Architektura systemów komputerowych

- Architektura systemu komputerowego - sposób kompozycji systemu z bloków funkcjonalnych.
- System komputerowy - urządzenie do wykonywania obliczeń zgodnie z pewnym modelem.
- Model obliczeń - sekwencyjny von Neumana, obliczenie jest przedstawiane w formie algorytmu.
- Algorytm - skończona sekwencja kroków obliczeniowych - kompletny, jednoznaczny, masowy.

Algorytm

Przykład: algorytm sumowania liczb w zapisie U2
U2 jest zapisem binarnym, wagowym i systematycznym (bo wagi rosną liniowo).
propagacja (przeniesienie) - długotrwała
Reprezentacja liczby I:

$$l = -b_0 2^m + \sum b_i 2^{mi}$$

gdzie $b_i \in \{0,1\}$
"Najstarsza" pozycja ma wagę ujemną !!

Algorytm sumowania liczb A i B w U2 - zapis w języku naturalnym wzbogaconym o "kawałek" C:

- przeczytaj: m, A, B;
- p:=0;
- dla i := m aż do i = 0 wykonaj
- { C(i) := A(i) + B(i) + p;
- jeżeli C(i) < 2 to p := 0
- w przeciwnym wypadku
- {C(i) := C(i) - 2;
- p := 1}
- }
- jeżeli A(0) = B(0) oraz A(0) $\neq$ C(0)
- to sygnalizuj nadmiar
- Koniec

m = 6

A 0 0 1 1 0 1 1
B 0 0 1 0 1 0 1
  ----
  0 1 1 0 0 0 0

to przykład na **sekwencyjność**

Model

Pamięć przechowuje algorytm zapisany w języku programowania.
Język programowania (imperatywnego) składa się z **rozkazów (instrukcji)** dla procesora.
Procesor pobiera z pamięci rozkazy algorytmu i wykonuje je. Procesor “rozumie” rozkazy.

Pamięć

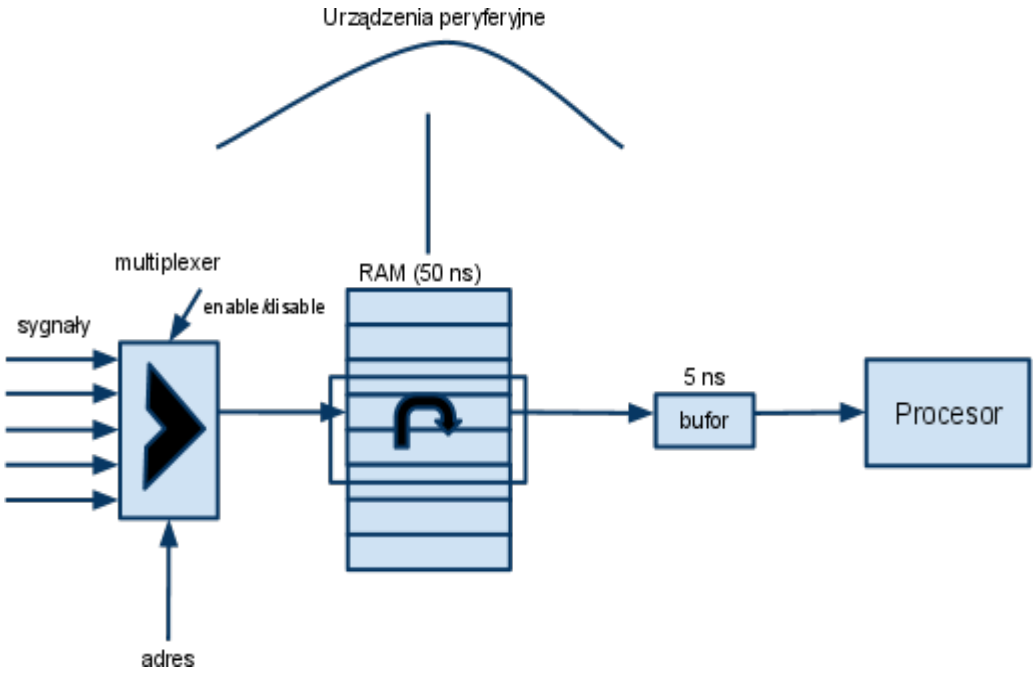
- operacyjna RAM (książka),
- zawiera adresowane komórki (numerowane strony),
- adres umożliwia dostęp do zawartości konkretnej komórki,
- jedna komórka zawiera słowo 0/1kowe,
- słowo może być zakodowanym rozkazem lub daną,
- pojedynczą komórkę można zapisać lub odczytać,
- zapis niszczy poprzednią zawartość komórki, odczyt nie,
- wyłączenie zasilania niszczy zawartość całej pamięci.

Algorytm procesora

- Pobierz zawartość komórki pamięci operacyjnej (RAM) o adresie wskazywanym przez rejestr PC (Program Counter),
- Potraktuj tą zawartość jako rozkaz do wykonania:
    - zdekoduj treść rozkazu,
    - wytwórz sygnały sterujące potrzebne do wykonania rozkazu,
    - dodaj 1 do zawartości PC.
- Wykonaj rozkaz,
- Powtórz od początku.

racjonalne zakończenie - rozkaz zawiera sygnał stop

ASK



- **Bloki** funkcjonalne → {klocki Lego}, {budynki, ulice, place, zieleń, …},
- Sposób poprawnej kompozycji (jak można łączyć bloki funkcjonalne - **składnia**) → {bolce i otwory}, {budynków nie można stawiać na ulicach, ulice powinny łączyć budynki, zieleń powinna przylegać do budynków, …},
- Znaczenie (co powstało w wyniku kompozycji - **semantyka**) → {helikopter}, {osiedle}, {dom}.

Bloki funkcjonalne:

1. Pamięci

a) podstawowe (układy scalone):
- rejestr - 1ns (najmniejsza, najszybsza pamięć, gdzie można zapisać słowo)
- blok rejestrów (połączone rejestry),
- stos (charakterystyczny rejestr),
- notatnikowa, podręczne - 5ns,
- operacyjna RAM - 50ns,
- sterowania ROM - 200ns (w pamięci ROM “zaszyte” jest sterowanie procesora - gdyby jej nie było, stracilibyśmy możliwość sterowania procesorem)

b) pomocnicze (magnetyczne i optyczne)

2. Wykonawcze

- ALU - układy arytmetyczno - logiczne, kombinacyjne,
- CPU - procesory uniwersalne (ALU + rejestry + …)
- Koprocesory - procesory wyspecjalizowane, dodatek do CPU,
- ASIP - procesory wyspecjalizowane, samodzielne,
- ASIC - wyspecjalizowane układy cyfrowe dla jednej aplikacji.

3. Komunikacyjne

- magistrale - łącza multiplekserowane w czasie (magistrala jest zawsze multiplekserowalna w czasie)
- multipleksery - przełączniki,
- DMA - Direct Memory Access,
- sterowniki we/wy.

4. Sterujące

- zegary - do taktowania operacji (synchronizacji pracy),
- dekodery / enkodery,
- układy arbitrażu,
- automaty sterujące.

Co robi architekt komputerowy?

Problem architekta komputerowego:
Jak optymalnie
(koszt vs wydajność vs niezawodność)
złożyć system z bloków funkcjonalnych?

Co ważniejsze, funkcja, czy struktura?

Blok funkcjonalny = funkcje + struktura, np. ALU
System Komputerowy = funkcje + struktura, np. PC

Funkcje: CO TO JEST SYSTEM KOMPUTEROWY?

UŻYTKOWNICY

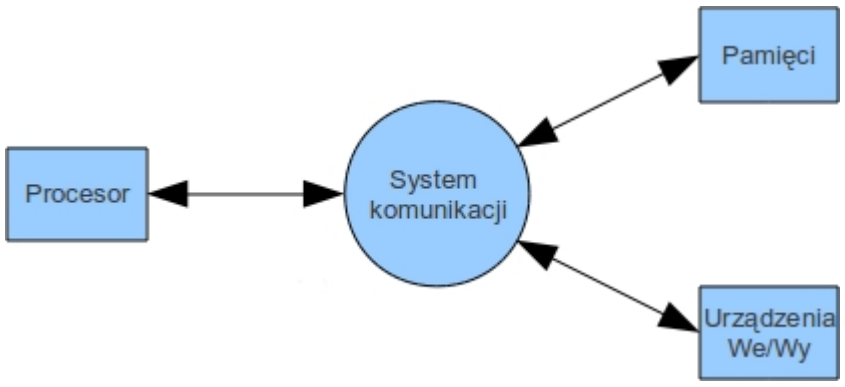
IT(SOA)

JĘZYKI WYSOKIEGO POZIOMU
SYSTEM OPERACYJNY
JĘZYK WEWNĘTRZNY (LISTA ROZKAZÓW)
MIKROKOD
ZESPOŁY FUNKCJONALNE (ALU, Rejestry, Magistrale)
PODSTAWOWE ELEMENTY LOGICZNE (np. bramki logiczne)
ELEMENTY ELEKTRYCZNE (np. rezystory)
MASKI DO PRODUKCJI

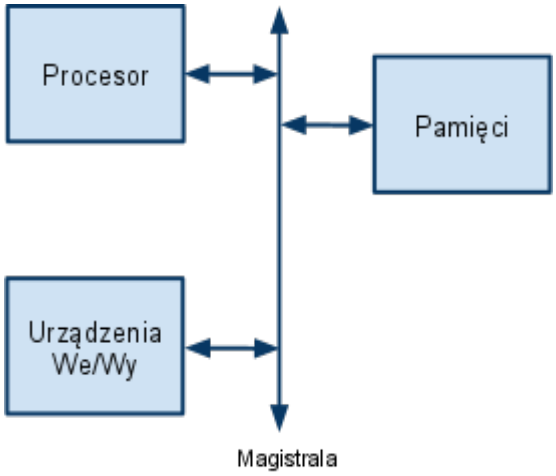
Język zewnętrzny zbudowany jest z mikrokodu, który buduje się na podstawie zespołów funkcjonalnych, je natomiast z podstawowych elementów logicznych, które są budowane z elementów elektrycznych. Maski do produkcji odpowiadają za technologie wytwarzania elementów elektrycznych.

STRUKTURA: Co to jest system komputerowy?

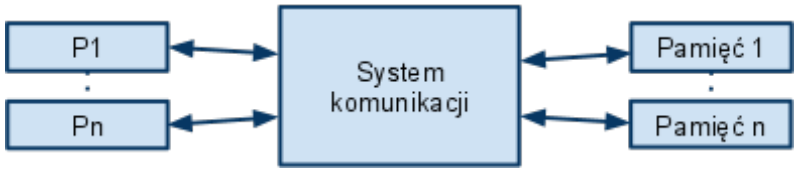
1. Jednoprocesorowy



- architektura magistrali

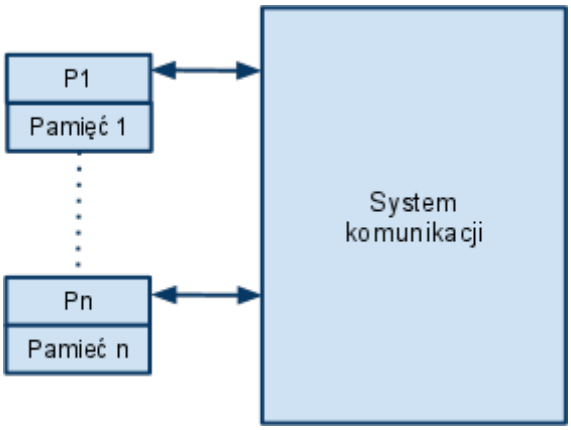


2. Wieloprocesorowy



Np. Magistrala + dzielona pamięć
dzielona pamięć (shared memory) - jest to określony moduł pamięci. służy do komunikacji między procesorami.

3. Sieciowy



Np. Magistrala + wymiana komunikatów

Podsumowanie

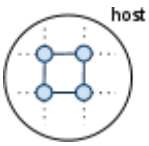
System cyfrowy:

- funkcjonalnie struktura warstwowa (im wyższa warstwa tym funkcje bardziej skomplikowane),
- strukturalnie kompozycja (architektura) modułów o precyzyjnie zdefiniowanych funkcjach i zasadach wzajemnej komunikacji.

System działa według pewnego modelu obliczeniowego.

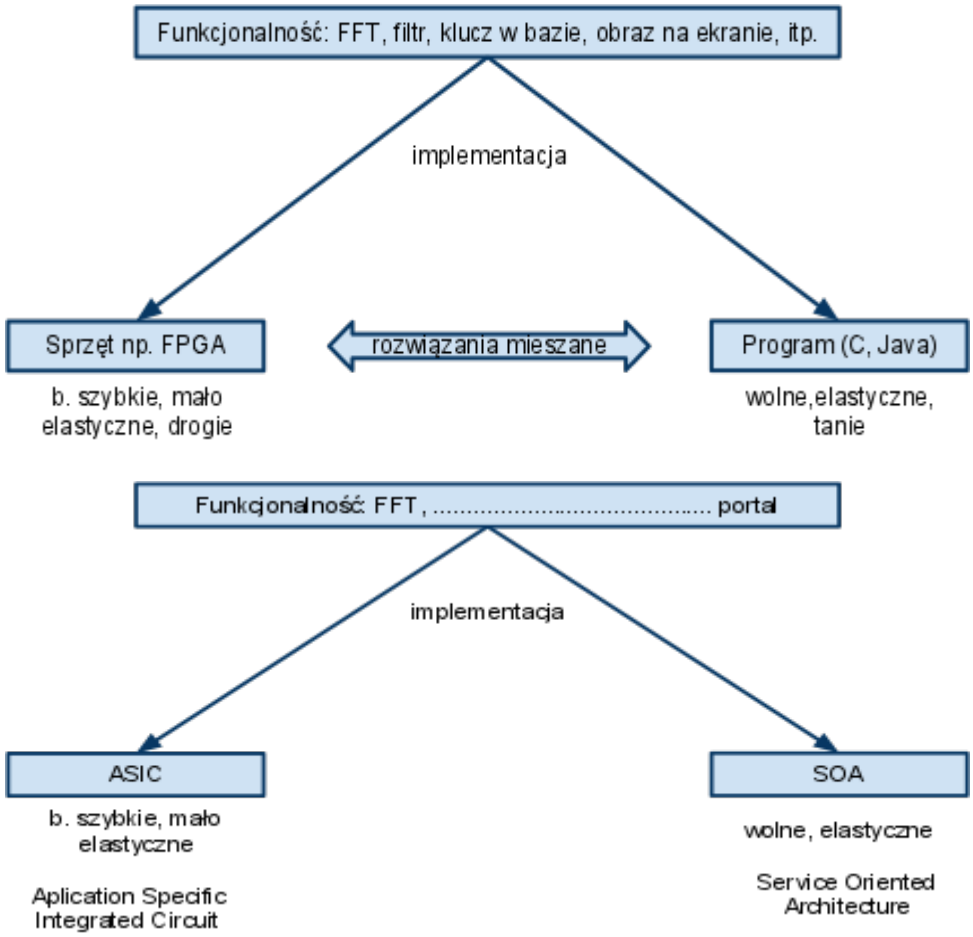
Modele:

- sekwencyjny (von Neumana),
- przepływowy (obliczenia "płyną" jak mogą (porównanie do wody) operacja jest wykonywana wtedy, gdy gotowe są jej argumenty),
- systoliczny (porównanie do serca, skurcz, rozkurcz, wszystkie procesory liczą, potem wszystkie wysyłają do sąsiadów, wszystkie liczą, wszystkie wysyłają itd. związany z tym modelem jest host, który tym zarządza, przykłady: do prognozowania pogody, modelowanie przyrody),



- obiektowy (do modelowania rzeczywistości, współpraca, wymiana usług - od usługi do usługi).

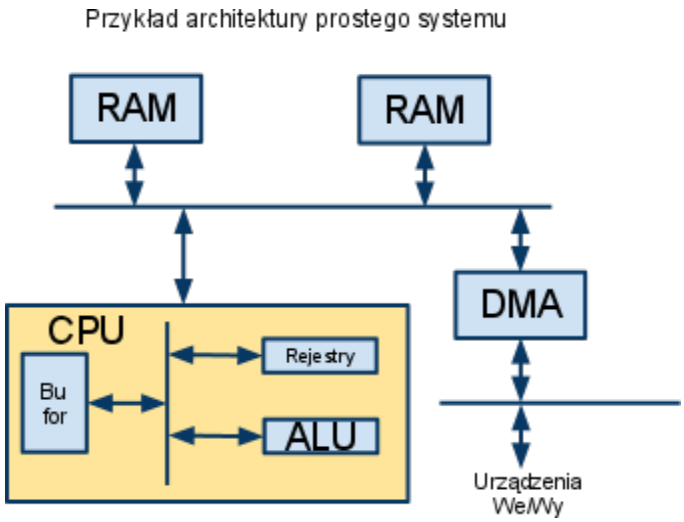
Dlaczego architektura systemów komputerowych jest ważna?



