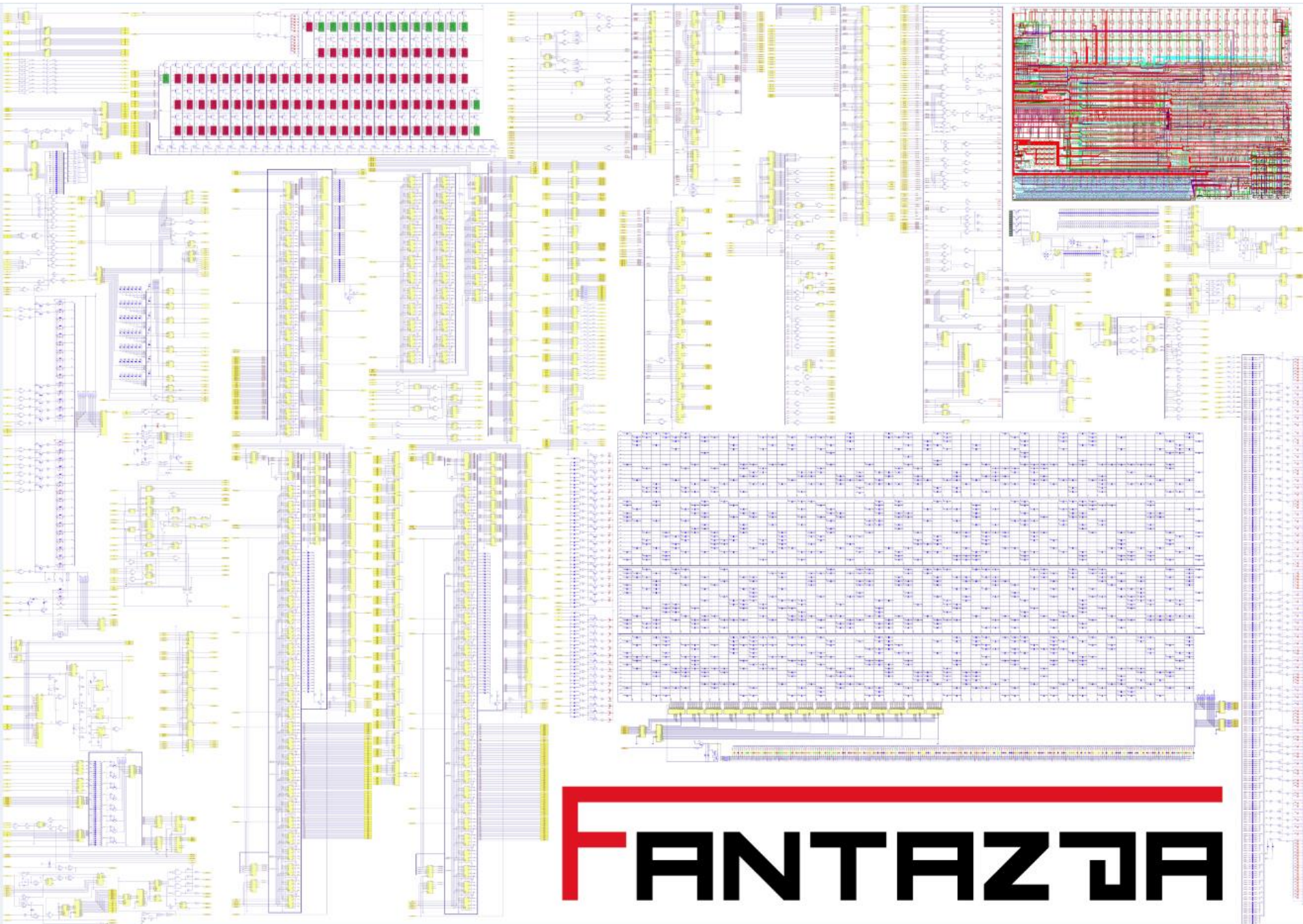


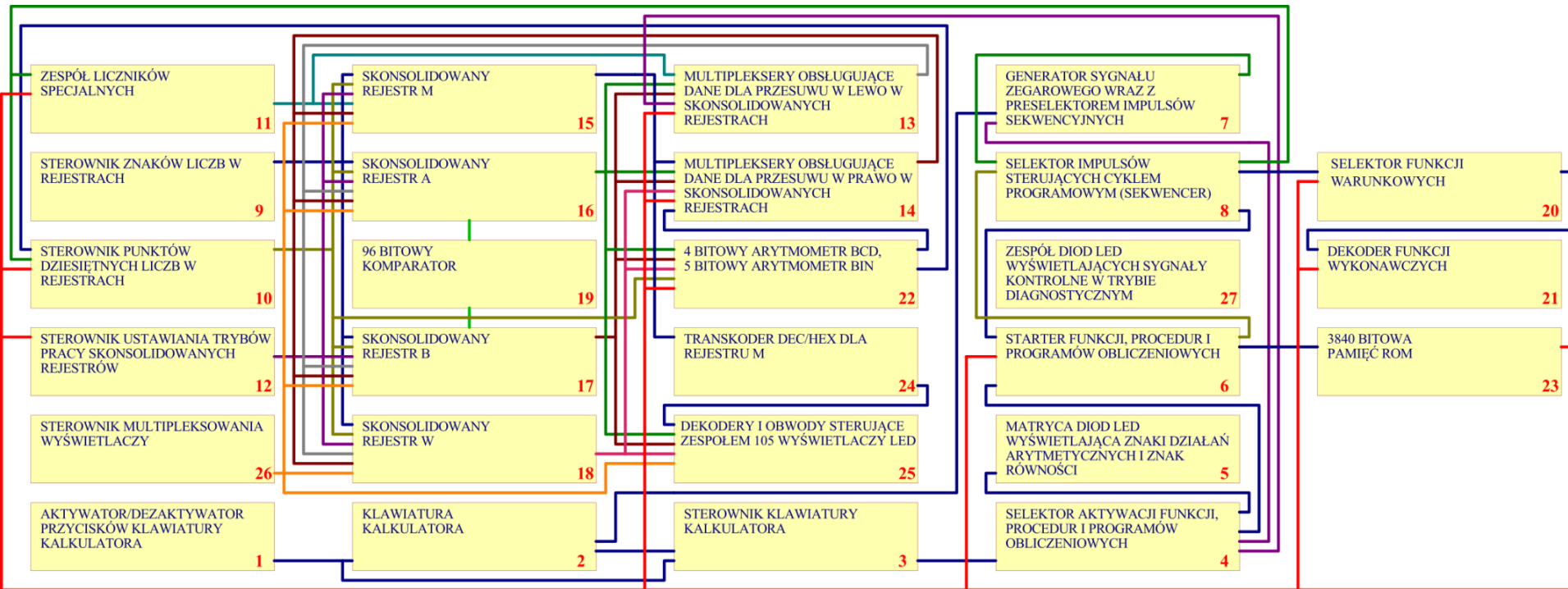


# KALKULATOR NA UKŁADACH TTL Z TRYBEM DIAGNOSTYCZNYM



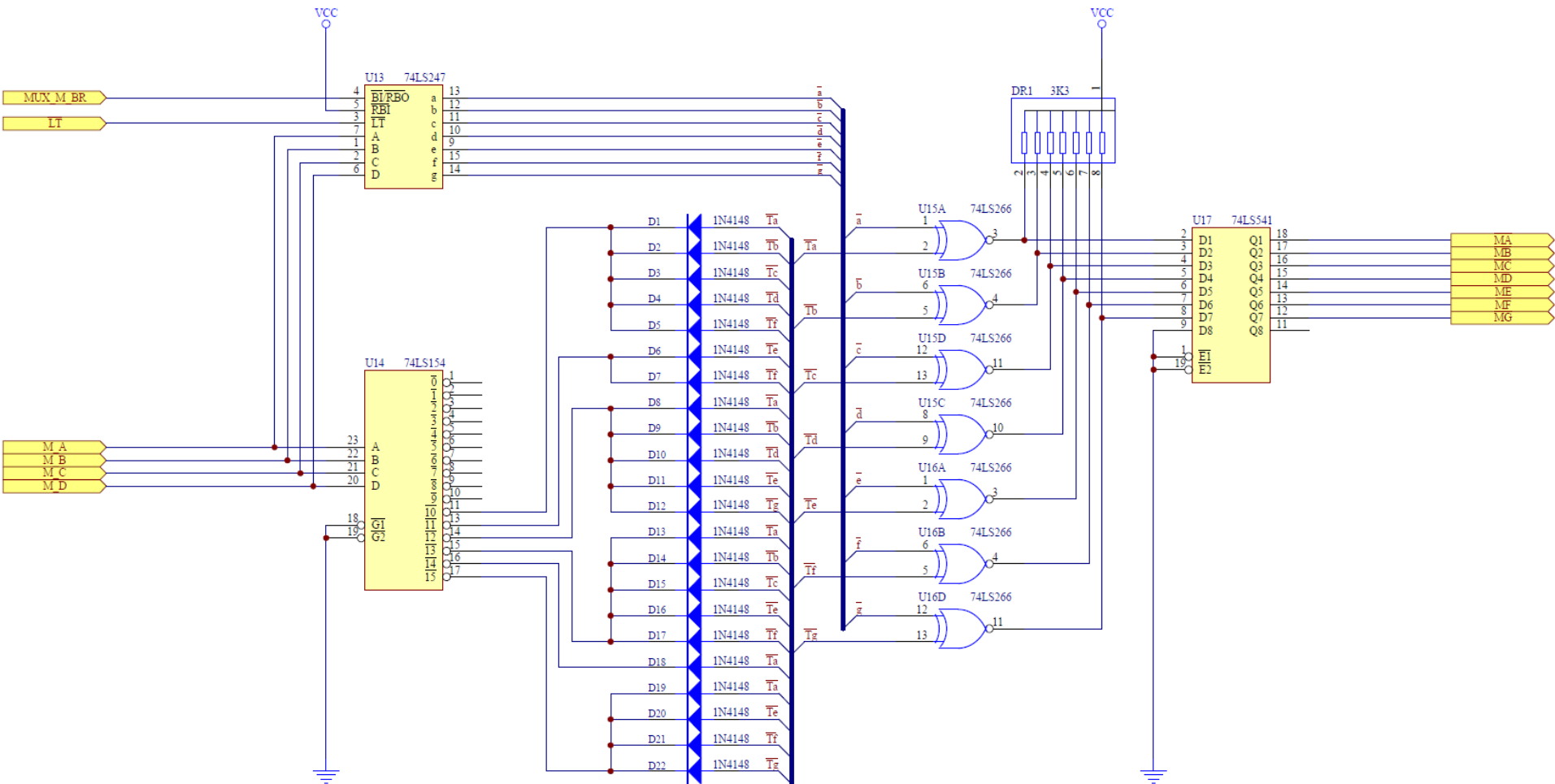
**FANTAZJA**

# SCHEMAT BLOKOWY URZĄDZENIA

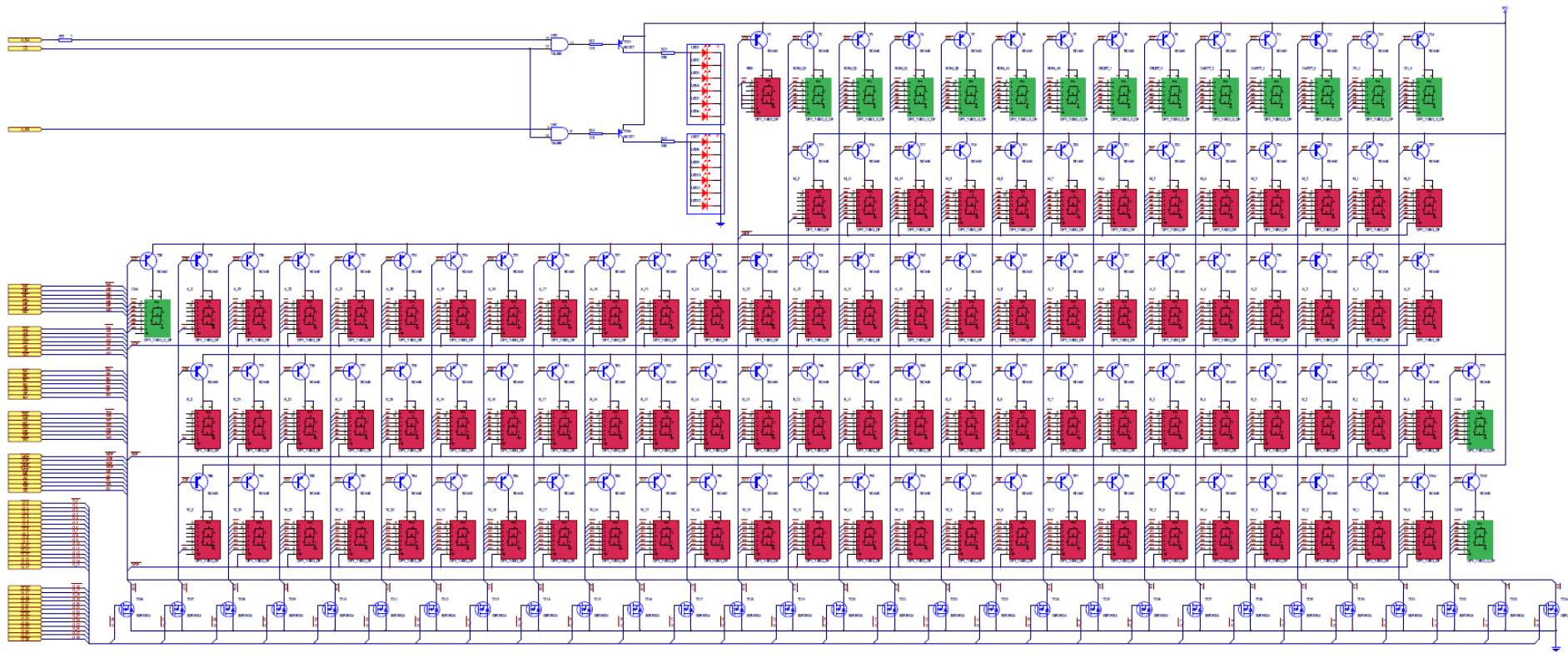




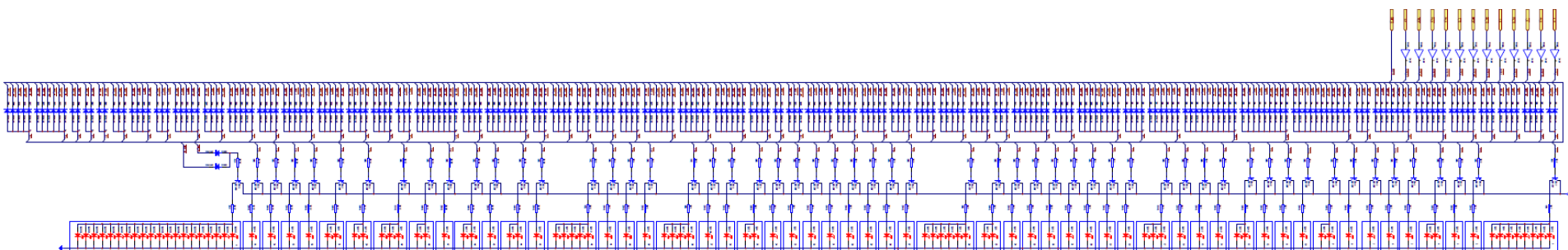
# 2. TRANSKODER DEC/HEX DLA WYŚWIETLACZY REJESTRU M



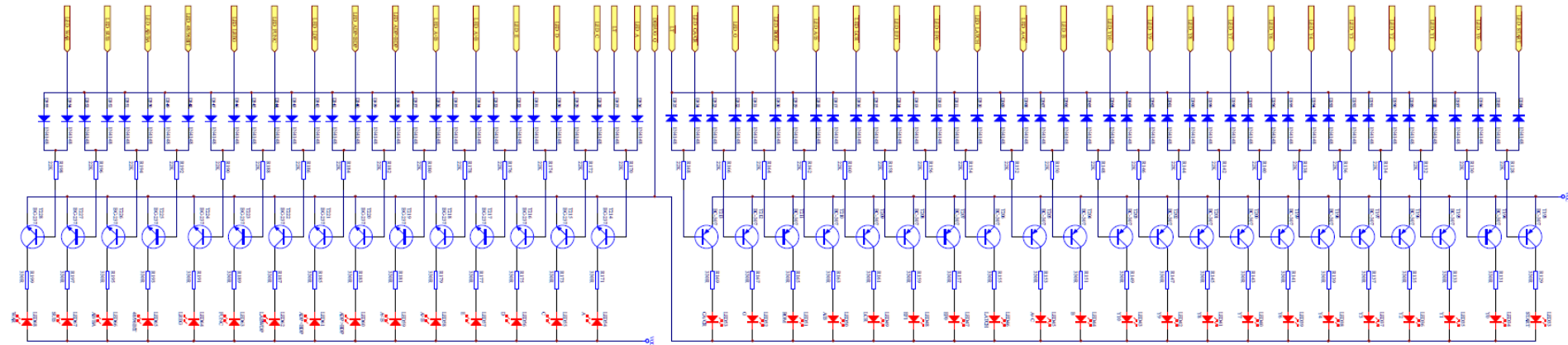
# 3. ZESPÓŁ 105 WYŚWIETLACZY SIEDMIOSEGMENTOWYCH WRAZ Z DIODAMI LED WYBORU AKTYWNEGO REJESTRU



