AKADEMIA GÓRNICZO – HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I ELEKTRONIKI

KATEDRA AUTOMATYKI NAPĘDU I URZĄDZEŃ PRZEMYSŁOWYCH

MIKROPROCESOROWE METODY STEROWANIA

Mikrokontrolery rodziny MCS-51

Cz. I.

Autor:

Dr inż. Zbigniew Waradzyn

Kraków 2005

LITERATURA

Literatura podstawowa

[1] Materiały do wykładów z przedmiotu *Technika Impulsowa i Cyfrowa* – część dotycząca mikrokontrolerów rodziny MCS51.

[2] Rydzewski A.: *Mikrokomputery jednoukładowe rodziny MCS-51*. Wydawnictwa Naukowo – Techniczne. Warszawa 1992.

lub

[3] Doliński J.: *Mikrokomputer jednoukładowy Intel 8051*. Wydawnictwo PLJ. Warszawa 1993.

Literatura uzupełniająca

[4] Wybrane artykuły z *Elektroniki dla wszystkich* przygotowane przez wykładowcę.

[5] Dyrcz K. P., Kowalski C. T., Żarczyński Z.: Podstawy techniki mikroprocesorowej. Oficyna Wydawnicza Politechniki Wrocławskiej. Wrocław 1999.

[6] Janiczek J., Stępień A.: Systemy mikroprocesorowe. Mikrokontrolery. Wydawnictwo Centrum Kształcenia Praktycznego. Wrocław 1997.

[7] Janiczek J., Stępień A.: Systemy mikroprocesorowe. Laboratorium systemów mikroprocesorowych cz. I i cz. II. Wydawnictwo Elektronicznych Zakładów Naukowych. Wrocław 1995, 1996.

[8] Gałka P., Gałka P.: Podstawy programowania mikrokontrolera 8051. Mikom. Warszawa 1995.

[9] Starecki T.: Mikrokontrolery 8051 w praktyce. Wydawnictwo BTC. Warszawa 2002.

[10] Majewski J., Kardach K.: Programowanie mikrokontrolerów z serii 8x51 w języku C. Oficyna Wydawnicza Politechniki Wrocławskiej. Wrocław 2002.

[11] Bogusz J.: Programowanie mikrokontrolerów 8051 w języku C w praktyce. Wydawnictwo BTC. Warszawa 2005.

Ze względu na niewielki wymiar godzinowy wykładów materiały do wykładów [1] nie zawierają kompletnych informacji dotyczących mikrokontrolerów. Informacje te są jednak wystarczające do zaliczenia testów, napisania prac kontrolnych oraz zdania egzaminu.

Dodatkowe informacje można znaleźć w pozycjach [2] i [3], które są typu encyklopedycznego oraz w pozycjach [4] ÷ [9], przy czym niektóre zagadnienia dotyczące mikrokontrolerów opisane i wyjaśnione są w prosty i przystępny sposób w pozycji [4].

Pozycje [10] i [11] dotyczą tematyki programowania mikrokontrolerów rodziny 8051 w języku C, która nie będzie poruszana na wykładach.

1. ZAPIS DZIESIĘTNY, DWÓJKOWY (BINARNY) I SZESNASTKOWY (HEKSADECYMALNY)

1.1. Liczby 8-bitowe

W **naturalnym kodzie dwójkowym (binarnym)** za pomocą 8 bitów można przedstawić liczby z zakresu **0 ÷ 255 (0 ÷ 2^8 –1).**

a) zestawienie **– liczby bez znaku w naturalnym kodzie binarnym (dwójkowym)**

waga	dec			bin								hex	
	10^2	10^1	10^0	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	16^1	16^0
	100	10	1	128	64	32	16	8	4	2	1	16	1
wartości			**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
			1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**
			2	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**2**
			3	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**3**
			4	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**4**
	.	.	.	.	.	.	.	.	.	.	.		
			9	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**9**
		1	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**A**
		.	.	.	.	.	.	.	.	.	.	.	.
		1	**5**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**F**
		1	**6**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**
		1	**7**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**1**
		.	.	.	.	.	.	.	.	.	.	.	.
	1	**2**	**7**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**7**	**F**
	1	**2**	**8**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**8**	**0**
	1	**2**	**9**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**8**	**1**
	.	.	.	.	.	.	.	.	.	.	.	.	.
	2	**5**	**4**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**F**	**E**
	2	**5**	**5**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**F**	**F**

Uwaga: **A** = 10, **B**=11, **C**=12, **D**=13, **E**=14, **F**=15

b) sposób oznaczania systemu, w jakim liczba jest przedstawiona:

- **system dziesiętny:** za liczbą podaje się literę ***d***, ***D*** lub nie podaje się żadnej litery, np. *127 = 127d = 127D,*
- **system binarny (dwójkowy):** za liczbą podaje się literę ***b*** lub ***B***, np. 0111 1111***b*** = = 0111 1111***B*** (= *127d*),
- **system heksadecymalny (szesnastkowy):** za liczbą podaje się literę ***h*** lub ***H***, np. 7F***h*** **=** = 7F***H*** (= *127d*),

c) przedstawienie przykładowej liczby z uwzględnieniem wagi poszczególnych cyfr:

* liczba dziesiętna **127*d*** = **1***10^2+**2***10^1+**7***10^0 = 127,
* ***** liczba binarna **0111 1111*b*** = **0***2^7+**1***2^6+**1***2^5+**1***2^4+**1***2^3+**1***2^2+**1***2^1+**1***2^0 = 127,
* liczba szesnastkowa **7F*h*** = **7***16^1+**F***16^0=7*16+15*1 = 127.

d) przekształcenie liczby dwójkowej na szesnastkową

Liczbę w postaci dwójkowej dzielimy na grupy po 4 cyfry każda (zaczynając od końca) i liczbę przedstawianą przez każdą taką grupę zapisujemy w postaci szesnastkowej.

Zadanie: Przedstawić w postaci szesnastkowej liczbę ***01101101b***.

Rozwiązanie: W niniejszym zadaniu są to grupy *0110* i *1101*, przedstawiające liczby odpowiednio *6h* i *Dh*, więc dana liczba ma w postaci szesnastkowej wartość ***6Dh***.

$$\underbrace{0110}_{4+2=6h} \qquad \underbrace{1101}_{8+4+0+1=13=Dh}$$

Gdyby ilość cyfr liczby dwójkowej nie była podzielna przez 4, dopisujemy brakujące zero (lub zera) z **lewej** strony. Przykładowo, gdyby powyższą liczbę przedstawiono jako ***1101101b*** (7 cyfr), to zaczynając od prawej strony otrzymujemy jako drugą grupę cyfr *1101*, zaś pierwszą grupę *110* (3-cyfrową) uzupełniamy zerem na początku otrzymując ***0****110.*

e) przekształcenie liczby szesnastkowej na dwójkową

Każdą cyfrę postaci szesnastkowej zapisujemy w postaci dwójkowej.

Zadanie: Przedstawić w postaci dwójkowej liczbę ***B5h***.

Rozwiązanie: Liczba *Bh* ma postać dwójkową *1011b* (11=8+2+1), zaś liczba 5 – postać *0101b* (5=4+1). Zatem dwójkowa postać zadanej liczby to ***1011 0101b***.

f) przekształcenie liczby dziesiętnej na binarną i szesnastkową.

Zadaną liczbę dziesiętną dzielimy przez 16 (waga starszej cyfry postaci szesnastkowej) otrzymując starszą cyfrę szukanej liczby szesnastkowej. Młodszą cyfrę szukanej liczby stanowi reszta z wykonanego dzielenia.

Zadanie: zapisać liczbę ***105*** w systemie szesnastkowym i dwójkowym.

Rozwiązanie.: 105/16 = 6 reszta 9 (105=6 ∗16+9), czyli 105d = ***69h*** = ***0110 1001b.***

1.2. Liczby 16-bitowe

W **naturalnym kodzie dwójkowym (binarnym)** za pomocą 16 bitów można przedstawić liczby z zakresu **0 ÷ 65535 (0 ÷ 2^{16}-1)**

dec	bin	hex	
0	0000 0000 0000 0000	0000	
255	0000 0000 1111 1111	**00FF**	*256 B*
256	0000 0001 0000 0000	0100	
1023	0000 0011 1111 1111	**03FF**	*1 KB*
1024	0000 0100 0000 0000	0400	
4096	0001 0000 0000 0000	1000	
8191	0001 1111 1111 1111	**1FFF**	*8 KB*
32767	0111 1111 1111 1111	**7FFF**	*32 KB*
32768	1000 0000 0000 0000	8000	
61440	1111 0000 0000 0000	F000	
65535	1111 1111 1111 1111	**FFFF**	*64 KB*

1 KB = 1024 B

Wagi kolejnych cyfr liczb zapisanych w postaci **binarnej**:

2^{15}, 2^{14}, 2^{13}, 2^{12}, 2^{11}, 2^{10}, 2^{9}, 2^{8}, 2^{7}, ... , 2^{1}, 2^{0}

32768, 16384, 8192, 4096, 2048, 1024, 512, 256, 128, ... , 2, 1

Wagi kolejnych cyfr liczb zapisanych w postaci **szesnastkowej**:

16^{3}, 16^{2}, 16^{1}, 16^{0}

4096, **256**, **16**, **1**

Zadanie: Zapisać liczbę ***50 001*** w systemie szesnastkowym.

Rozwiązanie: Z zakresu liczby widać, że do jej zapisu trzeba wykorzystać 16 bitów.

a) najpierw dzielimy zadaną liczbę przez **4096** (waga najstarszej cyfry postaci szesnastkowej) otrzymując *50001:4096 = **12** reszta 849*. Otrzymany wynik z dzielenia to najstarsza cyfra postaci szesnastkowej, czyli jest to ***Ch***,

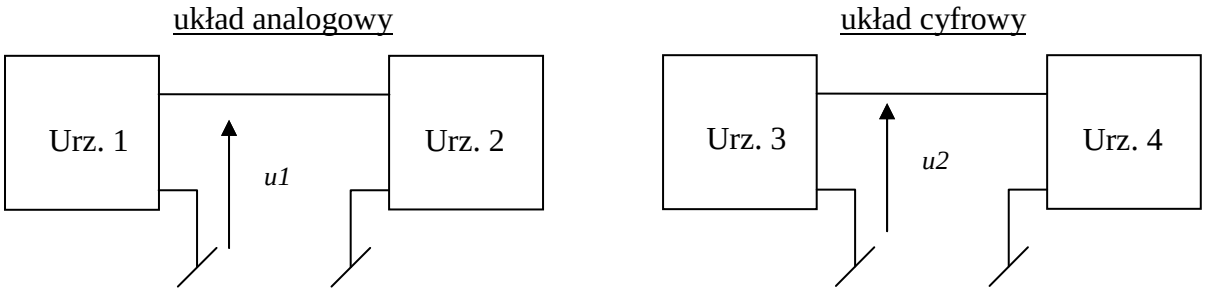
b) resztę otrzymaną z powyższego dzielenia dzielimy przez **256**, czyli wagę kolejnej cyfry postaci szesnastkowej. Otrzymujemy *849:256 = **3** reszta 81*. Kolejną cyfrą postaci szesnastkowej jest więc ***3h***,

c) otrzymaną resztę dzielimy przez **16**, czyli wagę następnej cyfry postaci szesnastkowej, co daje *81:16 = **5** reszta **1***. Kolejnymi cyframi postaci szesnastkowej są więc ***5h*** i ***1h***.

