A
S>=	Wykonanie, jeśli większe lub równe/jeśli nie mniejsze (argumenty dotyczące liczb ze znakiem)	S=O			A≥B lub B≥A lub A<B lub B<A
S<	Wykonanie, jeśli mniejsze/jeśli nie większe i nie równe (argumenty dotyczące liczb ze znakiem)	S≠O			A<B lub B<A lub A≥B lub B≥A
S<=	Wykonanie, jeśli mniejsze lub równe/jeśli nie większe (argumenty dotyczące liczb ze znakiem)	Z=1 lub S≠O			A≤B lub B≤A lub A≥B lub B≥A

Zmieniane flagi dla argumentu docelowego w tej instrukcji

Opis	Flagi specjalne: C6	C5	Flaga znaku C4	Flagi warunku: C3	C2	C1	C0
Nie liczba: +SNaN	0	0	0	0	1	0	0
Nie liczba: -SNaN	0	0	1	0	1	0	1
Nie liczba: +QNaN	0	0	0	0	1	1	0
Nie liczba: -QNaN	0	0	1	0	1	1	1
Dodatnia nieskończona liczba	0	0	0	1	0	0	0
Ujemna nieskończona liczba	0	0	1	1	0	0	1
Dodatnia znormalizowana liczba	0	0	0	1	0	1	0
Ujemna znormalizowana liczba	0	0	1	1	0	1	1
Dodatnia zdenormalizowana liczba	0	0	0	1	1	0	0
Ujemna zdenormalizowana liczba	0	0	1	1	1	0	1
Dodatnie zero	0	0	0	1	1	1	0
Ujemne zero	0	0	1	1	1	1	1

Flagi specjalne: **C6** i **C5** w tej instrukcji nie mają przydzielonej funkcji (są wyzerowane)

WAŻNE! Instrukcja jest w stanie sumować liczby, których różnica wykładników nie przekracza **63**. Gdy ta różnica jest większa – otrzymywana wartość w operandzie docelowym będzie nieprecyzyjna, będzie identyczna z wartością większą znajdującą się w operandach wejściowych. Ma to oczywiście związek ze skończoną ilością bitów w mantysie liczby w formacie rozszerzonej precyzji (nie można przesunąć mniejszej z dwóch wartości ponad granicę fizycznego rejestru).

WAŻNE! Instrukcja działa z maksymalną możliwą precyzją bez zaokrąglania wartości w wyniku – ponieważ nie istnieją dodatkowe bity, które by to umożliwiały. Rejestry, które zawierają mantysę liczby zmiennoprzecinkowej o rozszerzonej precyzji są 64 bitowe.

WAŻNE! Instrukcja w pierwszej kolejności dokonuje analizy operandu źródłowego i docelowego pod kątem występowania wyjątków zmiennoprzecinkowych (przed wykonywaną operacją sumowania). Analiza ta może wywołać adekwatnie, któryś z poniższych wyjątków jeśli te nie są maskowane:

#A – gdy zawartość operandu źródłowego lub i docelowego jest nie liczbą w jakimkolwiek jej formacie (NaN).
#D – gdy zawartość operandu źródłowego lub i docelowego jest wartością zdenormalizowaną.
#P – gdy nie ma możliwości obliczenia sumy dwóch operandów bez utraty precyzji.

Gdy wystąpi wyjątek **#A**, który jest zamaskowany, istnieją dwie możliwości reakcji sprzętowej:
Nie liczba znajduje się tylko w operandzie źródłowym – w tym przypadku nienaruszona zostanie skopiowana do operandu docelowego i instrukcja zakończy działanie.
Nie liczba znajduje się w operandzie źródłowym i docelowym lub tylko w docelowym: – nie zostaną podjęte żadne działania, Nie liczba pozostanie nienaruszona w operandzie docelowym i instrukcja zakończy działanie.

Gdy wystąpi wyjątek **#P** który jest zamaskowany, istnieją trzy możliwości reakcji sprzętowej:

3	2	1	0
Kod F. warunkowej			

NOWY CEL

KOMPUTER NA UKŁADACH TTL

DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU

Gdy wystąpi wyjątek **#P**, który jest zamaskowany, istnieją trzy możliwości reakcji sprzętowej:
W operandach źródłowym i docelowym znajdują się liczby znormalizowane lub w którymś z operandów znajduje się liczba zdenormalizowana i znormalizowana. Należy również przyjąć, iż we wszystkich przypadkach różnica ich wykładników przekracza 63 – w operandzie docelowym zawsze znajdzie się liczba większa.
W operandach źródłowym i docelowym znajdują się nieskończoności o tym samym znaku lub w którymś z operandów znajduje się dodatnia/ujemna liczba zdenormalizowana/znormalizowana albo dodatnie/ujemne zero i dodatnia/ujemna nieskończoność – w operandzie docelowym zawsze znajdzie się nieskończoność o znaku takim samym jak w operandzie wejściowym.
W operandach źródłowym i docelowym znajdują się nieskończoności o różnym znaku – w operandzie docelowym zawsze zastaniemy Nie liczbę **QNaN**. Zawsze w następującym formacie: **1 11111111111 1100000000000000000000000000000000000000000000000000000000000000**.

W dalszej kolejności dokonywana jest operacja sumowania dwóch operandów w formacie rozszerzonej precyzji. Następnie dokonywana jest analiza operandu docelowego pod kątem występowania wyjątków zmiennoprzecinkowych. Analiza ta może wywołać adekwatnie, któryś z poniższych wyjątków jeśli te nie są maskowane:

#O – gdy wystąpiło przepełnienie liczbowe.
#U – gdy wystąpił niedomiar liczbowy.

Gdy wystąpi wyjątek **#O**, który jest zamaskowany – wartością w operandzie docelowym będzie dodatnia lub ujemna nieskończoność.
Gdy wystąpi wyjątek **#U**, który jest zamaskowany – wartością w operandzie docelowym będzie ujemna lub dodatnia liczba zdenormalizowana.

Poniższa tabela przedstawia wszystkie możliwe relacje między wartościami w operandzie źródłowym i docelowym na których wykonywane jest dodawanie zmiennoprzecinkowe. Aby wszystkie operacje odbywały się całkowicie swobodnie nie może być żadnych wyjątków lub muszą być zamaskowane odpowiednie wyjątki zmiennoprzecinkowe.

	OPERAND DOCELOWY:									
OPERAND ŹRÓDŁOWY:	Z+D=D	−∞	−F	−D	−0	+0	+D	+F	+∞	NaN
	−∞	−∞	−∞	−∞	−∞	−∞	−∞	−∞	QNaN	NaN
	−F	−∞	−F lub −∞	−F	−F	−F	−D lub −F	±F lub +0	+∞	NaN
	−D	−∞	−D lub −F	−D lub −F	−D	−D	±D lub +0	+D lub +F	+∞	NaN
	−0	−∞	−F	−D	−0	+0	+D	+F	+∞	NaN
	+0	−∞	−F	−D	+0	+0	+D	+F	+∞	NaN
	+D	−∞	−D lub −F	±D lub +0	+D	+D	+D lub +F	+F	+∞	NaN
	+F	−∞	±F lub +0	+D lub +F	+F	+F	+F	+F lub +∞	+∞	NaN
	+∞	QNaN	+∞	+∞	+∞	+∞	+∞	+∞	+∞	NaN
	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

UWAGI:
±D – Zdenormalizowana liczba zmiennoprzecinkowa w formacie rozszerzonej precyzji.
±F – Skończona liczba zmiennoprzecinkowa w formacie rozszerzonej precyzji.

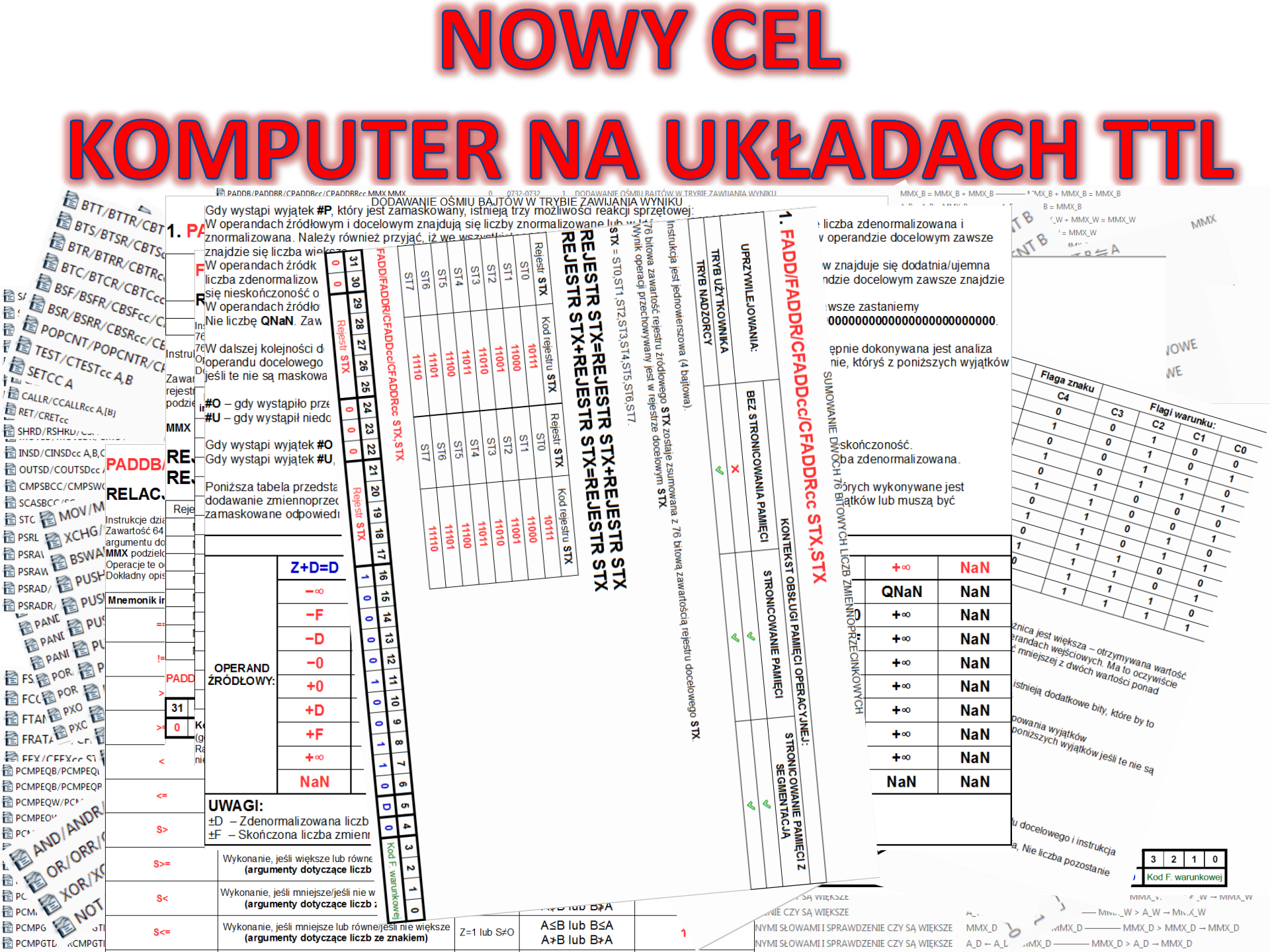
S>=	Wykonanie, jeśli większe lub równe/jeśli nie mniejsze **(argumenty dotyczące liczb ze znakiem)**	S=O	A≥B lub B≥A A≮B lub B≠A
S<	Wykonanie, jeśli mniejsze/jeśli nie większe i nie równe **(argumenty dotyczące liczb ze znakiem)**	S≠O	A<B lub B<A A≯B lub B≯A
S<=	Wykonanie, jeśli mniejsze lub równe/jeśli nie większe **(argumenty dotyczące liczb ze znakiem)**	Z=1 lub S≠O	A≤B lub B≤A A≯B lub B≯A

Flaga znaku	Flagi warunku:				
C4	C3	C2	C1	C0
0	0	1	0	0
1	0	1	0	1
0	0	1	1	0
1	0	1	1	1
0	1	0	0	0
1	1	0	0	1
0	1	0	1	0
1	1	0	1	1
0	1	1	0	0
1	1	1	0	1
0	1	1	1	0
1	1	1	1	1