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

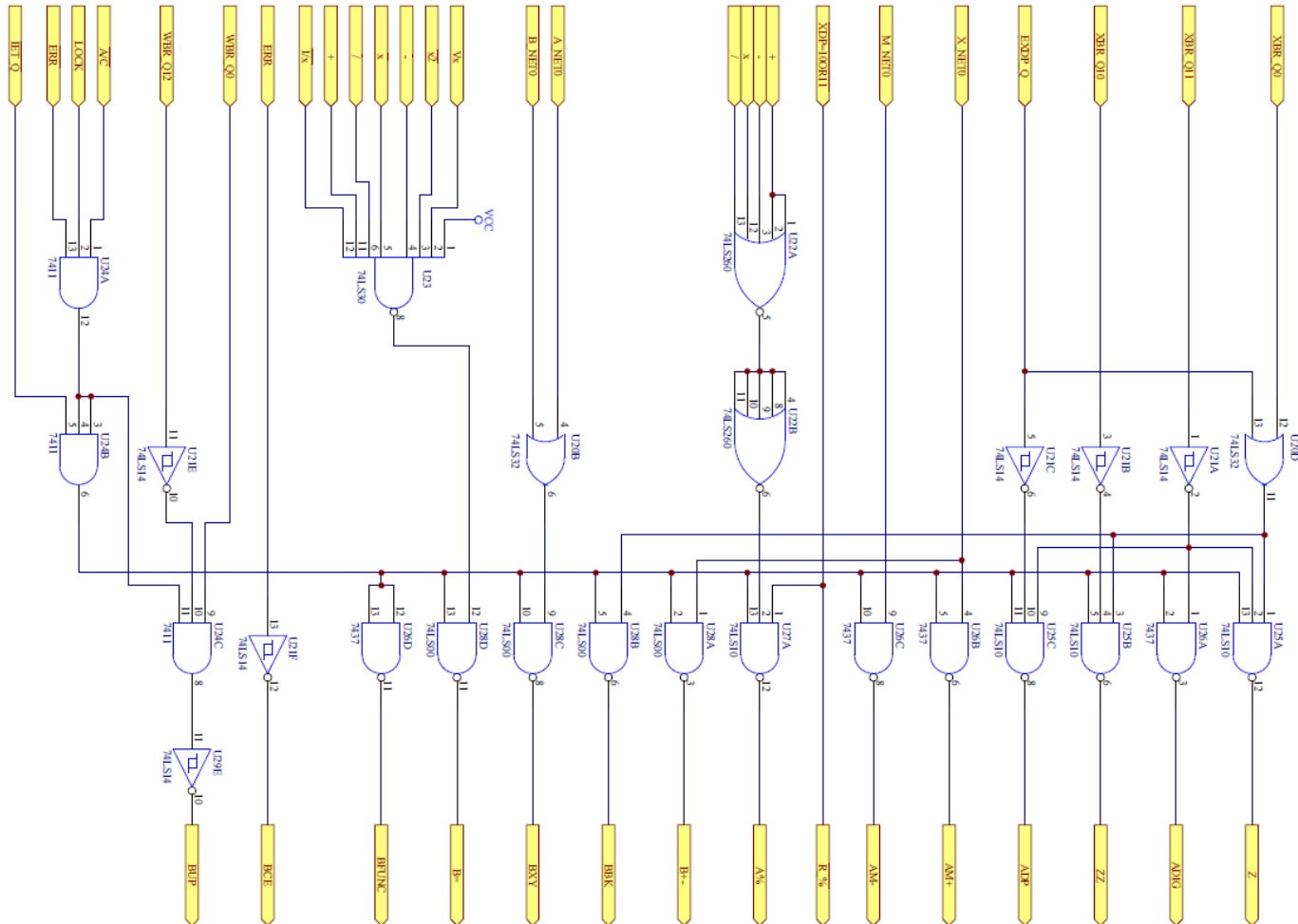


# 5. ZESPÓŁ DIOD LED WYŚWIETLAJĄCYCH SYGNAŁY KONTROLNE W TRYBIE KROKOWYM

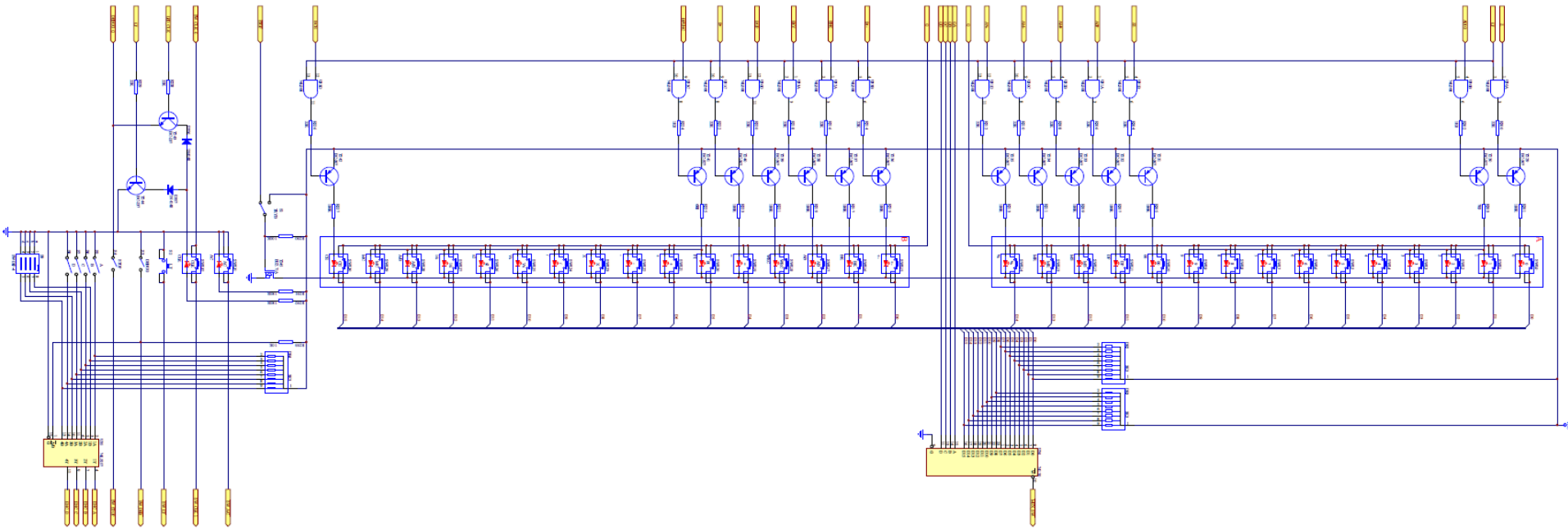




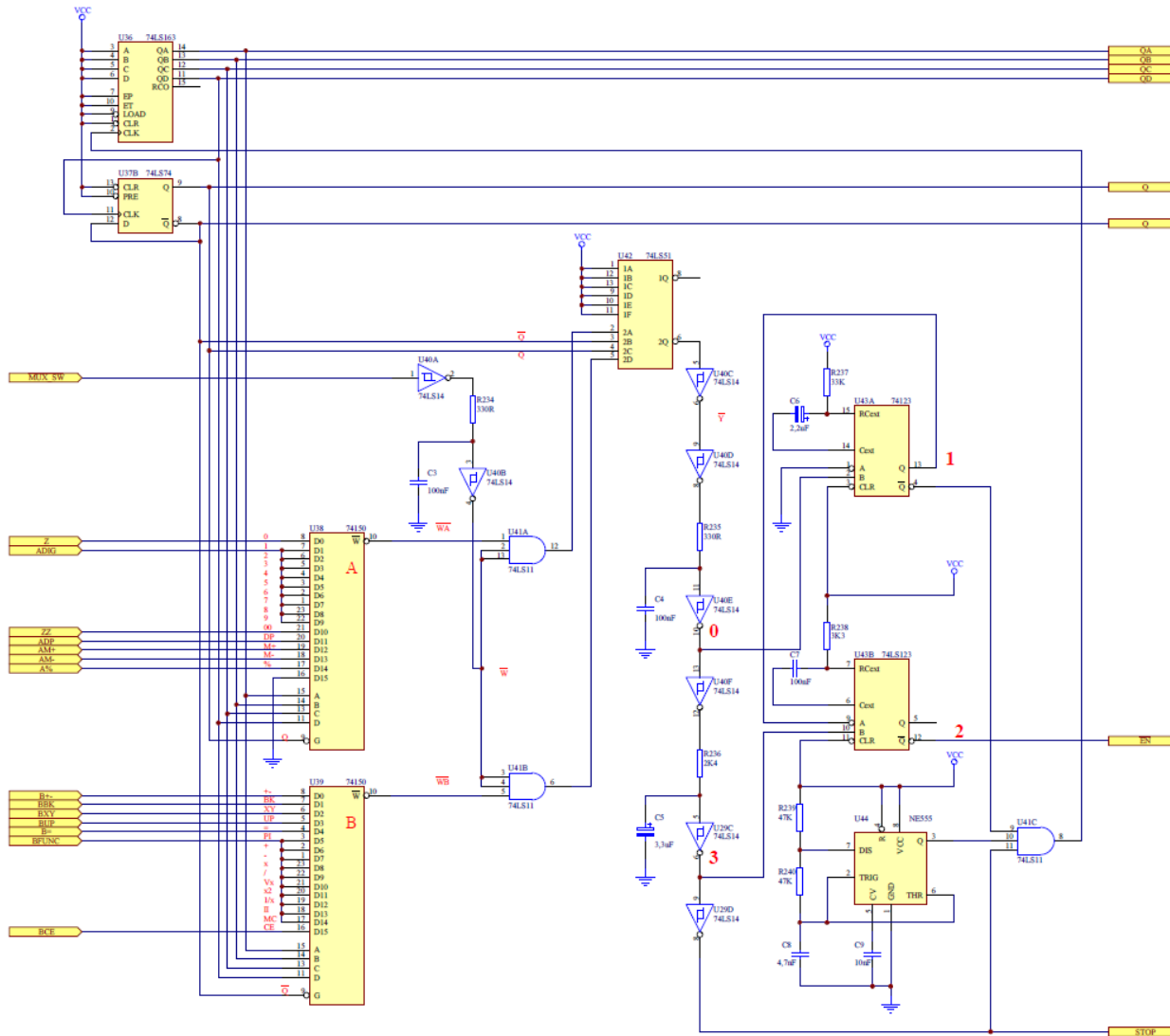
# 6. AKTYWATOR/DEZAKTYWATOR PRZYCISKÓW KLAWIATURY KALKULATORA



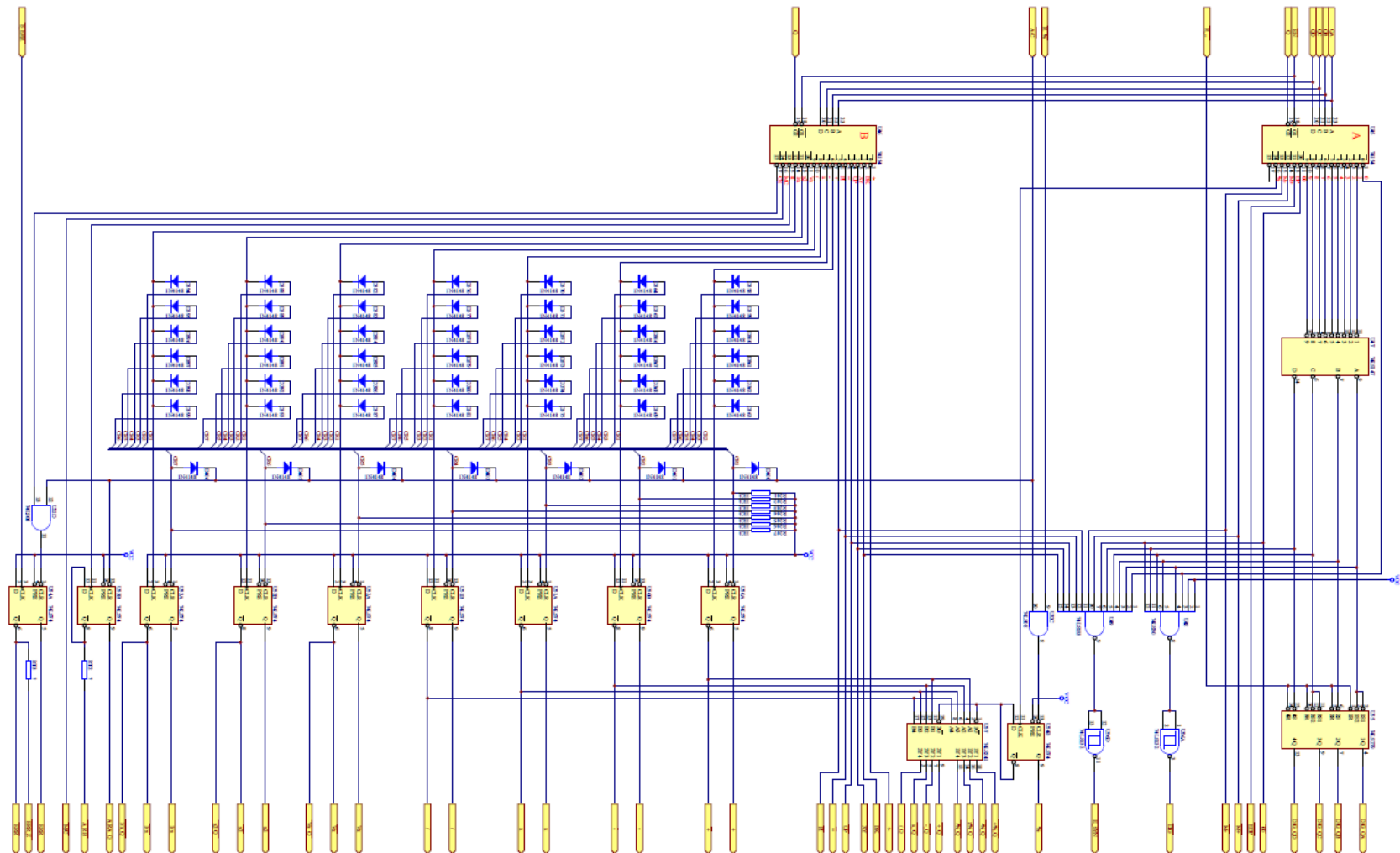
# 7. KLAWIATURA KALKULATORA



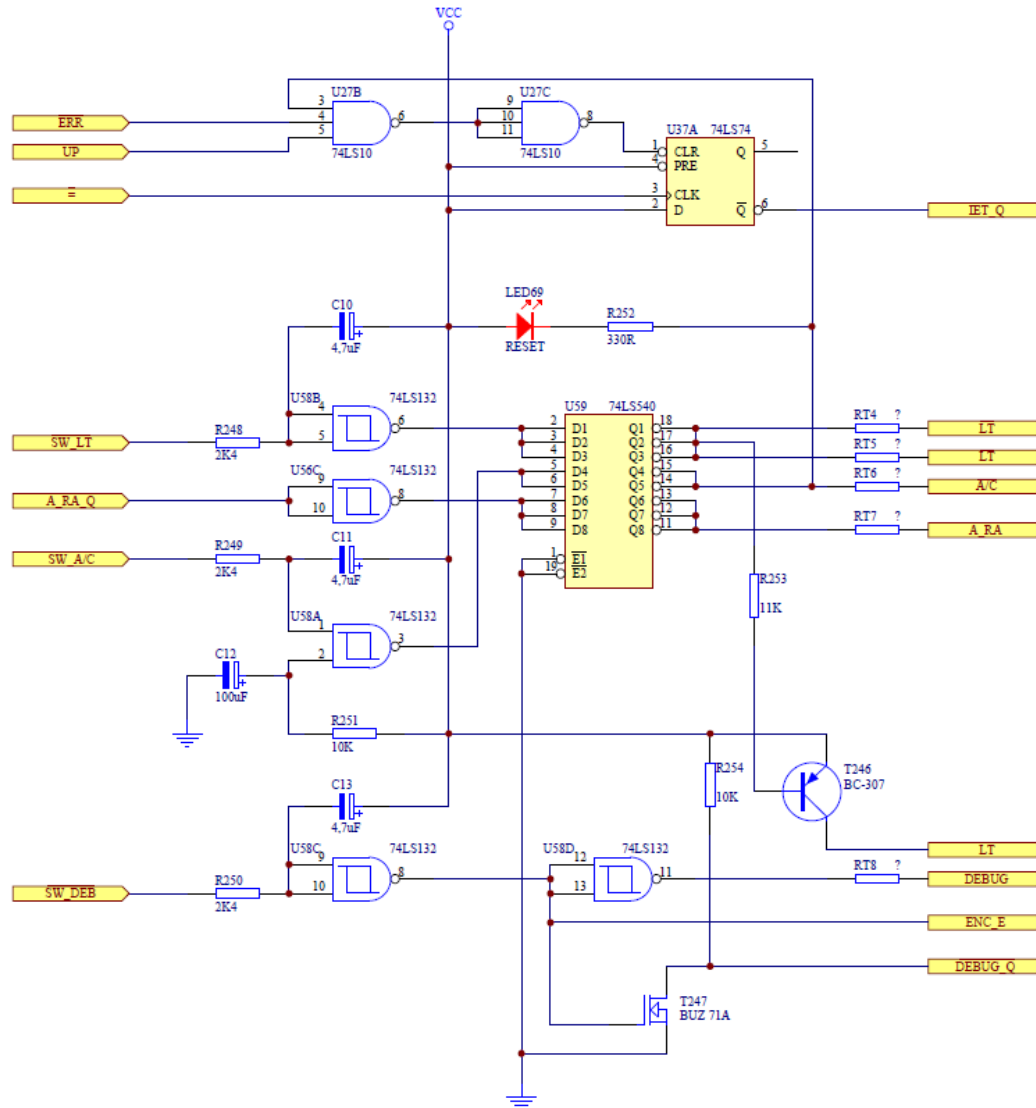
# 8. STEROWNIK KLAWIATURY KALKULATORA



# 9. SELEKTOR AKTYWACJI FUNKCJI, PRECEDUR I PROGRAMÓW OBLICZENIOWYCH

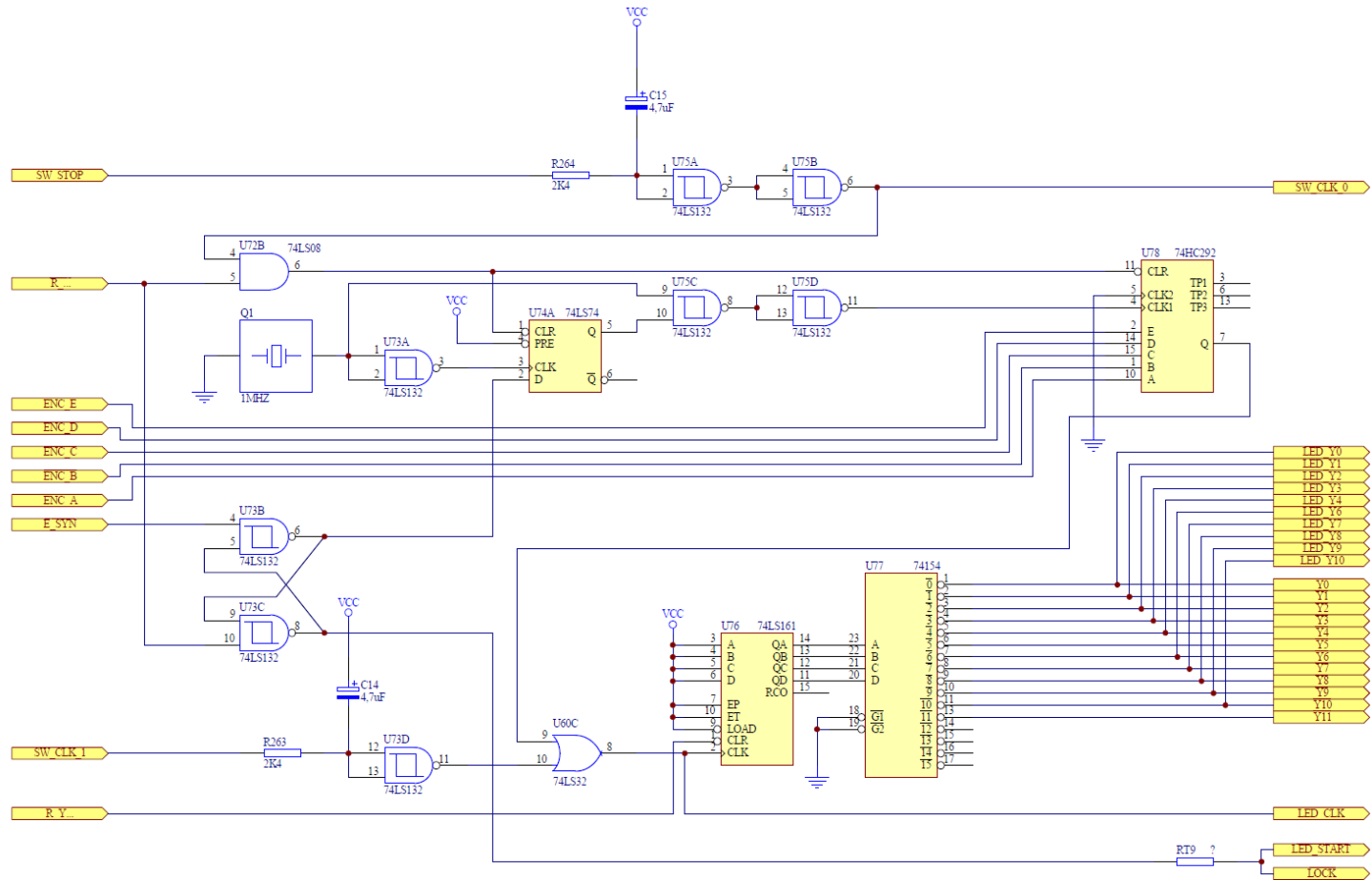


# 10. BUFOR SYGNAŁÓW NADRZĘDNYCH

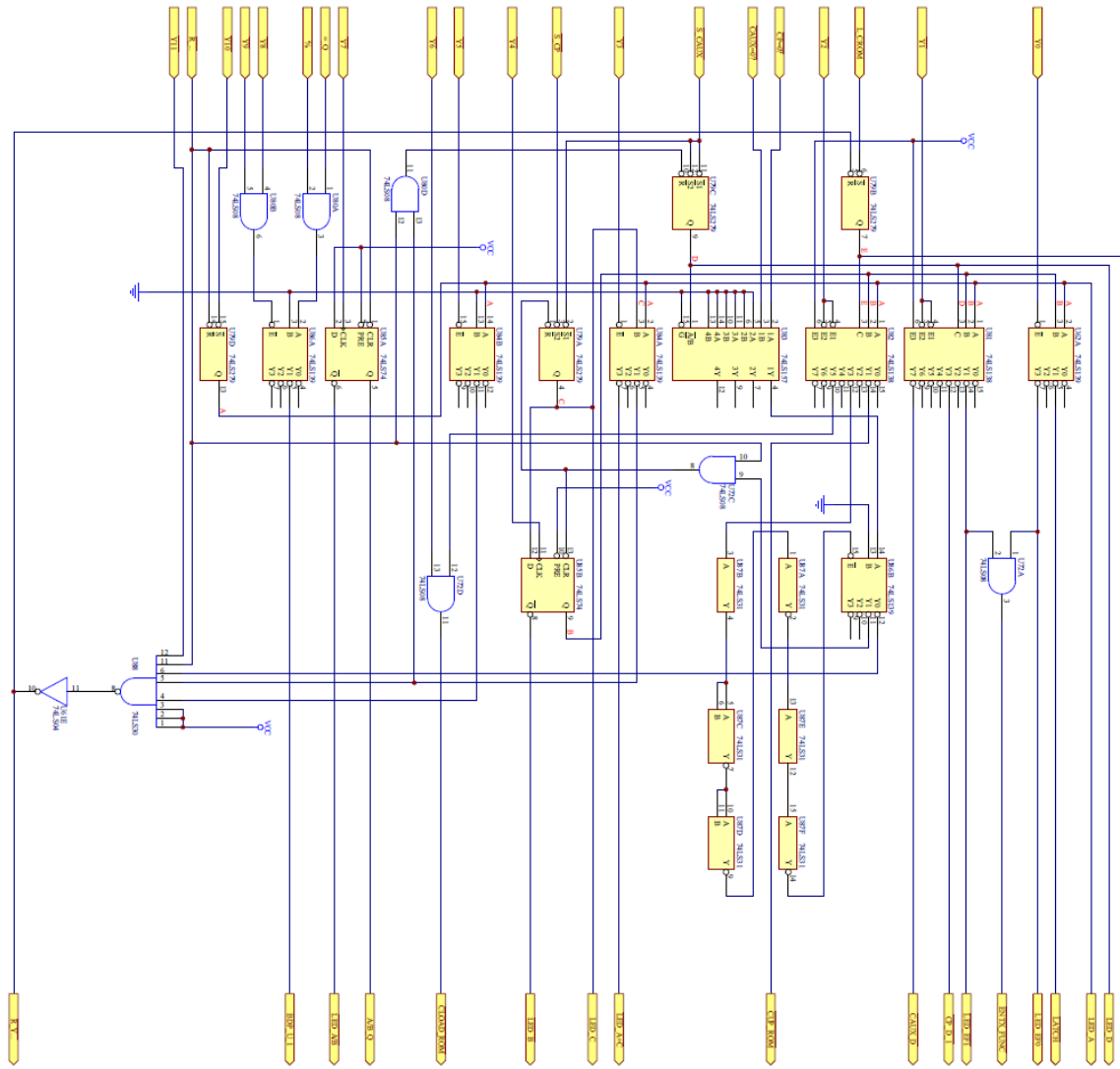




# 12. GENERATOR SYGNAŁU ZEGAROWEGO WRAZ Z PRESELEKTOREM IMPULSÓW SEKWENCYJNYCH

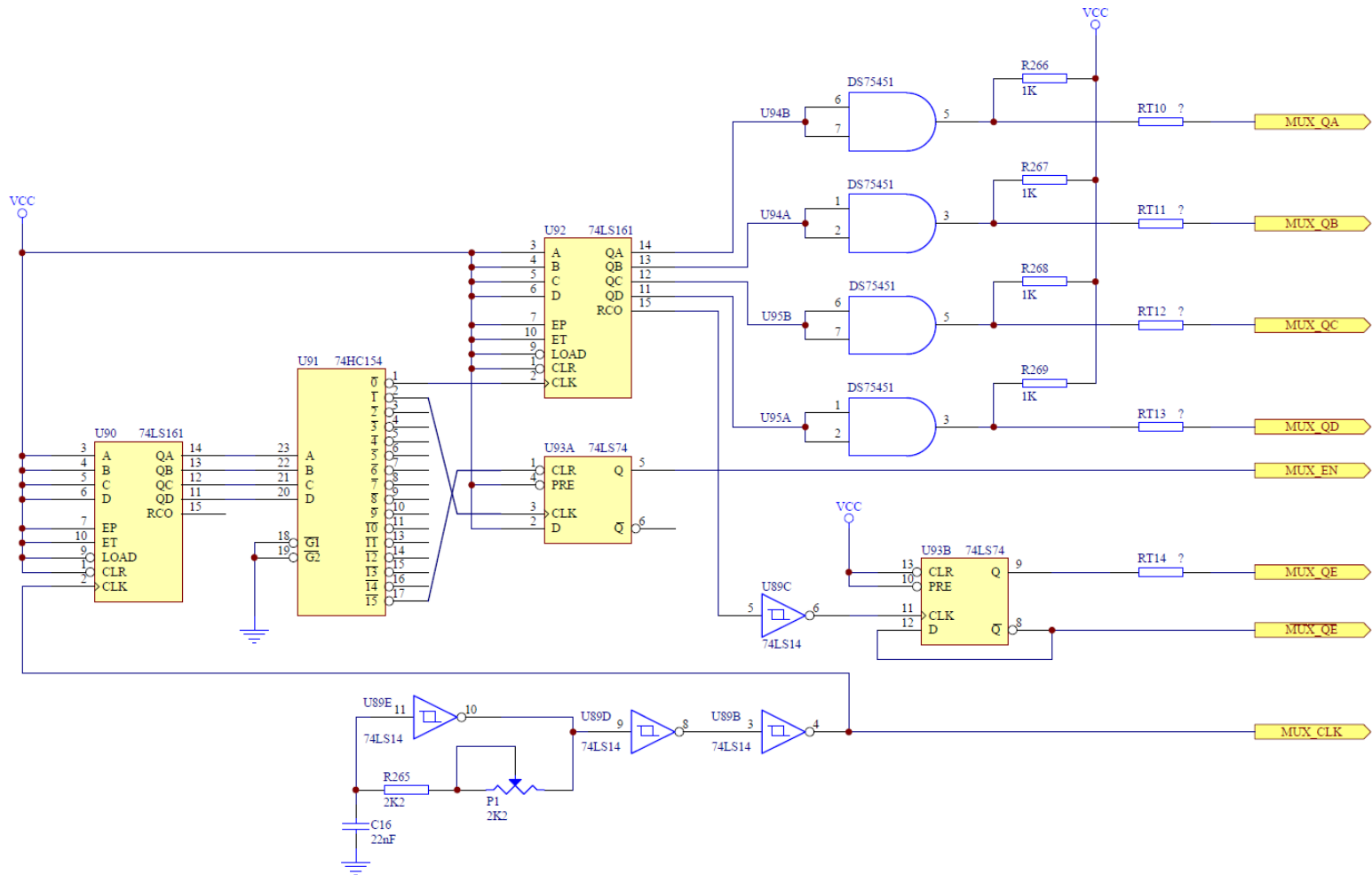


# 13. SELEKTOR IMPULSÓW STERUJĄCYCH CYKLEM PROGRAMOWYM (SEKWENCER)

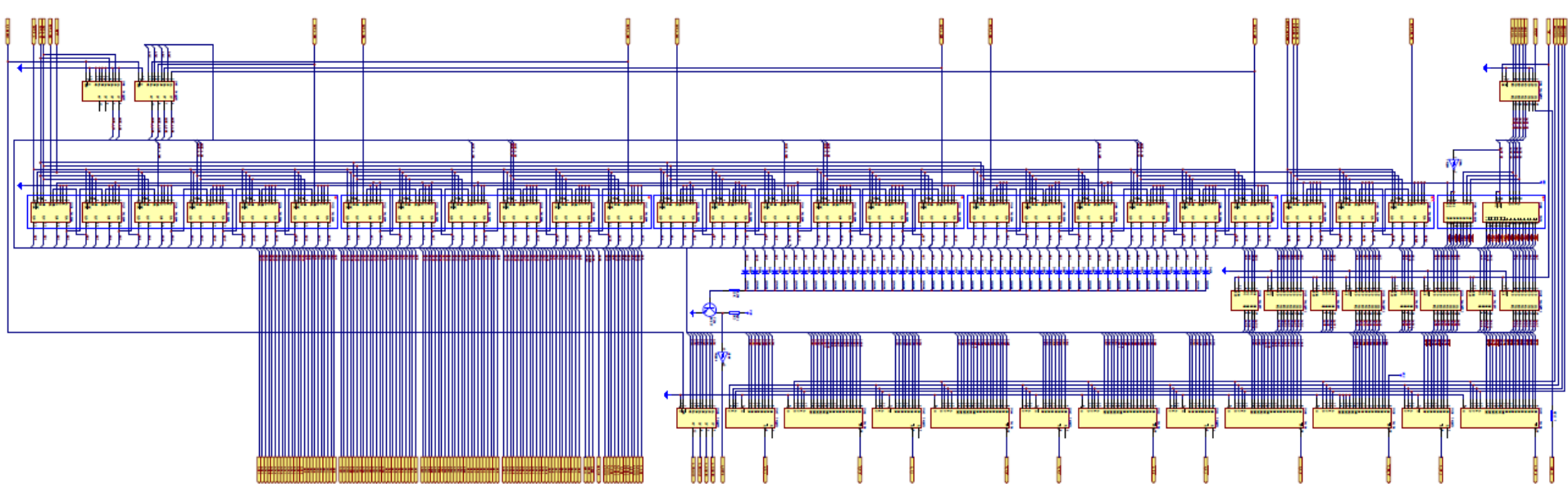




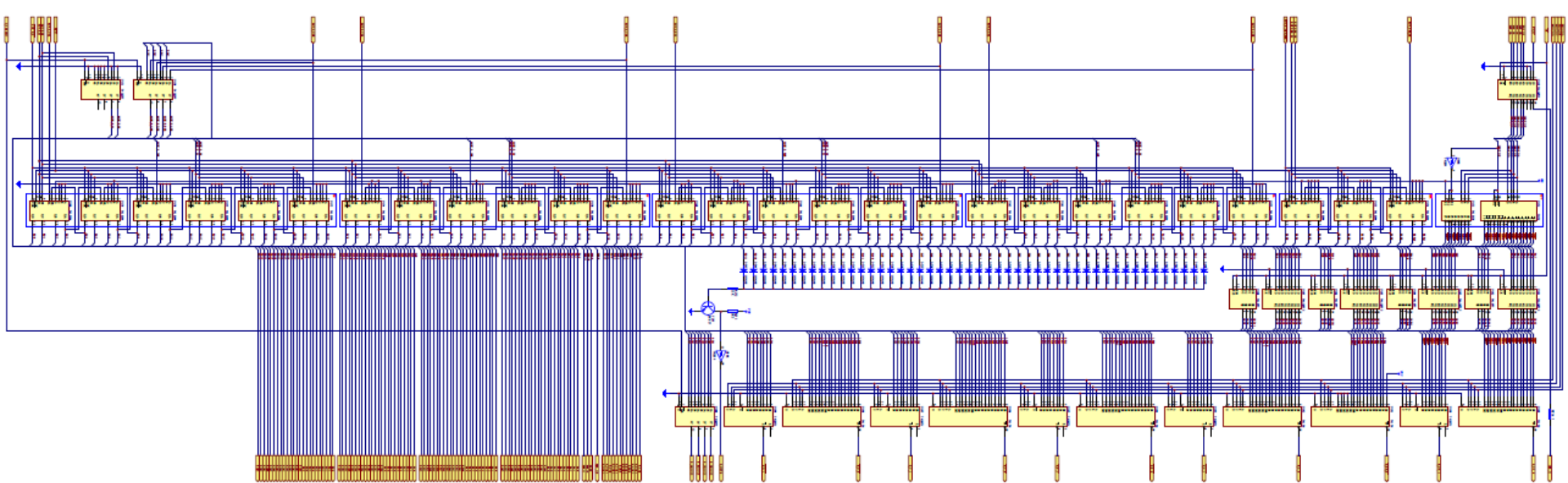
# 14. STEROWNIK MULTIPLEKSOWANIA WYŚWIETLACZY



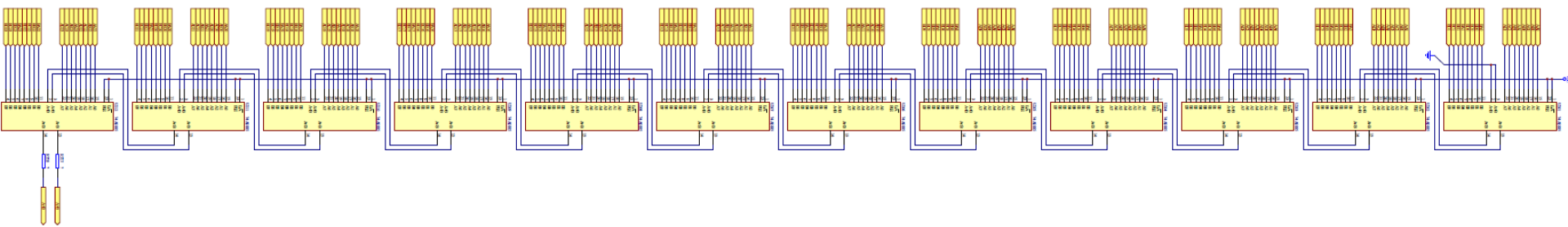
# 15. ZESPÓŁ REJESTRÓW I DEKODERÓW PREZENTUJĄCYCH WARTOŚĆ LICZBY „A”



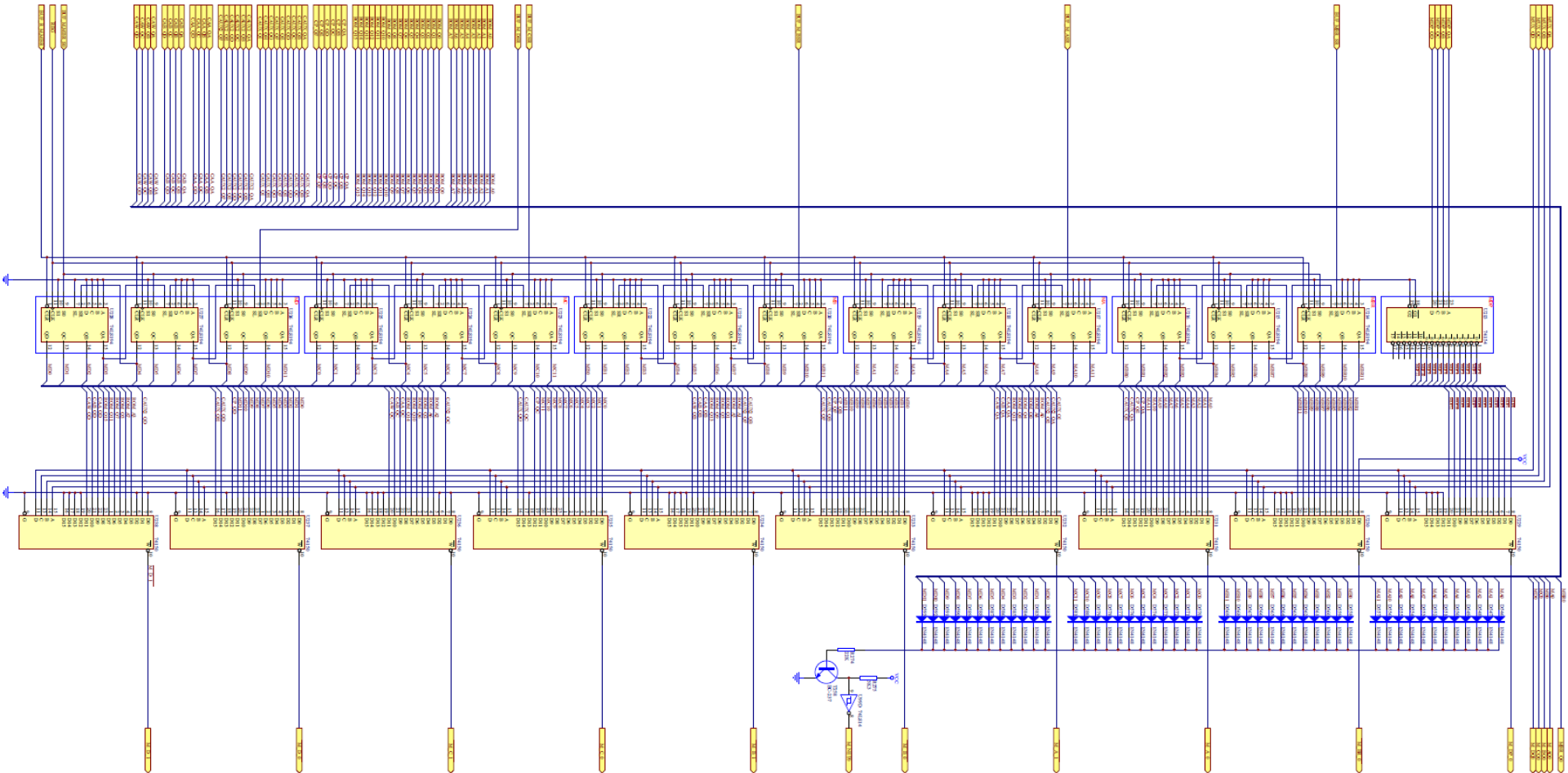
# 16. ZESPÓŁ REJESTRÓW I DEKODERÓW PREZENTUJĄCYCH WARTOŚĆ LICZBY „B”



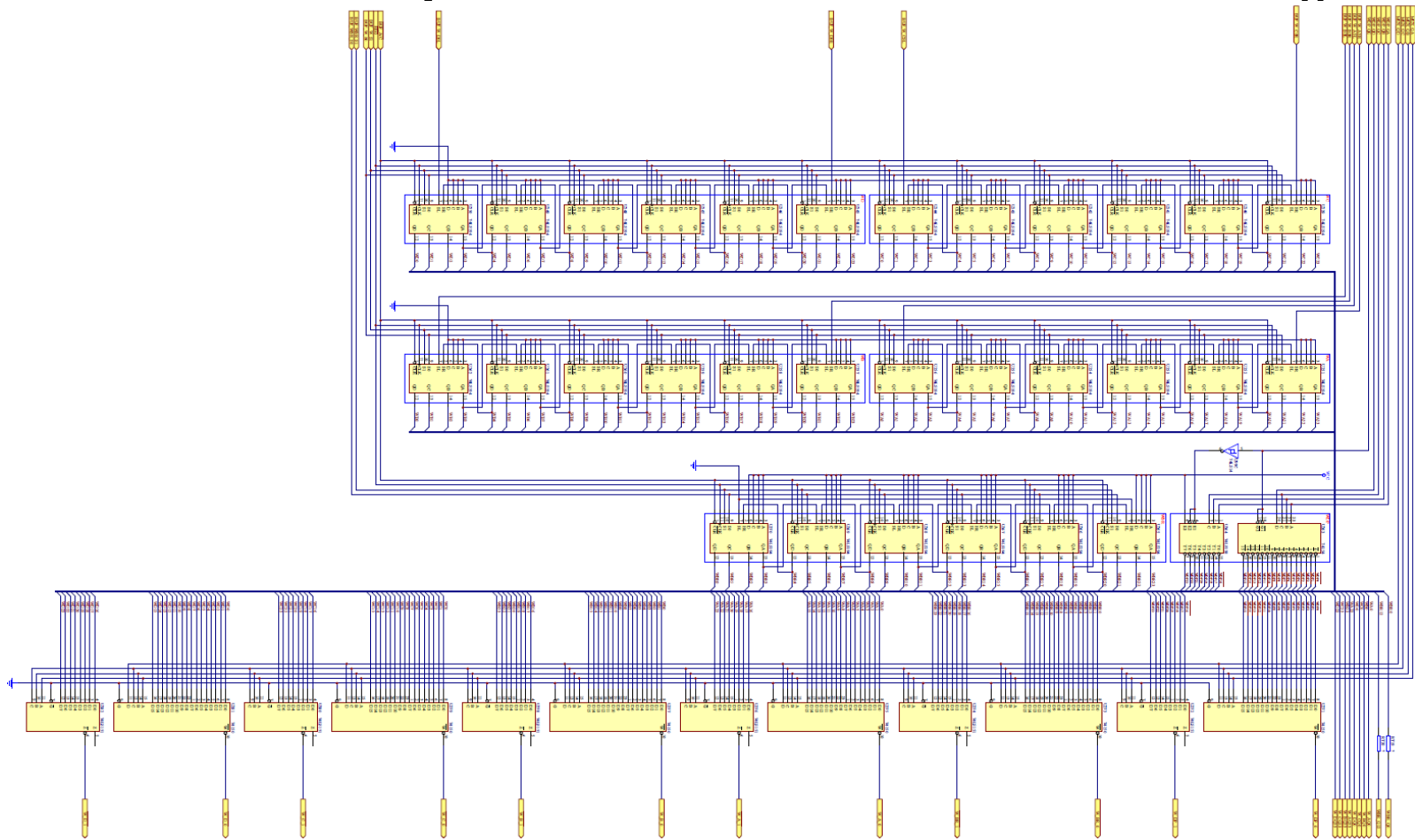
# 17.96 BITOWY KOMPARATOR



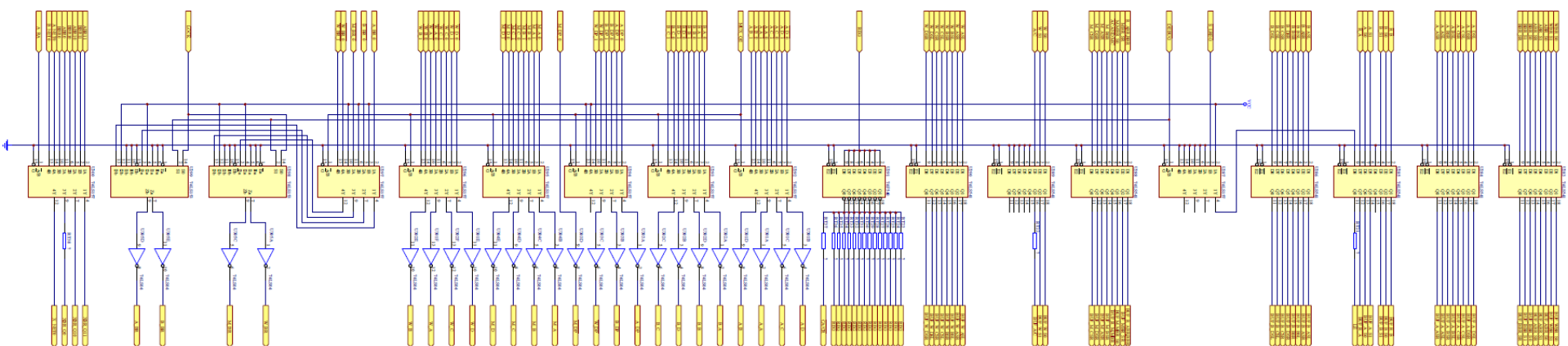
# 18. ZESPÓŁ REJESTRÓW I DEKODERÓW PREZENTUJĄCYCH WARTOŚĆ LICZBY „M”



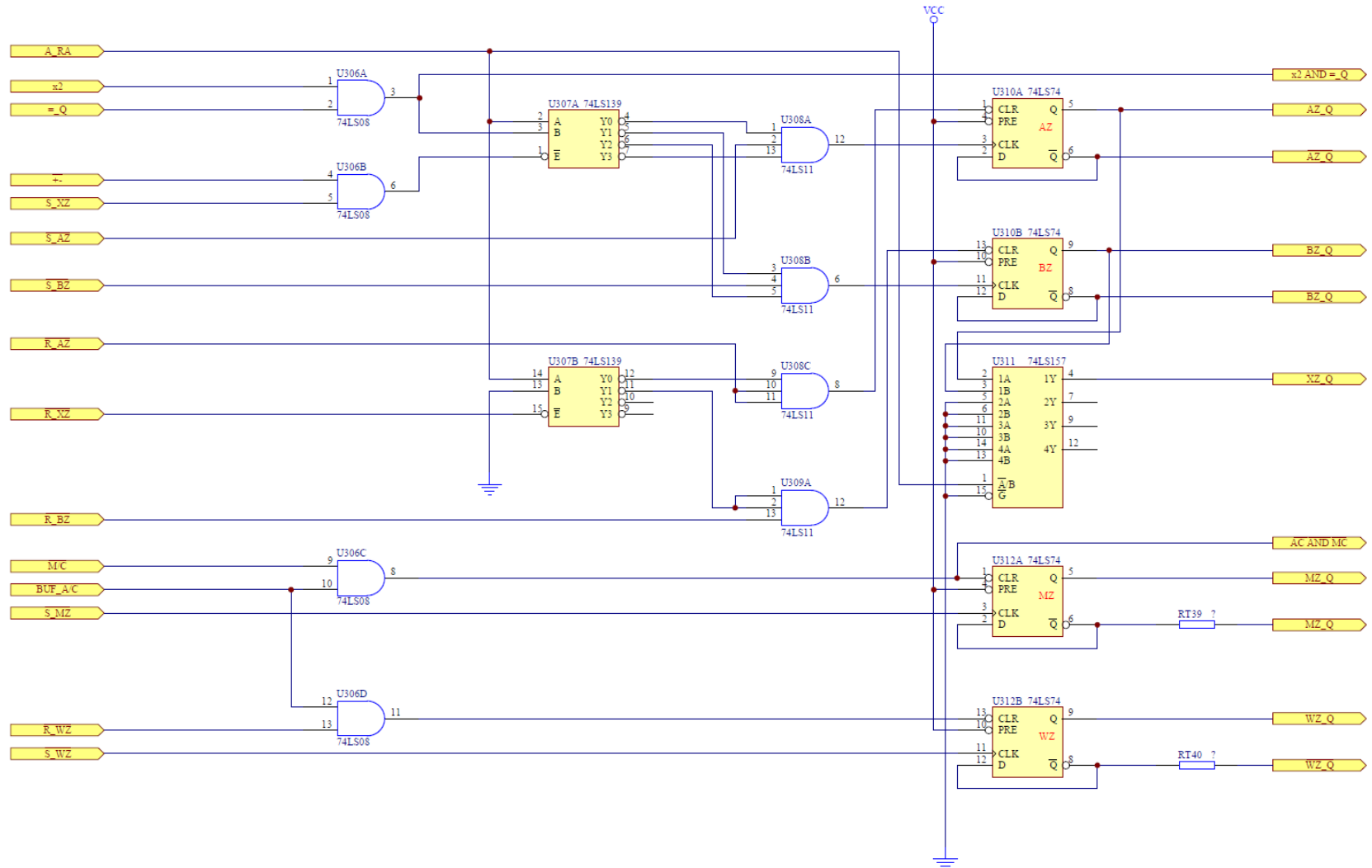
# 19. ZESPÓŁ REJESTRÓW I DEKODERÓW PREZENTUJĄCYCH WARTOŚĆ LICZBY „W”