Architektura systemu: kompozycja bloków funkcjonalnych

Bloki funkcjonalne:
2x RAM,
2x magistrala
DMA
CPU: ALU
Rejestry
magistrala
bufor

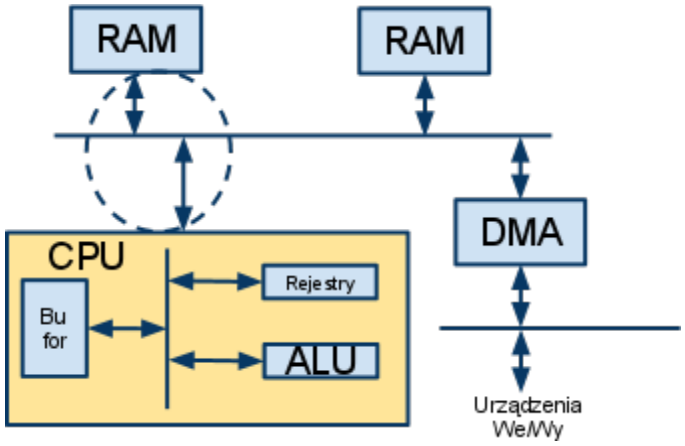
RTL (Register Transfer Level):
Rejestry,
Przesłania



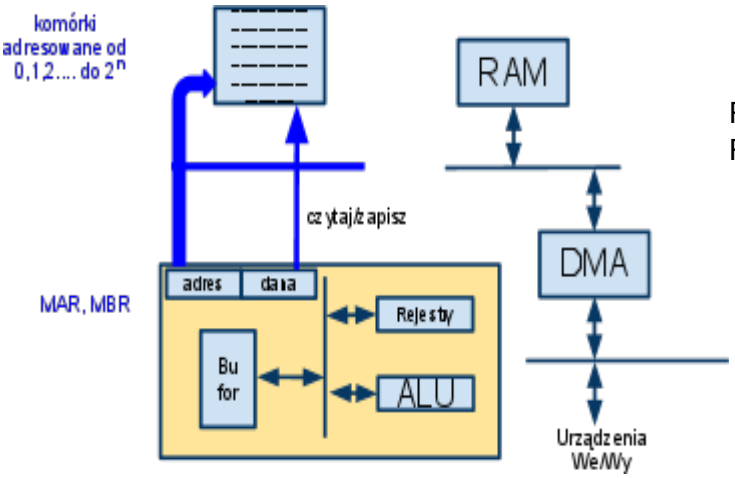
Jak CPU współpracuje z RAM:

Przykład architektury prostego systemu

Model von Neumana CPU wykonuje rozkazy pobierane z RAM



Model von Neumana CPU wykonuje rozkazy pobierane z rozkazy zajmują kolejne komórki



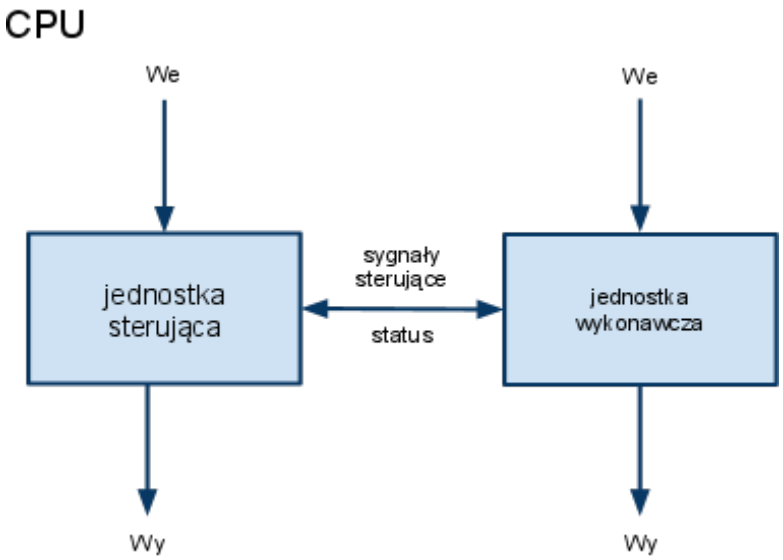
RAM,
RAM

Architektura CPU: podział funkcji

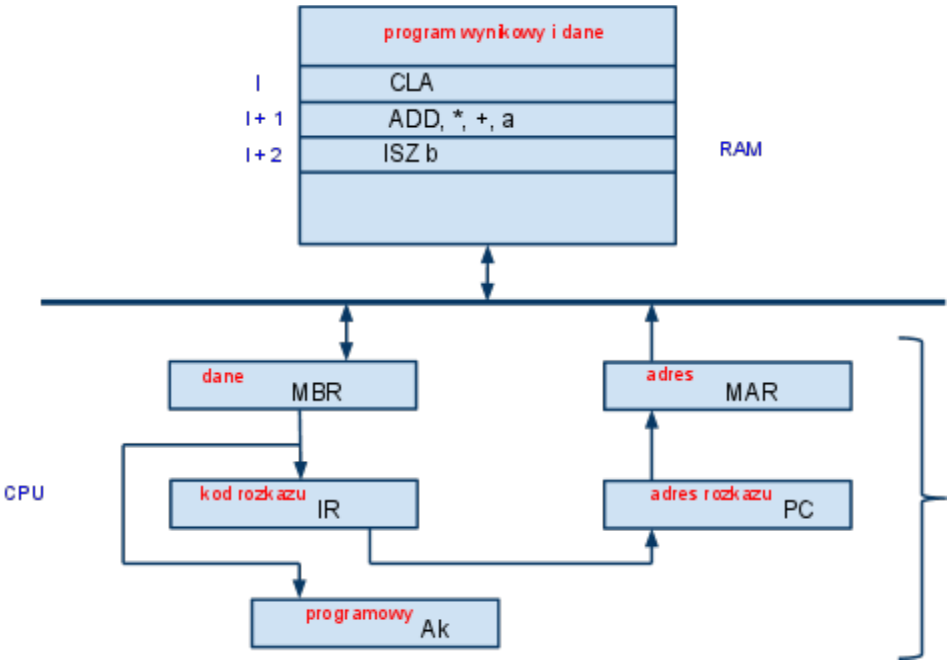
Jednostka wykonawcza CPU:
ALU,
Rejestry,
magistrala,
Bufor.

Jednostka sterująca CPU:
Automat(y)

Sygnały sterujące
Bity statusu



Translacja na język wewnętrzny komputera (kompilator)



Po translacji program źródłowy jest zapisywany w pamięci RAM komputera w postaci słów 01-kowych; dla czytelności słowom tym przypisujemy mnemoniki.

Ak - jest to rejestr dostępny dla kompilatora (może być takich wiele)

Rejestr MBR jest podzielony na dwie części:

- część rozkazową (przechowującą rozkaz)
- część adresową (w przypadku instrukcji adresowej przechowuje adres operandu)

Znaczenie mnemoników

- CLA - Ak <= 0
- ADD - Ak <= Ak + M (adres)
  - * - pośrednio
  - + - z autonastępnikowaniem
- ISZ - M <= M + 1, gdy M = 0 wtedy skip
- SKIP - PC <= PC + 1
- BRA - PC <= adres
- STO - M <= Ak (adres)

Jak to wygląda na różnych poziomach

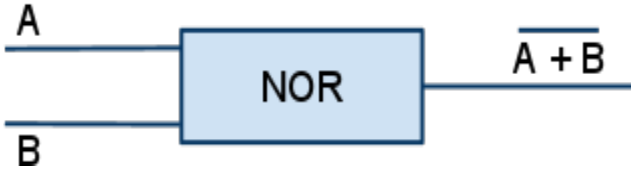
- Języki wysokiego poziomu:

```
for  i:=1 to n do
        S : = S + A[i];
```

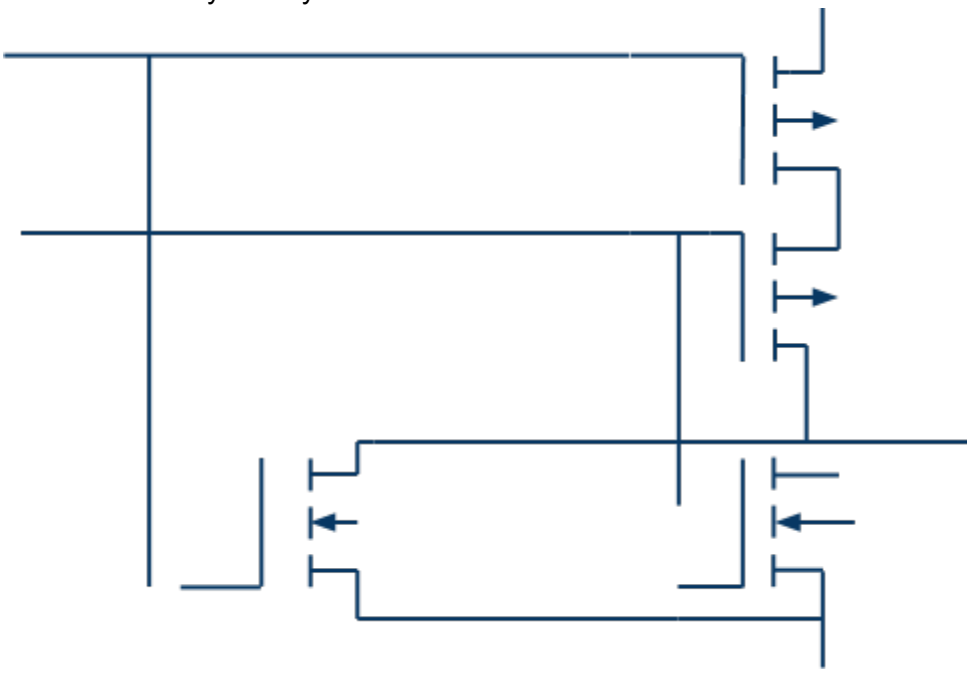
- Język wewnętrzny

```
I           CLA
I + 1       ADD, *, +, a
I + 2       ISZ b
I + 3       BRA I+1
I + 4       STO
```

- Podstawowe elementy logiczne



- Elementy elektryczne



Program wynikowy w języku wewnętrznym procesora

Kod wynikowy .EXE symbolicznie:

- ...
- a adres A[1]
- b -n
- c S
- …
- l CLA
- l+1 ADD, *, +, a
- l+2 ISZ b
- l+3 BRA l+1
- l+4 STO
- …
-

n - liczba elementów wektora
A[1] - adres wektora
CLA - przykład rozkazu bezadresowego

Zawartość RAM :

- 100 300 (Adres A[1])
- 101 -5
- 102 ???
- ….
- 200 CLA ← Program Counter
- 201 ADD, *, +, 100 ↓
- 202 ISZ 101 ↓
- 203 BRA 201 ↓
- 204 STO 102 ↓
- ….
- 300 25
- 301 -10
- 302 -5
- 303 20
- 304 12

Sterowanie CPU:

- **1. PC <= I (adres początku programu w RAM)**
- **2. MAR <= PC, RAM start odczyt**
- **PC <= PC + 1**
- **3. MBR <= M (odczyt RAM)**
- **4. IR <= część operacyjna MBR (bo cykl pobrania rozkazu)**
- **5. dekodowanie IR ("zrozumienie rozkazu i wysterowanie jednostki wykonawczej),**

wykonanie rozkazu

- **6. powtórzenie od 2.**

Wykonanie przykładowego programu

Program zaczyna się wykonywać od komórki pamieci o adresie 200 - rozkaz CLA , jest to wyzerowanie rejestru A, następnie zwieksza sie ProgramCounter i wykonany zostaje rozkaz z komórki 201... jest to rozkaz ADD z adresowaniem pośrednim (znak *) i autonastępnikowaniem (znak +), dodaje on do akumulatora zawartość komórki której adres wskazuje komórka o adresie 100 (czyli dodaje zawartość komórki 300), następnie wartość komórki 100 zostaje zwiększona o jeden czyli zawiera teraz adres 301-wszej komórki. PC zwiększa się, przeskakując do "ISZ 101", która zwiększa o jeden zawartość komórki o adresie 101 i jeśli jej zawartość będzie równa zero to przeskakuje nastepną komendę. W 203 jest skok do miejsca 201, co tworzy nam pętle. Kończy się ona po 5 przeskokach, wtedy 101 będzie miało wartość 0, a komedna "ISZ 101" przeskoczy nam komende "BRA 201" (która tak naprawdę tworzy naszą pętle), do komendy "STO 102" , która zapisze zawartość akumulatora (nasz zsumowany wektor od 300 - 304) do komórki 102.

Sterowanie wykonaniem rozkazów:

- **ISZ b**

5.1 MAR <= część adresowa MBR
RAM start odczyt
5.2 MBR <= M
5.3 MBR <= MBR + 1, Ram start zapis
5.4 jeżeli MBR = 0 to
PC <= PC + 1,

- **BRA l +**

5.1 PC <= część adresowa MBR

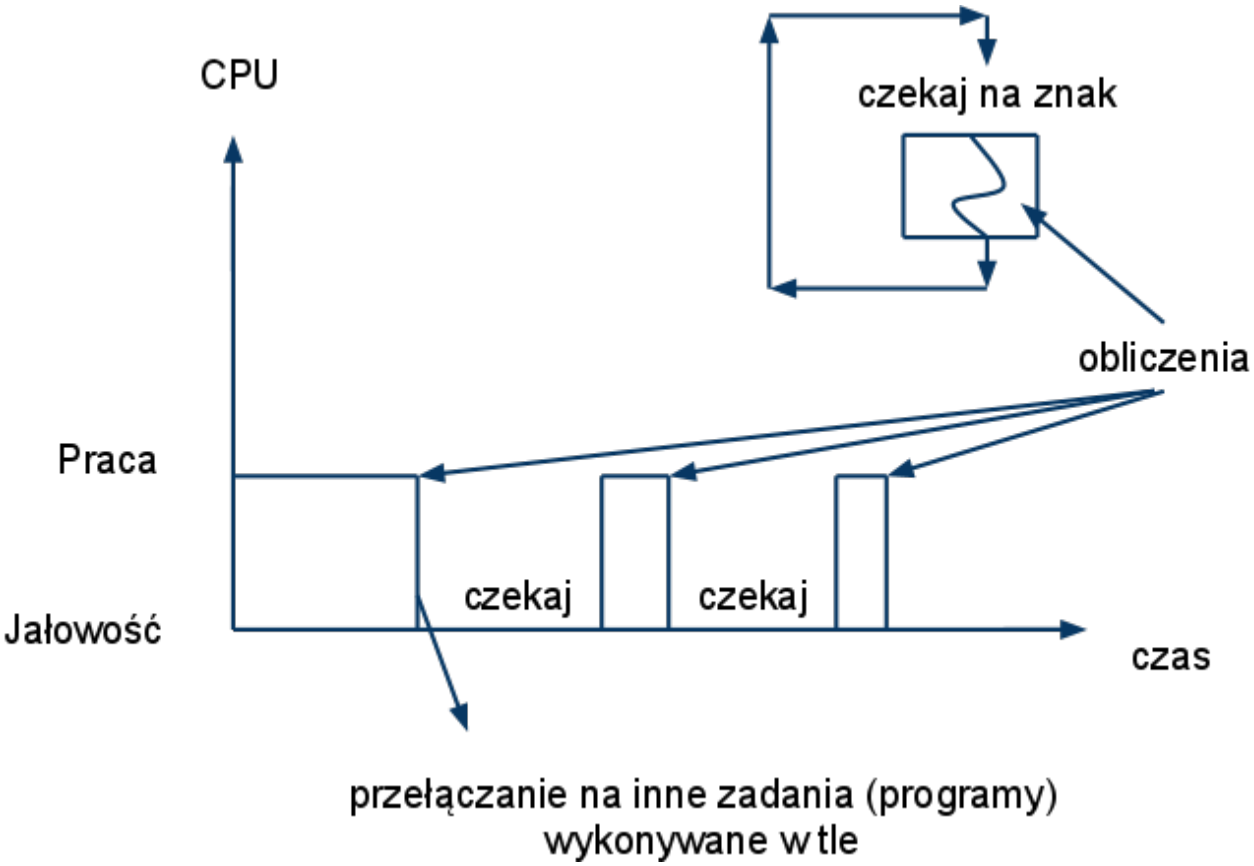
Co decyduje o wydajności procesora (CPU) ?

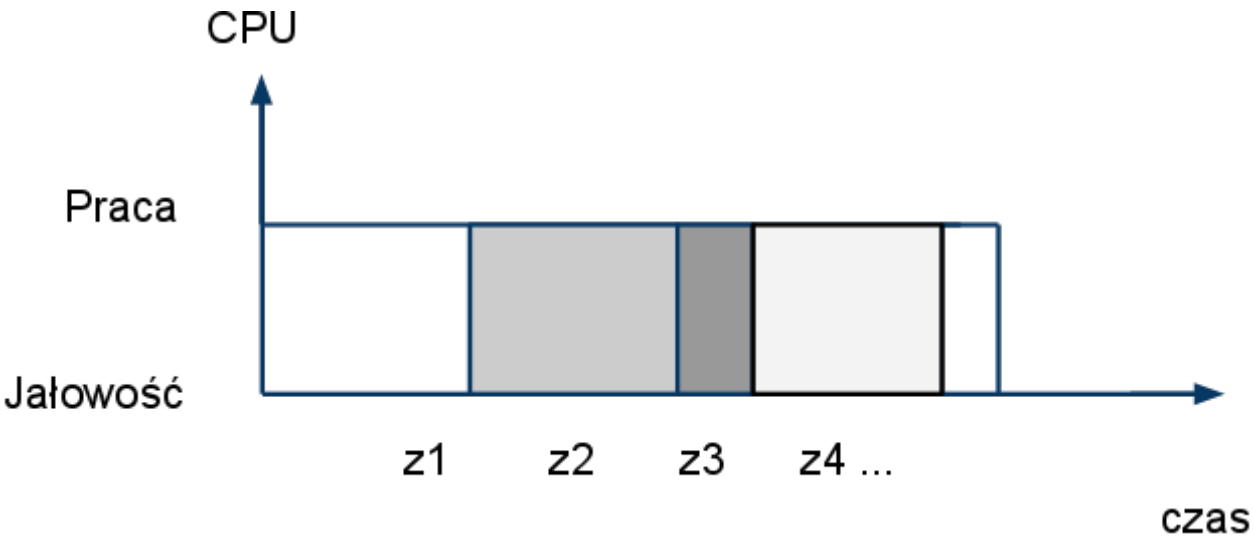
- częstotliwość zegara
- czas cyklu RAM
- moc listy rozkazów CPU
- sprawność kompilatora
- architektura !!!

Przypomnienie: Stosujemy klasyczny model obliczeniowy Von Neumana.

