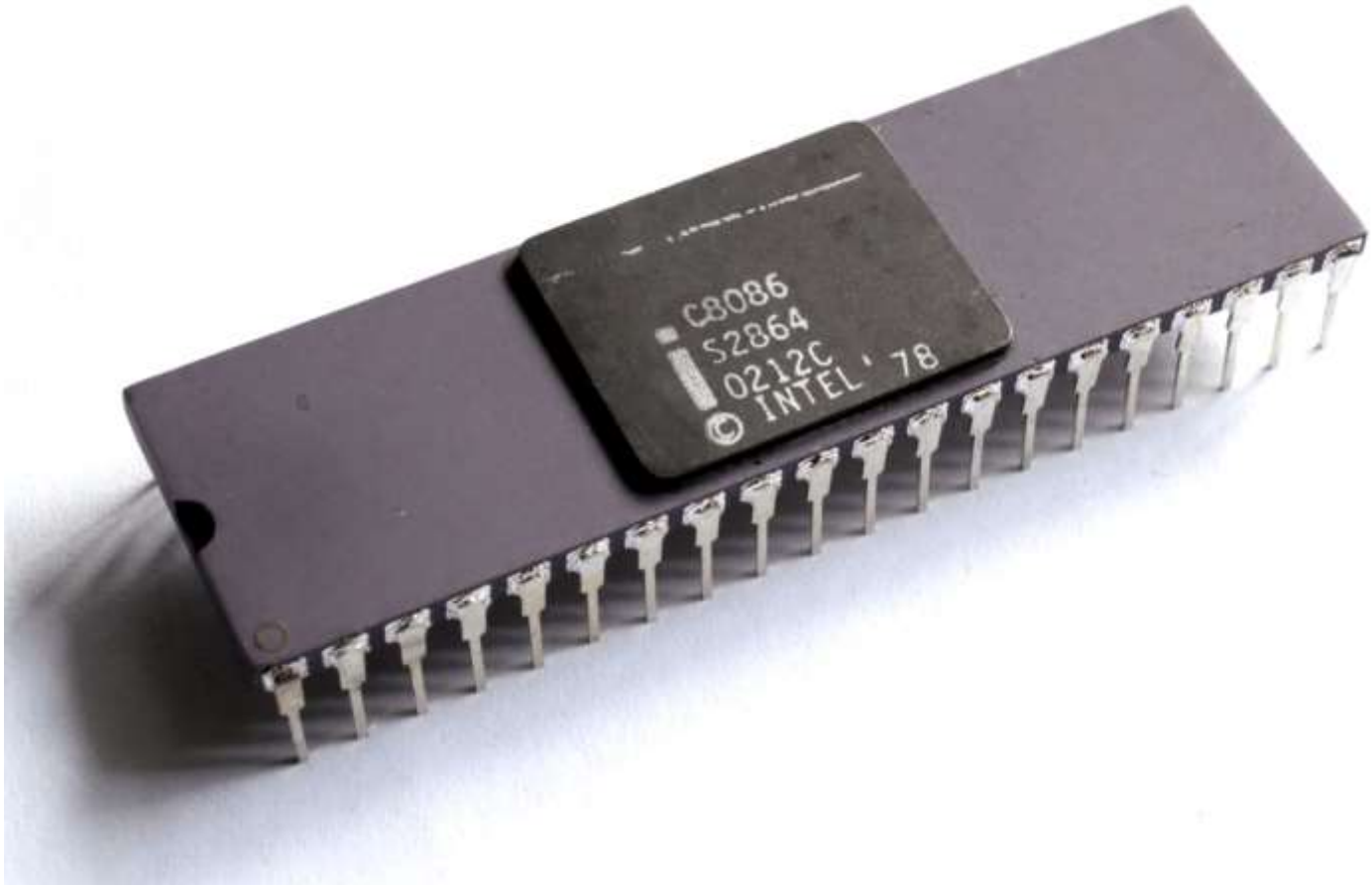

Uniwersytet
w Siedlcach
Architektura
Systemów
Komputerowych
dr Marcin
Stępniak

# Intel 8086






https://pl.wikipedia.org/wiki/Intel_8086

# Procesor 8086 firmy Intel



- 8086 – 16-bitowy mikroprocesor wprowadzony na rynek 8 czerwca 1978 roku
- został zaprojektowany w technologii 3 µm HMOS (ang. *High performance Metal-Oxide Semiconductor)* jako rozszerzenie 8-bitowego 8080/8085
- Jego zastosowanie (w szczególności jego późniejszej odmiany z 8-bitowym interfejsem – 8088) w pierwszych ogólnodostępnych komputerach osobistych (IBM PC), doprowadziło do jego wielkiej popularyzacji i dalszego rozwoju tej rodziny procesorów (architektura x86).
- W związku z historycznym znaczeniem procesora 8086 firmie Intel przydzielono identyfikator 0x8086 na liście identyfikatorów (PCI ID) dostawców urządzeń dla magistrali PCI

# PCI



- PCI (ang. *Peripheral Component Interconnect*) –
- magistrala komunikacyjna służąca do przyłączania kart rozszerzeń do płyty głównej w komputerach klasy PC.
- Po raz pierwszy została publicznie zaprezentowana w czerwcu 1992 r. jako rozwiązanie umożliwiające szybszą komunikację pomiędzy procesorem i kartami niż stosowane dawniej ISA.
- Dodatkową zaletą PCI jest to, że nie ma znaczenia czy w gnieździe jest karta sterownika dysków (np. SCSI), sieciowa czy graficzna.
- Każda karta, pasująca do gniazda *PCI*, funkcjonuje bez jakichkolwiek problemów, gdyż nie tylko sygnały, ale i przeznaczenie poszczególnych styków gniazda są znormalizowane.

# Podstawowe parametry mikroprocesora 8086



- Architektura CISC
- Przestrzeń adresowa pamięci – 1 MB w trybie rzeczywistym
- 16-bitowa magistrala danych
- 20-bitowa magistrala adresowa
- Częstotliwość sygnału zegarowego do 10 MHz
- 91 podstawowych typów rozkazów; samych rozkazów jest znacznie więcej

# Podstawowe parametry mikroprocesora 8086



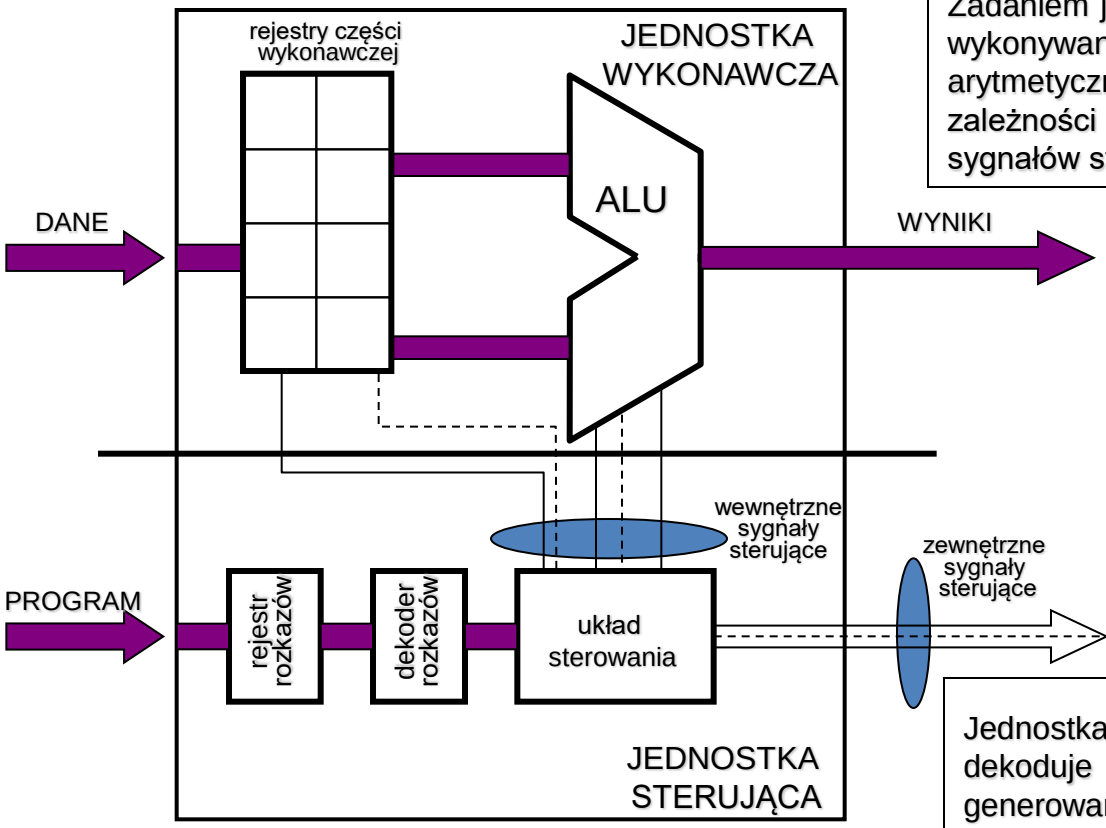
- Przestrzeń adresowa urządzeń wejścia/wyjścia – 64 kB,
- Możliwość wykonywania operacji bitowych, bajtowych, o długości słowa i łańcuchowych,
- 7 trybów adresowania argumentów w pamięci,
- 16-bitowa jednostka arytmetyczno-logiczna (ALU),
- 16-bitowe rejestry ogólnego przeznaczenia,
- 6-bajtowa kolejka rozkazów

# Podstawowe parametry mikroprocesora 8086



- dwa tryby pracy – minimalny i maksymalny:
  - W trybie minimalnym procesor steruje całym systemem mikrokomputerowym pełniąc rolę kontrolera magistrali.
    - Zwykle system taki składa się z jednego obwodu drukowanego i kilku urządzeń peryferyjnych.
  - W trybie maksymalnym magistrala jest współdzielona pomiędzy mikroprocesor a procesory wspomagające.
    - Funkcje sterownika magistrali przejmuje wtedy osobny element systemu mikrokomputerowego zwany kontrolerem magistrali.
    - Tryb ten stosowany jest w przypadku systemów wieloprocesorowych (np. system w którego skład wchodzi mikroprocesor wraz z koprocesorem matematycznym)

# SCHEMAT BLOKOWY PROCESORA 8086




Zadaniem jednostki wykonawczej jest wykonywanie operacji arytmetycznych i logicznych w zależności od wewnętrznych sygnałów sterujących

Jednostka sterująca z programu dekoduje rozkaz i na jego podstawie generowane są wewnętrzne i zewnętrzne sygnały sterujące

# SCHEMAT BLOKOWY PROCESORA 8086



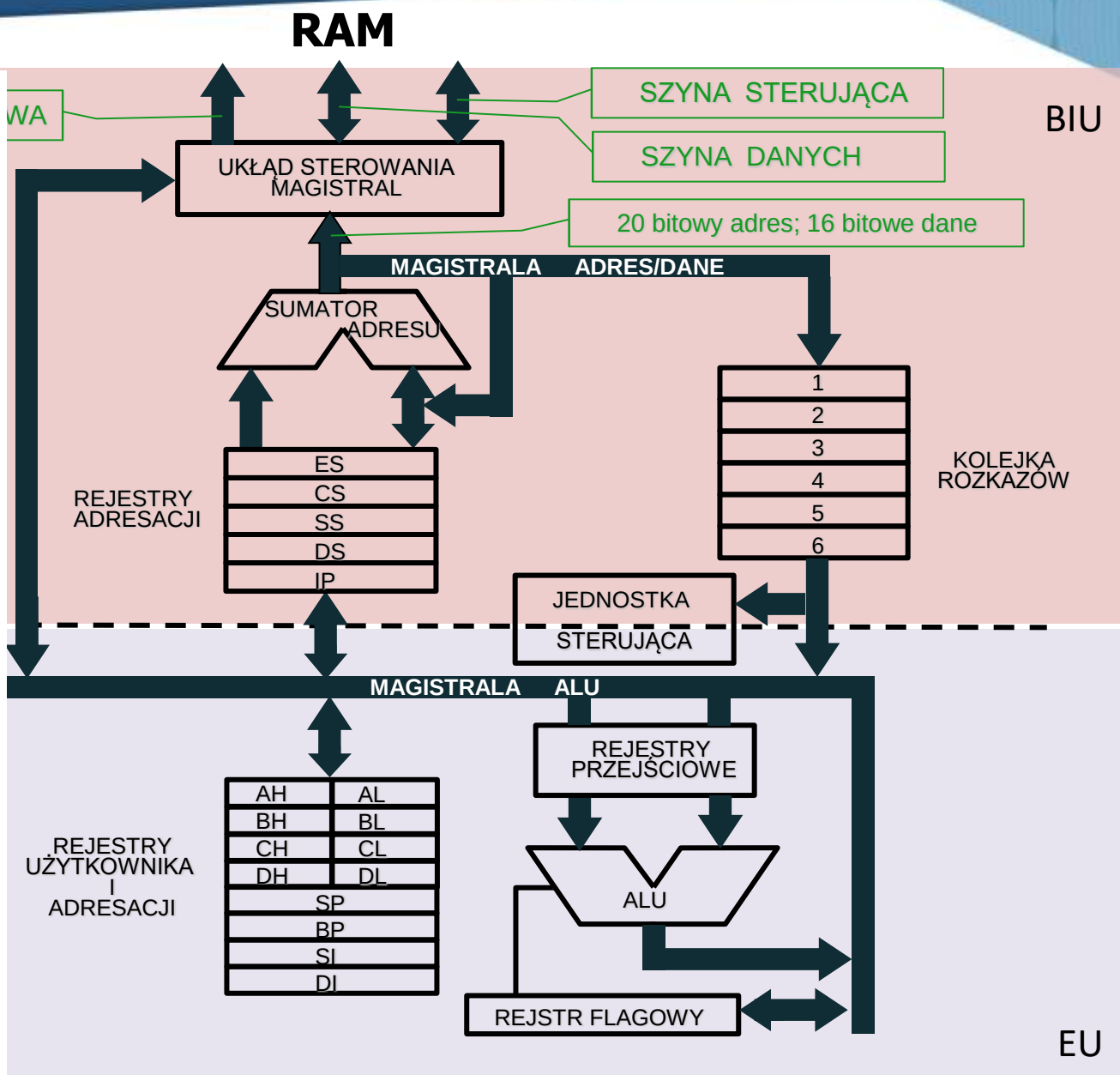
Układ wykonawczy zawiera:
- 16-bitową jednostkę arytmetyczno-logiczną,
- układ sterowania z rejestrem rozkazów,
- cztery 16-bitowe rejestry użytkownika
- cztery 16-bitowe rejestry adresacji
- 16-bitowy rejestr wskaźników (rejestr flagowy)

Zadaniem układu wykonawczego jest dekodowanie i wykonywanie rozkazów gromadzonych w kolejce. W trakcie wykonywania rozkazów w układzie wykonawczym układ sprzęgający magistrali zewnętrznej otrzymuje zezwolenie na pobranie

Układ sprzęgający magistrali zawiera:
- układ generacji adresu fizycznego,
- układ kolejkowania rozkazów,
- 16-bitowe rejestry adresacji (segmentowe i IP)
- bufor WE/WY

Układ sprzęgający przesyła dane między procesorem a pamięcią operacyjną lub układami WE/WY pod nadzorem układu wykonawczego. W czasie gdy układ wykonawczy realizuje kolejny rozkaz, BIU pobiera nowy rozkaz z pamięci operacyjnej i przekazuje go do kolejki.

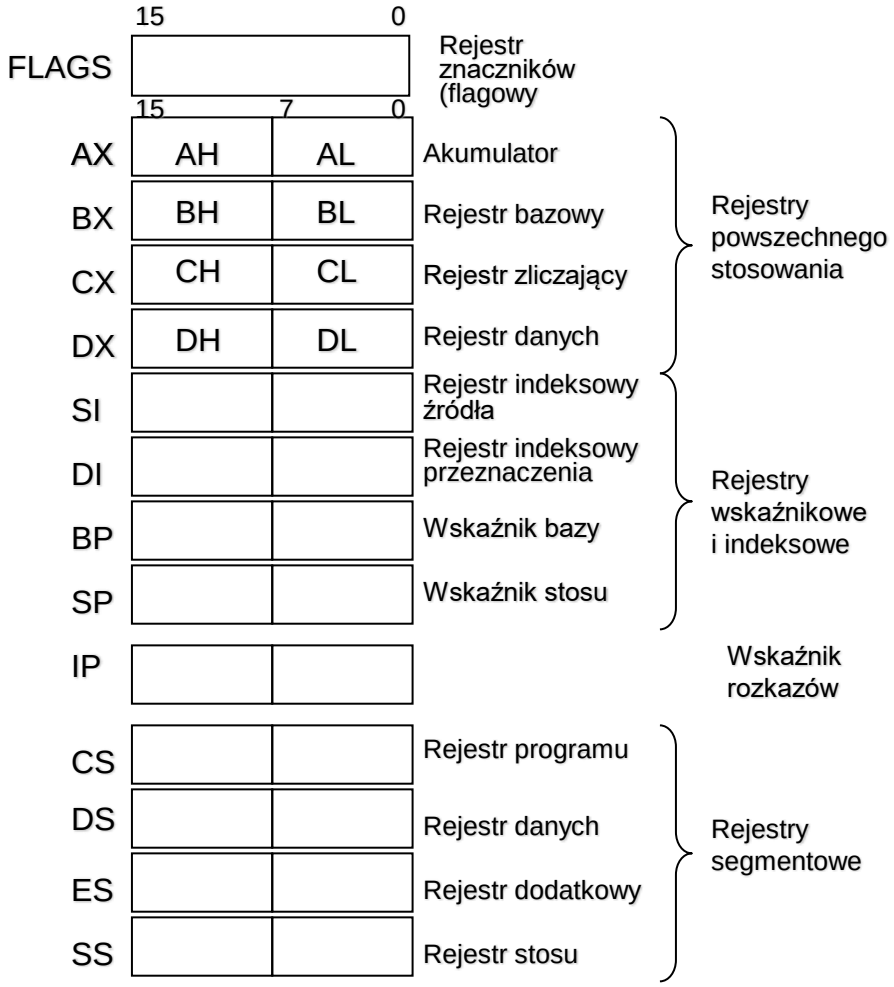
Drugim istotnym zadaniem układu sprzęgającego BIU jest wyznaczania adresu fizycznego (m.in. na podstawie danych przekazywanych z EU)

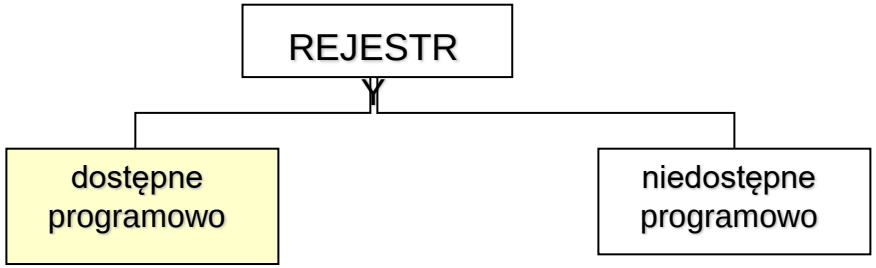


# REJESTRY



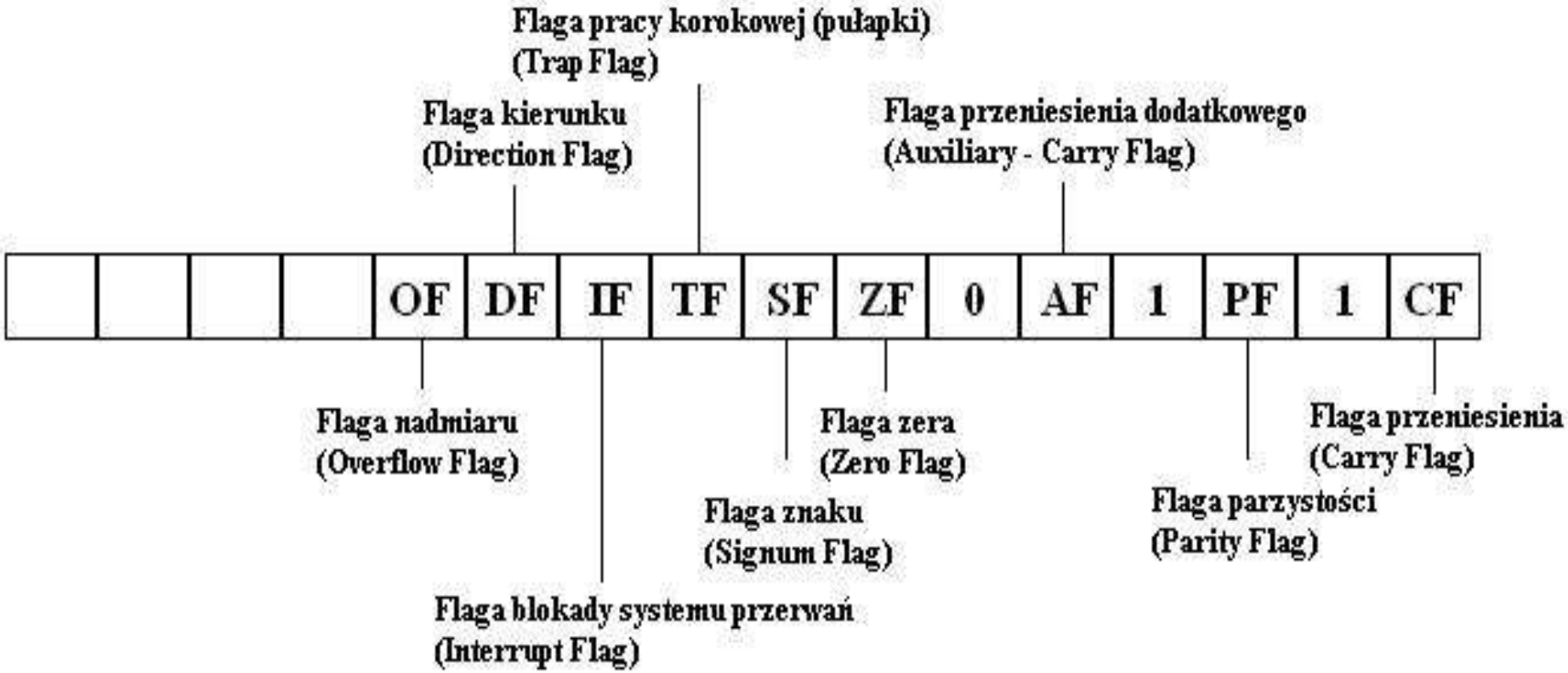
(na przykładzie procesora 8086/8088)