# 20. BUFORY SYGNAŁÓW WEJŚCIOWYCH DLA ZESPOŁÓW REJSTRÓW, DEKODERÓW ORAZ MULTIPLEKSESY STERUJĄCE ZESPOŁEM WYŚWIETLACZY

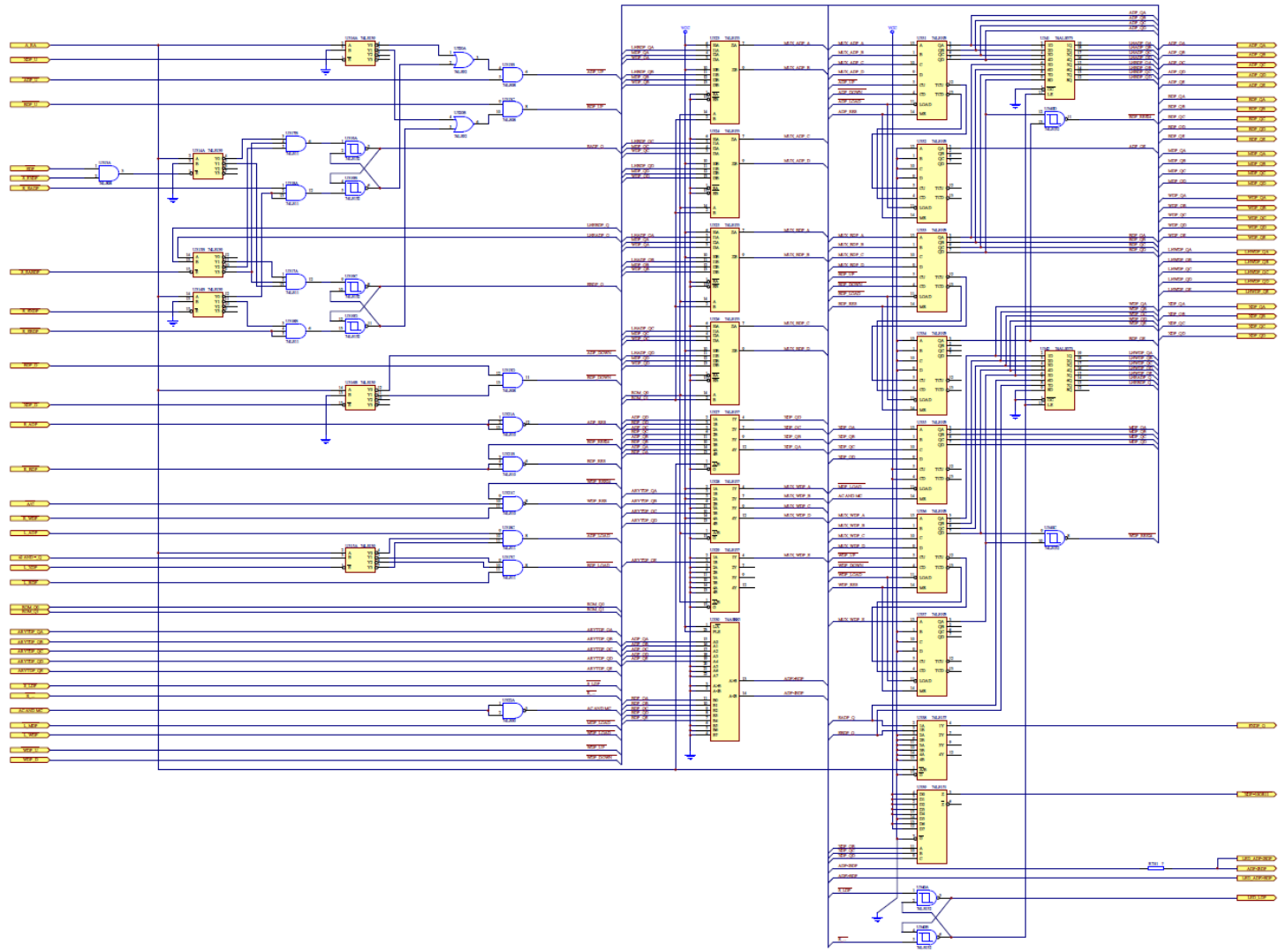


# 21. STEROWNIK ZNAKÓW WARTOŚCI LICZB



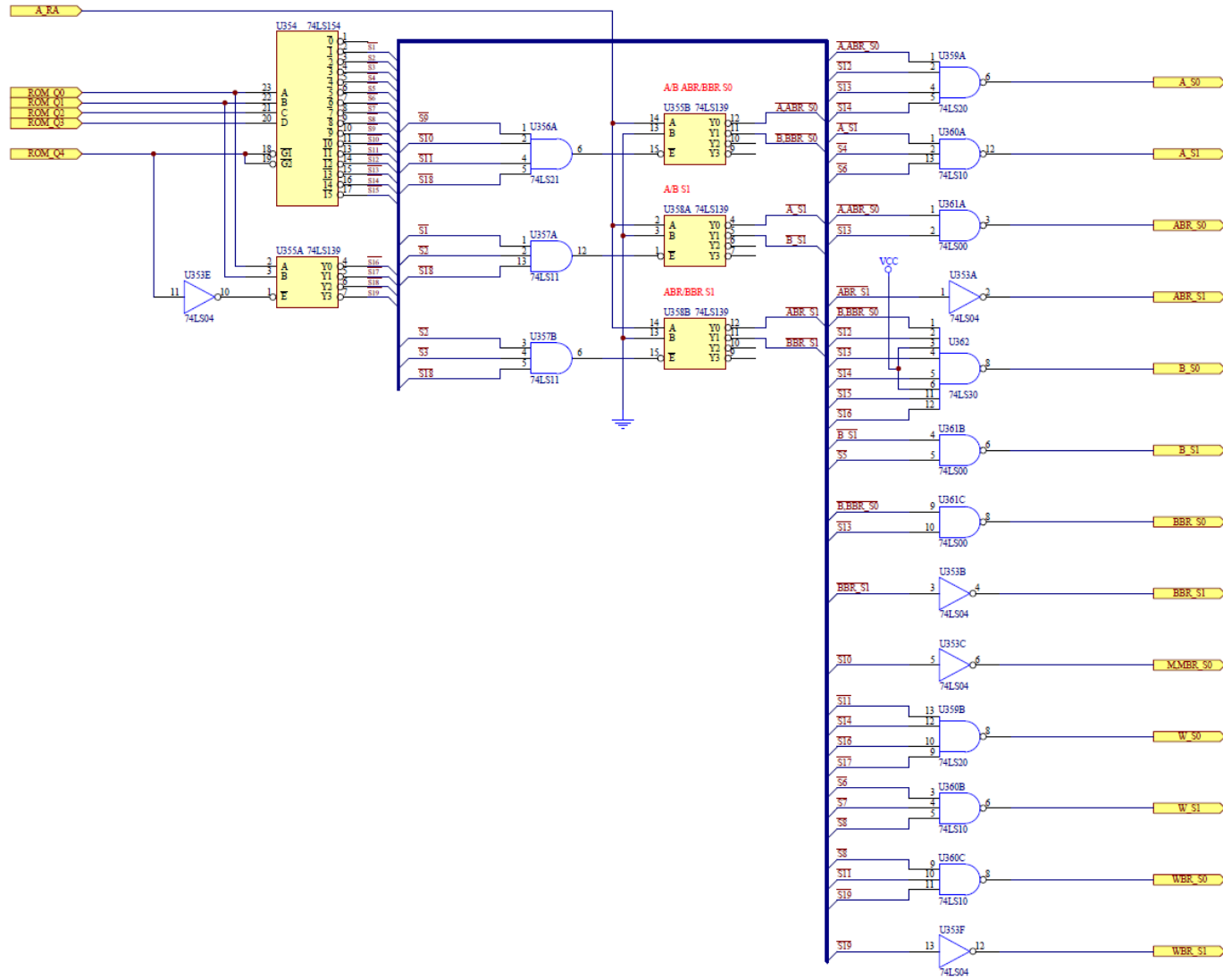


# 22. STEROWNIK ZESPOŁÓW LICZNIKÓW PUNKTÓW DZIESIĘTNYCH

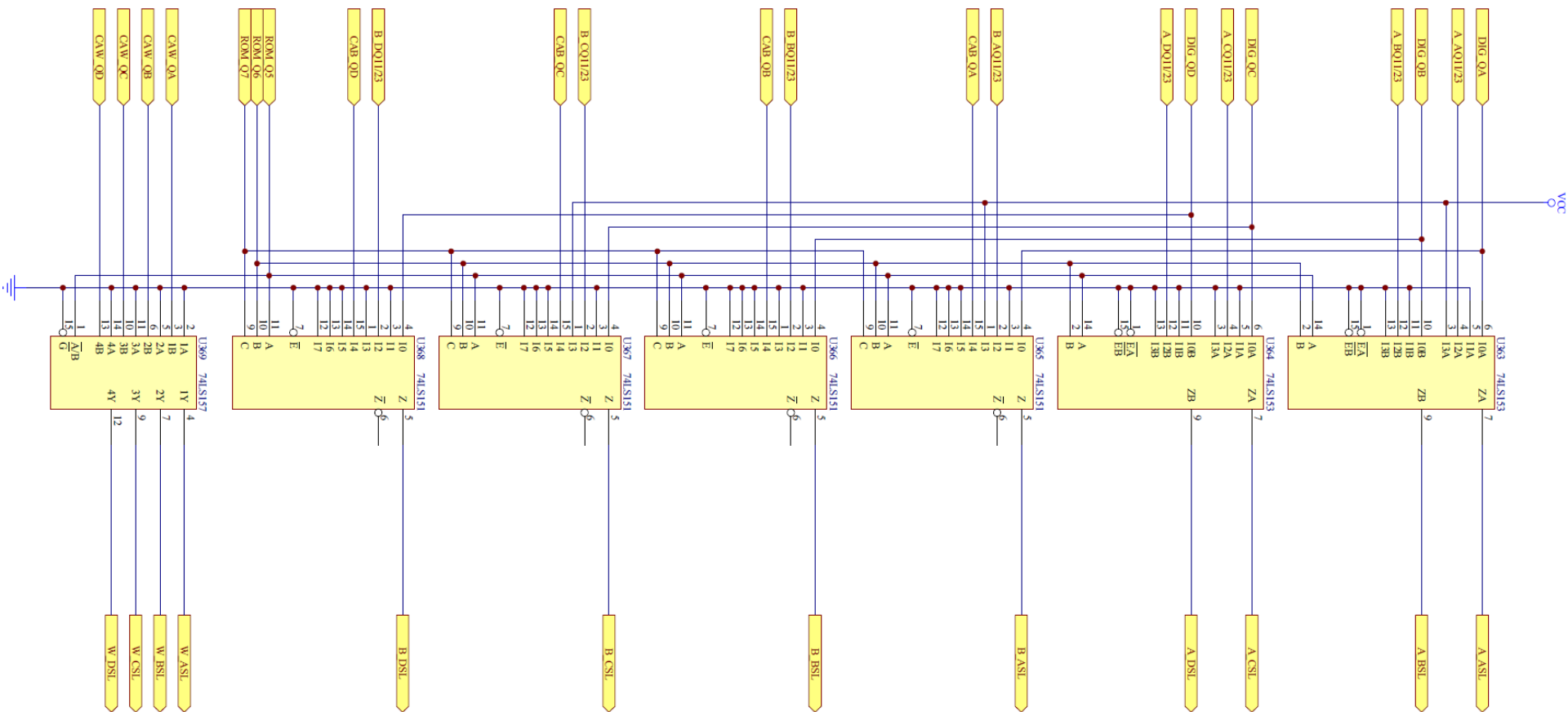




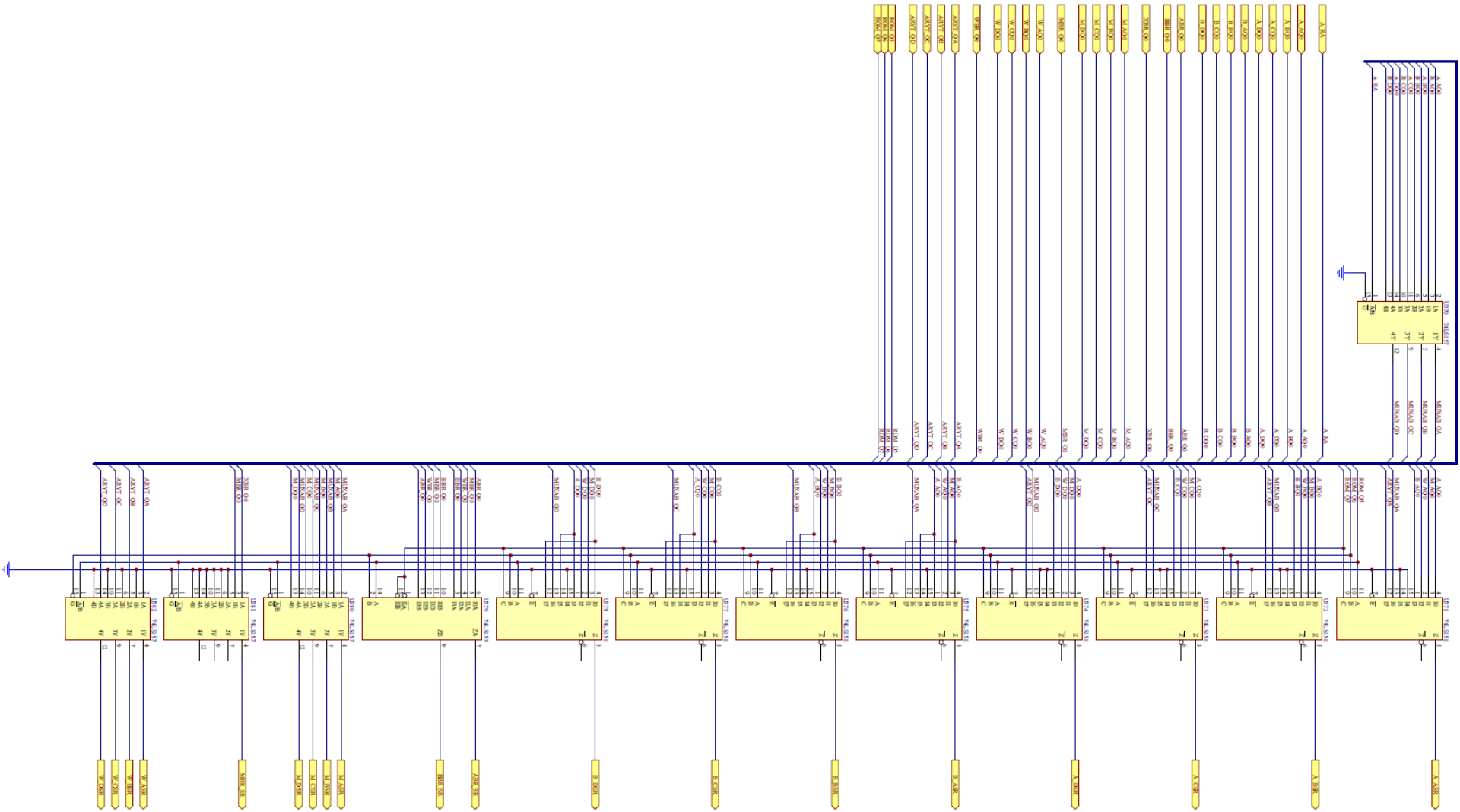
# 24. STEROWNIK USTAWIANIA TRYBÓW PRACY DLA ZESPOŁU REJESTRÓW



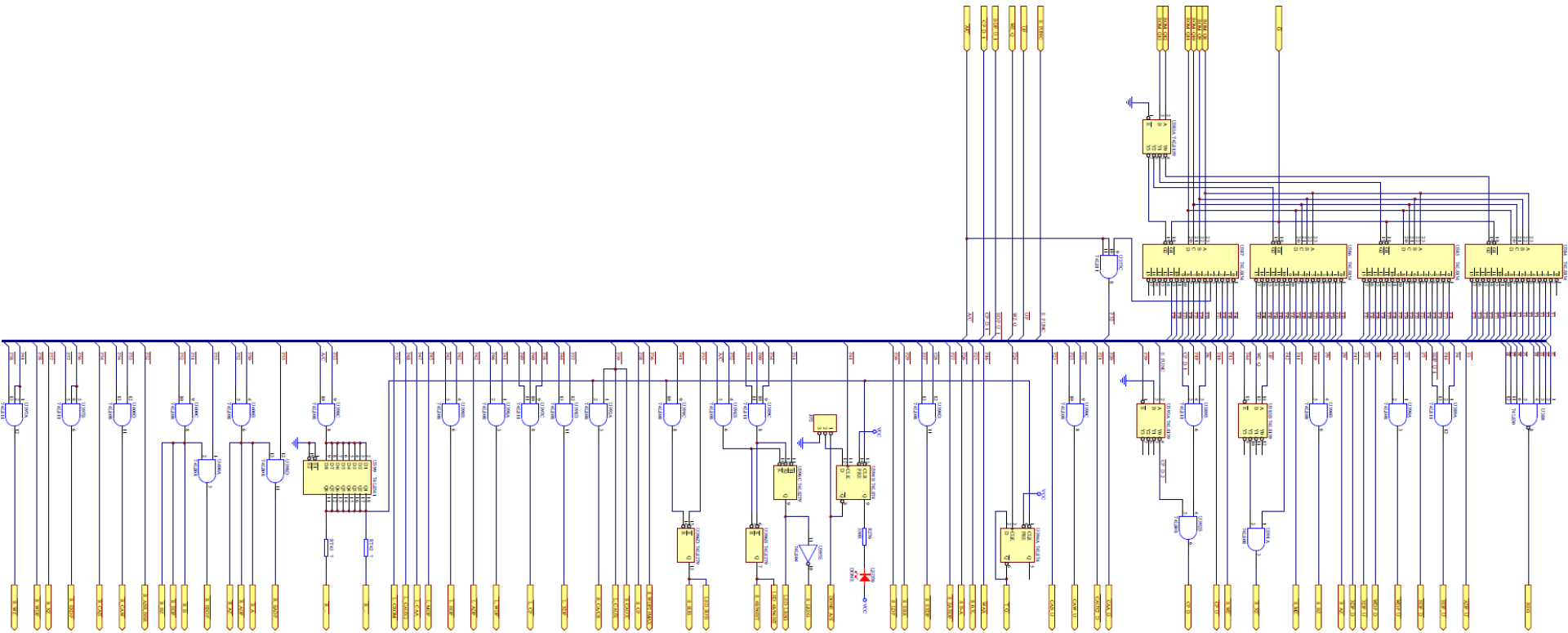
# 25. MULTIPLESERY OBSŁUGUJĄCE DANE DLA PRZESUWU W LEWO W ZESPOŁACH REJSTRÓW



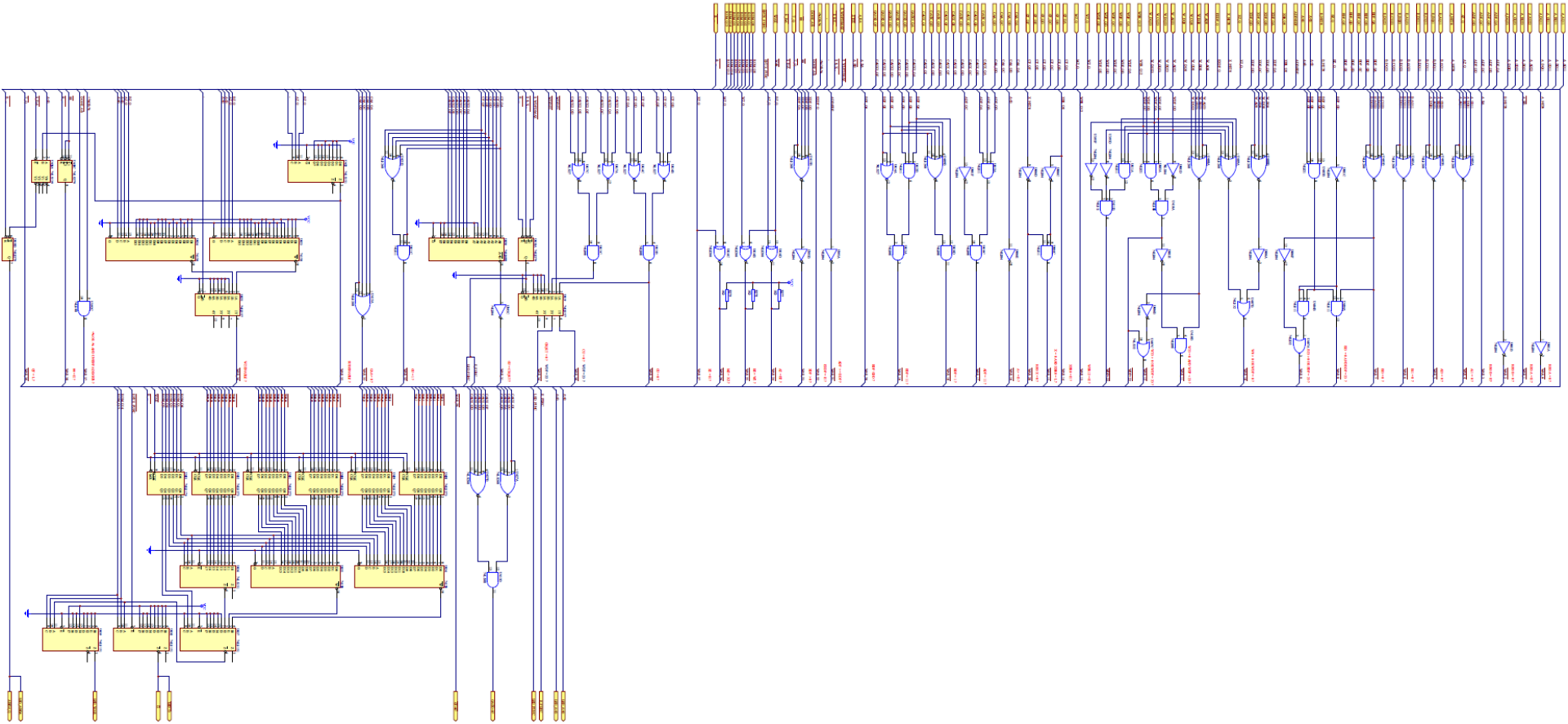
# 26. MULTIPLESERY OBSŁUGUJĄCE DANE DLA PRZESUWU W PRAWO W ZESPOŁACH REJESTRÓW



# 27. DEKODER FUNKCJI WYKONAWCZYCH



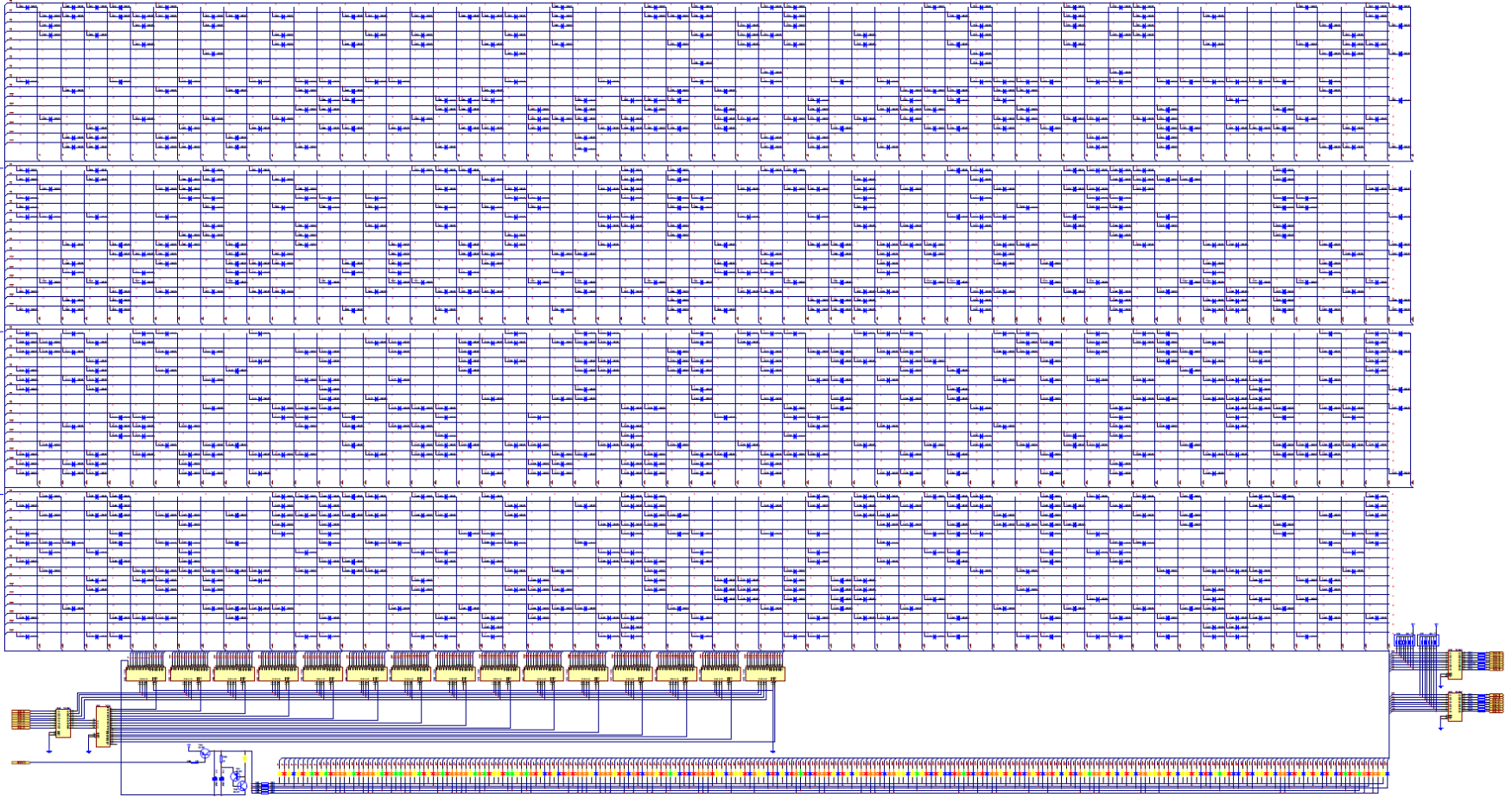
# 28. SELEKTOR FUNKCJI WARUNKOWYCH





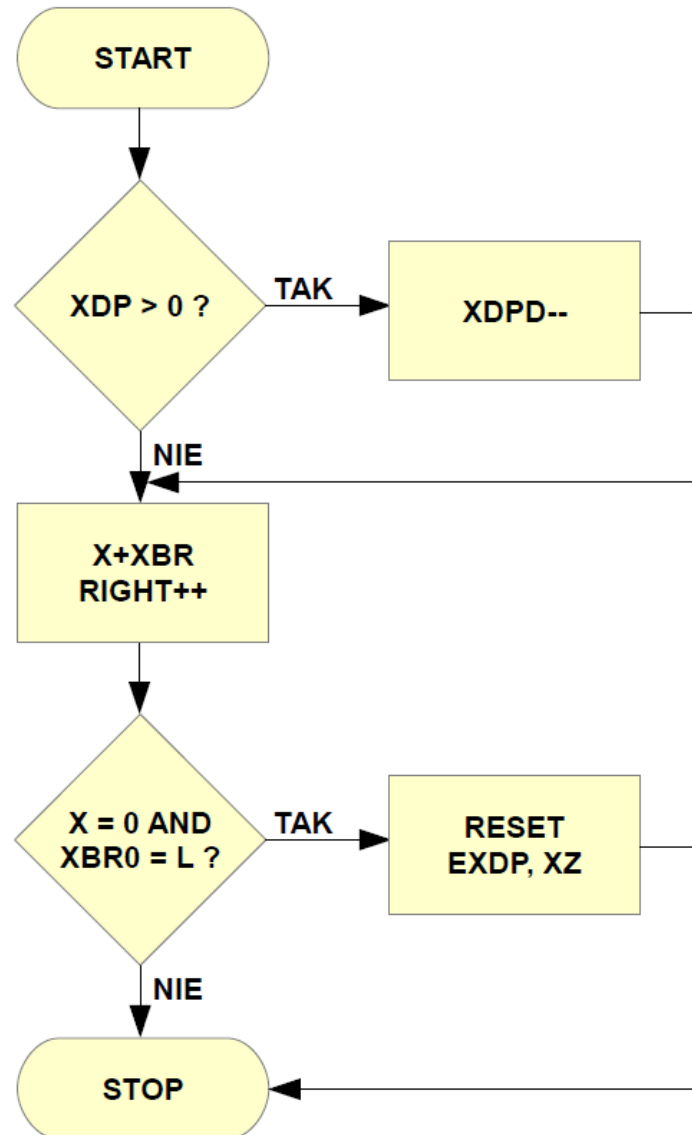


# 30.3840 BITOWA PAMIĘĆ ROM





# 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.



**(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 funkcje: **R\_EXDP** i **R\_XZ**, które załączają ewentualną blokadę licznikowi aktywnego punktu dziesiątego 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

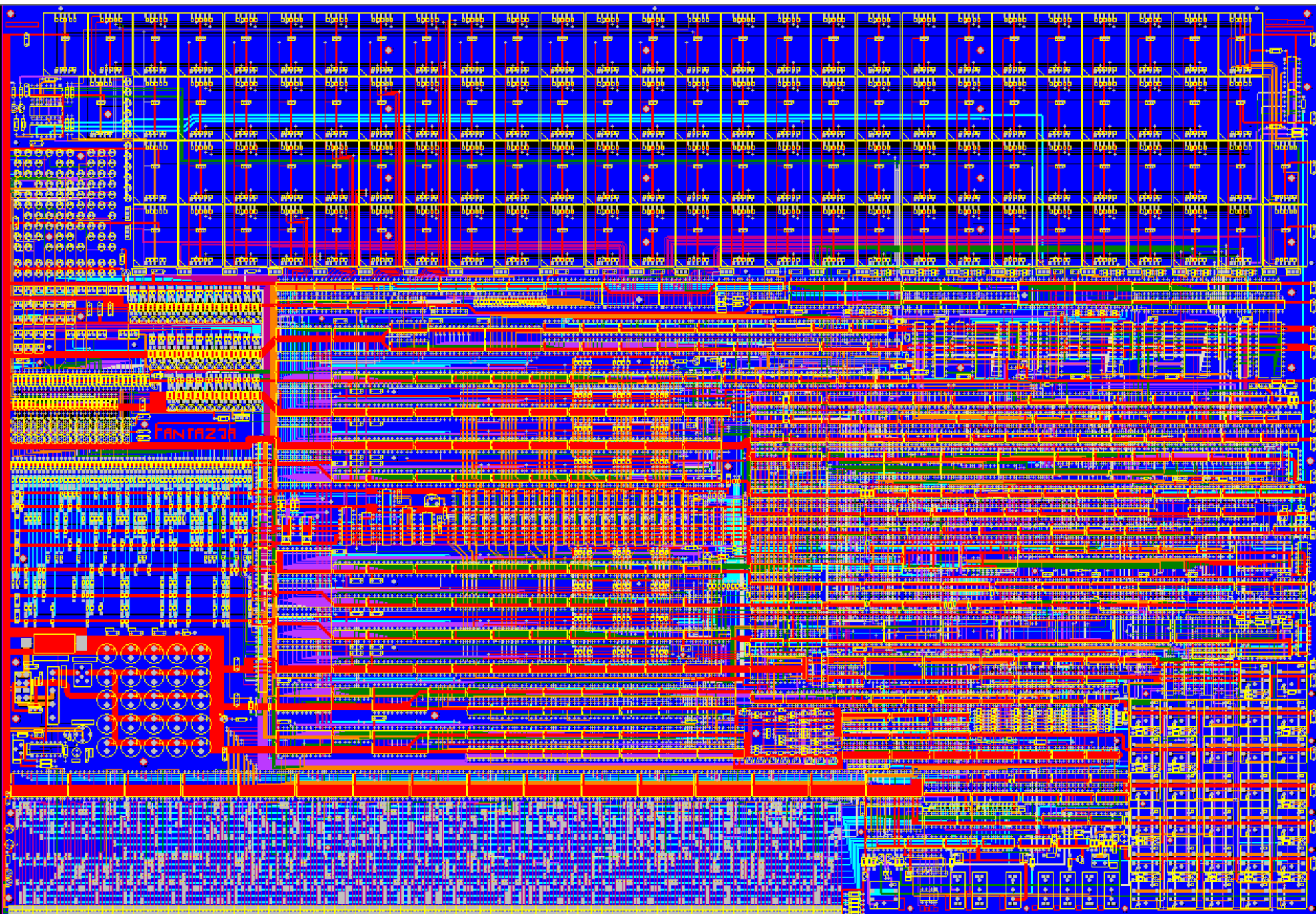
**6**

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**

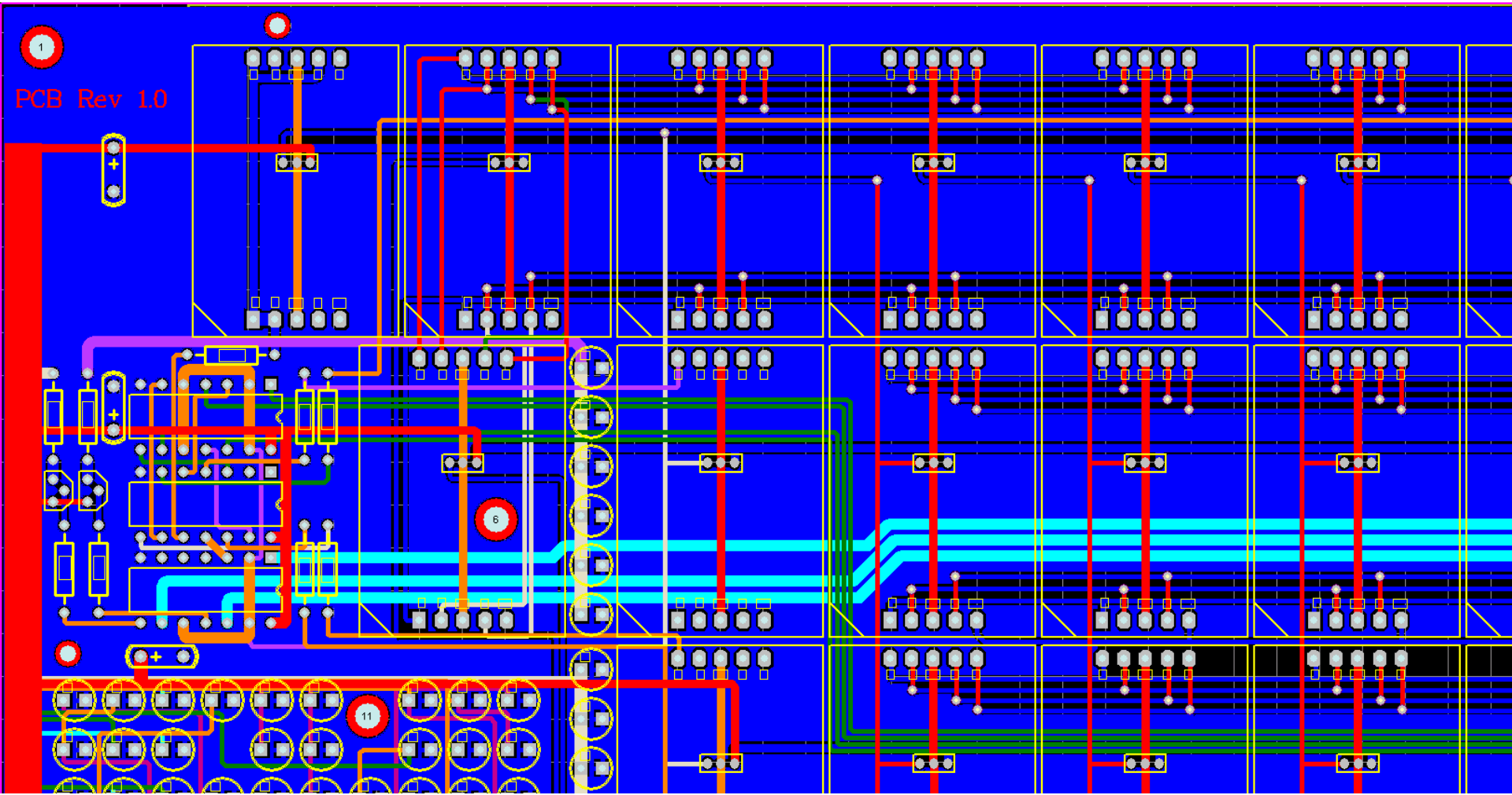
Instrukcja ta aktywuje funkcję **R\_...**, która zakańcza algorytm (następuje resetowanie obwodów, które biorą udział w przetwarzaniu algorytmów, zostaje również ustawiony adres pamięci ROM na pozycję początkową).

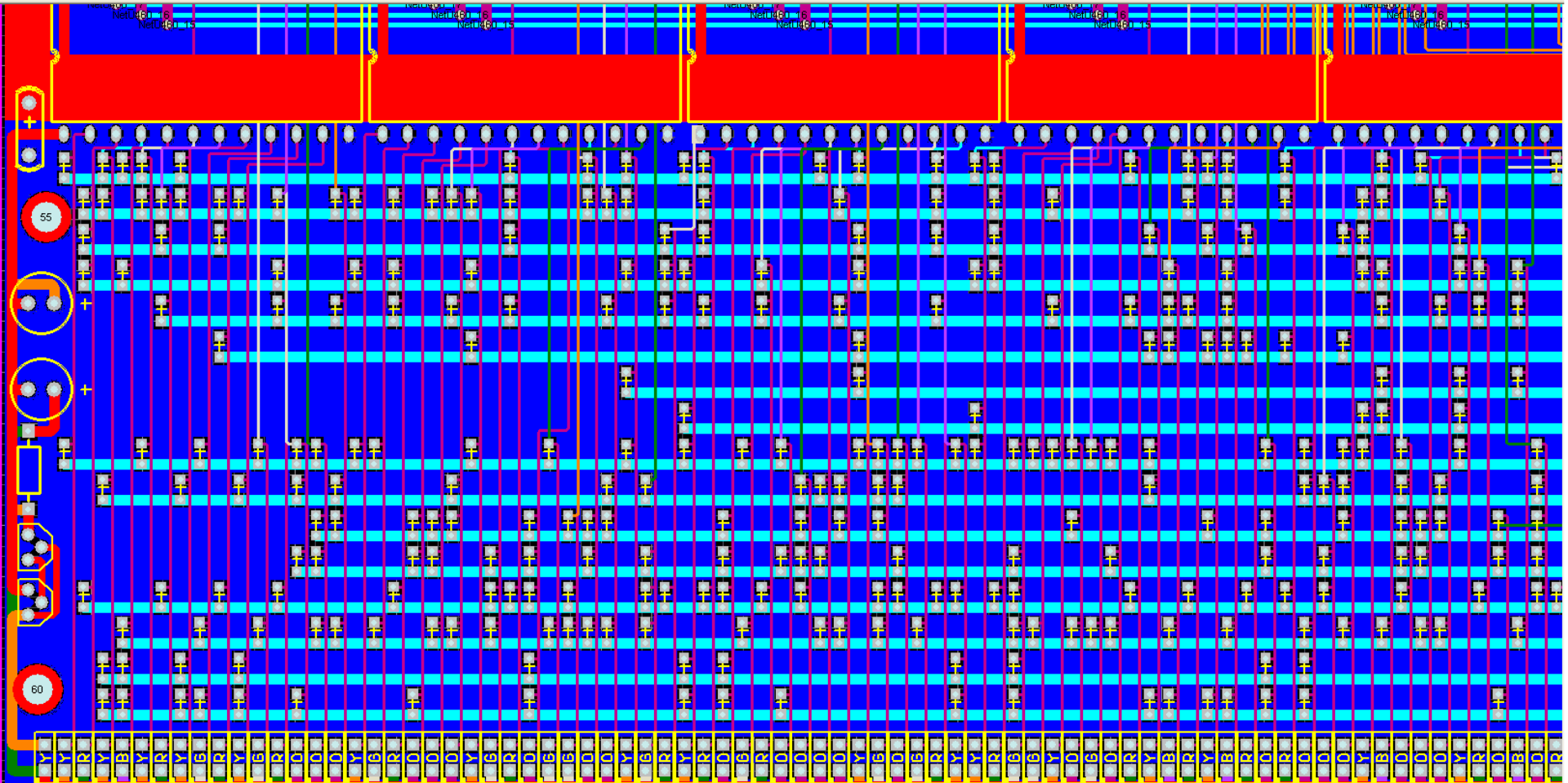


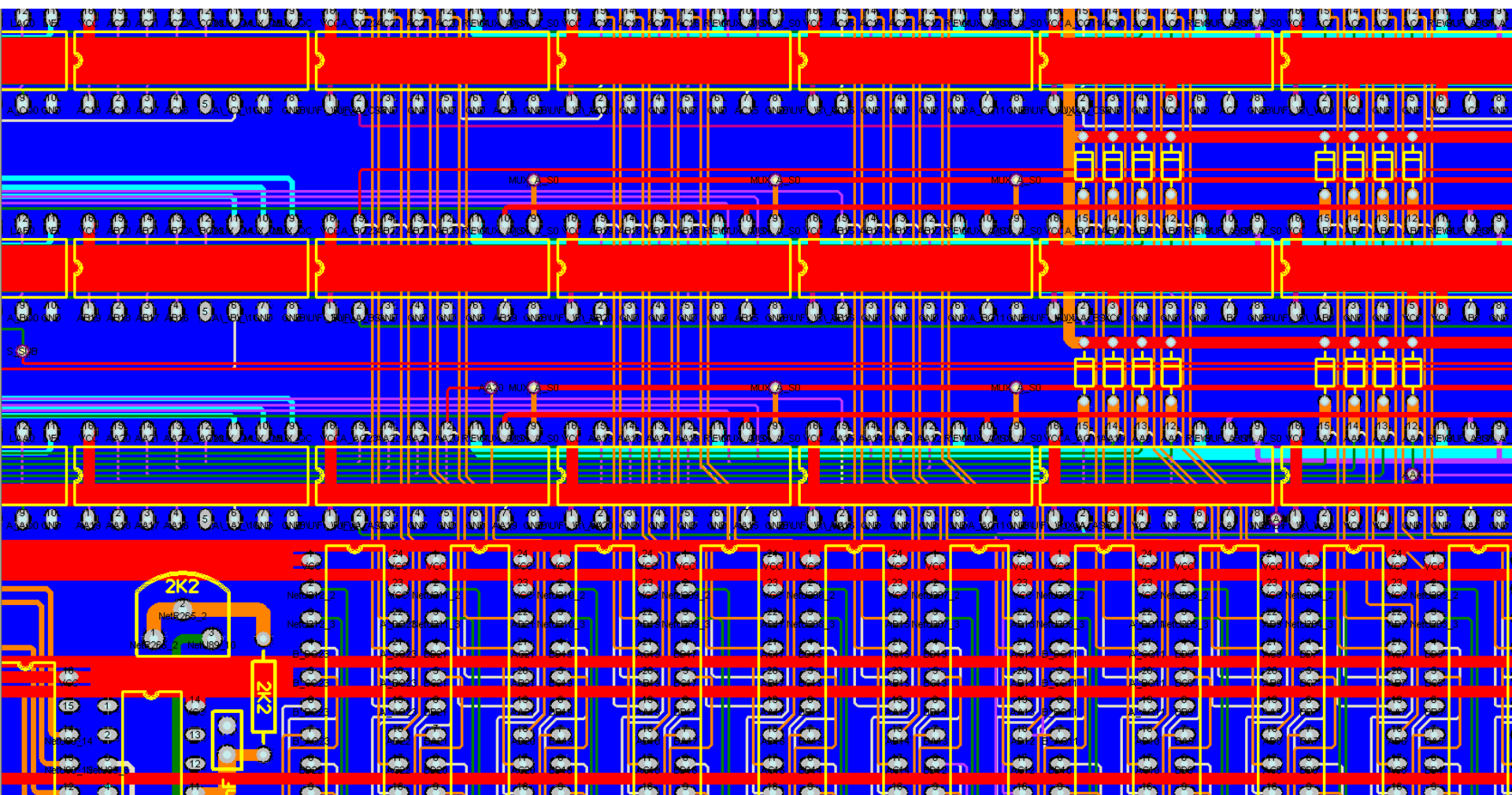


1

PCB Rev 1.0









**NOWY CEL**

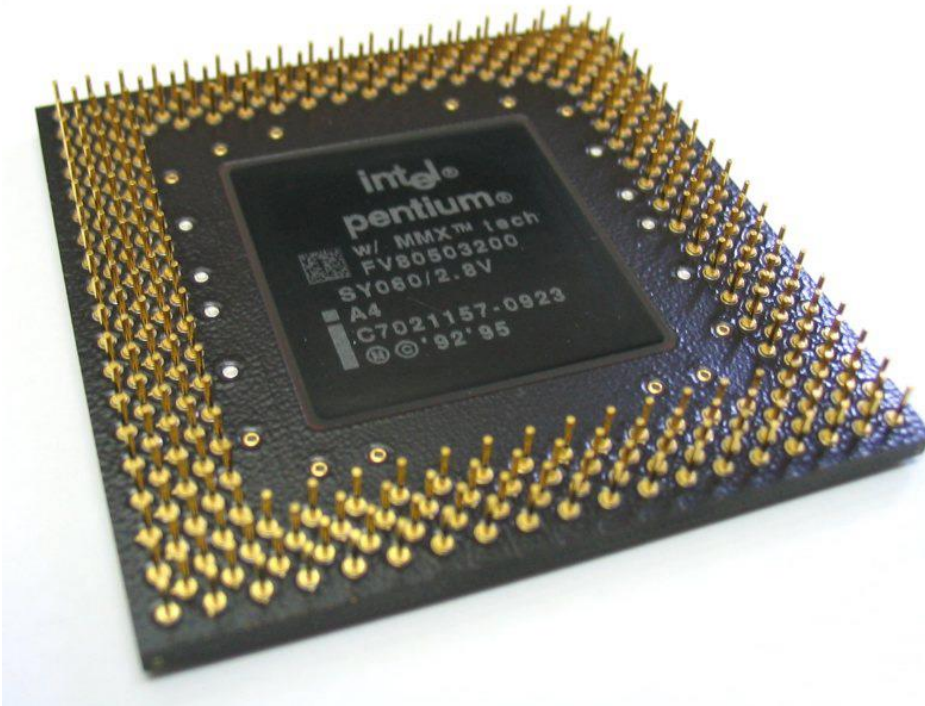
**KOMPUTER NA UKŁADACH TTL**

# NOWY CEL KOMPUTER NA UKŁADACH TTL



# NOWY CEL

# KOMPUTER NA UKŁADACH TTL



**intel.**

**MMX™ Technology Overview**

**MMX™ Instruction Set Summary**

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

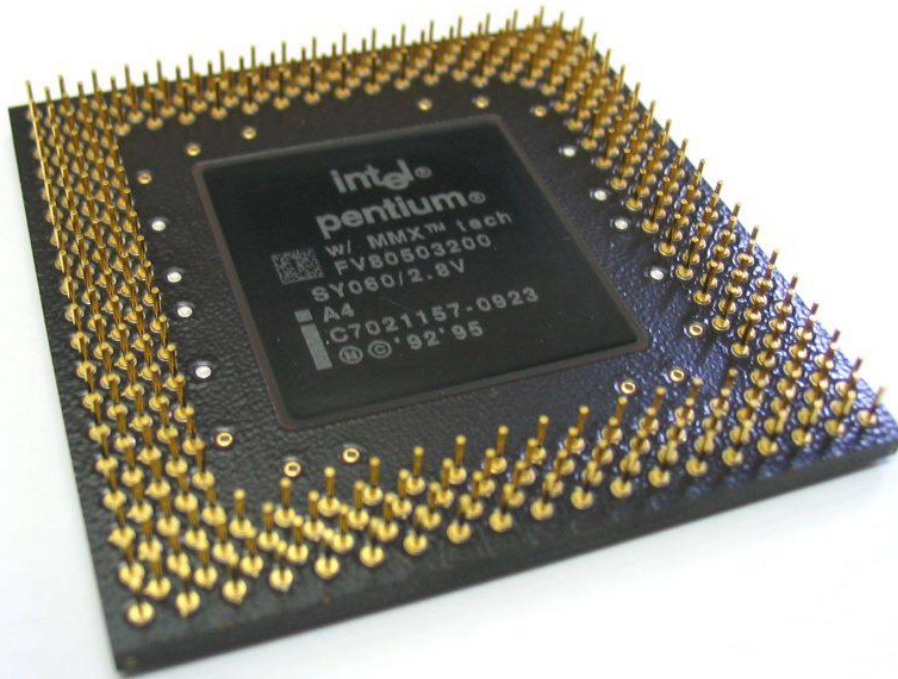
If an instruction supports multiple data types—byte (B), word (W), doubleword (D), or quadword (Q), the datatypes are listed in brackets. Only one data type may be chosen for a given instruction. For example, the base mnemonic **PADD** (packed add) has the following variations: **PADDB**, **PADDW**,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

Category	Mnemonic	Number of Operands	Description
Arithmetic	<b>PADD{B,W,D}</b>	2	Add with wrap-around on {byte, word, doubleword}
	<b>PADDQ{B,W}</b>	2	Add signed with saturation on {byte, word}
	<b>PADCQB{B,W}</b>	2	Add unsigned with saturation on {byte, word}
	<b>PADDB{B,W,D}</b>	2	Subtract signed with wrap-around on {byte, word}
	<b>PADSB{B,W}</b>	2	Subtract signed with saturation on {byte, word}
	<b>PADSQB{B,W}</b>	2	Subtract unsigned with saturation on {byte, word}
Comparison	<b>PCMPQB{B,W,D}</b>	2	Compare {byte, word, doubleword} for equality
Conversion	<b>PACQ{B,W,D}</b>	1	Packed multiply high on words
	<b>PACQTB{B,W,D}</b>	1	Packed multiply low on words
	<b>PACSS{B,W,D}</b>	2	Packed compare for equality and add resulting pairs
Logical	<b>PUNPCKQB{B,W,D,Q}</b>	2	Unpack {byte, word, doubleword} into words
	<b>PUNPCKWB{B,W,D,Q}</b>	2	Unpack {word, doubleword} into words
	<b>PUNPCKQB{B,W,D,Q}</b>	2	Unpack {byte, word, doubleword} into words
Shift	<b>PSRL{B,W,D,Q}</b>	1	Shift right logical {word, doubleword, quadword} by amount specified in ALU register or by immediate value
	<b>PSRLQB{B,W,D,Q}</b>	1	Shift right logical {word, doubleword, quadword} by amount specified in ALU register or by immediate value
	<b>PSRLQ{B,W,D,Q}</b>	1	Shift right logical {word, doubleword, quadword} by amount specified in ALU register or by immediate value
Data Transfer	<b>PSHU{B,W,D,Q}</b>	1	Shift right packed {word, doubleword, quadword} by amount specified in ALU register or by immediate value
	<b>PSHUQB{B,W,D,Q}</b>	1	Shift right packed {word, doubleword, quadword} by amount specified in ALU register or by immediate value
FP & ALU State Register	<b>EMMS</b>	1	Empty ALU state



# NOWY CEL

# KOMPUTER NA UKŁADACH TTL



intel.