Odpowiedź: 50001d = ***C351h***.

2. UKŁADY ANALOGOWE A UKŁADY CYFROWE

2.1. Porównanie układów analogowych i cyfrowych



Rys. 2.1. Układ analogowy i układ cyfrowy

Rysunek 2.1 przedstawia szkic układu analogowego i układu cyfrowego. Załóżmy, że maksymalna wartość zarówno sygnału *u1* (układ analogowy), jak i sygnału *u2* (układ cyfrowy) wynosi 5V.

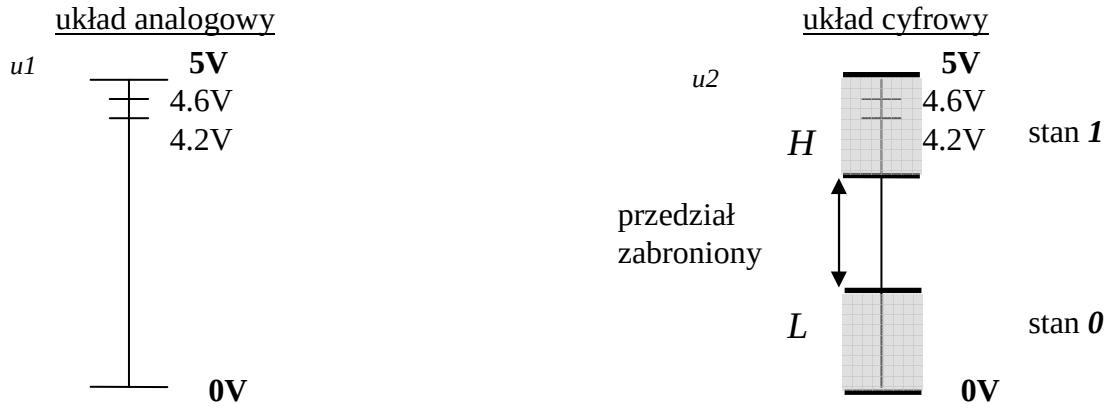
Pytanie: Ile różnych informacji można przekazać w układzie analogowym, a ile w cyf□owym?

Odp.:

a) w układzie analogowym każda wartość napięcia w zakresie 0 ÷ 5V to inna informacja; można więc przesłać z urządzenia 1 do urządzenia 2 nieskończenie wiele różnych informacji.

b) w układzie cyfrowym rozróżnia się tylko dwa stany określane jako ***0*** i ***1***. Każdemu stanowi odpowiada pewien przedział napięć. Przy przyjęciu tzw. **logiki dodatniej** stanowi ***0*** odpowiada niski poziom napięcia (oznaczany przez ***L***), zaś stanowi ***1*** – wysoki poziom napięcia (oznaczany przez ***H***). Z urządzenia 3 można więc przesłać do urządzenia 4 tylko **dwie** różne **informacje**, np. stan *0* – zimno, stan *1* – ciepło.

Przykład:



Rys. 2.2. Przykładowe poziomy napięć w układzie analogowym i cyfrowym

Podane na rysunku 2.2 przykładowe poziomy napięć 4.2V i 4.6V są w układzie analogowym traktowane jako dwie różne wartości, natomiast w układzie cyfrowym są odczytywane jako ten sam stan *H*. Jeśli na skutek zakłócenia odczytano np. 4.6V zamiast np. 4.2V, to zakłócenie to wprowadza błąd tylko do układu analogowego – w układzie cyfrowym przekaz informacji jest

nadal prawidłowy – przecież oba poziomy napięć oznaczają ten sam stan *H*. Układy cyfrowe charakteryzują się więc **dużą odpornością na zakłócenia** i jest to ich podstawowa zaleta.

Należy jeszcze podkreślić, że w układzie analogowym napięcie może przyjmować dowolną wartość z zakresu pomiędzy wartością minimalną a maksymalną (w powyższym przykładzie będzie to dowolna wartość z zakresu 0 ÷ 5V), zaś w układzie cyfrowym istnieje pewien zabroniony przedział napięć. Gdyby napięcie *u2* (rys. 2.1 i 2.2) przyjęło wartość z tego przedziału, urządzenie 4 „nie wiedziałoby", czy to jest stan *0*, czy też stan *1*. W standardzie TTL ten zabroniony przedział napięć to 0.8 ÷ 2.0 V.

2.2. Ilość informacji przekazywana w układzie cyfrowym

W poprzednim punkcie stwierdzono, że przy zastosowaniu jednej linii sygnałowej (jednego przewodu – pomijając przewód masy) można przesłać w systemie cyfrowym jedynie dwie różne informacje: *0* lub *1*.

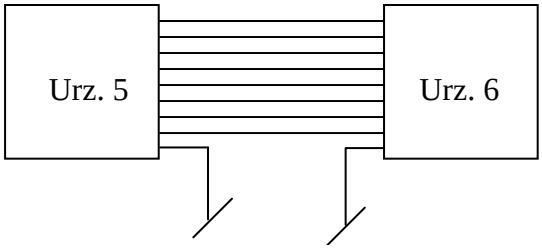
Pytanie: Co należy zrobić, aby w układzie cyfrowym można było przekazać więcej różnych informacji?

Odp.: Należy użyć większej ilości **linii sygnałowych** (przewodów, ścieżek na płytce drukowanej, itp.). Poniższa tabela przedstawia ilość różnych informacji, jaką można przesłać w systemie cyfrowym w zależności od ilości linii sygnałowych.

Ilość linii sygnałowych	**Ilość różnych informacji**	**Wartości [bin]**	**Zakres * [dec]**
1	$2 = 2^1$	0, 1	$0 \div 1$ $(0 \div 2^1-1)$
2	$4 = 2^2$	00, 01, 10, 11	$0 \div 3$ $(0 \div 2^2-1)$
3	$8 = 2^3$	000, 001, 010, 011, ..., 111	$0 \div 7$ $(0 \div 2^3-1)$
8	**256** $= 2^8$	**0000 0000, 0000 0001, ... , ... , 1111 1110, 1111 1111**	$0 \div 255$ $(0 \div 2^8-1)$
16	**65536** $= 2^{16}$	**0000 0000 0000 0000, ... , ... , 1111 1111 1111 1111**	$0 \div 65535$ $(0 \div 2^{16}-1)$
32	2^{32}	32 pozycje	$0 \div 2^{32}-1$
64	2^{64}	64 pozycje	$0 \div 2^{64}-1$
n	2^n	*n* pozycji	$0 \div 2^n-1$

Przy użyciu 8 linii sygnałowych (nie licząc przewodu masowego) uzyskujemy **8-bitową magistralę** (**szynę**) umożliwiającą przesłanie 256 różnych informacji (którym można przyporządkować liczby od 0 do 255). Rysunek 2.3 przedstawia szkic połączenia dwóch urządzeń (urządzenia 5 z urządzeniem 6) przy zastosowaniu magistrali **ośmiobitowej**.

* *Uwaga:* Zakres podano przy założeniu, że stosujemy naturalny kod dwójkowy (binarny)

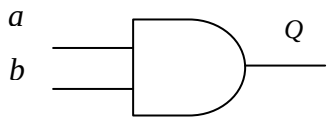


Rys. 2.3. Magistrala (szyna) ośmiobitowa

Natomiast przy użyciu 16 linii sygnałowych uzyskujemy **16-bitową** magistralę (szynę) umożliwiającą przesłanie 65536 $\left(2^{16}\right)$ różnych informacji (którym można przyporządkować liczby od 0 do 65535).

2.3. Rodzaje układów cyfrowych:

a) **układy kombinacyjne** – aktualny stan wyjścia Q w danej chwili zależy tylko od kombinacji stanów wejść $a, b, c, ...$ w tejże chwili (nie zależy od stanu w chwili poprzedniej), czyli $Q = f(a,b,c,...)$, np. $Q = a \cdot b$. Taką funkcję realizuje bramka AND.

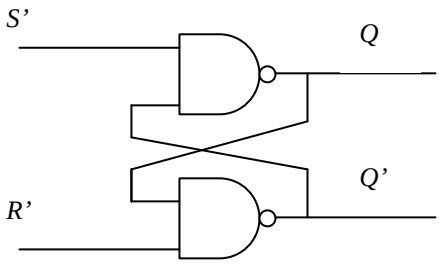


Rys. 2.4. Bramka AND oraz tabela jej stanów

Tabela stanów bramki AND		
a	***b***	***Q***
0	0	0
0	1	0
1	0	0
1	1	1

Jak widać z powyższej tabeli stanów, każdej kombinacji wejść odpowiada ściśle określony stan wyjścia.

b) **układy sekwencyjne** – aktualny stan wyjścia Q_n zależy nie tylko od kombinacji stanów wejść, ale także od stanu wyjścia Q_{n-1} poprzedzającego stan aktualny, czyli $Q_n = f(a,b,c,...,Q_{n-1})$. Układy sekwencyjne nazywane są też **układami z pamięcią**, najprostsze z nich to **przerzutniki**. Przykładem jest przerzutnik $S'R'$.



Rys. 2.5. Przerzutnik S'R' oraz tabela jego stanów

Tabela stanów przerzutnika *S'R'*		
S'	***R'***	Q_n
0	0	-
0	1	1
1	0	0
1	1	Q_{n-1}

stan zabroniony

Jeżeli na wejściach *S'* i *R'* jest stan 1, to aktualny stan wyjścia Q_n jest równy stanowi poprzedniemu Q_{n-1}:

a) jeśli poprzednio było $S' = 0$ i $R' = 1$, to $Q_n = 1$,

b) jeśli poprzednio było $S' = 1$ i $R' = 0$, to $Q_n = 0$.