3	2	1	0
Kod F. warunkowej			

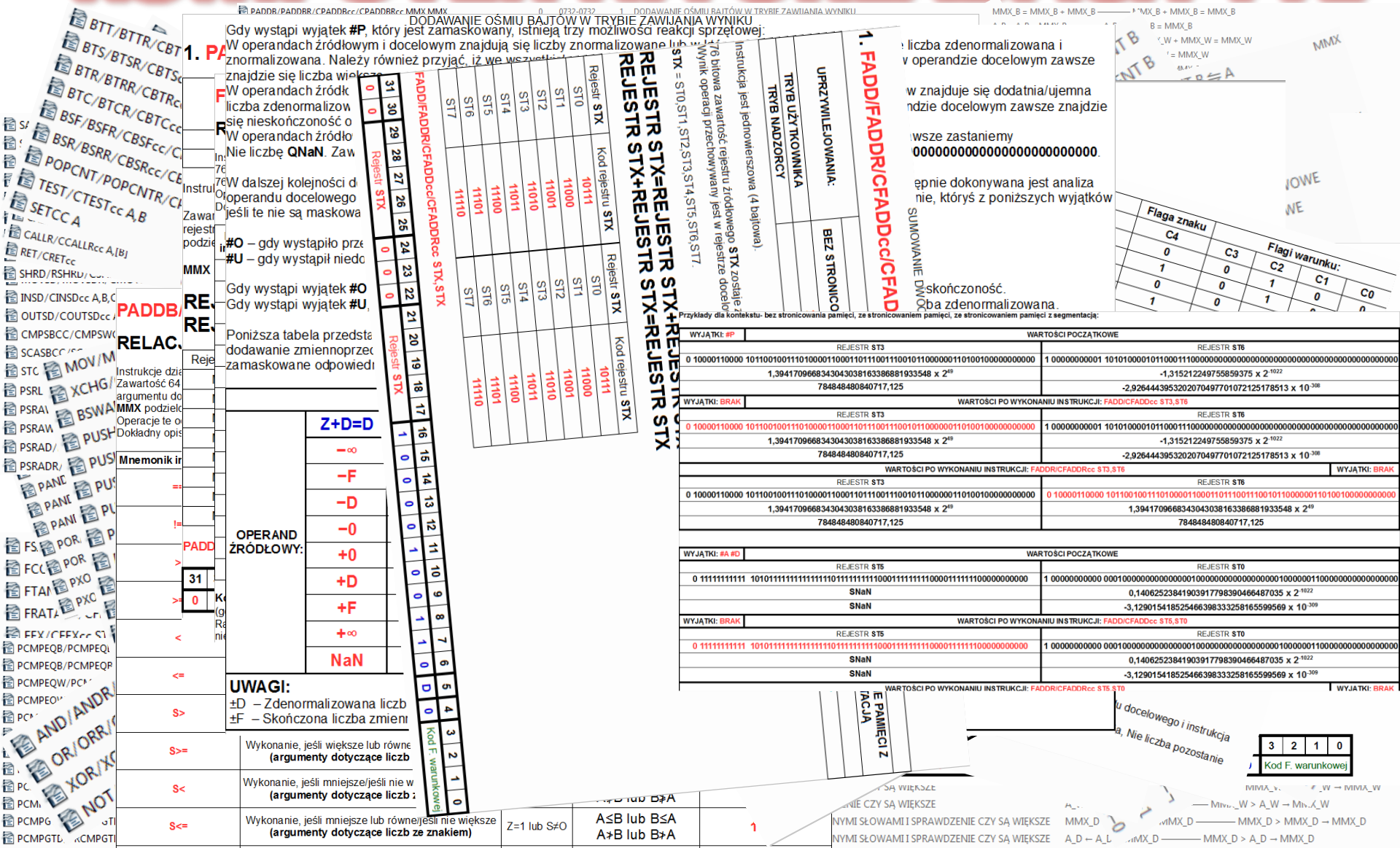
NOWY CEL

KOMPUTER NA UKŁADACH TTL



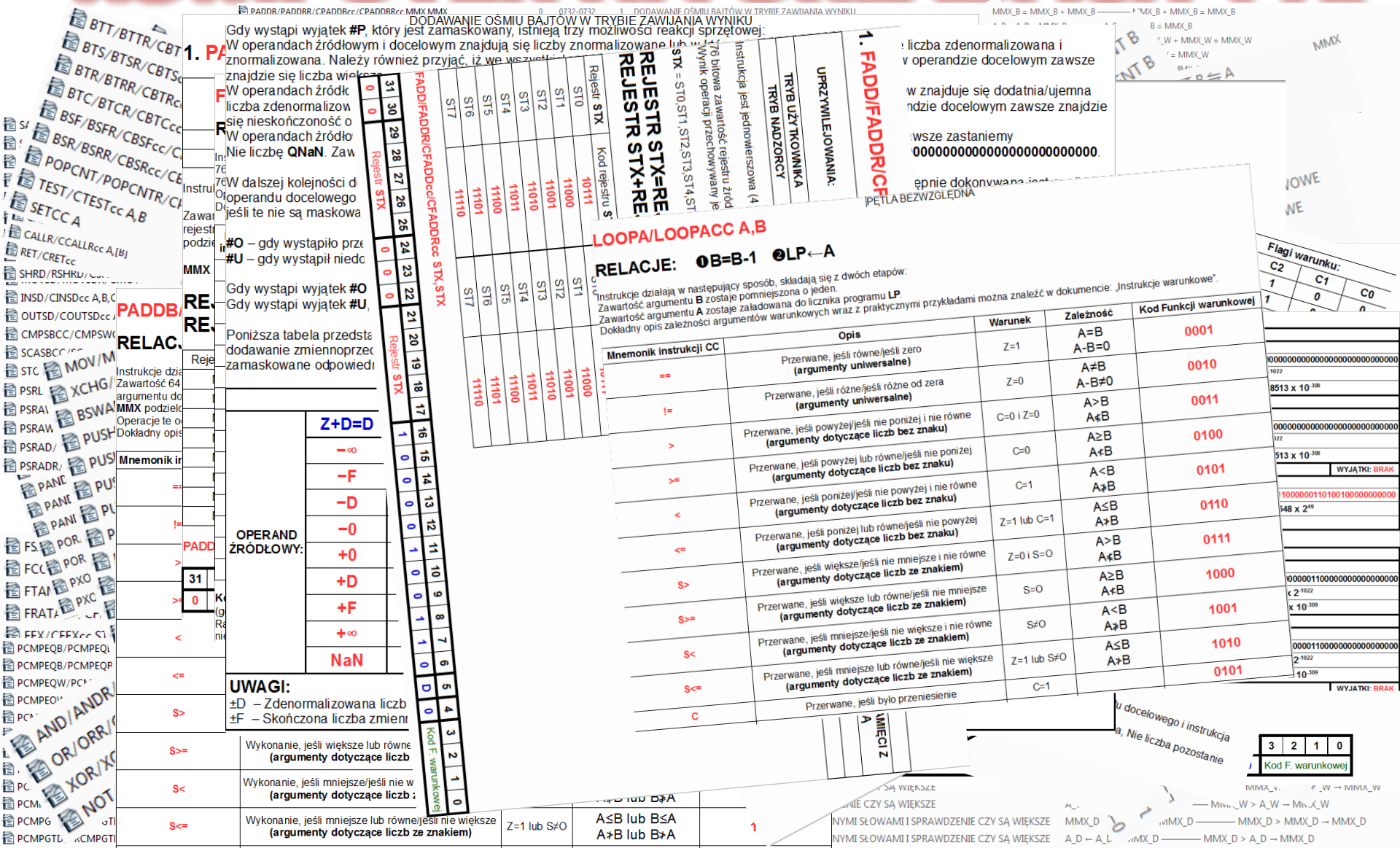
NOWY CEL

KOMPUTER NA UKŁADACH TTL



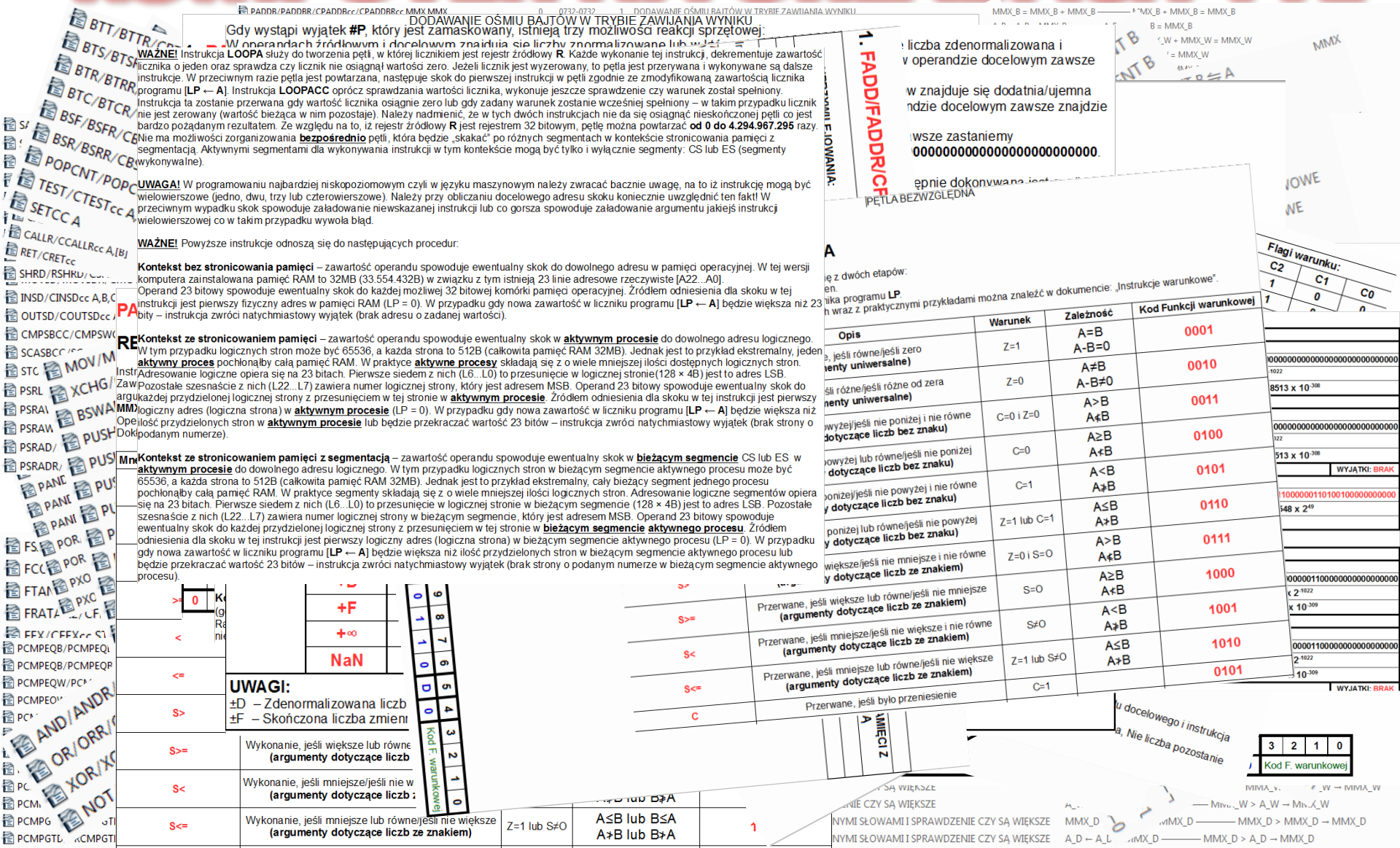
NOWY CEL

KOMPUTER NA UKŁADACH TTL



NOWY CEL

KOMPUTER NA UKŁADACH TTL



DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU

Gdy wystąpi wyjątek **#P**, który jest zamaskowany, istnieją trzy możliwości reakcji sprzętowej:

WAŻNE! Instrukcja **LOOPA** służy do tworzenia pętli, w której licznikiem jest rejestr źródłowy **R**. Każde wykonanie tej instrukcji, dekrementuje zawartość licznika o jeden oraz sprawdza czy licznik nie osiągnął wartości zero. Jeżeli licznik jest wyzerowany, to pętla jest przerywana i wykonywane są dalsze instrukcje. W przeciwnym razie pętla jest powtarzana, następuje skok do pierwszej instrukcji w pętli zgodnie ze zmodyfikowaną zawartością licznika programu [**LP ← A**]. Instrukcja **LOOPACC** oprócz sprawdzania wartości licznika, wykonuje jeszcze sprawdzenie czy warunek został spełniony. Instrukcja ta zostanie przerwana gdy wartość licznika osiągnie zero lub gdy zadany warunek zostanie wcześniej spełniony – w takim przypadku licznik nie jest zerowany (wartość bieżąca w nim pozostaje). Należy nadmienić, że w tych dwóch instrukcjach nie da się osiągnąć nieskończonej pętli co jest bardzo pożądanym rezultatem. Ze względu na to, iż rejestr źródłowy **R** jest rejestrem 32 bitowym, pętlę można powtarzać **od 0 do 4.294.967.295** razy. Nie ma możliwości zorganizowania **bezpośrednio** pętli, która będzie „skakać" po różnych segmentach w kontekście stronicowania pamięci z segmentacją. Aktywnymi segmentami dla wykonywania instrukcji w tym kontekście mogą być tylko i wyłącznie segmenty: CS lub ES (segmenty wykonywalne).

UWAGA! W programowaniu najbardziej niskopoziomowym czyli w języku maszynowym należy zwracać bacznie uwagę, na to iż instrukcję mogą być wielowierszowe (jedno, dwu, trzy lub czterowierszowe). Należy przy obliczaniu docelowego adresu skoku koniecznie uwzględnić ten fakt! W przeciwnym wypadku skok spowoduje załadowanie nieswkazanej instrukcji lub co gorsza spowoduje załadowanie argumentu jakiejś instrukcji wielowierszowej co w takim przypadku wywoła błąd.

WAŻNE! Powyższe instrukcje odnoszą się do następujących procedur:

Kontekst bez stronicowania pamięci – zawartość operandu spowoduje ewentualny skok do dowolnego adresu w pamięci operacyjnej. W tej wersji komputera zainstalowana pamięć RAM to 32MB (33.554.432B) w związku z tym istnieją 23 linie adresowe rzeczywiste [A22...A0]. Operand 23 bitowy spowoduje ewentualny skok do każdej możliwej 32 bitowej komórki pamięci operacyjnej. Źródłem odniesienia dla skoku w tej instrukcji jest pierwszy fizyczny adres w pamięci RAM (LP = 0). W przypadku gdy nowa zawartość w liczniku programu [**LP ← A**] będzie większa niż 23 bity – instrukcja zwróci natychmiastowy wyjątek (brak adresu o zadanej wartości).

Kontekst ze stronicowaniem pamięci – zawartość operandu spowoduje ewentualny skok w **aktywnym procesie** do dowolnego adresu logicznego. W tym przypadku logicznych stron może być 65536, a każda strona to 512B (całkowita pamięć RAM 32MB). Jednak jest to przykład ekstremalny, jeden **aktywny proces** pochłonąłby całą pamięć RAM. W praktyce **aktywne procesy** składają się z o wiele mniejszej ilości dostępnych logicznych stron. Adresowanie logiczne opiera się na 23 bitach. Pierwsze siedem z nich (L6...L0) to przesunięcie w logicznej stronie(128 × 4B) jest to adres LSB. Pozostałe szesnaście z nich (L22...L7) zawiera numer logicznej strony, który jest adresem MSB. Operand 23 bitowy spowoduje ewentualny skok do każdej przydzielonej logicznej strony z przesunięciem w tej stronie w **aktywnym procesie**. Źródłem odniesienia dla skoku w tej instrukcji jest pierwszy logiczny adres (logiczna strona) w **aktywnym procesie** (LP = 0). W przypadku gdy nowa zawartość w liczniku programu [**LP ← A**] będzie większa niż ilość przydzielonych stron w **aktywnym procesie** lub będzie przekraczać wartość 23 bitów – instrukcja zwróci natychmiastowy wyjątek (brak strony o podanym numerze).

Kontekst ze stronicowaniem pamięci z segmentacją – zawartość operandu spowoduje ewentualny skok w **bieżącym segmencie** CS lub ES w **aktywnym procesie** do dowolnego adresu logicznego. W tym przypadku logicznych stron w bieżącym segmencie aktywnego procesu może być 65536, a każda strona to 512B (całkowita pamięć RAM 32MB). Jednak jest to przykład ekstremalny, cały bieżący segment jednego procesu pochłonąłby całą pamięć RAM. W praktyce segmenty składają się z o wiele mniejszej ilości logicznych stron. Adresowanie logiczne segmentów opiera się na 23 bitach. Pierwsze siedem z nich (L6...L0) to przesunięcie w logicznej stronie w bieżącym segmencie (128 × 4B) jest to adres LSB. Pozostałe szesnaście z nich (L22...L7) zawiera numer logicznej strony w bieżącym segmencie, który jest adresem MSB. Operand 23 bitowy spowoduje ewentualny skok do każdej przydzielonej logicznej strony z przesunięciem w tej stronie w **bieżącym segmencie aktywnego procesu**. Źródłem odniesienia dla skoku w tej instrukcji jest pierwszy logiczny adres (logiczna strona) w bieżącym segmencie aktywnego procesu (LP = 0). W przypadku gdy nowa zawartość w liczniku programu [**LP ← A**] będzie większa niż ilość przydzielonych stron w bieżącym segmencie aktywnego procesu lub będzie przekraczać wartość 23 bitów – instrukcja zwróci natychmiastowy wyjątek (brak strony o podanym numerze w bieżącym segmencie aktywnego procesu).