MMX™ Technology Overview

MMX™ Technology Overview

### Instruction Examples

The following section will describe briefly five examples of MMX instructions. For illustration, the data type shown in this section will be the 16-bit word data type; most of these operations also exist for 8-bit or 32-bit packed data types.

The following example shows a packed add word with wrap around. It performs four additions of the eight, 16-bit elements, with each addition independent of the others and in parallel. In this case, the right-most result exceeds the maximum value representable in 16-bits—thus it wraps-around. This is the way regular IA arithmetic behaves.  $FFFFh + 8000h$  would be a 17 bit result. The 17th bit is lost because of wrap around, so the result is  $7FFFh$ .

a3	a2	a1	FFFFh
+	+	+	+
b3	b2	b1	8000h
a3+b3	a2+b2	a1+b1	7FFFh

PADDQW: Wrap-around Add

The following example is for a packed add word with unsigned saturation. This example uses the same data values from before. The right-most add generates a result that does not fit into 16 bits; consequently, in this case saturation occurs. Saturation means that if addition results in overflow or subtraction results in underflow, the result is clamped to the largest or the smallest value representable. For an unsigned, 16-bit word, the largest and the smallest representable values are  $FFFFh$  and  $0x0000$ . For a signed word the largest and the smallest representable values are  $7FFFh$  and  $0x8000$ . This is important for pixel calculations where this would prevent a wrap-around add from causing a black pixel to suddenly turn white while, for example, doing a 3D graphics Gouraud shading loop.

a3	a2	a1	FFFFh
+	+	+	+
b3	b2	b1	8000h
a3+b3	a2+b2	a1+b1	FFFFh

PADDUSW: Saturating Arithmetic

The specific instruction here is Packed Add Unsigned Saturation Word (PADDUSW). A complete set of ADD operations exists for signed and unsigned cases. The number  $FFFFh$ , treated as unsigned (65,535 decimal), is added to  $0x8000$  unsigned (32,768), and the result saturates to  $FFFFh$  - the largest representable unsigned 16-bit value.

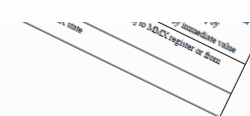
There is no "saturation mode bit" as a new mode bit would require a change to the operating system. Separate instructions are used to generate wrap-around and saturating results.

The next example shows the key instruction used for multiply-accumulate operations, which are fundamental to many signal processing algorithms like vector-dot-products, matrix multiplies, FIR and IIR filters, FFT's, DCT's etc. This instruction is the packed multiply-add (PMADD).

intel.

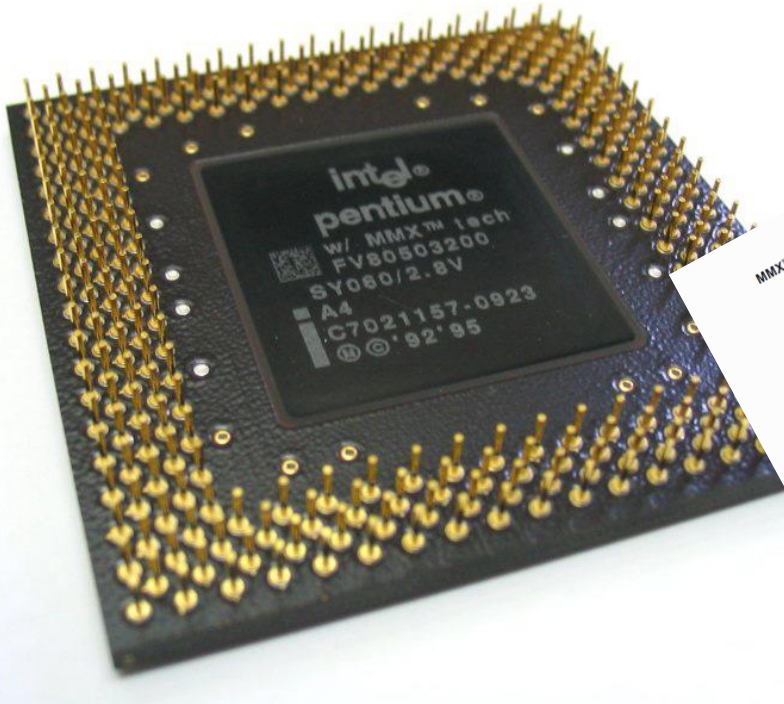
rounded by function word (DW), or be chosen for a following 16 each base

Data  
FP &  
State High



# NOWY CEL

# KOMPUTER NA UKŁADACH TTL



MMX™ Technology Overview

a3	a2	a1	a0
*	*	*	*
b3	b2	b1	b0
<b>a3*b3+a2*b2+a1*b1+a0*b0</b>			

**PMADDQW:** 16b × 16b → 32b Multiply-Add

The PMADD instruction starts from a 16-bit, packed data type and generates a 32-bit packed, data type result. It multiplies all the corresponding elements generating four 32-bit results, and adds the two products on the left together for one result and the two products on the right together for the other result. To complete a multiply-accumulate operation, the results would then be added to another register which is used as the accumulator.

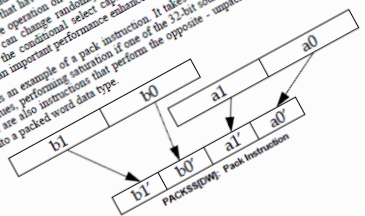
The following example is a packed parallel compare. This example compares four pairs of 16-bit words. It creates a result of true (FFFFh) or false (0000h). This result is a packed mask of ones for each true condition, or zeros for each false condition. The following example shows an example of a compare - greater than - on packed word data. There are no new condition code flags, nor are any existing IA condition code flags affected by this instruction.

23	16	34
gt?	gt?	gt?
31	7	67
0000h	FFFFh	0000h

**PCMPGTQW:** Parallel Compares

The packed compare result can be used as a mask to select elements from different inputs using a logical operation, eliminating the need for a branch or a set of branch instructions. The ability to do a conditional move instead of using branch instructions is an important performance enhancement in advanced processors that have deep pipelines and employ branch prediction. A branch based on the result of a compare operation on the incoming data is usually difficult to predict as incoming data in many cases can change randomly. Eliminating branches that are used to perform data selection by using the conditional select capability, together with the parallelism of the MMX instruction set, is an important performance enhancement feature of the MMX technology.

The following is an example of a pack instruction. It takes four 32-bit values and packs them into four 16-bit values performing a select operation if one of the 32-bit source values does not fit into a 16-bit result. There are also instructions that perform the opposite - unpack, for example, a packed byte data type into a packed word data type.



# 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 mispredictions, 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.

intel.

packed, data and address together for a to be added to

for pairs of 16-bit A mask of ones for shows an example of code flags, not are any

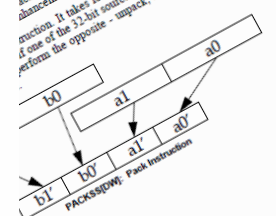
tion, the also exist

us of this

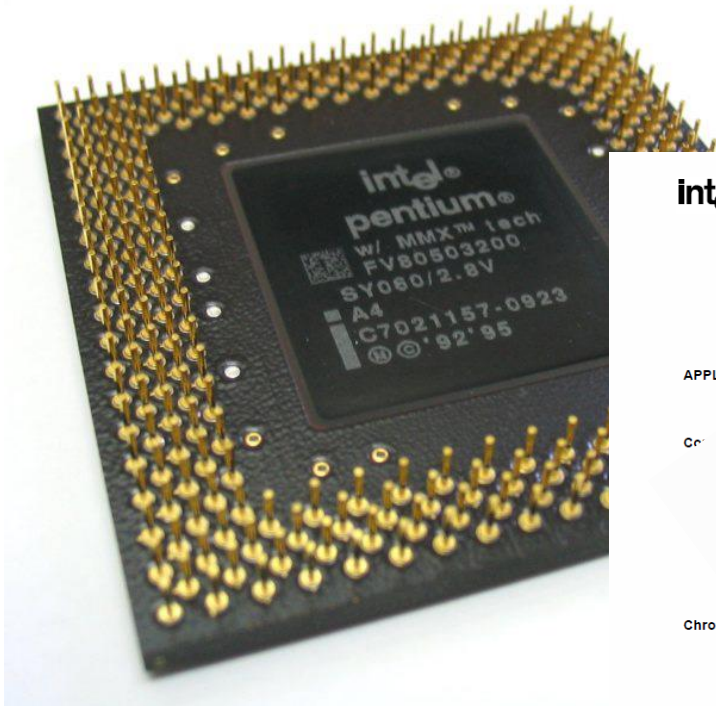


ject elements from different inputs using a set of branch instructions. The ability to do one to an supported performance enhancement A simple branch prediction A branch based on g data is usually difficult to predict as incoming matching branches that are used to perform data ability, together with the parallelism of the MMX enhancement feature of the MMX technology.

struction. It takes four 32-bit values and packs them into if one of the 32-bit source value does not fit into a 16-bit perform the opposite - unpack, for example, a packed byte



# NOWY CEL KOMPUTER NA UKŁADACH TTL



intel.

The pack and unpack instructions exist to facilitate these operations. These are especially important when performing calculations, as in image filtering, operations between filter coefficients together. These multi-ported data type of the pixels perform the calculations before storing them.

#### APPLICATION EXAMPLE

The

Color

#### Chroma Key

Most examples allow the potential.



First we'll take four pixels from the data below representing pixels. The compare instruction builds a mask of ones or all zeros representing the unwanted background and what we want to keep.

MMX™

#### Vector Dot Product

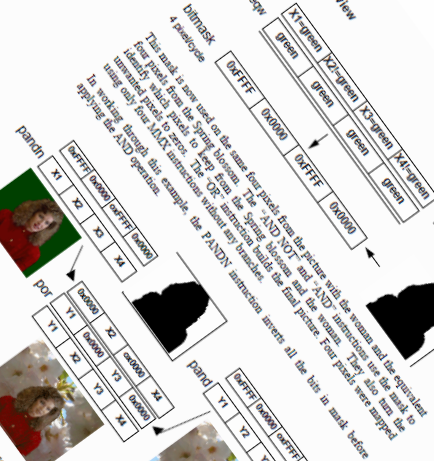
Without MMX technology, each pixel is processed separately and requires a conditional branch such as the previous example. Using MMX instructions, eight 8-bit pixels can be processed in parallel and no conditional branches are involved.

The vector dot product is one of the most basic algorithms used in signal processing of natural images. It is used to compare two vectors. The dot product is calculated by multiplying corresponding elements of two vectors and summing the results. In MMX, the dot product is calculated by multiplying corresponding elements of two vectors and summing the results. The dot product is calculated by multiplying corresponding elements of two vectors and summing the results.

Assuming that the previous example is used, the dot product is calculated by multiplying corresponding elements of two vectors and summing the results. The dot product is calculated by multiplying corresponding elements of two vectors and summing the results.

intel.

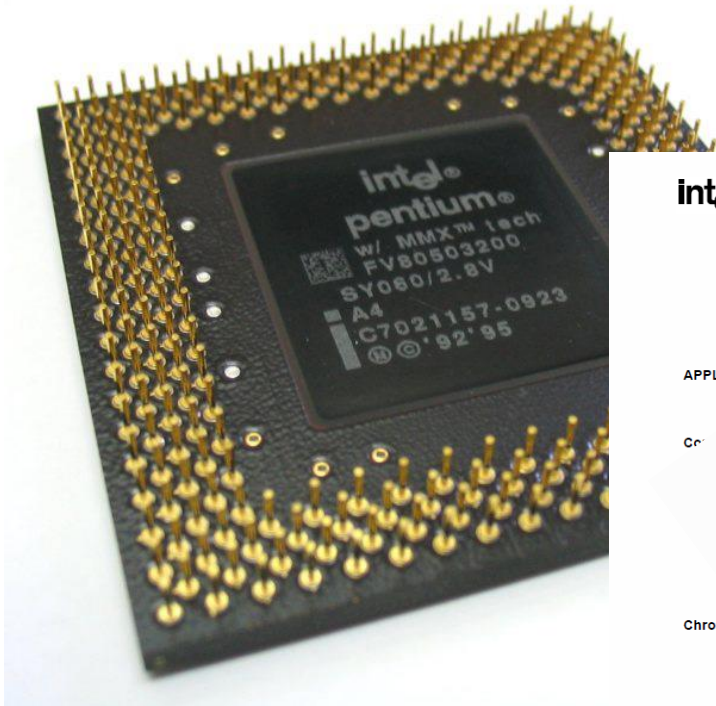
#### MMX™ Technology Overview



intel.

intel.

# NOWY CEL KOMPUTER NA UKŁADACH TTL



intel.

The pack and unpack instructions exist to facilitate these operations. These are especially important when performing calculations, as in image filtering, operations between filter coefficients and pixel values together. These multiplies the data type of the pixels. The MMX instructions perform the calculations on the pixels before storing them.

#### APPLICATION

The

#### Color

#### Chroma Key

Most examples allow the user to select a color to key out from a video frame. We'll use the example below to illustrate the potential.



First we'll take four pixels from the data below which represents pixels. The compare instruction builds a mask of ones or all zeros representing the background and what we want to keep.

MMX

#### 24-Bit Color

The MMX instructions set offers an application for 24-bit color. The MMX instructions set offers an application for 24-bit color. The MMX instructions set offers an application for 24-bit color. The MMX instructions set offers an application for 24-bit color.

Instruction	Number of Instructions	Number of Bytes
MMX instructions	12	96
MMX instructions	12	96
MMX instructions	12	96
MMX instructions	12	96
MMX instructions	12	96
MMX instructions	12	96
MMX instructions	12	96
MMX instructions	12	96
MMX instructions	12	96
MMX instructions	12	96
MMX instructions	12	96
MMX instructions	12	96

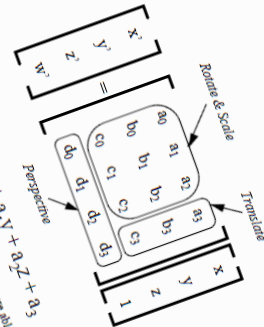
intel.

MMX™ Technology Overview

PCMPGE

4 p

MMX™ Technology Overview



intel.

# NOWY CEL KOMPUTER NA UKŁADACH TTL



intel.

The pack and unpack instructions exist to facilitate these operations. These are especially important when performing calculations, as in image filtering, operations between filter coefficients and pixel values together. These multiply the data type of the pixels. The processor performs the calculations on the pixels before storing them.

APPLICATION

Tr

Cr

Chroma Key

Most examples allow the potential.



First we'll take four pixels from the data below representing pixels. The compare instruction builds a mask of ones or all zeros representing the unwanted background and what we want to

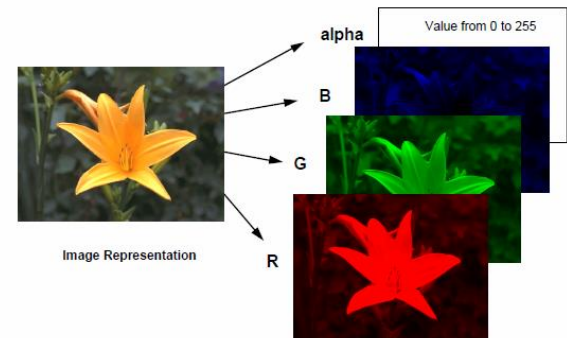
MMX

MMX Technology Overview

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.

$$\text{Result\_pixel} = \text{Flower\_pixel} * (\text{alpha}/255) + \text{Swan\_pixel} * [1 - (\text{alpha}/255)]$$

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.

# NOWY CEL

# KOMPUTER NA UKŁADACH TTL



intel.

The pack and unpack instructions - These are especially important for calculations, as in image operations between 8-bit values together. They perform operations on 8-bit data type of pixels.

APPLICATION

Cr

Chroma Key

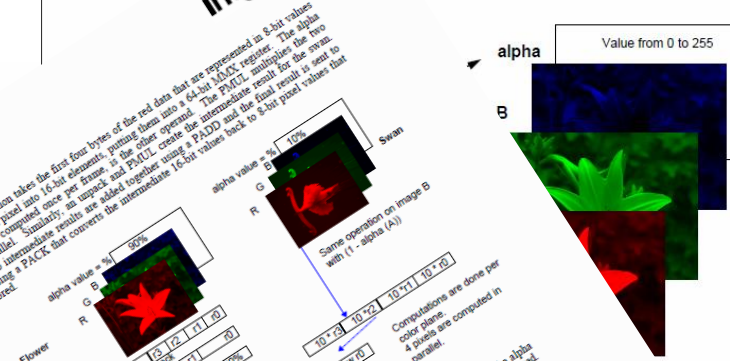
Mo. exam. We'll allow the potential.



First we'll take four pixels from the data below represents pixels. The compare instruction builds a mask of ones or all zeros representing the unwanted background and what we want to

## MMX™ Technology Overview

The unpack instruction takes the first four bytes of the red data that are represented as 8-bit values and unpacks each pixel into 16-bit elements, putting them into a 64-bit MMX register. The alpha value, which is computed once per frame, is the other operand. The PANDL multiplies the two vectors in parallel. Similarly, an unpack and PANDL create the immediate result for the alpha. Now the two intermediate results are added together using a PADL and the final result is sent to memory using a PACK that converts the intermediate 16-bit values back to 8-bit pixel values that can be stored.



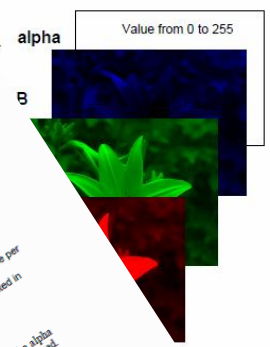
If these images use 640x480 resolution, and the dissolve technique uses all 255 steps of the alpha value, then 117 million PUNCK, and PANDL, and 38 million PADDD, and PACKs are used. Comparing instruction counts with and without MMX technology for this operation yields the following:

Operation	without MMX Technology	with MMX Technology	Number of MMX Instructions
Load	640*480*255*2	-	117 million
Unpack	640*480*255*2	-	117 million
Multiply	640*480*255*2	470 million	38 million
Add	-	230 million	38 million
Pack	640*480*255*2	230 million	38 million
Store	-	230 million	38 million
Total	-	1.4 billion	235 million

Almost 1 billion fewer instructions are used in this example. The dissolve technique, sometimes called combine, is one of several commonly used image compositing techniques used in multimedia applications and can be sped up substantially with MMX technology.

## MMX™ Technology Overview

are operations that can be performed on 24-bit color



this ve

ame are nes: one for ne swan will be .n and blue planes.

# NOWY CEL KOMPUTER NA UKŁADACH TTL



intel.

The pack and unpack instructions - These are especially important calculations, as in image operations between 32-bit values together. They perform a data type of 32-bit pixels.

MMX™ Technology Overview

The unpack instruction and unpacks each pixel value, which is converted in parallel. Now the two instructions can be stored.

#### APPLICATION

Th...

Cr...

#### Chroma Key

Most exam. We'll allow the potential.



First we'll take four pixels from the data below represents pixels. The compare instruction builds a mask of ones or all zeros representing the unwanted background and what we want to...

intel.

#### MMX™ Technology Overview

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

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)

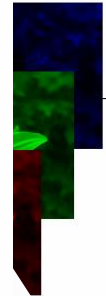
Refer to Intel's corporate website for the latest information on related documentation:

<http://www.intel.com/>

erview

n 24-bit color

from 0 to 255



this ve

4 image subly was

17

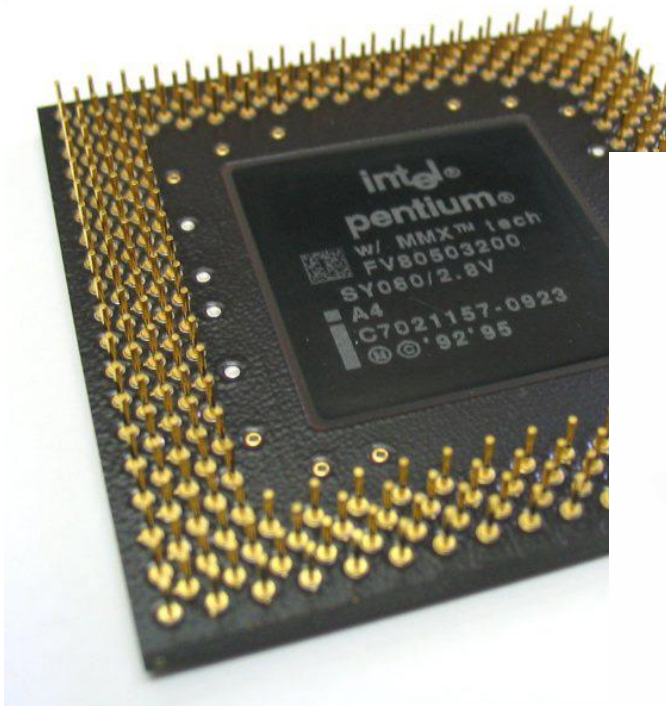
Millions of instructions are used in multimedia applications. The distinctive techniques are the compositing techniques used in multimedia applications. MMX technology.

same are... one for... the swan will be... and blue planes.

15



# NOWY CEL KOMPUTER NA UKŁADACH TTL



## Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 2 (2A, 2B, 2C & 2D):  
Instruction Set Reference, A-Z

**NOTE:** The Intel 64 and IA-32 Architectures Software Developer's Manual consists of four volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-Z*, Order Number 325383; *System Programming Guide*, Order Number 325384; *Model-Specific Registers*, Order Number 335592. Refer to all four volumes when evaluating your design needs.

### MMX™ Technology Overview

dest
mask
background
opacity
transparency
plane(A)
plane(B)

velopers that is similar to image compositing. Alpha through fog or smoke, allows a more realistic view be. In these examples, the alpha values wouldn't if the basic concept remains the same.

ultimedia and communication applications. MMX that can process data in parallel. MMX technology ms and application software.

o the PC platform and enables new applications and ligm in the industry with the PC as an improved us enabled with MMX technology will ramp in high lgy in multiple processor generations.

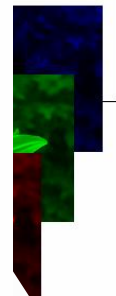
information on MMX technology:  
*Developers' Manual* (Order Number 243013)  
*Programmer's Reference Manual* (Order Number

information on related documentation:

### Overview

in 24-bit color

from 0 to 255



this  
ve

4 image  
ally was

Order Number: 325383-075US  
June 2021

17

ones or all zeros representing the Book  
unwanted background and what we want to

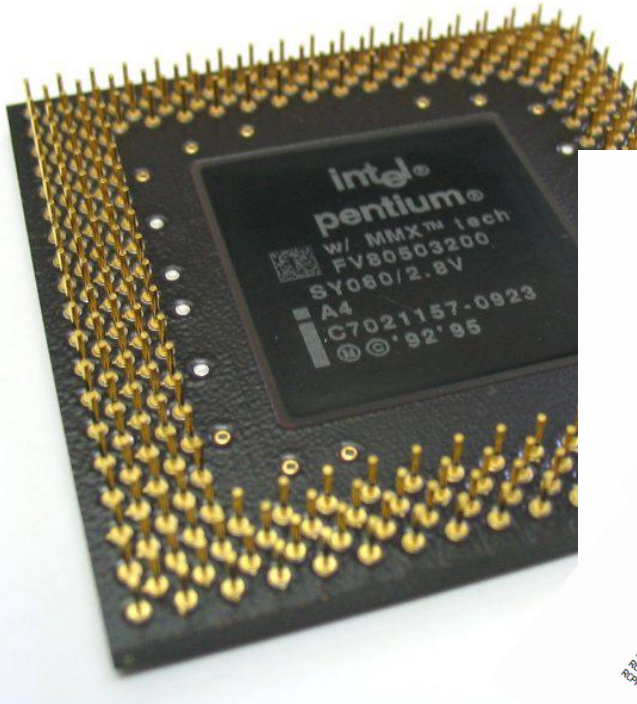
Align  
The disassembler technique, sometimes called co-  
compositing techniques used in multimedia applica-  
MMX technology.

same are  
planes: one for  
the swan will be  
and blue planes.

15

# NOWY CEL

# KOMPUTER NA UKŁADACH TTL



**intel**

**MMX™ Technology Overview**

dest
mask
background
opacity
transparency

phst(A)()  
1 - Alpha(B))

velopers that is similar to image compositing. Alpha through fog or smoke, allows a more realistic view be. In these examples, the alpha values wouldn't the basic concept remains the same.

communication applications. MMX data is parallel. MMX technology

enables new applications and PC as an improved will ramp in high

amber

from 0 to 255

this ve

4 image ally wan

Alm  
The disto... technique, sometimes called co... compositing techniques used in multimedia app... MMX technology.

ome are nes: one for the swan will be .n and blue planes.

15

Vol. 24 11

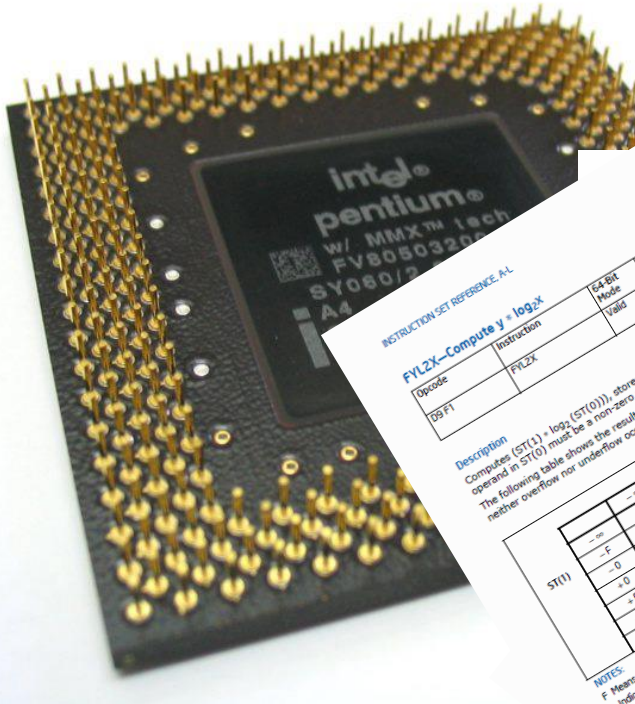
17

18

CONTENTS

TOPIC	PAGE
MMX™ Technology Overview	1-2
MMX™ Instruction Set	3-10
MMX™ Register File	11-15
MMX™ Instruction Formats	16-25
MMX™ Instruction Execution	26-35
MMX™ Instruction Execution Flow	36-45
MMX™ Instruction Execution Flow	46-55
MMX™ Instruction Execution Flow	56-65
MMX™ Instruction Execution Flow	66-75
MMX™ Instruction Execution Flow	76-85
MMX™ Instruction Execution Flow	86-95
MMX™ Instruction Execution Flow	96-105
MMX™ Instruction Execution Flow	106-115
MMX™ Instruction Execution Flow	116-125
MMX™ Instruction Execution Flow	126-135
MMX™ Instruction Execution Flow	136-145
MMX™ Instruction Execution Flow	146-155
MMX™ Instruction Execution Flow	156-165
MMX™ Instruction Execution Flow	166-175
MMX™ Instruction Execution Flow	176-185
MMX™ Instruction Execution Flow	186-195
MMX™ Instruction Execution Flow	196-205
MMX™ Instruction Execution Flow	206-215
MMX™ Instruction Execution Flow	216-225
MMX™ Instruction Execution Flow	226-235
MMX™ Instruction Execution Flow	236-245
MMX™ Instruction Execution Flow	246-255
MMX™ Instruction Execution Flow	256-265
MMX™ Instruction Execution Flow	266-275
MMX™ Instruction Execution Flow	276-285
MMX™ Instruction Execution Flow	286-295
MMX™ Instruction Execution Flow	296-305
MMX™ Instruction Execution Flow	306-315
MMX™ Instruction Execution Flow	316-325
MMX™ Instruction Execution Flow	326-335
MMX™ Instruction Execution Flow	336-345
MMX™ Instruction Execution Flow	346-355
MMX™ Instruction Execution Flow	356-365
MMX™ Instruction Execution Flow	366-375
MMX™ Instruction Execution Flow	376-385
MMX™ Instruction Execution Flow	386-395
MMX™ Instruction Execution Flow	396-405
MMX™ Instruction Execution Flow	406-415
MMX™ Instruction Execution Flow	416-425
MMX™ Instruction Execution Flow	426-435
MMX™ Instruction Execution Flow	436-445
MMX™ Instruction Execution Flow	446-455
MMX™ Instruction Execution Flow	456-465
MMX™ Instruction Execution Flow	466-475
MMX™ Instruction Execution Flow	476-485
MMX™ Instruction Execution Flow	486-495
MMX™ Instruction Execution Flow	496-505
MMX™ Instruction Execution Flow	506-515
MMX™ Instruction Execution Flow	516-525
MMX™ Instruction Execution Flow	526-535
MMX™ Instruction Execution Flow	536-545
MMX™ Instruction Execution Flow	546-555
MMX™ Instruction Execution Flow	556-565
MMX™ Instruction Execution Flow	566-575
MMX™ Instruction Execution Flow	576-585
MMX™ Instruction Execution Flow	586-595
MMX™ Instruction Execution Flow	596-605
MMX™ Instruction Execution Flow	606-615
MMX™ Instruction Execution Flow	616-625
MMX™ Instruction Execution Flow	626-635
MMX™ Instruction Execution Flow	636-645
MMX™ Instruction Execution Flow	646-655
MMX™ Instruction Execution Flow	656-665
MMX™ Instruction Execution Flow	666-675
MMX™ Instruction Execution Flow	676-685
MMX™ Instruction Execution Flow	686-695
MMX™ Instruction Execution Flow	696-705
MMX™ Instruction Execution Flow	706-715
MMX™ Instruction Execution Flow	716-725
MMX™ Instruction Execution Flow	726-735
MMX™ Instruction Execution Flow	736-745
MMX™ Instruction Execution Flow	746-755
MMX™ Instruction Execution Flow	756-765
MMX™ Instruction Execution Flow	766-775
MMX™ Instruction Execution Flow	776-785
MMX™ Instruction Execution Flow	786-795
MMX™ Instruction Execution Flow	796-805
MMX™ Instruction Execution Flow	806-815
MMX™ Instruction Execution Flow	816-825
MMX™ Instruction Execution Flow	826-835
MMX™ Instruction Execution Flow	836-845
MMX™ Instruction Execution Flow	846-855
MMX™ Instruction Execution Flow	856-865
MMX™ Instruction Execution Flow	866-875
MMX™ Instruction Execution Flow	876-885
MMX™ Instruction Execution Flow	886-895
MMX™ Instruction Execution Flow	896-905
MMX™ Instruction Execution Flow	906-915
MMX™ Instruction Execution Flow	916-925
MMX™ Instruction Execution Flow	926-935
MMX™ Instruction Execution Flow	936-945
MMX™ Instruction Execution Flow	946-955
MMX™ Instruction Execution Flow	956-965
MMX™ Instruction Execution Flow	966-975
MMX™ Instruction Execution Flow	976-985
MMX™ Instruction Execution Flow	986-995
MMX™ Instruction Execution Flow	996-1005

# NOWY CEL KOMPUTER NA UKŁADACH TTL



INSTRUCTION SET REFERENCE AL  
FYLZX—Compute  $y = \log_2 x$

OpCode	Instruction	64-bit Mode	Compat/ Legacy Mode	Description
D9 F1	FYLZX	Valid	Valid	Replace ST(1) with $ST(1) + \log_2 ST(0)$ and pop the register stack.

**Description**  
Computes  $ST(1) + \log_2 (ST(0))$  and stores the result in register ST(1), and pops the FPU register stack. The source operand in ST(0) must be a non-zero positive number. The following table shows the results obtained when taking the log of various classes of numbers, assuming that neither overflow nor underflow occurs.

**Table 3-48. FYLZX Results**

ST(0)	-F	-D	+F	+D	+F > 1	+∞	NaN
-∞	*	**	-∞	-∞	-∞	-∞	NaN
-F	*	**	-∞	-∞	-∞	-∞	NaN
-D	*	**	-∞	-∞	-∞	-∞	NaN
+∞	*	**	+∞	+∞	+∞	+∞	NaN
+F	*	**	+∞	+∞	+∞	+∞	NaN
+D	*	**	+∞	+∞	+∞	+∞	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

**NOTES:**  
 \* Means finite floating-point value.  
 \*\* Indicates floating-point invalid-operation (I/O) exception.  
 +∞ Indicates floating-point zero-divide (FZ) exception.  
 \*\* Indicates floating-point zero-divide (FZ) exception.