# Rejestr znaczników w mikroprocesorze 8086





# Rejestr znaczników w mikroprocesorze 8086



- *SF* (*sign flag*) – znacznik znaku – równy najbardziej znaczącemu bitowi wyniku
  - *0* – wynik operacji dodatni
  - *1* – wynik operacji ujemny
- *ZF* (*zero flag*) – znacznik zera
  - *0* – wynik operacji różny od zera
  - *1* – wynik operacji równy zeru
- *PF* (*parity flag*) – znacznik parzystości – ustawiany w zależności od liczby jedynek w najmniej znaczących 8 bitach wyniku
  - *0* liczba jedynek w wyniku operacji nieparzysta
  - *1* liczba jedynek w wyniku operacji parzysta

# Rejestr znaczników w mikroprocesorze 8086



- *AF* (*auxiliary carry flag*) – znacznik przeniesienia połówkowego (pomocniczego)
  - *0* – brak przeniesienia pomiędzy trzecim i czwartym bitem bajta (BCD)
  - *1* – występuje przeniesienie
- *CF* (*carry flag*) – znacznik przeniesienia
  - *0* – wynik operacji arytmetycznej nie powoduje powstania przeniesienia z najbardziej znaczącego bitu
  - *1* – wynik takie przeniesienie powoduje
- *OF* (*overflow flag*) – znacznik nadmiaru
  - *0* – suma modulo 2 przeniesień z najbardziej znaczącej pozycji i pozycji przedostatniej jest równa 0
  - *1* – suma modulo 2 przeniesień z najbardziej znaczącej pozycji i pozycji przedostatniej jest równa 1 (przekroczenie zakresu w kodzie U2)

# Rejestr znaczników w mikroprocesorze 8086



- *IF* (*interrupt flag*) – znacznik przerwań
  - *0* – brak zezwolenia na przyjmowanie przerwań z wejścia *INT*
  - *1* – zezwolenie na przyjmowanie przerwań
- *DF* (*direction flag*) – znacznik kierunku, wskazuje, czy zawartości rejestrów SI i DI mają być zwiększane lub zmniejszane o jeden w czasie wykonywania operacji łańcuchowych
  - *0* – rejestry są zwiększane
  - *1* – rejestry są zmniejszane
- *TF* (*trap flag*) – znacznik pułapki umożliwiającej pracę krokową
  - *0* – praca krokowa wyłączona
  - *1* – praca krokowa włączona, mikroprocesor po wykonaniu każdego rozkazu wykona skok do odpowiedniego podprogramu obsługi przerwania.

# Rejestry ogólnego przeznaczenia: arytmetyczne,

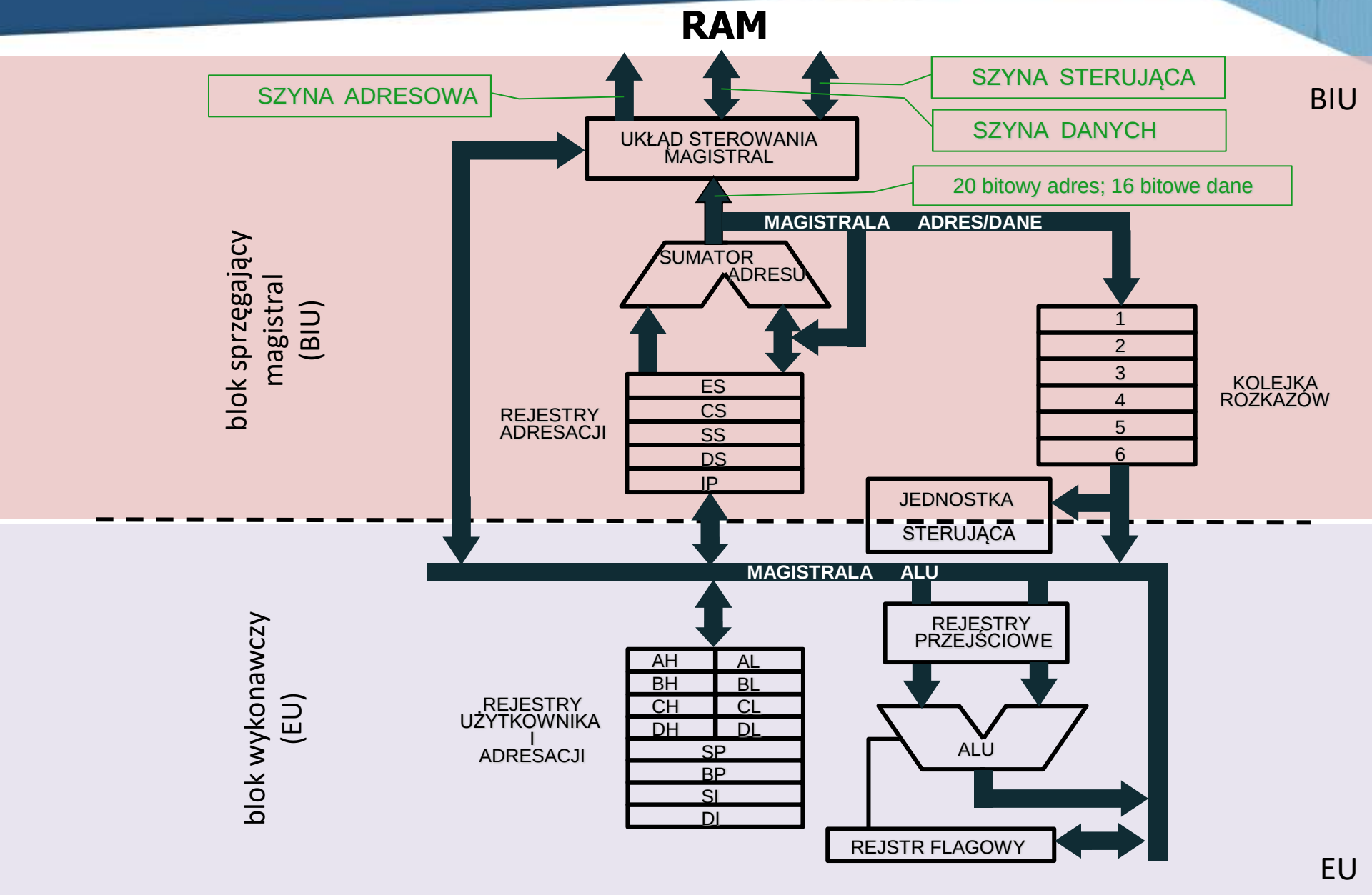


- Rejestry arytmetyczne to cztery 16-bitowe rejestry ogólnego przeznaczenia: AX, BX, CX, DX.

  Każdy z tych rejestrów może również działać jako dwa niezależne rejestry 8-bitowe:
  - *AX* lub *AH*, *AL*
  - *BX* lub *BH*, *BL*
  - *CX* lub *CH*, *CL*
  - *DX* lub *DH*, *DL*

# SCHEMAT BLOKOWY PROCESORA 8086





# Rejestry ogólnego przeznaczenia



- AX – Akumulator (Accumulator)
  Rejestr ten bezpośrednio współpracuje z jednostką arytmetyczno-logiczną.
  - Niektóre operacje, których argumenty znajdują się w akumulatorze, wykonywane są szybciej niż ich odpowiedniki wykorzystujące inne rejestry.
  - Takie rozkazy jak: mnożenie, dzielenie i operacje wejścia/wyjścia wymagają użycia akumulatora do przechowywania argumentu bądź też zapisu wyniku.
- BX – Baza (Base)
  Rejestr ten może być używany do adresowania argumentu, znajdującego się w pamięci, stanowiąc bazę do obliczania adresu.

# Rejestry ogólnego przeznaczenia: arytmetyczne



- CX – Licznik (Counter)
  Rejestr ten jest używany jako licznik w operacjach łańcuchowych oraz pętlach.
- Po każdej iteracji jego zawartość jest automatycznie dekrementowana. W rozkazach przesunięć, rejestr CL (mniej znaczący bajt rejestru CX), wykorzystywany jest jako licznik bitów.
- DX – Dane (Data)
  Rejestr ten jest wykorzystywany w niektórych operacjach arytmetycznych do przechowywania części argumentu lub wyniku operacji (mnożenie i dzielenie 16-bitowe).
  - Zawiera on także adres urządzenia w operacjach wejścia/wyjścia.

# Rejestry ogólnego przeznaczenia: wskaźnikowe i indeksowe



- 16-bitowe rejestry adresowe: SP, BP, SI, DI. Ich głównym zadaniem jest wskazywanie miejsca w pamięci, w którym znajdują się argumenty rozkazu.
- Wykorzystywane są w adresowaniu indeksowym, bazowym oraz indeksowo-bazowym.
- Można ich również używać do przechowywania argumentu bądź wyniku operacji

# Rejestry ogólnego przeznaczenia: wskaźnikowe i indeksowe



- *SP* – wskaźnik stosu (*Stack Pointer*).
  Wskazuje adres ostatnio zapisanego słowa na stosie. Jego wartość jest automatycznie ++ lub -- w zależności od wykonywanej operacji (POP, PUSH).

- *BP* – wskaźnik bazy (*Base Pointer*).
  Pełni on funkcję wskaźnika ogólnego przeznaczenia. Wykorzystywany jest do adresowania bazowego danych w segmencie stosu.

# Rejestry ogólnego przeznaczenia: wskaźnikowe i indeksowe



- *SI* – rejestr indeksowy źródła (*Source Index*). Uniwersalny rejestr indeksowy. Wykorzystywany dla dwuargumentowych operacji łańcuchowych do przetrzymywania adresu źródła danych. Po każdej kolejnej iteracji jego wartość jest ++ lub --, zależnie od ustawienia flagi kierunku.
- *DI* – rejestr indeksowy przeznaczenia (*Destination Index*). Uniwersalny rejestr indeksowy. Jest wykorzystywany w czasie dwuargumentowych operacji łańcuchowych do przetrzymywania adresu przeznaczenia danych. Po każdej kolejnej iteracji jego zawartość jest ++ lub --, zależnie od ustawienia flagi kierunku.

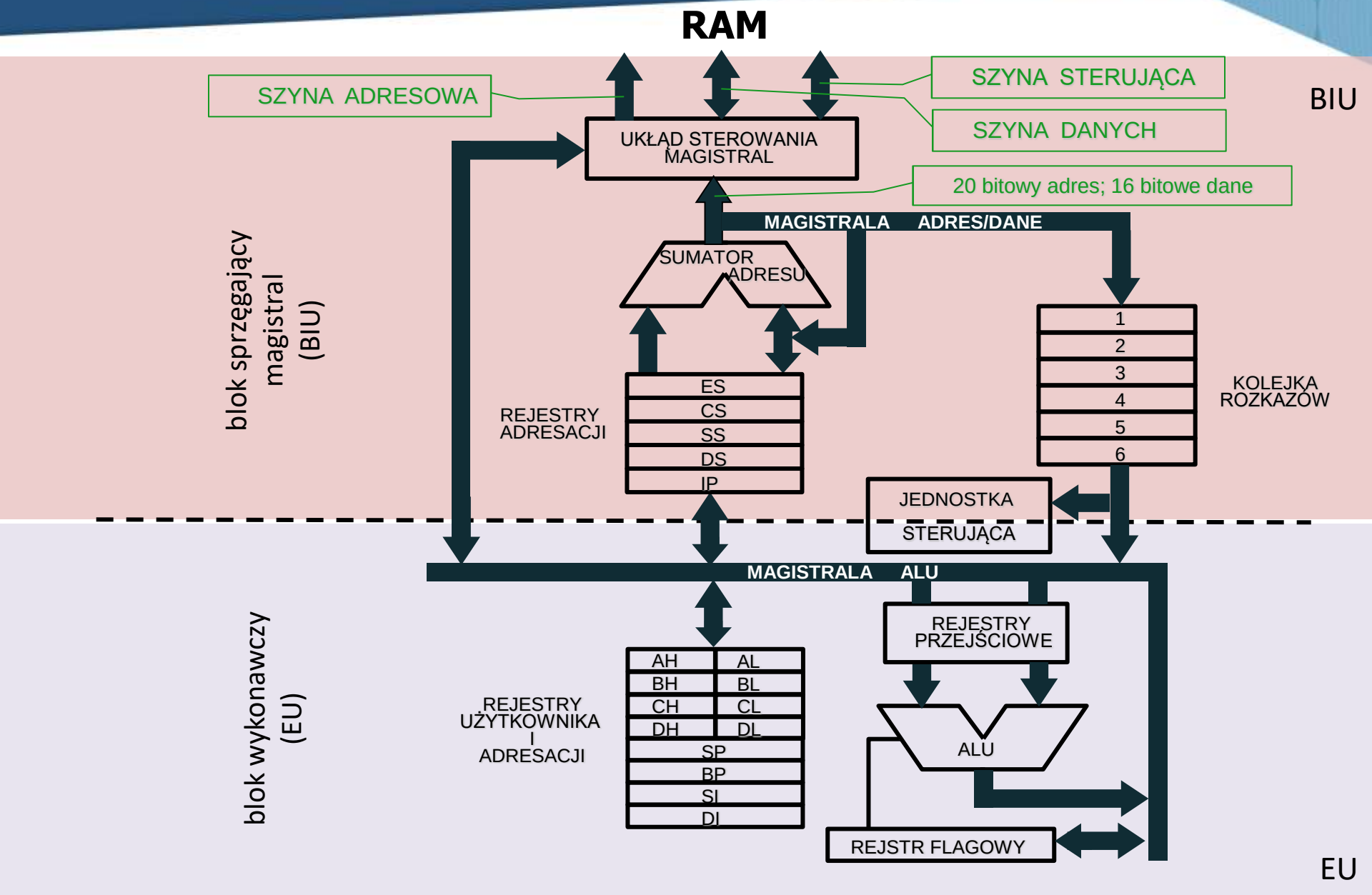
# Rejestry segmentowe



- 16-bitowe rejestry, dostępne dla programisty, których zawartość służy do obliczania adresu fizycznego komórki pamięci.
- Rejestry te zawierają adres początkowy danego segmentu pamięci.
- Mikroprocesor w zależności od rodzaju segmentu pamięci, do którego chce się odwołać, wykorzystuje odpowiedni z rejestrów.
- Programista ma możliwość zmiany automatycznie wykorzystywanego rejestru poprzez umieszczenie odpowiedniego prefiksu przed rozkazem, dla którego zmiana ma zostać zastosowana.

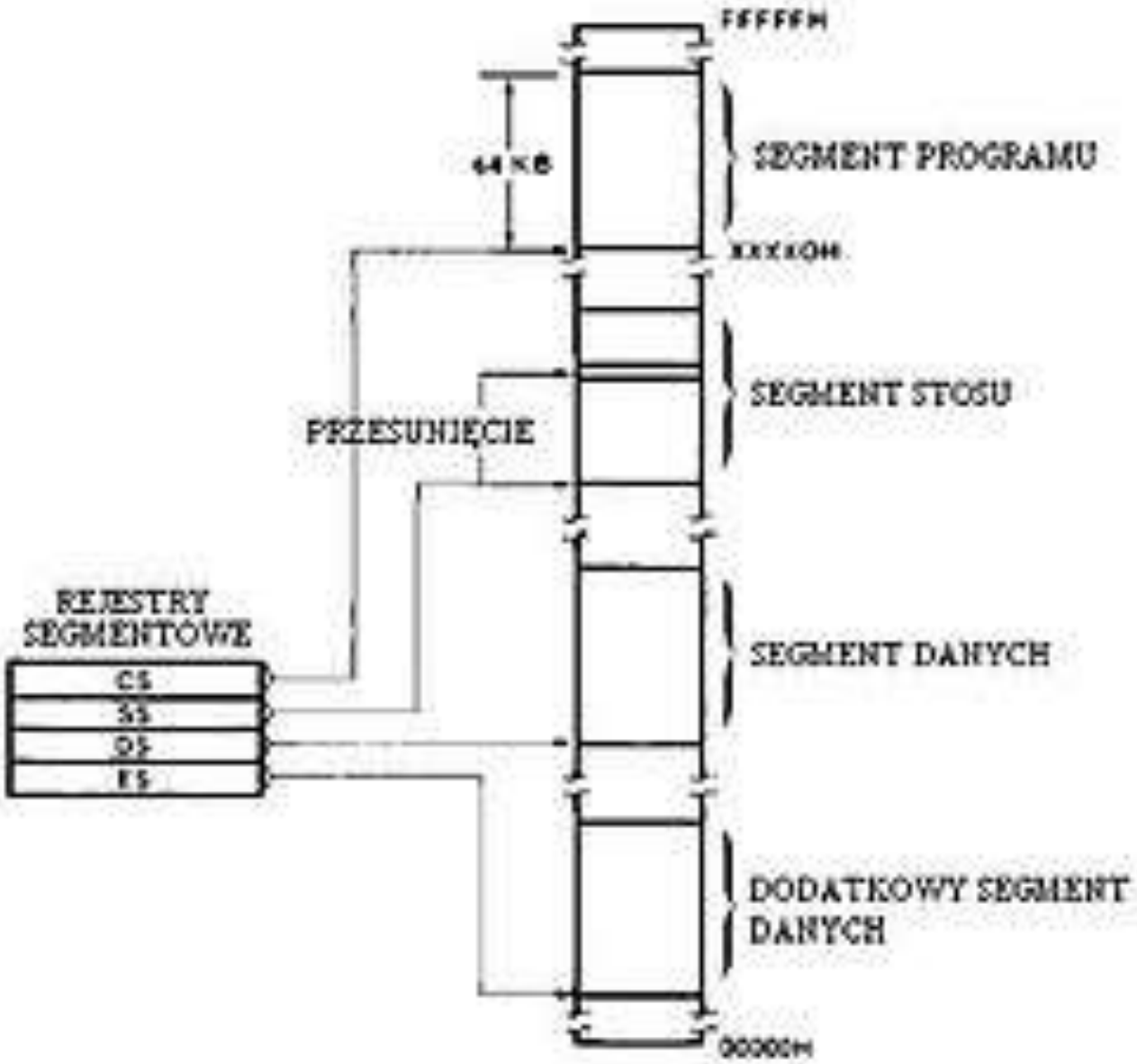
# SCHEMAT BLOKOWY PROCESORA 8086





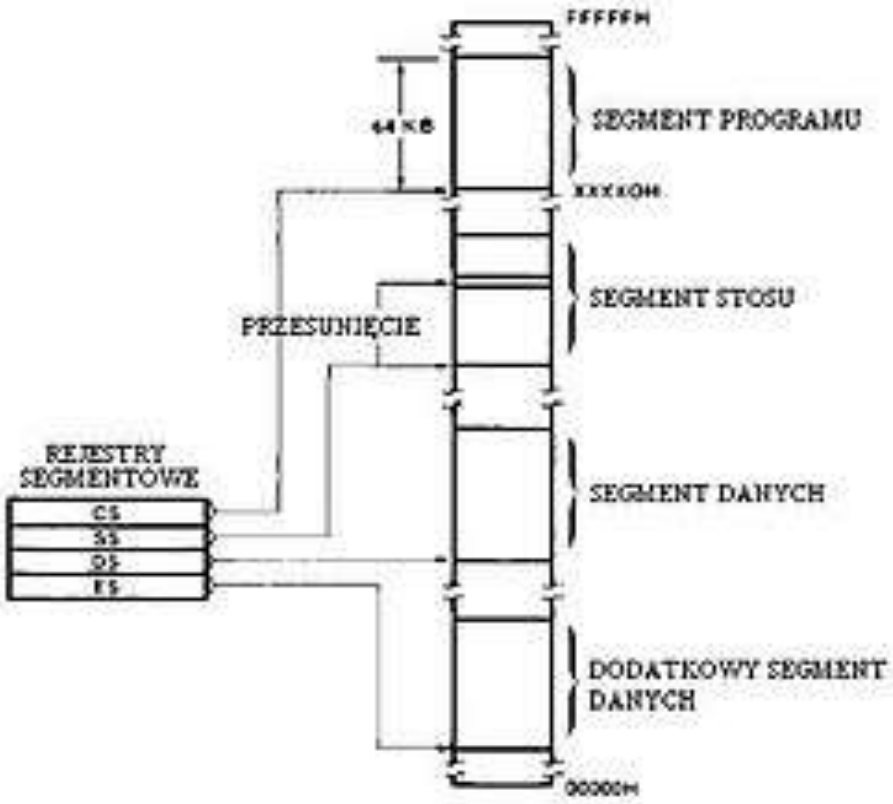
# REJESTRY SEGMENTOWE





# Segmentacja pamięci w mikroprocesorze 8086 Intel






- Mikroprocesor 8086 ma 20-bitową magistralę adresową. Pozwala ona na zaadresowanie do 1 MB pamięci operacyjnej.