3. PAMIĘĆ

a) podstawowa jednostka przechowywania informacji w systemie cyfrowym to **bit**,

b) bit może przyjmować jedną z dwu wartości: *0* (*L*) lub *1* (*H*),

c) **bajt** - *uporządkowane* 8 bitów; *jednostka pochodna*; symbolem bajta jest duża litera **B**,

d) do przechowywania informacji służą **rejestry** zbudowane z przerzutników:

- każdy przerzutnik zapamiętuje jeden bit (na jego wyjściu może być stan *0* lub *1*),
- rejestr **8-bitowy** pozwala zapamiętać 1 bajt,

e) **pamięć** - zestaw ponumerowanych rejestrów, najczęściej 8-bitowych:

- te rejestry to **komórki pamięci**,
- numer każdego rejestru to **adres** – musi on być podany przy każdej operacji odczytu pamięci lub zapisu do niej,

f) każda pamięć wymaga odpowiedniej ilości linii:

- **szyna danych** (8 linii dla pamięci o rejestrach 8-bitowych),
- **szyna adresowa** (aby zapewnić możliwość zaadresowania każdej z komórek),
- **szyna sterująca** – dodatkowe sygnały wymagane do obsługi pamięci, czyli odczytu i ewentualnie zapisu do pamięci,

g) **pojemność pamięci** - ilość dostępnych komórek pamięci; zależy od rodzaju pamięci (konkretnego układu scalonego),

h) jednostki pojemności pamięci: **bit**, **bajt (B), kilobajt (KB)**;

- 1 KB = 2^{10} B = 1024 B,

i) Zależność pomiędzy **pojemnością** pamięci a ilością **linii szyny** adresowej:

1 linia → 2^1 *= 2 komórki adresowalne,*

2 linie → 2^2 *= 4 komórki adresowalne,*

8 linii → 2^8 *= 256 komórek adresowalnych,* *13 linii* → 2^{13} *= 8192 komórki (8 KB),*

10 linii → 2^{10} *= 1024 komórki (1 KB),* *16 linii* → 2^{16} *= 65536 komórek (64 KB).*

j) przy odwoływaniu się do pamięci stosuje się zapis **szesnastkowy** (**heksadecymalny**),

k) rodzaje pamięci

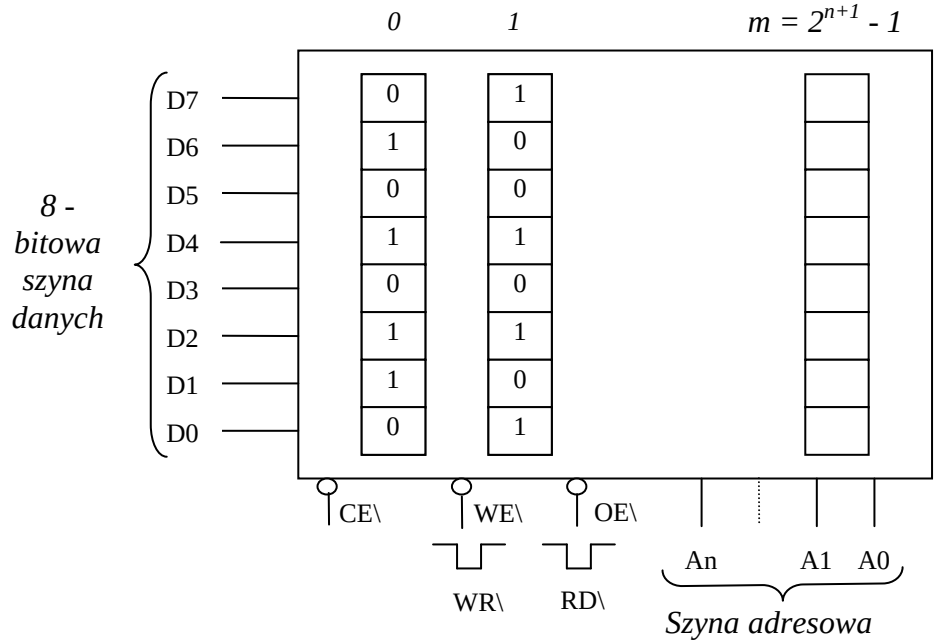
- **RAM** (ang. *Random Access Memory*) – pamięć o dostępie swobodnym (zapisywalna), możliwość zapisu i odczytu, np. pamięć operacyjna IBM PC; dane są tracone po wyłączeniu napięcia zasilającego; często stosuje się **baterijne podtrzymanie** pamięci RAM; tego typu pamięć często jest stosowana jako **pamięć danych**,
- **ROM** (ang. *Read Only Memory*) - pamięć stała, nie jest możliwa zmiana jej zawartości, ani jej skasowanie,

- **EPROM** (ang. *Erasable and Programmable Read Only Memory*) – pamięć reprogramowalna, kasuje się ją promieniami UV, programuje elektrycznie,
- **EEPROM** (ang. *Electrically Erasable and Programmable Read Only Memory*) – pamięć reprogramowalna, kasuje się ją i programuje elektrycznie,
- **Flash** - pamięć reprogramowalna, kasuje się ją i programuje elektrycznie.

l) przykłady pamięci:

- **27C64** - 8 KB EPROM (8192 słowa x 8 bitów),
- **27C512** - 64 KB EPROM (65536 słów x 8 bitów),
- **6116** - 2 KB RAM (2048 słów x 8 bitów),
- **6264** - 8 KB RAM (8192 słowa x 8 bitów).

Na rysunku 3.1 przedstawiono szkic typowej 8-bitowej **pamięci RAM**, zbudowanej jako pojedynczy układ scalony. Tego typu pamięć może pracować jako jedno z urządzeń w systemie mikroprocesorowym (rys. 5.1), pełniąc funkcję zewnętrznej pamięci danych.



Rys. 3.1. Szkic typowej 8-bitowej pamięci RAM

Poniżej podano podstawowe informacje o przedstawionej pamięci RAM:

a) pamięć ta zawiera 8-bitowe rejestry o adresach od *0* do *m*,

b) przykładowo pokazano, że komórka o adresie *0* zawiera wartość *01010110b*, a komórka o adresie *1* - wartość *10010101b*,

c) pamięć jest dołączana do **systemu mikroprocesorowego** przez szynę (magistralę) systemową zawierającą szynę danych, szynę adresową oraz szynę sygnałów sterujących (rys. 5.1),

d) aby odczytać zawartość którejkolwiek komórki pamięci lub wpisać do niej nową wartość, **adres** tej komórki musi pojawić się **na szynie adresowej** (tutaj wyprowadzenia *An, ..., A1* i *A0*). Adres jest wystawiany przez mikroprocesor (mikrokontroler),

e) **dane** mogą być przesyłane pomiędzy pamięcią RAM a mikroprocesorem (w dowolną stronę) poprzez 8-bitową **szynę danych** – tutaj wyprowadzenia *D7, ..., D0*:

- wartość wystawiona przez mikroprocesor na szynę danych zostaje z niej **przepisana** do zaadresowanej komórki pamięci po pojawieniu się na **wejściu *WE*** (ang. *write enable*) układu pamięci impulsu *L* (na to wejście podany jest sygnał $\overline{WR}$ z mikroprocesora);
- po pojawieniu się na **wejściu *OE*** (ang. *output enable*) układu pamięci impulsu *L* (na to wejście podany jest sygnał $\overline{RD}$ z mikroprocesora) zawartość zaadresowanej komórki pamięci zostaje **wystawiona przez pamięć na szynę danych**, a następnie przepisana do mikroprocesora,

f) dodatkowym warunkiem odczytu lub zapisu danej z/do pamięci jest jej **uaktywnienie**, co uzyskuje się prze podanie stanu *L* na **wejście wybierające *CE*** (ang. *chip enable*) pamięci. Jeśli w systemie jest więcej układów pamięci, to należy zadbać o to, aby, celem uniknięcia konfliktów, przy każdej operacji zapisu lub odczytu był uaktywniony tylko jeden układ pamięci.

Uwaga1: Ukośnik wsteczny (*ang. backslash*) „\" podany bezpośrednio za nazwą sygnału, np. WR\, lub kreska nad nazwą sygnału, np. $\overline{WR}$, oznacza, że sygnał jest **aktywny,** gdy przyjmuje **stan 0.** W powyższym przypadku oznacza to, że gdy ma być dokonany zapis do pamięci, sygnał $\overline{WR}$ przyjmuje stan 0 i ma na przykład kształt impulsu  .

Uwaga2: Jak wynika z powyższego, stosowane w tych materiałach zapisy typu $\overline{WR}$ i *WR*\ oraz np. $\overline{WE}$ i *WE*\ są równoważne.

Uwaga3: Niektóre moduły pamięci mogą mieć nieco inny układ wejść służących do obsługi operacji zapisu i odczytu, niż to przedstawiono na rys. 3.1.

4. MIKROPROCESOR

Mikroprocesor jest to ***cyfrowy układ scalony*** (ang. *chip*) o wielkim stopniu scalenia, przeznaczony do bardzo szybkiego wykonywania dowolnego ciągu prostych operacji wybieranych spośród ustalonego zbioru operacji podstawowych (arytmetycznych i logicznych).

Działanie mikroprocesora jest sterowane pobieranymi z zewnątrz **rozkazami** (zgodnie z programem napisanym przez użytkownika). To właśnie **odróżnia** mikroprocesor od innych układów cyfrowych.

Pierwszy mikroprocesor powstał w firmie INTEL w roku 1971, został oznaczony symbolem 4004, składał się z 2300 tranzystorów i był 4-bitowy.

W budowie mikroprocesorów wytwarzanych przez różnych producentów występują znaczne różnice. Jednakże można wyróżnić pewne bloki i cechy wspólne dla wszystkich mikroprocesorów, w tym także stosowanych w komputerach klasy IBM PC.

Zasadnicze układy mikroprocesora to:

a) jednostka arytmetyczno-logiczna (ang. *ALU – Arithmetic-Logic Unit*),
b) zespół rejestrów uniwersalnych (ang. *register bank*),
c) układ sterujący (ang. *control circuit*).