If the divide-by-zero exception is masked and register ST(0) contains +∞, the instruction returns +∞ with a sign that is the opposite of the sign of the source operand in register ST(1).  
 The FYLZX instruction is designed with a built-in multiplication to optimize the calculation of logarithms with an arbitrary positive base (b):  
 $\log_2 x = (\log_b x) / \log_2 b$   
 This instruction's operation is the same in non-64-bit modes and 64-bit mode.

**Operation**  
 $ST(1) = ST(1) + \log_2 ST(0)$   
 PopRegisterStack  
**FPU Flags Affected**  
 C1  
 C0, C2, C3



## MMX™ Technology Overview

Direct (RAW)
Background
Capacity
Responsivity
Plan(A/B)
1 - Alpha(B)

velopers that is similar to image compositing. Alpha through fog or smoke, allows a more realistic view be. In these examples, the alpha values wouldn't the basic concept remains the same.

communication applications. MMX with its parallel MMX technology software. MMX technology enables new applications and PC as an improved PC will ramp in high

**CONTENTS**

Page	Page
1-1	1-1
1-2	1-2
1-3	1-3
1-4	1-4
1-5	1-5
1-6	1-6
1-7	1-7
1-8	1-8
1-9	1-9
1-10	1-10
1-11	1-11
1-12	1-12
1-13	1-13
1-14	1-14
1-15	1-15
1-16	1-16
1-17	1-17
1-18	1-18
1-19	1-19
1-20	1-20
1-21	1-21
1-22	1-22
1-23	1-23
1-24	1-24
1-25	1-25
1-26	1-26
1-27	1-27
1-28	1-28
1-29	1-29
1-30	1-30
1-31	1-31
1-32	1-32
1-33	1-33
1-34	1-34
1-35	1-35
1-36	1-36
1-37	1-37
1-38	1-38
1-39	1-39
1-40	1-40
1-41	1-41
1-42	1-42
1-43	1-43
1-44	1-44
1-45	1-45
1-46	1-46
1-47	1-47
1-48	1-48
1-49	1-49
1-50	1-50
1-51	1-51
1-52	1-52
1-53	1-53
1-54	1-54
1-55	1-55
1-56	1-56
1-57	1-57
1-58	1-58
1-59	1-59
1-60	1-60
1-61	1-61
1-62	1-62
1-63	1-63
1-64	1-64
1-65	1-65
1-66	1-66
1-67	1-67
1-68	1-68
1-69	1-69
1-70	1-70
1-71	1-71
1-72	1-72
1-73	1-73
1-74	1-74
1-75	1-75
1-76	1-76
1-77	1-77
1-78	1-78
1-79	1-79
1-80	1-80
1-81	1-81
1-82	1-82
1-83	1-83
1-84	1-84
1-85	1-85
1-86	1-86
1-87	1-87
1-88	1-88
1-89	1-89
1-90	1-90
1-91	1-91
1-92	1-92
1-93	1-93
1-94	1-94
1-95	1-95
1-96	1-96
1-97	1-97
1-98	1-98
1-99	1-99
1-100	1-100

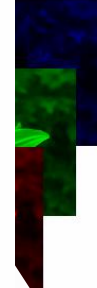
FYLZX—Compute  $y = \log_2 x$

**AIM:**  
 The disto... technique, sometimes called co-compositing techniques used in multimedia applications.

erview

n 24-bit color

from 0 to 255



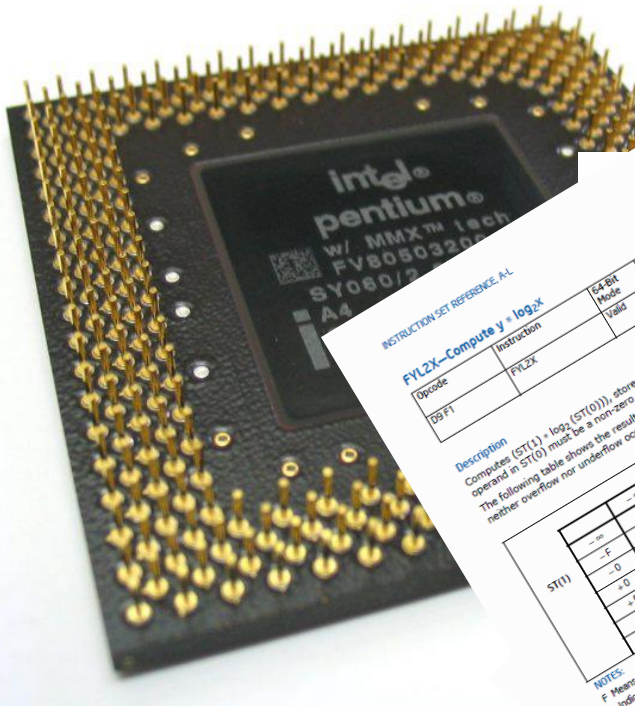
this ve

4 image ally wan

ame are nes: one for the swan will be and blue planes.

# NOWY CEL

# KOMPUTER NA UKŁADACH TTL



INSTRUCTION SET REFERENCE, A-L  
**FYLZX—Compute  $y = \log_2 x$**

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
D9 F1	FYLZX	Valid	Valid	Replace ST(1) with $\log_2(\text{ST}(1) + \log_2 \text{ST}(0))$ and pop the register stack.

**Description**

Computes  $\text{ST}(1) = \log_2(\text{ST}(0))$  and stores the result in register ST(1), and pops the FPU register stack. The operand in ST(0) must be a non-zero positive number. The following table shows the results obtained when taking the log of various classes of numbers, assuming neither overflow nor underflow occurs:

**Table 3-4B: FYLZX Results**

ST(0)	-∞	-F	-D	-∞	+∞	+F	+D	+∞
-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞
-F	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞
-D	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞
+∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞
+F	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞
+D	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

**NOTES:**  
 F Means finite floating-point value.  
 ∞ Indicates floating-point invalid-operation (I#A) exception.  
 -∞ Indicates floating-point zero-divide (F#Z) exception.  
 NaN Indicates floating-point zero-divide (F#Z) exception.

If the divide-by-zero exception is masked and register ST(0) contains the opposite of the sign of the source operand in register ST(1), the FYLZX instruction is designed with a built-in multiplication to  $\log_2 = \log_2(x^2) = 2 \log_2 x$ . This instruction's operation is the same in non-64-bit mode.

**Operation**  
 $\text{ST}(1) = \text{ST}(1) + \log_2 \text{ST}(0)$   
 PopRegisterStack  
**FPU Flags Affected**  
 C1 Set to 0 if stack underflow  
 C0, C2, C3 Set to 0 if result was rounded  
 Undefined.

**FXCH—Exchange Register Contents**

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
D9 C8+1	FXCH ST(i)	Valid	Valid	Exchange the contents of ST(0) and ST(i).
D9 C9	FXCH	Valid	Valid	Exchange the contents of ST(0) and ST(1).

**Description**

Exchanges the contents of registers ST(0) and ST(i). If no source operand is specified, the contents of ST(0) and ST(1) are exchanged.

This instruction provides a simple means of moving values in the FPU register stack to the top of the stack [ST(0)], so that they can be operated on by those floating-point instructions that can only operate on values in ST(0). For example, the following instruction sequence takes the square root of the third register from the top of the register stack:

```
FXCH ST(3);
FSQRT;
FXCH ST(3);
```

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

**Operation**

```
IF (Number-of-operands) is 1
THEN
  temp := ST(0);
  ST(0) := SRC;
  SRC := temp;
ELSE
  temp := ST(0);
  ST(0) := ST(1);
  ST(1) := temp;
```

**Ft**

**FPU Flags Affected**  
 C1 Set to 0.  
 C0, C2, C3 Undefined.

**Floating-Point Exceptions**  
 #IS Stack underflow occurred.

**Protected Mode Exceptions**  
 #NM CR0.EM[bit 2] or CR0.TS[bit 3] = 1.  
 #MF If there is a pending x87 FPU exception.  
 #UD If the LOCK prefix is used.

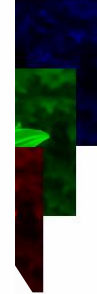
**Real-Address Mode Exceptions**  
 Same exceptions as in protected mode.

**Virtual-8086 Mode Exceptions**  
 Same exceptions as in protected mode.

Overview

24-bit color

from 0 to 255

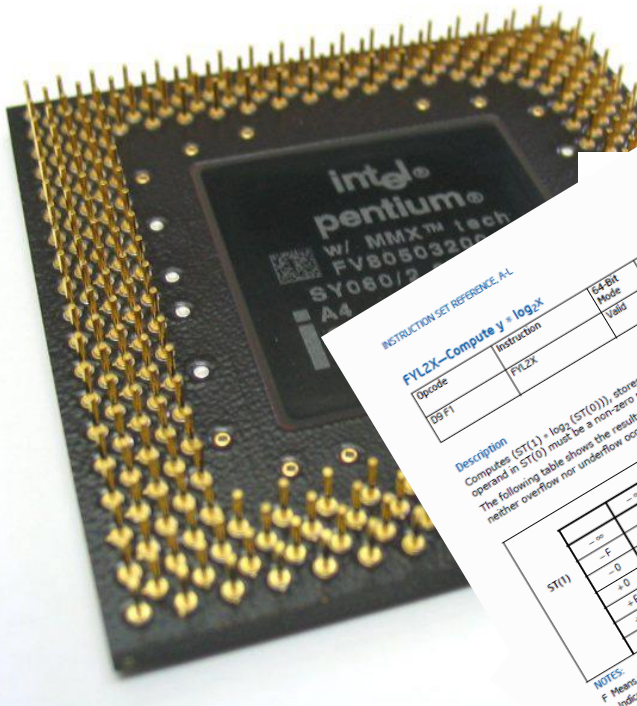


this

image  
 ally was

are  
 nes: one for  
 the swan will be  
 and blue planes.

# NOWY CEL KOMPUTER NA UKŁADACH TTL



INSTRUCTION SET REFERENCE, A-L

**FYLXZ—Compute  $y = \log_2 x$**

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode
D9 F1	FYLXZ	Valid	Valid

**Description**  
Computes  $(ST(1)) = \log_2 (ST(0))$  unless the operand in  $ST(0)$  must be a non-zero power of 2. The following table shows the results of neither overflow nor underflow occurs.

$ST(0)$	Result
$\infty$	$\infty$
$-\infty$	$-\infty$
$-F$	$-F$
$-D$	$-D$
$-F$	$-F$
$-D$	$-D$
$0$	$0$
$F$	$F$
$D$	$D$
$\infty$	$\infty$

**NOTES:**  
F Means finite  
D Indicates  
∞ Indicates

If the  
is to  
7

INSTRUCTION SET REFERENCE, A-L

## FSUBR/FSUBRP/FISUBR—Reverse Subtract

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
DB 75	FSUBR $m32fp$	Valid	Valid	Subtract $ST(0)$ from $m32fp$ and store result in $ST(0)$ .
DC 75	FSUBR $m64fp$	Valid	Valid	Subtract $ST(0)$ from $m64fp$ and store result in $ST(0)$ .
DB E8+	FSUBR $ST(0), ST(i)$	Valid	Valid	Subtract $ST(0)$ from $ST(i)$ and store result in $ST(0)$ .
DC E0+	FSUBR $ST(i), ST(0)$	Valid	Valid	Subtract $ST(i)$ from $ST(0)$ and store result in $ST(i)$ .
DE E0+	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 75	FISUBR $m32int$	Valid	Valid	Subtract $ST(0)$ from $m32int$ and store result in $ST(0)$ .
DE 75	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  $-\infty$  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  $\infty$ , the result is  $\infty$  of the expected sign. If both operands are  $\infty$  of the same sign, an invalid-operation exception is generated.

ology Overview

erview

ITION SET REFERENCE, A-L

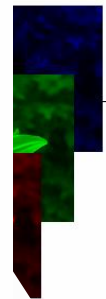
n 24-bit color

from 0 to 255

$ST(0)$  and  $ST(i)$ ,  
 $ST(0)$  and  $ST(1)$ .

ontents of  $ST(0)$  and

i of the stack [ $ST(0)$ ],  
i values in  $ST(0)$ . For  
the top of the register



this  
ve

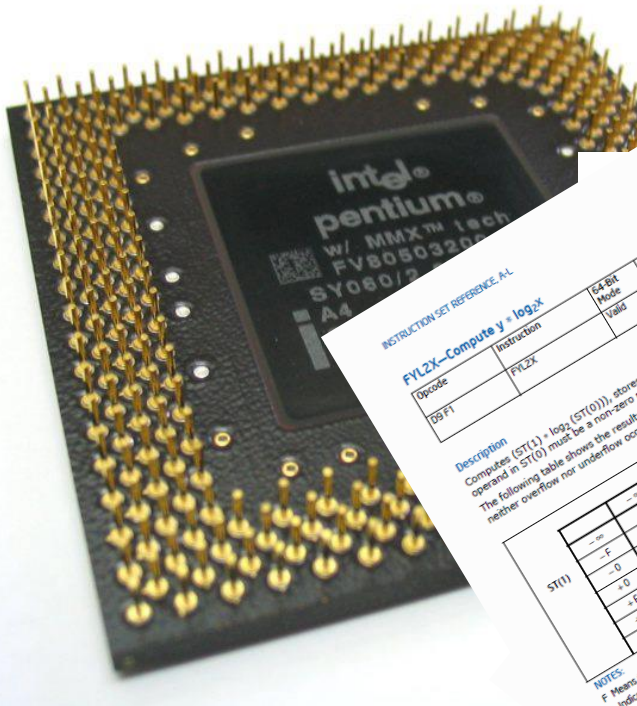
4 image  
ally was

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

Vol.2A 3-445

# NOWY CEL

# KOMPUTER NA UKŁADACH TTL



INSTRUCTION SET REFERENCE, A-L

FV LZx—Compute  $y = \log_2 x$

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode
D9 F1	FV LZx	Valid	Valid

**Description**  
Computes  $(ST(1) + \log_2(ST(0)))$  stores the operand in  $ST(0)$  must be a non-zero power of 2.  
The following table shows the results of neither overflow nor underflow occurs.

ST(1)	ST(0)	Result
0	0	0
0	1	1
1	0	0
1	1	1

**NOTES:**  
F Means false  
\* Indicates  
\*\* Indicates

If the is to 7

INSTRUCTION SET REFERENCE, A-L

MMX™ Technology Overview

INSTRUCTION SET REFERENCE, A-L

ed on 24-bit color

from 0 to 255

INSTRUCTION SET REFERENCE, A-L

FSQR—Square Root

**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 of neither overflow nor underflow occurs.

SRX (ST(0))	FSQR (ST(0))
0	0
1	1
2	1.414213562
3	1.732050808
4	2
5	2.236067977
6	2.449489743
7	2.645751311
8	2.828427125
9	3
10	3.16227766
11	3.31662479
12	3.464101615
13	3.605551275
14	3.741657387
15	3.872983346
16	4
17	4.123105626
18	4.262659732
19	4.399154979
20	4.532692567
21	4.663559178
22	4.791262846
23	4.916084081
24	5.038562627
25	5.159038811
26	5.277061945
27	5.392785771
28	5.506348617
29	5.617854601
30	5.727321551
31	5.834854421
32	5.940454444
33	6.044231734
34	6.146286395
35	6.246627073
36	6.345253811
37	6.442166617
38	6.537366504
39	6.630854471
40	6.722631528
41	6.812707776
42	6.901083215
43	6.987757845
44	7.072731675
45	7.156014706
46	7.237607037
47	7.317508768
48	7.395720000
49	7.472240731
50	7.547071962
51	7.620213703
52	7.691666054
53	7.761429915
54	7.829505286
55	7.895893167
56	7.960593568
57	8.023606489
58	8.084932930
59	8.144572891
60	8.202526372
61	8.258793383
62	8.313373924
63	8.366268005
64	8.417475726
65	8.466997087
66	8.514832088
67	8.560981729
68	8.606446010
69	8.651224931
70	8.695318502
71	8.738726723
72	8.781449594
73	8.823487015
74	8.864838986
75	8.905505507
76	8.945486578
77	8.984782199
78	9.023392370
79	9.061317091
80	9.098556362
81	9.135110183
82	9.170978554
83	9.206161475
84	9.240658946
85	9.274470967
86	9.307597538
87	9.340038659
88	9.371794330
89	9.402865551
90	9.433252322
91	9.462954643
92	9.491972514
93	9.520305935
94	9.547954906
95	9.574919427
96	9.601199498
97	9.626795119
98	9.651706290
99	9.675932911
100	9.699475082
101	9.722332803
102	9.744506174
103	9.765995195
104	9.786800876
105	9.806924207
106	9.826366188
107	9.845126809
108	9.863207070
109	9.880607981
110	9.897329542
111	9.913371753
112	9.928734514
113	9.943417725
114	9.957421486
115	9.970755797
116	9.983420558
117	9.995425769
118	10.006771430
119	10.017457641
120	10.027484402
121	10.036861713
122	10.045589574
123	10.053667985
124	10.061106946
125	10.067906457
126	10.074076518
127	10.079617129
128	10.084528290
129	10.088809901
130	10.092462062
131	10.095494773
132	10.097917934
133	10.099730545
134	10.100942606
135	10.101554167
136	10.101565728
137	10.101077289
138	10.100088850
139	10.098590411
140	10.096582072
141	10.094063733
142	10.091135394
143	10.087797055
144	10.084049716
145	10.079893377
146	10.075328038
147	10.070353799
148	10.064970560
149	10.059178321
150	10.052976082
151	10.046363843
152	10.039341604
153	10.031909365
154	10.024067126
155	10.015814887
156	10.007152648
157	9.997080409
158	9.985608170
159	9.972735931
160	9.958463692
161	9.942791453
162	9.925719214
163	9.907246975
164	9.887374736
165	9.866092497
166	9.843400258
167	9.819298019
168	9.793785780
169	9.766863541
170	9.738541302
171	9.708819063
172	9.677696824
173	9.645174585
174	9.611252346
175	9.575930107
176	9.539207868
177	9.491085629
178	9.441563390
179	9.390641151
180	9.338318912
181	9.284596673
182	9.229474434
183	9.172952195
184	9.115030056
185	9.055707817
186	8.994985578
187	8.932863339
188	8.869341090
189	8.804418851
190	8.738096612
191	8.670374373
192	8.601252134
193	8.530729895
194	8.458807656
195	8.385485417
196	8.310763178
197	8.234640939
198	8.157118700
199	8.078196461
200	8.0

**Description**

Subtracts the destination operand from the source operand. The destination operand is stored in the destination register. Source operands in 64-bit mode are doubleword integers.

These instructions support the SSE and SSE2 instructions. The no-operation instruction stores the result of a memory version, subtraction.

The FSUBRP instruction. To pop the register (TOP) by 1. The next stack being popped.

The FISUBR instruction before performing the operation. The following table shows assuming that neither overflow nor underflow occurs.

When the difference between the source operand is an integer and the destination register is a floating-point value, the operation exception is generated.

When one operand is a floating-point value and the other is an integer, the operation exception is generated.

When one operand is a floating-point value and the other is a floating-point value, the operation exception is generated.

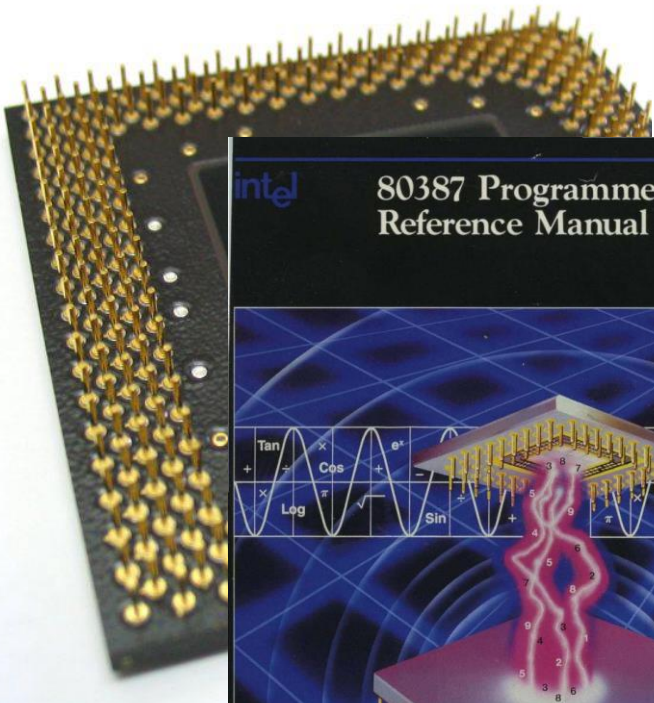
FSUBR/FSUBRP/FISUBR—Reverse Subtract

3460 Vol. 2a

A-37

# NOWY CEL

# KOMPUTER NA UKŁADACH TTL



**intel**

## 80387 Programmer's Reference Manual

Order Number: 231917-001

### FSUBR/FSUBRP/FISUBR—Reverse Subtract

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DB 75	FISUBR m327n	Valid	Valid	Subtract ST(0) from m327n 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(l) and store result in ST(0).
		Valid	Valid	Subtract ST(l) from ST(0) and store result in ST(l).
		Valid	Valid	Subtract ST(0) from ST(0), store result in pop register stack.
		Valid	Valid	Subtract ST(1) from ST(0), store result in pop register stack.
		Valid	Valid	Subtract ST(1) from ST(1), store result in pop register stack.
		Valid	Valid	Subtract ST(1) from ST(1), store result in pop register stack.
		Valid	Valid	Subtract ST(1) from ST(1), store result in pop register stack.
		Valid	Valid	Subtract ST(1) from ST(1), store result in pop register stack.

INSTRUCTION SET REFERENCE, A-L

ology Overview

erview

INSTRUCTION SET REFERENCE, A-L

24-bit color

from 0 to 255

### FSQR—Square Root

Opcode	Instruction	64-Bit Mode	Compat/Leg Mode	Description
DB 76	FISQR m327n	Valid	Valid	Compute the square root of m327n and store the result in ST(0).
		Valid	Valid	Compute the square root of m64fp and store the result in ST(0).
		Valid	Valid	Compute the square root of ST(l) and store the result in ST(0).
		Valid	Valid	Compute the square root of ST(0) and store the result in ST(l).
		Valid	Valid	Compute the square root of ST(0) and store the result in pop register stack.
		Valid	Valid	Compute the square root of ST(1) and store the result in pop register stack.
		Valid	Valid	Compute the square root of ST(1) and store the result in pop register stack.
		Valid	Valid	Compute the square root of ST(1) and store the result in pop register stack.
		Valid	Valid	Compute the square root of ST(1) and store the result in pop register stack.
		Valid	Valid	Compute the square root of ST(1) and store the result in pop register stack.

**NOTES:**

- 1. When the floating-point value is NaN, the instruction's operation is the same in non-64-bit modes and 64-bit mode.
- 2. When the floating-point value is a denormalized value, the instruction's operation is the same in non-64-bit modes and 64-bit mode.

**Operation:** FSQR—Square Root

**Flags Affected:** C0, C1, C2, C3

**Privileged Mode Exceptions:** Set to 0 if stack underflow occurred. If a result was rounded up, cleared, otherwise, undefined.

**Floating-Point Exceptions:** Stack underflow occurred. Source operand is an NaN value or unrepresented format. Source operand is a denormalized value. Value cannot be represented exactly in destination format.

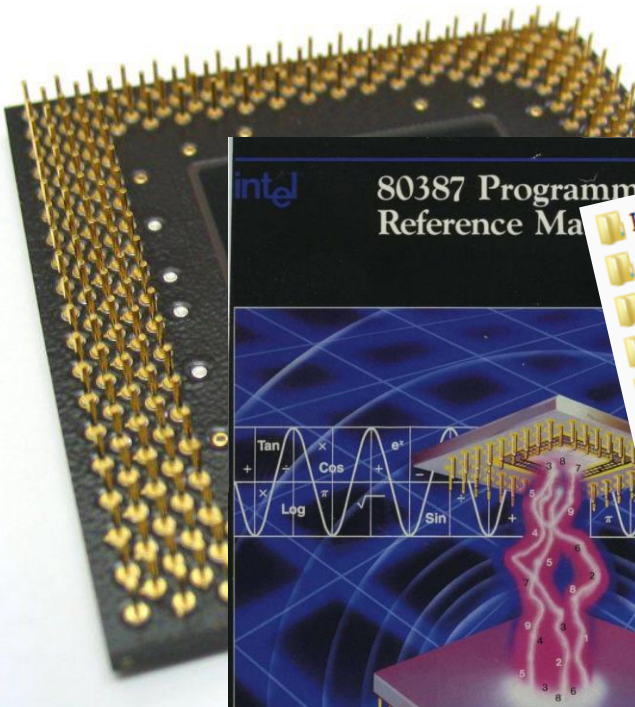
**Protected Mode Exceptions:** C00-Bit [3] or C00-Tbit [3] = 1. If there is a rounding of ST(0) exception. If the LOCK prefix is used, the LOCK prefix is in protected mode.

**FSQR—Square Root**

... are ...  
... one for ...  
... and blue planes.

# NOWY CEL

# KOMPUTER NA UKŁADACH TTL



Intel 80387 Programmable Floating-Point Coprocessor Reference Manual

Order Number: 231917-001

- 📁 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

### FSUBR/FSUBRP/FISUBR—Reverse Subtract

Opcode	Instruction	64-Bit Instruction
DB 75	FSUBR m32fp	

Description
Subtract ST(0) from m32fp and store result in ST(0).
Subtract ST(0) from m64fp and store result in ST(0).
Subtract ST(0) from ST(0) and store result in ST(0).
Subtract ST(0) from ST(0) and store result in ST(0).
Subtract ST(0) from ST(0), store result in ST(0), and pop the top element from the register stack.
Subtract ST(1) from ST(0) and store result in ST(0).
Subtract ST(1) from ST(0) and store result in ST(0).

**NOTES:**

- 1. When using floating-point data, the instruction's operation is the same in non-64-bit modes and 64-bit mode.
- 2. When using integer data, the instruction's operation is the same in non-64-bit modes and 64-bit mode.

**Operation:** SRC1[ST(0)] - SRC2[ST(0)]

# occurred.  
1 up: cleared, otherwise, set.

0: 15-bit or unrounded format.  
1: 64-bit or unrounded format.  
2: 64-bit FPU exception.  
3: used.

INSTRUCTION SET REFERENCE, A-1

Technology Overview

Overview

INSTRUCTION SET REFERENCE, A-1

24-bit color

from 0 to 255

### FSQRT—Square Root

#### 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 the register overflow or underflow occurs.

Table 3-27. FSQRT Results

Opcode	Instruction	64-Bit Mode	Control Word	Description
DB 7A	FSQRT	16-bit	Control Word	Computes square root of ST(0) and stores the result in ST(0).

... are  
... are: one for  
... the swan will be  
... and blue planes.



# NOWY CEL

# KOMPUTER NA UKŁADACH TTL



Intel 80387 Programm...  
 matematyki binarnej

ADD/ADDR/CADDcc/CADDRcc A,B	0	0115-0129	15	SUMOWANIE LICZB CAŁKOWITYCH
ADC/ADCR/CADCcc/CADCRCc A,B	0	0130-0144	15	SUMOWANIE LICZB CAŁKOWITYCH Z PRZENIESIENIEM
SUB/SUBR/CSUBcc/CSUBRCc A,B	0	0145-0159	15	ODEJMOWANIE LICZB CAŁKOWITYCH
SBB/SBBR/CSBBcc/CSBBRCc A,B	0	0160-0174	15	ODEJMOWANIE LICZB CAŁKOWITYCH Z PRZENIESIENIEM
RSUB/RSUBR/CRSUBcc/CRSUBRCc A,B	0	0175-0189	15	ODWROTNE ODEJMOWANIE LICZB CAŁKOWITYCH
RSBB/RSBBR/CRSBBcc/CRSBBRCc A,B	0	0190-0204	15	ODWROTNE ODEJMOWANIE LICZB CAŁKOWITYCH Z PRZENIESIENIEM
MUL/MULR/CMULcc/CMULRCc A,B	0	0205-0219	15	MNOŻENIE LICZB CAŁKOWITYCH BEZ ZNAKU
IMUL/IMULR/CMULcc/CMULRCc A,B	0	0220-0234	15	MNOŻENIE LICZB CAŁKOWITYCH ZE ZNAKIEM
DIV/IDIVR/CDIVcc/CDIVRCc A,B	0	0235-0249	15	DZIELENIE LICZB CAŁKOWITYCH BEZ ZNAKU BEZ RESZTY
IDIV/IDIVR/CDIVcc/CDIVRCc A,B	0	0250-0264	15	DZIELENIE LICZB CAŁKOWITYCH ZE ZNAKIEM BEZ RESZTY
RDIV/RDIVR/CRDIVcc/CRDIVRCc A,B	0	0265-0279	15	ODWROTNE DZIELENIE LICZB CAŁKOWITYCH BEZ ZNAKU BEZ RESZTY
RIDIV/RIDIVR/CRDIVcc/CRDIVRCc A,B	0	0280-0294	15	ODWROTNE DZIELENIE LICZB CAŁKOWITYCH ZE ZNAKIEM BEZ RESZTY
INC/CINcc A	0	0295-0300	6	ZWIĘKSZANIE ZAWARTOŚCI O JEDEN
DEC/CDECcc A	0	0301-0306	6	ZMNIJSZANIE ZAWARTOŚCI O JEDEN
NEG/CNEGcc A	0	0307-0312	6	NEGACJA DO 02
CMP/RCMP/CCMPcc/CRCMPcc A,B	0	0313-0327	15	PORÓWNIANIE PRZEZ ODEJMOWANIE
ADDE/ADDER/CADDEcc/CADDERcc A,B,C	1	0001-0017	17	SUMOWANIE LICZB CAŁKOWITYCH
ADCE/ADDER/CADCEcc/CADDERcc A,B,C	1	0018-0034	17	SUMOWANIE LICZB CAŁKOWITYCH Z PRZENIESIENIEM
SUBE/SUBER/CSUBEcc/CSUBERcc A,B,C	1	0035-0051	17	ODEJMOWANIE LICZB CAŁKOWITYCH
SBBE/SBBER/CSBBEcc/CSBBERcc A,B,C	1	0052-0068	17	ODEJMOWANIE LICZB CAŁKOWITYCH Z PRZENIESIENIEM
RSUBE/RSUBER/CRSUBEcc/CRSUBERcc A,B,C	1	0069-0085	17	ODWROTNE ODEJMOWANIE LICZB CAŁKOWITYCH
RSBBE/RSBBER/CRSBBEcc/CRSBBERcc A,B,C	1	0086-0102	17	ODWROTNE ODEJMOWANIE LICZB CAŁKOWITYCH Z PRZENIESIENIEM
MULE/MULER/CMULEcc/CMULERcc A,B,C,D	1	0103-0103	1	MNOŻENIE LICZB CAŁKOWITYCH BEZ ZNAKU
IMULE/IMULER/CMULEcc/CMULERcc A,B,C,D	1	0104-0104	1	MNOŻENIE LICZB CAŁKOWITYCH ZE ZNAKIEM
DIVE/IDIVER/CDIVEcc/CDIVERcc A,B,C,D	1	0105-0105	1	DZIELENIE LICZB CAŁKOWITYCH BEZ ZNAKU Z RESZTA
IDIVE/IDIVER/CDIVEcc/CDIVERcc A,B,C,D	1	0106-0106	1	DZIELENIE LICZB CAŁKOWITYCH ZE ZNAKIEM Z RESZTA
RDIVE/RDIVER/CRDIVEcc/CRDIVERcc A,B,C,D	1	0107-0107	1	ODWROTNE DZIELENIE LICZB CAŁKOWITYCH BEZ ZNAKU Z RESZTA
RIDIVE/RDIVER/CRDIVEcc/CRDIVERcc A,B,C,D	1	0108-0108	1	ODWROTNE DZIELENIE LICZB CAŁKOWITYCH ZE ZNAKIEM Z RESZTA

### FSUBR/FSUBRP/FISUBR—Reverse Subtract

Opcode	Instruction	64-Bit	16-Bit
DB/5	FSUBR,m32fp		

**Description**  
 Subtract ST(0) from m32fp and store result in ST(0).  
 Subtract ST(0) from m64fp and store result in ST(0).  
 Subtract ST(0) from ST(i) and store result in ST(0).  
 Subtract ST(i) from ST(0) and store result in ST(i).

ARGUMENT A  $\approx$  B = ARGUMENT A + ARGUMENT B  
 ARGUMENT A  $\approx$  B = ARGUMENT A + ARGUMENT B + c  
 ARGUMENT A  $\approx$  B = ARGUMENT A - ARGUMENT B  
 ARGUMENT A  $\approx$  B = ARGUMENT A - (ARGUMENT B + c)  
 ARGUMENT A  $\approx$  B = ARGUMENT B - ARGUMENT A  
 ARGUMENT A  $\approx$  B = ARGUMENT B - (ARGUMENT A + c)  
 ARGUMENT A  $\approx$  B = ARGUMENT A  $\times$  ARGUMENT B  
 ARGUMENT A  $\approx$  B = ARGUMENT A / ARGUMENT B  
 ARGUMENT A  $\approx$  B = ARGUMENT A / ARGUMENT B  
 ARGUMENT A  $\approx$  B = ARGUMENT B / ARGUMENT A  
 ARGUMENT A  $\approx$  B = ARGUMENT B / ARGUMENT A  
 ARGUMENT A = ARGUMENT A+1  
 ARGUMENT A = ARGUMENT A-1  
 ARGUMENT A = 0-ARGUMENT A  
 REGISTER FLAG = ARGUMENT A - ARGUMENT B  $\approx$  ARGUMENT B - ARGUMENT A  
 ARGUMENT A = ARGUMENT B + ARGUMENT C — ARGUMENT A + ARGUMENT B = ARGUMENT C  
 ARGUMENT A = ARGUMENT B + ARGUMENT C + c — ARGUMENT A + ARGUMENT B + c = ARGUMENT C  
 ARGUMENT A = ARGUMENT B - ARGUMENT C — ARGUMENT A - ARGUMENT B = ARGUMENT C  
 ARGUMENT A = ARGUMENT B - (ARGUMENT C + c) — ARGUMENT A - (ARGUMENT B + c) = ARGUMENT C  
 ARGUMENT A = ARGUMENT C - ARGUMENT B — ARGUMENT B - ARGUMENT A = ARGUMENT C  
 ARGUMENT A = ARGUMENT C - (ARGUMENT B + c) — ARGUMENT B - (ARGUMENT A + c) = ARGUMENT C  
 ARGUMENT ZESPOLONY AB = ARGUMENT C  $\times$  ARGUMENT D — ARGUMENT A  $\times$  ARGUMENT B = ARGUMENT ZESPOLONY CD  
 ARGUMENT ZESPOLONY AB = ARGUMENT C  $\times$  ARGUMENT D — ARGUMENT A  $\times$  ARGUMENT B = ARGUMENT ZESPOLONY CD  
 ARGUMENT ZESPOLONY AB = ARGUMENT C / ARGUMENT D — ARGUMENT A / ARGUMENT B = ARGUMENT ZESPOLONY CD  
 ARGUMENT ZESPOLONY AB = ARGUMENT C / ARGUMENT D — ARGUMENT A / ARGUMENT B = ARGUMENT ZESPOLONY CD  
 ARGUMENT ZESPOLONY AB = ARGUMENT D / ARGUMENT C — ARGUMENT B / ARGUMENT A = ARGUMENT ZESPOLONY CD  
 ARGUMENT ZESPOLONY AB = ARGUMENT D / ARGUMENT C — ARGUMENT B / ARGUMENT A = ARGUMENT ZESPOLONY CD

INSTRUCTION SET REFERENCE, A-L

ology Overview

erview

TION SET REFERENCE, A-L

n 24-bit color

from 0 to 255

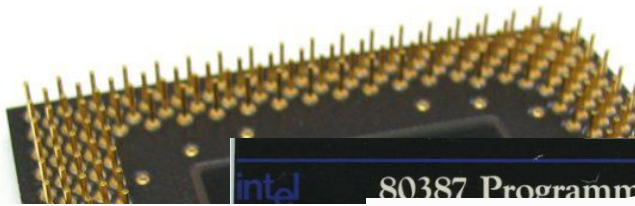


... are  
 ... one for  
 ... will be  
 ... and blue planes.



# NOWY CEL

# KOMPUTER NA UKŁADACH TTL



### FSUBR/FSUBRP/FSIBR—Reverse Subtract

Opcode	Instruction	64-Bit Description
DB 75	FSUBR, m32b	Subtract ST(0) from m32b and store result in ST(0).
		Subtract ST(0) from m64b and store result in ST(0).
		Subtract ST(0) from ST(0) and store result in ST(0).
		Subtract ST(0) from ST(0) and store result in ST(0).