Przerwania - wielozadaniowość

Współpraca z klawiaturą





przerwanie:
- zapamiętaj stan przerwanego programu (procesu),
- pobierz do wykonania pierwsze zadanie z kolejki zadań oczekujących na wykonanie (funkcje systemu operacyjnego (wielozadaniowego)).

Przegląd mikrooperacji

- przesyłania informacji,
- arytmetyczne,
- logiczne,
- przesuwania informacji.

Mikrooperacja = operacja, którą można wykonać w czasie jednego cyklu zegara.

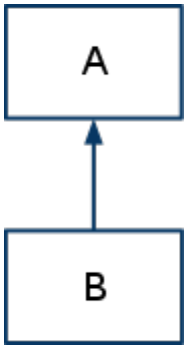
Przesyłanie informacji (pomiędzy rejestrami)

- równoległe: A <- B

F: A <- B, C <-D

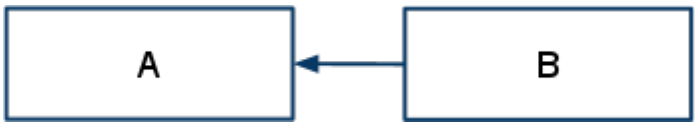
warunek

funkcja sterująca



przesyłana jest cała komórka

- szeregowe: A_i <- B_i i = 1, …, n



przesyłanie bit po bicie

- za pomocą magistrali:

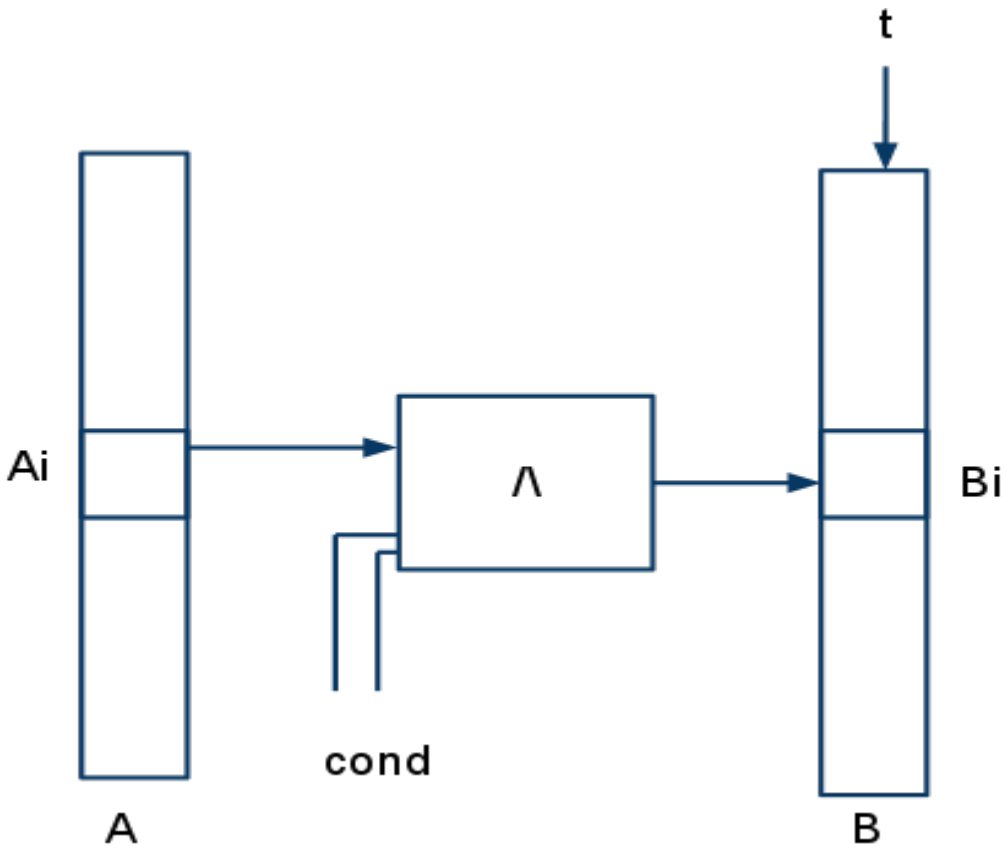
BUS <- A

B <- BUS

- z i do PAO:

MAR, MBR, cykl odczyt - zapis

Przesyłanie równoległe - implementacja



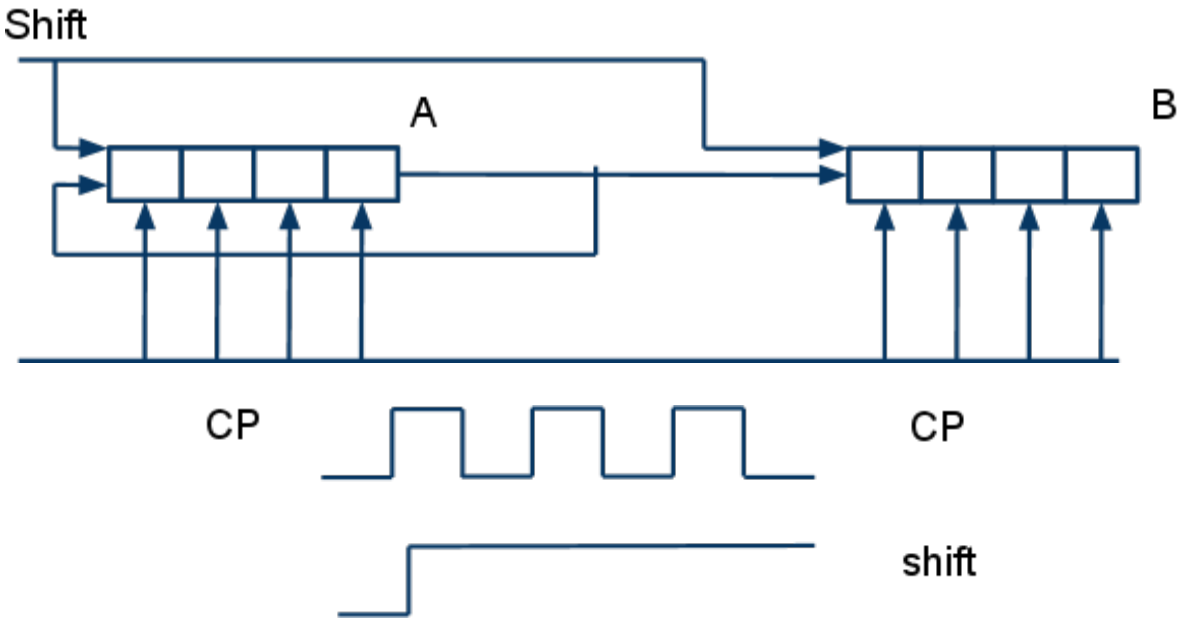
t cond : B ← A

cond musi ustabilizować się przed pojawieniem się t !!

układ w środku - realizuje cond - warunek

Przesyłanie szeregowe - implementacja

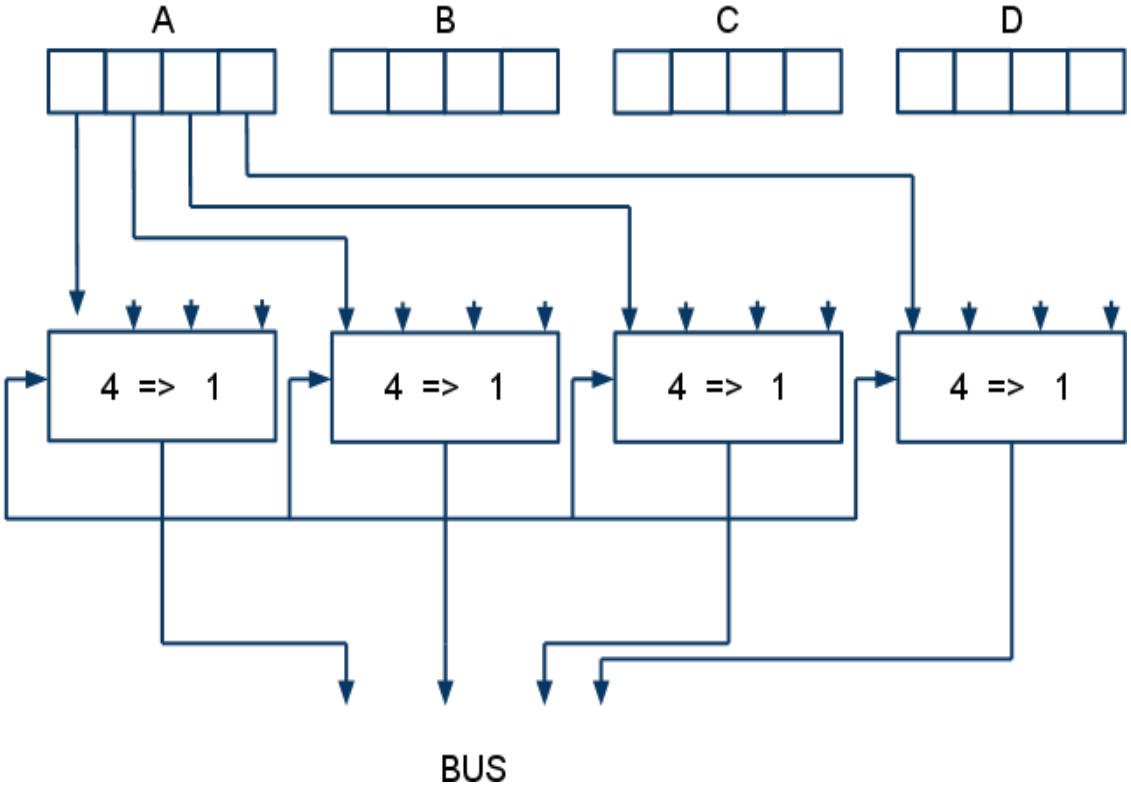
shift: $B_1 \leftarrow A_4$, $A_1 \leftarrow A_4$,
$A_i \leftarrow A_{i-1}$, $B_i \leftarrow B_{i-1}$, i = 2, 3, 4



shift - sygnał sterujący
CP - sygnał taktujący

Przesyłanie magistralą - implementacja

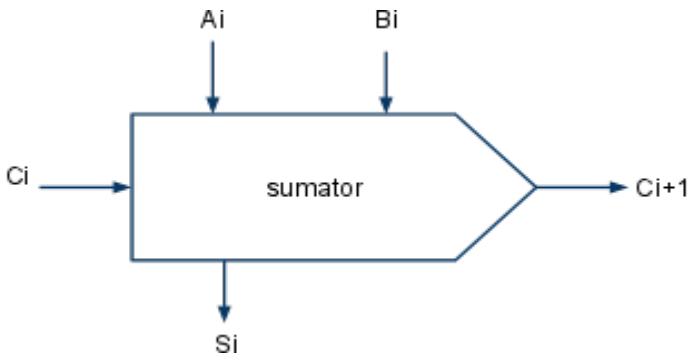
jest drogie i nieefektywne



W praktyce: bramki 3-stanowe.

Arytmetyczne

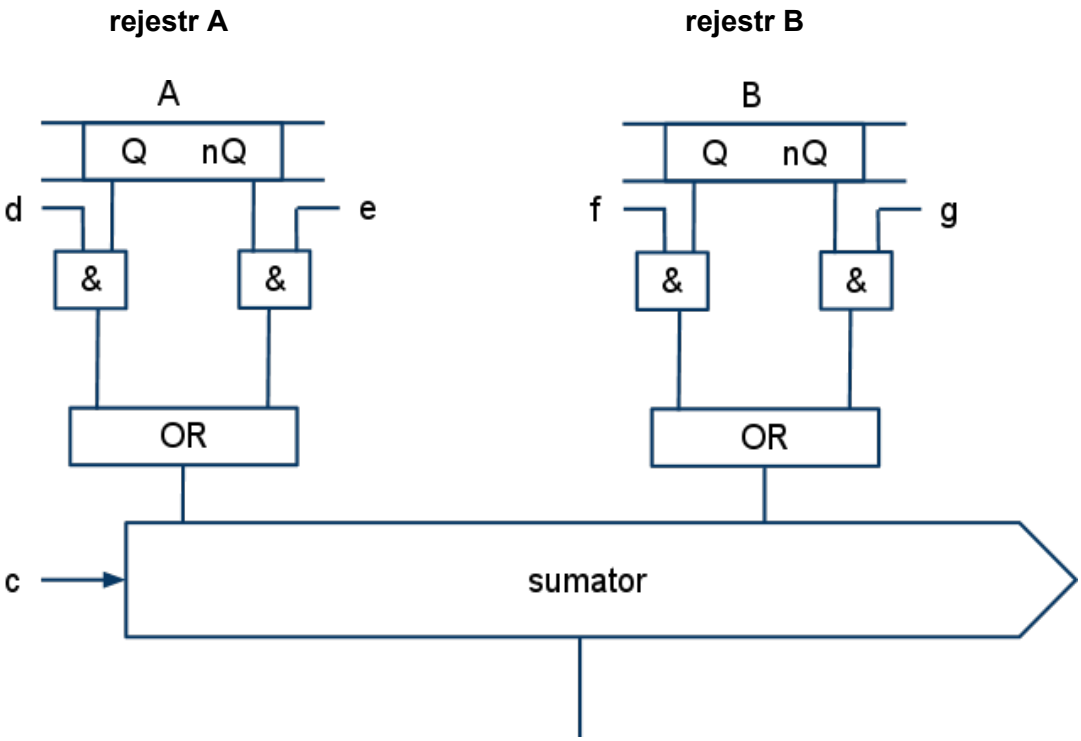
S <- A + B
A - B
A + 1
A - 1
¬A
¬A + 1
¬A + B
¬A + B + 1
…



U2: $A = - a_{n+1} 2^n + \Sigma a_1$
$A = (0101)_{U2} = 5_{10}$
$B = (1101)_{U2} = -8 + 5 = -3$

$-A = \neg A + 1$
$A = 0101, \neg A = 1010$
$\neg A + 1 = 1011 = -8 + 3 = -5_{10}$

Implementacja



Logiczne

x	y	f
0	0	
0	1	
1	0	
1	1	

4 wartości => 2^4 różnych funkcji f

np

x	y	~(x v y)	x v ~y …..
0	0	1	1
0	1	0	0
1	0	0	1
1	1	0	1

Przesuwania informacji

Np.: 01011001

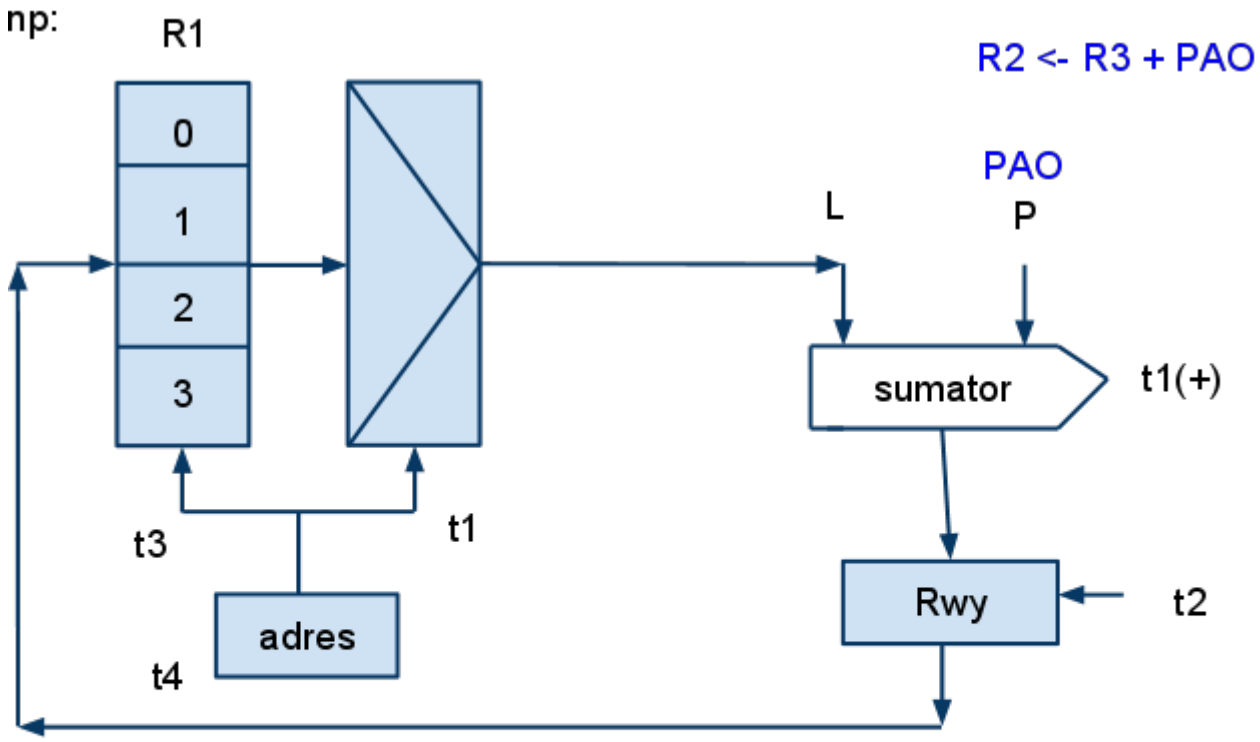
- shl -> 10110010
- shr -> 00101100

1 - nadmiar

- ashl -> 00110010 mnożenie przez 2
- ashr -> 00101100 dzielenie przez 2

- cil -> 10110010
- cil -> 10101100

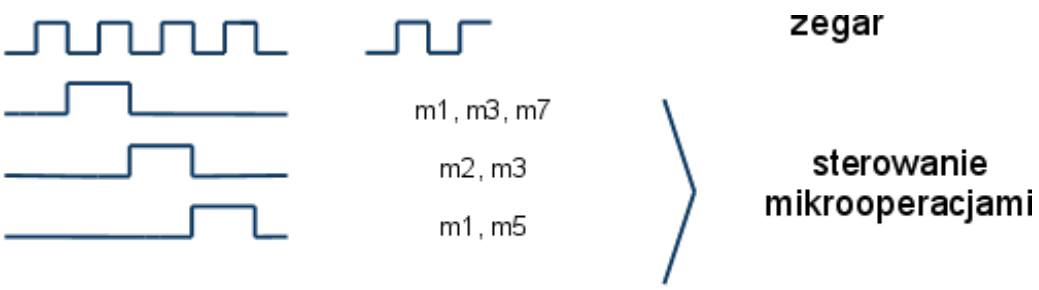
Funkcje sterujące (warunki)