UWAGI:
±D – Zdenormalizowana liczb
±F – Skończona liczba zmien

S>=	Wykonanie, jeśli większe lub równe **(argumenty dotyczące liczb**
S<	Wykonanie, jeśli mniejsze/jeśli nie w **(argumenty dotyczące liczb**
S<=	Wykonanie, jeśli mniejsze lub równe/jeśli nie większe **(argumenty dotyczące liczb ze znakiem)**

Opis	Warunek	Zależność	Kod Funkcji warunkowej
Przerwane, jeśli równe/jeśli zero **(argumenty uniwersalne)**	Z=1	A=B A-B=0	0001
Przerwane, jeśli różne/jeśli różne od zera **(argumenty uniwersalne)**	Z=0	A≠B A-B≠0	0010
Przerwane, jeśli mniejsze/jeśli nie poniżej i nie równe **(argumenty dotyczące liczb bez znaku)**	C=0 i Z=0	A>B A⩽B	0011
Przerwane, jeśli większe lub równe/jeśli nie poniżej **(argumenty dotyczące liczb bez znaku)**	C=0	A≥B A<B	0100
Przerwane, jeśli poniżej/jeśli nie powyżej i nie równe **(argumenty dotyczące liczb bez znaku)**	C=1	A<B A⩾B	0101
Przerwane, jeśli poniżej lub równe/jeśli nie powyżej **(argumenty dotyczące liczb bez znaku)**	Z=1 lub C=1	A≤B A>B	0110
Przerwane, jeśli większe/jeśli nie mniejsze i nie równe **(argumenty dotyczące liczb ze znakiem)**	Z=0 i S=O	A>B A⩽B	0111
Przerwane, jeśli większe lub równe/jeśli nie mniejsze **(argumenty dotyczące liczb ze znakiem)**	S=O	A≥B A<B	1000
Przerwane, jeśli mniejsze/jeśli nie większe i nie równe **(argumenty dotyczące liczb ze znakiem)**	S≠O	A<B A⩾B	1001
Przerwane, jeśli mniejsze lub równe/jeśli nie większe **(argumenty dotyczące liczb ze znakiem)**	Z=1 lub S≠O	A≤B A>B	1010
Przerwane, jeśli było przeniesienie	C=1		0101

Flagi warunku:		
C2	C1	C0
1	0	0

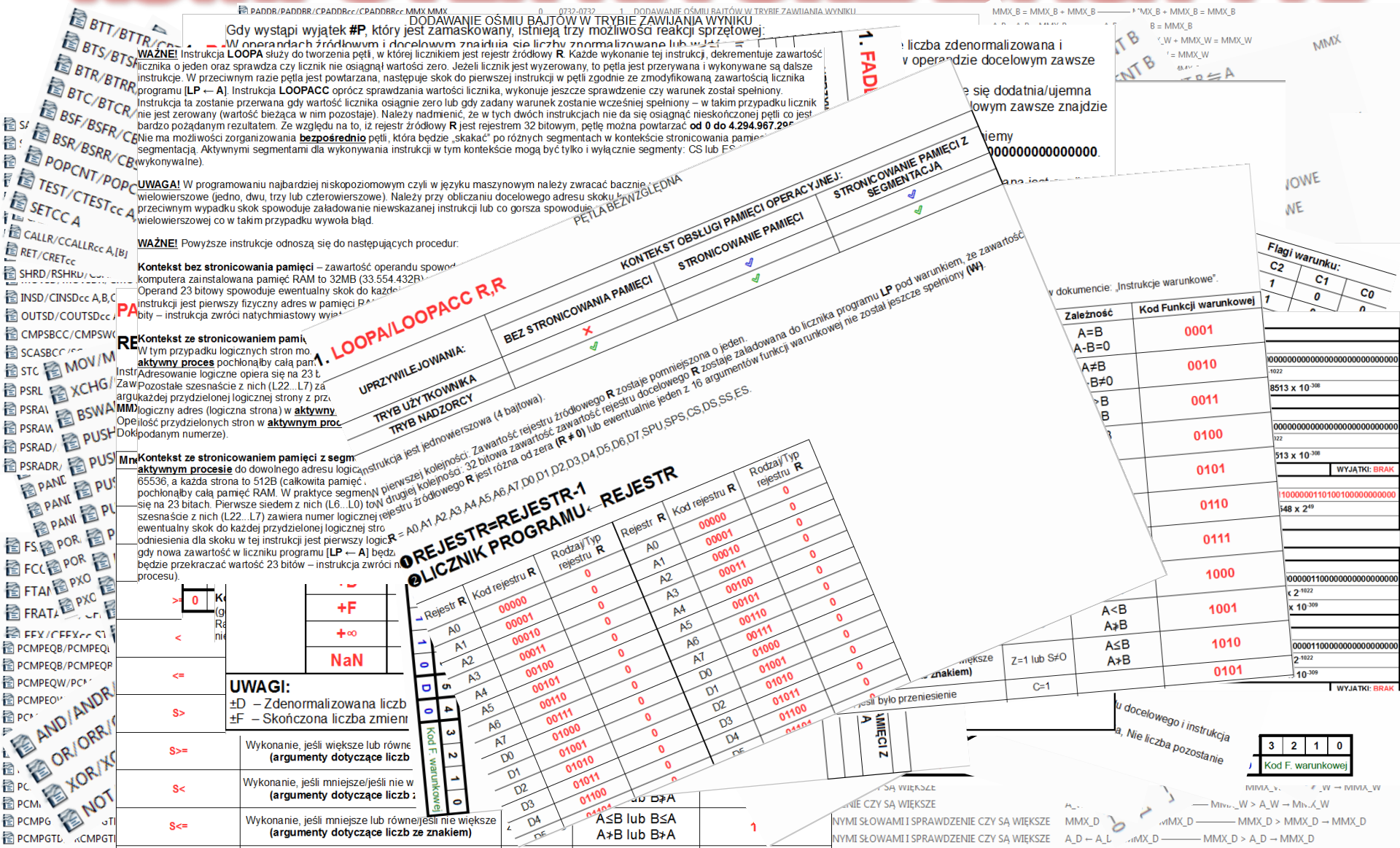
WYJĄTKI: BRAK

WYJĄTKI: BRAK

3	2	1	0
Kod F. warunkowej			

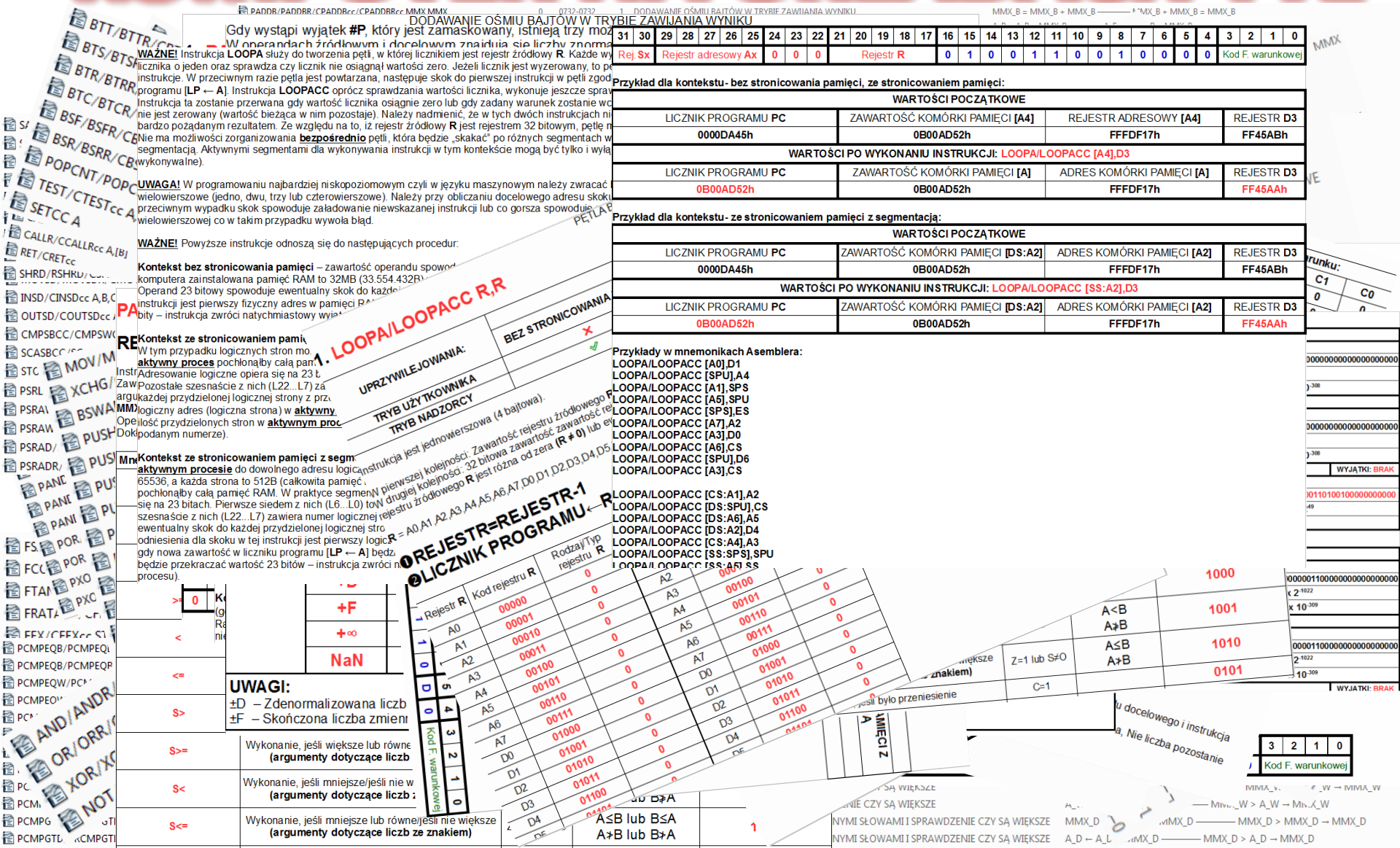
NOWY CEL

KOMPUTER NA UKŁADACH TTL



NOWY CEL

KOMPUTER NA UKŁADACH TTL



DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rej. Sx		Rejestr adresowy Ax					0	0	0	Rejestr R					0	1	0	0	1	1	0	0	1	0	0	0	0	Kod F. warunkowej			

Przykład dla kontekstu- bez stronicowania pamięci, ze stronicowaniem pamięci:

WARTOŚCI POCZĄTKOWE			
LICZNIK PROGRAMU **PC**	ZAWARTOŚĆ KOMÓRKI PAMIĘCI **[A4]**	REJESTR ADRESOWY **[A4]**	REJESTR **D3**
0000DA45h	**0B00AD52h**	**FFFDF17h**	**FF45ABh**
WARTOŚCI PO WYKONANIU INSTRUKCJI: LOOPA/LOOPACC [A4],D3			
LICZNIK PROGRAMU **PC**	ZAWARTOŚĆ KOMÓRKI PAMIĘCI **[A]**	ADRES KOMÓRKI PAMIĘCI **[A]**	REJESTR **D3**
0B00AD52h	**0B00AD52h**	**FFFDF17h**	**FF45AAh**

Przykład dla kontekstu- ze stronicowaniem pamięci z segmentacją:

WARTOŚCI POCZĄTKOWE			
LICZNIK PROGRAMU **PC**	ZAWARTOŚĆ KOMÓRKI PAMIĘCI **[DS:A2]**	ADRES KOMÓRKI PAMIĘCI **[A2]**	REJESTR **D3**
0000DA45h	**0B00AD52h**	**FFFDF17h**	**FF45ABh**
WARTOŚCI PO WYKONANIU INSTRUKCJI: LOOPA/LOOPACC [SS:A2],D3			
LICZNIK PROGRAMU **PC**	ZAWARTOŚĆ KOMÓRKI PAMIĘCI **[DS:A2]**	ADRES KOMÓRKI PAMIĘCI **[A2]**	REJESTR **D3**
0B00AD52h	**0B00AD52h**	**FFFDF17h**	**FF45AAh**

Przykłady w mnemonikach Asemblera:

```
LOOPA/LOOPACC [A0],D1
LOOPA/LOOPACC [SPU],A4
LOOPA/LOOPACC [A1],SPS
LOOPA/LOOPACC [A5],SPU
LOOPA/LOOPACC [SPS],ES
LOOPA/LOOPACC [A7],A2
LOOPA/LOOPACC [A3],D0
LOOPA/LOOPACC [A6],CS
LOOPA/LOOPACC [SPU],D6
LOOPA/LOOPACC [A3],CS

LOOPA/LOOPACC [CS:A1],A2
LOOPA/LOOPACC [DS:SPU],CS
LOOPA/LOOPACC [DS:A6],A5
LOOPA/LOOPACC [DS:A2],D4
LOOPA/LOOPACC [SS:A4],A3
LOOPA/LOOPACC [SS:SPS],SPU
```

UWAGI:

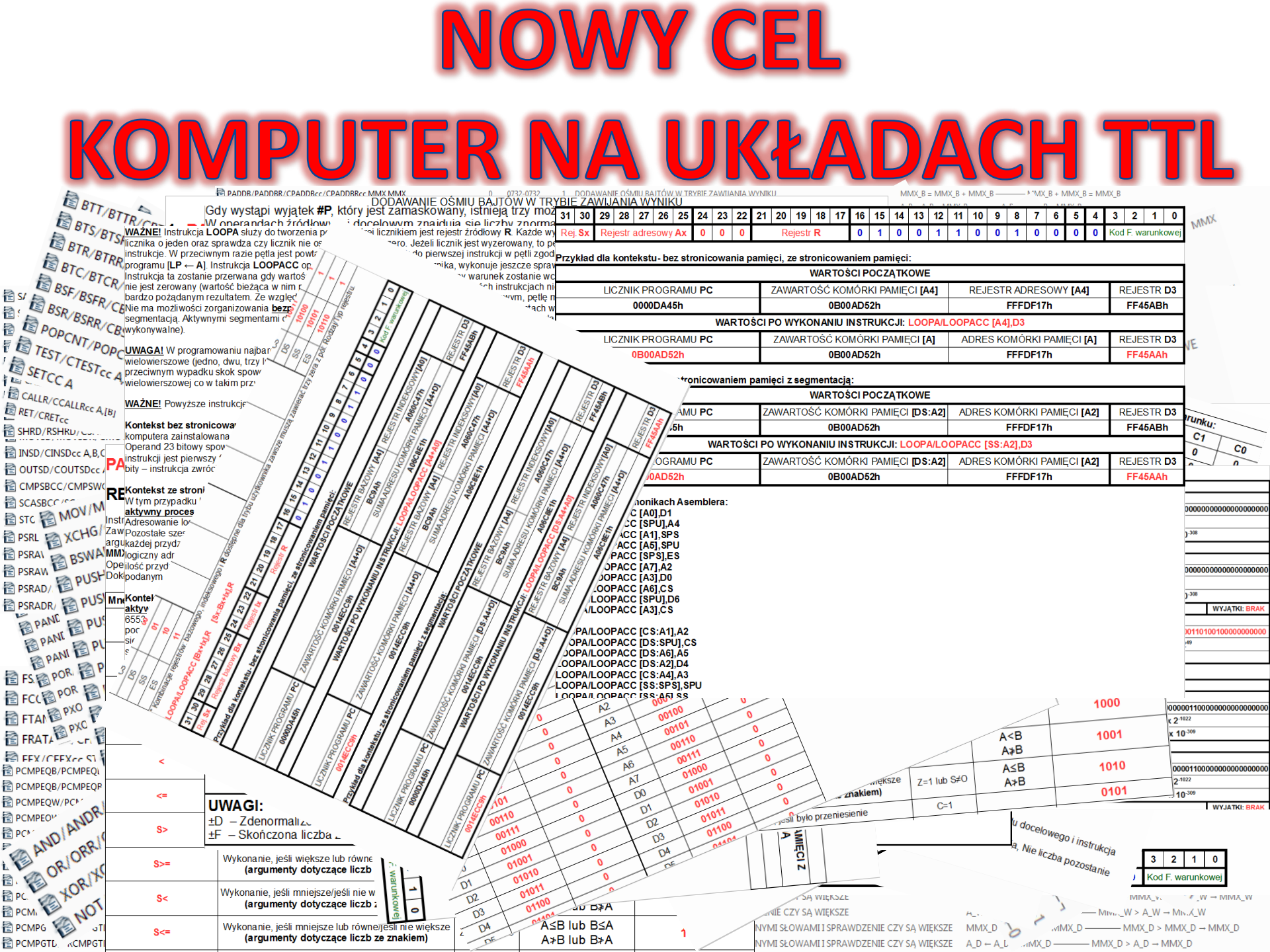
±D – Zdenormalizowana liczb
±F – Skończona liczba zmien

S>=	Wykonanie, jeśli większe lub równe **(argumenty dotyczące liczb**
S<	Wykonanie, jeśli mniejsze/jeśli nie w **(argumenty dotyczące liczb**
S<=	Wykonanie, jeśli mniejsze lub równe/jeśli nie większe **(argumenty dotyczące liczb ze znakiem)**