Najważniejsze cechy mikroprocesora i **pojęcia** z nim związane:

a) **rejestry** to specjalne komórki pamięci:
   * służą do chwilowego przechowywania informacji,
   * są wykorzystane przy wykonywaniu różnych operacji arytmetycznych i logicznych.
b) **jednostka arytmetyczno-logiczna (***ALU***)** realizuje przetwarzanie danych zawartych w rejestrach wewnętrznych mikroprocesora lub komórkach pamięci zewnętrznej,
c) **układ sterujący** steruje przepływem danych między rejestrami, pamięcią i układem arytmetyczno-logicznym, decydując o tym, jakie operacje i na jakich danych mają być wykonane,
d) działanie procesora jest **cykliczne** – w kolejnych cyklach pracy (cyklach rozkazowych) pobierane są kolejne rozkazy,
e) ciąg rozkazów realizowanych przez mikroprocesor to **program** - zawarty w **pamięci programu**,
f) rozkazy pobierane są *zasadniczo* po kolei:
   * adres komórki, z której należy pobrać kolejny rozkaz znajduje się w specjalnym rejestrze mikroprocesora - liczniku rozkazów,
   * po pobraniu rozkazu stan licznika rozkazów jest zwiększany automatycznie,

* jednakże istnieją też rozkazy skoków, umożliwiające pobór kolejnej instrukcji z innego miejsca w pamięci programu,

g) realizacja programu zaczyna się od odczytu pierwszej instrukcji ze ściśle określonego adresu pamięci programu:
   * po załączeniu napięcia zasilającego,
   * po zresetowaniu (wyzerowaniu) mikroprocesora, co jest realizowane przez podanie odpowiedniego sygnału na wejście RESET,

h) zarówno rozkazy, jak i dane są zapisywane w identycznej postaci słów zerojedynkowych,

i) liczba bitów w słowie mikroprocesora (długość słowa) jest równa liczbie linii jego magistrali danych. W różnych mikroprocesorach stosuje się różną długość słowa; zwykle jest to wielokrotność 8 bitów (stosuje się słowa 8-bitowe, 16-, 32-, 64-bitowe, itd.),
   * w szczególności, jeśli podaje się, że mikroprocesor jest 8-bitowy, oznacza to, że operacje wykonywane są na danych 8-bitowych, czyli długość słowa wynosi 8 bitów,

j) pobieranie rozkazów sterowane jest generatorem impulsów taktujących (zegarowych) (rys. 8.4) o stałej częstotliwości, zwykle stabilizowanym za pomocą rezonatora kwarcowego popularnie nazywanego „kwarcem” (rys. 8.2 a),

k) **cykl maszynowy** - powtarzająca się sekwencja przebiegów - zwykle kilka lub kilkanaście okresów sygnału taktującego (zegarowego). Najprostsze rozkazy wykonywane są w jednym cyklu maszynowym, zaś rozkazy bardziej złożone mogą wymagać kilku cykli (rys. 8.4),

l) **przerwanie** - chwilowe zawieszenie wykonywania bieżącego programu i skok w inne miejsce programu, celem wykonania odrębnego fragmentu programu; po jego zakończeniu następuje powrót do fragmentu programu wykonywanego przed pojawieniem się przerwania.

5. SYSTEM MIKROPROCESOROWY: mikroprocesor + elementy zewnętrzne

Sam mikroprocesor nie jest zdolny do samodzielnego funkcjonowania. Do jego prawidłowej pracy potrzebne są dwa typy układów dodatkowych:

- układy do wprowadzania i wyprowadzania informacji, zwane **układami wejścia–wyjścia**, w skrócie **We/Wy** (ang. *input-output*),
- **pamięć** (ang. *memory*), w której jest przechowywany program oraz dane i wyniki obliczeń, zarówno pośrednie, jak i końcowe.

Mikroprocesor jako **jednostka centralna** (ang. *CPU – Central Processing Unit*) wraz z zestawem układów dodatkowych tworzy **system mikroprocesorowy**, zwany również **mikrokomputerem** (**komputerem**).

Do mikroprocesora zwykle podłączone są bezpośrednio:

- **generator** impulsów zegarowych (często tylko rezonator kwarcowy),
- układ do wytwarzania sygnału **RESET** (do zerowania mikroprocesora).

Pozostałe elementy zewnętrzne dołączone są zwykle przez **szynę (magistralę) systemową**, w skład której wchodzą:

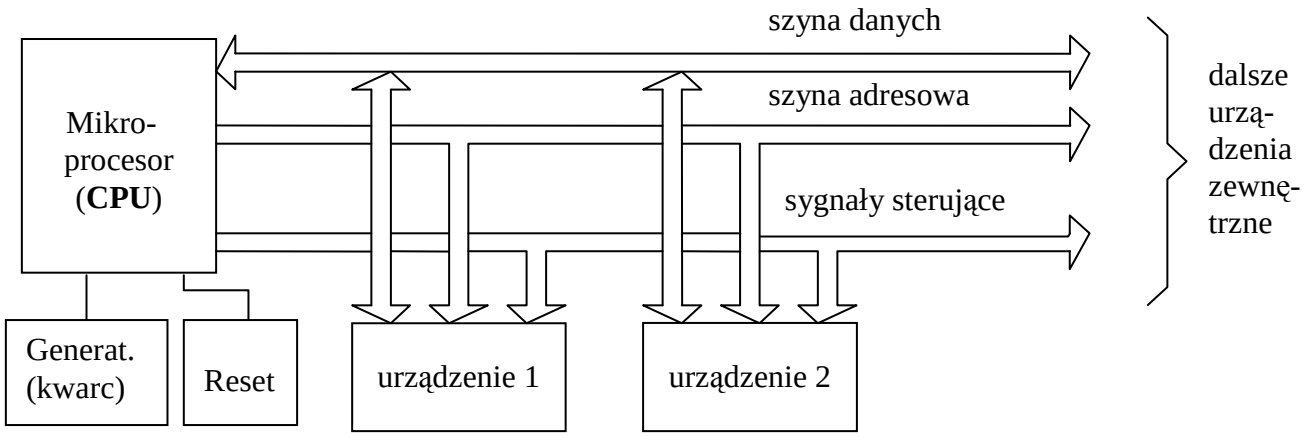
- magistrala **danych** (dwukierunkowa), po której są przesyłane informacje (kody rozkazów i dane) między mikroprocesorem a pozostałymi urządzeniami, oznaczona np. *D0...D7*,
- magistrala **adresowa** (jednokierunkowa), po której mikroprocesor wysyła adres pamięci lub urządzenia wejścia-wyjścia, oznaczona np. *A0...A15*,
- magistrala **sterująca** (jednokierunkowa), po której mikroprocesor przesyła sygnały określające rodzaj operacji, jaką ma wykonać układ współpracujący, np. $\overline{RD}$ - odczyt zewnętrznej pamięci, $\overline{WR}$ - zapis do zewnętrznej pamięci.

Szyna (magistrala) - zespół linii, którymi są przesyłane sygnały. Interpretacja fizyczna:

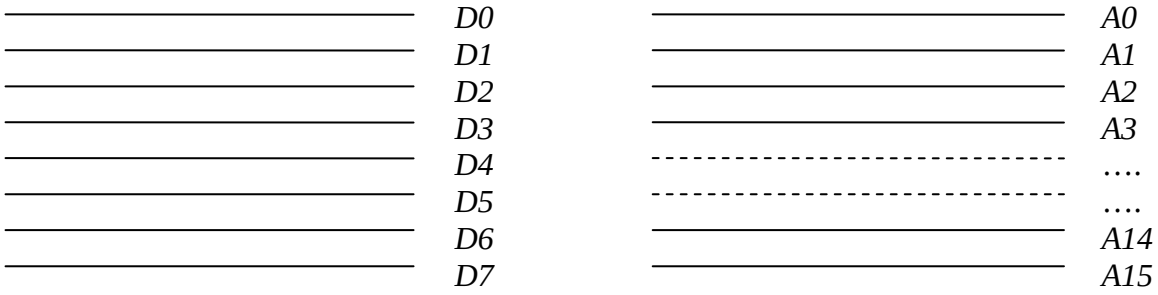
* mogą to być przewody,
* na płytce są to ścieżki,
* przewód masowy nie jest tu uwzględniany.

Do szyny systemowej dołącza się zwykle następujące urządzenia:

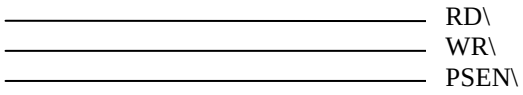
a) pamięć programu,
b) pamięć danych,
c) układy licznikowe czasowe,
d) porty wejść / wyjść,
e) sterowniki transmisji danych,
f) przetworniki A/C,



a) schemat systemu mikroprocesorowego



b) 8 - bitowa szyna danych *c) 16-bitowa szyna adresowa*



d) przykładowa szyna sygnałów sterujących

Rys. 5.1. System mikroprocesorowy

g) przetworniki C/A,

h) zegary czasu rzeczywistego,

i) wyświetlacze.

Do systemu mikroprocesorowego mogą być również dołączane **inne** elementy **nie** poprzez szynę systemową, np.:

a) klawiatura,

b) wyświetlacze – innego typu niż podane wyżej,

c) różnego typu czujniki.

6. MIKROKONTROLER – informacje ogólne

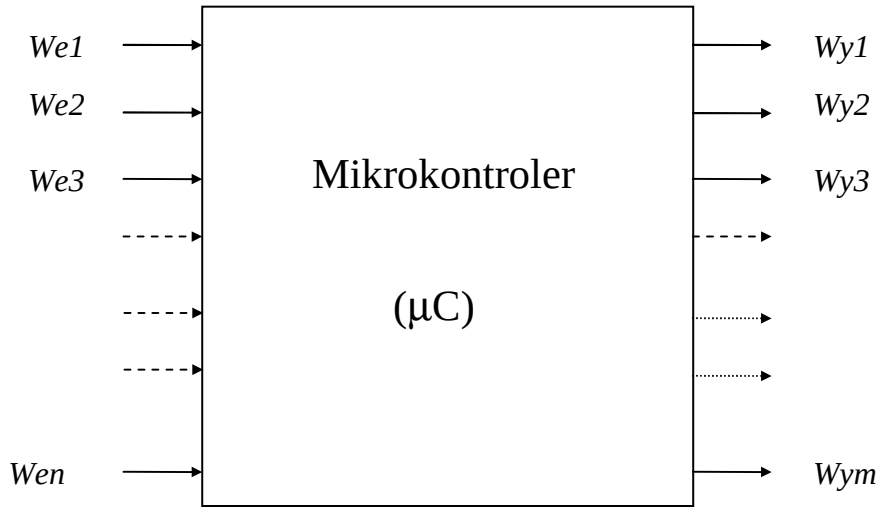
Urządzeniem, które wykonuje funkcje przedstawione podczas demonstracji i które będzie **przedmiotem** wykładu, jest *mikrokontroler*.

Mikrokontroler to **cyfrowy układ scalony** zawierający w jednej obudowie mikroprocesor (podstawowa część) oraz elementy zewnętrzne w stosunku do mikroprocesora (rys. 6.1).

W efekcie pozostaje niewiele elementów zewnętrznych, np. rezonator i układ RESET, co zostanie omówione bardziej szczegółowo w punkcie 8.

Inna nazwa mikrokontrolera to **komputer jednoukładowy.**

Z punktu widzenia użytkownika istotne są sygnały wejściowe podane na urządzenie oraz uzyskanie odpowiednich sygnałów na wyjściach. Mikrokontroler możemy więc przedstawić w uproszczeniu jako **„czarną skrzynkę”** z wejściami i wyjściami (rys. 6.2).