Przestrzeń adresowa została podzielona na segmenty o długości 64 kB, rozpoczynające się co 16 bajtów (kolejne segmenty pamięci mogą nakładać się na siebie).

# Rejestry segmentowe



- *CS* – rejestr segmentowy programu (*Code Segment register*).
- Zawartość tego rejestru wyznacza początek aktualnie używanego segmentu programu.
- Wartość ta wykorzystywana jest do obliczania adresu fizycznego kolejnego rozkazu do pobrania z pamięci.

# Rejestry segmentowe



- *DS* – rejestr segmentowy danych (*Data Segment register*).

- Zawartość tego rejestru wyznacza początek aktualnie używanego segmentu danych.
- Wartość ta wykorzystywana jest do obliczania adresu fizycznego argumentu lub wyniku aktualnie wykonywanego rozkazu.
- Wyjątek stanowią rozkazy łańcuchowe, w których zawartość tego rejestru służy jedynie do obliczania adresu źródła danych.

# Rejestry segmentowe



- *SS* – rejestr segmentowy stosu (*Stack Segment register*).

- Zawartość tego rejestru wyznacza początek aktualnie używanego segmentu stosu.
- Wartość ta wykorzystywana jest do obliczania adresu fizycznego komórki pamięci, na którą wskazuje wskaźnik stosu (rejestr SP).

# Rejestry segmentowe



- *ES* – rejestr segmentowy dodatkowy (*Extra Segment register*).

- Zawartość tego rejestru wyznacza początek aktualnie używanego dodatkowego segmentu danych.
- Wartość ta wykorzystywana jest do obliczania adresu fizycznego przeznaczenia dla operacji łańcuchowych (np. rozkazu przenoszenia bloku danych).

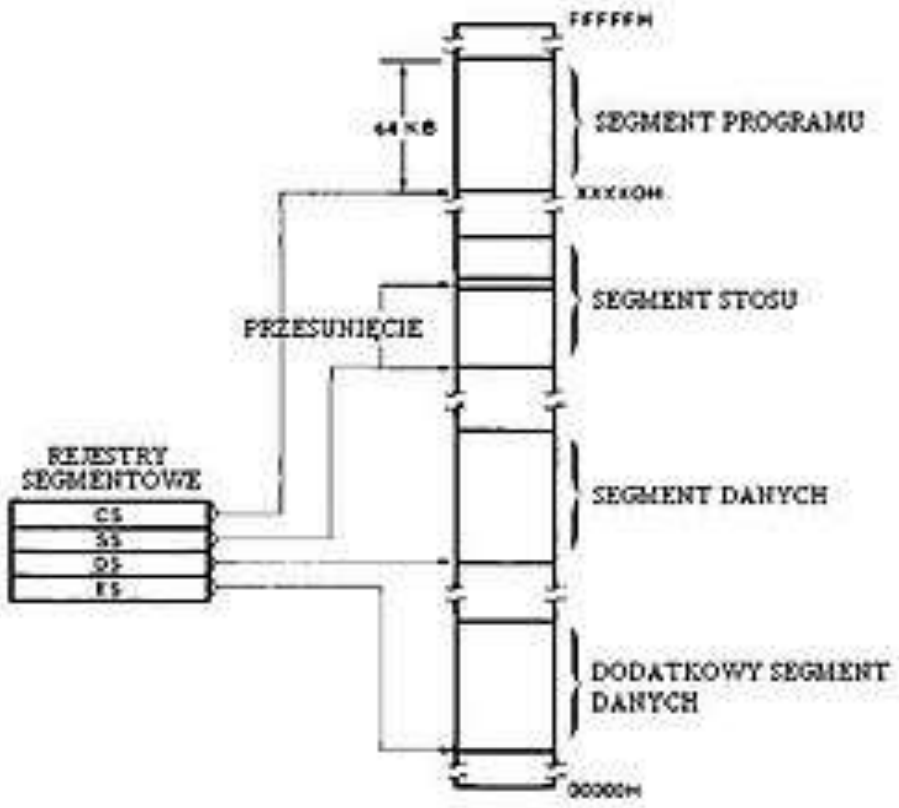
# Licznik rozkazów (Instruction Pointer – IP)



- 16-bitowy rejestr, którego zawartość służy do obliczania adresu fizycznego następnego słowa rozkazu (instrukcji) do pobrania z pamięci.
- Stanowi on rejestr indeksowy dla rejestru CS wyznaczającego segment z kodem programu.
- CS:IP to adres logiczny tej nastepnej instrukcji
- Jego zawartość jest automatycznie inkrementowana po pobraniu każdego bajtu rozkazu (w przypadku pobrania słowa jego wartość wzrasta o 2).
- Programista ma możliwość zmiany zawartości licznika rozkazów poprzez zastosowanie rozkazu skoku.

# Segmentacja pamięci w mikroprocesorze 8086 Intel



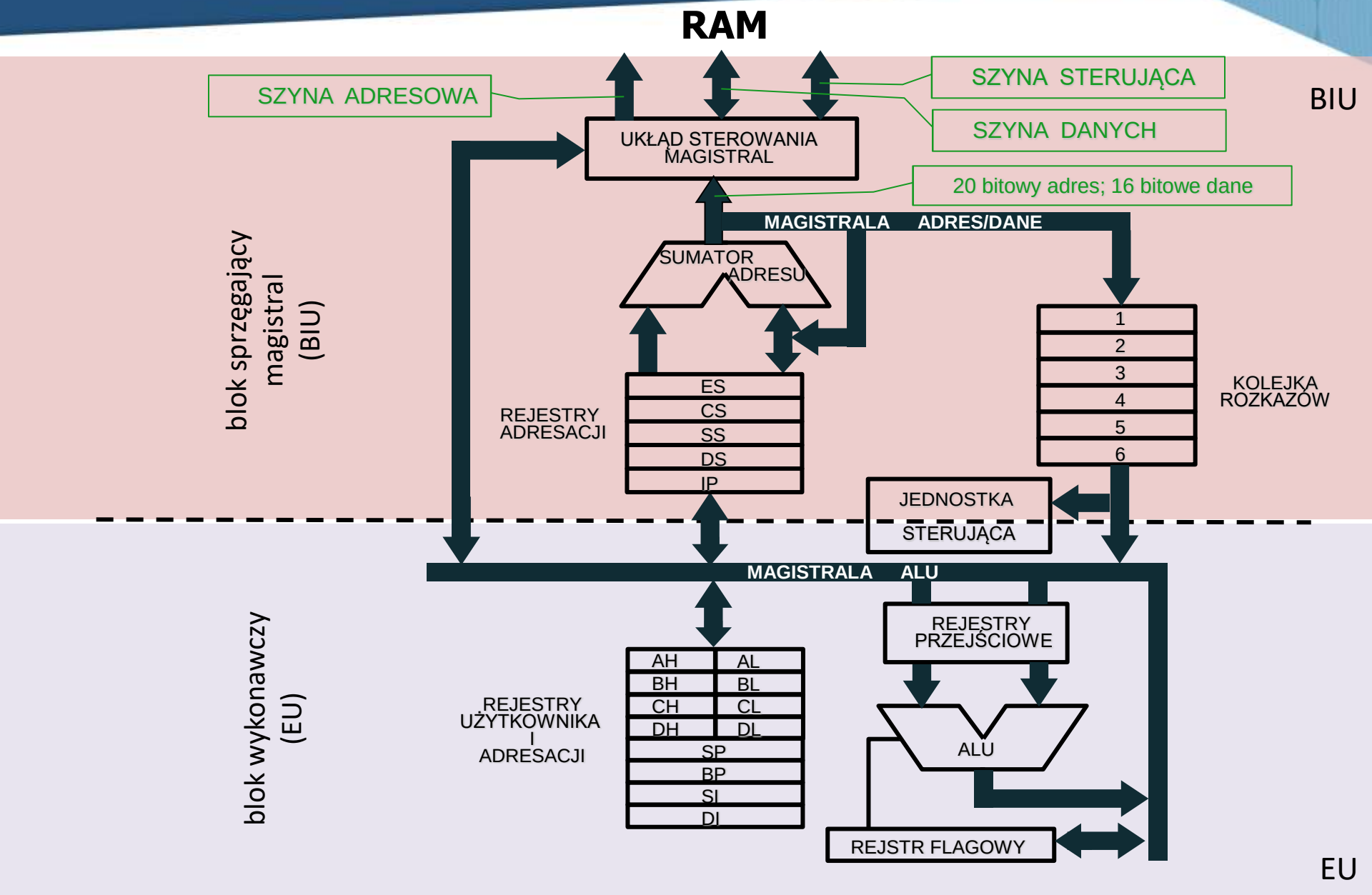



- Adres fizyczny komórek pamięci obliczany jest na podstawie dwóch 16-bitowych składników tj. adresu segmentu oraz adresu efektywnego (przesunięcia).

Taki sposób adresowania nazywa się adresowaniem segmentowym.

# SCHEMAT BLOKOWY PROCESORA 8086





- Generator adresu fizycznego
  - Jest to 20-bitowy sumator służący do obliczania adresu fizycznego komórki pamięci.

- Kolejka rozkazów
  - Jest to 6 bajtowa pamięć zorganizowana w słowa (trzy 2-bajtowe komórki) wykorzystywane przez mikroprocesor do przechowywania pobranych wcześniej rozkazów.

# Generator adresu fizycznego



- Do otrzymania adresu fizycznego stosuje się tzw. generator adresu fizycznego, znajdujący się w jednostce interfejsowej.

- Adres segmentu mnożony jest przez 16, co powoduje że zostaje on przesunięty o 4 bity w lewo (zwolnione z prawej strony bity przyjmują wartość 0), a następnie dodaje się do niego adres efektywny obliczony z zastosowaniem wszystkich modyfikatorów (w przypadku danych) lub zawartość licznika rozkazów (w przypadku rozkazów).

# Generator adresu fizycznego



- *Przykład generacji adresu fizycznego*
- Adres logiczny: 0010h:000Fh (adres początku segmentu:wartość przesunięcia (tzw. adres efektywny))
- 0010h * 0010h (16 dziesiętne) = 00100h (dwudziestobitowy adres początku segmentu)
- 00100h + 000Fh = 0010Fh (adres fizyczny komórki pamięci)

# Generator adresu fizycznego



- Procesor 8086 przesuwa 16-bitowy segment tylko o cztery bity przed dodaniem go do 16-bitowego offsetu

  ( 16×segment + offset )

- tworząc w ten sposób 20-bitowy zewnętrzny (lub efektywny lub fizyczny) adres z 32-bitowej pary segment:przesunięcie.

- W rezultacie do każdego adresu zewnętrznego można się odnieść przez 2^12 = 4096 różnych par segment:przesunięcie.

```
  0110 1000 1000 0111  0000 Segment, 16 bits, shifted 4 bits left (or multiplied by 0x10)
+       0011 0100 1010 1001 Offset,     16 bits
-------------------------------
  0110 1011 1101 0001 1001 Address, 20 bits
```

# Tryby adresowania



- Trybem adresowania to sposób wyznaczania adresu argumentów i wynikiów operacji. W mikroprocesorze 8086 każdy z rejestrów ogólnego przeznaczenia może służyć do przechowywania adresu lub jego składnika. Adres operandu obliczany jest zgodnie z równaniem
- $EA = BR + IR + p$

przy czym:

- *EA* – adres efektywny,
- *BR* – rejestr bazowy,
- *IR* – rejestr indeksowy,
- *p* – przemieszczenie.

# Tryby adresowania



- Adres efektywny *EA* jest adresem logicznym „widzianym” przez program.
- Na podstawie *EA* układy segmentacji obliczają adres rzeczywisty w pamięci operacyjnej.
- Rejestrem bazowym może być rejestr *BP* lub *BX*, a rejestrem indeksowym może być rejestr *SI* lub *DI*.
- Przemieszczenie jest zawarte w rozkazie i może mieć długość ośmiu lub szesnastu bitów.

# Tryby adresowania



- Mikroprocesor 8086 realizuje następujące tryby adresowania:
  - natychmiastowe,
  - rejestrowe,
  - bezpośrednie,
  - pośrednie,
  - bazowe,
  - indeksowe,
  - indeksowo-bazowe

# Tryby adresowania



- Adresowanie natychmiastowe
  - W adresowaniu natychmiastowym argument pobierany jest bezpośrednio z rozkazu.
  - W tym trybie wskazywany jest wyłącznie operand źródłowy. Np. MOV AX, 20 – w rejestrze *AX* zostanie zapisana liczba 20h.

# Tryby adresowania



- Adresowanie rejestrowe
  - W adresowaniu rejestrowym operandy znajdują się w rejestrach wewnętrznych mikroprocesora.
  - Jeżeli operand znajduje się w pamięci, to zespół wykonawczy *EU* oblicza jego 16-bitowy adres (przesunięcie) wewnątrz segmentu.
  - Zespół *BIU* oblicza adres rzeczywisty na podstawie otrzymanego przesunięcia (adresu efektywnego *EA*) i zawartości wybranego rejestru segmentowego.
  - Np. MOV AX, BX – w rejestrze *AX* zostanie zapisana zawartość rejestru *BX*.
    - w naszym symulatorze tak nie można

# Tryby adresowania



- Adresowanie bezpośrednie
- W adresowaniu bezpośrednim adres operandu znajduje się bezpośrednio w rozkazie.
  - Np. MOV AX, [40] – w rejestrze *AX* zostanie zapisana zawartość komórki pamięci (segment danych) o adresie 40.
- Standardowy adres operandu jest przesunięciem w segmencie danych (*DS*), można to nadpisać poprzez wskazanie innego segmentu.
  - Np. MOV AX, CS:[40] – w rejestrze *AX* zostanie zapisana zawartość z komórki pamięci (segment PROGRAMU(kodu)) o offsecie 40.

# Tryby adresowania



- Adresowanie pośrednie
- W tym trybie odwołujemy się do jednego z rejestrów roboczych procesora (np. *BX*) lub do komórki pamięci (np. 19).
- W rejestrze (*BX*) zapisany jest numer komórki pamięci, do której trzeba sięgnąć aby odczytać tam zawarty adres i przenieść do drugiego rejestru (*AX*).
- Dla adresowania pośredniego z pamięci odczytujemy numer komórki pamięci z dwóch komórek (komórki 19 i komórki 20) w taki sposób, że zawartość tej pierwszej (19) stanowi ważniejszą część tego numeru, zaś zawartość drugiej komórki (20) mniej ważną część tego numeru.

# Tryby adresowania



- Adresowanie pośrednie (cd)
- Dalej postępujemy podobnie jak przy adresowaniu pośrednim z rejestru – przenosimy zawartość do rejestru *AX*.
  - Np. MOV AX, [CX] – w rejestrze AX zostanie zapisana zawartość komórki pamięci o adresie, który znajduje się w rejestrze *CX*.
- Wszystkie rejestry wskazują offset w segmencie danych (*DS*), poza rejestrem *BP*, który jest przesunięciem w segmencie stosu (*SS*). Można to zmienić określając segment w rozkazie.
  - Np. MOV AX, SS:[CX] – w rejestrze *AX* zostanie zapisana zawartość komórki pamięci z segmentu stosu o adresie, który znajduje się w rejestrze *CX*.

# Tryby adresowania



- Adresowanie bazowe

- jest to rodzaj adresowania pośredniego, gdzie rozkaz wskazuje na jeden z rejestrów bazowych *BX* lub *BP* i może zawierać 8- lub 16-bitową wartość stanowiącą lokalne przemieszczenie.

- Adresem efektywnym jest suma zawartości rejestru bazowego i lokalnego przemieszczenia. Np. MOV AX, [BP+2].

# Tryby adresowania



- Adresowanie indeksowe jest rodzajem adresowania pośredniego, gdzie adres efektywny jest sumą zawartości rejestru indeksowego *SI* lub *DI* i lokalnego przemieszczenia. Np. MOV AX, [SI+3].

# Tryby adresowania



- Adresowanie bazowo-indeksowe

- W adresowaniu bazowo-indeksowym, adres efektywny jest sumą zawartości jednego z rejestrów bazowych, jednego z rejestrów indeksowych i lokalnego przemieszczenia.
  - Np. MOV AX, [SI+BP+4].

# Rozkazy (instrukcje)



- Rozkazy operujące na ciągach słów
  - Rozkazy operujące na ciągach słów posługują się rejestrami indeksowymi.
  - Rejestry *SI* i *DI* zawierają adresy efektywne pierwszego słowa odpowiednio w ciągu źródłowym i wynikowym.
  - Po każdej transmisji rejestry indeksowe są automatycznie inkrementowane lub dekrementowane w zależności od ustawienia bitu *DF* w rejestrze znaczników.
- Rozkazy operujące na rejestrach WE/WY
  - Rozkazy operujące na rejestrach WE/WY zawierają adres WE/WY (adres natychmiastowy) lub posługują się zawartością rejestru *DX* (adresowanie pośrednie).

# Rozkazy (instrukcje)



- Grupy rozkazów mikroprocesora 8086:
  - arytmetyczno-logiczne
  - przesłań
  - skoków, obsługi pętli, wywołań i powrotów z podprogramu
  - dotyczące rejestrów segmentowych
  - wykonujące operacje na ciągach słów
  - wejścia/wyjścia
  - inne
- Liczba bajtów każdego rozkazu zależy od jego rodzaju i może wynosić od jednego do sześciu.

# Rozkazy (instrukcje)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| kod operacji | | | | | | D | W |
| MOD | | REG | | | R/M | | |



- Pierwszy bajt zawiera sześciobitowy kod operacji oraz dwa bity (kierunku i szerokości). Bit *D* określa kierunek transmisji (*0* – wynik operacji jest przesyłany z rejestru do pamięci, *1* – z pamięci do rejestru).
- W zależności od wartości tego bitu w rozkazie rozróżniane są operandy źródłowe i operandy przeznaczenia.
- Bit *W* określa szerokość operandu danego rozkazu (0 – operacje bajtowe, 1 – operacje na słowie 16-bitowym).
- Jeżeli rozkaz jest wielobajtowy, to drugi bajt rozkazu określa sposób adresowania argumentów. Zawiera trzy grupy bitów:
  - dwubitowa grupa *MOD* określa tryb adresowania
  - trzybitowa grupa *REG* określa numer rejestru, w którym znajduje się operand
  - trzybitowa grupa *R/M* określa sposób wyznaczenia miejsca operandu

# Rozkazy (instrukcje)



- format rozkazu sześciobajtowego:
  - dwa ostatnie bajty to argument natychmiastowy dla tego rozkazu.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MOD | | REG | | | R/M | | | kod operacji | | | | | | D | W |
| przemieszczenie | | | | | | | | | | | | | | | |
| dana | | | | | | | | | | | | | | | |

- są także rozkazy jednobajtowe, np. rozkaz wymiany zawartości akumulatora z zawartością wybranego rejestru.
- W kodzie tego rozkazu pięć bitów stanowi kod operacji, a pozostałe trzy bity wskazują numer rejestru, którego ten rozkaz dotyczy.

# Przykład prostego kodu



```
                        ; _memcpy(dst, src, len)
                        ; Copy a block of memory from one location to another.
                        ;
                        ; Entry stack parameters
                        ;       [BP+6] = len, Number of bytes to copy
                        ;       [BP+4] = src, Address of source data block
                        ;       [BP+2] = dst, Address of target data block
                        ;
                        ; Return registers
                        ;       AX = Zero

0000:1000                           org     1000h       ; Start at 0000:1000h

0000:1000               _memcpy     proc
0000:1000 55                        push    bp          ; Set up the call frame
0000:1001 89 E5                     mov     bp,sp
0000:1003 06                        push    es          ; Save ES
0000:1004 8B 4E 06                  mov     cx,[bp+6]   ; Set CX = len
0000:1007 E3 11                     jcxz    done        ; If len = 0, return
0000:1009 8B 76 04                  mov     si,[bp+4]   ; Set SI = src
0000:100C 8B 7E 02                  mov     di,[bp+2]   ; Set DI = dst
0000:100F 1E                        push    ds          ; Set ES = DS
0000:1010 07                        pop     es

0000:1011 8A 04         loop        mov     al,[si]     ; Load AL from [src]
0000:1013 88 05                     mov     [di],al     ; Store AL to [dst]
0000:1015 46                        inc     si          ; Increment src
0000:1016 47                        inc     di          ; Increment dst
0000:1017 49                        dec     cx          ; Decrement len
0000:1018 75 F7                     jnz     loop        ; Repeat the loop

0000:101A 07            done        pop     es          ; Restore ES
0000:101B 5D                        pop     bp          ; Restore previous call frame
0000:101C 29 C0                     sub     ax,ax       ; Set AX = 0
0000:101E C3                        ret                 ; Return
0000:101F                           end proc
```

# Jeszcze prostszy kod



- Procesor 8086 zapewnia dedykowane instrukcje do kopiowania ciągów bajtów.
- Instrukcje te zakładają, że dane źródłowe są przechowywane w DS:SI, dane docelowe są przechowywane w ES:DI, a liczba elementów do skopiowania jest przechowywana w CX.
- Poniższa procedura wymaga, aby blok źródłowy i docelowy znajdowały się w tym samym segmencie, dlatego DS jest kopiowany do ES.
- Poprzedni kod może być zastąpiony przez:

```
0000:1011 FC            cld             ; Copy towards higher addresses
0000:1012 F3    loop    rep             ; Repeat until CX = 0
0000:1013 A4            movsb           ; Move the data block
```

- Kopiowane jest po jednym bajcie na raz.