A≤B lub B≤A	
A≠B lub B≠A	1

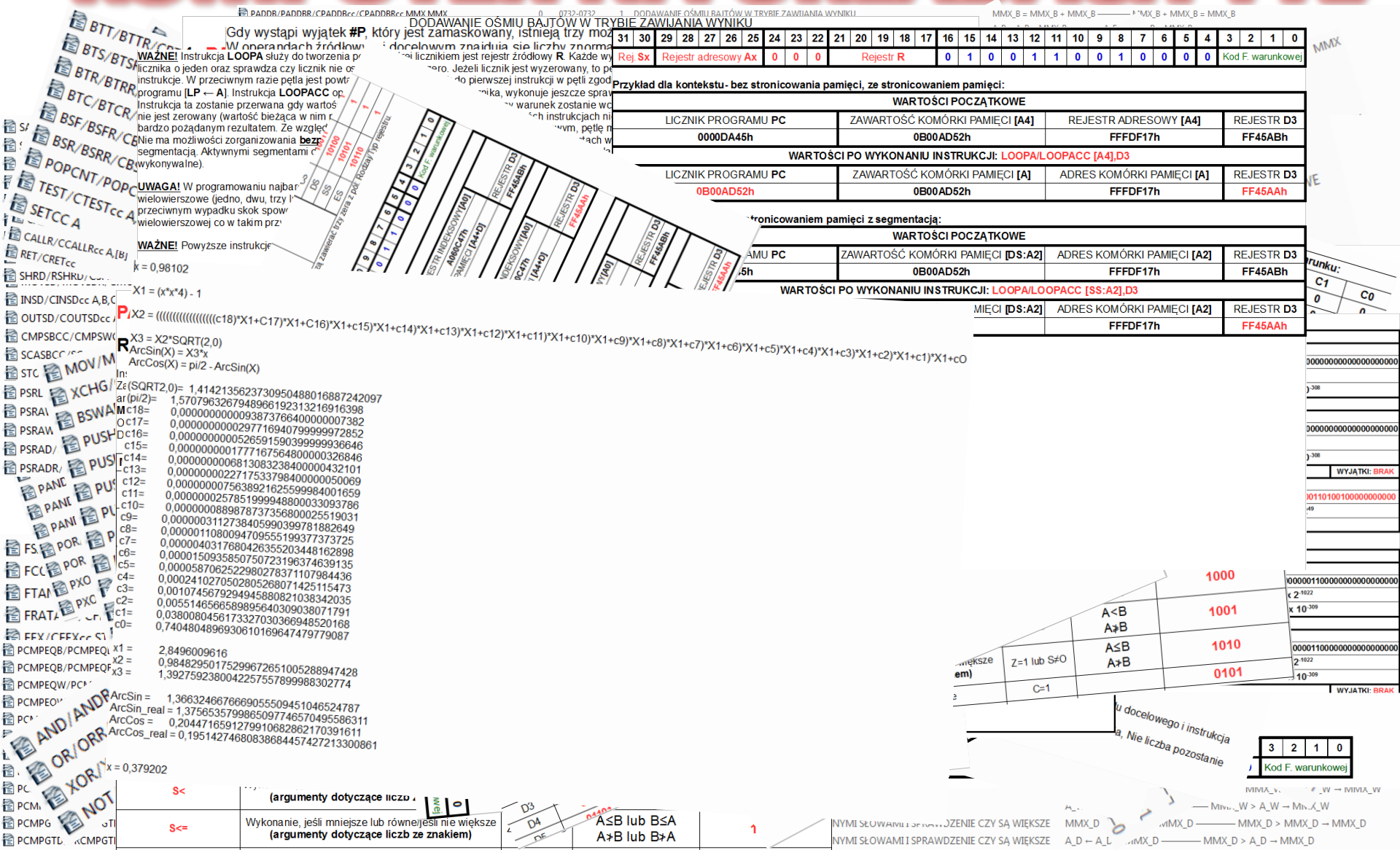
NOWY CEL

KOMPUTER NA UKŁADACH TTL



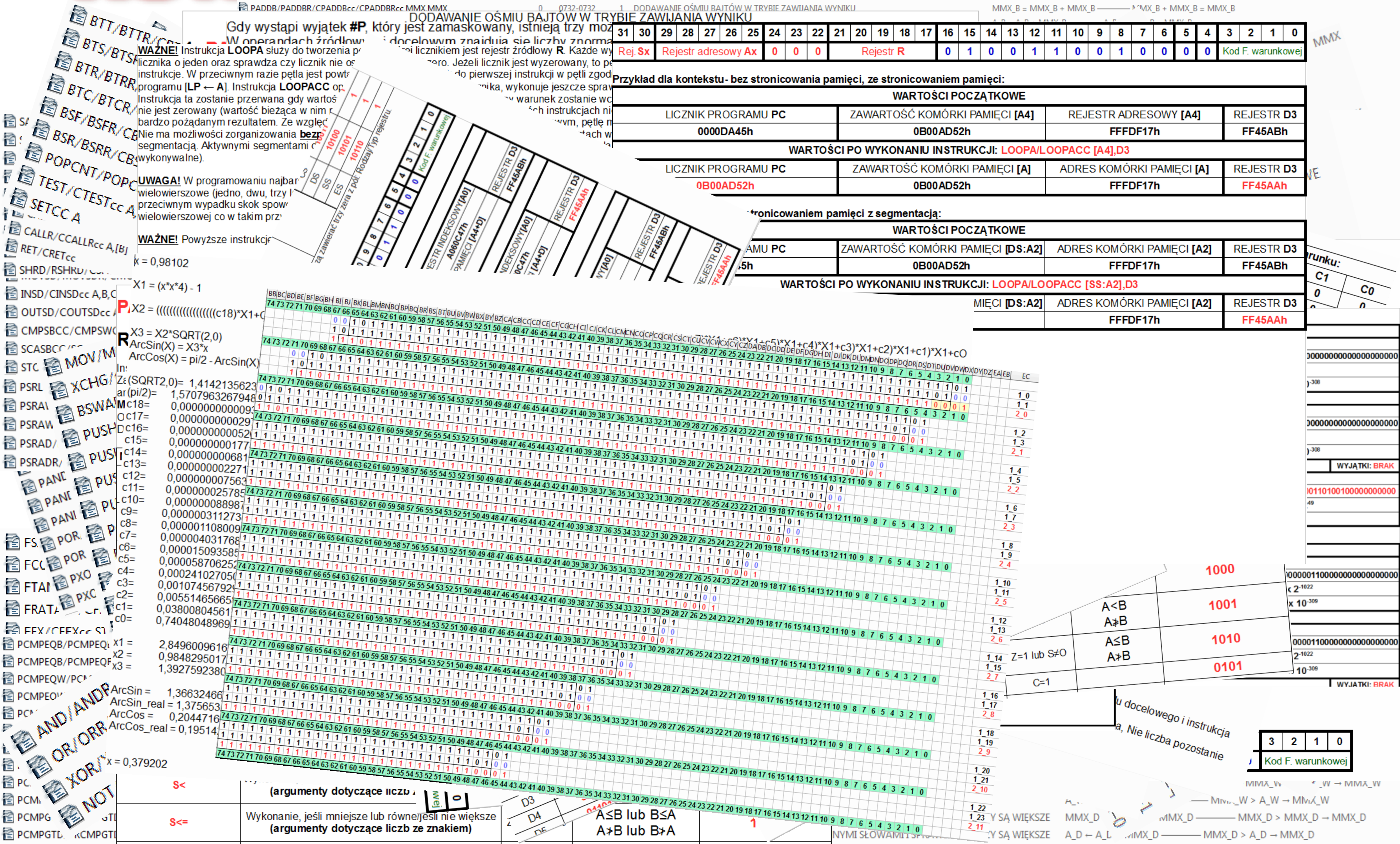
NOWY CEL

KOMPUTER NA UKŁADACH TTL



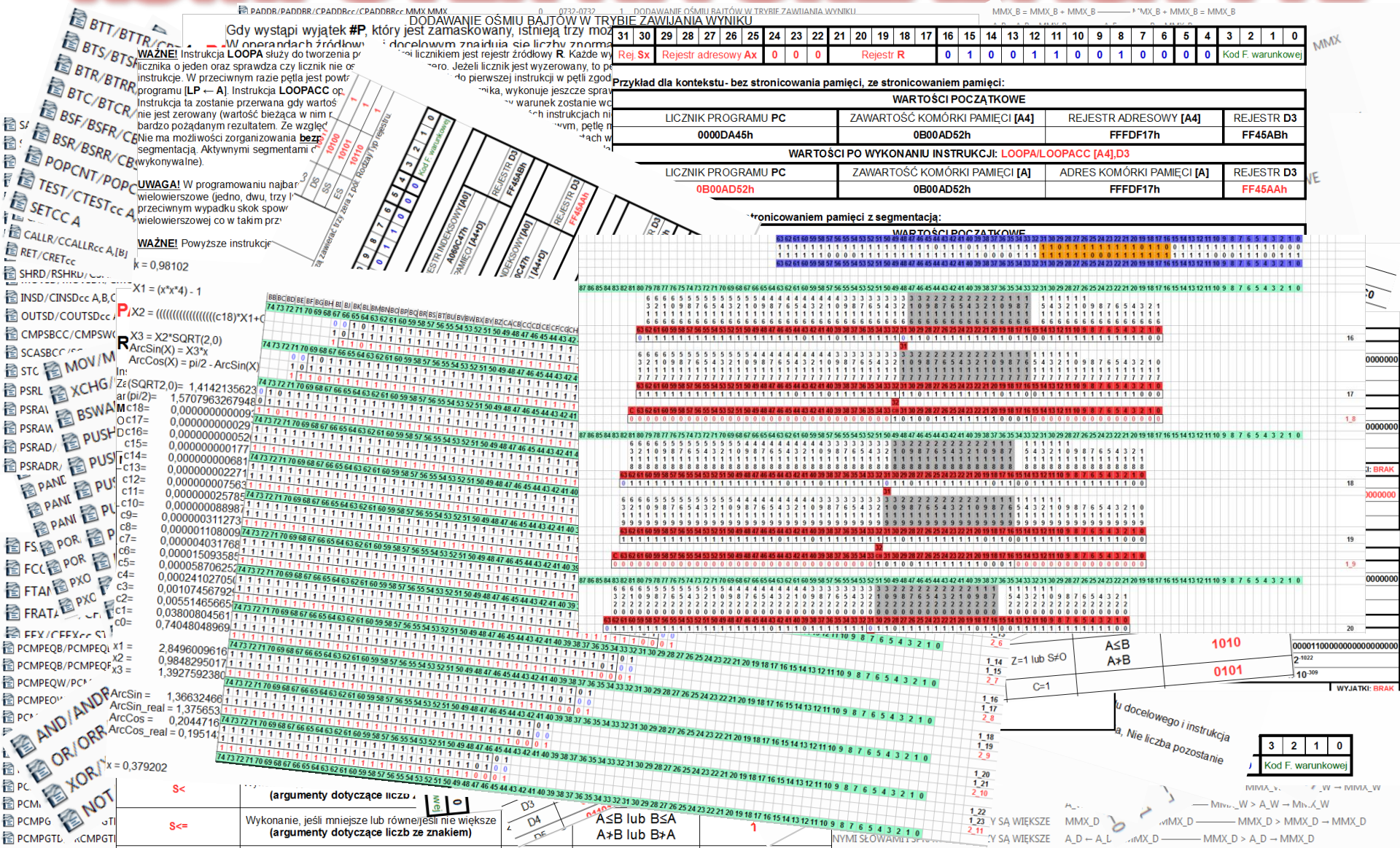
NOWY CEL

KOMPUTER NA UKŁADACH TTL



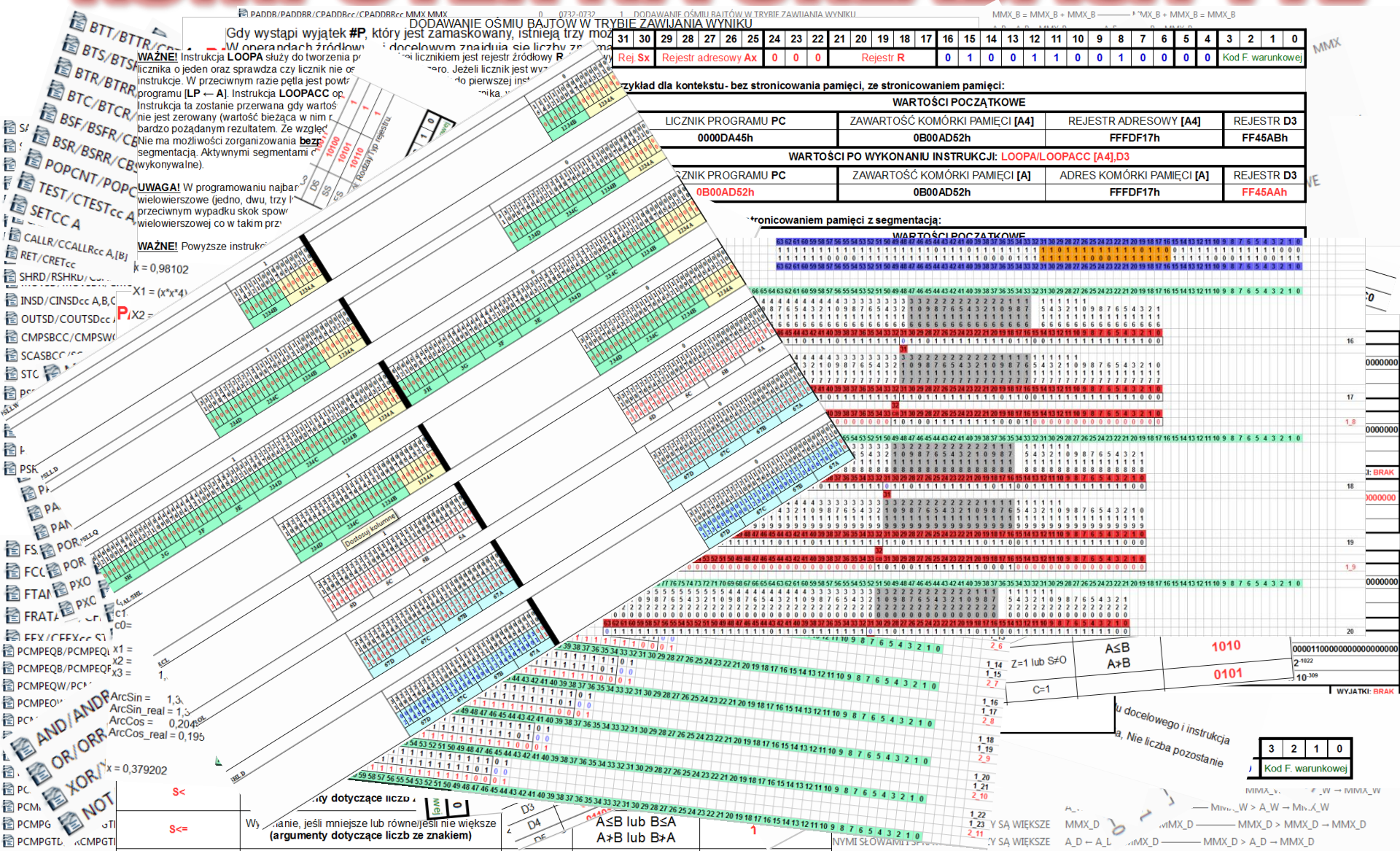
NOWY CEL

KOMPUTER NA UKŁADACH TTL