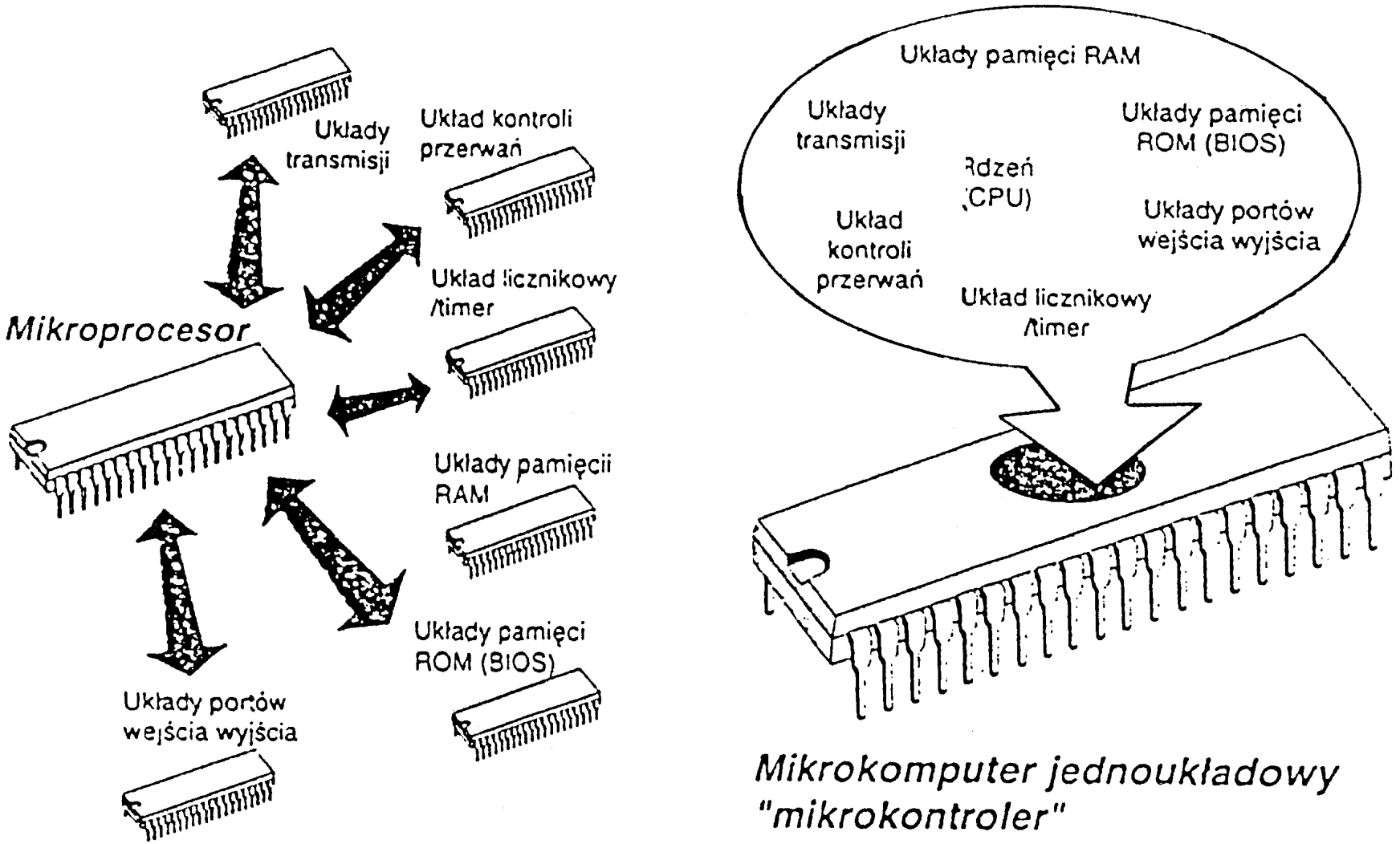


Rys. 6.2. Mikrokontroler jako „czarna skrzynka”

Stan każdego wyjścia może zależeć od stanu wejść i wyjść w chwili bieżącej, ale także i w chwilach wcześniejszych, czyli inaczej mówiąc może zależeć także od czasu. Stan dowolnego wyjścia (którego numer oznaczono tu indeksem *i*) można zapisać następująco:

$$Wyi = f_i(We1, We2, \ldots, Wen, Wy1, Wy2, \ldots, Wym, t) \quad i = 1, 2, \ldots, m$$

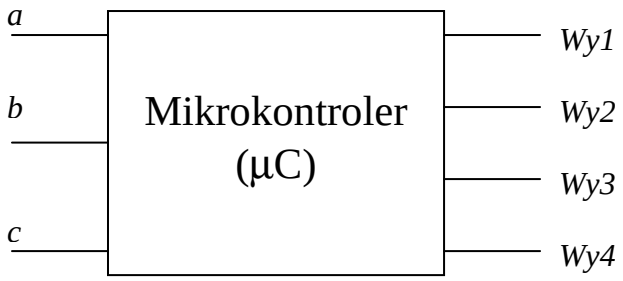
Funkcje opisujące stany poszczególnych wyjść zapisane są w odpowiedni sposób w **programie**, zaś program – to **ciąg instrukcji** wykonywanych przez mikrokontroler. Ponieważ stan dowolnego wyjścia może zależeć od czasu, może on więc być zależny także od stanu tego samego wyjścia w chwili poprzedniej. Wynika z tego, że mikrokontroler umożliwia realizację funkcji wykonywanych zarówno przez układy kombinacyjne, jak i układy sekwencyjne – porównaj p. 2.3.



Rys. 6.1. Mikroprocesor do pracy potrzebuje wielu dodatkowych układów peryferyjnych, a typowy "mikrokontroler jednoukładowy" ma je wbudowane [4]

Przykład

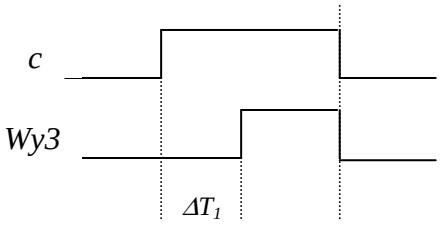
Na rysunku 6.3 przedstawiono przykładowy schemat układu z mikrokontrolerem jako „czarną skrzynką" z trzema wejściami *a, b, c* oraz czterema wyjściami *Wy1, Wy2, Wy3* i *Wy4*.



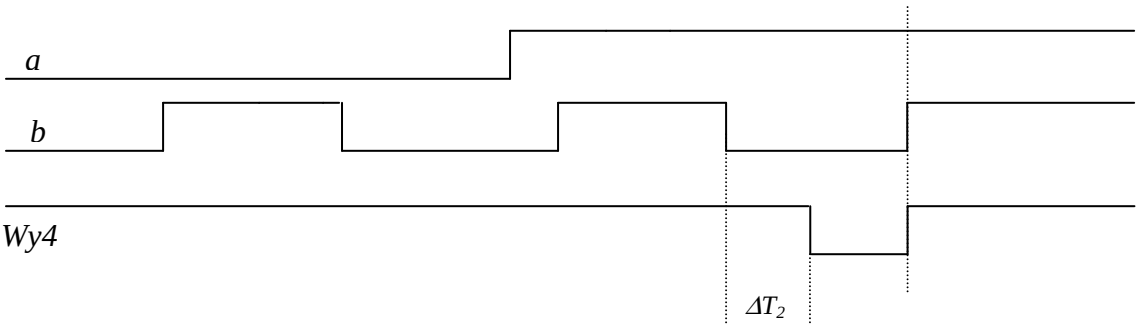
Rys. 6.3. Przykładowy schemat układu z mikrokontrolerem jako „czarną skrzynką"

A oto *przykładowe funkcje* realizowane przez poszczególne wyjścia:

- *Wy1* realizuje iloczyn logiczny: $Wy1 = a \cdot b \cdot c$ (*Wy1* przyjmuje stan *1* tylko wtedy, gdy wszystkie trzy wejścia są w stanie *1*),
- *Wy2* realizuje negację sumy logicznej wejść *a* oraz *c*: $Wy2 = \overline{a + b}$ (*Wy2* przyjmuje stan *1* tylko wtedy, gdy oba wejścia są w stanie *0*),
- *Wy3*:
  - przyjmuje stan 0, jeśli wejście *c* ma stan *0*,
  - zmienia swój stan z *0* na *1* z opóźnieniem ΔT_1 po takiej samej zmianie wejścia *c*, co przedstawiono na rysunku obok,



- *Wy4* zmienia swój stan z *1* na *0* z opóźnieniem ΔT_2 po takiej samej zmianie wejścia *b*, pod warunkiem, że wejście *a* jest w stanie *1*; w przeciwnym razie stan *Wy4* to *1*. Zależność stanu *Wy4* od stanu wejść *a* oraz *b* przedstawia rysunek poniżej.



Są to tylko proste przykłady funkcji, jakie można zrealizować przy wykorzystaniu mikrokontrolera. W praktyce można użyć większej ilości zarówno wejść, jak i wyjść oraz realizować bardziej złożone zależności między wyjściami a wejściami.

Mikrokontrolery umożliwiają budowę kompletnych sterowników mikroprocesorowych, gdzie wszystkie **funkcje kontrolne** spełnia jeden układ scalony. W ten sposób mogą zastąpić nawet złożone układy elektroniczne zbudowane z elementów dyskretnych.

Niektóre **cechy** i **możliwości** mikrokontrolera:

a) przeznaczony głównie do sterowania,
b) posiada wejścia \ wyjścia cyfrowe,
c) może także posiadać wejścia analogowe,
d) możliwość generacji opóźnień czasowych,
e) możliwość dokonywania obliczeń arytmetycznych,
f) możliwość współpracy z zewnętrznymi pamięciami programu i danych,
g) możliwość współpracy z innymi urządzeniami zewnętrznymi, np. przetwornikami A/C i C/A,
h) możliwość dokonywania pomiarów temperatury lub innych wielkości fizycznych przy współpracy z odpowiednimi czujnikami,
i) możliwość wymiany informacji z innymi urządzeniami przez standardowe złącze szeregowe.

Podstawowe zalety mikrokontrolera:

- duża elastyczność: ten sam sprzęt może wykonywać bardzo różne zadania,
- łatwość wprowadzania zmian i poprawek przy uruchamianiu programu,
- wysoka niezawodność wynikająca z niewielkiej ilości elementów w układzie,
- małe wymiary systemu,
- niskie koszty projektowania i realizacji systemów użytkowych.