# Jeszcze prostszy kod



- Instrukcja REP powoduje, że następujący MOVSB powtarza się, aż CX wynosi zero, automatycznie zwiększając SI i DI oraz zmniejszając CX w miarę powtarzania.
- Alternatywnie instrukcja MOVSW może być użyta do jednoczesnego kopiowania 16-bitowych słów (podwójne bajty) (w takim przypadku CX zlicza liczbę skopiowanych słów zamiast liczby bajtów).
- Można też  REP MOVSB   w jednej lini

```
0000:1011 FC            cld             ; Copy towards higher addresses
0000:1012 F3  loop      rep             ; Repeat until CX = 0
0000:1013 A4            movsb           ; Move the data block
```

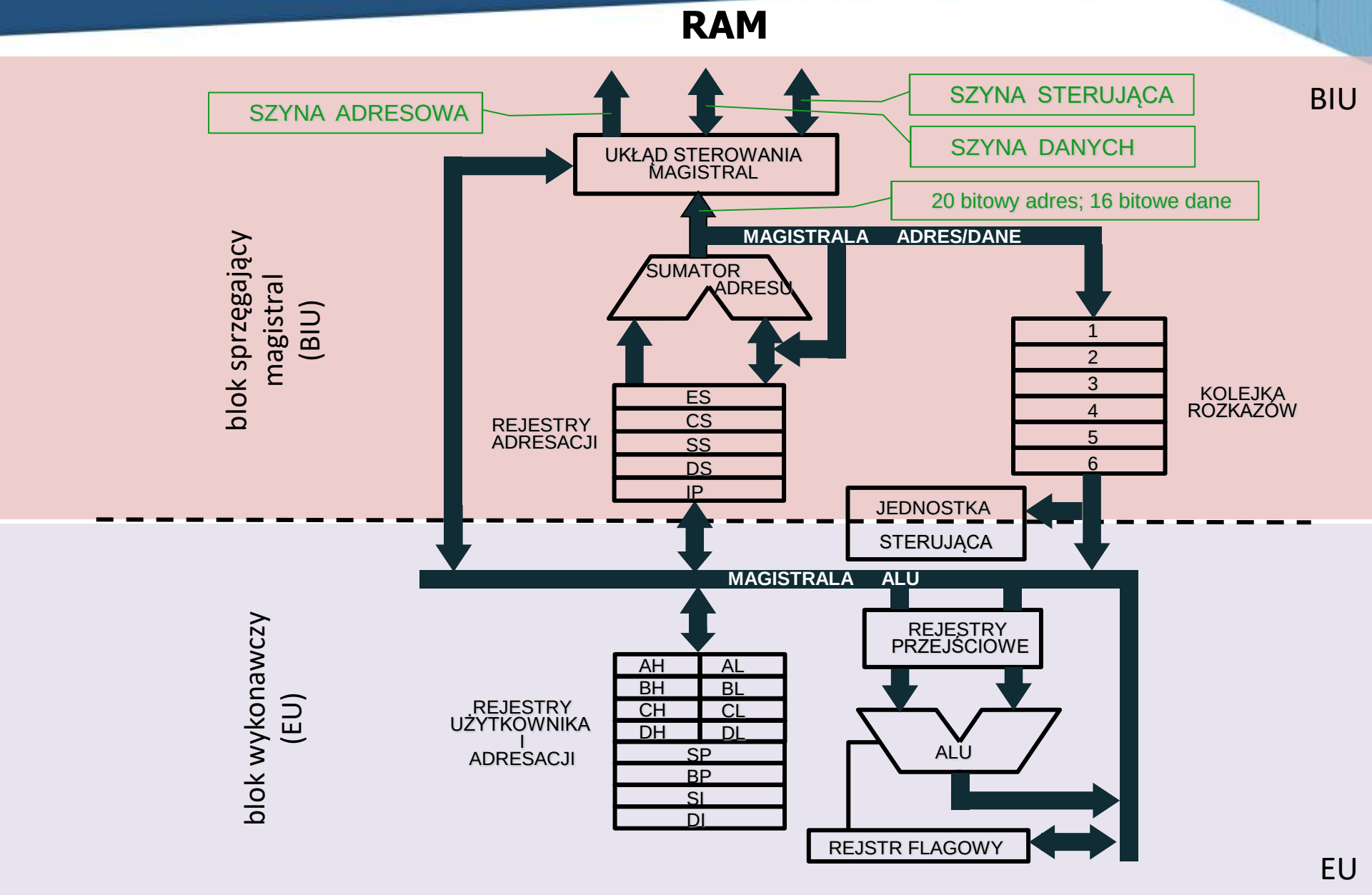
Ale te proste kody muszą być przetłumaczone na język maszynowy bezpośrednio do wykonania

- Czasy wykonania typowych instrukcji (w cyklach zegara)
- EA = czas na obliczenie efektywnego adresu, w zakresie od 5 do 12 cykli. ALU to elementarne instrukcje artym-log

| instruction | register-register | register immediate | register-memory | memory-register | memory-immediate |
|---|---|---|---|---|---|
| mov | 2 | 4 | 8+EA | 9+EA | 10+EA |
| ALU | 3 | 4 | 9+EA, | 16+EA, | 17+EA |
| jump | *register* ≥ 11 ; *label* ≥ 15 ; *condition,label* ≥ 16 | | | | |
| integer multiply | 70~160 (depending on operand *data* as well as size) *including* any EA | | | | |
| integer divide | 80~190 (depending on operand *data* as well as size) *including* any EA | | | | |

# SCHEMAT BLOKOWY PROCESORA 8086

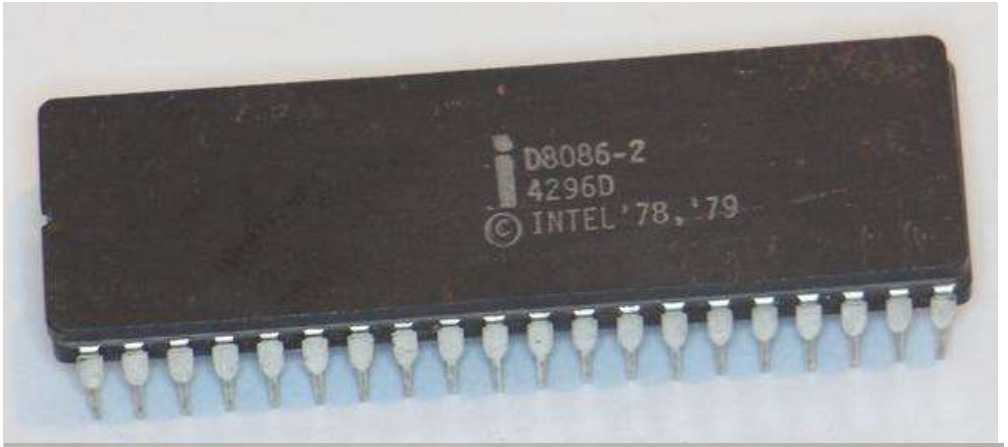


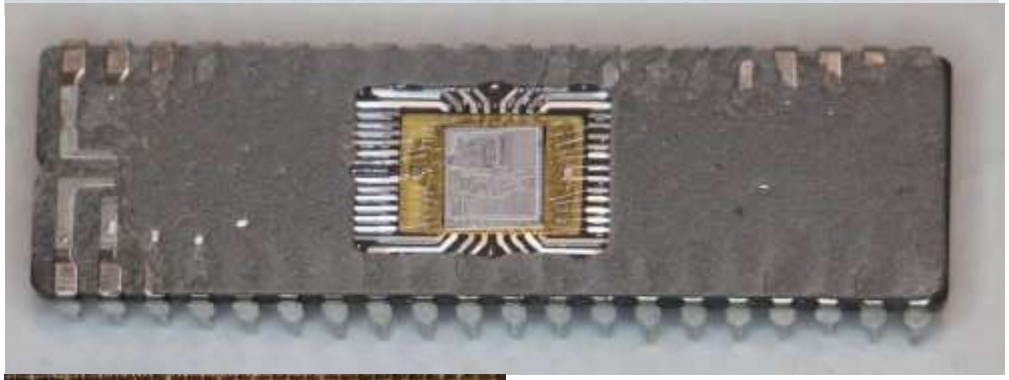


# Co jest w środku procesora 8086



źródło http://www.righto.com/2020/06/a-look-at-die-of-8086-processor.html

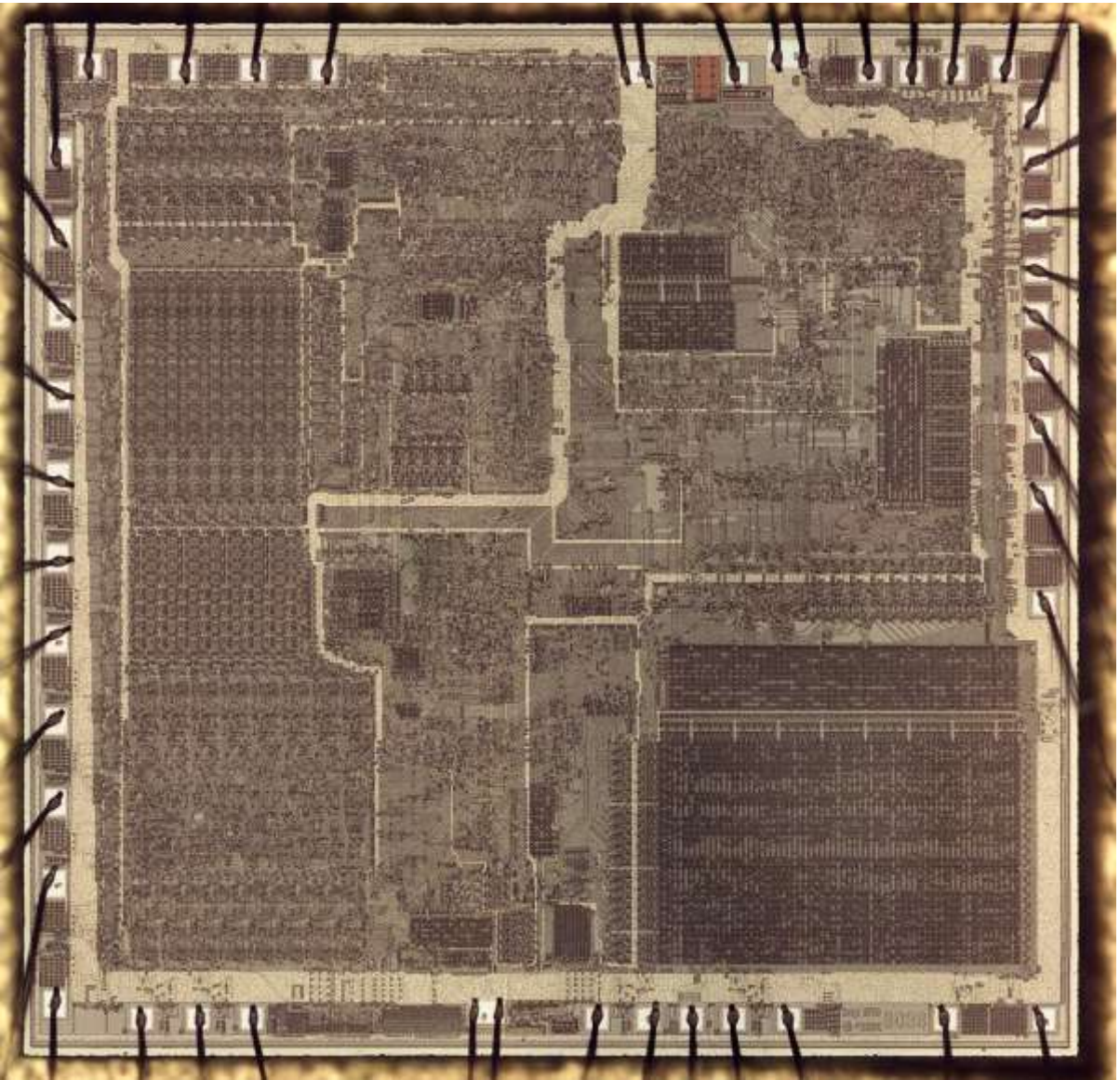



- Po zdjęciu obudowy silikonowa matryca jest widoczna pośrodku.
- Matryca jest połączona z metalowymi pinami chipa za pomocą maleńkich drutów łączących. Jest to 40-pinowy pakiet DIP, standardowe opakowanie dla mikroprocesorów w tamtym czasie.Sama matryca krzemowa zajmuje niewielką część rozmiaru chipa.
- Pod mikroskopem widoczny jest numer katalogowy 8086 oraz data praw autorskich. Przewód łączący jest podłączony do podkładki. Część pamięci ROM z mikrokodem znajduje się na górze.

# Co jest w środku procesora 8086

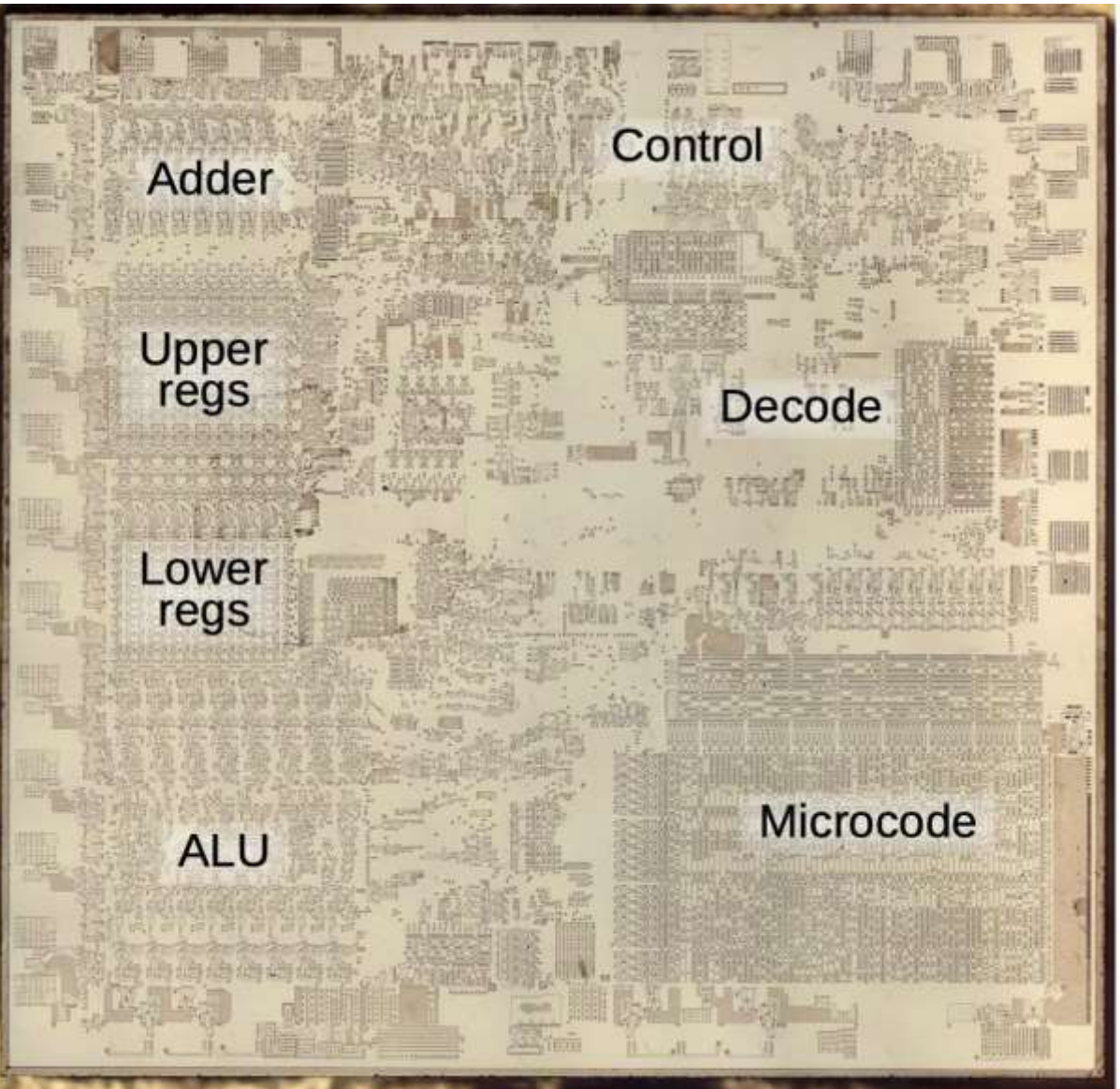




- widoczna jest metalowa warstwa chipa, w większości zasłaniająca znajdujący się pod nią krzem.
- Wokół krawędzi matrycy cienkie druty łączące zapewniają połączenia między padami na chipie a zewnętrznymi pinami.
- (Podkładki zasilania i uziemienia mają po dwa przewody łączące, aby wspierać wyższy prąd.)
- Układ był złożony jak na swoje czasy i zawierał 29 tys. tranzystorów.

# Co jest w środku procesora 8086

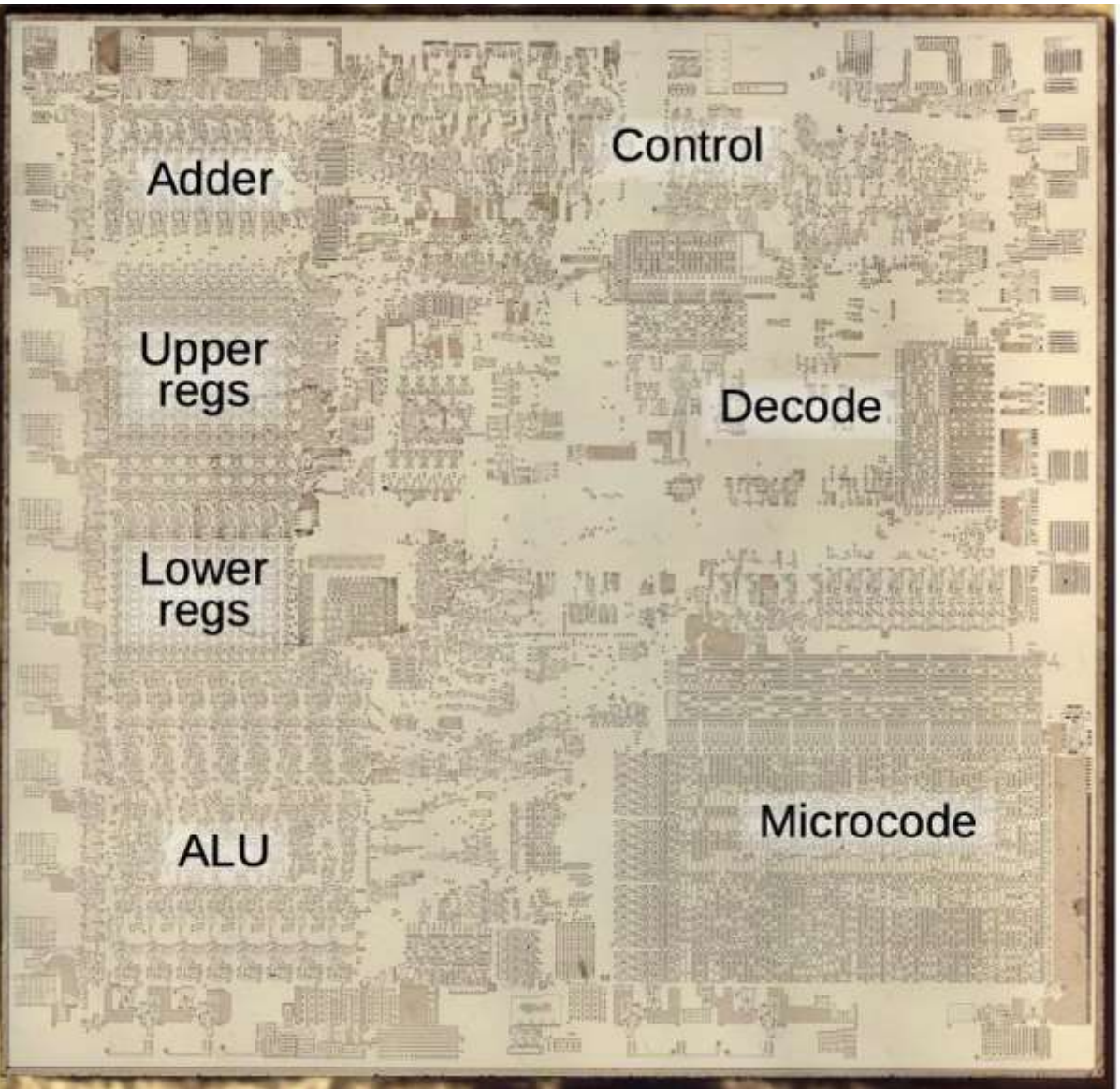





- warstwy metalu i polikrzemu zostały usunięte, ukazując leżący pod spodem krzem z 29 tys.  tranzystorów
- Etykiety pokazują główne bloki funkcjonalne, oparte na tzw. inżynierii odwrotnej.
- Lewa strona układu zawiera 16-bitową ścieżkę danych: rejestry układu i obwody arytmetyczne..