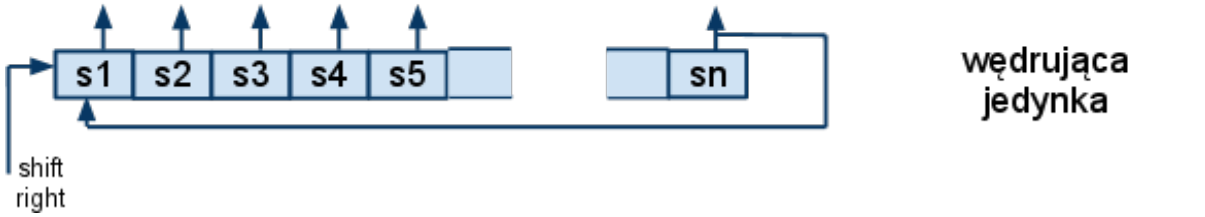
adres (a0 a1)	rejestr
0 0	-> R0
0 1	-> R1
1 0	-> R2
1 1	-> R3

ao a1 t1 : Wy <- L + P { L = R3}
t2: Rwy <- Wy, adres <- 10
t3: Rj <- Rwy

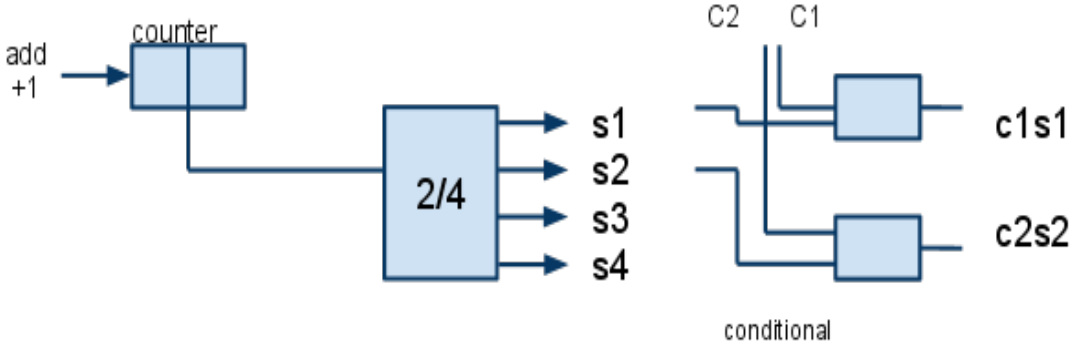
Generatory sygnałów synchronizacji



np: m1 oznacza MAR <- PC (bezwarunkowo)



stany
uwarunkowane



Przykładowa architektura procesora (projekt: poziom języka wewnętrznego -> poziom RTL)

Założenia: - procesor ogólnego przeznaczenia
- możliwie najprostszy

a) co wynika z modelu obliczeniowego (v.N.) ?

PAMIĘĆ----------łącze----------PROCESOR
zakodowany algorytm wykonuje rozkazy
sekwencja rozkazów

------>
rozkazy
←-----
dane

Pamięć = pamięć operacyjna (PAO)

Co wynika z konfrontacji koszt vs efektywność programowania?

- zestaw rejestrów ogólnego przeznaczenia,
- długość słowa i pojemność PAO
- lista rozkazów (instrukcji)
- zestaw formatów danych i rozkazów

Zgodnie z założeniem:

- jeden rejestr A
- słowo 16b
- min PAO: 64k słów
- min lista rozkazów:
    - przesłanie do / z PAO i rejestrów
    - arytmetyczne, logiczne, porzesuwanie
    - sterowanie, badanie warunków
    - wejście / wyjście
    - stop

Program wynikowy w języku wewnętrznym procesora

Zawartość RAM - przykład:

- 100 300 00.....
- 101 -5 1.....
- 102 ??? ……..
- …

- 201 ADD *,+ 100

kod rozkazu | tryby adresowania | adres

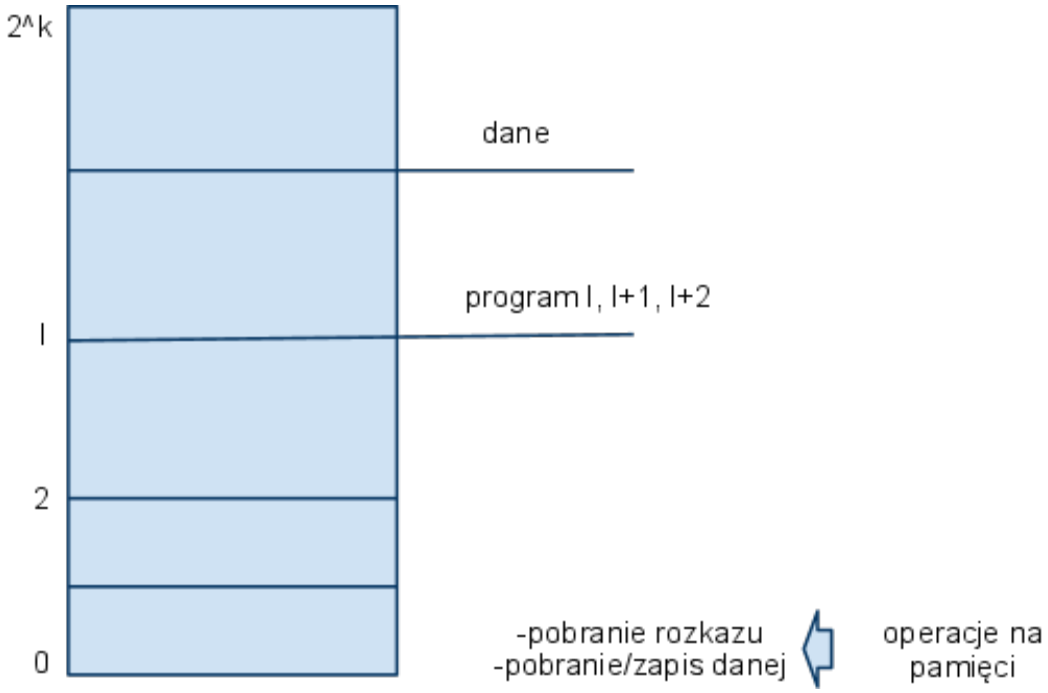
0000 | 11 | 1100100 -> słowo 01kowe zapisane w RAM

słowo 16 bitowe

część operacyjna | część adresowa

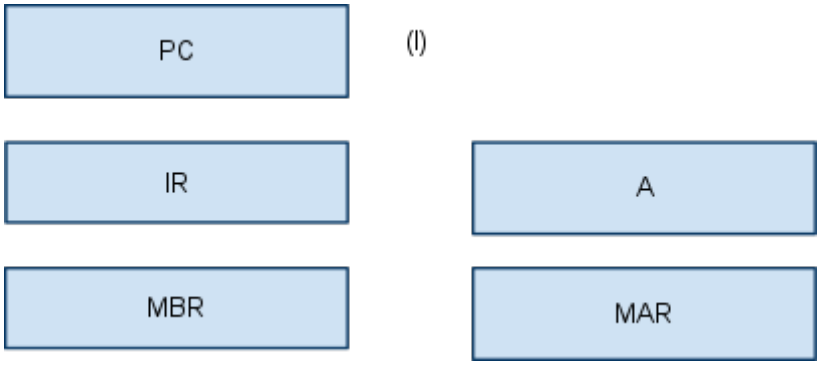
Pamięć operacyjna (PAO) i rejestry procesora

PAO



adresy PAO

rejestry procesora



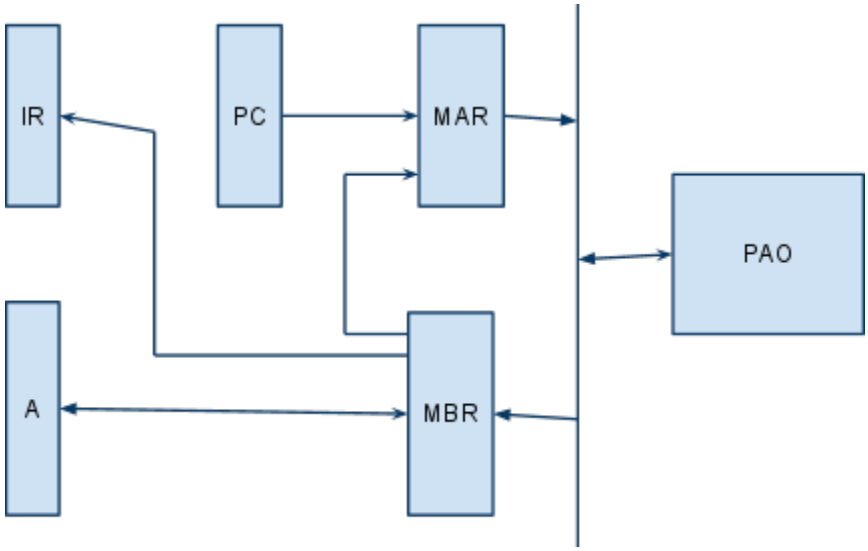
Długość rejestrów zależy od wielkości PAO, długości słowa, liczby rozkazów, organizacji procesora itd.

- cykl pobrania rozkazu:

MAR <- PC
odczyt PAO
MBR <- M
IR <- część operacyjna
MBR (*)

- cykl wykonania rozkazu:
A <- dana z PAO:

MAR <- adres MBR
odczyt PAO
MBR <- M
A <- MBR (*)

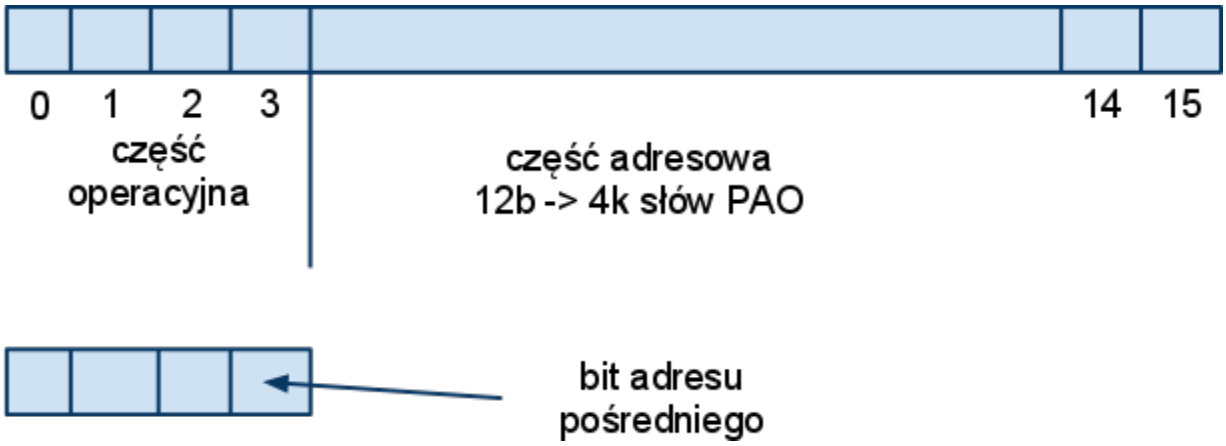


(*) - należy rozróżnić (sterowanie)

Pierwsze zadanie architekta

Jakie instrukcje wybrać i jak optymalnie przydzielić im kody?

Podział słowa rozkazowego:



8 kodów rozkazów
- 7 rozkazów adresowych
- 1 rozkaz bezadresowy

adres pośredni - rozszerza PAO
rozkazy bezpośrednie - rozszerzają listę rozkazów

Lista rozkazów

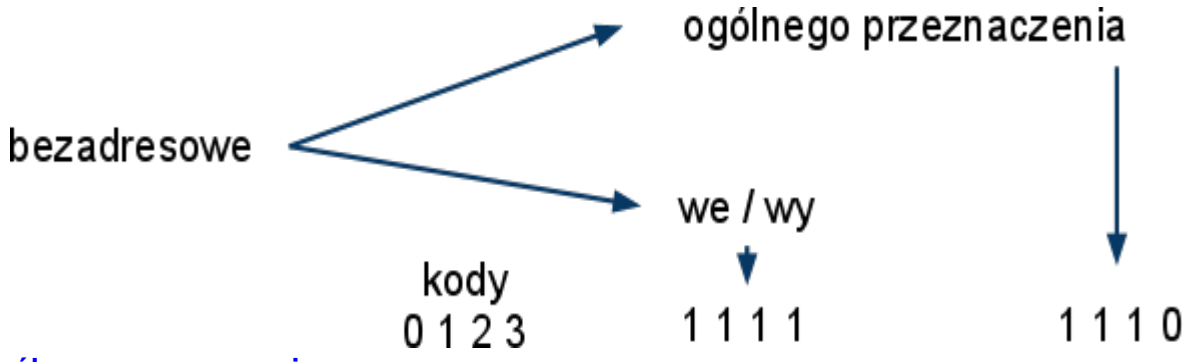
Rozkazy adresowe:

- **ADD** **A <- M + A** **dodaj do rejestru PAO**
- **LDA** **A <- M** **załaduj rejest z PAO**
- **AND** **A <- A & M** **iloczyn logiczny**
- **STA** **M <- A** **zapamiętaj w PAO**
- **BRA** **PC <- adres** **skocz**
- **JS** **stos <- pc** **skocz ze śladem**
  PC <- adres
- **ISZ** **M <- M + 1** **inkrementuj PAO, skip gdy 0**
  gdy M=0 wtedy
  PC <- PC+1

Podobieństwa i różnice

• Zawartość RAM - kod wynikowy EXE symbolicznie: • … • a adres A[1] • b -n • c S • … • I CLA • I+1 ADD,*,+,a • I+2 ISZ b • I+3 BRA I+1 • I+4 STO • ….	• zawartość RAM - kod wynikowy EXE symbolicznie • … • a adres A[1] • b -n • c S • … • I CLA • I+1 ADD,*,a • I+2 ISZ a • I+3 ISZ b • I+4 BRA I+1 • I+5 STO • ….

Lista rozkazów



Ogólnego przeznaczenia:

CLA A<-0
CLO O<-0
NEGA A<- ~A
NEGO O<- ~O
INC A<-A+1
…..

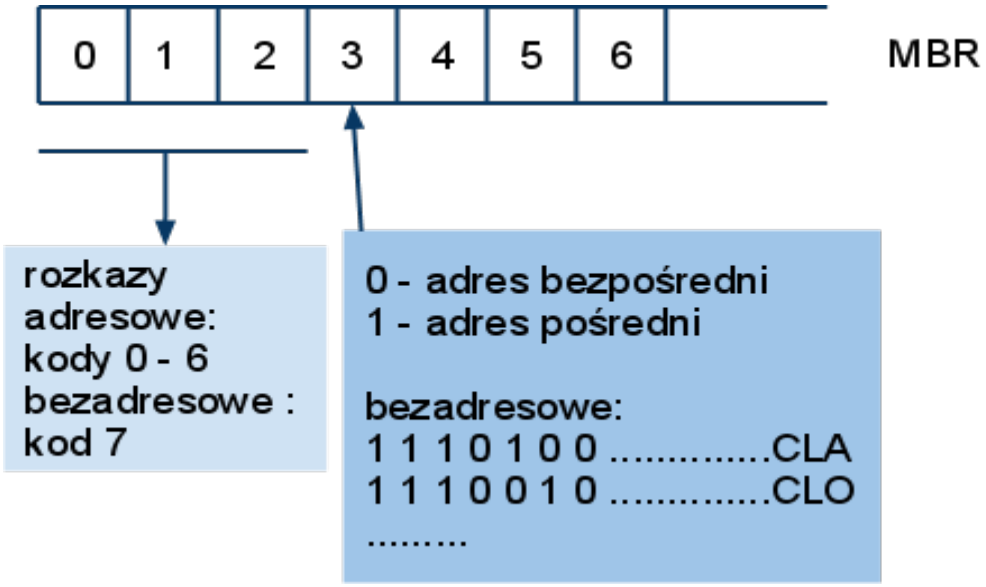
(arytmetyka)

SPA PC <- PC + 1 gdy A>=0
SWA PC <- PC + 1 gdy A<0
SZA PC <- PC + 1 gdy A=0
…..
STOP

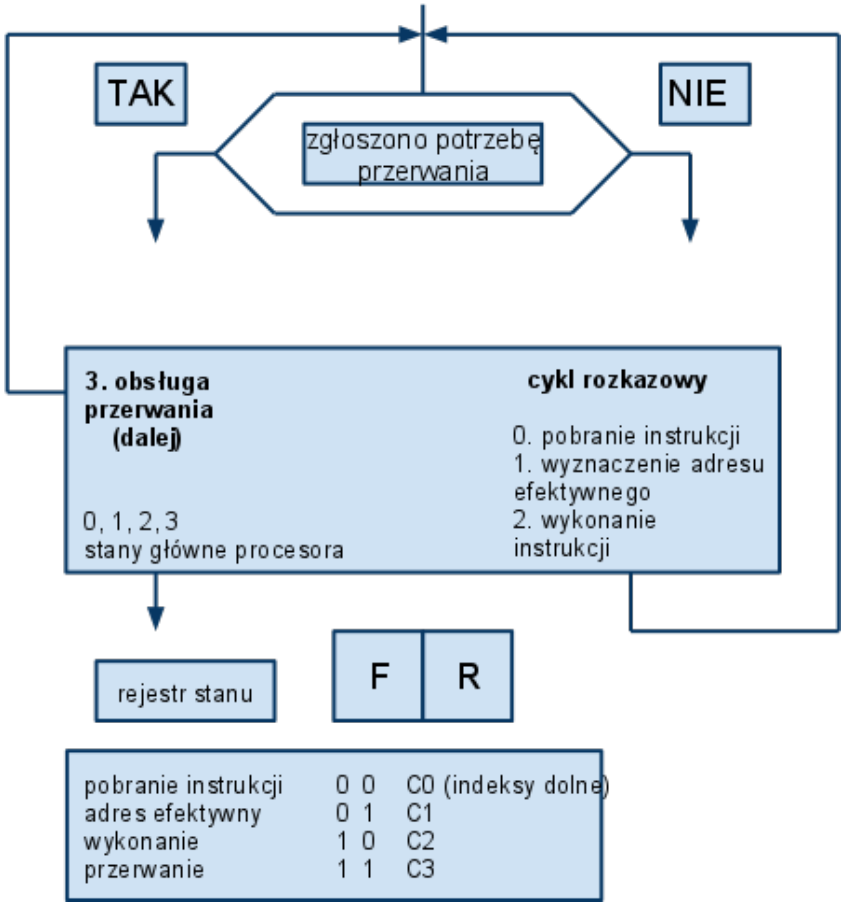
(sterowanie)