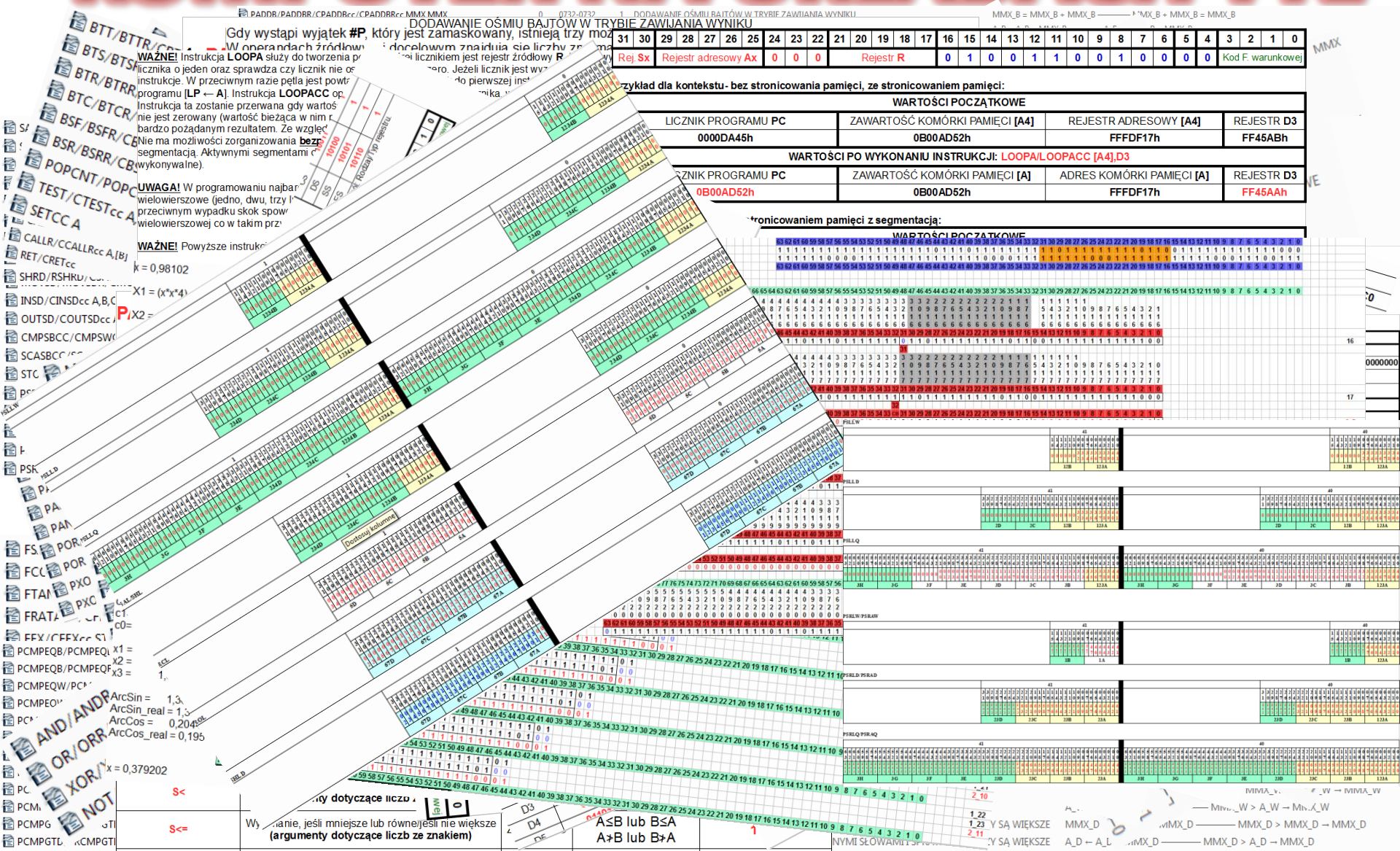
NOWY CEL

KOMPUTER NA UKŁADACH TTL



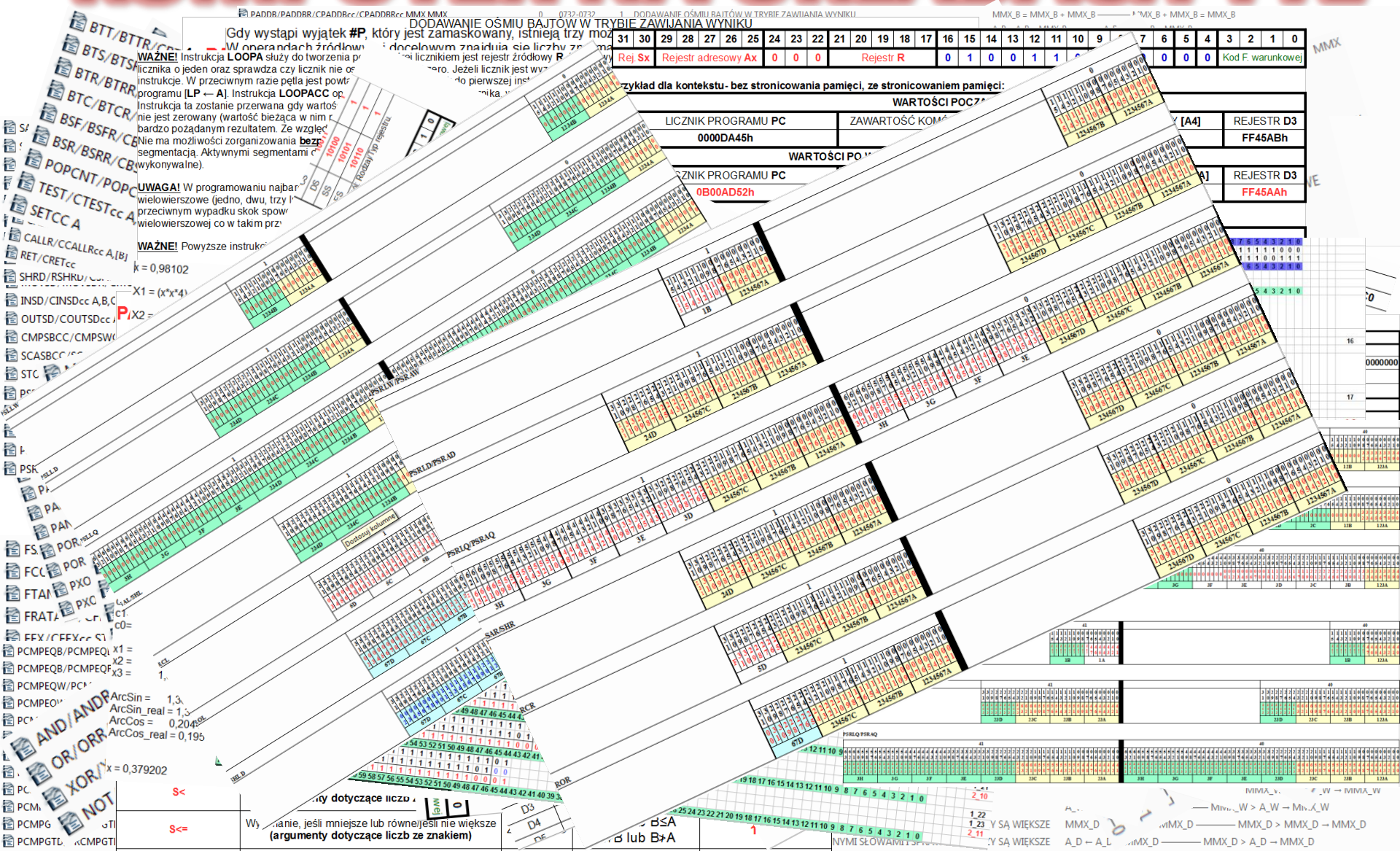
NOWY CEL

KOMPUTER NA UKŁADACH TTL



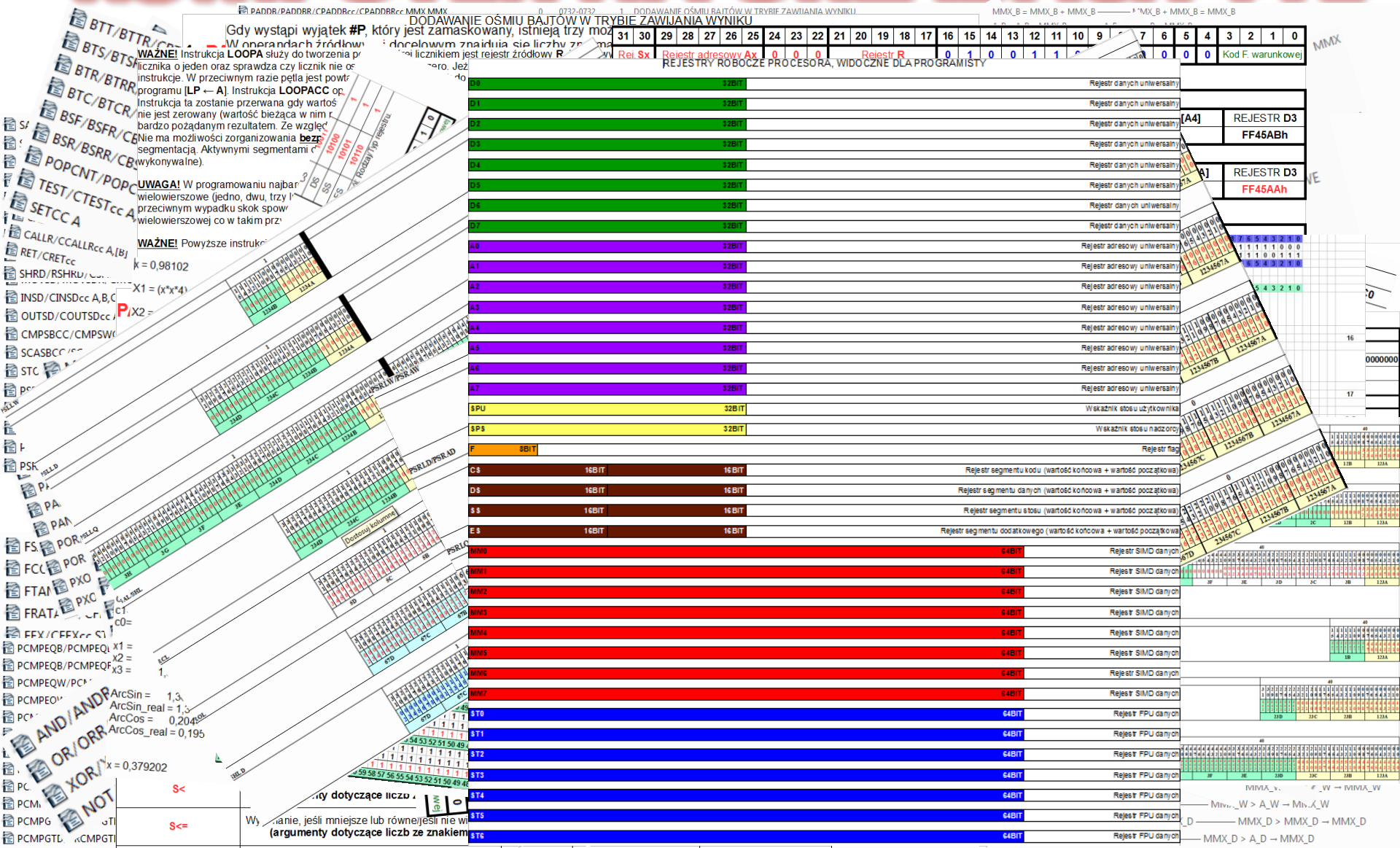
NOWY CEL

KOMPUTER NA UKŁADACH TTL



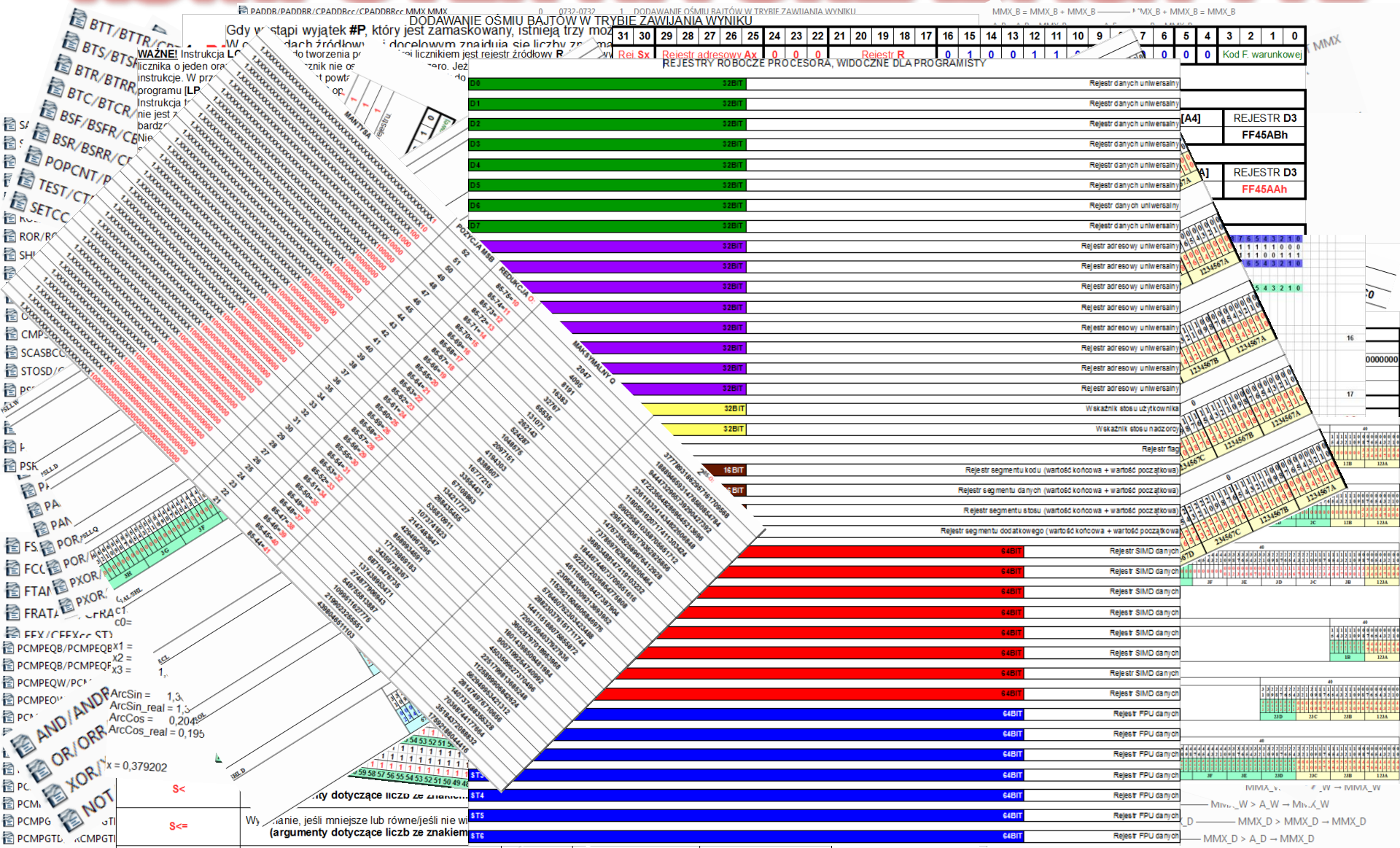
NOWY CEL

KOMPUTER NA UKŁADACH TTL



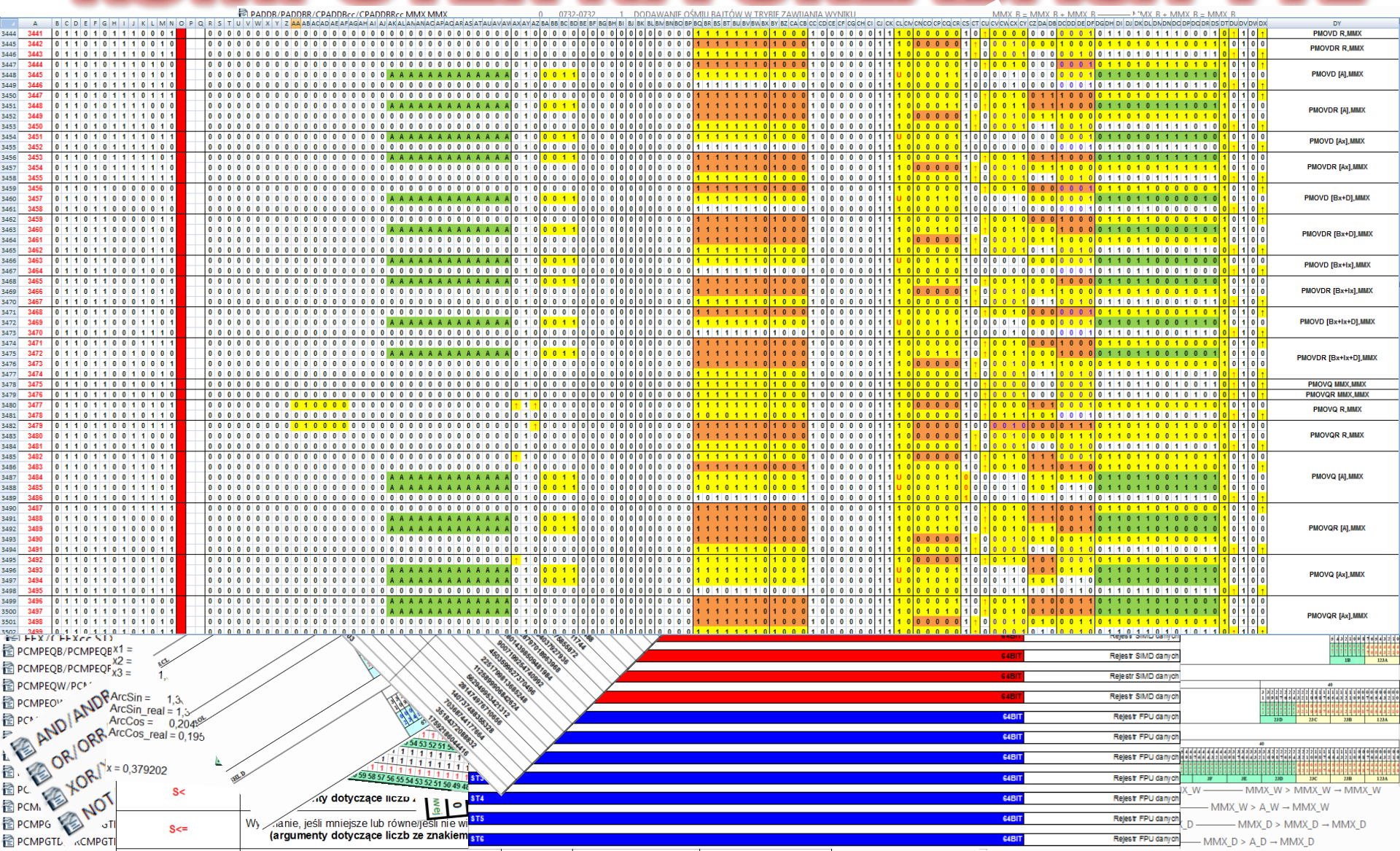