INSTRUCTION SET REFERENCE, A-L

Technology Overview

Overview

INSTRUCTION SET REFERENCE, A-L

24-bit color

from 0 to 255



Opcode	Instruction	64-Bit Description	64-Bit Description
FDZ/CFLDZcc STX	0 1182-1182	1 ARGUMENT STX ← 0	ZAŁADOWANIE WARTOŚCI LICZBY 0 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
FLD1/CFLD1cc STX	0 1183-1183	1 ARGUMENT STX ← 1	ZAŁADOWANIE WARTOŚCI LICZBY 1 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
FLDPI/CFLDPIcc STX	0 1184-1184	1 ARGUMENT STX ← π	ZAŁADOWANIE WARTOŚCI LICZBY π W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
FLDLN2/CFLDLN2cc STX	0 1185-1185	1 ARGUMENT STX ← ln2	ZAŁADOWANIE WARTOŚCI LICZBY ln2 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
FLDLNT/CFLDLNTcc STX	0 1186-1186	1 ARGUMENT STX ← ln10	ZAŁADOWANIE WARTOŚCI LICZBY ln10 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
FLDL2E/CFLDL2Ecc STX	0 1187-1187	1 ARGUMENT STX ← log2e	ZAŁADOWANIE WARTOŚCI LICZBY log2e W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
FLDL2T/CFLDL2Tcc STX	0 1188-1188	1 ARGUMENT STX ← log2_10	ZAŁADOWANIE WARTOŚCI LICZBY log2_10 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
FLDLG2/CFLDLG2cc STX	0 1189-1189	1 ARGUMENT STX ← log10_2	ZAŁADOWANIE WARTOŚCI LICZBY log10_2 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
FLDLGE/CFLDLGEcc STX	0 1190-1190	1 ARGUMENT STX ← log10_e	ZAŁADOWANIE WARTOŚCI LICZBY log10_e W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

- FIADD/FRADDcc STX
- FSUBR/FRSUBR/CFSUBRcc A,STX
- FRSUBR/CFRSUBRcc A,STX
- FRSUBR/CFRSUBRcc/CFSUBRcc STX,STX
- FSUBR/FRSUBR/CFSUBRcc A,STX
- FSUBR/CFSUBRcc A,STX
- FSUBR/CFSUBRcc/CFMULcc/CFRMULcc STX,STX
- FMUL/FRMUL/CFMULcc/CFRMULcc A,STX
- FMUL/CFMULcc A,STX
- FMUL/CFMULcc/CFDIVcc/CFRDIVcc STX,STX
- FDIV/FRDIV/CFDIVcc/CFRDIVcc A,STX
- FRDIV/CFRDIVcc A,STX
- FRDIV/CFRDIVcc/CFDIVcc STX,STX
- FDIV/CFDIVcc A,STX
- FDIV/CFDIVcc A,STX
- FDIV/CFDIVcc/CFPREMcc STX,STX
- FPREM/CFPREMcc STX,STX
- FPREM/CFPREMcc STX,STX
- FABS/CFABScc STX
- FCBS/CFCHSc STX
- FRNDINT/CFRNDINTcc STX
- FXTRACT/CFXTRACTcc STX,STX
- FSCALE/CFSCALEcc STX,STX
- FSQRT/CFSQRTcc STX

0 1088-1088	6	0	ODEJMOWANIE
0 1089-1094	6	1	MNOŻENIE DWAŃ 76 BITOWYCH LICZBY CAŁKOWITEJ U2 Z 76 BITOWYCH LICZBY ZMIENNOPRZECINKOWEJ
0 1095-1101	6	1	MNOŻENIE 32 lub 64 BITOWEJ LICZBY CAŁKOWITEJ U2 Z 76 BITOWYCH LICZBY ZMIENNOPRZECINKOWEJ
0 1101-1101	6	1	MNOŻENIE 32 lub 64 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ PRZEZ 32 lub 64 BITOWĄ LICZBĘ CAŁKOWITĄ U2
0 1102-1107	6	1	DZIELENIE DWAŃ 76 BITOWYCH LICZBY ZMIENNOPRZECINKOWEJ PRZEZ 32 lub 64 BITOWĄ LICZBĘ CAŁKOWITĄ U2
0 1108-1113	6	1	DZIELENIE 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ PRZEZ 76 BITOWĄ LICZBĘ ZMIENNOPRZECINKOWĄ
0 1114-1114	6	1	DZIELENIE 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ PRZEZ 76 BITOWĄ LICZBĘ ZMIENNOPRZECINKOWĄ
0 1115-1120	6	1	ODWROTNE DZIELENIE DWAŃ 76 BITOWYCH LICZBY ZMIENNOPRZECINKOWEJ PRZEZ 76 BITOWĄ LICZBĘ ZMIENNOPRZECINKOWĄ
0 1121-1126	6	1	DZIELENIE 32 lub 64 BITOWEJ LICZBY CAŁKOWITEJ U2 Z 76 BITOWYCH LICZBY ZMIENNOPRZECINKOWEJ
0 1127-1127	6	1	DZIELENIE 32 lub 64 BITOWEJ LICZBY CAŁKOWITEJ U2 Z 76 BITOWYCH LICZBY ZMIENNOPRZECINKOWEJ
0 1128-1133	6	1	OBLICZANIE RESZTY Z DZIELENIA DWAŃ 76 BITOWYCH LICZBY ZMIENNOPRZECINKOWEJ
0 1134-1139	6	1	OBLICZANIE RESZTY Z DZIELENIA DWAŃ 76 BITOWYCH LICZBY ZMIENNOPRZECINKOWEJ
0 1140-1140	6	1	OBLICZANIE WARTOŚCI ABSOLUTNEJ 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
0 1141-1141	6	1	OBLICZANIE WARTOŚCI ABSOLUTNEJ 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
0 1142-1142	6	1	ZMIANA ZNAKU 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
0 1143-1143	6	1	ZAKRĄGLANIE 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ DO NAJBLIŻSZEJ WARTOŚCI CAŁKOWITEJ
0 1144-1144	6	1	ROZDZIELENIE MANTYSY I WYKŁADNIKA 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
0 1145-1145	6	1	POTĘGOWANIE PRZEZ WYKŁADNIK CAŁKOWITY 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
0 1146-1146	6	1	OBLICZANIE PIERWIĄSKA KWADRATOWEGO 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
0 1147-1147	6	1	OBLICZANIE PIERWIĄSKA KWADRATOWEGO 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ARGUMENT STX ← ARGUMENT A  
 ARGUMENT STX = ARGUMENT A + ARGUMENT B  
 ARGUMENT STX = STX / STX  
 ARGUMENT STX = STX / STX  
 ARGUMENT [STX] ← STX  
 ARGUMENT ± STX ← ± STX  
 ARGUMENT ± STX ← ± STX  
 MANTYSA W STX ← STX - WYKŁADNIK W STX  
 ARGUMENT STX = STX × 2<sup>A</sup> STX  
 ARGUMENT ± STX ← STX

- + c) = ARGUMENT C
- ARGUMENT C
- c) = ARGUMENT C
- ARGUMENT B = ARGUMENT ZESPOLONY CD
- ARGUMENT B = ARGUMENT ZESPOLONY CD
- ARGUMENT B = ARGUMENT ZESPOLONY CD
- ARGUMENT B = ARGUMENT ZESPOLONY CD
- ARGUMENT B = ARGUMENT ZESPOLONY CD
- ARGUMENT B = ARGUMENT ZESPOLONY CD
- ARGUMENT B = ARGUMENT ZESPOLONY CD



... are  
 ... one for  
 ... one swan will be  
 ... and blue planes.

# NOWY CEL

# KOMPUTER NA UKŁADACH TTL

- FCOM/FCOM/CFCOMcc/CFRCOMcc STX,STX
- FCOM/FCOM/CFCOMcc/CFRCOMcc A,STX
- FCOM/FRICOM/CFICOMcc/CFRICOMcc A,STX
- FCOM/FRICOM/CFICOMcc/CFRICOMcc STX,STX
- FCOM/FRICOM/CFICOMcc/CFRICOMcc A,STX
- FTST/CFSTcc STX
- FXAM/CFXAMcc STX
- FLDZ/CFLDZcc STX
- FLD1/CFLD1cc STX
- FLDPI/CFLDPIcc STX
- FLDLN2/CFLDLN2cc STX
- FLDLNT/CFLDLNTcc STX
- FLDL2E/CFLDL2Ecc STX
- FLDL2T/CFLDL2Tcc STX
- FLDLG2/CFLDLG2cc STX
- FLDLGE/CFLDLGEcc STX
- FIAD/FRSIBR/CFISUBcc...
- FSUBR/CFRSUBcc A,STX
- FRSIBR/CFRSUBcc/CFRSUBcc STX,STX
- FRSIBR/CFRSUBcc/CFRSUBcc A,STX
- FSUB/CFISUBcc A,STX
- FSUBR/CFISUBcc A,STX
- FMUL/FRMULR/CFMULcc/CFRMULcc STX,STX
- FMULR/CFMULcc A,STX
- FMULR/CFMULcc/CFRDIVcc STX,STX
- FDIV/CFRDIVcc A,STX
- FRDIVR/CFRDIVcc STX,STX
- FRDIVR/CFRDIVcc A,STX
- FRDIV/CFRDIVcc A,STX
- FDIVR/CFDIVcc A,STX
- FIDIVR/CFIDIVcc A,STX
- FPREM/CFPREMcc STX,STX
- FPREM/CFPREMcc A,STX
- FABS/CFABScc STX
- FCHS/CFCHScc STX
- FRNDINT/CFRNDINTcc STX
- FXTRACT/CFXTRACTcc STX,STX
- FSCALE/CFSCALEcc STX,STX
- FSQRT/CFSQRTcc STX

Opcode	Instruction	64-Bit	Description
0B 75	FSUBR m32fp		Subtract ST(0) from m32fp and store result in ST(0).
0B 76	FSUBR m64fp		Subtract ST(0) from m64fp and store result in ST(0).
0B 77	FSUBR		Subtract ST(0) from ST(i) and store result in ST(0).
0B 78	FSUBR		Subtract ST(i) from ST(0) and store result in ST(0).

FSUBR/FSUBRP/FISUBR—Reverse Subtract

...metyki binarnej

ZAŁADOWANIE WARTOŚCI LICZBY 0 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY 1 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY π W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY ln2 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY ln10 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY ln2 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY ln10 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY π W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY ln2 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY ln10 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY π W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY ln2 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY ln10 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY π W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY ln2 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY ln10 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY π W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY ln2 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY ln10 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY π W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY ln2 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY ln10 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

ZAŁADOWANIE WARTOŚCI LICZBY π W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

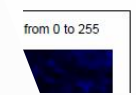
INSTRUCTION SET REFERENCE, A-L

ology Overview

erview

TION SET REFERENCE, A-L

a 24-bit color



# NOWY CEL

## KOMPUTER NA UKŁADACH TTL

- FCOM/FCOM/CFCOMcc/CFRCOMcc STX,STX
- FCOM/FCOM/CFCOMcc/CFRCOMcc A,STX
- FCOM/FRICOM/CFICOMcc/CFRCOMcc STX,STX
- FCOM/FRICOM/CFICOMcc/CFRCOMcc A,STX
- FCOM/FRICOM/CFICOMcc/CFRCOMcc STX,STX
- FTST/CFTSTcc STX
- FXAM/CFXAMcc STX
- FLDZ/CFLDZcc STX
- FLD1/CFLD1cc STX
- FLDPI/CFLDPIcc STX
- FLDLN2/CFLDLN2cc STX
- FLDLNT/CFLDLNTcc STX
- FLDL2E/CFLDL2Ecc STX
- FLDL2T/CFLDL2Tcc STX
- FLDLG2/CFLDLG2cc STX
- FLDLGE/CFLDLGEcc STX
- FSIN/CFOScc STX
- FCOS/CFOScc STX
- FTAN/CFTANcc STX
- FRATAN2/CFRATAN2cc STX,STX
- FEX/CFEXcc STX
- FL2X/CFL2Xcc STX
- FIDIVR/CFRIVRcc STX,STX
- FPREM/CFPREMcc STX,STX
- FPREM/CFPREMcc STX,STX
- FABS/CFABScc STX
- FCHS/CFCHScc STX
- FRNDINT/CFRNDINTcc STX
- FXTRACT/CFXTRACTcc STX,STX
- FSCALE/CFSCALEcc STX,STX
- FSQRT/CFSQRTcc STX

Opcode	Instruction	Description
0B 75	FSUBR m32fp	
0	1148-1148	1 PORÓWNYWANIE PRZEZ ODEJMOWANIE DWÓCH 76 BITOWYCH LICZB ZMIENNOPRZECINKOWYCH
0	1149-1154	6 PORÓWNYWANIE PRZEZ ODEJMOWANIE 32 lub 64 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
0	1155-1160	6 PORÓWNYWANIE PRZEZ ODEJMOWANIE 32 lub 64 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
0	1161-1161	6 PORÓWNYWANIE PRZEZ ODEJMOWANIE DWÓCH 76 BITOWYCH LICZB ZMIENNOPRZECINKOWYCH
0	1162-1167	6 PORÓWNYWANIE PRZEZ ODEJMOWANIE 32 lub 64 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
0	1168-1173	6 PORÓWNYWANIE PRZEZ ODEJMOWANIE DWÓCH 76 BITOWYCH LICZB ZMIENNOPRZECINKOWYCH
0	1174-1174	6 PORÓWNYWANIE PRZEZ ODEJMOWANIE 32 lub 64 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
0	1175-1175	6 PORÓWNYWANIE PRZEZ ODEJMOWANIE 32 lub 64 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
0	1185-1185	6 ANALIZA 76 BITOWEJ WARTOŚCI ZMIENNOPRZECINKOWEJ Z ZEREM ZMIENNOPRZECINKOWYM
0	1186-1186	1 ARGUMENT STX ← log <sub>2</sub> 10
0	1187-1187	1 ARGUMENT STX ← log <sub>10</sub> 2
0	1188-1188	1 ARGUMENT STX ← log <sub>10</sub> e
0	1088-1088	6 ODEJMOWANIE DWÓCH 76 BITOWYCH LICZB ZMIENNOPRZECINKOWYCH
0	1089-1094	6 MNOŻENIE DWÓCH 76 BITOWYCH LICZB ZMIENNOPRZECINKOWYCH
0	1095-1100	6 MNOŻENIE 32 lub 64 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ PRZEZ 32 lub 64 BITOWĄ LICZBĘ CAŁKOWITĄ
0	1101-1101	6 MNOŻENIE DWÓCH 76 BITOWYCH LICZB ZMIENNOPRZECINKOWYCH
0	1102-1107	6 DZIELENIE DWÓCH 76 BITOWYCH LICZB ZMIENNOPRZECINKOWYCH
0	11176-11176	1 OBLICZANIE FUNKCJI SINUS 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
0	11177-11177	1 OBLICZANIE FUNKCJI COSINUS 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
0	11178-11178	1 OBLICZANIE FUNKCJI TANGENS 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ
0	11179-11179	1 OBLICZANIE KĄTA W UKŁADZIE WSPÓRZĘDNYCH POMIĘDZY DWIEMA 76 BITOWYMI LICZBAMI ZMIENNOPRZECINKOWYMI
0	11180-11180	1 POTĘGOWANIE PRZY PODSTAWIE LICZBY EULERA PRZEZ 76 BITOWY WYKŁADNIK ZMIENNOPRZECINKOWY
0	11181-11181	1 OBLICZANIE LOGARYTMU O PODSTAWIE DWA 76 BITOWY LICZBY ZMIENNOPRZECINKOWEJ
0	1143-1143	1 ZAOKRĄGLANIE
0	1143-1143	1 ROZDZIELENIE MANTYSY I WYKŁADNIKA
0	1144-1144	1 POTĘGOWANIE PRZEZ WYKŁADNIK CAŁKOWITY
0	1145-1145	1 OBLICZANIE PIERWIASKA KWADRATOWEGO
0	1146-1146	1 OBLICZANIE PIERWIASKA KWADRATOWEGO
0	1147-1147	1 OBLICZANIE PIERWIASKA KWADRATOWEGO

### FSUBR/FSUBRP/FISUBR—Reverse Subtract

INSTRUCTION SET REFERENCE, A-L

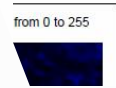
ology Overview

erview

Opcode	Instruction	Description
0B 75	FSUBR m32fp	Subtract ST(0) from m32fp and store result in ST(0).
0B 76	FSUBR m64fp	Subtract ST(0) from m64fp and store result in ST(0).
0B 77	FSUBR ST(0)	Subtract ST(0) from ST(0) and store result in ST(0).

ITION SET REFERENCE, A-L

n 24-bit color



ZAŁADOWANIE WARTOŚCI LICZBY 0 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ  
 ZAŁADOWANIE WARTOŚCI LICZBY 1 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ  
 ZAŁADOWANIE WARTOŚCI LICZBY π W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ  
 ZAŁADOWANIE WARTOŚCI LICZBY ln 2 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ  
 ZAŁADOWANIE WARTOŚCI LICZBY ln 10 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ  
 ZAŁADOWANIE WARTOŚCI LICZBY 1 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ  
 ZAŁADOWANIE WARTOŚCI LICZBY π W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ  
 ZAŁADOWANIE WARTOŚCI LICZBY ln 2 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ  
 ZAŁADOWANIE WARTOŚCI LICZBY ln 10 W POSTACI 76 BITOWEJ LICZBY ZMIENNOPRZECINKOWEJ

REJESTR FLAG FPU = ARGUMENT STX - ARGUMENT STX ≈ ARGUMENT STX - ARGUMENT STX  
 REJESTR FLAG FPU = ARGUMENT A - ARGUMENT STX ≈ ARGUMENT STX - ARGUMENT A  
 REJESTR FLAG FPU = ARGUMENT A - ARGUMENT STX ≈ ARGUMENT STX - ARGUMENT A  
 REJESTR FLAG FPU = ARGUMENT STX - ARGUMENT STX ≈ ARGUMENT STX - ARGUMENT STX  
 REJESTR FLAG FPU = ARGUMENT A - ARGUMENT STX ≈ ARGUMENT STX - ARGUMENT A  
 REJESTR FLAG FPU = ARGUMENT A - ARGUMENT STX ≈ ARGUMENT STX - ARGUMENT A  
 REJESTR FLAG FPU = ARGUMENT STX - ARGUMENT STX ≈ ARGUMENT STX - ARGUMENT STX  
 REJESTR FLAG FPU = ARGUMENT STX - ARGUMENT STX ≈ ARGUMENT STX - ARGUMENT STX  
 ARGUMENT STX = SIN(STX)  
 ARGUMENT STX = COS(STX)  
 ARGUMENT STX = TAN(STX)  
 ARGUMENT STX = ARCTAN(STX/STX)  
 ARGUMENT STX = e^(STX)  
 ARGUMENT STX = LOG2(STX)























# NOWY CEL

# KOMPUTER NA UKŁADACH TTL

Symbol	Opis	Adres	Wersja	Opis	Opis
PADDB/PADDR/CPADDcc/CPADDRcc MMX,MMX	0	0732-0732	1	DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWJANIA WYNIKU	$MMX_B = MMX_B + MMX_B$
PADDB/PADDR/CPADDcc/CPADDRcc A,MMX	0	0733-0738	6	DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWJANIA WYNIKU	$A, B = A, B + MMX, B$
PADDW/PADDWR/CPADDWcc/CPADDWRcc MMX,MMX	0	0739-0739	1	DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWJANIA WYNIKU	$MMX_W = MMX_W + MMX_W$
PADDW/PADDWR/CPADDWcc/CPADDWRcc A,MMX	0	0740-0745	6	DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWJANIA WYNIKU	$A, W = A, W + MMX, W$
PADDQ/PADDR/CPADDcc/CPADDRcc MMX,MMX	0	0746-0746	1	DODAWANIE DWÓCH PODWÓJNYCH SŁÓW W TRYBIE ZAWJANIA WYNIKU	$MMX_D = MMX_D + MMX_D$
PADDQ/PADDR/CPADDcc/CPADDRcc A,MMX	0	0747-0752	6	DODAWANIE DWÓCH PODWÓJNYCH SŁÓW W TRYBIE ZAWJANIA WYNIKU	$A, D = A, D + MMX, D$
PADDSDR/CPADDSDcc/CPADDSDRcc MMX,MMX	0	0753-0753	1	DODAWANIE OŚMIU BAJTÓW ZE ZNAKIEM WYNIKU	$M^*$
PADDSDR/CPADDSDcc/CPADDSDRcc A,MMX	0	0754-0758	6	DODAWANIE OŚMIU BAJTÓW ZE ZNAKIEM WYNIKU	$M^*$
SAHL/SAHLR/CSAHLcc/CSAHLRcc A,B	0	0505-0519	15	PRZESUNIĘCIE ARYTMETYCZNE / LOGICZNE W LEWO	$c-[BIT\ MSB]=A=[BIT\ LSB]-0 \Leftrightarrow [NOS++]\ B$
SAR/SARR/CSARcc/CSARRcc A,B	0	0520-0534	15	PRZESUNIĘCIE ARYTMETYCZNE W PRAWO	$\Rightarrow [BIT\ MSB]=A=[BIT\ LSB]-c \Leftrightarrow [NOS++]\ B$
SHR/SHRR/CSHRcc/CSHRRcc A,B	0	0535-0549	15	PRZESUNIĘCIE LOGICZNE W PRAWO	$0-[BIT\ MSB]=A=[BIT\ LSB]-c \Leftrightarrow [NOS++]\ B$
RCL/RCLR/CRCLcc/CRCLRcc A,B	0	0550-0564	15	OBRÓT W LEWO POPRZEC FLAGĘ PRZENIESIENIA	$0-[BIT\ MSB]=A=[BIT\ LSB]-c \Leftrightarrow [NOS++]\ B$
RCR/RCRR/RCRcc/RCRRcc A,B	0	0565-0579	15	OBRÓT W PRAWO POPRZEC FLAGĘ PRZENIESIENIA	$c-[BIT\ MSB]=A=[BIT\ LSB]-c \Leftrightarrow [NOS++]\ B$
ROL/ROLR/CROLcc/CROLRcc A,B	0	0580-0594	15	OBRÓT W LEWO Z UJĘCIEM FLAGI PRZENIESIENIA	$c-[BIT\ MSB]=A=[BIT\ LSB]-[BIT\ MSB] \Leftrightarrow [NOS++]\ A$
ROR/RORR/RORcc/RCORcc A,B	0	0595-0609	15	OBRÓT W PRAWO Z UJĘCIEM FLAGI PRZENIESIENIA	$[BIT\ LSB]-[BIT\ MSB]=A=[BIT\ LSB]-c \Leftrightarrow [NOS++]\ B$
SHLD/SHLDR/CSHLDRcc/CRSHDRcc A,B,C	1	0109-0125	17	PRZESUNIĘCIE LOGICZNE W LEWO DWÓCH OPERANDÓW	$c-[BIT\ MSB]=A=[BIT\ LSB]-[BIT\ MSB]=B=[BIT\ LSB] \Leftrightarrow [NOS++]\ C$
SHRD/SHRDR/CSHRDRcc/CRSHDRcc A,B,C	1	0126-0142	17	PRZESUNIĘCIE LOGICZNE W PRAWO DWÓCH OPERANDÓW	$[BIT\ MSB]=A=[BIT\ LSB]-[BIT\ MSB]=B=[BIT\ LSB]-c \Leftrightarrow [NOS++]\ C$
INS/INSR/CINSRcc A,B,C	1	0144-0144	1	ŁAŃCUCHOWE ODCZYTYWANIE DANYCH Z URZĄDZENIA	$ENT\ A \leftarrow [ARGUMENT\ B \Leftrightarrow PORT\ I/O] \Leftrightarrow [DA++]\ C$
OUTS/OUTSR/COUTSRcc A,B,C	1	0145-0145	1	ŁAŃCUCHOWE ZAPISYWANIE DANYCH DO URZĄDZENIA	$ENT\ A \leftarrow [ARGUMENT\ B \Rightarrow PORT\ I/O] \Leftrightarrow [DA++]\ C$
CMP5BCC/CMP5WCC/CMP5DCC/RCMP5BCC/RCMP5WCC/RCMP5DCC A,B,C	1	0146-0146	1	PORÓWNYWANIE PRZEZ ODEJMOWANIE ŁAŃCUCHÓW DAN	$RG = ARGUMENT\ A - ARGUMENT\ B \Leftrightarrow ARGUMENT\ B - ARGUMENT\ A \Leftrightarrow [DA++]\ C$
SCASBCC/SCASWCC/SCASDCC/RSCASBCC/RSCASWCC/RSCASDCC A,B,C	1	0147-0147	1	PORÓWNYWANIE PRZEZ ODEJMOWANIE WZORCA DANYCH	$RG = ARGUMENT\ A - ARGUMENT\ B \Leftrightarrow ARGUMENT\ B - ARGUMENT\ A \Leftrightarrow [DA++]\ C$
STOSD/CSTOSDcc A,B,C	1	0148-0148	1	POWIELANIE WZORCA DANYCH	$ARGUMENT\ (B) \Leftrightarrow [DA++]\ C$
PSRLQR/PSRLQRcc A,MMX	0	0969-0975	7	PRZESUNIĘCIE	$[BIT\ MSB]=MMX\_Q$
PSRAW/PSRAWR/PSRAWcc/PSRAWRcc MMX,MMX	0	0976-0976	1	PRZESUNIĘCIE	$W \Leftrightarrow [NOS++]\ MMX \Rightarrow [BIT\ MSB]=MMX\_W$
PSRAWR/PSRAWRcc A,MMX	0	0977-0983	7	PRZESUNIĘCIE	$[MSB]=MMX\_W$
PSRAD/PSRADR/PSRADcc/PSRADRcc MMX,MMX	0	0984-0984	1	PRZESUNIĘCIE	$[NOS++]\ MMX \Rightarrow [BIT\ MSB]=MMX\_D$
PSRADR/PSRADRcc A,MMX	0	0985-0991	7	PRZESUNIĘCIE	$=MMX\_D\_W$
PAND/PANDR/CPANDcc/CPANDRcc A,B	0	0454-0468	15	AND	$MMX\_D \Leftrightarrow MMX\_D$
PANDR/CPANDRcc A,B	0	0469-0483	15	AND	$MMX\_D \Leftrightarrow MMX\_D$
FS/POR/PORR/CPORcc/CPORRcc MMX,MMX	0	0484-0498	6	AND	$MMX\_W \Leftrightarrow MMX\_W$
FC/PCXOR/PCXORR/CPXORcc/CPXORRcc MMX,MMX	0	0499-0504	6	AND	$MMX\_W \Leftrightarrow MMX\_W$
FTAI/PCXOR/PCXORR/CPXORcc/CPXORRcc A,MMX	0	0499-0504	6	AND	$MMX\_W \Leftrightarrow MMX\_W$
FRAT/PCXOR/PCXORR/CPXORcc/CPXORRcc A,MMX	0	0499-0504	6	AND	$MMX\_W \Leftrightarrow MMX\_W$
KATAN2cc STX,STX	0	0499-0504	6	AND	$MMX\_W \Leftrightarrow MMX\_W$
FFX/CFXcc STX	1	0858-0858	1	DODAWANIE DRZY PODSTAWIE I	$MMX\_W \Leftrightarrow MMX\_W$
PCMPQEB/PCMPQEBR/CPMPQEBcc/CPMPQEBRcc A,B	0	0859-0864	6	PORÓWNYWANIE OŚMIU BAJTÓW Z OŚMIOMA BAJTAMI I SPRAWDZENIE CZY SĄ RÓWNE	$MMX\_B = MMX\_B - MMX\_B$
PCMPQEBR/PCMPQEBRcc A,B	0	0859-0864	6	PORÓWNYWANIE OŚMIU BAJTÓW Z OŚMIOMA BAJTAMI I SPRAWDZENIE CZY SĄ RÓWNE	$MMX\_B = MMX\_B - MMX\_B$
PCMPQEW/PCMPQEWcc A,B	0	0865-0865	1	PORÓWNYWANIE CZTERECH SŁÓW Z CZTEREMA SŁOWAMI I SPRAWDZENIE CZY SĄ RÓWNE	$MMX\_W = MMX\_W - MMX\_W$
PCMPQEWcc A,B	0	0866-0871	6	PORÓWNYWANIE CZTERECH SŁÓW Z CZTEREMA SŁOWAMI I SPRAWDZENIE CZY SĄ RÓWNE	$MMX\_W = MMX\_W - MMX\_W$
PCMPQWR/PCMPQWRcc A,B	0	0872-0872	1	PORÓWNYWANIE DWÓCH PODWÓJNYCH SŁÓW Z DWOMA PODWÓJNYMI SŁOWAMI I SPRAWDZENIE CZY SĄ RÓWNE	$MMX\_D = MMX\_D - MMX\_D$
PCMPQWRcc A,B	0	0873-0878	6	PORÓWNYWANIE DWÓCH PODWÓJNYCH SŁÓW Z DWOMA PODWÓJNYMI SŁOWAMI I SPRAWDZENIE CZY SĄ RÓWNE	$MMX\_D = MMX\_D - MMX\_D$
PCMPGTB/PCMPGTBcc A,B	0	0879-0879	1	PORÓWNYWANIE ZE ZNAKIEM OŚMIU BAJTÓW Z OŚMIOMA BAJTAMI I SPRAWDZENIE CZY SĄ WIĘKSZE	$MMX\_B > MMX\_B \rightarrow MMX\_B$
PCMPGTBcc A,B	0	0880-0885	6	PORÓWNYWANIE ZE ZNAKIEM OŚMIU BAJTÓW Z OŚMIOMA BAJTAMI I SPRAWDZENIE CZY SĄ WIĘKSZE	$MMX\_B > MMX\_B \rightarrow MMX\_B$
PCMPGTW/PCMPGTWcc A,B	0	0886-0886	1	PORÓWNYWANIE ZE ZNAKIEM CZTERECH SŁÓW Z CZTEREMA SŁOWAMI I SPRAWDZENIE CZY SĄ WIĘKSZE	$MMX\_W > MMX\_W \rightarrow MMX\_W$
PCMPGTWcc A,B	0	0887-0892	6	PORÓWNYWANIE ZE ZNAKIEM CZTERECH SŁÓW Z CZTEREMA SŁOWAMI I SPRAWDZENIE CZY SĄ WIĘKSZE	$MMX\_W > MMX\_W \rightarrow MMX\_W$
PCMPGD/PCMPGDcc A,B	0	0893-0893	1	PORÓWNYWANIE ZE ZNAKIEM DWÓCH PODWÓJNYCH SŁÓW Z DWOMA PODWÓJNYMI SŁOWAMI I SPRAWDZENIE CZY SĄ WIĘKSZE	$MMX\_D > MMX\_D \rightarrow MMX\_D$
PCMPGDcc A,B	0	0894-0899	6	PORÓWNYWANIE ZE ZNAKIEM DWÓCH PODWÓJNYCH SŁÓW Z DWOMA PODWÓJNYMI SŁOWAMI I SPRAWDZENIE CZY SĄ WIĘKSZE	$MMX\_D > MMX\_D \rightarrow MMX\_D$











# NOWY CEL

## KOMPUTER NA UKŁADACH TTL

**BTT/BTRR/CBTTcc/CBTRcc A,B**  
**BTS/BTSR/CBTScc/CBTRcc A,B**  
**BTR/BTRR/CBTRcc/CBTRcc A,B**  
**BTC/BTCR/CBTCcc/CBTRcc A,B**  
**BSF/BSFR/CBSFcc/CBSFcc A,B**  
**BSR/BSRR/CBSRcc/CBSRcc A,B**  
**POPCNT/POPCNTR/CPOPCNTcc/CPOPCNTRcc A,B**  
**TEST/CTESTcc A,B**  
**SETCC A**  
**CALLR/CCALLRcc A,[B]**  
**RET/CRETcc**  
**SHRD/RSHRU**  
**INSD/CINSDcc A,B,C**  
**OUTSD/COUTSDcc A,B,C**  
**CMPBCC/CMPSWCC/CMPSBCC/CMPSWCC**  
**SCASBCC**  
**STC**  
**PSRL**  
**PSRAI**  
**PSRAW**  
**PSRAD/**  
**PSRADR/**  
**PANB**  
**PANL**  
**PANI**  
**FS**  
**FCCL**  
**FTAI**  
**FRAT**  
**FFX/CFXcc**  
**PCMPQB/PCMPQB**  
**PCMPQB/PCMPQB**  
**PCMPQB/PCMPQB**  
**PCMPQB/PCMPQB**  
**AND/ANDR/**  
**OR/ORR/CC**  
**XOR/XORR/CXORcc/CXORcc A**  
**NOT/CNOTcc A**

**PADD/PADDB/CPADDcc/CPADDBcc MMX,MMX**  
**PADD/PADDBR/CPADDcc/CPADDBRcc A,MMX**  
**PADDW/PADDWR/CPADDWcc/CPADDWRcc MMX,MMX**  
**PADDW/PADDWR/CPADDWcc/CPADDWRcc A,MMX**  
**PADDQ/PADDQR/CPADDQcc/CPADDQRcc MMX,MMX**  
**PADDQ/PADDQR/CPADDQcc/CPADDQRcc A,MMX**  
**PADDD/PADDDR/CPADDDcc/CPADDDRcc MMX,MMX**  
**PADDD/PADDDR/CPADDDcc/CPADDDRcc A,MMX**  
**PADDDQ/PADDDQR/CPADDDQcc/CPADDDQRcc MMX,MMX**  
**PADDDQ/PADDDQR/CPADDDQcc/CPADDDQRcc A,MMX**

0 0732-0732 1 DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWJANIA WYNIKU  
 0 0733-0738 6 DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWJANIA WYNIKU  
 0 0739-0739 1 DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWJANIA WYNIKU  
 0 0740-0745 6 DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWJANIA WYNIKU  
 0 0746-0746 1 DODAWANIE DWÓCH PODWÓJNYCH SŁÓW W TRYBIE ZAWJANIA WYNIKU

$MMX_B = MMX_B + MMX_B$   
 $A_B = A_B + MMX_B$   
 $MMX_W = MMX_W + MMX_W$   
 $A_W = A_W + MMX_W$   
 $MMX_D = MMX_D$

### 64 BITOWE KOPIOWANIE DANYCH

#### PMOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MMX,MMX

RELACJE:  $MMX \leftarrow MMX$      $MMX \rightarrow MMX$

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 dokumentacji: „Instrukcje warunkowe”.

Mnemonic instrukcji	Opis
==	Porównywanie czterech bajtów
!=	Porównywanie czterech bajtów
>	Porównywanie czterech bajtów
>=	Porównywanie czterech bajtów
<	Porównywanie czterech bajtów
<=	Porównywanie czterech bajtów
S>	Porównywanie czterech bajtów
S<	Porównywanie czterech bajtów
S<=	Porównywanie czterech bajtów
C	Porównywanie czterech bajtów

### 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	

$MMX_D = MMX_D + MMX_D$   
 $A_D = A_D + MMX_D$   
 $MMX_D = MMX_D + MMX_D$   
 $A_D = A_D + MMX_D$

# NOWY CEL

# KOMPUTER NA UKŁADACH TTL

- BTT/BTTR/CBTTcc/CBTTR
- BTS/BTSR/CBTScc/CBTSRc
- BTR/BTRR/CBTRcc/CBTRR
- BTC/BTCR/CBTCcc/CBTCF
- BSF/BSFR/CBSFcc/CBSFRc
- BSR/BSRR/CBSRcc/CBSRRc
- POPCNT/POPCNTR/CPOPCNTcc
- TEST/CTESTcc A,B
- SETCCA
- CALLR/CCALLRcc A,[B]
- RET/CRETcc
- SHRD/RSHRU
- INSD/CINSDcc A,B,C
- OUTSD/COUTSDcc A,B,C
- CMP5BCC/CMP5WCCcc
- SCASBCC
- STC
- MOV/MOVR/CMOVB
- XCHG/CXCHG
- BSWAP/CBSWAP
- PUSHU/CPUSHU
- PUSHS/CPUSHS
- PUSHB/CPUSHB
- PUSHW/CPUSHW
- PUSHD/CPUSHD
- POP
- POPS/CPOPS
- POPA/CPOPA
- POPB/CPOPB
- POPW/CPOPW
- POPD/CPOPD
- POPSA/CPOPSAcc
- IN/CINcc A,B
- OUT/OUTcc A,B
- PACKLB/CPACKLBcc/CPACKLMB
- PACKMB/CPACKMBcc/CPACKMBR
- AND/ANDR/CMOVB
- OR/ORR/CCOR
- XOR/XORR/CXORcc
- NOT/CNOTcc A

PADD/PADDB/CPADDcc/CPADDRcc MMX,MMX	0	0732-0732	1	DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWJANIA WYNIKU
PADB/PADDBR/CPADDBcc/CPADDBRcc A,MMX	0	0733-0738	6	DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWJANIA WYNIKU
PADDW/PADDWR/CPADDWcc/CPADDWRcc MMX,MMX	0	0739-0739	1	DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWJANIA WYNIKU
PADDD/PADDDR/CPADDDcc/CPADDDRcc A,MMX	0	0740-0745	6	DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWJANIA WYNIKU
PADDQ/PADDQR/CPADDQcc/CPADDQRcc MMX,MMX	0	0746-0746	1	DODAWANIE DWÓCH PODWÓJNYCH SŁÓW W TRYBIE ZAWJANIA WYNIKU

0	0732-0732	1	DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWJANIA WYNIKU
6	0733-0738	6	DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWJANIA WYNIKU
1	0739-0739	1	DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWJANIA WYNIKU
6	0740-0745	6	DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWJANIA WYNIKU
1	0746-0746	1	DODAWANIE DWÓCH PODWÓJNYCH SŁÓW W TRYBIE ZAWJANIA WYNIKU

MMX_B = MMX_B + MMX_B	A ← MMX_B + MMX_B = MMX_B
A_B = A_B + MMX_B	B = MMX_B
MMX_W = MMX_W + MMX_W	MMX_W + MMX_W = MMX_W
A_W = A_W + MMX_W	MMX_W
MMX_D = MMX_D	MMX_D = MMX_D

**64 BITOWE KOPIOWANIE DANYCH**

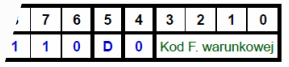
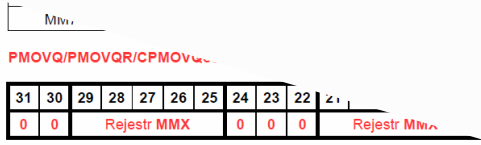
**MOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MMX,MMX**

**Przykład dla kontekstu- bez stronicowania pamięci, ze stronicowaniem pamięci, ze stronicowaniem pamięci z segmentacją:**

REJESTR MM3	WARTOŚCI PO WYKONANIU INSTRUKCJI: PMOVQ/CPMOVQcc MM3,MM6
45DDC405FF78ACC6h	89DDC045AAFF96EEh
WARTOŚCI PO WYKONANIU INSTRUKCJI: PMOVQ/CPMOVQRcc MM3,MM6	REJESTR MM6
REJESTR MM3	89DDC045AAFF96EEh
WARTOŚCI PO WYKONANIU INSTRUKCJI: PMOVQR/CPMOVQRcc MM3,MM6	REJESTR MM6
REJESTR MM3	89DDC045AAFF96EEh
WARTOŚCI PO WYKONANIU INSTRUKCJI: PMOVQ/CPMOVQcc MM3,MM6	REJESTR MM6
45DDC405FF78ACC6h	45DDC405FF78ACC6h

Przykłady w mnemonikach Asemblera:  
 PMOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MM4,MM4  
 PMOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MM2,MM3  
 PMOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MM3,MM5  
 PMOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MM6,MM7  
 PMOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MM1,MM6  
 PMOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MM0,MM1  
 PMOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MM2,MM2  
 PMOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MM7,MM4  
 PMOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MM3,MM5  
 PMOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MM3,MM6

Instrukcja dostępna w trybie nadzorczy i użytkownika.  
 Kod operacji dla instrukcji użytkownika i nadzorczy: **A58h, A5Ah**



MMX\_A ← MMX\_A + MMX\_A  
 MMX\_W ← MMX\_W + MMX\_W  
 MMX\_D ← MMX\_D + MMX\_D  
 MMX\_D > MMX\_D → MMX\_D  
 MMX\_D > MMX\_D → MMX\_D  
 MMX\_D > MMX\_D → MMX\_D

# NOWY CEL

# KOMPUTER NA UKŁADACH TTL

PADDB/PADDBR/CPADDBcc/CPADDBRcc MMX,MMX  
 PADDB/PADDBR/CPADDBcc/CPADDBRcc A,MMX  
 PADDW/PADDWR/CPADDWcc/CPADDWRcc MMX,MMX  
 PADDW/PADDWR/CPADDWcc/CPADDWRcc A,MMX

0 0732-0732 1 DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIĄJANIA WYNIKU  
 0 0733-0738 6 DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIĄJANIA WYNIKU  
 0 0739-0739 1 DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWIĄJANIA WYNIKU  
 0 0740-0745 6 DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWIĄJANIA WYNIKU  
 0 0746-0746 1 DODAWANIE DWÓCH PODWOJNYCH SŁÓW W TRYBIE ZAWIĄJANIA WYNIKU

MMX\_B = MMX\_B + MMX\_B  
 A\_B = A\_B + MMX\_B  
 MMX\_W = MMX\_W + MMX\_W  
 A\_W = A\_W + MMX\_W  
 MMX\_D = MMX\_D + MMX\_D

**64 BITOWE KOPIOWANIE DANYCH**  
**MOVQ/PMOVQR/CPMOVQcc/CPMOVQRcc MMX,MMX**  
 \*MMX ← MMX MMX → MMX

Wartość jest do 64 bitowej zawartości argumentu docelowego MMX.  
 Operacja 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”.