O to bit nadmiaru
We/wy zostanie omówione dalej

Kody rozkazów:



Sterowanie procesora - stany (cykle) główne



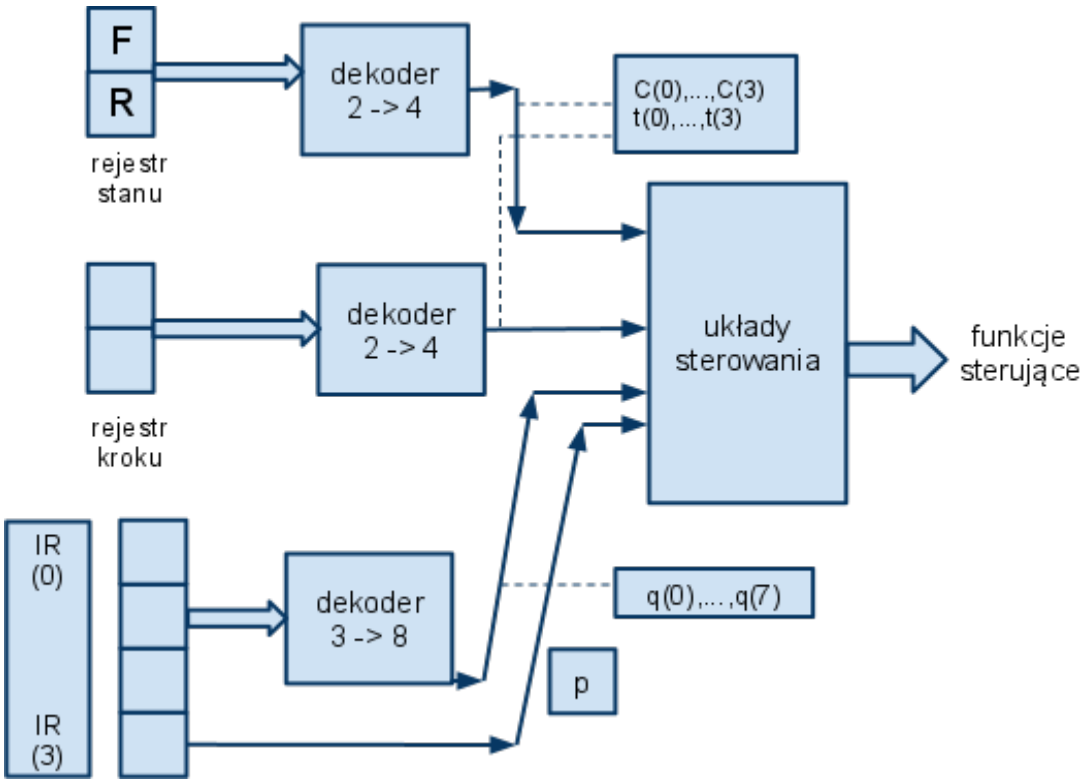
Sterowanie procesora - taktowanie wewnątrz cykli

Na ile taktów (kroków) zegara trzeba rozbić najdłuższą sekwencję sterowania wewnątrz cykli głównych?

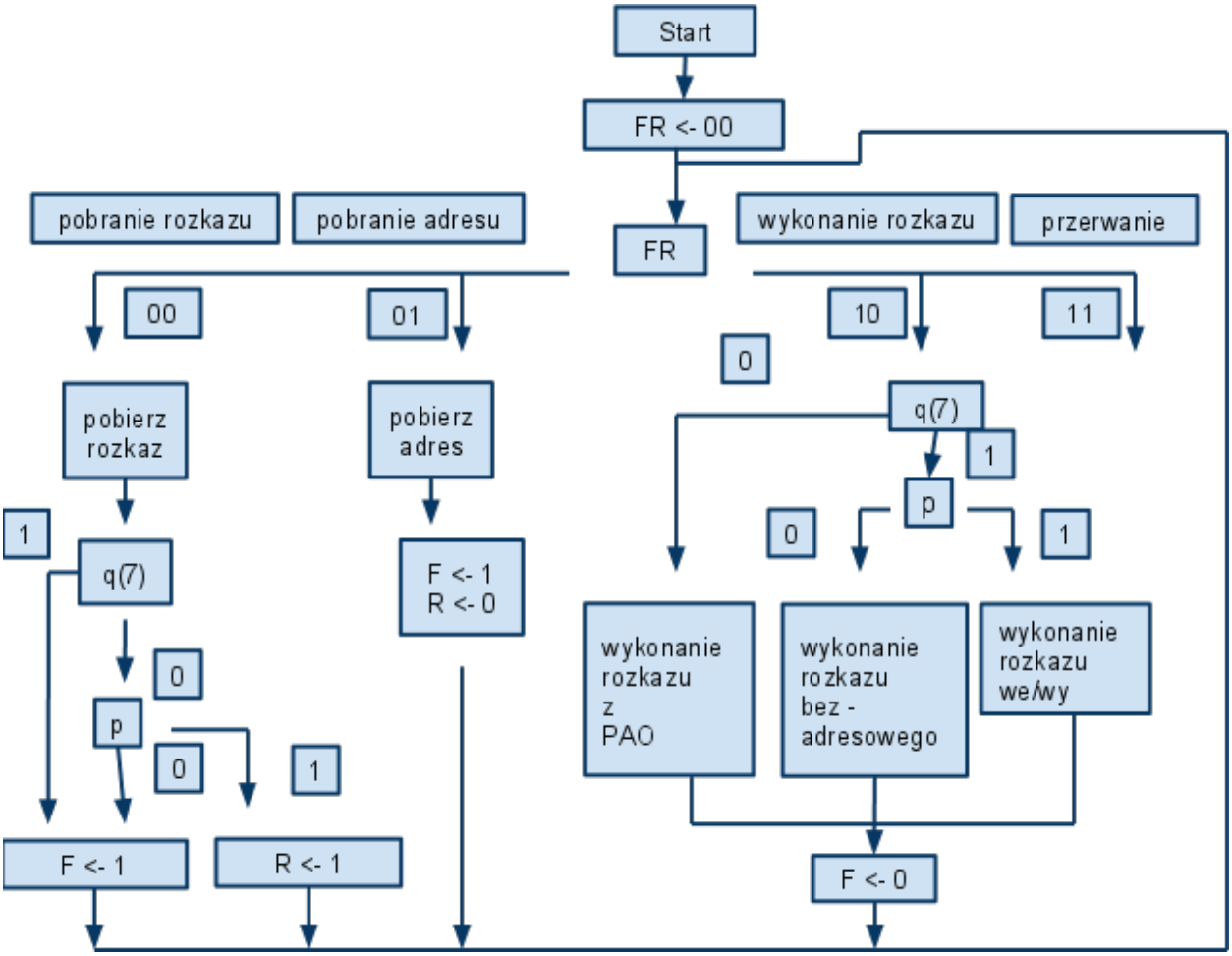
np. pobranie instrukcji (rozkazu):

- t_0 : MAR <- PC, PAO start odczyt
- t_1 : MBR <- M {odczyt PAO}
- t_2 : IR[0 - 3] <- MBR [0 - 3] {kod rozkazu}
- t_3 : jeżeli IR[3] = 1 i IR[0 - 2] $\neq$ 111 to przejdź do C_1 w przeciwnym razie do C_2

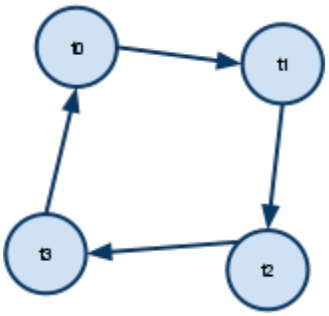
Struktura jednostki sterującej procesora



Algorytm sterowania



Cykl pobrania rozkazu



- C_0t_0 : MAR <- PC, PAO start odczyt
- C_0t_1 : MBR <- M, PC <- PC + 1 {rozkaz w MBR}
- C_0t_2 : IR[0-2] <- MBR[0-2], IR[3] <- MBR[3], MBR[0-3] <- 0

- $\sim q_7 p C_0 t_3$: R <- 1
- $(q_7 \vee \sim p) C_0 t_3$: F <- 1

Cykl pobrania adresu pośredniego

- C_1t_0 : MAR <- MBR, PAO start odczyt {adres adresu operandu}
- C_1t_1 : MBR <- M
- C_1t_2 : “nic”
- C_1t_3 : F <- 1, R <- 0 {zawsze wykonanie rozkazu}

Cykle wykonania rozkazów

ADD {A <- A + M}

- $C_2q_0t_0$: MAR <- MBR, PAO start odczyt
- $C_2q_0t_1$: MBR <- M
- $C_2q_0t_2$: OA <- A + MBR

LDA {A <- M}

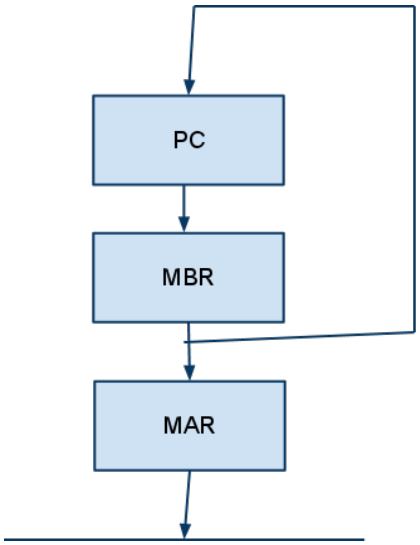
- $C_2q_1t_0$: MAR <- MBR, PAO start odczyt
- $C_2q_1t_1$: MBR <- M, A <- 0
- $C_2q_1t_2$: A <- A + MBR

BRA {PC <- adres}

- $C_2q_2t_0$: PC <- MBR

przyjmujemy, że w t0 i t1 dokonujemy operacji na pamięci a w potem dopiero operacji rozkazowych, tak aby było łatwiej testować

JS (skocz ze śladem)



- $C_2q_5t_0$: MAR <- MBR, MBR <- PC, PC <- MBR, PAO start zapis
- $C_2q_5t_1$: M <- MBR
- $C_2q_5t_2$: PC <- PC + 1

ISZ

- $C_2q_6t_0$: MAR <- MBR, PAO start odczyt
- $C_2q_6t_1$: MBR <- M
- $C_2q_6t_2$: MBR <- MBR + 1, PAO start zapis
- $C_2q_6t_3$: M <- MBR, gdy MBR = 0 to PC <- PC + 1

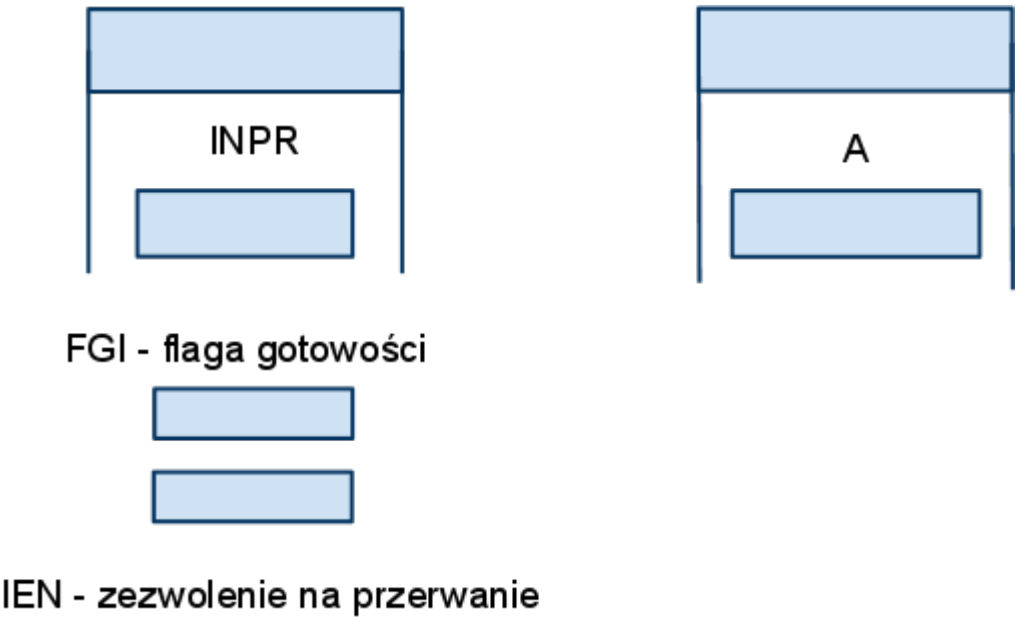
Bezadresowe

- $C_2q_7p\ t_3$: MBR_i : i-ty rozkaz bezadresowy

NEGA

- $C_2q_0p\ t_0$: MBR_7 : A <- ¬A

We/wy, system przerwań



Transmisja pod kontrolą programu

klawiatura:
Procesor ma 3 rejestry na obsluge klawiatury - jeden na kod znaku(8 bitowy), flage gotowosci i zezwolenie na przerwanie : INPR, FGI, IEN

- FGI <- 0 (restart)
- jeżeli FGI - 0 i nadano “znak” na klawiaturze
  to INPR <- znak oraz FGI <- 1

procesor:

- jeżeli FGI = 1 to A <- INPR
  oraz FGI <- 0

protokół transmisji

potrzebne rozkazy we:

- INP -> A[8-15] <- INPR, FGI <- 0
- SKI -> gdy FGI = 1 to PC <- PC + 1

program procesora:
- I SKI
- I+1 BRA I
- I+2 INP

sterowanie: $C_2q_7p\ t_3$: $MBR_{5,6}$

program transmisji

Transmisja za pomocą przerwania

klawiatura: po zapełnieniu INPR zgłasza potrzebę przerwania bieżącego programu

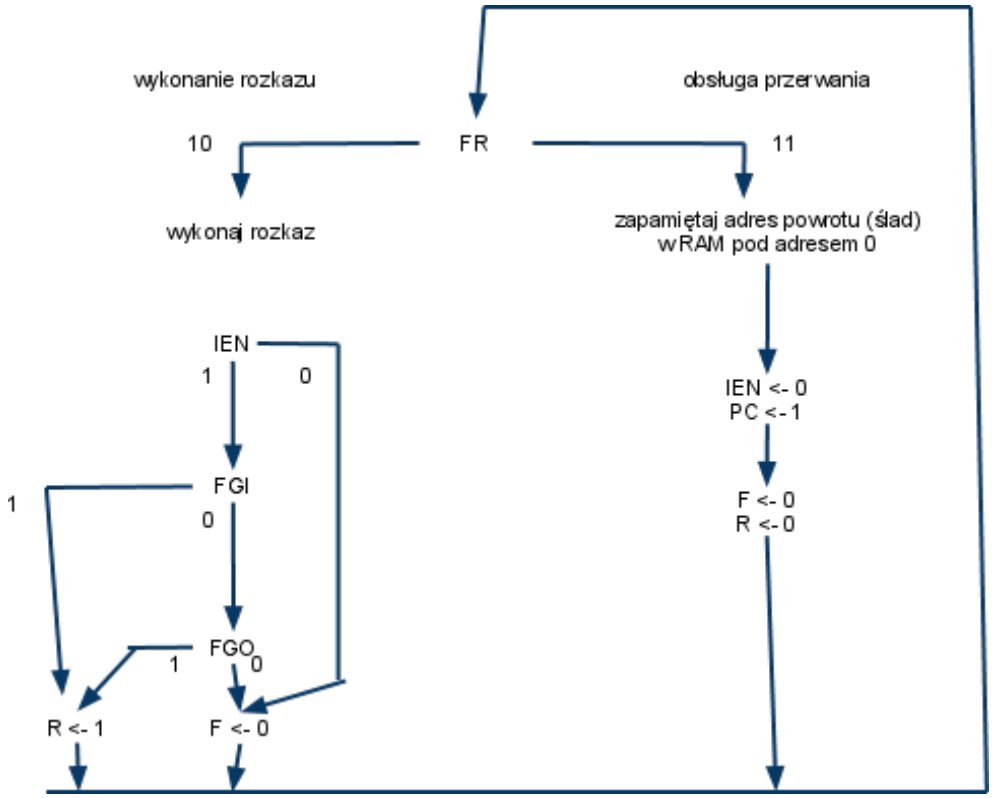
procesor: wywołuje program obsługi klawiatury

potrzebne **rozkazy we:**

- ION -> IEN <- 1 zezwolenie na przerwanie
- IOF -> IEN <- 0 zabronione przerwanie

Sterowanie: $C_2q_7p\ t_3 : MBR_{7,8}$

Uzupełnienie algorytmu sterowania



Modyfikacje

Modyfikacja cyklu wykonania rozkazu:

- C_2t_3(IEN (FGI v FGO)) : R←1
- C_2t_3~(IEN (FGI v FGO)) : F←0

Cykl Przerwania:

- C_3t_0 MBR ← PC, PC←0
- C_3t_1 MAR ← PC, PC←PC+1, PAO start zapis
- C_3t_2 M←MBR, IEN←0
- C_3t_3 F←0, R←0

Podsumowanie

Specyfikacja procesora (poziom RTL) zawiera:

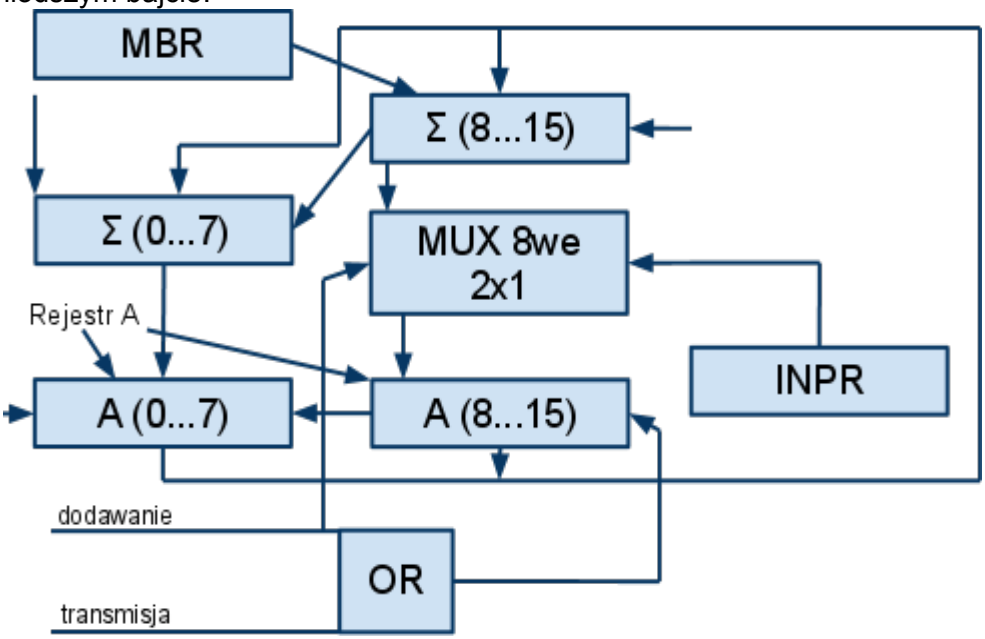
- Zestaw rejestrów (8) i przerzutników (8), dekoderów (3), generator zegara (t_0 - t_3), funkcje sterujące.
- Mikrooperacje dla wszystkich cykli i rozkazów.