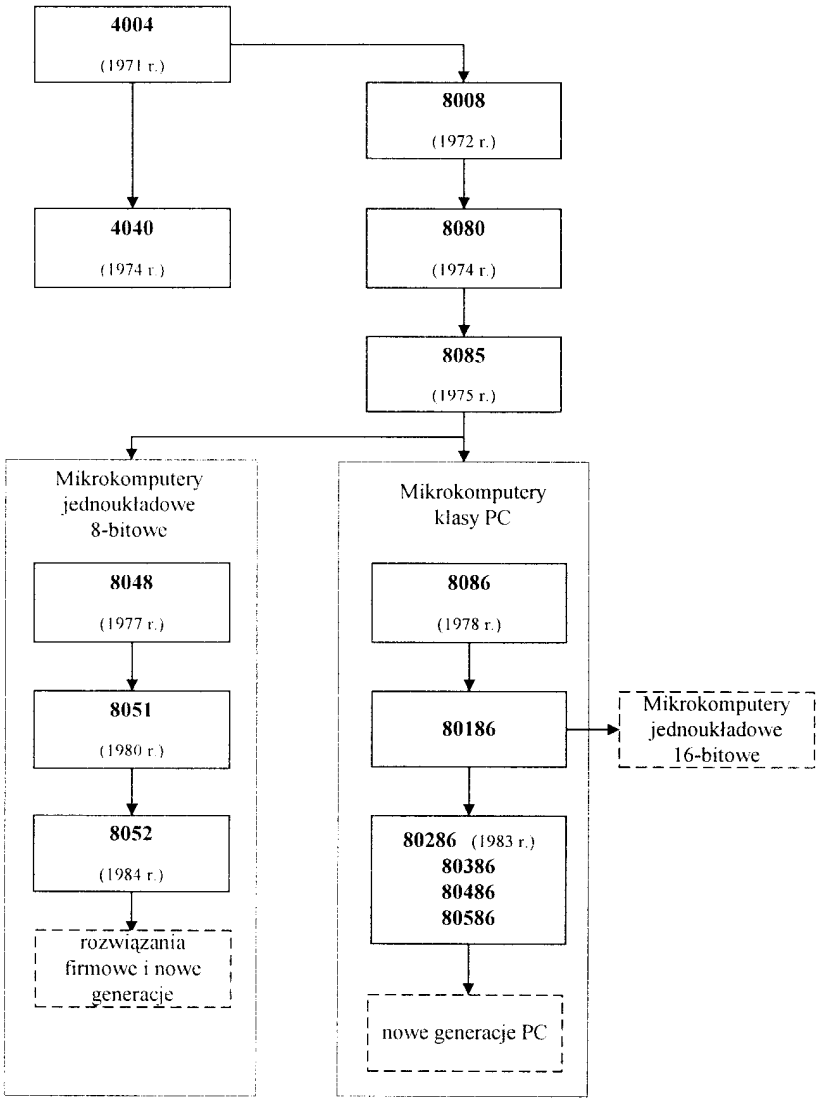
Niektóre zastosowania mikrokontrolerów:

a) sprzęt powszechnego użytku, np. odbiorniki radiowe i telewizyjne:
   - programowanie funkcji, cyfrowe sterowanie parametrami odbiorników, telegazeta, itp.,
b) motoryzacja
   - sterowanie silników, klimatyzacja, ABS, itp.,
c) aparaty fotograficzne i kamery video,
d) artykuły gospodarstwa domowego:
   - sterowanie pracą pralek automatycznych, itp.,
e) rejestratory danych w różnych urządzeniach,
f) wiele innych.

7. NIEKTÓRZY PRZEDSTAWICIELE MIKROKONTROLERÓW RODZINY INTEL MCS51 (8051)

W ramach niniejszych materiałów zostaną przedstawione wybrane właściwości **8-bitowego mikrokontrolera 8051**, podstawowego przedstawiciela mikrokontrolerów rodziny INTEL MCS-51, nazywanej także rodziną INTEL 8051, rodziną '51 lub serią 8x51. Mikrokontrolery te zdobyły bardzo dużą popularność w świecie i wciąż powstają ich różnorodne modyfikacje (istnieje już kilkadziesiąt różnorodnych odmian o wielu dodatkowych funkcjach w stosunku do wersji podstawowej). Rokuje to mikrokontrolerom tej rodziny jeszcze wiele lat obecności na rynku i zastosowań w różnorodnych urządzeniach.

Na rysunku 7.1 przedstawiono „pochodzenie" mikrokontrolera 8051 i jego „usytuowanie" względem procesorów stosowanych w mikrokomputerach klasy IBM PC.



Rys.7.1 „Drzewo genealogiczne" mikroprocesorów firmy INTEL

źródło: Dyrcz K. i inni: "Podstawy ..."

Protoplasta rodziny **MCS-51** to rodzina *MCS-48* – firma ***Intel***.

Omawiany mikrokontroler **8051** produkowany jest przez firmę *Intel* od 1980 r. Posiada on także możliwość podłączenia zewnętrznej pamięci programu i zewnętrznej pamięci danych o pojemności do **64 KB** (16-bitowa szyna adresowa).

Niektórzy przedstawiciele tej rodziny:

a) ***80C51*** - 4 KB ROM, od niego często stosowany symbol **8051**,
b) ***80C31*** - bez pamięci programu,
c) ***87C51*** - 4 KB EPROM lub EPROM OTP (ang. *One Time Programmable*),
d) ***80C52*** - jak 80C51, ale 8KB ROM + timer T2 + dodatkowo 128B RAM,
e) *80C32* - jak 80C31 + timer T2 + dodatkowo 128B RAM,
f) *87C52* - jak 87C51, ale 8KB EPROM + timer T2 + dodatkowo 128B RAM,
g) *80C535*- we A/C i wiele innych dodatkowych możliwości - firma ***Siemens***,
h) ***80C537*** - we A/C, wy PWM i wiele innych dodatkowych możliwości - firma ***Siemens***,
i) *AT89C1051* - 1 KB Flash, 64B RAM, 1 timer - (**AT**... - firma ***Atmel***),
j) *AT89C2051* - 2 KB Flash, 128B RAM, 2 timery,
k) *AT89C51* - 4 KB Flash,
l) *AT89C52* - 8 KB Flash,
m) ***AT89S8252*** – 8 KB Flash + dodatkowa wewnętrzna pamięć danych typu EEPROM + możliwość programowania przez złącze szeregowe RS232 mikrokomputera PC (interfejs SPI) - nie jest wymagany specjalny programator.
n) jest jeszcze wiele innych odmian o różnych możliwościach, niektóre są o zmienionej architekturze, np. mikrokontrolery firmy ***Dallas*** *DS80C320*.

Uwaga: Litera **C** w symbolu oznacza, że układ wykonany w technologii CMOS.

Są też inne rodziny mikrokontrolerów, np. rodzina *MCS96* – mikrokontrolery 16-bitowe.

8. SCHEMAT MIKROKONTROLERA 8051

8.1. Schemat funkcjonalny mikrokontrolera 8051 (rys. 8.1, porównaj z rys. 6.1).

Poniżej przedstawiono krótko bloki funkcjonalne mikrokontrolera. Bardziej szczegółowo zostaną one omówione w dalszej części materiałów.

a) 8 - bitowa jednostka centralna **CPU** (ang. *Central Processing Unit*) - wykonuje rozkazy. Jest **111** rozkazów (instrukcji), rozkazy są 1- , 2- i 3-bajtowe.

Ta jednostka to „serce" mikrokontrolera, zawiera mikroprocesor (p. 4).

b) zegar (generator impulsów taktujących) (rys. 8.1) stabilizowany zewnętrznym rezonatorem kwarcowym (rys. 8.2 a),

c) 4 KB pamięci **ROM** przeznaczonej do przechowywania programu,

Pytanie.: Ile programu się tu zmieści ?

Odp.: Przy założeniu, że średnia długość instrukcji wynosi 2 bajty, daje to ok. 2000 rozkazów.

W mikrokontrolerze 8051 (80C51) ta pamięć programowana **jest fabrycznie**, więc dla nas zwykle będzie nieprzydatna – trzeba wtedy korzystać z **zewnętrznej pamięci programu**. Sposób jej podłączenia zostanie podany w dalszej części materiałów.

d) pamięć **RAM** - dwie części:

- pamięć **użytkownika** - 128 B (w niektórych wersjach mikrokontrolera ta pamięć ma pojemność 256 B):
  - przechowywanie bieżących danych,
  - w tym obszarze jest umiejscowiony *stos*,
- obszar specjalnego przeznaczenia - (**SFR - rejestry specjalne)** - 128 B.

e) system przerwań - 5 źródeł, 2 poziomy.

f) układ czasowo - licznikowy

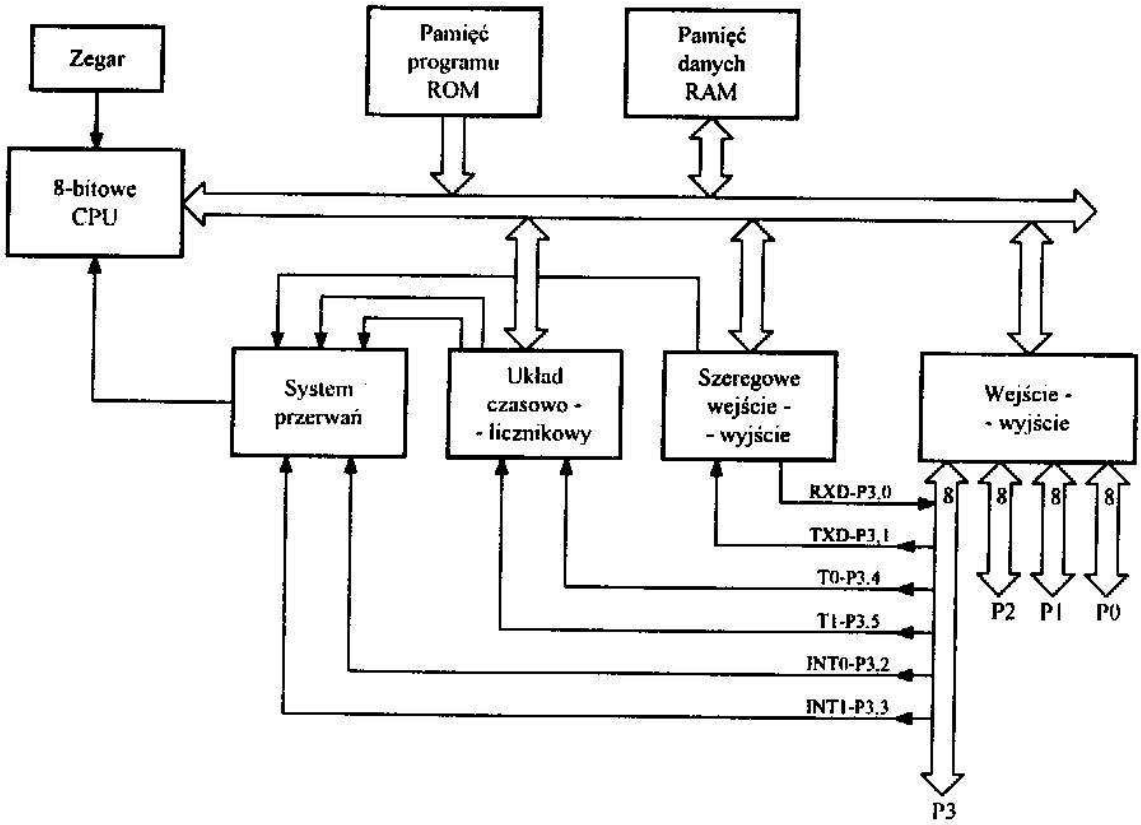
- **dwa** 16-bitowe czasomierze (timery) \ liczniki ***T0*** i ***T1***,
- możliwość zliczania **impulsów zegarowych** (pomiar czasu) lub **impulsów zewnętrznych**,