# Co jest w środku procesora 8086

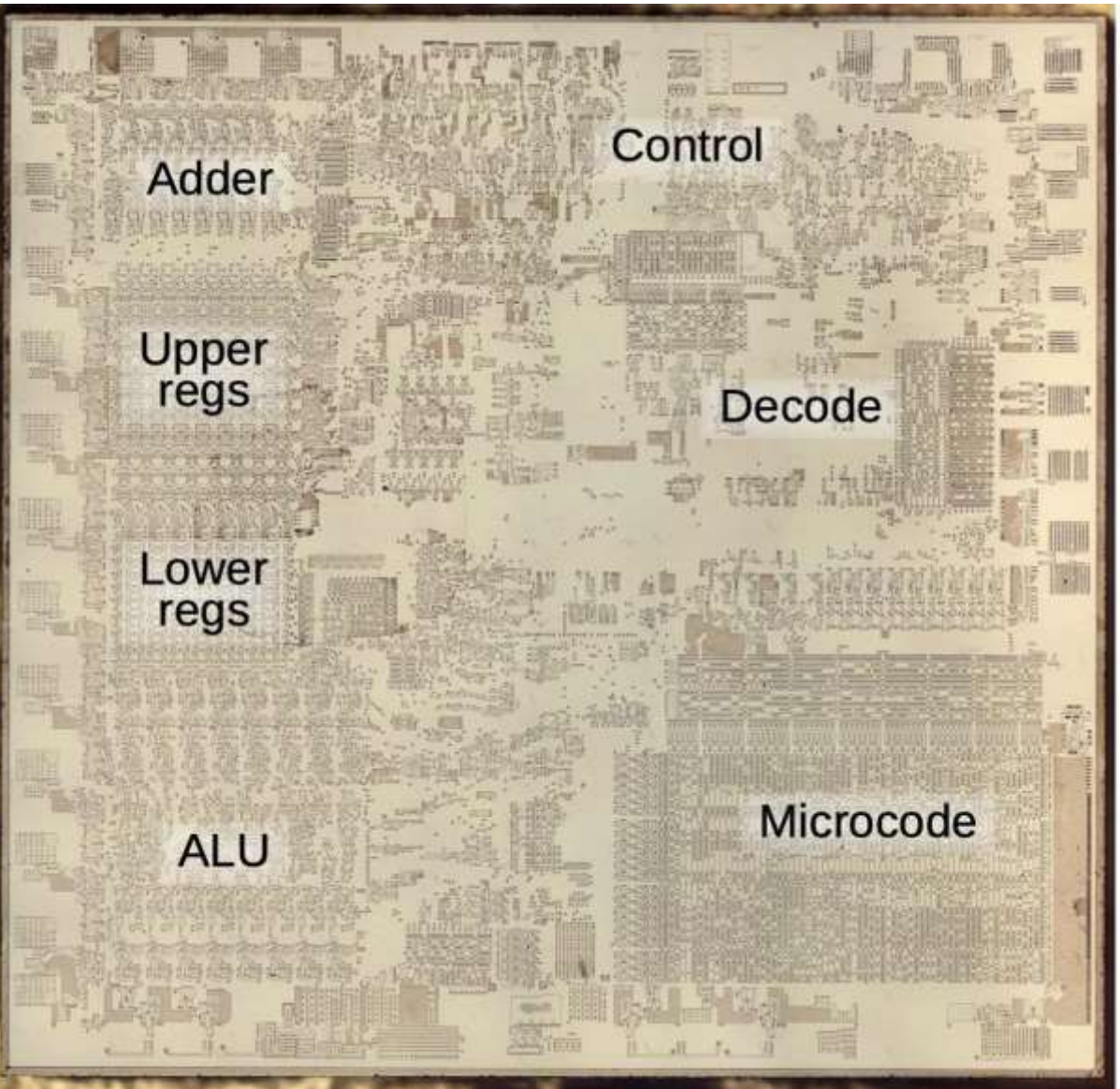





- Rejestry sumujące i rejestry górne tworzą Jednostkę Interfejsu Magistrali, która komunikuje się z pamięcią zewnętrzną RAM, podczas gdy rejestry dolne i ALU tworzą Jednostkę Wykonawczą przetwarzającą dane.
- Po prawej stronie chipa znajdują się obwody sterujące i dekodowanie instrukcji, a także mikrokod ROM, który kontroluje każdą instrukcję.

# Co jest w środku procesora 8086

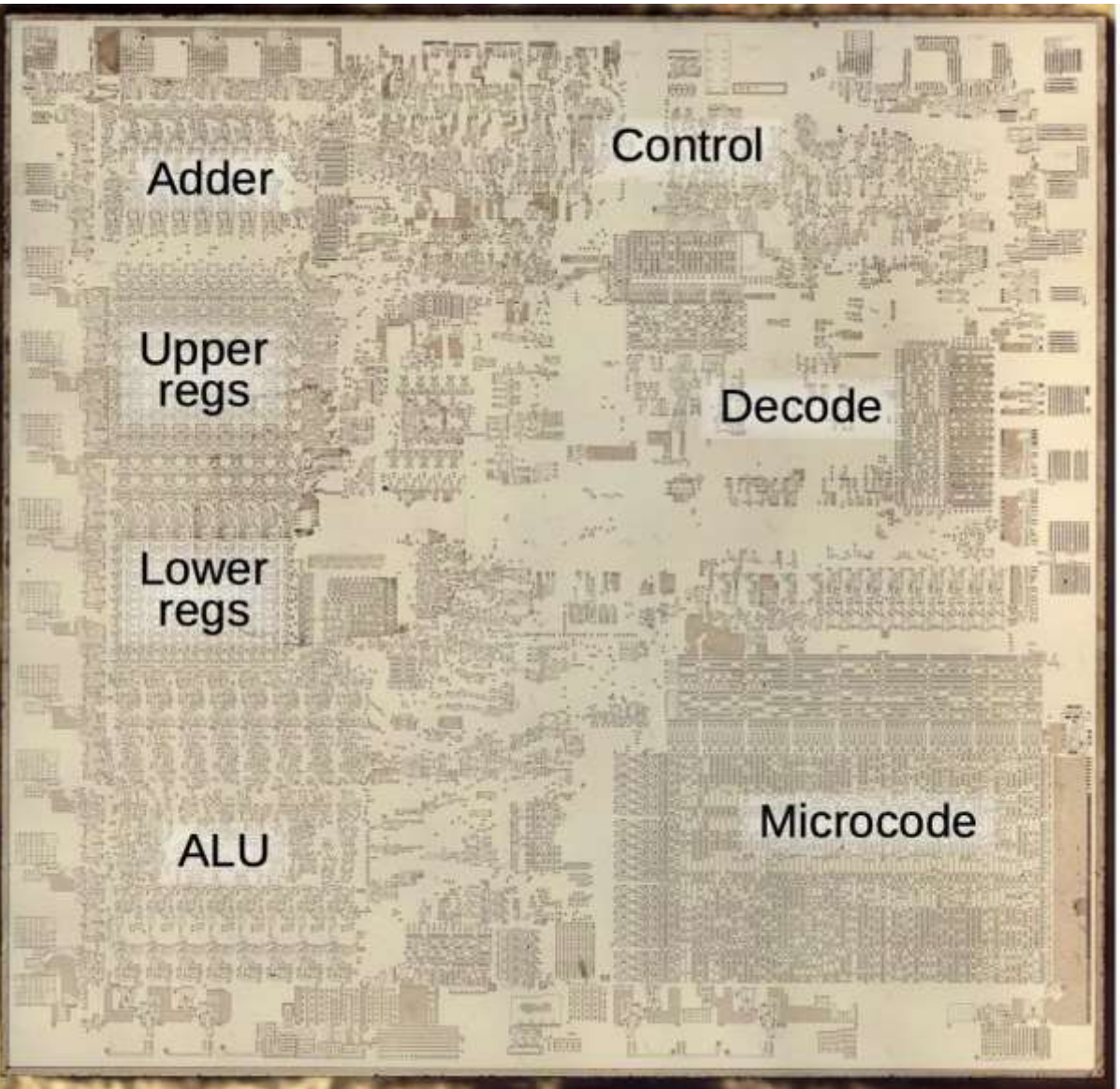





- Ważną cechą 8086 było pobieranie instrukcji z wyprzedzeniem, które poprawiało wydajność poprzez pobieranie instrukcji z pamięci, zanim były potrzebne.
- Zostało to zaimplementowane przez moduł interfejsu magistrali w lewym górnym rogu, który miał dostęp do pamięci zewnętrznej.

# Co jest w środku procesora 8086

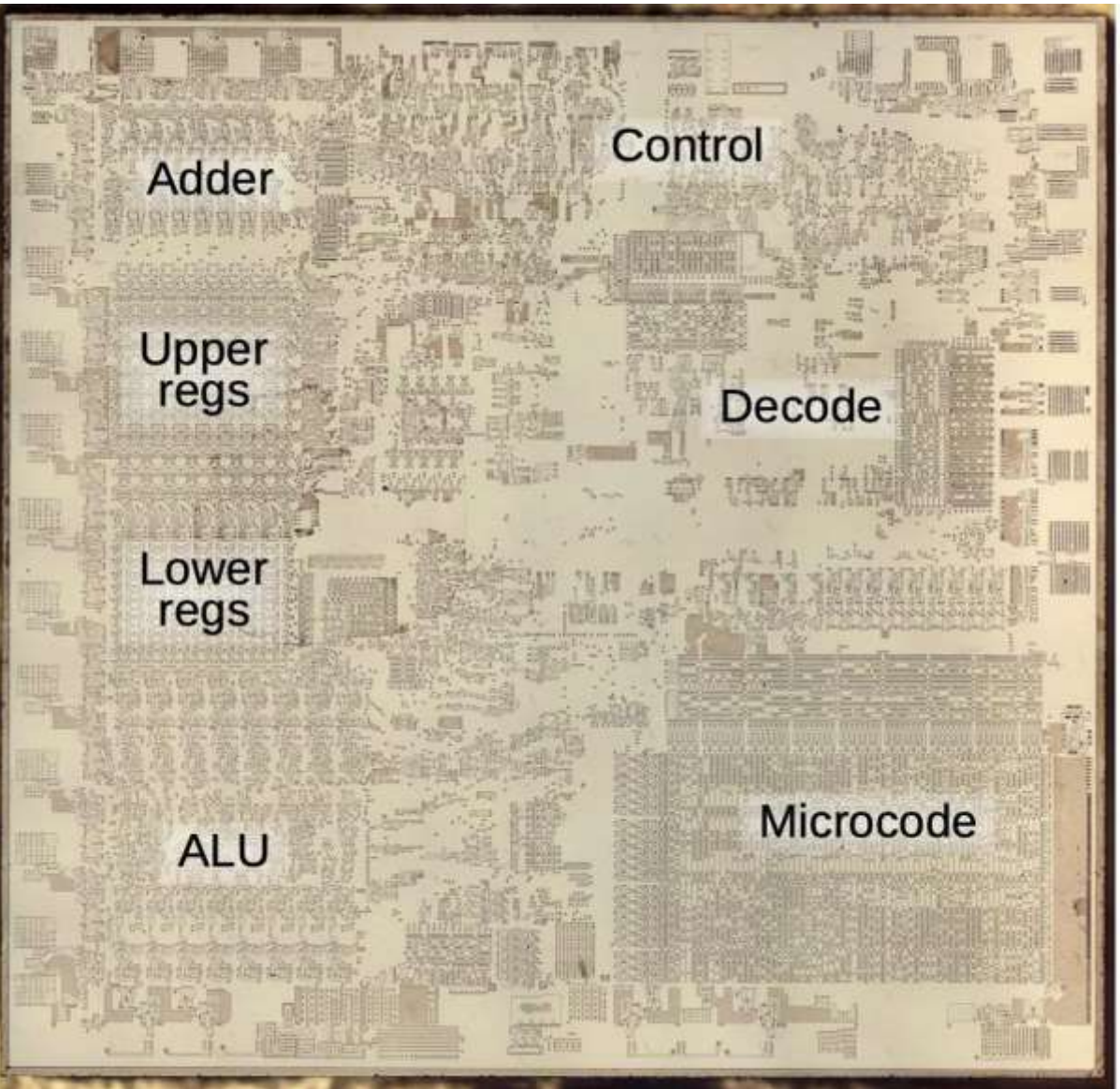





- Górne rejestry obejmują niesławne rejestry segmentowe 8086, które zapewniały dostęp do większej przestrzeni adresowej niż 64 kilobajty dozwolone przez 16-bitowy adres.
- Dla każdego dostępu do pamięci dodano rejestr segmentowy i przesunięcie pamięci, aby utworzyć ostateczny adres pamięci.

# Co jest w środku procesora 8086

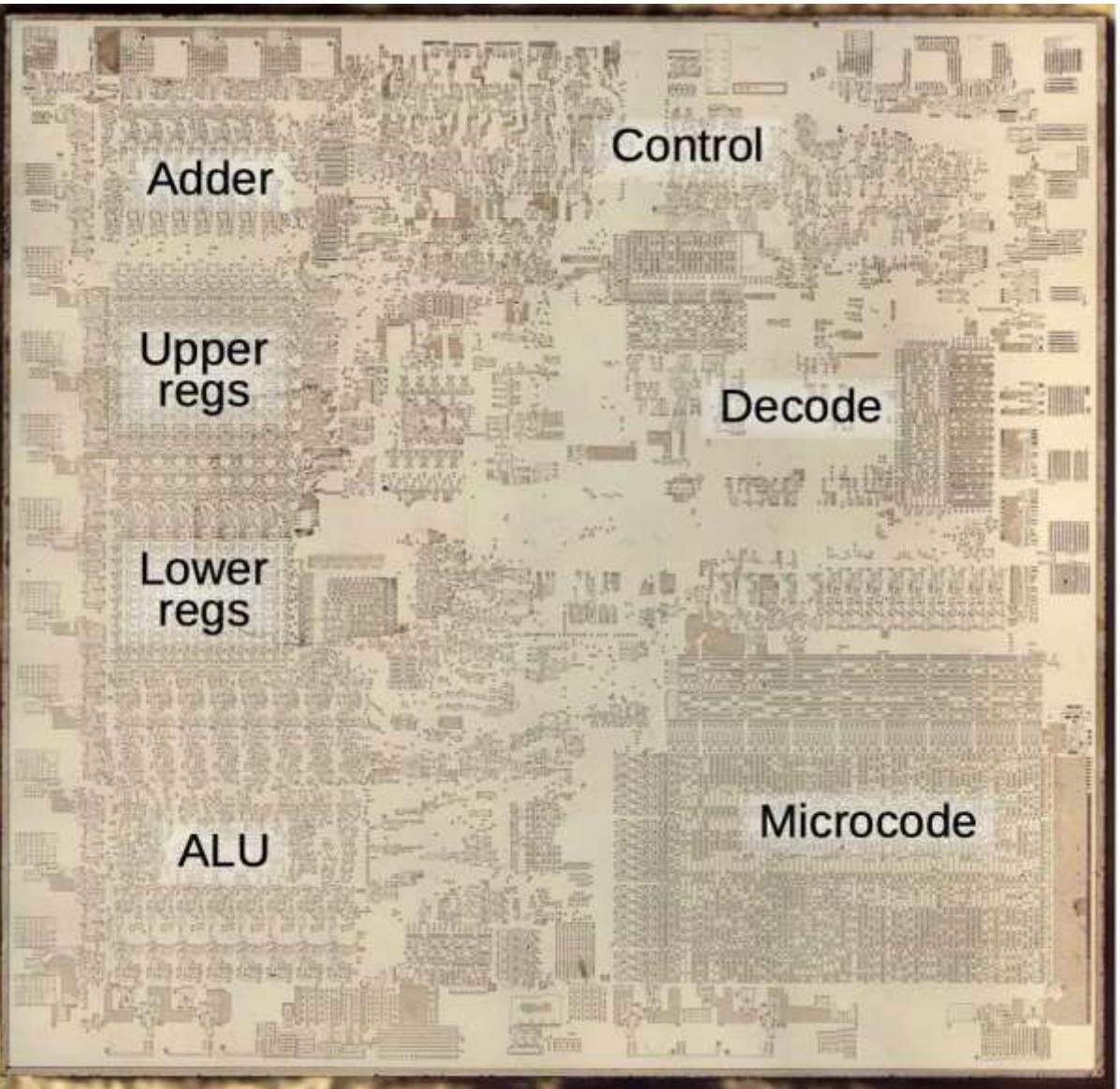





- Aby poprawić wydajność, 8086 miał oddzielny sumator (Adder) do tych obliczeń adresów pamięci, zamiast używać jednostki ALU.
- Górne rejestry zawierają również sześć bajtów bufora wstępnego pobierania instrukcji i licznik programu.

# Co jest w środku procesora 8086

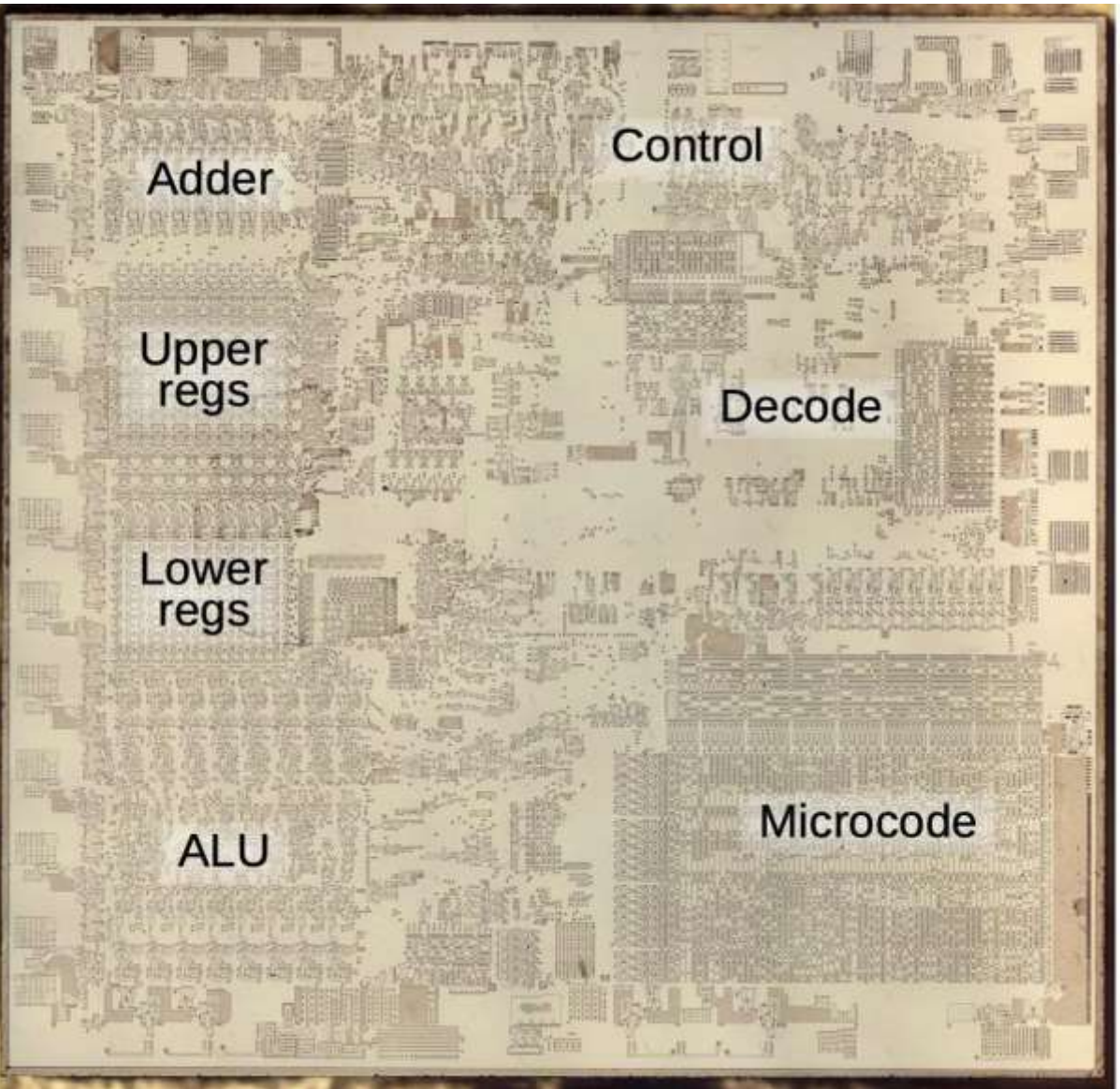





- W lewym dolnym rogu chipa znajduje się Jednostka Wykonawcza, która wykonuje operacje na danych.
- Niższe rejestry obejmują rejestry ogólnego przeznaczenia i rejestry indeksowe, takie jak wskaźnik stosu.
- 16-bitowa jednostka ALU wykonuje operacje arytmetyczne (dodawanie i odejmowanie), operacje logiczne i przesunięcia.
- w ALU nie ma mnożenia ani dzielenia; operacje te są wykonywane przez sekwencję przesunięć i dodawania/odejmowania, więc są stosunkowo powolne.