**Przykład dla kontekstu bez stronicowania pamięci, ze stronicowaniem pamięci, z wartością początkową**

REJESTR MM3	45DDC405FF78ACC6h
WARTOŚCI PO WYKONANIU INSTRUKCJI	89DDC045AFFF96EEh

## PADDB/PADDBR/CPADDBcc/CPADDBRcc MMX,MMX

### RELACJE: MMX\_B=MMX\_B+MMX\_B MMX\_B+MMX\_B=MMX\_B

Instrukcje działają w następujący sposób:  
 Za wartość 64 bitowego argumentu źródłowego MMX podzieloną na osiem obszarów ośmiobitowych (bajtów) sumowana jest z zawartością 64 bitowego argumentu docelowego MMX podzielonego na osiem obszarów ośmiobitowych (bajtów). Wynik umieszczany jest w 64 bitowym argumencie docelowym MMX podzielonym na osiem obszarów ośmiobitowych (bajtów).  
 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”.

Mnemonic 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=0	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=0	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≠0	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≠0	A≤B lub B≤A A≠B lub B≠A	1010

**64 BITOWE KOPIOWANIE DANYCH**  
**CPMOVQcc MM3,MM6**  
**REJESTR MM6**  
**89DDC045AFFF96EEh**

**CPMOVQRcc MM3,MM6**  
**REJESTR MM6**  
**89DDC045AFFF96EEh**

**CPMOVQcc MM3,MM6**  
**REJESTR MM6**  
**45DDC405FF78ACC6h**



ZMIENIENIE CZY SĄ WIĘKSZE  
 ZMIENIENIE CZY SĄ MIEJSZE  
 ZMIENIENIE CZY SĄ WIĘKSZE  
 ZMIENIENIE CZY SĄ MIEJSZE

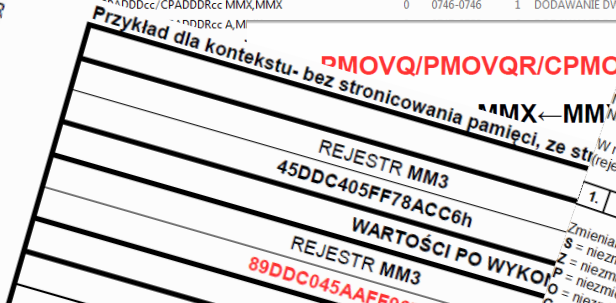
MMX\_A, MMX\_B, MMX\_C, MMX\_D, MMX\_W, MMX\_X, MMX\_Y, MMX\_Z  
 MMX\_D > MMX\_W → Min\_X\_W  
 MMX\_D > MMX\_D → MMX\_D  
 MMX\_D > A\_D → MMX\_D

# NOWY CEL KOMPUTER NA UKŁADACH TTL

PADDB/PADDBR/CPADDBcc/CPADDBRcc MMX,MMX  
 PADDP/PADDBR/CPADDBcc/CPADDBRcc A,MMX  
 PADDW/PADDWR/CPADDWcc/CPADDWRcc MMX,MMX  
 PADDW/PADDWR/CPADDWcc/CPADDWRcc A,MMX  
 PADDW/PADDWR/CPADDWcc/CPADDWRcc MMX,MMX

0 0732-0732 1 DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU  
 0 0733-0738 6 DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU  
 0 0739-0739 1 DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWIJANIA WYNIKU  
 0 0740-0745 6 DODAWANIE CZTERECH SŁÓW W TRYBIE ZAWIJANIA WYNIKU  
 0 0746-0746 1 DODAWANIE DWÓCH SŁÓW W TRYBIE ZAWIJANIA WYNIKU

$MMX_B = MMX_B + MMX_B$   
 $A_B = A_B + MMX_B$   
 $MMX_W = MMX_W + MMX_W$   
 $A_W = A_W + MMX_W$   
 $MMX_D = MMX_D$

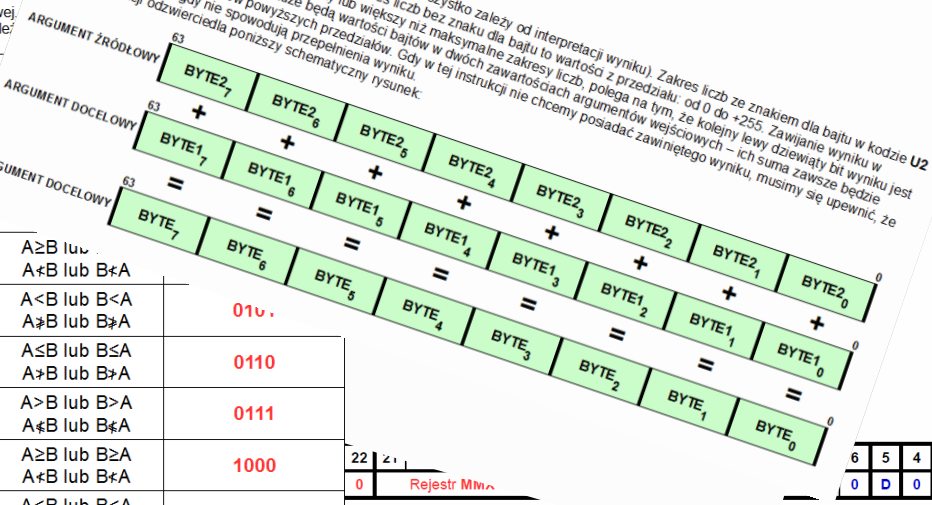


DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU  
**PADDB/PADDBR/CPADDBcc/CPADDBRcc MMX,MMX**

**RELACJE:  $MMX_B = MMX_B + MMX_B$   $MMX_B + MMX_B = MMX_B$**

Instrukcje działają w następujący sposób:  
 Zawartość 64 bitowego argumentu źródłowego MMX podzielona na osiem obszarów ośmiobitowych (baity). Argument docelowy MMX podzielony na osiem obszarów ośmiobitowych (bajty). Wynik instrukcji MMX podzielony na osiem obszarów ośmiobitowych (bajty).  
 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 podręczniku.

Mnemonic instrukcji cc	Opis	Warunek
==	Wykonanie, jeśli równe/jeśli zero (argumenty uniwersalne)	Z=1
!=	Wykonanie, jeśli różne/jeśli różne od zera (argumenty uniwersalne)	Z=0
>	Wykonanie, jeśli powyżej/jeśli nie poniżej i nie równe (argumenty dotyczące liczb bez znaku)	C=0 i Z=0
>=	Wykonanie, jeśli powyżej lub równe/jeśli nie poniżej (argumenty dotyczące liczb bez znaku)	C=0
<	Wykonanie, jeśli poniżej/jeśli nie powyżej i nie równe (argumenty dotyczące liczb bez znaku)	C=1
<=	Wykonanie, jeśli poniżej lub równe/jeśli nie powyżej (argumenty dotyczące liczb bez znaku)	Z=1 lub C=1
>S	Wykonanie, jeśli większe/jeśli nie mniejsze i nie równe (argumenty dotyczące liczb ze znakiem)	Z=0 i S=0
>=S	Wykonanie, jeśli większe lub równe/jeśli nie mniejsze (argumenty dotyczące liczb ze znakiem)	S=0
<S	Wykonanie, jeśli mniejsze/jeśli nie większe i nie równe (argumenty dotyczące liczb ze znakiem)	S=0
<=S	Wykonanie, jeśli mniejsze lub równe/jeśli nie większe (argumenty dotyczące liczb ze znakiem)	Z=1 lub S=0



ZMIENIENIE LICZBY SA WIĘKSZE  
 ZMIENIENIE CZY SA WIĘKSZE  
 NIMI SŁÓWAMI I SPRAWDZENIE CZY SA WIĘKSZE  
 NIMI SŁÓWAMI I SPRAWDZENIE CZY SA WIĘKSZE

$MMX_D = MMX_D + MMX_D$   
 $A_D = A_D + MMX_D$   
 $MMX_D = MMX_D$   
 $A_D = A_D + MMX_D$



# NOWY CEL

# KOMPUTER NA UKŁADACH TTL

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**

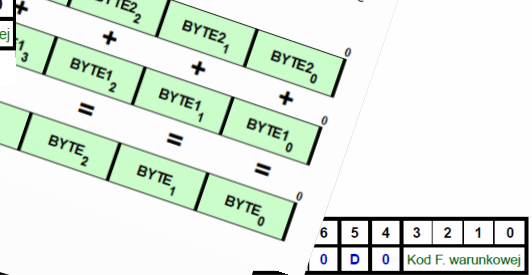
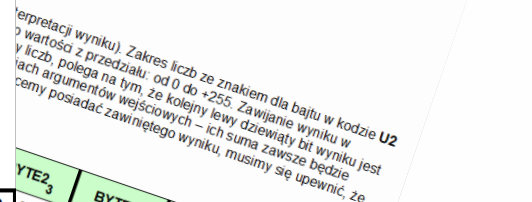
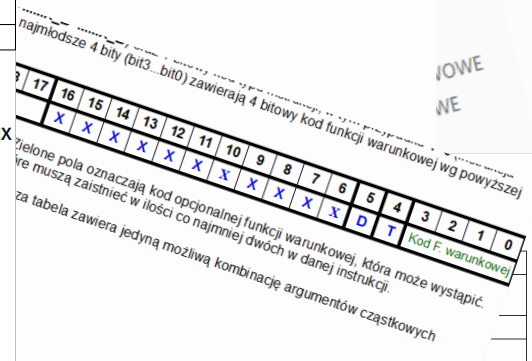
Rejestr MMX	Kod rejestru MMX	Rejestr MMX	Kod rejestru MMX
MM0	10111	MM0	10111
MM1	11000	MM1	11000
MM2	11001	MM2	11001
MM3	11010	MM3	11010
MM4	11011	MM4	11011
MM5	11100	MM5	11100
MM6	11101	MM6	11101
MM7	11110	MM7	11110

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					

Symbol	Opis	Warunek	Wynik
<	(argumenty dotyczące liczb bez znaku)		$A \neq B$ lub $B \neq A$
<=	Wykonanie, jeśli poniżej lub równe/jeśli nie powyżej (argumenty dotyczące liczb bez znaku)	$Z=1$ lub $C=1$	$A \leq B$ lub $B \leq A$ $A \geq B$ lub $B \geq A$
>	Wykonanie, jeśli większe/jeśli nie mniejsze i nie równe (argumenty dotyczące liczb ze znakiem)	$Z=0$ i $S=0$	$A > B$ lub $B > A$ $A \neq B$ lub $B \neq A$
>=	Wykonanie, jeśli większe lub równe/jeśli nie mniejsze (argumenty dotyczące liczb ze znakiem)	$S=0$	$A \geq B$ lub $B \geq A$ $A \neq B$ lub $B \neq A$
<>	Wykonanie, jeśli mniejsze/jeśli nie większe i nie równe (argumenty dotyczące liczb ze znakiem)	$S \neq 0$	$A < B$ lub $B < A$ $A \neq B$ lub $B \neq A$
<=>	Wykonanie, jeśli mniejsze lub równe/jeśli nie większe (argumenty dotyczące liczb ze znakiem)	$Z=1$ lub $S \neq 0$	$A \leq B$ lub $B \leq A$ $A \neq B$ lub $B \neq A$

MMX\_B = MMX\_B + MMX\_B  
 $A_B = A_B + MMX_B$   
 $MMX_W = MMX_W + MMX_W$   
 $MMX_D = MMX_D$



MMX\_D = MMX\_D + MMX\_D  
 $MMX_D > A_D - MMX_D$   
 $MMX_D > A_D - MMX_D$

# NOWY CEL

# KOMPUTER NA UKŁADACH TTL

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

UPRZYWILEJOWANIA:	KONTEKST OBSŁUGI PAMIĘCI OPERACJI	
	BEZ STRONICOWANIA PAMIĘCI	STRONICOWANIE PAMIĘCI
TRYB UŻYTKOWNIKA	✗	
TRYB NADZORCY		✓

Instrukcja jest jednowierszowa (4 bajtowa).

Zawartość 64 bitowego rejestru źródłowego MMX podzielona na osiem obszarów 8 bajtowych, jest kopiowana do docelowego MMX podzielonego na osiem obszarów 8 bajtowych (bajty).

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

### REJESTR MMX\_B=REJESTR MM\_A REJESTR MMX\_B+REJESTR MM\_A

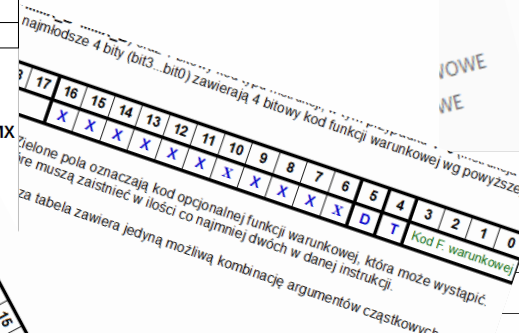
Rejestr MMX	Kod rejestru MMX	Rejestr MMX	Kod
MM0	10111	MM0	
MM1	11000	MM1	11
MM2	11001	MM2	110
MM3	11010	MM3	11010
MM4	11011	MM4	11011
MM5	11100	MM5	11100
MM6	11101	MM6	11101
MM7	11110	MM7	11110

### PADDB/PADDBR/CPADDBcc/CPADDBRcc MMX,MMX

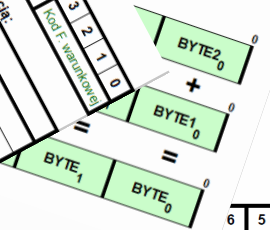
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14
0	0	Rejestr MMX				0	0	0	Rejestr MMX				0	1	0		

Symbol	Opis	Warunek	Wynik
<	(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 A ≥ B lub B ≥ A
>	Wykonanie, jeśli większe/jeśli nie mniejsze i nie równe (argumenty dotyczące liczb ze znakiem)	Z=0 i S=0	A > B lub B > A A < B lub B < A
>=	Wykonanie, jeśli większe lub równe/jeśli nie mniejsze (argumenty dotyczące liczb ze znakiem)	S=0	A ≥ B lub B ≥ A A < B lub B < A
<=	Wykonanie, jeśli mniejsze/jeśli nie większe i nie równe (argumenty dotyczące liczb ze znakiem)	S≠0	A < B lub B < A A ≥ B lub B ≥ A
<=>	Wykonanie, jeśli mniejsze lub równe/jeśli nie większe (argumenty dotyczące liczb ze znakiem)	Z=1 lub S≠0	A ≤ B lub B ≤ A A ≥ B lub B ≥ A

MMX\_B = MMX\_B + MMX\_A  
A\_B = A\_B + MMX\_B  
MMX\_W = MMX\_W + MMX\_A  
A\_W = A\_W + MMX\_A  
MMX\_D = MMX\_A



zakres liczb ze znakiem dla bajtu w kodzie U2 od 0 do +255. Zawijanie wyniku w kolejny lewy dziewiąty bit wyniku będzie...  
Kod F. warunkowej



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

# NOWY CEL

# KOMPUTER NA UKŁADACH TTL

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

**FADD/FADDR/CFADDcc/CFADDRcc STX,STX**  
SUMOWANIE DWOCH 76 BITOWYCH LICZB ZMIENNOPRZECINKOWYCH

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

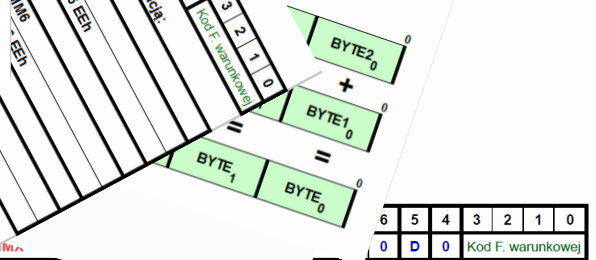
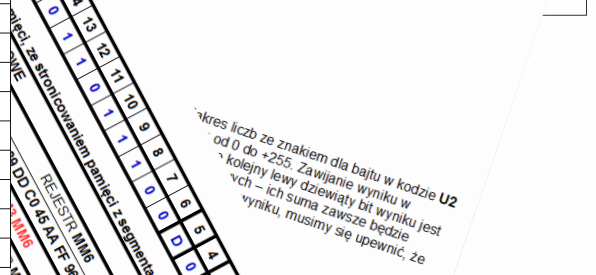
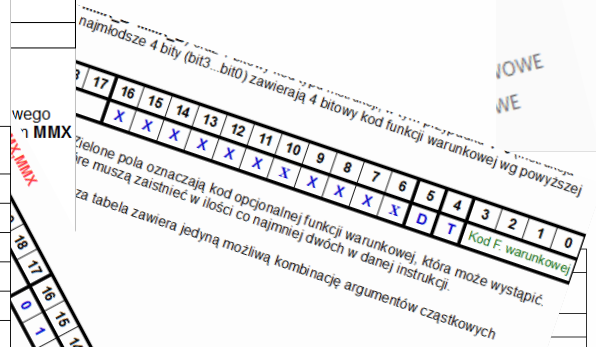
Instrukcje działają w następujący sposób:  
76 bitowa zawartość argumentu źródłowego STX sumowana jest z 76 bitową zawartością argumentu docelowego STX. Wynik umieszczany jest w 76 bitowym argumencie docelowym STX.  
Operacje te odbywają się tradycyjnie lub po spełnieniu jednego z 16 argumentów zmiennoprzecinkowej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

Mnemonic instrukcji cc	Opis	Flagi warunk:				Zależność	Kod Funkcji warunkowej
		C3	C2	C1	C0		
<b>==</b>	Wykonanie, jeśli równe	0	0	0	1	A=B lub B=A	0001
<b>&gt;</b>	Wykonanie, jeśli większe	0	0	1	0	A>B lub B>A	0010
<b>&lt;</b>	Wykonanie, jeśli mniejsze	0	0	1	1	A<B lub B<A	0011
<b>+SN</b>	Wykonanie, jeśli +SNaN	0	1	0	0	A=+SNaN	0100
<b>-SN</b>	Wykonanie, jeśli -SNaN	0	1	0	1	A=-SNaN	0101
<b>+QN</b>	Wykonanie, jeśli +QNaN	0	1	1	0	A=+QNaN	0110
<b>-QN</b>	Wykonanie, jeśli -QNaN	0	1	1	1	A=-QNaN	0111
<b>+I</b>	Wykonanie, jeśli dodatnia nieskończona liczba	1	0	0	0	A=+∞	1000
<b>-I</b>	Wykonanie, jeśli ujemna nieskończona liczba	1	0	0	1	A=-∞	1001
<b>+N</b>	Wykonanie, jeśli dodatnia znormalizowana liczba	1	0	1	0	A=+N	1010
<b>-N</b>	Wykonanie, jeśli ujemna znormalizowana liczba	1	0	1	1	A=-N	1011
<b>+D</b>	Wykonanie, jeśli dodatnia zdenormalizowana liczba	1	1	0	0	A=+D	1100
<b>-D</b>	Wykonanie, jeśli ujemna zdenormalizowana liczba	1	1	0	1	A=-D	1101
<b>+Z</b>	Wykonanie, jeśli dodatnie zero	1	1	1	0	A=+0	1110
<b>-Z</b>	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 (argumenty dotyczące liczb bez znaku)

			A≠B lub B≠A
<b>&lt;=</b>	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
<b>S&gt;</b>	Wykonanie, jeśli większe/jeśli nie mniejsze i nie równe (argumenty dotyczące liczb ze znakiem)	Z=0 i S=0	A>B lub B>A A≠B lub B≠A
<b>S&gt;=</b>	Wykonanie, jeśli większe lub równe/jeśli nie mniejsze (argumenty dotyczące liczb ze znakiem)	S=0	A≥B lub B≥A A≠B lub B≠A
<b>S&lt;</b>	Wykonanie, jeśli mniejsze/jeśli nie większe i nie równe (argumenty dotyczące liczb ze znakiem)	S≠0	A<B lub B<A A≠B lub B≠A
<b>S&lt;=</b>	Wykonanie, jeśli mniejsze lub równe/jeśli nie większe (argumenty dotyczące liczb ze znakiem)	Z=1 lub S≠0	A≤B lub B≤A A≠B lub B≠A

MMX\_B = MMX\_B + MMX\_B  
A\_B = A\_B + MMX\_B  
MMX\_W = MMX\_W + MMX\_W  
A\_W = A\_W + MMX\_W  
MMX\_D = MMX\_D + MMX\_D



MMX\_D = MMX\_D + MMX\_D  
A\_D = A\_D + MMX\_D  
MMX\_D > A\_D → MMX\_D  
MMX\_D > A\_D → MMX\_D

# NOWY CEL

# KOMPUTER NA UKŁADACH TTL

**1. PADDB/PADDBR/CPADDBcc/CPADDBRcc MMX, MMX**  
DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU

**FADD/FADDR/CFADDcc/CFADDRcc STX, STX**  
SUMOWANIE DWOCH 76 BITOWYCH LICZB ZMIENNOPRZECINKOWYCH

**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.  
Operacje te odbywają się tradycyjnie lub po spełnieniu jednego z 16 argumentów zmiennoprzecinkowych.  
Dokładny opis zależności argumentów warunkowych wraz z praktycznymi przykładami można znaleźć w podręczniku.

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

**Kodem operacji dla instrukcji standardowej jest:** 11 bitowy kod instrukcji (gdzie D=0 to STX=STX+STX, a D=1 to STX+STX=STX) oraz 1 bitowy kod wystąpienia błędów (Z=1 lub S=0).  
Razem to 13 bitów (bit16...bit4). Pierwsze najmłodsze 4 bity (bit3...bit0) zawierają 4 bity nieaktywne i musi zawierać cztery logiczne zera (argumenty dotyczące liczb bez znaku).

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

MMX\_B = MMX\_B + MMX\_B  
A\_B = A\_B + MMX\_B  
MMX\_W = MMX\_W + MMX\_W  
A\_W = A\_W + MMX\_W  
MMX\_D = MMX\_D + MMX\_D

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

**WAŻNE!** Instrukcja jest w stanie sumować liczby, których różnica wykładników nie przekracza 63. Gdy ta różnica jest większa – otrzymywana wartość 0 w operandzie docelowym będzie nieprecyzyjna, będzie identyczna z wartością zdenormalizowaną (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 umożliwiły. Rejestry, które zawierają mantysę liczb zmiennoprzecinkowych o rozszerzonej precyzji są 64 bitowe.

**WAŻNE!** Instrukcja w pierwszej kolejności dokonuje analizy operandu źródłowego i docelowego pod kątem występowania wyjątków. Ma to oczywiście związek ze skończoną ilością bitów w mantysie liczb w formacie rozszerzonej precyzji (nie można przesunąć mniejszej z dwóch wartości ponad granicę fizycznego rejestru).

**#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:  
1. Nie liczba znajduje się tylko w operandzie źródłowym – w tym przypadku nienaruszona zostanie wartość docelowego i instrukcja zakończy działanie.  
2. Nie liczba znajduje się w operandzie docelowym i instrukcja zakończy działanie.

Wyjątek **#P** który jest zamaskowany istnieją trzy możliwości reakcji sprzętowej:  
1. Nie liczba znajduje się tylko w operandzie źródłowym – w tym przypadku nienaruszona zostanie wartość docelowego i instrukcja zakończy działanie.  
2. Nie liczba znajduje się w operandzie docelowym i instrukcja zakończy działanie.  
3. Nie liczba znajduje się w obu operandach – nie zostaną podjęte żadne działania, Nie liczba pozostanie nienaruszona w operandzie docelowym i instrukcja zakończy działanie.

3	2	1	0
Kod F. warunkowej			







# NOWY CEL

## KOMPUTER NA UKŁADACH TTL

**1. PADD/FADD/RIC/CFADDR/CFADDRcc STX,STX**

Gdy wystąpi wyjątek #P, który jest zamaskowany, istnieją trzy możliwości reakcji sprzętowej:  
 W operandach źródłowym i docelowym znajdują się liczby znormalizowane i znormalizowana. Należy również przyjąć, iż we wszystkich operandach źródłowych i docelowych znajduje się liczba znormalizowana. Należy również przyjąć, iż we wszystkich operandach źródłowych i docelowych znajduje się nieskończoność o nie liczbę QNaN. Zaw  
 W dalszej kolejności d  
 Operandy docelowego  
 Jeśli te nie są maskowa

**REJESTR STX=RE REJESTR STX+RE**

Instrukcja jest jednowerszowa (4  
 bitowa zawartość rejestru źród  
 Wymnik operacji przeliczony w  
 STX = ST0 ST1 ST2 ST3 ST4 ST  
 ST5 ST6 ST7

**1. FADD/FADDR/CFADDR/CFADDRcc STX,STX**

**UPRZEWILEJOWANIA:**  
**TRYB UŻYTKOWNIKA**  
**TRYB NADZORCY**

liczba zdenormalizowana i  
 v operandzie docelowym zawsze  
 w znajduje się dodatnia/ujemna  
 ndzie docelowym zawsze znajdzie  
 wsze zastaniemy  
**0000000000000000000000000000**  
 ępnie dokonwane jest  
 PETLA BEZWZGLĘDNA

**RELACJE: ○B=B-1 ○LP←A**

Instrukcje działają w następujący sposób, składają się z dwóch etapów:  
 Zawartość argumentu B zostaje pomniejszona o jeden.  
 Zawartość argumentu A zostaje załadowana do licznika programu LP.  
 Dokładny opis zależności argumentów warunkowych wraz z praktycznymi przykładami można znaleźć w dokumencie „Instrukcje warunkowe”.

Flagi warunku:	C2	C1	C0
1	0	0	0
1	0	0	0

Mnemonic instrukcji CC	Opis	Warunek	Zależność	Kod Funkcji warunkowej
Z=1	Przerwane, jeśli równe/jeśli zero (argumenty uniwersalne)	Z=1	A=B A-B=0	0001
Z=0	Przerwane, jeśli różne/jeśli różne od zera (argumenty uniwersalne)	Z=0	A≠B A-B≠0	0010
C=0 i Z=0	Przerwane, jeśli powyżej/jeśli nie poniżej i nie równe (argumenty dotyczące liczb bez znaku)	C=0 i Z=0	A>B A<B	0011
C=0	Przerwane, jeśli powyżej lub równej/jeśli nie poniżej (argumenty dotyczące liczb bez znaku)	C=0	A<B A≠B	0100
C=1	Przerwane, jeśli poniżej/jeśli nie powyżej i nie równe (argumenty dotyczące liczb bez znaku)	C=1	A>B A≠B	0101
Z=1 lub C=1	Przerwane, jeśli poniżej lub równej/jeśli nie powyżej (argumenty dotyczące liczb bez znaku)	Z=1 lub C=1	A≤B A≥B	0110
Z=0 i S=0	Przerwane, jeśli większe/jeśli nie mniejsze i nie równe (argumenty dotyczące liczb ze znakiem)	Z=0 i S=0	A≥B A<B	1000
S=0	Przerwane, jeśli większe lub równej/jeśli nie mniejsze (argumenty dotyczące liczb ze znakiem)	S=0	A<B A≠B	1001
S≠0	Przerwane, jeśli mniejsze/jeśli nie większe i nie równe (argumenty dotyczące liczb ze znakiem)	S≠0	A>B A≠B	1010
Z=1 lub S≠0	Przerwane, jeśli mniejsze lub równej/jeśli nie większe (argumenty dotyczące liczb ze znakiem)	Z=1 lub S≠0	A≤B A≥B	1011
C=1	Przerwane, jeśli było przemieszenie	C=1		0101

**OPERAND ŹRÓDŁOWY:**

**UWAGI:**  
 ±D – Zdenormalizowana liczba  
 ±F – Skończona liczba zmienn

Mnemonic instrukcji	Opis	Warunek	Zależność	Kod Funkcji warunkowej
Z=1 lub S≠0	Wykonanie, jeśli większe lub równe (argumenty dotyczące liczb bez znaku)	Z=1 lub S≠0	A≤B lub B≤A A≥B lub B≥A	1
S=0	Wykonanie, jeśli mniejsze/jeśli nie większe (argumenty dotyczące liczb bez znaku)	S=0		
S≠0	Wykonanie, jeśli mniejsze lub równej/jeśli nie większe (argumenty dotyczące liczb ze znakiem)	S≠0		

**OPERA**

**3 2 1 0**  
 Kod F. warunkowej

**WYJĄTKI: BRAK**

**WYJĄTKI: BRAK**



# NOWY CEL

# KOMPUTER NA UKŁADACH TTL

0322-0732 DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIĄZANIA WYNIKU

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

W warunkach źródłowym i docelowym zmierzania się linch, znormalizowane lub...  
**WAŻNE!** Instrukcja LOOPA służy do tworzenia pętli, w której licznikiem jest rejestr źródłowy R. Każde wykonanie tej instrukcji, dekrementuje wartość 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 wartością 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 **bezporednie** 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 instrukcje mogą być wielowierszowe (jedno, dwu, trzy lub czterowierszowe). Należy przy obliczaniu docelowego adresu skoku koniecznie uwzględnić ten fakt! W przeciwnym wypadku skok spowoduje załadowanie niewskazanej 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** pochłoniąłby 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 x 4B) jest to adres LSB. Pozostałe sześć z nich (L22..L7) zawiera numer logicznej strony, który jest adresem MSB. Operand 23 bitowy spowoduje ewentualny skok do argumentu 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 pochłoniąłby 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 x 4B) jest to adres LSB. Pozostałe sześć 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 liczba  
 ±F – Skończona liczba zmiennoprzecinkowa

S>= Wykonanie, jeśli większe lub równe (argumenty dotyczące liczb)

S< Wykonanie, jeśli mniejsze/jeśli nie (argumenty dotyczące liczb)

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

9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0

S>=	Przerwane, jeśli większe lub równe/jeśli nie (argumenty dotyczące liczb ze znakiem)	S=0
S<	Przerwane, jeśli mniejsze/jeśli nie większe i nie równe (argumenty dotyczące liczb ze znakiem)	S≠0
S<=	Przerwane, jeśli mniejsze lub równe/jeśli nie większe (argumenty dotyczące liczb ze znakiem)	Z=1 lub S≠0
C	Przerwane, jeśli było przeniesienie	C=1

1. FADDI/FADDRI/C

liczba zdenormalizowana i w operandzie docelowym zawsze w znajduje się dodatnia/ujemna ndzie docelowym zawsze znajdzie wsze zastaniemy 000000000000000000000000.

**A**

ę z dwóch etapów: bit. kilka programu LP. r wraz z praktycznymi przykładami można znaleźć w dokumencie „Instrukcje warunkowe”.

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

Flagi warunku:

C2	C1	C0
1	0	0

10000000000000000000000000000000
8513 x 10 <sup>-308</sup>
00000000000000000000000000000000
513 x 10 <sup>-308</sup>
WYJĄTKI: BRAK
100000011010010000000000000000
548 x 2 <sup>49</sup>
00000110000000000000000000000000
κ 2 <sup>-1022</sup>
κ 10 <sup>-309</sup>
00001100000000000000000000000000
2 <sup>-1022</sup>
10 <sup>-309</sup>

3	2	1	0
0	0	0	0

Kod F. warunkowej

- BTT/BTR
- BTS/BTSB
- BTR/BTRB
- BTC/BTCR
- BSF/BSFR
- BSR/BSRR
- POPCNT/POPCNTB
- TEST/CTEST
- SETCC
- CALLR/CCALLR
- RET/CRET
- SHRD/RSHR
- INSD/CINSD
- OUTSD/COU
- CMP
- SCAS
- STC
- PSRL
- PSRAI
- PSRAW
- PSRAD
- PSRADR
- PAN
- PANL
- FS
- FPC
- FTAI
- FRATA
- FFX
- PCMPEQB
- PCMPEQB
- PCMPEQB
- AND/ORR
- XOR/XOR
- NOT
- PCMPG
- PCMPGT

# NOWY CEL

# KOMPUTER NA UKŁADACH TTL

0732-0732 DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIĄZANIA WYNIKU

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

W warunkach źródłowym i docelowym zmierzania się linchv znana liżwane lub...  
**WAŻNE!** Instrukcja **LOOPA** służy do tworzenia pętli, w której licznikiem jest rejestr źródłowy **R**. Każde wykonanie tej instrukcji, dekrementuje wartość 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.  
 Nie ma możliwości zorganizowania **bezpośrednio** pętli, która będzie „skakać” po różnych segmentach w kontekście stronicowania pamięci. Segmentacja. Aktywnymi segmentami dla wykonywania instrukcji w tym kontekście mogą być tylko i wyłącznie segmenty: CS lub FS (wykonywalne).

**UWAGA!** W programowaniu najbardziej niskopoziomowym czyli w języku maszynowym należy zwracać bacznie uwagę na wielowierszowe (jedno, dwu, trzy lub czterowierszowe). Należy przy obliczaniu docelowego adresu skoku w przeciwnym wypadku skok spowoduje załadowanie niewskazanej instrukcji lub co gorsza spowoduje wielowierszowej co w takim przypadku wywoła błąd.

**WAŻNE!** Powyższe instrukcje odnoszą się do następujących procedur:

**Kontekst bez stronicowania pamięci** – zawartość operandu spowoduje załadowanie pamięci RAM do 32MB (33.554.432R).  
 Operand 23 bitowy spowoduje ewentualny skok do każdego z 8 segmentów.  
**PA** – instrukcja zwróci natychmiastowy wyjątek.

**Kontekst ze stronicowaniem pamięci**  
 W tym przypadku logicznych stron może być do 23.  
**aktywny proces** pochłonie całą pamięć RAM.  
 Instrukcja logiczne opiera się na 23 bitach. Pozostałe szesnastie z nich (L22...L7) są argumentem przydzielonej logicznej strony z prz. MMJ logiczny adres (logiczna strona) w **aktywnym** procesie przydzielonych stron w **aktywnym** procesie (dokładnym numerze).

**Kontekst ze stronicowaniem pamięci z segmentem**  
 Instrukcja jest jednowierszowa (4 bajtowa). Zawartość rejestru źródłowego **R** zostaje pomniejszona o jeden. Zawartość rejestru docelowego **R** zostaje załadowana do licznika programu **LP** pod warunkiem, że zawartość rejestru docelowego **R** jest różna od zera (**R ≠ 0**) lub ewentualnie jeden z 16 argumentów funkcji warunkowej nie został jeszcze spełniony (**WV**).