g) szeregowe wejście \ wyjście – komunikacja z otoczeniem w obie strony,

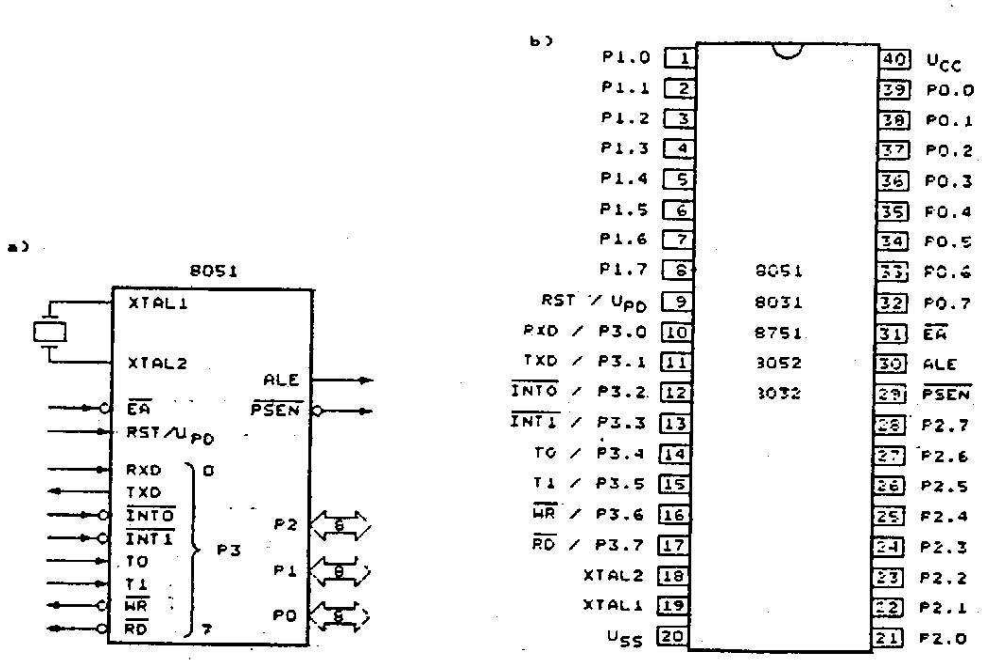
h) 4 porty we \ wy: *P0 ... P3* po 8 linii każdy (rys. 8.1, rys. 8.2 a) i b):

- poszczególne linie portów (każda linia to oddzielna „nóżka" układu scalonego) można wykorzystywać **dowolnie** jako wejścia lub wyjścia cyfrowe,
- linie portu ***P3*** mają **dodatkowe** funkcje (rys. 8.2),
- porty ***P0*** i ***P2*** są wykorzystane przy współpracy z zewnętrzną pamięcią programu i/lub danych.

i) wewnętrzna szyna systemowa łącząca poszczególne bloki funkcjonalne.



Rys. 8.1. Schemat funkcjonalny mikrokontrolera 8051 [2]



Rys. 8.2. Układ 8051: a) symbol logiczny; b) rozkład sygnałów na końcówkach [2]

8.2. Schemat logiczny mikrokontrolera 8051 oraz rozkład jego wyprowadzeń (rys. 8.2a)

Schemat logiczny zawiera *wszystkie sygnały wyprowadzone do końcówek układu.*

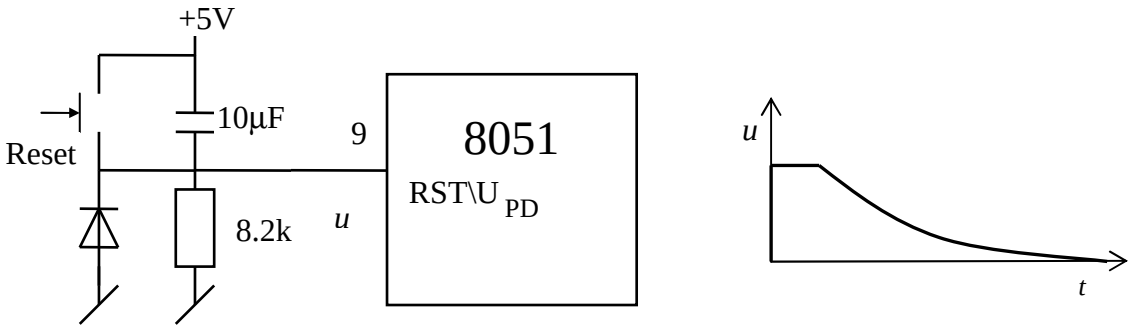
a) **XTAL1, XTAL2**:

- miejsca dołączenia **rezonatora kwarcowego** (o częstotliwości f_{XTAL}), standardowo $1{,}2 \div 12$ MHz; w nowszych rozwiązaniach może to być rezonator o większej częstotliwości,
- w mikrokontrolerze 80C51 można też dołączyć **zewnętrzny sygnał zegarowy** do *XTAL1* (wtedy *XTAL2* pozostaje nie podłączony).

b) **EA** - stan tego wejścia określa, **z której pamięci** należy **pobierać rozkazy:**

- wejście EA\ podłączone do +5V – rozkazy pobierane z pamięci wewnętrznej, ale rozkazy o adresach przekraczających pojemność pamięci wewnętrznej pobierane automatycznie z pamięci zewnętrznej.
- wejście EA\ podłączone do 0V – rozkazy pobierane z pamięci zewnętrznej; tak trzeba zrobić, aby wykorzystać układ 8051 z pamięcią ROM o nieznanej zawartości lub w przypadku stosowania mikrokontrolera bez wewnętrznej pamięci programu, np. 8031,

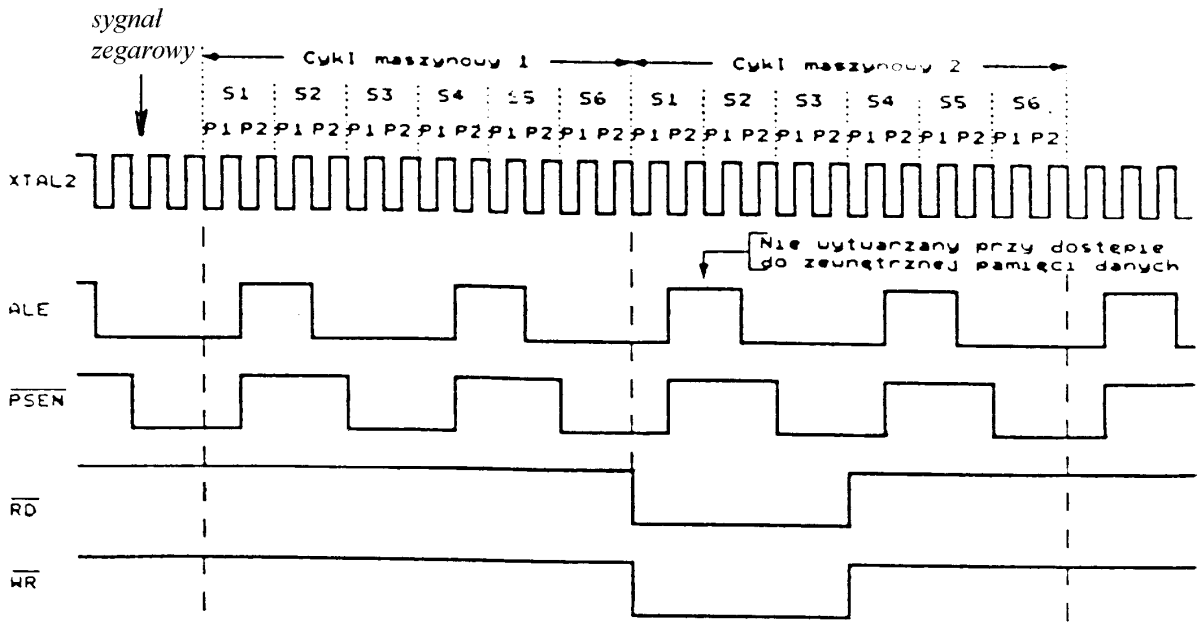
c) **RST\U$_{PD}$ –** podanie na to wejście stanu wysokiego (*H*) przez odpowiednio długi czas powoduje wyzerowanie mikrokontrolera, czyli jego ponowne zainicjowanie. Typowy układ zerowania mikrokontrolera przedstawia rys. 8.3. Po naciśnięciu (choćby na bardzo krótko) i zwolnieniu przycisku *Reset* napięcie na wejściu RST\U$_{PD}$ mikrokontrolera wzrasta szybko do 5V, a następnie maleje. Parametry obwodu muszą być tak dobrane, aby zapewnić wymagany czas trwania stanu wysokiego (rys. 8.3).



Rys. 8.3. Typowy układ zerowania mikrokontrolera oraz przebieg napięcia na wejściu RST\U$_{PD}$ podczas zerowania

d) **PSEN** - wyjście sterujące (rys. 8.4) wykorzystywane do odczytu **zewnętrznej pamięci programu**,

e) **ALE** – wyjście sygnału zegarowego o częstotliwości $\frac{f_{XTAL}}{6}$ (rys. 8.4); umożliwia zatrzaskiwanie młodszego bajta adresu przy współpracy z pamięcią zewnętrzną,



Rys. 8.4. Sygnały sterujące *(źródło: Rydzewski A.: "Mikrokomputery...")*

f) **RD** - sygnał odczytu z zewnętrznej pamięci danych (rys. 8.4),

g) **WR** - sygnał zapisu do zewnętrznej pamięci danych (rys. 8.4).

Uwaga: Sygnały $\overline{WR}$ i $\overline{RD}$ są aktywne stanem niskim i nigdy nie mogą równocześnie przyjmować stanu *0*.

h) **T0**, **T1** - wejścia impulsów zewnętrznych zliczanych przez liczniki odpowiednio **T0** i **T1**,

i) **INT0\, INT1** - wejścia sygnałów przerwań zewnętrznych,

j) **RXD**, **TXD** - wyprowadzenia używane przy transmisji szeregowej.