# Co jest w środku procesora 8086



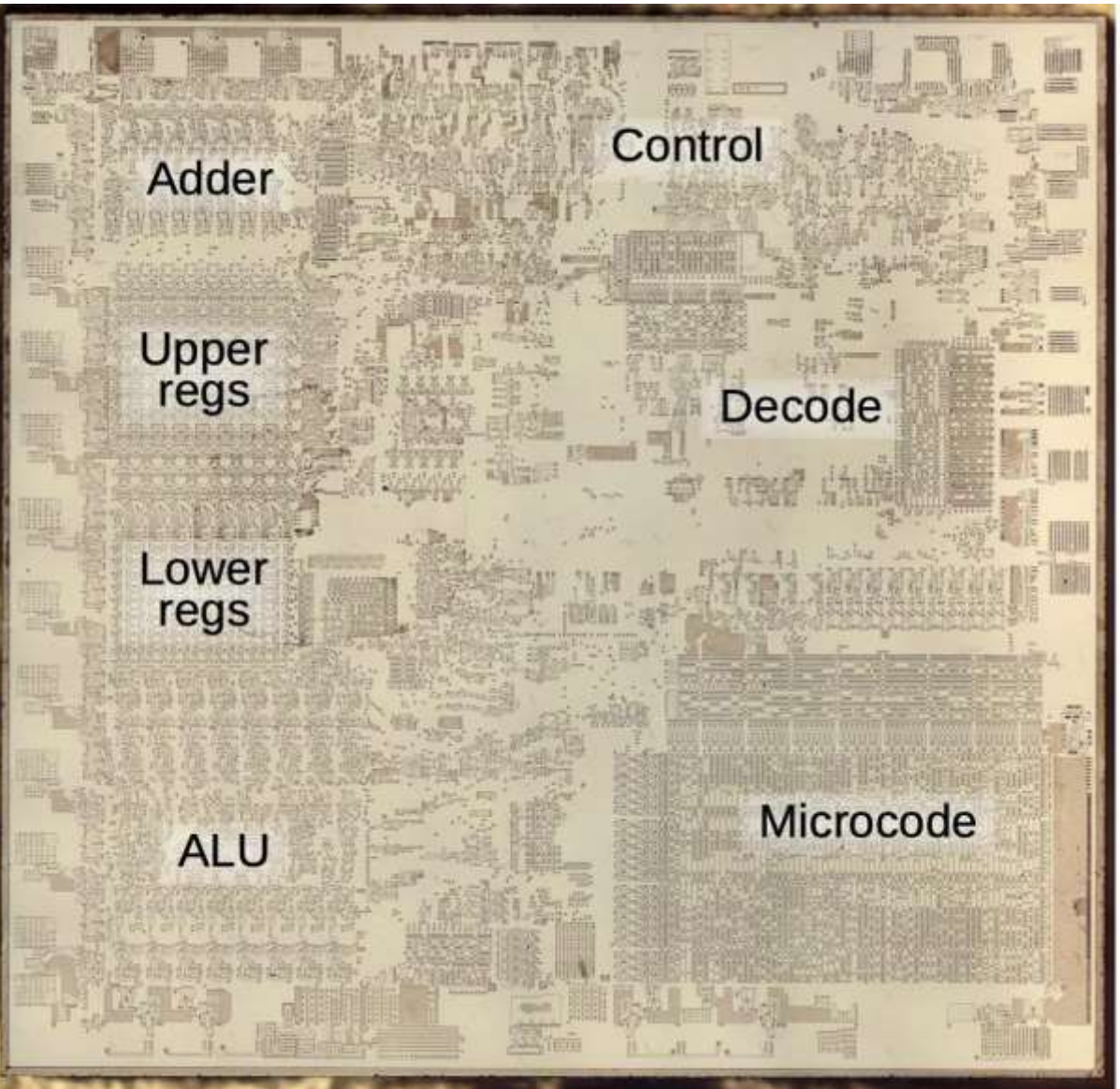



- Microcode. Jedną z najtrudniejszych części projektowania komputera jest stworzenie logiki sterującej, która mówi każdej części procesora, co zrobić, aby wykonać każdą instrukcję.

  W 1951 roku Maurice Wilkes wpadł na pomysł mikrokodu: zamiast budować logikę sterowania ze złożonych obwodów bramki logicznej, logikę sterowania można było zastąpić specjalnym kodem zwanym mikrokodem.

# Co jest w środku procesora 8086

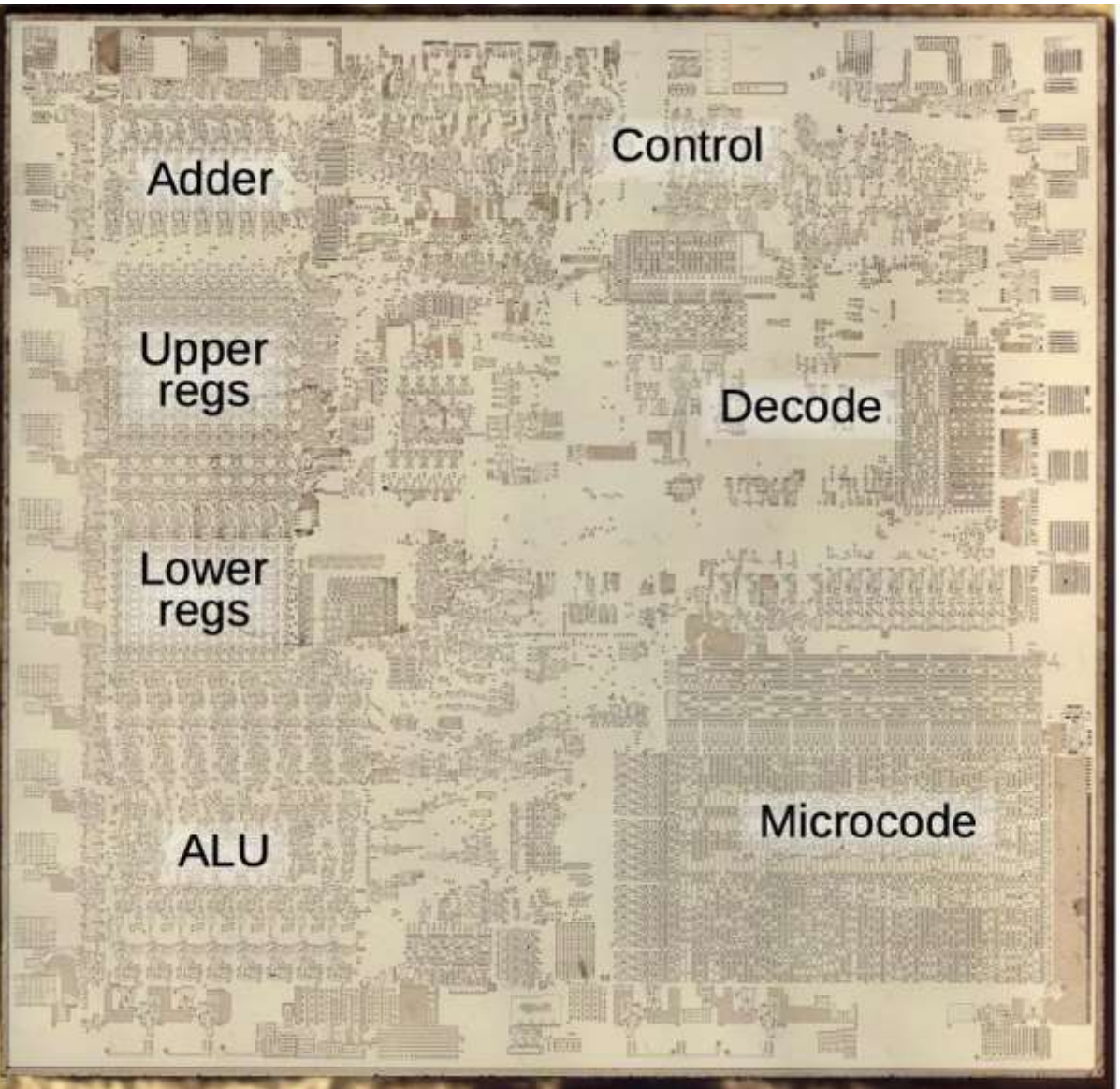





- Microcode.
- Aby wykonać instrukcję, komputer wewnętrznie wykonuje kilka prostszych mikrorozkazów, które są zdefiniowane przez mikrokod.
- Dzięki mikrokodowi budowanie logiki sterującej procesorem staje się zadaniem programistycznym, a nie zadaniem projektowania logiki układu scalonego.

# Co jest w środku procesora 8086

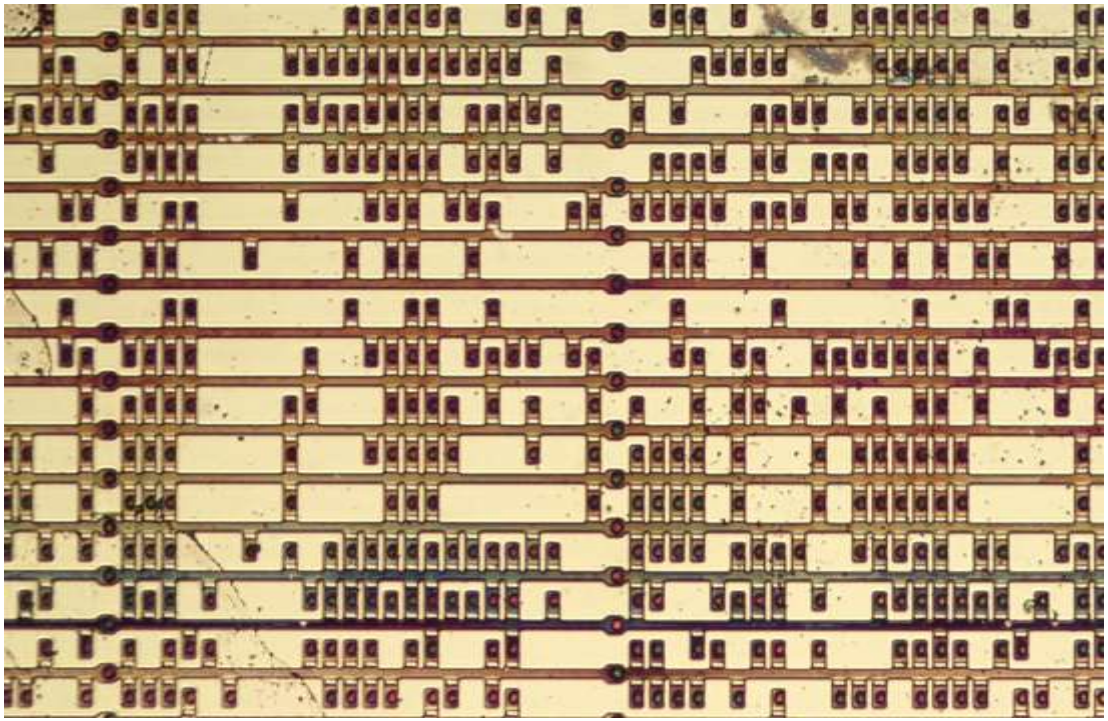





- Mikrokod był powszechny w komputerach typu mainframe w latach 60., ale wczesne mikroprocesory, takie jak 6502 i Z-80, nie wykorzystywały mikrokodu, ponieważ wczesne chipy nie miały miejsca na mikrokod.
- Jednak późniejsze układy, takie jak 8086 i 68000, wykorzystywały mikrokod, korzystając z rosnącej gęstości układów.
- To pozwoliło 8086 na zaimplementowanie złożonych instrukcji (takich jak mnożenie i kopiowanie ciągów) bez komplikowania obwodów scalonych.

# Co jest w środku procesora 8086

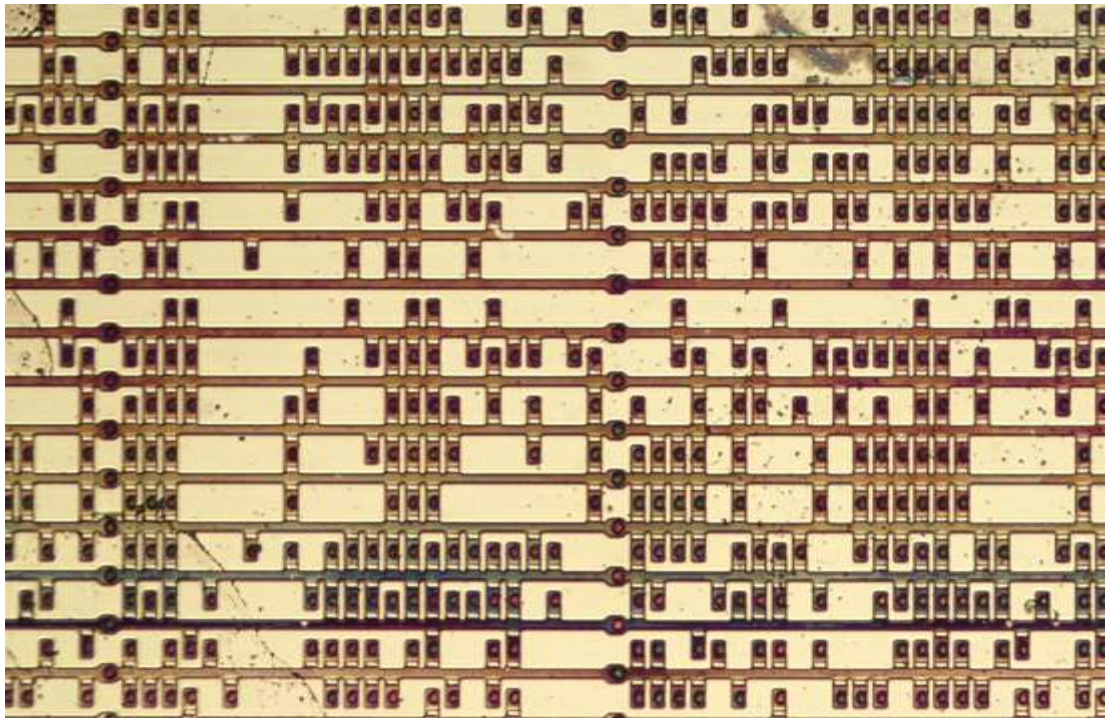




- część mikrokodu ROM widoczna pod mikroskopem Bity można odczytać na podstawie obecności lub braku tranzystorów w każdej pozycji.
- Tranzystory to małe białe prostokąty nad i/lub pod każdym ciemnym prostokątem.
- Ciemne prostokąty to połączenia z poziomymi szynami wyjściowymi w warstwie metalowej.
- Pamięć ROM składa się z 512 mikroinstrukcji, każda o szerokości 21 bitów. Mikroinstrukcja określa przepływ danych między źródłem a miejscem docelowym.

# Co jest w środku procesora 8086

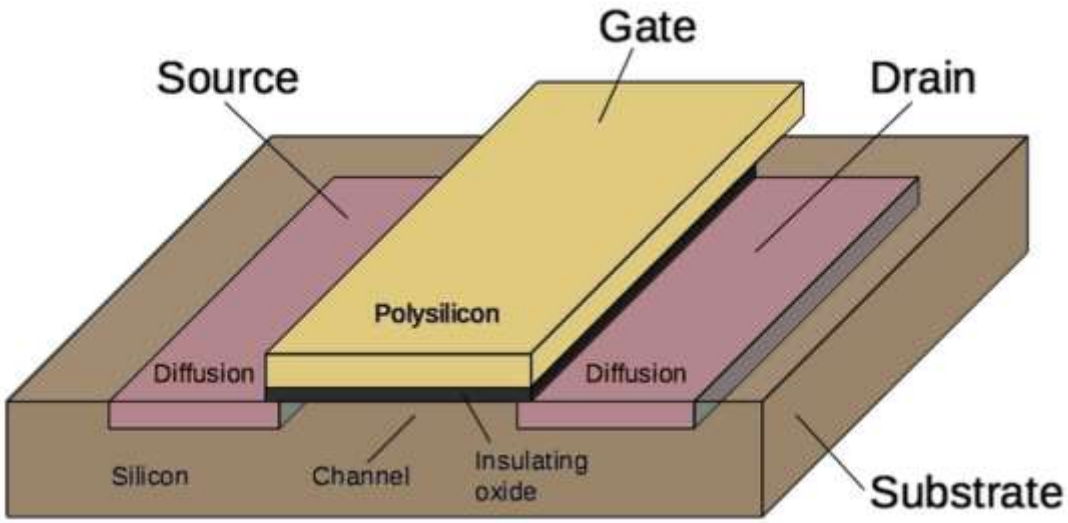




- mikroinstrukcja definiuje również mikrooperację, która może być skokiem, operacją ALU, operacją pamięci, wywołaniem podprogramu mikrokodu lub zapisem mikrokodu.
- Mikrokod jest dość wydajny; prosta instrukcja, taka jak inkrementacja lub dekrementacja, składa się z dwóch mikroinstrukcji, podczas gdy bardziej złożona instrukcja kopiowania ciągu jest zaimplementowana w ośmiu mikroinstrukcjach.

# Co jest w środku procesora 8086

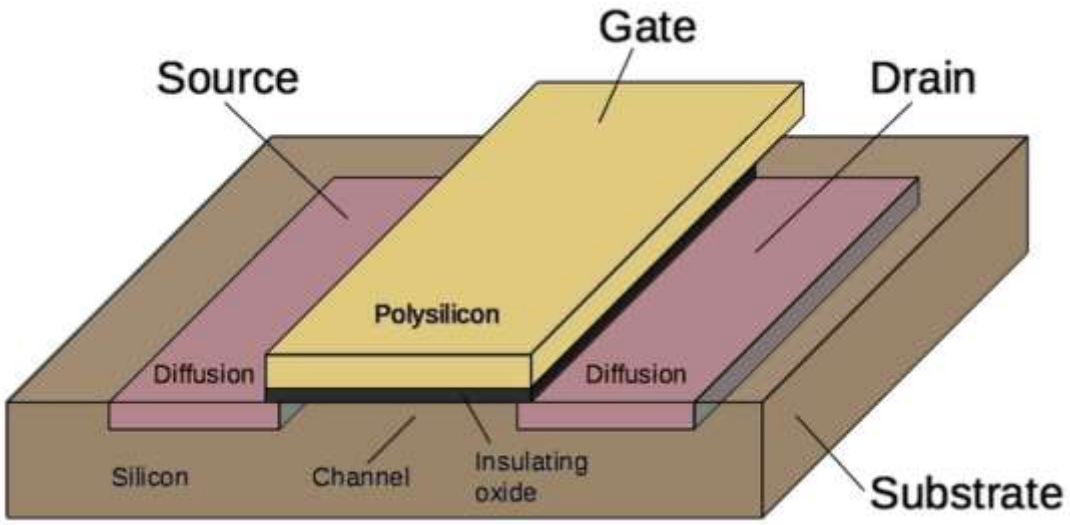





- Struktura tranzystora MOSFET (ang. *Metal-Oxide Semiconductor Field-Effect Transistor*) w układzie scalonym
- Układ 8086 został zbudowany z tranzystorów w technologii 3 µm HMOS (ang. *High performance Metal-Oxide Semiconductor)* .
- Tranzystor można uznać za przełącznik, kontrolujący przepływ prądu między dwoma obszarami zwanymi źródłem i drenem.
- Są zbudowane przez domieszkowanie obszarów podłoża krzemowego zanieczyszczeniami w celu wytworzenia obszarów „dyfuzji” o różnych właściwościach elektrycznych.

# Co jest w środku procesora 8086

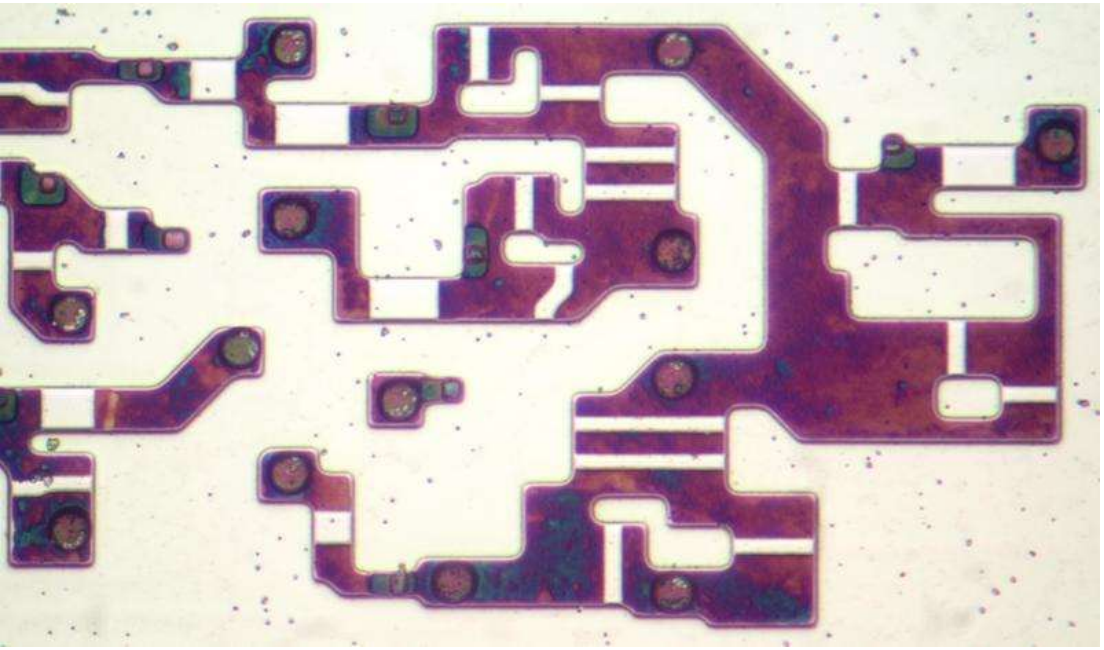





- Tranzystor jest aktywowany przez bramkę wykonaną ze specjalnego rodzaju krzemu zwanego polikrzemem, ułożonego warstwowo nad podłożem krzemowym.
- Tranzystory są połączone ze sobą metalową warstwą na górze, tworząc kompletny układ scalony.
- Podczas gdy współczesne procesory mogą mieć kilkanaście warstw metalu, 8086 miał pojedynczą warstwę metalu.