## REJESTR=REJESTR-1 LICZNIK PROGRAMU ← REJESTR

Rejestr R	Kod rejestru R	Rodzaj Typ rejestru R	Rejestr R	Kod rejestru R
A0	00000	0	A0	00000
A1	00001	0	A1	00001
A2	00010	0	A2	00010
A3	00011	0	A3	00011
A4	00100	0	A4	00100
A5	00101	0	A5	00101
A6	00110	0	A6	00110
A7	00111	0	A7	00111
D0	01000	0	D0	01000
D1	01001	0	D1	01001
D2	01010	0	D2	01010
D3	01011	0	D3	01011
D4	01100	0	D4	01100

**UWAGI:**  
 ±D – Zdenormalizowana liczba  
 ±F – Skończona liczba zmiennoprzecinkowa

Wykonanie, jeśli większe lub równe (argumenty dotyczące liczb)

Wykonanie, jeśli mniejsze/jeśli nie (argumenty dotyczące liczb)

Wykonanie, jeśli mniejsze lub równe/jeśli nie większe (argumenty dotyczące liczb ze znakiem)

liczba zdenormalizowana i w operandzie docelowym zawsze

się dodatnia/ujemna liczbą docelowym zawsze znajdzie

niemy 0000000000000000.

w dokumencie „Instrukcje warunkowe”.

Zależność	Kod Funkcji warunkowej
A=B	0001
A≠B	0010
A≠0	0011
B≠0	0100
A<B	0101
A>B	0110
A<=B	0111
A>=B	1000
A<A	1001
A<=B	1010
A>=B	0101

3	2	1	0
Kod F. warunkowej			

- BTT/BTR
- BTS/BTS
- BTR/BTR
- BTC/BTC
- BSF/BSFR
- BSR/BSRR
- POPCNT/POPCNT
- TEST/CTEST
- SETCCA
- CALLR/CCALLR
- RET/CRET
- SHRD/RSHR
- INSD/CINSD
- OUTSD/OUTSD
- CMPBCC/CMPBCC
- SCASBCC
- STC
- XCHG
- PSRL
- PSRAI
- PSRAW
- PSRAD
- PSRADR
- PANL
- PANL
- PANL
- FS
- FTAI
- FRATA
- FFX/CFX
- PCMPEQB/PCMPEQB
- PCMPEQB/PCMPEQB
- PCMPEQB/PCMPEQB
- PCMPEQB/PCMPEQB
- AND/AND
- OR/OR
- XOR/XOR
- NOT
- PC
- PCM
- PCMPG
- PCMPGT

WYJĄTKI: BRAK

# KOMPUTER NA UKŁADACH TTL

# KOMPUTER NA UKŁADACH TTL

0732-0732 DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIĄZANIA WYNIKU

MMX\_B = MMX\_B - MMX\_B

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	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	FFDF17h	FF45ABh
WARTOŚCI PO WYKONANIU INSTRUKCJI: LOOPA/LOOPACC [A4],D3			
LICZNIK PROGRAMU PC	ZAWARTOŚĆ KOMÓRKI PAMIĘCI [A]	ADRES KOMÓRKI PAMIĘCI [A]	REJESTR D3
0B00AD52h	0B00AD52h	FFDF17h	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	FFDF17h	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	FFDF17h	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 [CS:A4],J3  
 LOOPA/LOOPACC [SS:SPS],SPU  
 LOOPA/LOOPACC [SS:SPS],SS

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

- W programach źródłowym i docelowym znikają się linijki z numerami linii.
- Instrukcja LOOPA służy do tworzenia pętli, w której licznikiem jest rejestr źródłowy R. Każde wywołanie instrukcji o jeden oraz sprawdza czy licznik nie osiągnął wartości zero. Jeżeli licznik jest wyzerowany, to programista musi wyzerować licznik przed wykonaniem instrukcji. W przeciwnym razie pętla jest powtarzana, następuje skok do pierwszej instrukcji w pętli zgodny z instrukcją. Instrukcja LOOPACC oprócz sprawdzania wartości licznika, wykonuje jeszcze sprawdzenie warunków. Instrukcja ta zostanie przerwana gdy wartość licznika osiągnie zero lub gdy zadany warunek zostanie spełniony. Nie ma możliwości zorganizowania bezpośrednio pętli, która będzie „skakać” po różnych segmentach w pamięci. Aktywnymi segmentami dla wykonywania instrukcji w tym kontekście mogą być tylko i wyłącznie segmenty wywołane).

**UWAGA!** W programowaniu najbardziej niskopoziomowym czyli w języku maszynowym należy zwracać uwagę na wielowierszowe (jedno, dwa, trzy lub czterowierszowe). Należy przy obliczaniu docelowego adresu skoku instrukcji przewidzieć wielowierszowość co w takim przypadku wywoła błąd.

**WAŻNE!** Powyższe instrukcje odnoszą się do następujących procedur:

**Kontekst bez stronicowania pamięci** – zawartość operandu spowoduje skok do adresu podanego w instrukcji. Operandy 23 bitowe spowodują ewentualny skok do każdego z 8 segmentów. Instrukcja jest pierwszym fizycznym adresem w pamięci RAM. Instrukcja zwróci natychmiastowy wyjątek #P.

**Kontekst ze stronicowaniem pamięci** – w tym przypadku logiczne strony mogą być adresowane logicznie. Instrukcja jest pierwszym logicznym adresem (logiczna strona) w aktywnym procesie. Operandy 23 bitowe spowodują ewentualny skok do każdej z 8 logicznych stron w aktywnym procesie. Instrukcja zwróci natychmiastowy wyjątek #P.

**Kontekst ze stronicowaniem pamięci z segmentacją** – w tym przypadku logiczne strony mogą być adresowane logicznie. Instrukcja jest pierwszym logicznym adresem (logiczna strona) w aktywnym procesie. Operandy 23 bitowe spowodują ewentualny skok do każdej z 8 logicznych stron w aktywnym procesie. Instrukcja zwróci natychmiastowy wyjątek #P.

**REJESTR=REJESTR-1**  
**LICZNIK PROGRAMU**

Rejestr R	Kod rejestru R	Rodzaj Typ rejestru R
A0	00000	A2
A1	00001	A3
A2	00010	A4
A3	00011	A5
A4	00100	A6
A5	00101	A7
A6	00110	D0
A7	00111	D1
D0	01000	D2
D1	01001	D3
D2	01010	D4
D3	01011	DF
D4	01100	DF

**UWAGI:**  
 ±D – Zdenormalizowana liczba  
 ±F – Skończona liczba zmiennoprzecinkowa  
 NaN – Nieznana liczba zmiennoprzecinkowa

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

0	1000
A < B	1001
A ≥ B	1010
A > B	0101

3	2	1	0
Kod F. warunkowej			

- BTT/BTR
- BTS/BTS
- BTR/BTR
- BTC/BTC
- BSF/BSFR
- BSR/BSRR
- POPCNT/POPCNT
- TEST/CTEST
- SETCC
- CALLR/CCALLR
- RET/CRET
- SHRD/RSHR
- INSD/CINSD
- OUTSD/OUTSD
- CMPBCC/CMPBWC
- SCASBCC
- STC
- MOV/MOVB
- XCHG/XCHGB
- BSWAP/BSWAPB
- PUSH/PUSHB
- PSRAD/PSRADB
- PANIC/PANICB
- PANIC/PANICB
- FS/POR/PORB
- FTAI/FTAI
- FRAT/FRAT
- FFX/FFXB
- PCMPQB/PCMPQB
- PCMPQB/PCMPQB
- PCMPQB/PCMPQB
- AND/AND
- OR/OR
- XOR/XORB
- NOT
- PCMPGT/PCMPGT

Wyjątki: BRAK

00000000000000000000

00000000000000000000

00001100000000000000

Wyjątki: BRAK

00001100000000000000

Wyjątki: BRAK

Mnemoniczne słowo i instrukcja, Nie liczba pozostanie

MMX\_D > MMX\_D - MMX\_D

MMX\_D > A\_D - MMX\_D

# NOWY CEL KOMPUTER NA UKŁADACH TTL

0372-0732 DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIĄZANIA WYNIKU MMX\_B = MMX\_B + MMX\_B \* MMX\_B

Gdy wystąpi wyjątek #P, który jest zamaskowany, istnieją trzy możliwości: 1. W przypadku 30-31 bitów (zależnie od trybu) licznik jest zerowany (wartość bieżąca w nim nie jest zerowana). 2. Licznik jest zerowany tylko wtedy, gdy wartość bieżąca jest większa od wartości poprzedniej. 3. Licznik jest zerowany tylko wtedy, gdy wartość bieżąca jest równa wartości poprzedniej.

**WAŻNE!** Instrukcja LOOPA służy do tworzenia pętli. Licznikiem jest rejestr R. Każde wywołanie instrukcji LOOPA zmniejsza licznik o jeden oraz sprawdza czy licznik nie osiągnął wartości zero. Jeżeli licznik jest wyzerowany, to program przerywa wykonanie instrukcji LOOPA. Jeżeli licznik nie jest zerowany, to program wykonuje instrukcję, która następuje po instrukcji LOOPA, a następnie powraca do instrukcji LOOPA.

**UWAGA!** W programowaniu najpopularniejszą pętlą jest pętla LOOPA. Instrukcja LOOPACC operuje na liczniku podobnie jak LOOPA, ale nie jest zerowany (wartość bieżąca w nim nie jest zerowana). Instrukcja LOOPACC może być używana do tworzenia pętli, które nie są zerowane. Instrukcja LOOPACC może być używana do tworzenia pętli, które nie są zerowane.

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	Rejstr adresowy Ax	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
																Rejestr R															
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
0000DA5h	0B00AD52h	FFDF17h	FF45ABh

WARTOŚCI PO WYKONANIU INSTRUKCJI: LOOPA/LOOPACC [A4],D3			
LICZNIK PROGRAMU PC	ZAWARTOŚĆ KOMÓRKI PAMIĘCI [A]	ADRES KOMÓRKI PAMIĘCI [A]	REJESTR D3
0B00AD52h	0B00AD52h	FFDF17h	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
0000DA5h	0B00AD52h	FFDF17h	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	FFDF17h	FF45AAh

- Symbolika Asemblera:
- C [A0],D1
  - PC [SPU],A4
  - ACC [A1],SPS
  - ACC [A5],SPU
  - PACC [SPS],ES
  - JOPACC [A7],A2
  - JOPACC [A3],D0
  - LOOPACC [A6],CS
  - LOOPACC [SPU],D6
  - LOOPACC [A3],CS
  - PAILOOPACC [CS:A1],A2
  - OPALOOPACC [DS:SPU],CS
  - JOPALOOPACC [DS:A6],A5
  - OPALOOPACC [DS:A2],D4
  - LOOPA/LOOPACC [CS:A4],A3
  - LOOPA/LOOPACC [SS:SPS],A5
  - SS

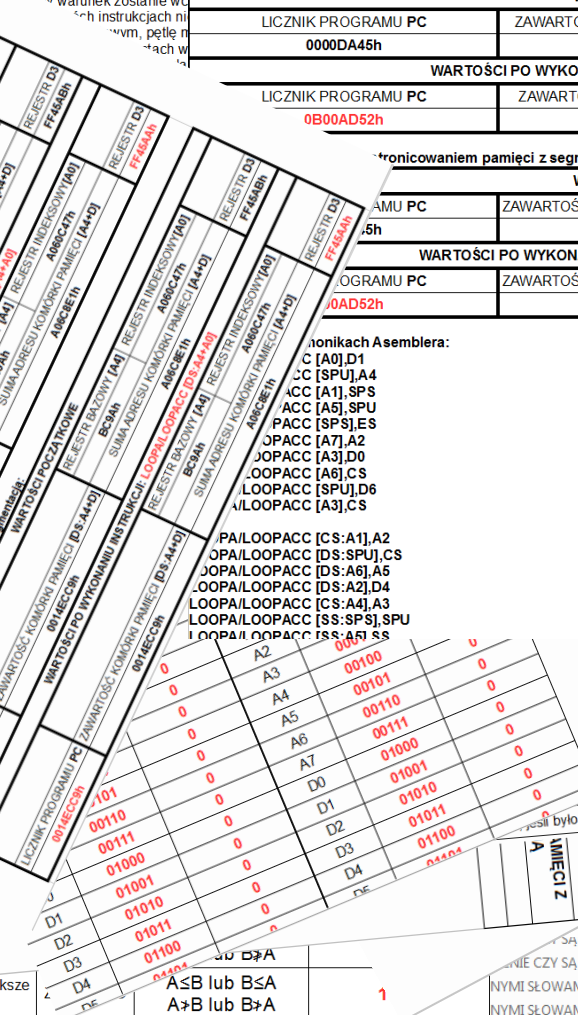
### UWAGI:

- ±D - Zdenormalizowane
- ±F - Skończona liczba

Wykonanie, jeśli większe lub równe (argumenty dotyczące liczb)

Wykonanie, jeśli mniejsze/jeśli nie (argumenty dotyczące liczb)

Wykonanie, jeśli mniejsze lub równe/jeśli nie większe (argumenty dotyczące liczb ze znakiem)



1000	000011000000000000000000
1001	κ 2-1022
1010	κ 10-309
0101	000011000000000000000000
	2-1022
	10-309

Wyjątki: BRAK

3	2	1	0
Kod F. warunkowej			

- BTT/BTR
- BTS/BTR
- BTR/BTR
- BTC/BTR
- BSF/BSR
- BSR/BSR
- POPCNT/POPC
- TEST/CTEST
- SETCC
- CALLR/CCALLR
- RET/CRET
- SHRD/RSHR
- INSD/CINSD
- OUTSD/COUSD
- CMPBCC/CMPBWC
- SCASBCC
- STC
- XCHG
- PSRL
- PSRAI
- PSRAW
- PSRAD
- PSRADR
- PANL
- PANL
- PANL
- FS
- FPC
- FTAI
- FRAT
- FFX/CFX
- PCMPEQB/PCMPEQL
- PCMPEQB/PCMPEQP
- PCMPEQW/PCMPEQD
- PCMPEQW/PCMPEQD
- AND/ANDR
- OR/ORR
- XOR/XORR
- NOT
- PC
- PCM
- PCMPG
- PCMPGT











# NOWY CEL

# KOMPUTER NA UKŁADACH TTL

**DODAWANIE OŚMIU BAJTÓW W TRYBIE ZAWIJANIA WYNIKU**

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

- W przypadku żrątkiw... (niezrozumiałe)
- ... (niezrozumiałe)
- ... (niezrozumiałe)

**WAŻNE!** Instrukcja LOOPA służy do tworzenia pętli. Nie ma możliwości zorganizowania bezsegmentacji. Aktywnymi segmentami (wykonywalne).

**UWAGA!** W programowaniu najbar... (niezrozumiałe)

**WAŻNE!** Powyższe instrukcje... (niezrozumiałe)

$X1 = (x \times 4)$

$PX2 =$

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

WARTOŚCI POZĄTKOWE			
LICZNIK PROGRAMU PC	ZAWARTOŚĆ KOMÓRKI PAMIĘCI [A4]	REJESTR ADRESOWY [A4]	REJESTR D3
0000DA5h	0B00AD52h	FFDF17h	FF45ABh

**WARTOŚCI PO WYKONANIU INSTRUKCJI: LOOPA LOOPACC [A4], D3**

LICZNIK PROGRAMU PC	ZAWARTOŚĆ KOMÓRKI PAMIĘCI [A]	ADRES KOMÓRKI PAMIĘCI [A]	REJESTR D3
0B00AD52h	0B00AD52h	FFDF17h	FF45AAh

**Stronicowaniem pamięci z segmentacją:**

**WARTOŚCI POZĄTKOWE**

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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

... (numerical data and diagrams) ...

**AND/ANDR** ArcSin = 1,3  
ArcSin\_real = 1,3

**OR/ORR** ArcCos = 0,2048  
ArcCos\_real = 0,195

$x = 0,37920$

**dotyczy liczu**

Wy... (niezrozumiałe)

**A<B lub B<A**  
**A>B lub B>A**

**MIMV\_A, #\_W -> MIMV\_W**

**MIMV\_W > A\_W - Min\_A\_W**

**MMX\_D > MMX\_D - MMX\_D**

**A\_D - A\_L, #\_MX\_D - MMX\_D > A\_D - MMX\_D**







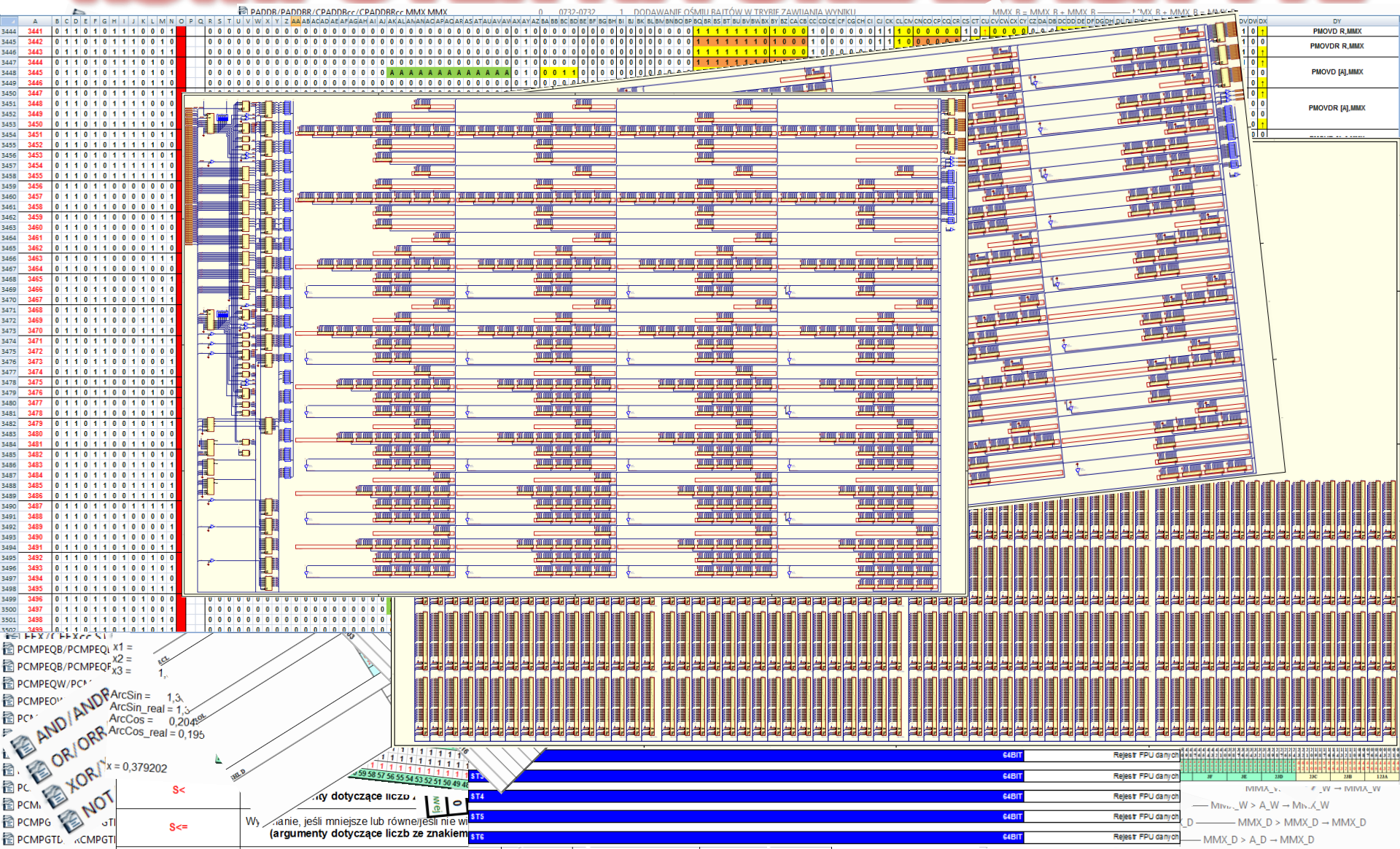






# NOWY CEL

# KOMPUTER NA UKŁADACH TTL







# NOWY CEL KOMPUTER NA UKŁADACH TTL

PCMPQB/PCMPQF X1 =  
PCMPQB/PCMPQF X2 =  
PCMPQB/PCMPQF X3 =  
PCMPQB/PC\*  
PCMPQV  
PC\* AND/ANDR ArcSin = 1,3  
ArcSin\_real = 1,3  
OR/ORR ArcCos = 0,2045  
ArcCos\_real = 0,195  
PC\* XOR/XOR/x = 0,379202  
PC  
PCM  
PCMPG  
PCMPGT

Wyliczenie, jeśli mniejsze lub równe jeśli nie w  
(argumenty dotyczące liczb ze znakami)

374  
375  
376

Rejestr FPU danych  
Rejestr FPU danych  
Rejestr FPU danych

MMX\_V  
MMX\_W  
MMX\_D > A\_W - MMX\_X  
MMX\_D > MMX\_D - MMX\_D  
MMX\_D > A\_D - MMX\_D

# NOWY CEL

# KOMPUTER NA UKŁADACH TTL

**PCMPPEQB/PCMPPEQ<sub>x1</sub> =**  
**PCMPPEQB/PCMPPEQ<sub>x2</sub> =**  
**PCMPPEQB/PCMPPEQ<sub>x3</sub> =**  
**PCMPPEQW/PC\***  
**PCMPPEOV** ArcSin = 1,3  
ArcSin\_real = 1,3  
**PC\*** ArcCos = 0,2045  
ArcCos\_real = 0,195  
**AND/ANDR**  
**OR/ORR**  
**XOR/XOR** /x = 0,379202  
**PC**  
**PCM**  
**PCMPG**  
**PCMPGT** /CMPGT

Wykreszenie, jeśli mniejsze lub równe jeśli nie w (argumenty dotyczące liczb ze znakiem

Rejestr FPU danych  
Rejestr FPU danych  
Rejestr FPU danych

MMX\_D > A\_W - MMX\_X\_W  
MMX\_D > MMX\_D - MMX\_D  
MMX\_D > A\_D - MMX\_D

Adres	Wartość
3441	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3442	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3443	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3444	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3445	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3446	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3447	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3448	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3449	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3450	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3451	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3452	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3453	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3454	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3455	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3456	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3457	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3458	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3459	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3460	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3461	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3462	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3463	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3464	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3465	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3466	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3467	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3468	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3469	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3470	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3471	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3472	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3473	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3474	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3475	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3476	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3477	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3478	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3479	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3480	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3481	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3482	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3483	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3484	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3485	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3486	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3487	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3488	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3489	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3490	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3491	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3492	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3493	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3494	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3495	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3496	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3497	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3498	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3499	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3500	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3501	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0
3502	0 1 1 1 0 1 0 1 1 1 0 0 0 1 0



# NOWY CEL

# KOMPUTER NA UKŁADACH TTL

PCMPPEQB/PCMPPEQ x1 =  
PCMPPEQB/PCMPPEQ x2 =  
PCMPPEQB/PCMPPEQ x3 =

AND/ORR ArcSin = 1,3  
AND/ORR ArcSin\_real = 1,3  
XOR ArcCos = 0,2040  
NOT ArcCos\_real = 0,195

Wywołanie, jeśli mniejsze lub równe (jeśli nie w argumenty dotyczące liczb ze znakiem)

Rejestr FPU danych  
Rejestr FPU danych  
Rejestr FPU danych

MMX\_D > A\_W - MMX\_W  
MMX\_D > MMX\_D - MMX\_D  
MMX\_D > A\_D - MMX\_D



# NOWY CEL

# KOMPUTER NA UKŁADACH TTL

The image displays a complex TTL circuit board layout. The central area is dominated by a dense grid of integrated circuits (ICs), likely microprocessors or memory chips, interconnected by a network of traces. The layout is organized into several distinct sections, with some components highlighted in blue and others in yellow. The board is populated with various components, including resistors, capacitors, and other passive elements. The layout is overlaid with a grid of coordinates, with row and column numbers visible on the left and top edges.

At the top of the page, there is a large red title "NOWY CEL" and a larger red title "KOMPUTER NA UKŁADACH TTL". Below the title, there is a data table with columns labeled A through S and rows numbered 3444 to 3502. The table contains binary data (0s and 1s) and some highlighted cells in yellow and red. The table is titled "PADDR/PADDR/CPADDR/CPADDRcc MMX MMX" and "DODAWANIE OŚMIU BAITÓW W TRYBIE ZAWIĄZANIA WYNIKU".

On the left side, there is a vertical list of component numbers from 3444 to 3502, corresponding to the rows in the table. The numbers are arranged in a column, with some numbers repeated. The numbers are: 3444, 3445, 3446, 3447, 3448, 3449, 3450, 3451, 3452, 3453, 3454, 3455, 3456, 3457, 3458, 3459, 3460, 3461, 3462, 3463, 3464, 3465, 3466, 3467, 3468, 3469, 3470, 3471, 3472, 3473, 3474, 3475, 3476, 3477, 3478, 3479, 3480, 3481, 3482, 3483, 3484, 3485, 3486, 3487, 3488, 3489, 3490, 3491, 3492, 3493, 3494, 3495, 3496, 3497, 3498, 3499, 3500, 3501, 3502.

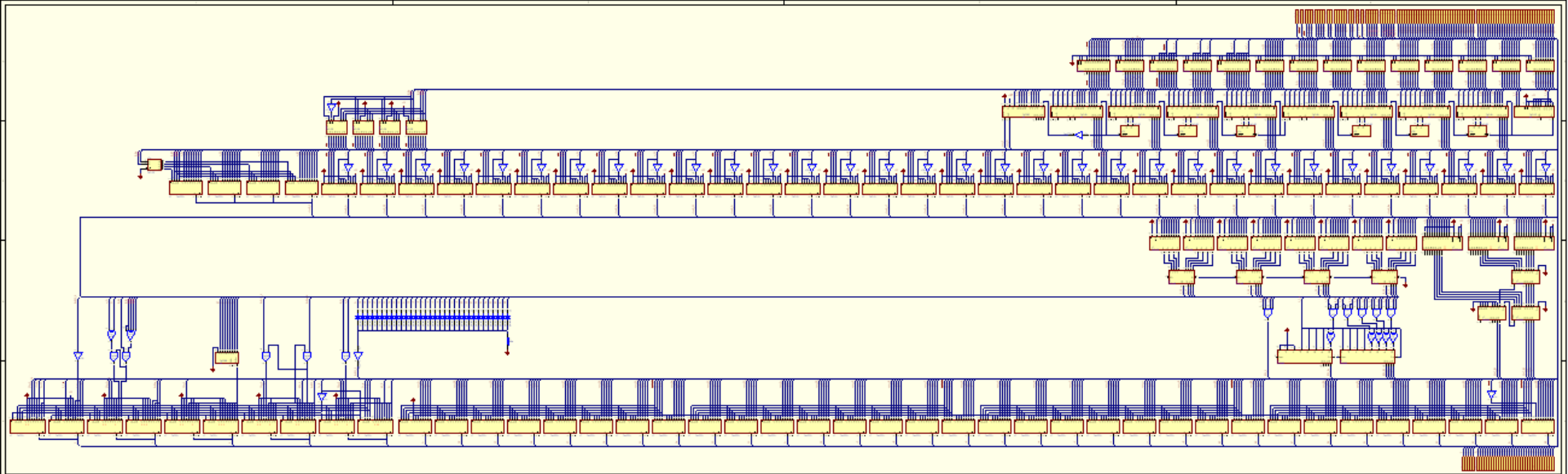
At the bottom left, there is a list of logic gates and their symbols: AND/ANDR, OR/ORR, XOR/XOR, and NOT. Below these, there are mathematical expressions:  $\text{ArcSin} = 1,3$ ,  $\text{ArcSin\_real} = 1,3$ ,  $\text{ArcCos} = 0,204$ , and  $\text{ArcCos\_real} = 0,195$ . There is also a value  $x = 0,379202$ .

At the bottom right, there is a legend for the layout, showing symbols for registers and data paths: "Rejestr FPU danych", "MIMAX", "MIMAX\_W", "MIMAX\_V", "MIMAX\_V\_W", "MIMAX\_D", "MIMAX\_D\_W", "MIMAX\_D\_V", "MIMAX\_D\_V\_W".

# NOWY CEL

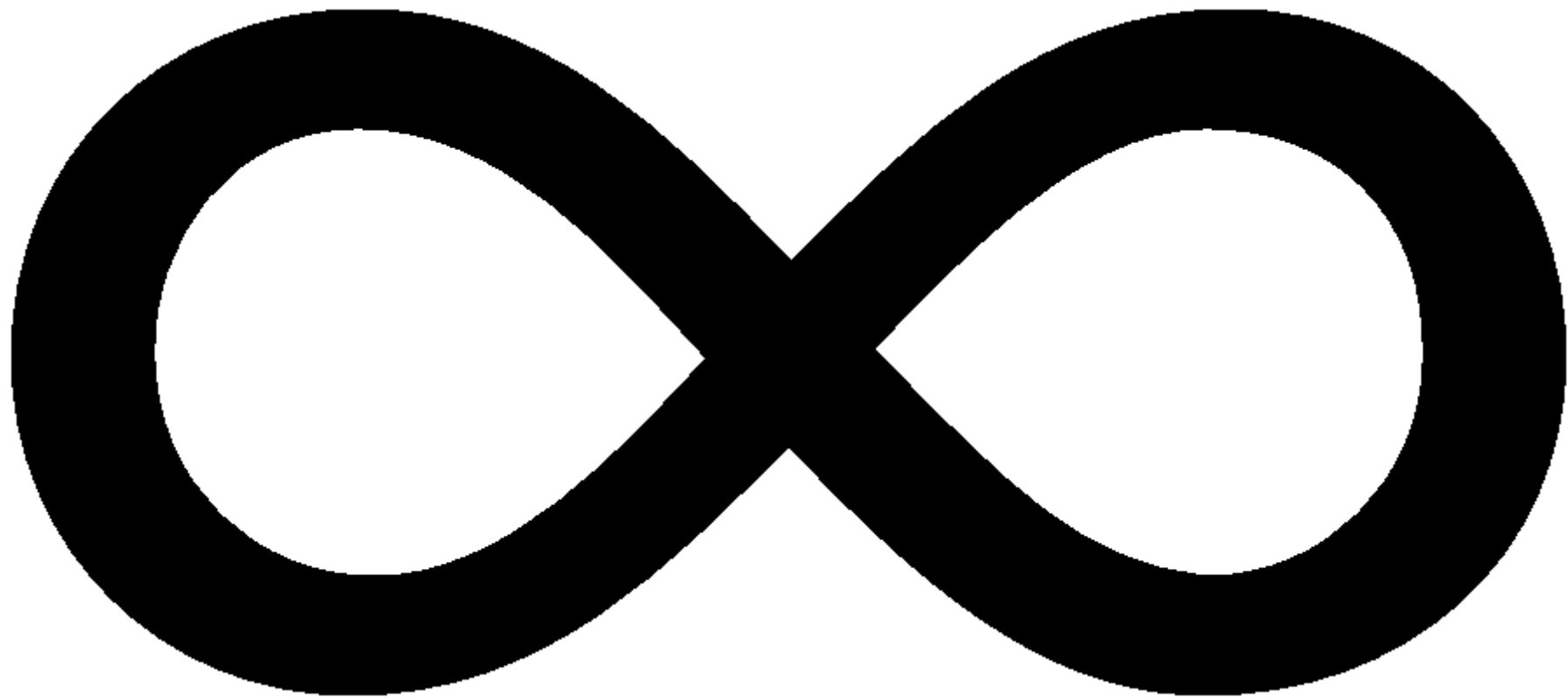
# KOMPUTER NA UKŁADACH TTL

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AL	AN	AO	AP	AQ	AR	AS	AT	AU	AV	AW	AX	AY	AZ	BA	BB	BC	BD	BE	BF	BG	BH	BI	BJ	BK	BL	BM	BN	BO	BP	BQ	BR	BS	BT	BU	BV	BW	BX	BY	BZ	CA	CB	CC	CD	CE	CF	CG	CH	CI	CJ	CK	CL	CM	CN	CO	CP	CQ	CR	CS	CT	CU	CV	CW	CX	CY	CZ	DA	DB	DC	DD	DE	DF	DG	DH	DI	DJ	DK	DL	DM	DN	DO	DP	DQ	DR	DS	DT	DU	DV	DW	DX	DY	DZ	EA	EB	EC	ED	EE	EF	EG	EH	EI	EJ	EK	EL	EM	EN	EO	EP	EQ	ER	ES	ET	EU	EV	EW	EX	EY	EZ	FA	FB	FC	FD	FE	FF	FG	FH	FI	FJ	FK	FL	FM	FN	FO	FP	FQ	FR	FS	FT	FU	FV	FW	FX	FY	FZ	GA	GB	GC	GD	GE	GF	GG	GH	GI	GJ	GK	GL	GM	GN	GO	GP	GQ	GR	GS	GT	GU	GV	GW	GX	GY	GZ	HA	HB	HC	HD	HE	HF	HG	HH	HI	HJ	HK	HL	HM	HN	HO	HP	HQ	HR	HS	HT	HU	HV	HW	HX	HY	HZ	IA	IB	IC	ID	IE	IF	IG	IH	II	IJ	IK	IL	IM	IN	IO	IP	IQ	IR	IS	IT	IU	IV	IW	IX	IY	IZ	JA	JB	JC	JD	JE	JF	JG	JH	JI	JJ	JK	JL	JM	JN	JO	JP	JQ	JR	JS	JT	JU	JV	JW	JX	JY	JZ	KA	KB	KC	KD	KE	KF	KG	KH	KI	KJ	KL	KM	KN	KO	KP	KQ	KR	KS	KT	KU	KV	KW	KX	KY	KZ	LA	LB	LC	LD	LE	LF	LG	LH	LI	LJ	LK	LL	LM	LN	LO	LP	LQ	LR	LS	LT	LU	LV	LW	LX	LY	LZ	MA	MB	MC	MD	ME	MF	MG	MH	MI	MJ	MK	ML	MM	MN	MO	MP	MQ	MR	MS	MT	MU	MV	MW	MX	MY	MZ	NA	NB	NC	ND	NE	NF	NG	NH	NI	NJ	NK	NL	NM	NN	NO	NP	NQ	NR	NS	NT	NU	NV	NW	NX	NY	NZ	OA	OB	OC	OD	OE	OF	OG	OH	OI	OJ	OK	OL	OM	ON	OO	OP	OQ	OR	OS	OT	OU	OV	OW	OX	OY	OZ	PA	PB	PC	PD	PE	PF	PG	PH	PI	PJ	PK	PL	PM	PN	PO	PP	PQ	PR	PS	PT	PU	PV	PW	PX	PY	PZ	QA	QB	QC	QD	QE	QF	QG	QH	QI	QJ	QK	QL	QM	QN	QO	QP	QQ	QR	QS	QT	QU	QV	QW	QX	QY	QZ	RA	RB	RC	RD	RE	RF	RG	RH	RI	RJ	RK	RL	RM	RN	RO	RP	RQ	RR	RS	RT	RU	RV	RW	RX	RY	RZ	SA	SB	SC	SD	SE	SF	SG	SH	SI	SJ	SK	SL	SM	SN	SO	SP	SQ	SR	SS	ST	SU	SV	SW	SX	SY	SZ	TA	TB	TC	TD	TE	TF	TG	TH	TI	TJ	TK	TL	TM	TN	TO	TP	TQ	TR	TS	TT	TU	TV	TW	TX	TY	TZ	UA	UB	UC	UD	UE	UF	UG	UH	UI	UJ	UK	UL	UM	UN	UO	UP	UQ	UR	US	UT	UU	UV	UW	UX	UY	UZ	VA	VB	VC	VD	VE	VF	VG	VH	VI	VJ	VK	VL	VM	VN	VO	VP	VQ	VR	VS	VT	VU	VV	VW	VX	VY	VZ	WA	WB	WC	WD	WE	WF	WG	WH	WI	WJ	WK	WL	WM	WN	WO	WP	WQ	WR	WS	WT	WU	WV	WW	WX	WY	WZ	XA	XB	XC	XD	XE	XF	XG	XH	XI	XJ	XK	XL	XM	XN	XO	XP	XQ	XR	XS	XT	XU	XV	XW	XX	XY	XZ	YA	YB	YC	YD	YE	YF	YG	YH	YI	YJ	YK	YL	YM	YN	YO	YP	YQ	YR	YS	YT	YU	YV	YW	YX	YY	YZ	ZA	ZB	ZC	ZD	ZE	ZF	ZG	ZH	ZI	ZJ	ZK	ZL	ZM	ZN	ZO	ZP	ZQ	ZR	ZS	ZT	ZU	ZV	ZW	ZX	ZY	ZZ
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----



3494	3495	3496	3497	3498	3499	3500	3501	3502	3503	$PCMPQB/PCMPQI$ $x_1 =$ $PCMPQB/PCMPQI$ $x_2 =$ $PCMPQB/PCMPQI$ $x_3 =$		$PCMPQW/PC$ $PCMPQW$ $ArcSin = 1,3$ $PCMPQW$ $ArcSin\_real = 1,3$ $PCMPQW$ $ArcCos = 0,2045$ $PCMPQW$ $ArcCos\_real = 0,195$		$AND/ANDR$ $OR/ORR$ $XOR/XOR$ $x = 0,379202$ $PC$ $PCM$ $PCMPG$ $PCMPGT$		Wyliczenie, jeśli mniejsze lub równe jeśli nie w (argumenty dotyczące liczb ze znakami)		374 375 376	64BIT 64BIT	Rejestr FPU danych Rejestr FPU danych Rejestr FPU danych	MMX_W → MMX_W MMX_D → MMX_D → MMX_D MMX_D → A_D → MMX_D
------	------	------	------	------	------	------	------	------	------	---	--	--	--	--	--	--	--	-------------------	----------------	--	---





**WIEDZA+WYOBRAŹNIA=**

**FANTAZJA**

**Dziękuję za uwagę  
Rafał Wiśniewski**