NOWY CEL

KOMPUTER NA UKŁADACH TTL



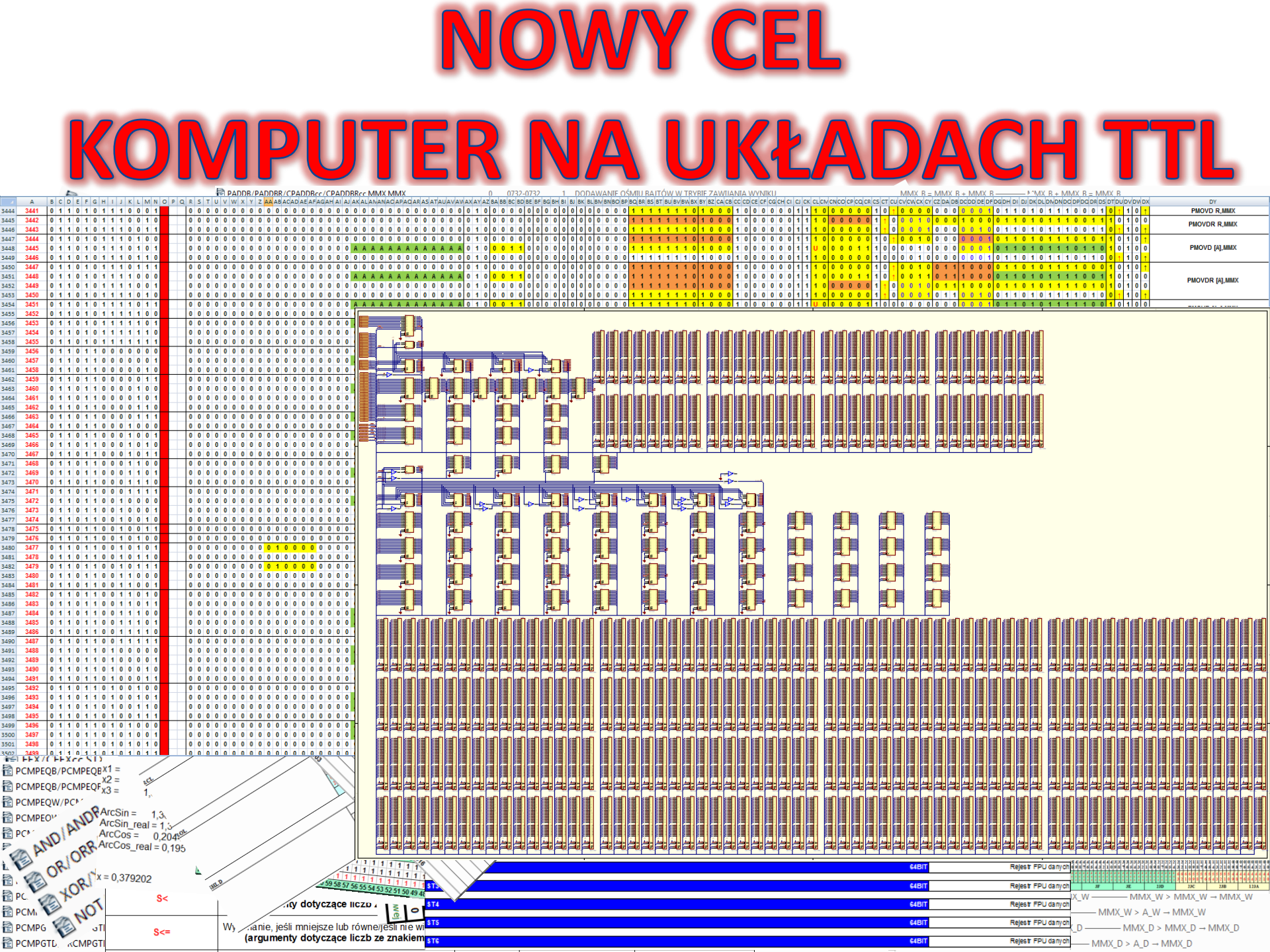
NOWY CEL

KOMPUTER NA UKŁADACH TTL



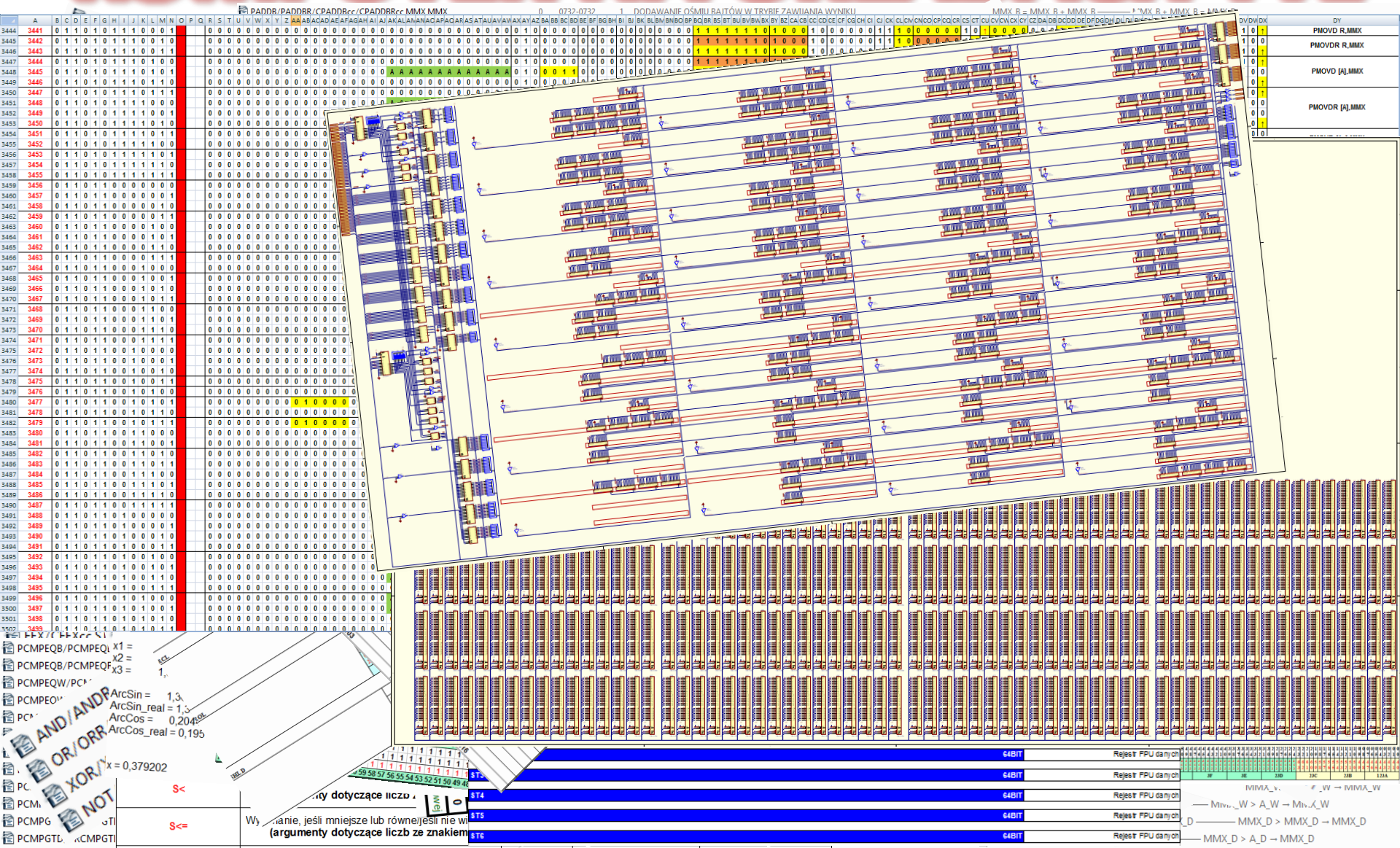
NOWY CEL

KOMPUTER NA UKŁADACH TTL



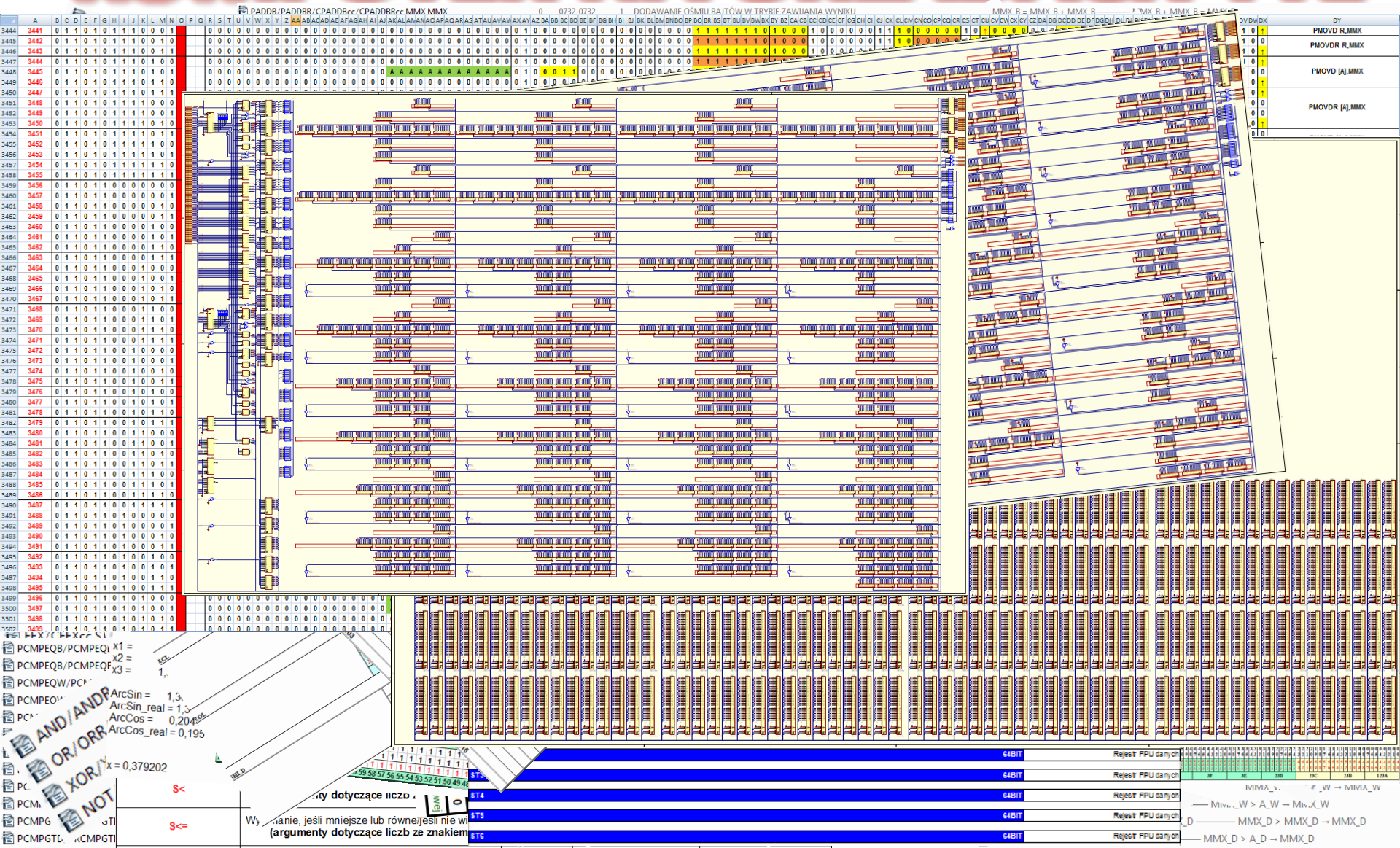
NOWY CEL

KOMPUTER NA UKŁADACH TTL



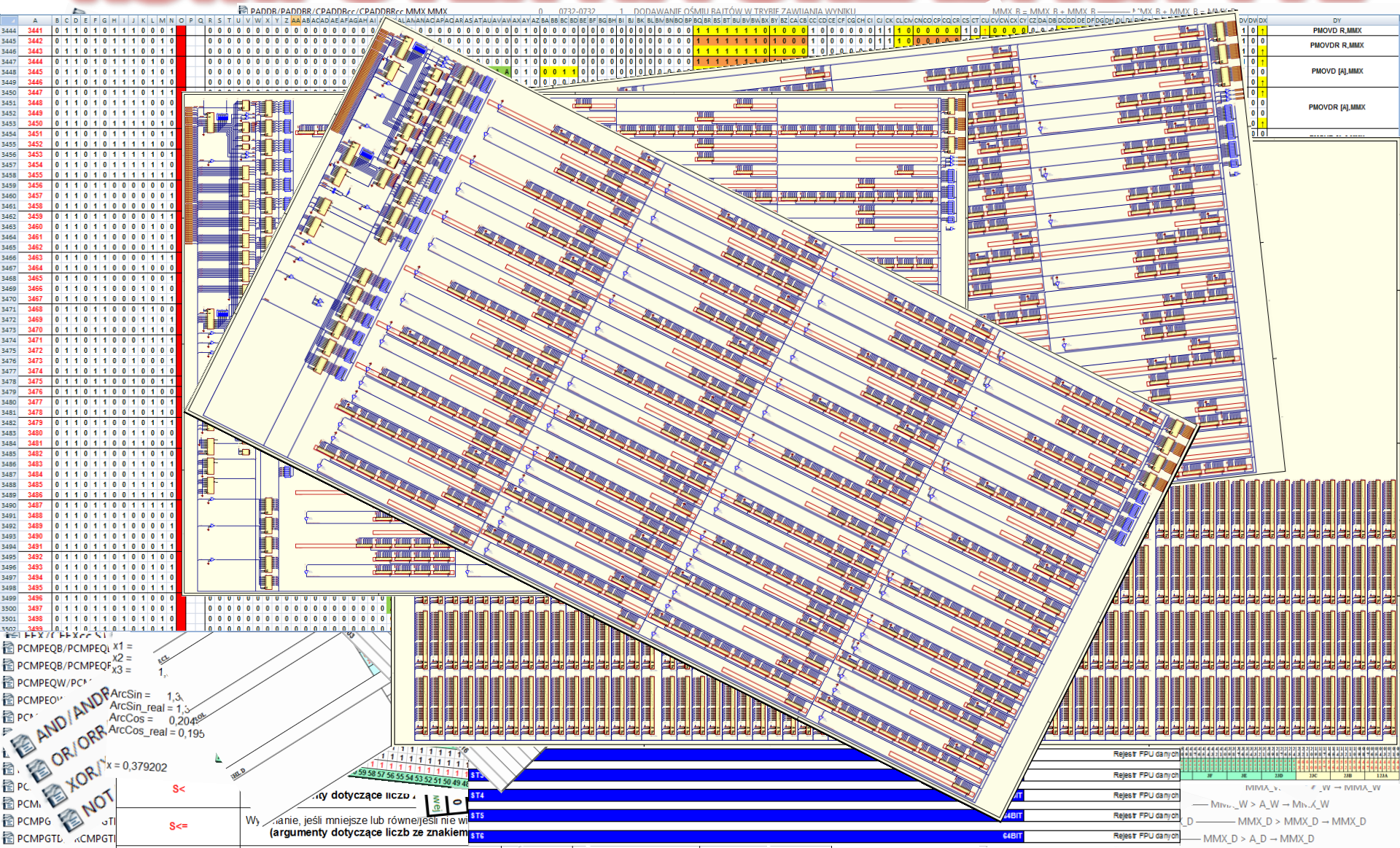
NOWY CEL

KOMPUTER NA UKŁADACH TTL