# Co jest w środku procesora 8086

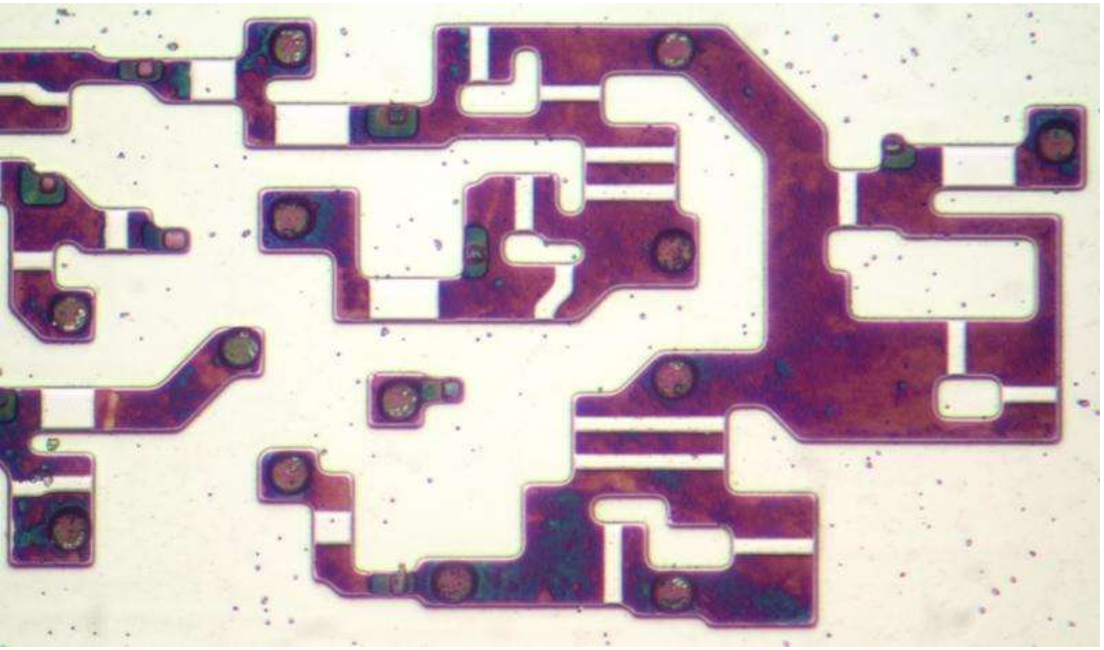




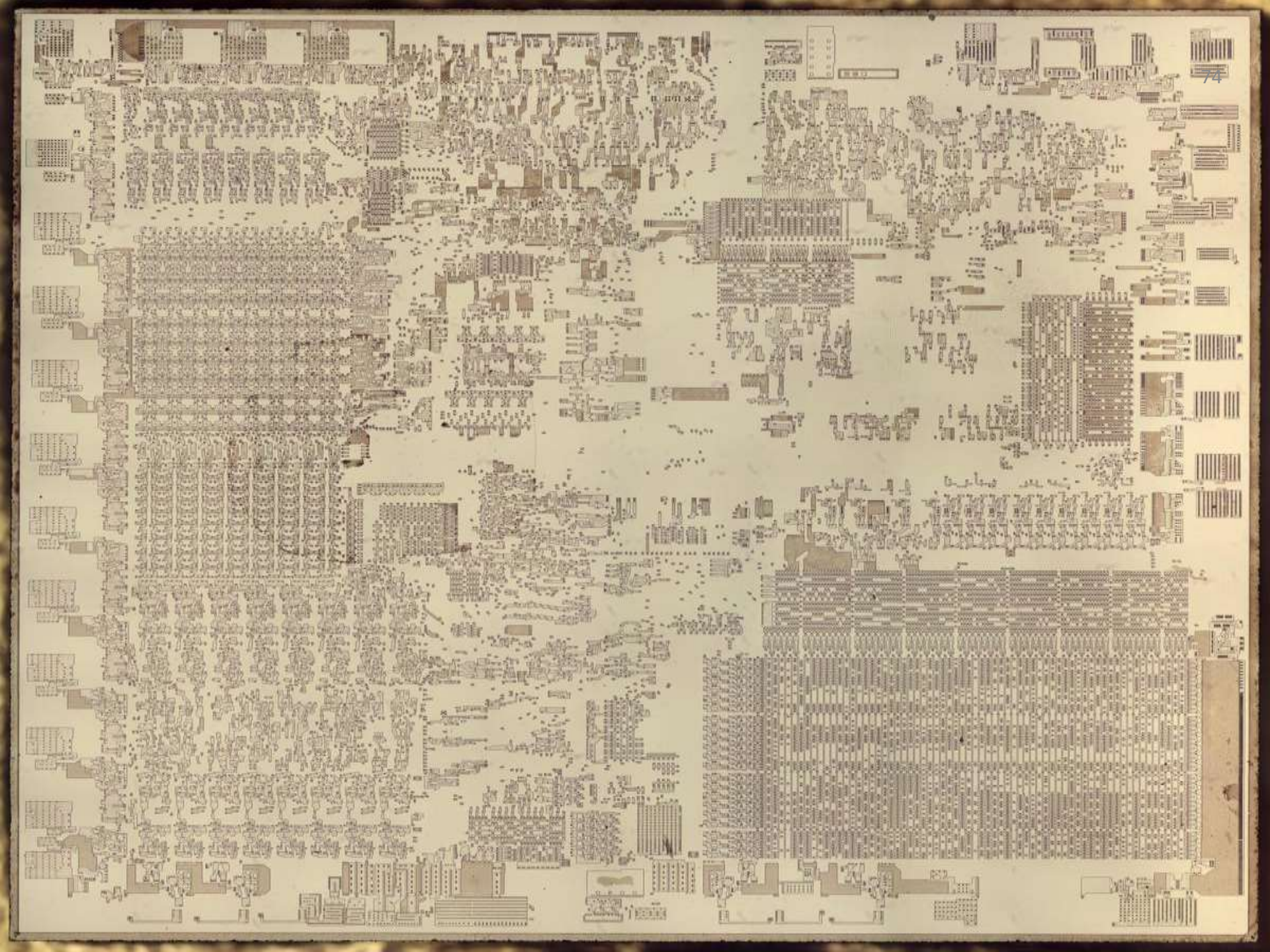
- Zdjęcie (po lewej) krzemu (z mikroskopu elektronowego) pokazuje niektóre tranzystory z jednostki arytmetyczno-logicznej (ALU).
- Domieszkowany, przewodzący krzem ma ciemnofioletowy kolor.
- Białe paski to miejsce, w którym drut polikrzemowy przecina krzem, tworząc bramkę tranzystora.
- Można policzyć, że 23 tranzystory tworzą 7 bramek.
- Tranzystory mają skomplikowane kształty, aby układ był jak najbardziej wydajny.

# Co jest w środku procesora 8086





- Ponadto tranzystory mają różne rozmiary, aby zapewnić wyższą moc tam, gdzie jest to potrzebne.
- Sąsiednie tranzystory mogą współdzielić źródło lub dren, powodując ich połączenie.
- Kółka to połączenia (tzw. przelotki) pomiędzy warstwą krzemu a metalowym okablowaniem, podczas gdy małe kwadraty to połączenia pomiędzy warstwą krzemu a polikrzemem.



# Historia procesora 8086



- Droga do 8086 nie była tak bezpośrednia i zaplanowana, jak można by się spodziewać.

Jego poprzednikiem był Datapoint 2200, komputer stacjonarny/terminal z 1970 roku.

Datapoint 2200 był jeszcze przed erą mikroprocesorów; wykorzystywał 8-bitowy procesor zbudowany z płyty pełnej pojedynczych układów scalonych TTL (Transistor-transistor logic).

# Historia procesora 8086



- Firma Datapoint zapytał Intel i Texas Instruments, czy byłoby możliwe zastąpienie tej płyty chipów pojedynczym chipem.
- Kopiując architekturę Datapoint 2200, firma Texas Instruments stworzyła procesor TMX 1795 (1971), a Intel stworzył procesor 8008 (1972).
- Jednak Datapoint odrzucił te procesory, co było fatalną decyzją.
- Texas Instruments nie mogła znaleźć klienta na procesor TMX 1795 i porzuciła go, Intel zdecydował się sprzedać 8008 jako produkt, tworząc rynek mikroprocesorów.
- Intel rozwijał 8008 do procesorów 8080 (1974) i 8085 (1976).

# Historia procesora 8086



- W 1975 roku kolejnym wielkim planem Intela był procesor 8800 zaprojektowany jako główna architektura Intela na lata 80-te.
- Ten procesor został nazwany „micromainframe” ze względu na planowaną wysoką wydajność.
- Miał zupełnie nowy zestaw instrukcji zaprojektowany dla języków wysokiego poziomu, takich jak Ada, oraz obsługiwał programowanie obiektowe i garbage collection (usuwanie śmieci) na poziomie sprzętowym.
- Niestety, ten chip był zbyt ambitny i drastycznie spóźniał się z harmonogramem.
- Ostatecznie został wprowadzony na rynek w 1981 roku (jako iAPX 432) z kiepską wydajnością i okazał się komercyjną porażką.

# Historia  procesora 8086



- Ponieważ projekt iAPX 432 był opóźniony, Intel zdecydował w 1976 roku, że potrzebuje prostego procesora typu stop-gap do sprzedaży, dopóki iAPX 432 nie będzie gotowy.
- w 1978 roku Intel szybko zaprojektował i zrealizową 8086 jako 16-bitowy procesor, w pewnym stopniu kompatybilny z 8-bitowym 8080.
- 8086 spowodował wielki przełom (wręcz rewolucję) wraz z wprowadzeniem na rynek komputera osobistego IBM (PC) w 1981 roku.
- W 1983 roku IBM PC był najlepiej sprzedającym się komputerem i stał się standardem dla komputerów osobistych.
- Procesor w IBM PC to 8088, wariant 8086 z 8-bitową magistralą. Sukces IBM PC sprawił, że architektura 8086 stała się standardem, który nadal obowiązuje, 44 lata później.

IBM PC 5150 z monochromatycznym [monitorem](#) 5151




- IBM XT - 4,77 MHz, 128 KB RAM i 10 MB HDD. W

- na 16-bitowym mikroprocesorze Intel 80286, 6 MHz, później 8 MHz, 16-bitowa szyna ISA.


- Komputery kompatybilne (tzw. klony) z tym modelem często zawierały inne procesory i pracowały przy innych częstotliwościach zegara taktującego. Nosiły one nazwy takie jak PC/AT 386 lub PC/AT 286 12 MHz.

# Historia  procesora 8086

- Dlaczego IBM PC wybrał procesor Intel 8088?
- Według dr Davida Bradleya, jednego z pierwszych inżynierów IBM PC, kluczowym czynnikiem była znajomość (przez zespół IBM) oprogramowania Intela dla jego procesorów.
- Użyli procesora Intel 8085 we wcześniejszym komputerze stacjonarnym IBM Datamaster.
- Inny inżynier, Lewis Eggebrecht, powiedział, że Motorola 68000 jest godnym konkurentem, ale jej 16-bitowa magistrala danych znacznie podniosłaby koszty (jak w przypadku 8086). Podkreślił także lepsze wsparcie i narzędzia programistyczne firmy Intel.

# Historia  procesora 8086



- W każdym razie decyzja o zastosowaniu procesora 8088 ugruntowała sukces rodziny x86.
- IBM PC AT (1984) został zaktualizowany do kompatybilnego, ale mocniejszego procesora 80286.
- W 1985 r. linia x86 przeszła na 32-bitową wersję 80386, a następnie 64-bitową w 2003 r. z architekturą Opteron firmy AMD.
- Architektura x86 jest wciąż rozszerzana o funkcje, takie jak operacje wektorowe AVX-512 (2016).
- Ale mimo wszystkich tych zmian architektura x86 zachowuje zgodność z oryginalnym 8086.

# Podsumowanie

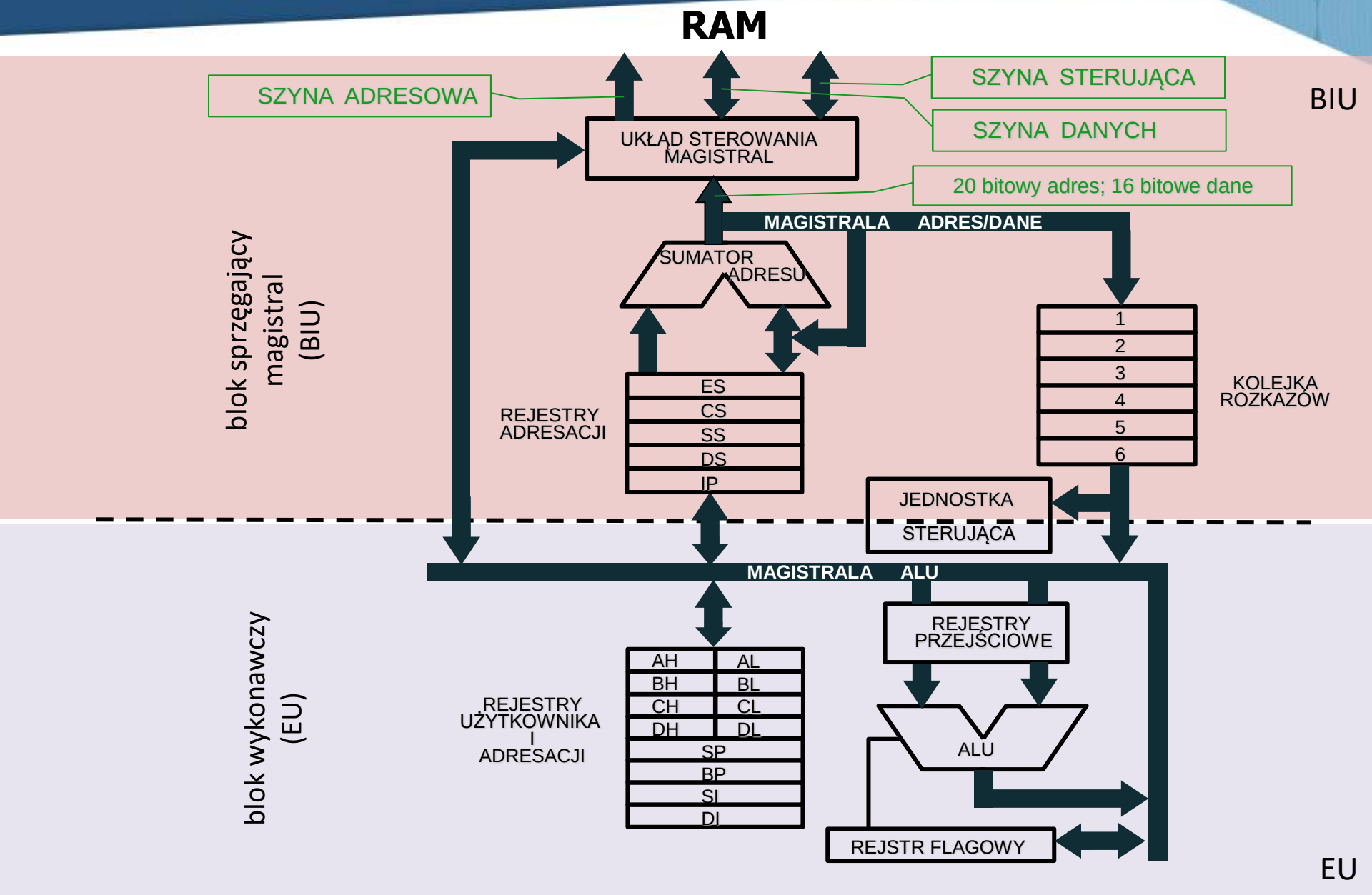


- procesor 8086 był pomyślany jako tymczasowy procesor typu „stop-gap”, dopóki Intel nie wypuścił swojego flagowego układu iAPX 432, i był potomkiem procesora zbudowanego z płyty pełnej układów TTL
- Jednak od tych skromnych początków architektura 8086 (x86) niespodziewanie zdominowała komputery stacjonarne i serwerowe aż do chwili obecnej.

- Intel 8086 jest oparty na CISC
- jest jeszcze RISC

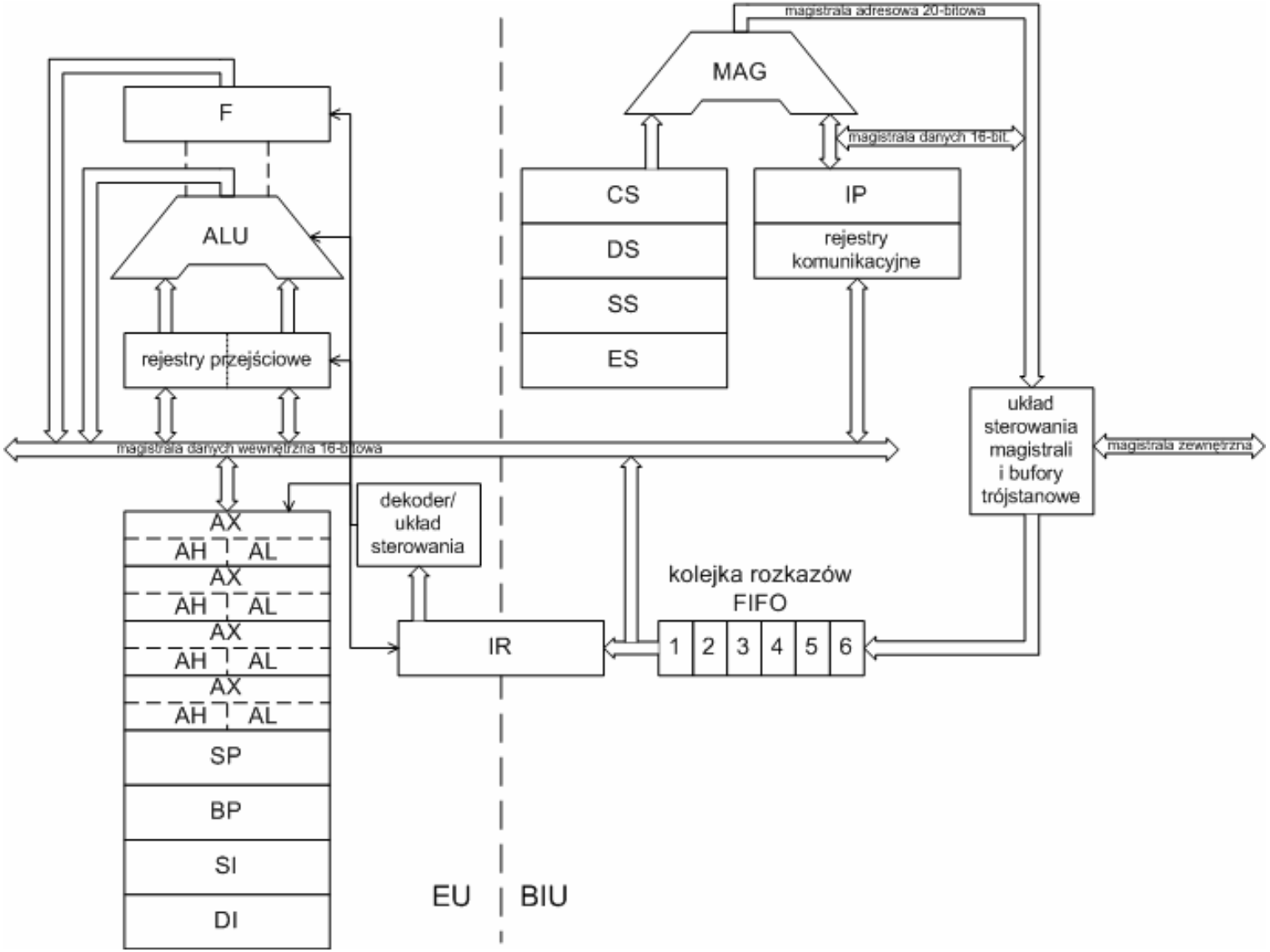
# SCHEMAT BLOKOWY PROCESORA 8086





# SCHEMAT BLOKOWY PROCESORA 8086





# REJESTRY OGÓLNEGO PRZEZNACZENIA



- Rejestry ogólnego przeznaczenia - rejestry przeznaczone do przechowywania dowolnych danych i wykonywania operacji (arytmetycznych i logicznych), ale jednocześnie spełniające pewne funkcje specjalne.
- Rejestry te podzielone są na dwie 8-bitowe części: dolną (L) i górną (H). Działając na danych 1-bajtowych każdy część może być wykorzystana niezależnie.
  - AX - (akumulator) rejestr przeznaczony do przechowywania jednego z operandów (argumentów) wykonywanej operacji oraz wyniku wykonywanej operacji
  - (tylko czasami wynik może być umieszczany w innym rejestrze)

# REJESTRY OGÓLNEGO PRZEZNACZENIA



- Rejestry ogólnego przeznaczenia – cd.
  - BX - rejestr bazowy wykorzystywany jest do wskazywania położenia i lokalizacji pamięci lub do adresowania położenia w pamięci. Domyślnie rejestr BX, wraz z rejestrem segmentowym DS, jest używany jako wskaźnik pamięci.
  - CX - rejestr wykorzystywany jest głównie jako licznik odliczający powtarzające się fragmenty programów bądź pojedynczych rozkazów.
  - DX - rejestr wykorzystywany głównie jako wskaźnik adresów w rozkazach IN i OUT (rozkazy wejścia i wyjścia). Adresowanie portów odbywać się może tylko poprzez użycie rejestru DX.