Jednostka wykonawcza

Operacje na rejestrze A ($r = c_2 \wedge q_7 \wedge \bar{p} \wedge t_3$ -> 4 wejściowa bramka AND)

- $C_2 \wedge t_2 \wedge (q_0 \vee q_1)$: A ← A+ MBR
- $C_2 \wedge t_2 \wedge q_2$: A ← A ∧ MBR
- $C_2 \wedge t_1 \wedge q_1 \vee r \wedge MBR$: A ← A ∧ MBR_5 : A←0
- $r \wedge MBR_7$: A ← ~A
- $r \wedge MBR_9$: shr A, A(8)←0
- $r \wedge MBR_{10}$: shl A, A(15)←0
- $r \wedge MBR_{11}$: A ← A + 1
- $C_2 \wedge t_3 \wedge q_7 \wedge p \wedge MBR_5$: A[8-15] < INPR

8 operacji: dodwanie, koniunkcja, zerowanie, negacja, przesunięcie w lewo i prawo, następnik, zapis na młodszym bajcie.



- iloczyn logiczny
- zerowanie,
- negacja,
- przesunięcie w lewo i prawo
- następowanie
- zapis równoległy

↑ przyjeto że rejestr A ma takie możliwości

MUX 8we 2x1 - zwalnia urządzenie
Suma(8....15) - tryb pracy
A - tryb pracy

Odwzorowanie techniczne:

- Historia :

W TTL rejestr A np: 74281, układy kombinacyjne na bramkach (ALU np.74281)

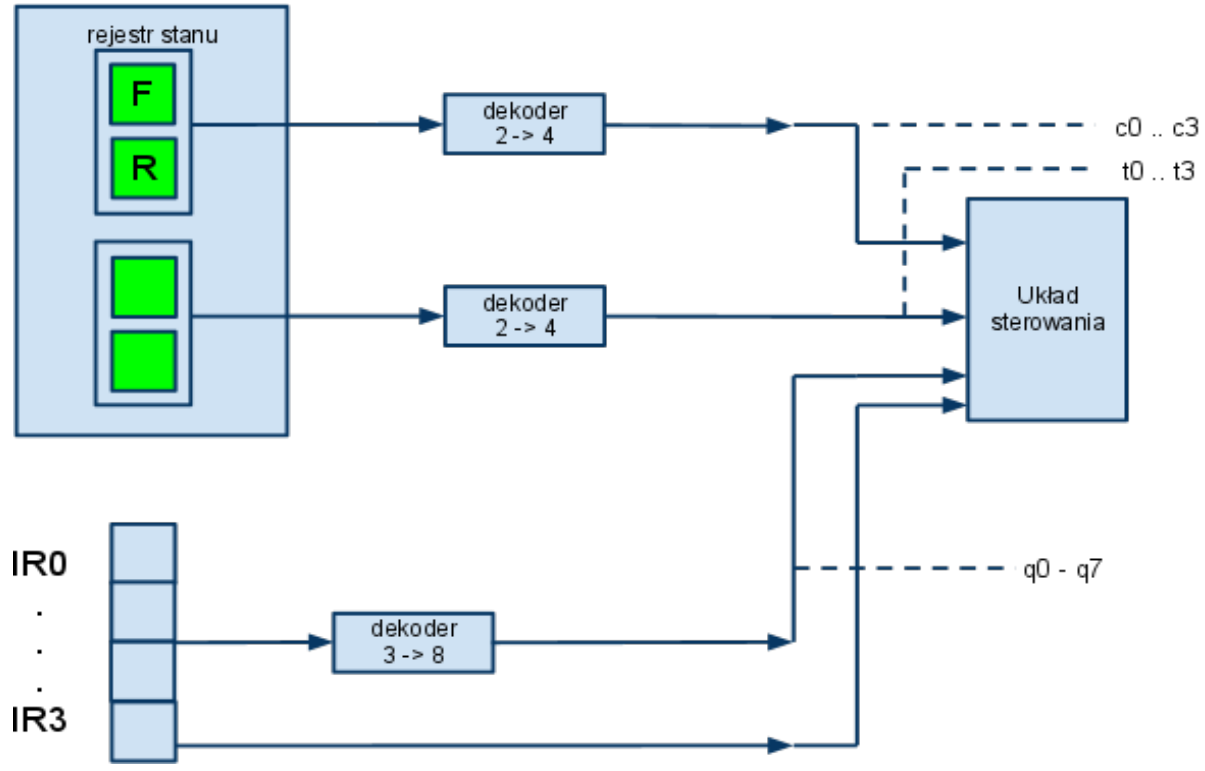
- Teraźniejszość:

Technika FPGA (semi custom), ASIC (full custom)

System automatycznego programowania:

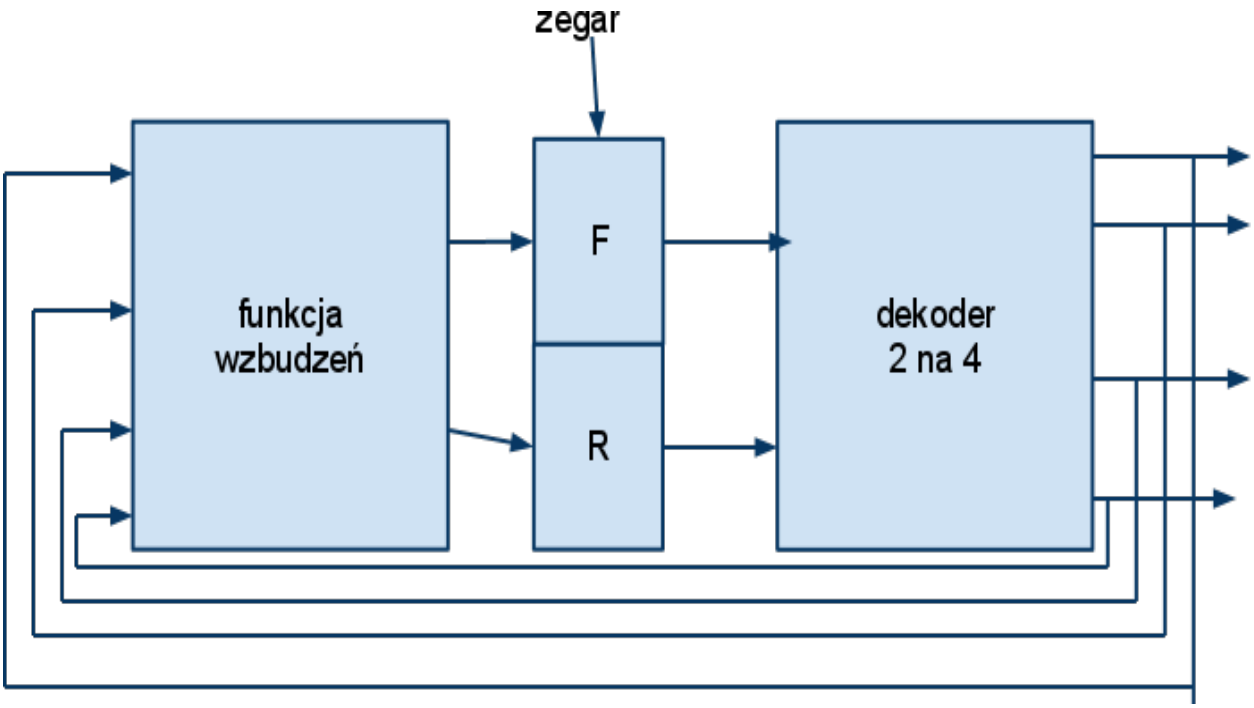
- System C (na bazie C)
- Systemy na bazie VHDLa lub AHDLa

Systemy wyspecjalizowane np. SIS - kodowanie stanów



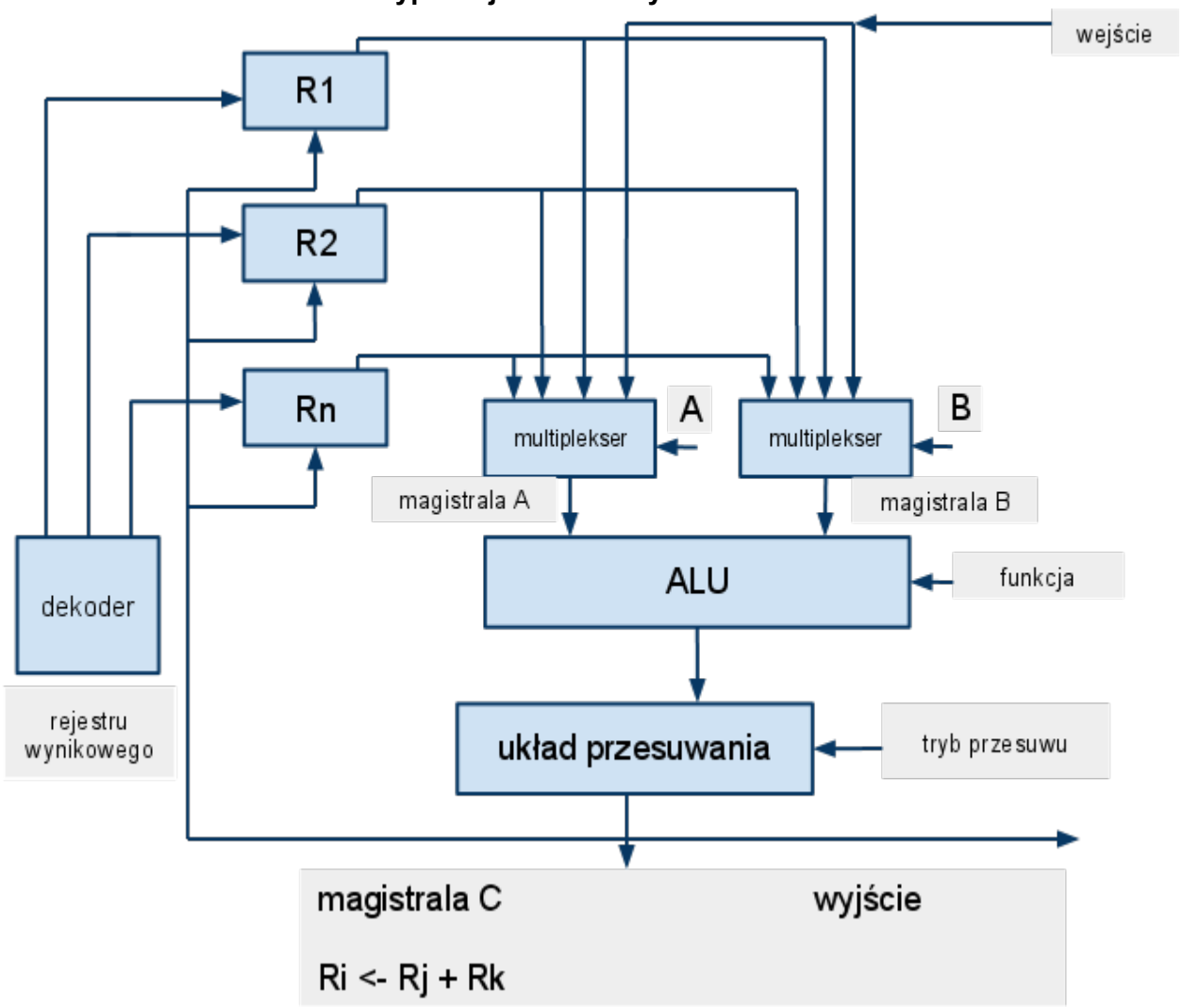
$\sim q_7 p C_0 t_3$ R←1
$(q_7 \vee \sim p)\ C_0 t_3$ F←1
$C_1 t_3$ F←1, R←0
$C_2 t_3$(IEN(FGI v FGO) R←1
$C_2 t_3 \sim$(IEN(FGI v FGO) F←0
$C_2 t_3$ F←0 R←0

Rejestr stanu głównego :



Inne elementy architektury procesora von Neumana

Typowa jednostka wykonawcza:



W typowym komputerze może być takich jednostek wykonawczych kilka.

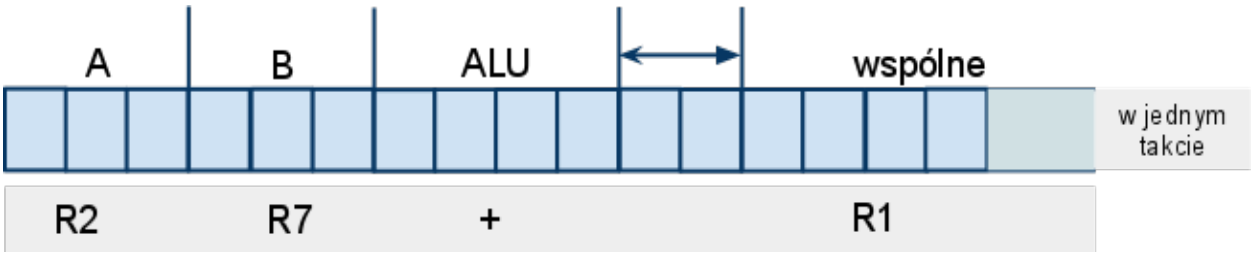
Dekoder służy do wskazania do którego rejestru powinien być zapisany wynik.
Typy danych:

- integer - uzupełnień do dwóch
- real - cecha i mantysa
- znakowe - reprezentacja znaków

Typowe sterowanie: mikroprogramowanie

W sterowaniu mikroprogramowym każdy rozkaz jest wykonywany przez mikroprogram (siedzi w ROM).

W ROM słowa sterujące jednostką wykonawczą
np : R1 <- R2 + R7



Jednostka sterująca steruje wyborem słów z ROM.

Mikroprogramowanie: rozkaz -> mikroprogram -> sekwencja mikrorozkazów

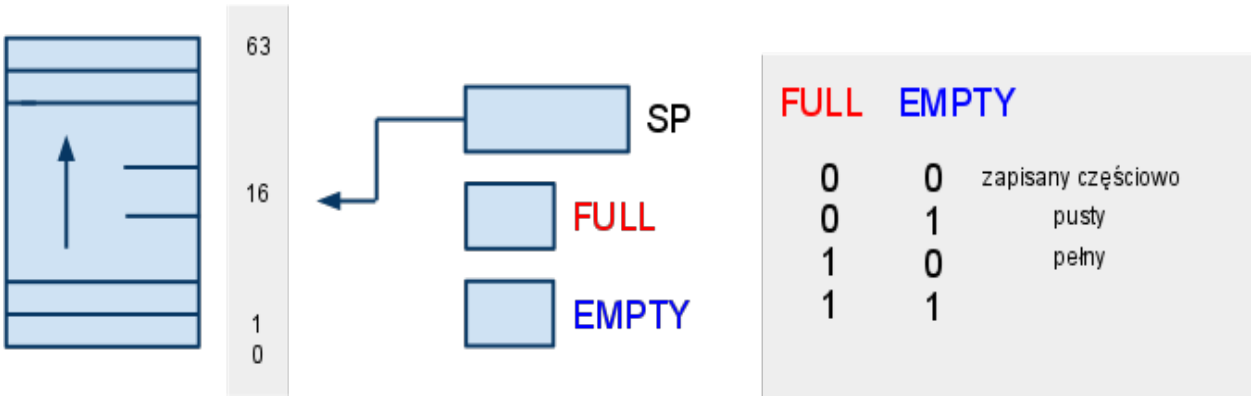
Mikrorozkaz: część wykonawcza + część sterująca (wraz z bitami statusu określa następny mikrorozkaz).

Zaleta: prostota układów JS.
Wada: wolniej niż ASIC.