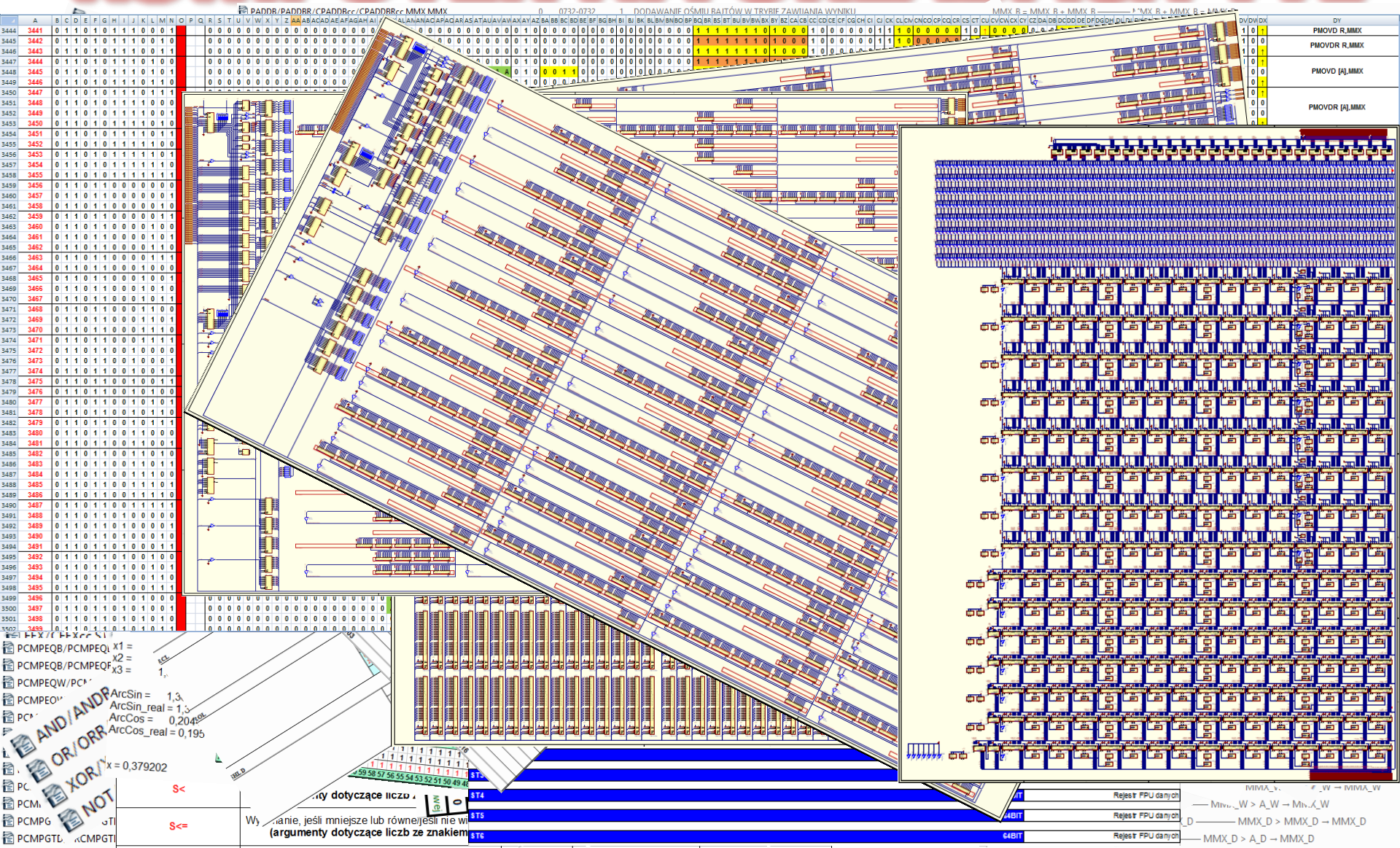
NOWY CEL

KOMPUTER NA UKŁADACH TTL



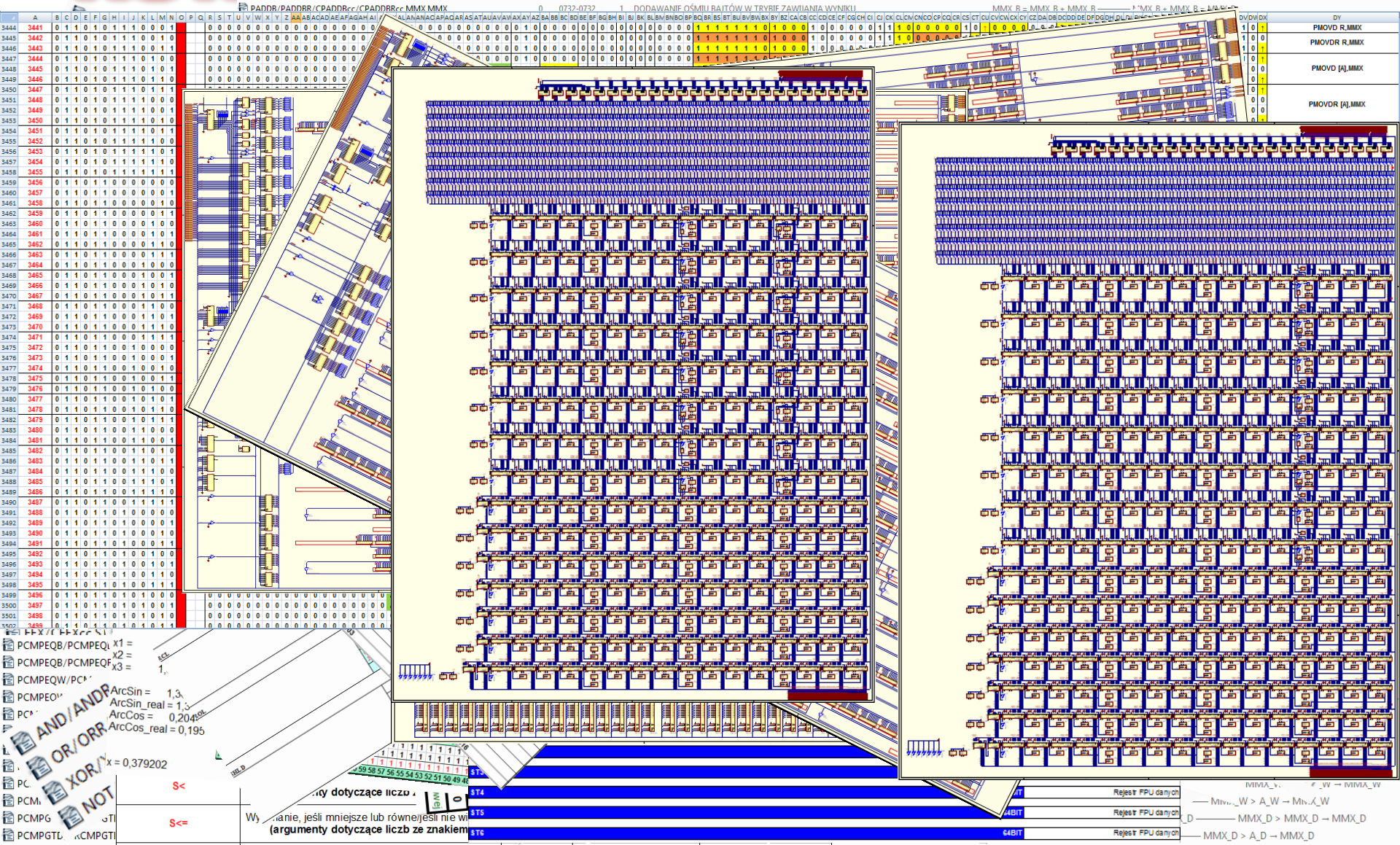
NOWY CEL

KOMPUTER NA UKŁADACH TTL



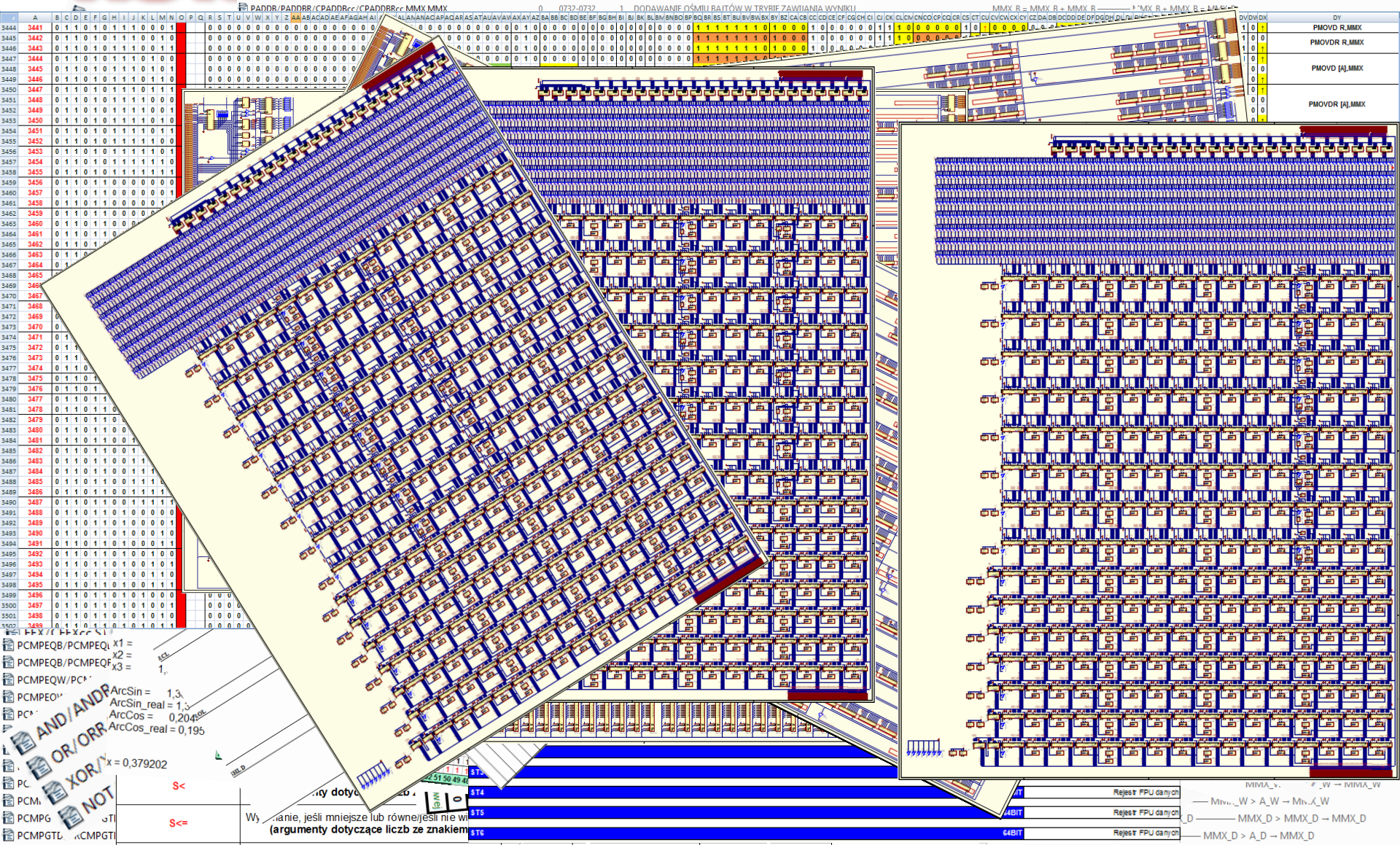
NOWY CEL

KOMPUTER NA UKŁADACH TTL



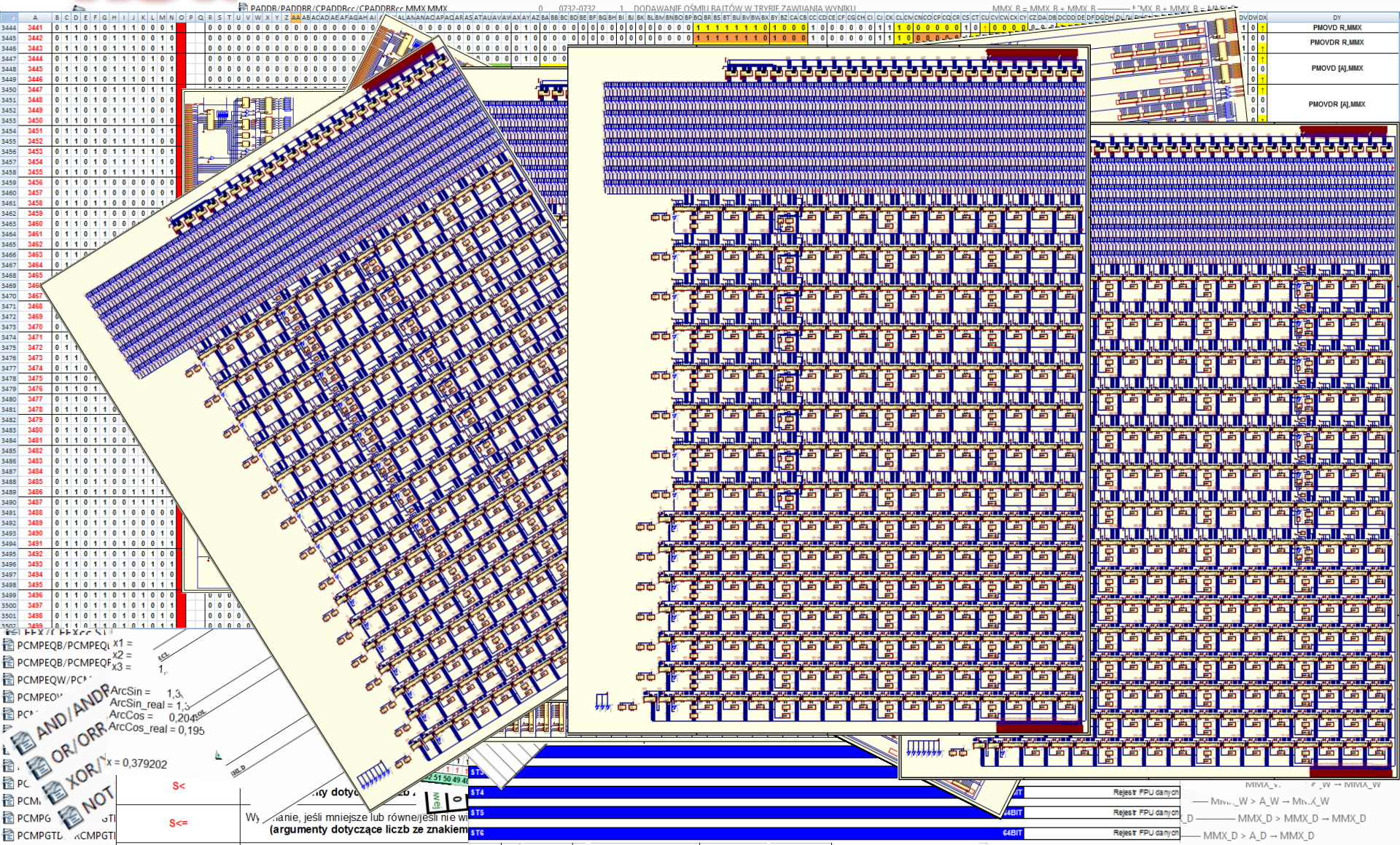
NOWY CEL

KOMPUTER NA UKŁADACH TTL



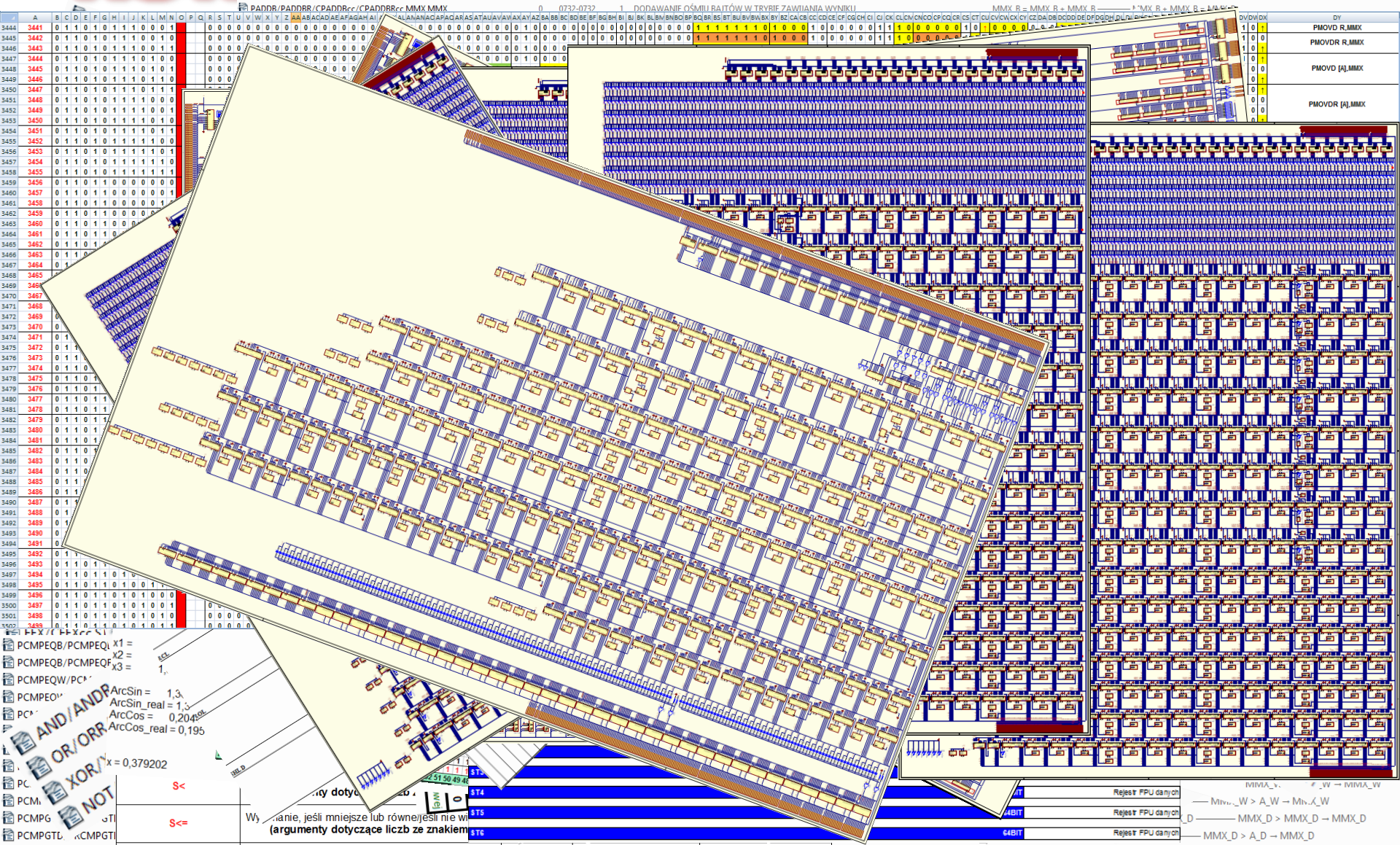