# REJESTRY OGÓLNEGO PRZEZNACZENIA



- Rejestr CS wskazuje na początek 64KB bloku pamięci lub na segment kodu, w którym rezyduje następny do wykonania rozkaz.
- Dokładne położenie tego rozkazu w segmencie wskazywane jest przez offset, którego wartość zawiera rejestr IP.
- Pełny adres to CS:IP. Rejestr CS może być zmieniany przez niektóre rozkazy ale nie można go ładować bezpośrednio.
- Rejestr DS wskazuje na początek segmentu danych czyli 64KB blok pamięci zawierający argumenty.
- Zazwyczaj rejestrami stowarzyszonymi z DS, określającymi offset w tym segmencie są rejestry BX, SI lub DI.

# REJESTRY OGÓLNEGO PRZEZNACZENIA



- Rejestr ES wskazuje na początek 64KB bloku pamięci zwanego dodatkowym.
- Segment ten nie jest przypisany do pojedynczych zastosowań - stosowany jest do różnych pojawiających się potrzeb.
- Często używany jest w operacjach łańcuchowych (wykorzystują pary rejestrów ES:DI) lub w operacjach na blokach np. kopiowanie, porównywanie, przeszukiwanie, czyszczenie.
- Rejestr SS wskazuje na początek 64KB bloku pamięci zwanego segmentem stosu.
- Wszystkie rozkazy na stosie - odkładania danych na stos, zdejmowania ze stosu, wywołania i powroty używają segmentu stosu (w tych rozkazach rejestr SP jest zdolny tylko do adresowania w obszarze stosu).
- Z rejestrem SS skojarzony jest rejestr BP do adresowania parametrów i zmiennych zawartych w stosie.

# REJESTRY OGÓLNEGO PRZEZNACZENIA



- SI - rejestr indeksowy źródła wykorzystywany jest (podobnie jak BX) głównie jako wskaźnik pamięci.
- DI - rejestr indeksowy przeznaczenia wykorzystywany jest (podobnie jak SI) głównie jako wskaźnik pamięci.
- Istnieje tu pewna analogia między rejestrem bazowym BX a rejestrami indeksowymi SI i DI. Oba te rejestry mogą być użyte jako wskaźniki pamięci podczas przetwarzania łańcuchów:
  - rejestr DI adresuje przesunięcie przeznaczenia danych w połączeniu z rejestrem segmentowym ES
  - rejestr SI adresuje przesunięcie źródła (wyniku) danych w połączeniu z rejestrem segmentowym DS.

# Rejestry wskaźnikowe i indeksowe

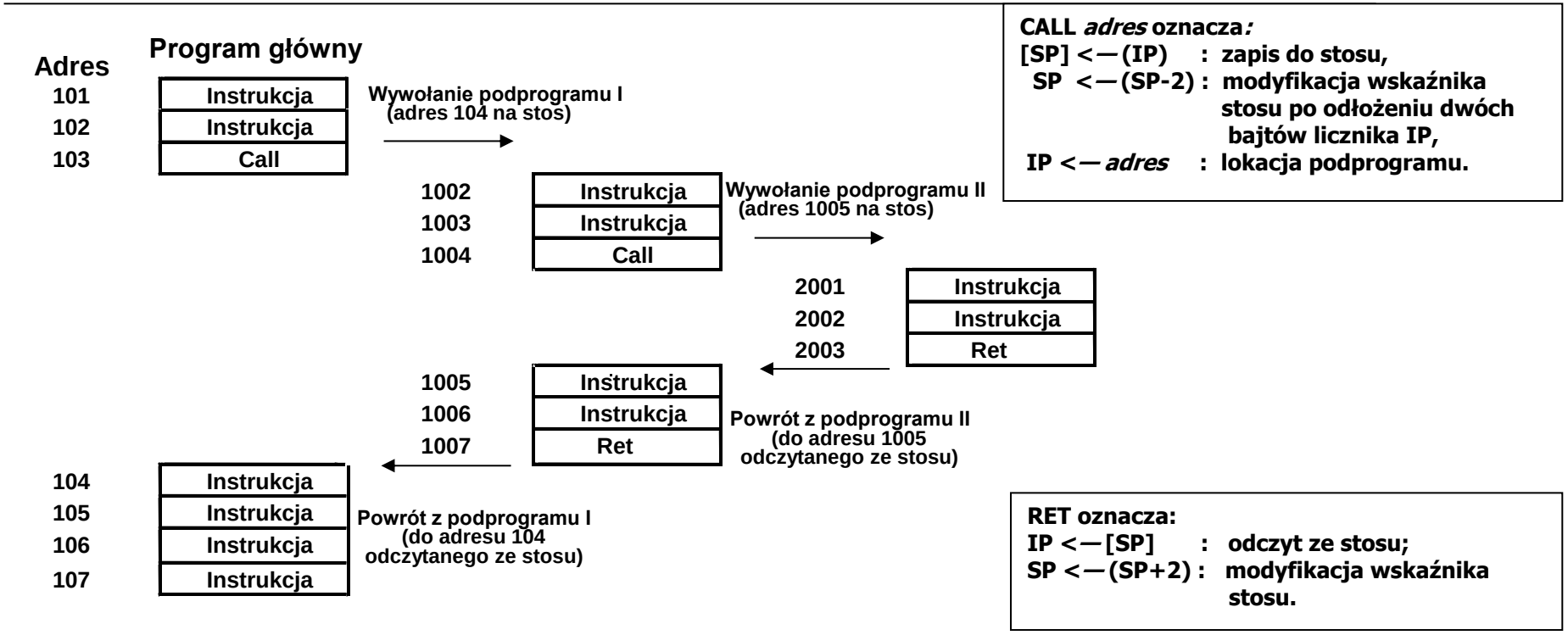


- Podczas przetwarzania danych nie łańcuchowych oba rejestry używane są jako wskaźniki pamięci zawsze względem rejestru segmentowego DS.
- BP (wskaźnik bazy) - rejestr ten używany jest jako wskaźnik pamięci, podobnie jak BX, SI, DI, z tą jednak różnicą, że dotyczy obszaru segmentu stosu i dlatego rejestr ten wykorzystywany jest wspólnie z rejestrem segmentowym SS.
- SP (wskaźnik stosu) - rejestr ten podaje bieżące położenie wierzchołka stosu.

# Stos



- Stosem nazywamy wyróżniony obszar w pamięci używany wg reguł:
  - informacje zapisane są na stos do kolejnych komórek (pod kolejnymi adresami), przy czym żadnego adresu nie wolno pominąć
  - odczytujemy informacje w kolejności odwrotnej do ich zapisu
  - informacje odczytujemy z ostatnio zapełnionej komórki, natomiast zapisujemy do pierwszej wolnej
- Czyli obowiązuje reguła LIFO - ostatni wchodzi pierwszy wychodzi



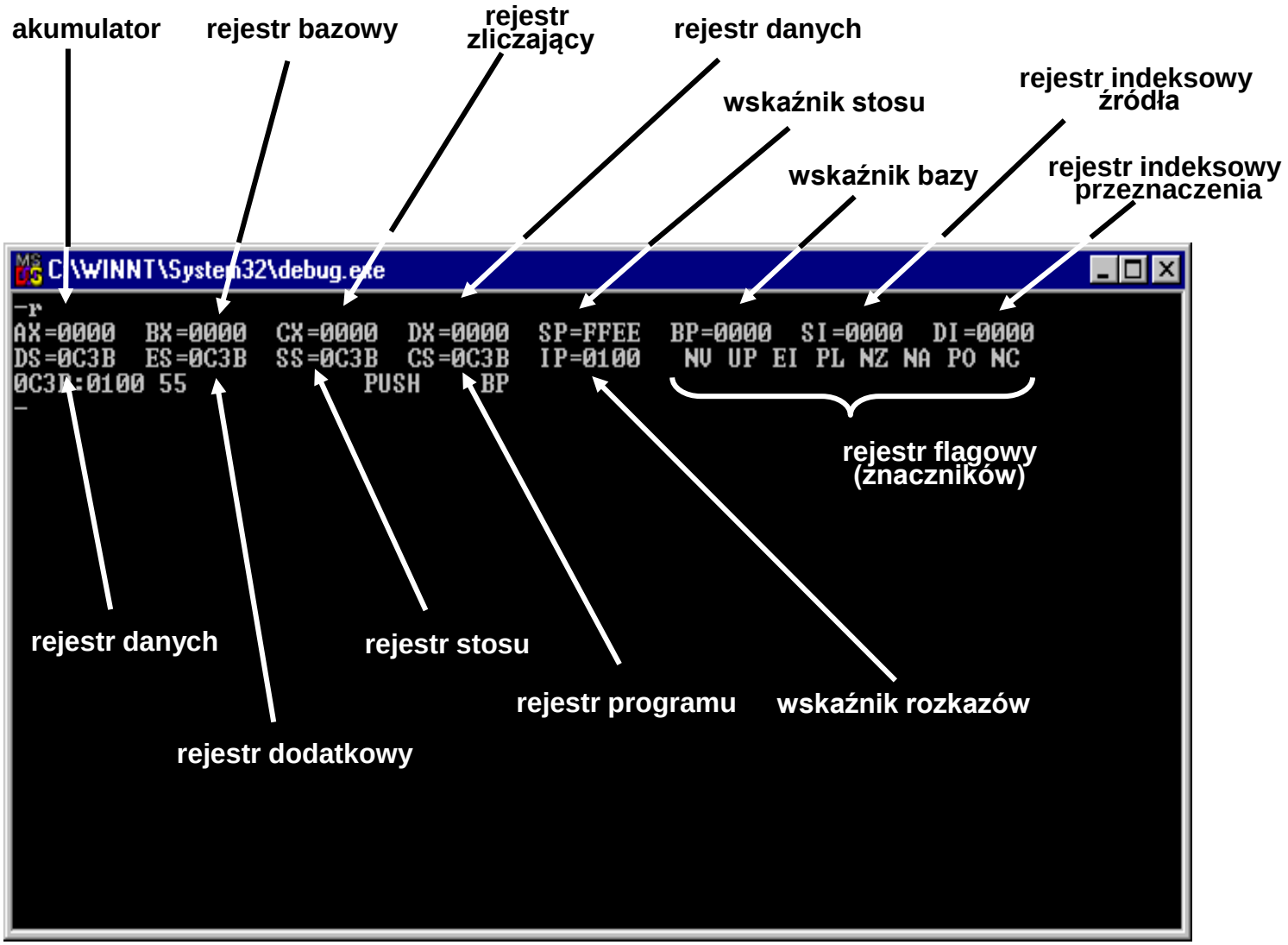
# Rejestr IP i rejestr FLAGOWY



- IP (wskaźnik rozkazów)- rejestr ten zawiera zawsze offset pamięci, w którym zawarty jest następny rozkaz do wykonania. Bazowy adres segmentu kodu zawarty jest w rejestrze CS. Stąd pełny adres logiczny wykonywanego rozkazu wskazywany jest parą rejestrów CS:IP.
- Oznacza to, że jeśli wykonywany jest rozkaz to wskaźnik rozkazów ustawiany jest do następnego adresu pamięci, pod którym znajduje się rozkaz do wykonania. Wyjątek stanowią rozkazy wywołania i skoku.
- FLAGS (rejestr znaczników, rejestr flagowy)- rejestr ten jest zbiorem poszczególnych bitów kontrolnych (znaczników), które wskazują wystąpienie określonego stanu procesora.

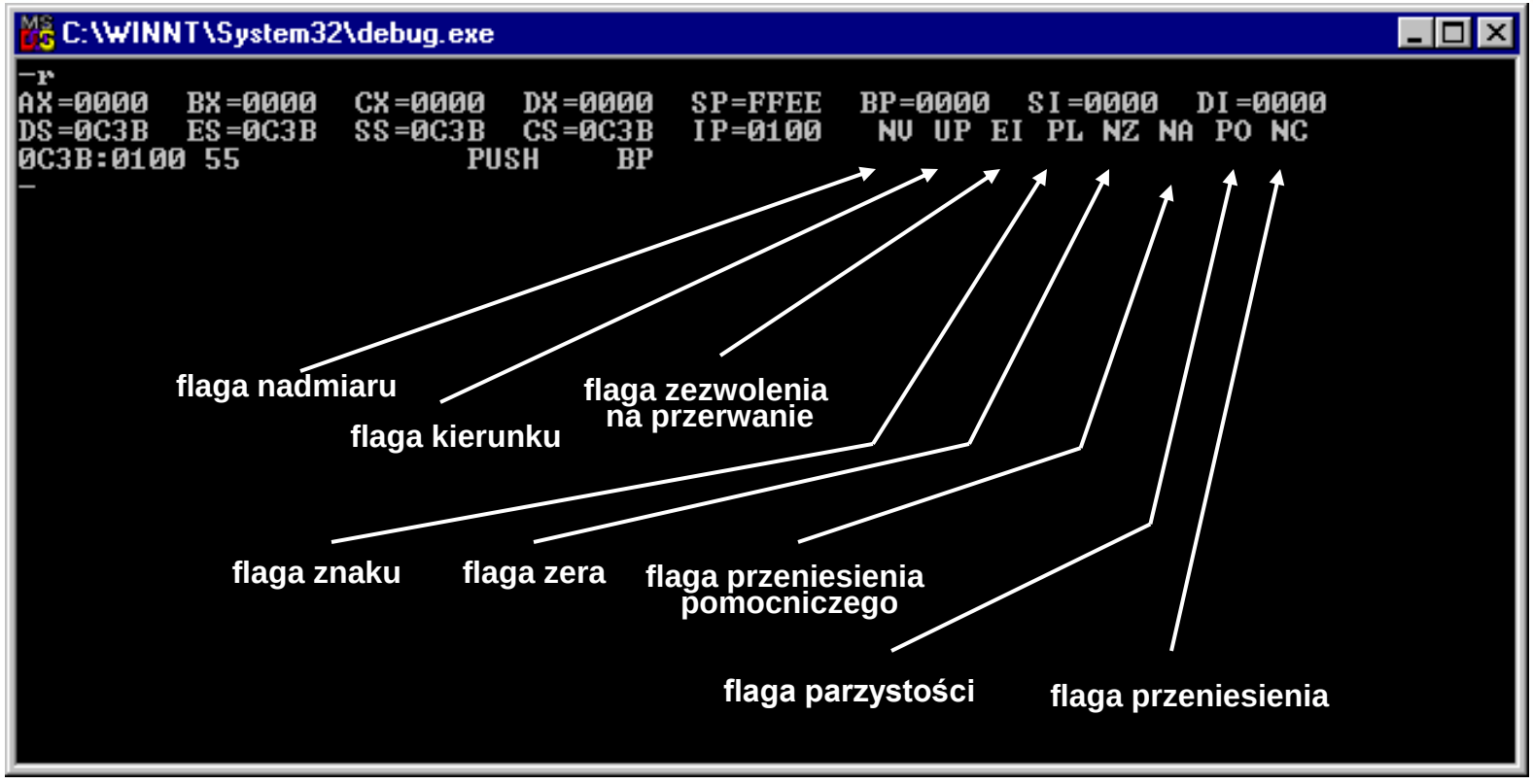
| Znaczniki stanu | =1 | =0 |
|---|---|---|
| OF - flaga nadmiaru (przepełnienia) | OV (over) | NV (not over) |
| SF - flaga znaku | NG (negative) | PL (plus) |
| ZF - flaga zera | ZR (zero) | NZ (not zero) |
| CF - flaga przeniesienia | CY (arry yes) | NC (no carry) |
| PF - flaga parzystości | PE (parity even) | PO (parity odd) |
| AF - flaga przeniesienia pomocniczego | AC (aux. carry) | NA (not aux. carry) |
| Znaczniki kontrolne | | |
| IF - flaga zezwolenia na przerwanie | EI (enables int) | DI (disables int) |
| DF - flaga kierunku | DN (down) | UP (up) |
| TF - flaga pracy krokowej | | |

# DEBUG


akumulator
rejestr bazowy
rejestr zliczający
rejestr danych
wskaźnik stosu
rejestr indeksowy źródła
wskaźnik bazy
rejestr indeksowy przeznaczenia
C:\WINNT\System32\debug.exe
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0C3B ES=0C3B SS=0C3B CS=0C3B IP=0100 NV UP EI PL NZ NA PO NC
0C3B:0100 55 PUSH BP
-
rejestr flagowy (znaczników)
rejestr danych
rejestr stosu
rejestr programu
wskaźnik rozkazów
rejestr dodatkowy

# DEBUG



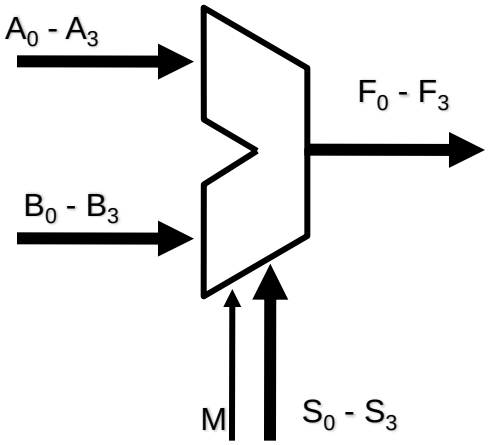



| Flaga (znacznik) | symbol | =1 | =0 |
|---|---|---|---|
| Flaga nadmiaru | OF | OV | NV |
| Flaga kierunku | DF | DN | UP |
| Flaga zezwolenia na przerwanie | IF | EI | DI |
| Flaga znaku | SF | NG | PL |
| Flaga zera | ZF | ZR | NZ |
| Flaga przeniesienia pomocniczego | AF | AC | NA |
| Flaga parzystości | PF | PE | PO |
| Flaga przeniesienia | CF | CY | NC |

# BLOK ARYTMETYCZNO-LOGICZNY

- Blok arytmetyczno-logiczny (ALU) - jest uniwersalnym układem kombinacyjnym, który realizuje operacje matematyczne i logiczne w zależności od zaprogramowanej operacji tj. rozkazu umieszczonego w programie.
- Rozkazy mogą dotyczyć
  - operacji dwuargumentowych: operacji arytmetycznych (dodawanie i odejmowanie) i operacji logicznych (sumowanie mnożenie, sumowanie mod 2, itp.)
  - operacji jednoargumentowych (np. negowanie bitów lub przesuwanie zawartości rejestrów)
- Argumentami rozkazów są najczęściej dwa słowa binarne, od długości których mówi się o liczbie bitów ALU.

Przykład 4 bitowego ALU SN74181




Sygnały sterujące do wyboru mikrooperacji logicznej (M=0) lub arytmetycznej (M=1)

| Lp. | $S_0$ - $S_3$ | M=0 | M=1 |
|---|---|---|---|
| 1 | 0000 | $F=\overline{A}$ | $F=A+C_0$ |
| 2 | 1000 | $F=\overline{A\vee B}$ | $F=(A\vee B)+C_0$ |
| 3 | 0100 | $F=\overline{A}B$ | $F=(A\vee \overline{B})+C_0$ |
| 4 | 1100 | $F=0$ | $F=C_0-1$ |
| 5 | 0010 | $F=\overline{AB}$ | $F=A+A\overline{B}+C_0$ |
| 6 | 1010 | $F=B$ | $F=(A\vee B)+A\overline{B}+C_0$ |
| 7 | 0110 | $F=A\oplus B$ | $F=A-B-(1-C_0)$ |
| 8 | 1110 | $F=A\overline{B}$ | $F=A-(1-C_0)$ |
| 9 | 0001 | $F=A\vee \overline{B}$ | $F=A+AB+C_0$ |
| 10 | 1001 | $F=\overline{A\oplus B}$ | $F=A+B+C_0$ |
| 11 | 0101 | $F=B$ | $F=(A\vee \overline{B})+AB+C_0$ |
| 12 | 1101 | $F=AB$ | $F=AB-(1-C_0)$ |
| 13 | 0011 | $F=1$ | $F=2A+C_0$ |
| 14 | 1011 | $F=A\vee \overline{B}$ | $F=(AB)+A+C_0$ |
| 15 | 0111 | $F=A\vee B$ | $F=(A\vee \overline{B})+A+C_0$ |
| 16 | 1111 | $F=A$ | $F=A-(1-C_0)$ |

# UKŁAD STEROWANIA

- Często (a może najczęściej) procesor wykonuje rozkazy nie w jednym kroku (jak np. dodawanie) ale w wielu krokach (np. mnożenie lub dzielenie jako ciąg dodawań i przesunięć).
- W tym celu potrzebny jest złożony automat sekwencyjny, generujący odpowiednie ciągi słów podawanych na wejścia sterujące układu ALU - układ sterowania.
- Cechy takiego automatu to:
  - konieczność posiadania bardzo dużej liczby stanów dostosowanych do wymaganej liczby wykonywanych rozkazów;
  - konieczność zapewnienia synchronizacji pracy układu sterowania i wykonawczego (uwzględnienie czasów wykonywania poszczególnych operacji).

- W praktyce realizowane są jako:
  - generatory sekwencyjne
  - układy mikroprogramowalne

# Dziękuję za uwagę!

Slajdy na podstawie wykładów prof. Stanisława Ambroszkewicza