Stos

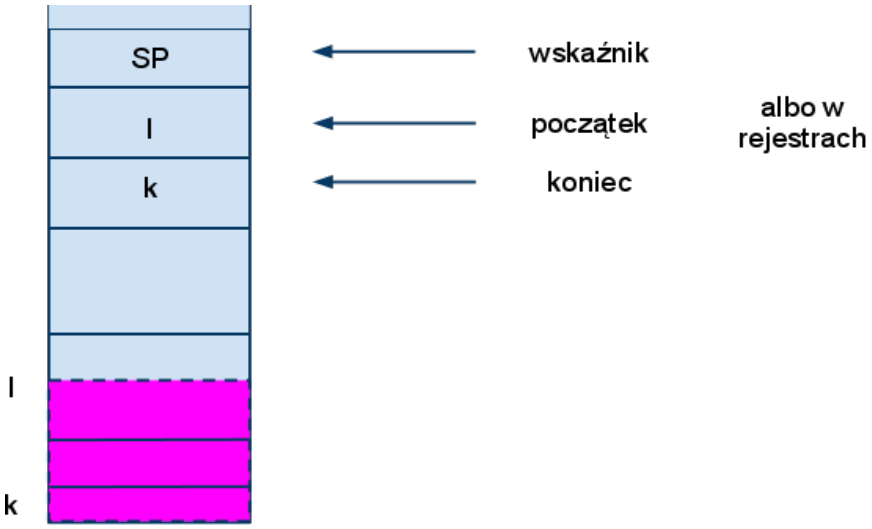
Stos jest nam niezbędny np do rekurencji, może też ułatwić nam zarządzanie przerwaniami.

a) samodzielny: SM - cykliczny po 63 znów jest 0



restart: SP = 0, FULL = 0, EMPTY = 1
push: SP <- SP + 1
SM(SP) <- MBR
gdy SP = 0 to FULL <- 1
EMPTY <- 0
pop: MBR <- SM(SP)
SP <- SP - 1
gdy SP = 0 to EMPTY <- 1
FULL <- 0

b) w PAO



Formaty rozkazów

- pole kodu operacji - 3bity
- pole adresu (rejestr, PAO) - 13 bitów
- pole trybu adresowania. - 1 bit

wartośći w układize, który zbudowaliśmy w czasie wykładu

Pole adresu: rejestr, PAO

Np. : X = (A + B) * (C + D)

- jeden adres w PAO (jeden rejestr programowany),

LDA A	LDA C
ADD B	ADD D
STO P1	MUL P1
	STO wynik

- dwa adresy, PAO i wiele rejestrów

MOV R1, A	MUL r1, R2
ADD R1, B	MOV adr. wyniku, R1
ADD R2, D	

- stos, rozkazy bezadresowe

PUSH A	PUSH C	MUL
PUSH B	PUSH D	POP
ADD	ADD	

Pole trybu adresowania

Cel: -> łatwość organizacji wskaźników, liczników, indeksacji, relokacji,
-> skrócenie pola adresu w rozkazie.

Tryby adresowania:

- domyślne - wynika z treści rozkazu np wyzerowanie akumulatora
- natychmiastowe - w części adresowej jest operand (odrazu jego wartość)
- rejestrowe - adres w rejestrze
- pośrednie
- z autoinkrementacją / autodekrementacją
- bezpośrednie
- względne - w części adresowej jest umieszczone przesunięcie względem innego rejestru
- indeksowe - procesor ma rejestr indeksowy
- z rejestrem bazowym - związane z realokowalnością programów

Pole kodu operacji

czy potrafisz zaprojektować każdą z tych operacji??

Przesyłanie danych:

- LD ładuj
- ST zapamiętaj
- MOV prześlij
- XCH zamień
- IN wprowadź
- OUT wyprowadź
- PUSH umieść
- POP pobierz

LD adr bezp.
@ adr pośr
$adr wzgl
#adr natych
adr(x) index
R1 rejestr
(R1) pośr, rej
(R1)+ autoindex
(standaryzacja trybu adresowania)

Operacje na danych:

arytmetyczne:
INC
DEC
ADD
SUB
MUL
DIV
ADDC
SUBC
NEG

logiczne na bitach:
CLR
COM - dopełnienie
AND
OR
XOR
CLRC - zeruj bit przeniesienia
SETC - ustaw bit przeniesienia
COMC
EI
OI

Przesuwania:

- SHR - w prawo logiczne
- SHRA - w prawo arytemetyczne
- ROR - cykliczne
- RORC - cyklizne z przeniesieniem
- SHL
- SHRL
- ROL
- ROLC

Sterowania:

- BR - skocz
- CALL - wywołaj podprogram
- CMP - porównaj
- JMP - skocze ze śladem
- RET - wróc z pod programu
- TST (testowanie)
- SKIP - przeskocz

Operacje warunkowe:

Bity statusu:

- c = 1 przeniesienie
  0
- s = 1 bit znaku
  0
- z = 1 wynik = 0
  0
- v = c_n+ c nadmiar w U2

- BZ (z=1)
- BP (s=0)
- BNZ (z=0)
- BM (s=1)
- BC
- BV
- BNC
- BNV
- BGE A>=B
- BE A=B
- BLE A<=B
- BNE A=/=B

dla liczb i modułów

Typy przerwań

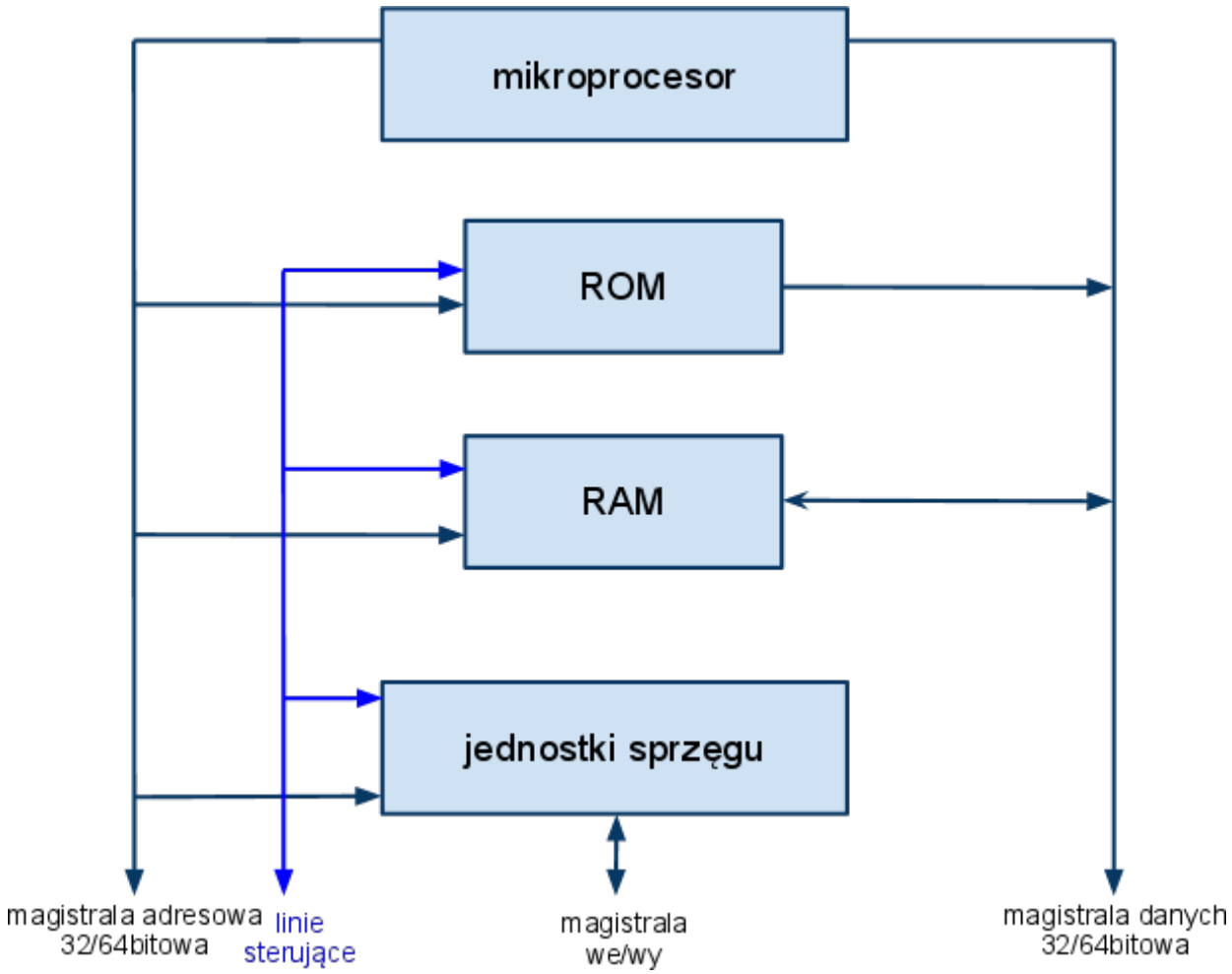
- Asynchroniczne -> zewnętrzne: we/wy, zegar czasu rzeczywistego, zasilanie. (poza procesorem)
- Synchroniczne -> wewnętrzne: pułapki, wyjątki(exceptions) (wewnątrz procesora)
- Programowe -> SV calls (przywołania systemu operacyjnego).

Wyjątki: nadmiar, dzielenie przez 0, niedopuszczalny kod rozkazu, przepełnienie stosu, naruszenie ochrony pamięci.

Pułapki używane są do debuggowania.

Przywołania: obsługa transmisji przejście ze stanu “użytkownik” do stanu “system operacyjny”.

Typowa organizacja mikrokomputera



jednostki sprzęgu - brak standardów

Sygnały sterujące mikroprocesor

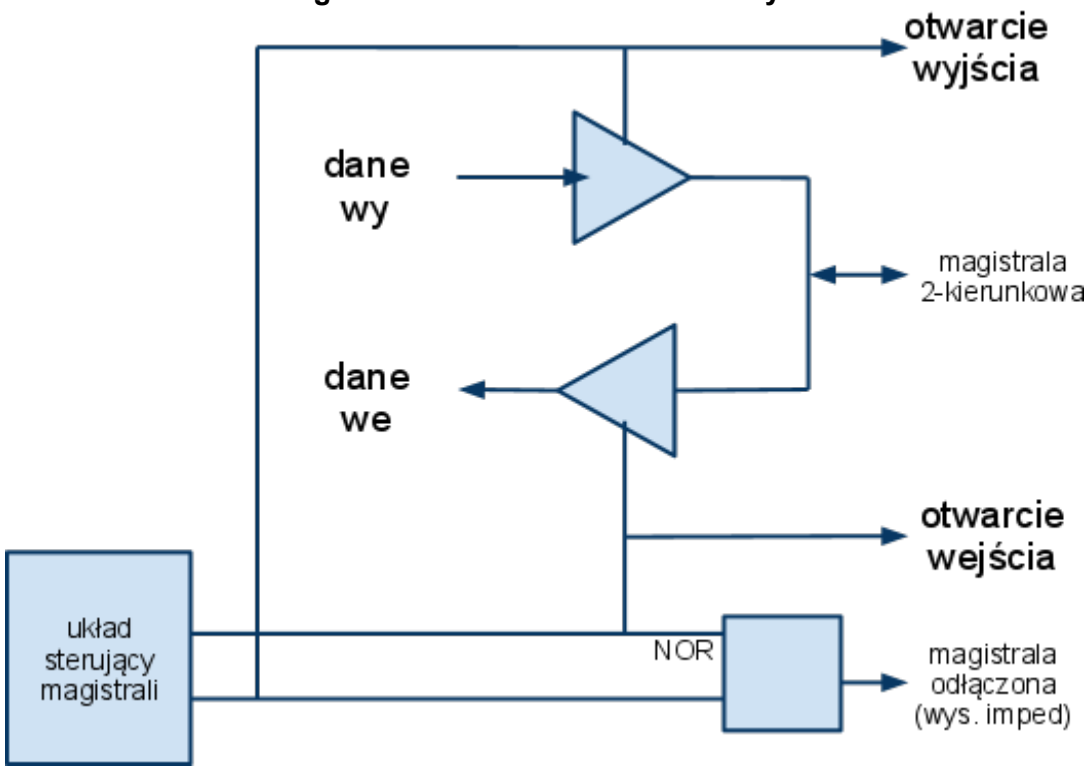
Wejściowe

-> zasilanie
-> we zegarowe
-> zerowanie
-> przyjęcie przerwania
-> żądanie przerwania
-> żądanie dostępu do magistrali
<-> magstrala danych

Wyjściowe

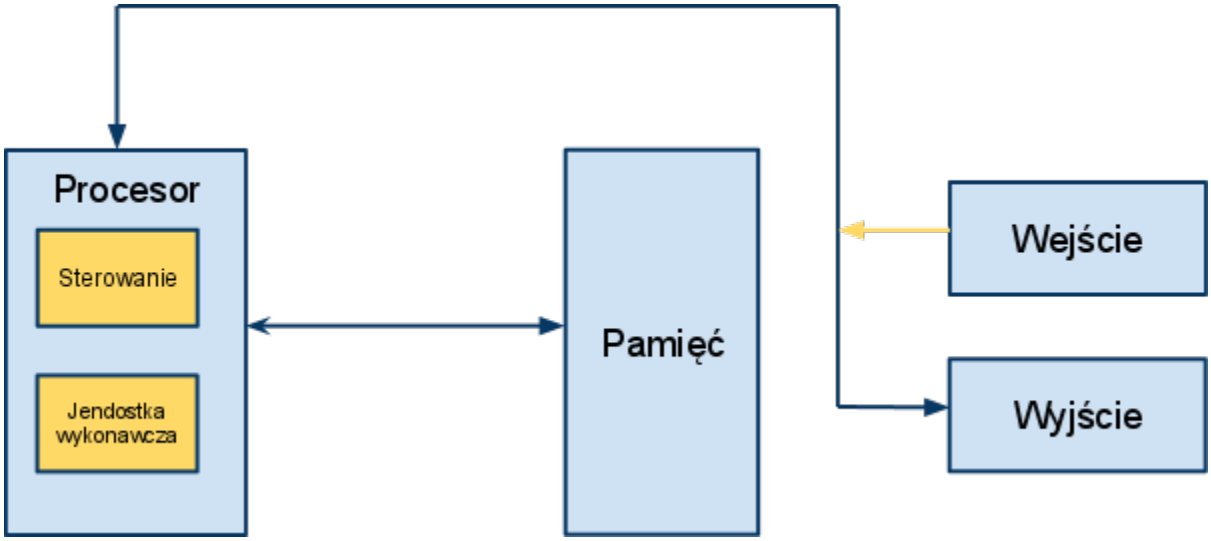
-> wy zegara
-> potwierdzenie gotowości przyjęcia przerwania
-> przydział magistrali
-> odczyt //PAO
-> zapis //PAO
-> magistrala adresu

Magistrala na elementach 3-stanowych



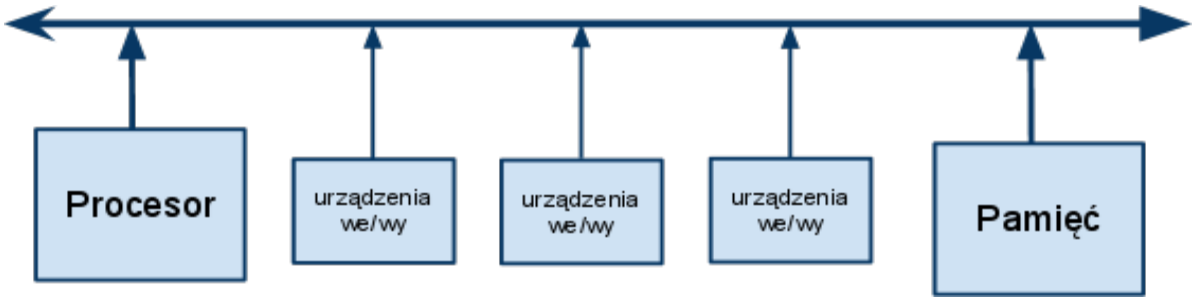
Magistrala: współdzielone łącze komunikacyjne

Architektura odpowiadająca modelowi von Neumana.



Tu pojawia się problem dobrania odpowiedniej długości drutu.

Magistrala



Zalety:

- uniwersalność -> dodaj/usuń urządzenie,
- niskie koszty -> współdzielone linie,
- standaryzuje projektowanie (standard magistrali).