NOWY CEL

KOMPUTER NA UKŁADACH TTL



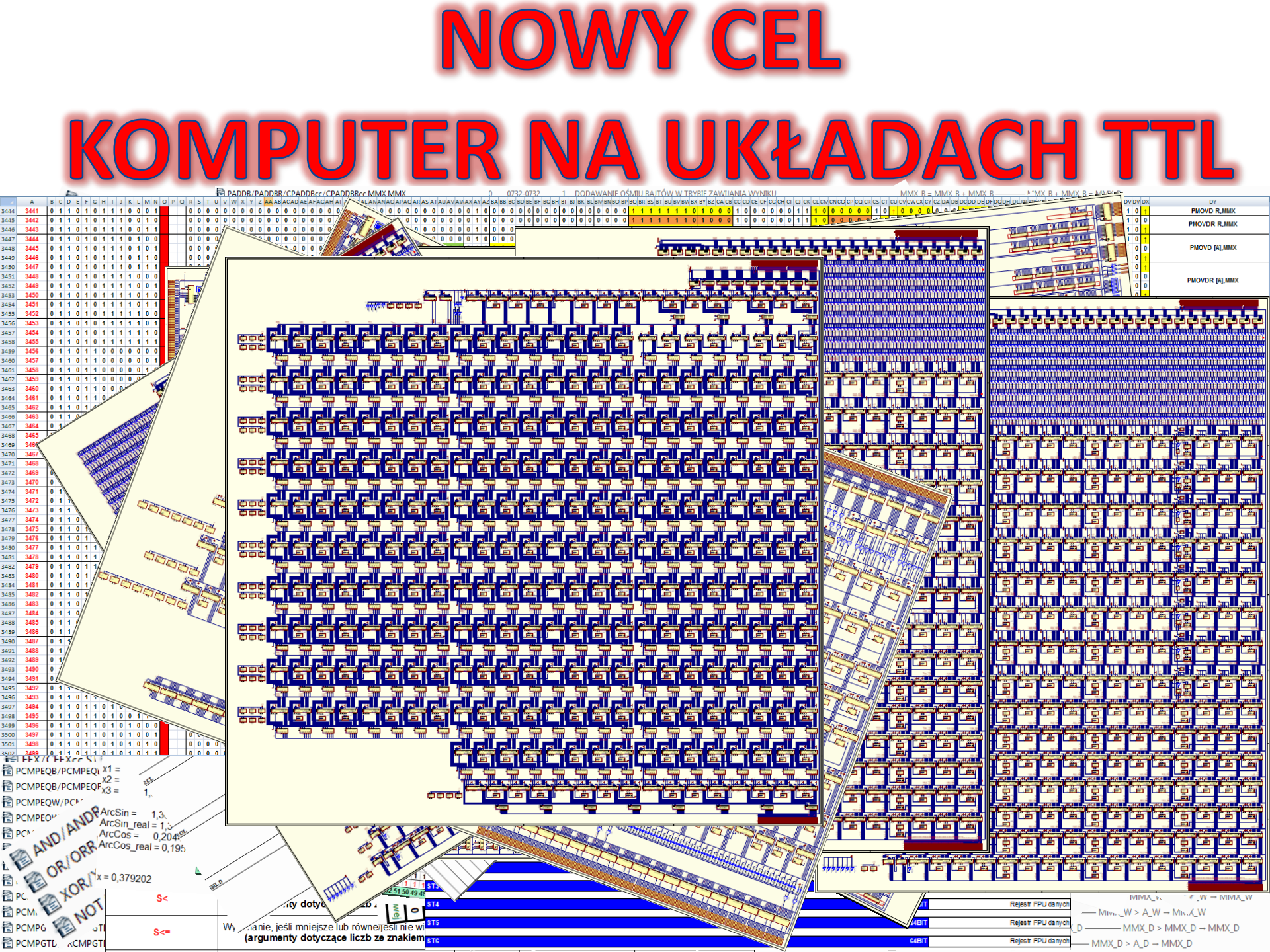
NOWY CEL

KOMPUTER NA UKŁADACH TTL



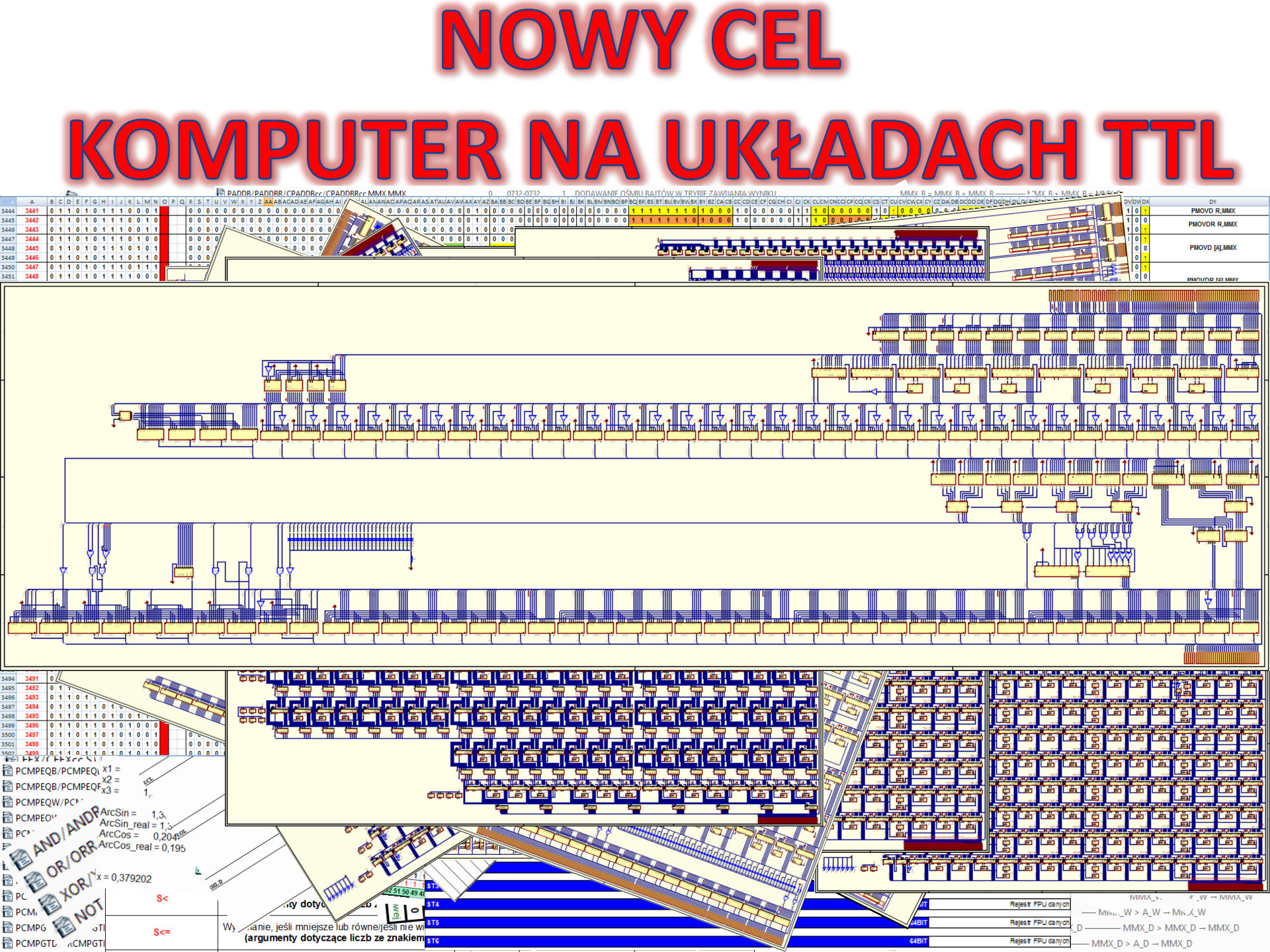
NOWY CEL

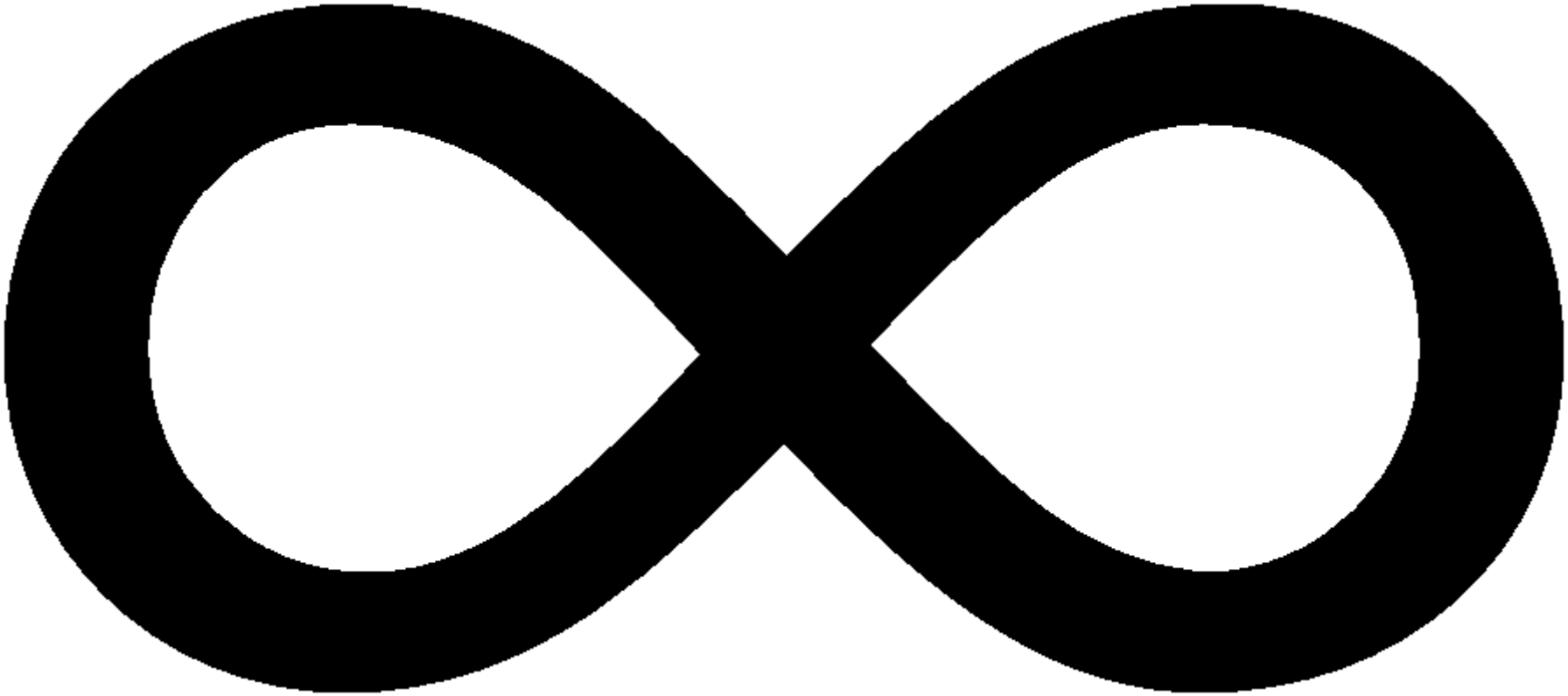
KOMPUTER NA UKŁADACH TTL



NOWY CEL

KOMPUTER NA UKŁADACH TTL





WIEDZA+WYOBRAŹNIA=

FANTAZJA

Dziękuje za uwagę
Rafał Wiśniewski