Wady:

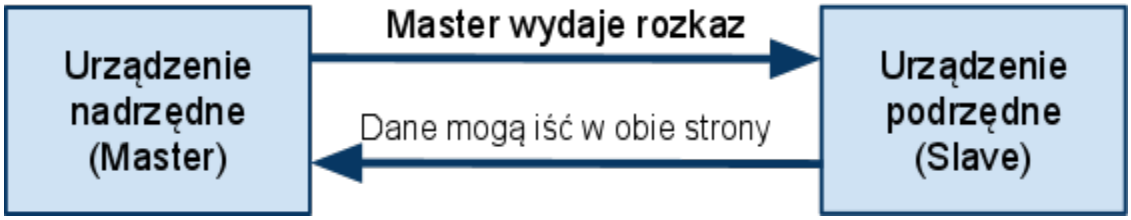
- “wąskie gardło” w przesyłaniu danych -> przepustowość magistrali może ograniczyć wydajność I/O.
- dysproporcja

Organizacja:

- linie sterowania (sygnały żądań i potwierdzeń, typ danych, błędy),

- linia danych (adresy, dane, rozkazy)

Typowa transakcja:



dokładnie jeden — może być wiele gdy odbiór

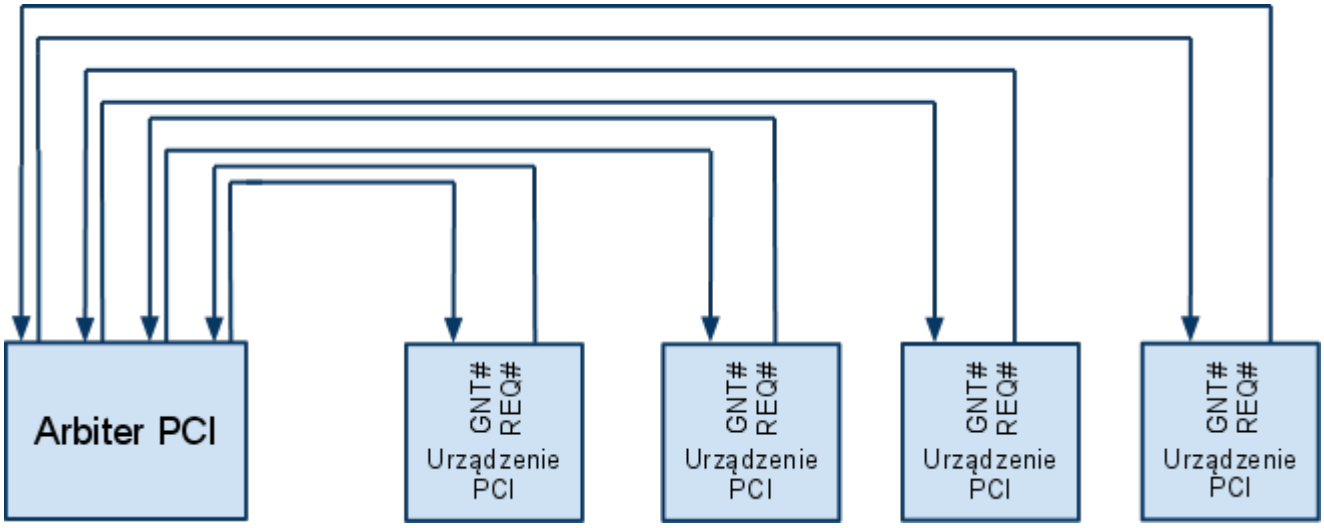
Przebieg transakcji:

- różne urządzenia **żądają dostępu** do magistrali (urządzenia we/wy, procesor, pamięć, karta graficzna, itd.),
- jedno staje się urządzeniem nadrzędnym (**bus master**) w tej transakcji, urządzenie nadrzędne wystawia na magistrale adres i typ żądania np. odczyt (zapis),
– urządzenie nadrzędne i to zaadresowane przez nie (**slave**) wysyłają dane w jedną lub drugą stronę (w zależności od złożonego żądania)
– urządzenie nadrzędne sygnalizuje, że skończyło operacje

Typy magistral

- **procesor - pamięć:** bardzo krótka, zwykle projektowana dla konkretnego procesora, wysoka przepustowość, (najszybsza)
- **systemowa** - łącząca procesor, pamięć, pobliskie komponenty (backplane - karty grafiki, karty sieciowe, ewentualnie szybkie urządzenia we/wy, inne wyspecjalizowane układy np.: DMA i ASIC), (średnia)
- **wejscia - wyjścia** - dołączająca pozostałe urządzenia we/wy (dyski, drukarki,...), inne komputery, (najwolniejsza)
    - podłączona do magistrali systemowej lub
    - do magistrali procesor - pamięć.

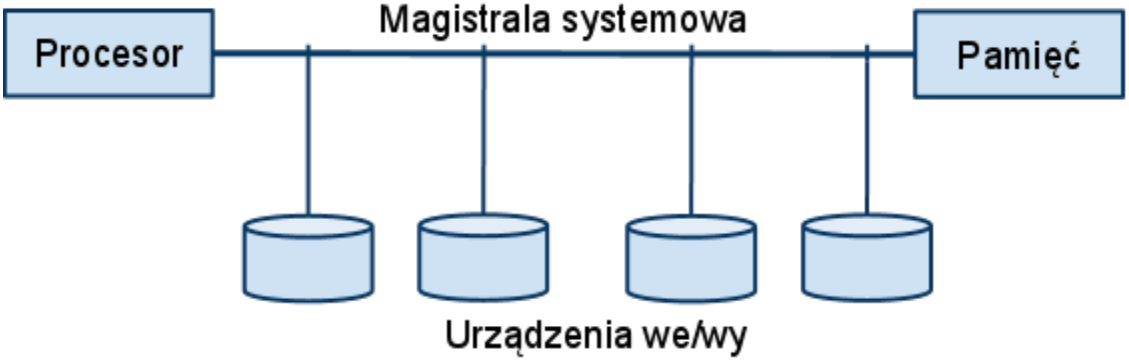
Magistrala PCI (independent request)



Specyfikacja PCI nie dyktuje szczególnego algorytmu arbitrażu.
Arbiter może wykorzystywać podejście “pierwszy zgłoszony - pierwszy obsłużony”, rozwiązanie cykliczne (round-robin) lub jakiś rodzaj układu priorytetów.

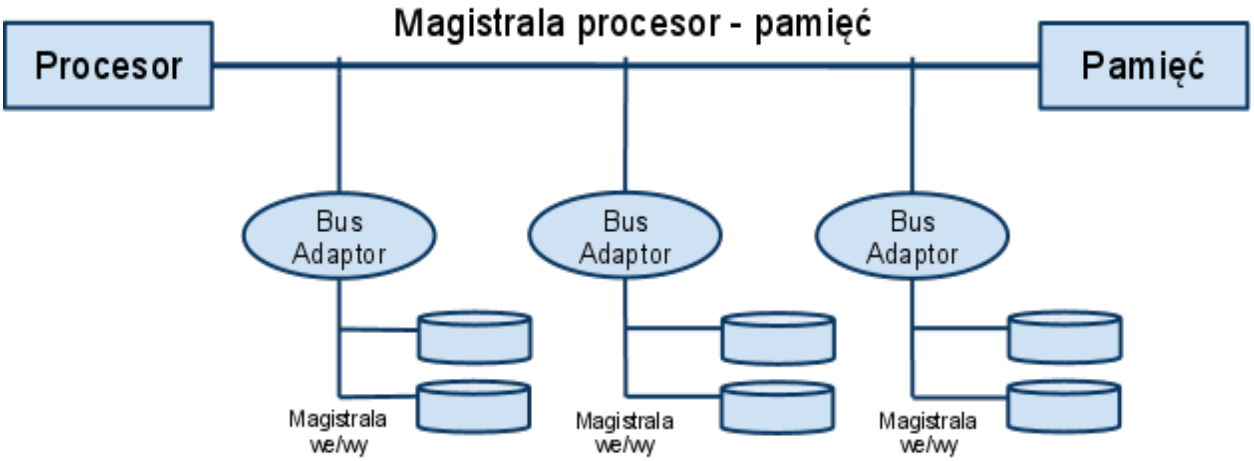
Moduł nadrzędny PCI musi korzystać z arbitrażu przy każdej transakcji, którą chce przeprowadzić, ponieważ pojedyncza transakcja składa się z fazy adresu, po której następuje jedna lub wiele faz danych.

System z jedną magistralą - Systemową

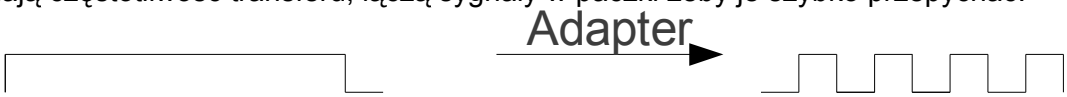


Działa wolno, ponieważ opóźniają ją urządzenia

System z dwoma magistralami

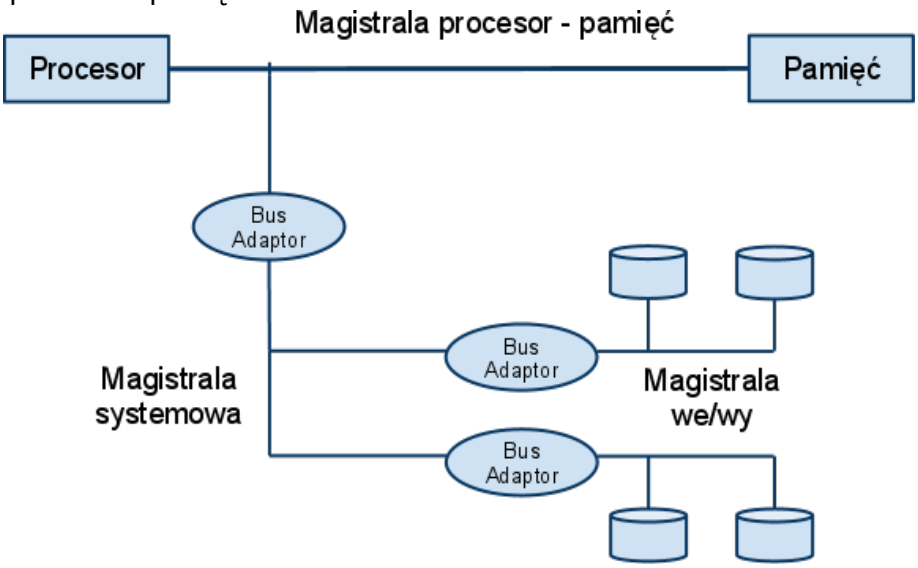


W każdym miejscu magistrali procesor-pamięć musi być taka sama częstotliwość, Adaptery zmieniają częstotliwość transferu, łączą sygnały w paczki żeby je szybko przepychać:



System z : trzema magistralami

- do magistrali systemowej podłączona jest magistrala wejścia/wyjścia, a magistrala systemowa do magistrali procesor - pamięć



Od dołu urządzenia najwolniejsze, im wyżej urządzenia szybsze.

Arbitraż

Rodzaje:

- **“bus master” = procesor** (rozwiązanie typu master - slave), procesor uczestniczy w każdej transakcji na magistrali -> niska wydajność systemu
- **“bus master” = jedno z wielu urządzeń.** tzw. **arbiter** decyduje, które z urządzeń będzie urządzeniem nadrzędnym (Według systemu priorytetów).

Arbitraż musi utrzymywać równowagę pomiędzy dwoma czynnikami:

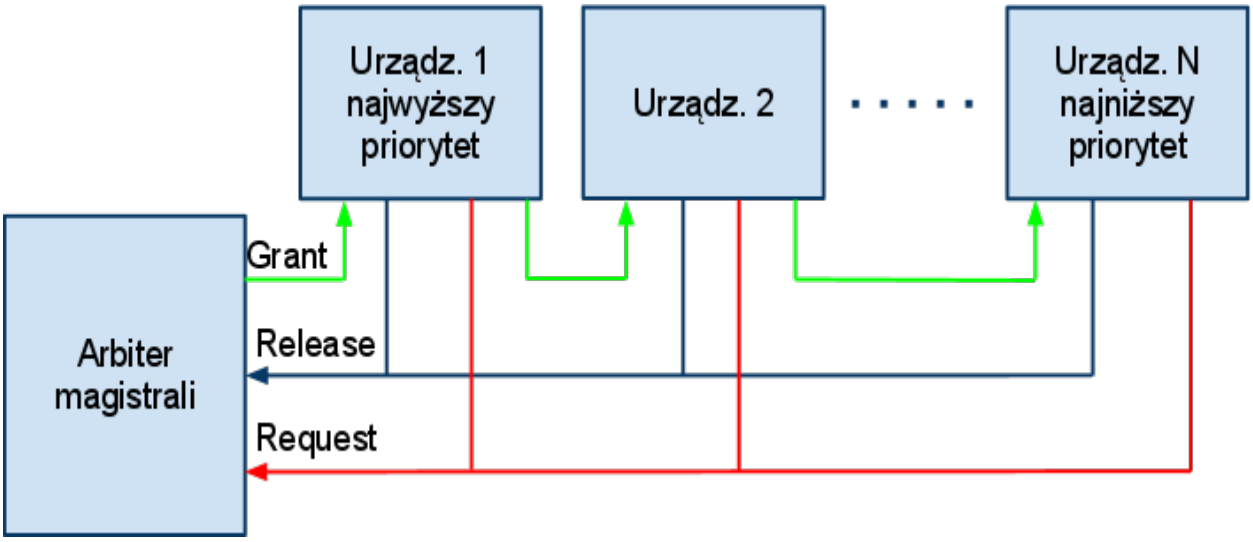
- **priorytet dostępu -** urządzenie o wyższympoziomie, powinno być obsłużone jako pierwsze.
- **dostęp do magistrali** nawet dla urządzeń o najniższym priorytecie (one również muszą uzyskać dostęp do magistrali)

Schematy Arbitrażu

“Daisy Chain”

+**Zaleta:** prostota!

-Wady: urządzenie o niższym priorytecie może nigdy nie uzyskać dostępu do magistrali, bo grant ogranicza prędkość magistrali.

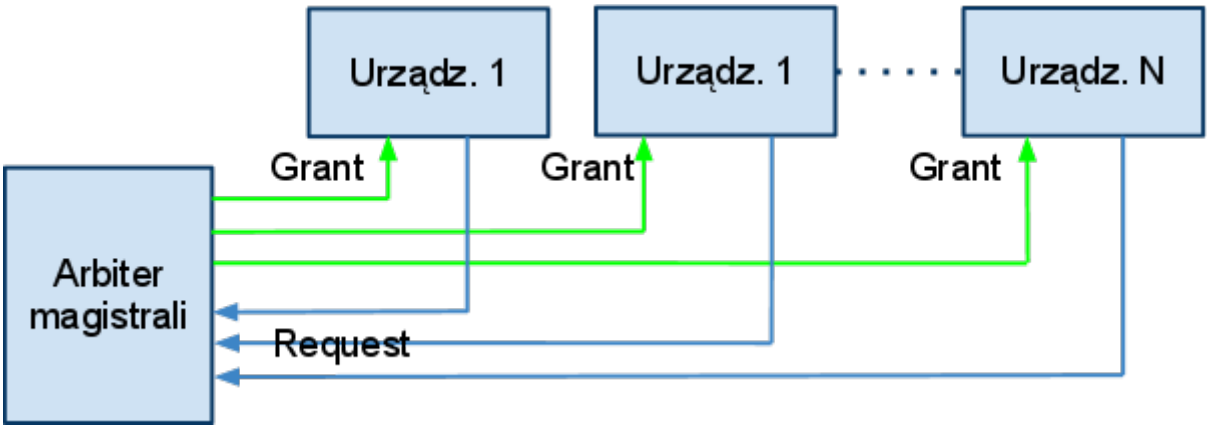


Jest to arbitraż bardzo podatny na zagrożenia. Wystarczy, że w jednym miejscu nastąpi uszkodzenie transmisji i wszystkie urządzenia z niższym priorytetem zostaną pozbawione sygnału grant, tzw. odcięcie (stack out)

Arbitraż scentralizowany (independent request)

+Zalety: najszybszy i najbezpieczniejszy

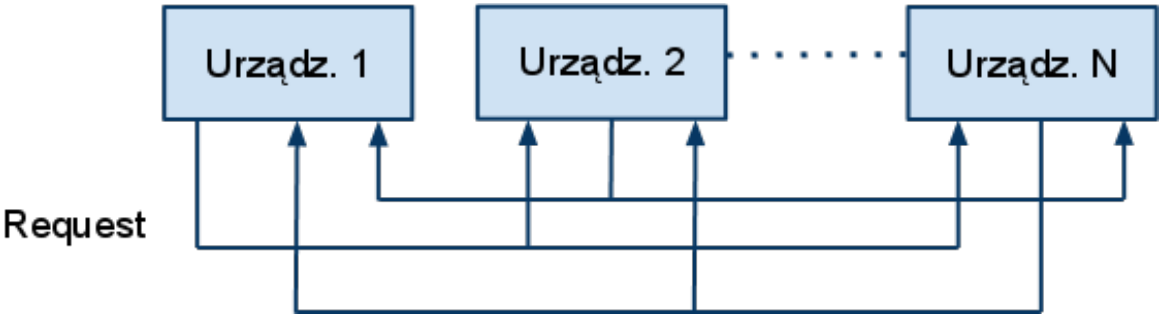
Używany w pierwszej kolejności we wszystkich magistralach *procesor-pamięć*, i bardzo szybkich magistralach wejścia/wyjścia.



Arbiter magistrali - dowolny algorytm wyboru

Arbitraż poprzez wzajemne wykluczanie

Jeżeli magistrala jest wolna to urządzenie chcące mieć dostęp do magistralii umieszcza na magistrali *swój adres (kod identyfikujący)* informując pozostałe urządzenia i przyjęciu uprawnień *master.*

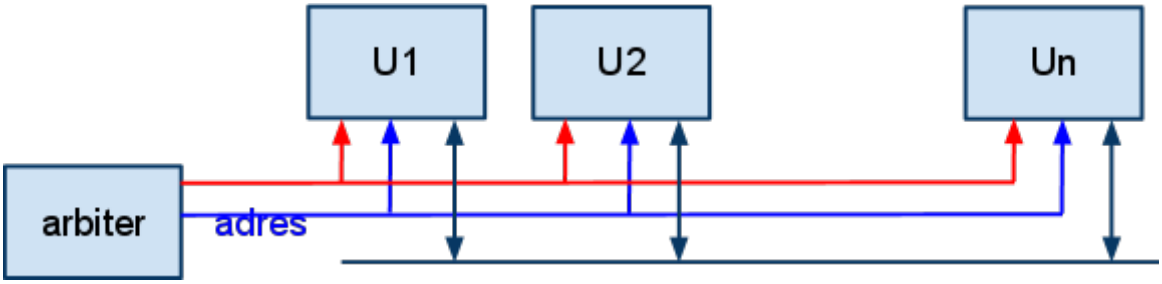


Losowanie (pooling)

Arbiter generuje adresy (kolejno, losowo, z priorytetem).

Urządzenie chcące nadawać czeka na swój adres, wtedy staje się “masterem” i blokuje arbitra.

+ Zalety: łatwość nadawania priorytetów
- Wady: zewnętrzny scentralizowany arbiter



średnio kosztowne, często stosowane