Uwaga: Sygnały wymienione w punktach od f) do j), czyli sygnały od $\overline{RD}$ do *TXD*, są dostępne na liniach portu *P3* (rys. 8.2); jeśli opisana wyżej funkcja któregokolwiek z tych sygnałów jest wykorzystana, nie może on być już wykorzystywany jako zwykła linia We\Wy. Przykładowo, jeśli korzystamy z zewnętrznej pamięci danych, to do jej obsługi konieczne są sygnały $\overline{WR}$ i $\overline{RD}$. Wobec tego linie *P3.6* i *P3.7* (rys. 8.2b) są zajęte i nie można do nich podłączać innych urządzeń, czy elementów.

k) $\mathbf{U_{CC}}$ – dodatni biegun „+” zasilania, napięcie zasilania z reguły nie może przekroczyć +6.5V, typowo $5V \pm 0.25V$; zasadą jest blokowanie tego wyprowadzenia kondensatorem 100nF do masy.

l) $\mathbf{U_{SS}}$ - masa układu,

m) porty ***P0***, ***P1***, ***P2*** i ***P3*** - zostaną one omówione bardziej szczegółowo w dalszej części materiałów.

8.3. Wewnętrzny schemat blokowy mikrokontrolera 8051 (rys. 8.5)

Przedstawiony wcześniej rysunek 8.1 pokazuje najważniejsze bloki funkcjonalne mikrokontrolera 8051, natomiast na rys. 8.5 przedstawiono elementy zawarte w strukturze mikrokontrolera w sposób bardziej szczegółowy. Poniżej podano tylko podstawowe informacje o niektórych elementach zawartych w strukturze mikrokontrolera. Wiele z nich zostanie opisanych dokładniej przy omawianiu poszczególnych bloków funkcjonalnych mikrokontrolera.

a) ***ALU*** (ang. *Arythmetic-Logic Unit*) - **jednostka arytmetyczno-logiczna** (p. 4) - wykonuje operacje arytmetyczno-logiczne na liczbach 8-bitowych, m. in.:

- operacje **arytmetyczne** - dodawanie, odejmowanie, mnożenie, dzielenie, inkrementacja (zwiększanie o 1), dekrementacja (zmniejszanie o 1),
- operacje **logiczne** - suma logiczna, iloczyn logiczny, różnica symetryczna, negacja zawartości akumulatora, przesuwanie cykliczne zawartości akumulatora w prawo lub lewo.

b) **akumulator *ACC* (*A*)** - jeden z podstawowych rejestrów 8-bitowych, bierze udział w operacjach wykonywanych przez *ALU*, wykorzystywany także przy pobieraniu i umieszczaniu danych w zewnętrznej pamięci danych,

c) **rejestr *B*** - 8-bitowy rejestr pomocniczy, bierze udział w operacjach mnożenia i dzielenia,

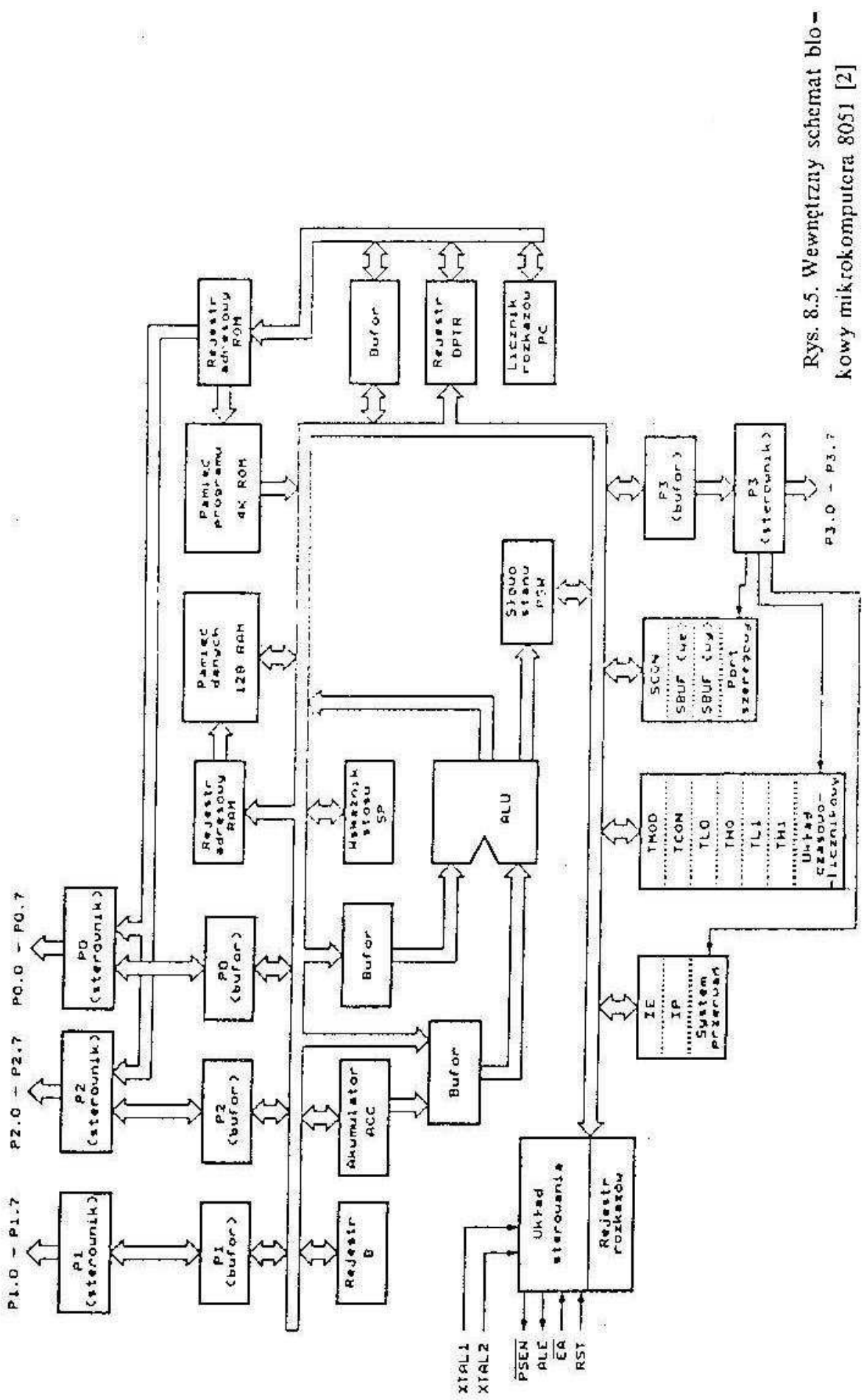
Uwaga: rejestry *A* i *B* mogą być także używane przez programistę jako rejestry uniwersalne.

d) ***PSW*** (ang. *Program Status Word*) - **słowo stanu programu** – rejestr 8-bitowy. Składa się z pojedynczych **znaczników** (bitów), zwanych także **flagami**, informujących o przebiegu oraz wyniku operacji arytmetycznych i logicznych. Poniżej przedstawiono poszczególne znaczniki znajdujące się w rejestrze *PSW*.

	7	*6*	*5*	*4*	*3*	*2*	*1*	*0*	
D0H	**CY**	AC	F0	RS1	RS0	OV	-	P	**PSW**
	(MSB)							(LSB)	

Każdy prostokącik odpowiada jednemu bitowi – razem jest ich 8, tyle ile bitów w bajcie. Najbardziej z lewej strony jest bit „najstarszy”, o numerze 7, nazywany ***MSB*** (ang. *Most Significant Bit*). Najbardziej z prawej strony jest bit „najmłodszy”, o numerze 0, nazywany ***LSB*** (ang. *Least Significant Bit*).

- najstarszy (najbardziej znaczący) bit rejestru *PSW* nazywa się *CY*, zaś najmłodszy (najmniej znaczący) bit jest nazwany *P*.
- podana z lewej strony liczba szesnastkowa *D0H* oznacza adres rejestru *PSW* w wewnętrznej pamięci mikrokontrolera (jest to tzw. obszar rejestrów specjalnych (***SFR***) – będzie on opisany w dalszej części opracowania).



Rys. 8.5. Wewnętrzny schemat blokowy mikrokomputera 8051 [2]

Uwaga: W podobny sposób, czyli wg powyższego schematu, będzie przedstawiana także zawartość innych rejestrów mikrokontrolera.

- najczęściej używanym znacznikiem zawartym w rejestrze *PSW* jest wspomniany już znacznik ***CY*** (ang. *Carry Flag*) – tzw. **znacznik przeniesienia.** Jest to siódmy, najstarszy bit rejestru *PSW*, można go również zapisać jako **PSW.7**:
  - pełni funkcję znacznika przeniesienia przy dodawaniu („jeden dalej"): jest ustawiany, jeśli wynik dodawania przekracza zakres 8-bitowy, w przeciwnym razie jest zerowany,
  - pełni funkcję znacznika pożyczki przy odejmowaniu: jest ustawiany, jeśli wynik odejmowania jest ujemny, w przeciwnym razie jest zerowany,
  - przy operacjach logicznych **na bitach** pełni rolę akumulatora boolowskiego,
  - przy programowaniu często występuje pod nazwą ***C***.

e) ***SP*** (*Stack Pointer*) - **wskaźnik stosu** – rejestr 8-bitowy; podaje adres ***wierzchołka stosu***, który jest w pamięci wewnętrznej RAM. Stos jest wykorzystywany do:

- automatycznego zapamiętywania adresu przy realizacji procedur obsługi przerwań i procedur użytkownika,
- przechowywania danych przez użytkownika.

Działanie stosu zostanie opisane w dalszej części opracowania.

f) **rejestry 16 - bitowe:**

- ***PC*** (ang. *Program Counter*) - **licznik rozkazów** (p. 4) - zawiera adres kolejnej instrukcji do pobrania z pamięci programu – nie jest dostępny programowo w sposób bezpośredni,
- ***DPTR*** (złożony z dwu rejestrów 8-bitowych: ***DPH*** - starsze 8 bitów i ***DPL*** młodsze 8 bitów) – wpisuje się do niego adres żądanej komórki w pamięci programu lub w zewnętrznej pamięci danych; będzie przez nas wykorzystywany przy współpracy mikrokontrolera z przetwornikami A/C i C/A,

g) **wewnętrzna szyna danych** - umożliwia przesyłanie danych między rejestrami wewnętrznymi oraz komunikację z urządzeniami peryferyjnymi (zewnętrznymi),

h) **wewnętrzna szyna adresowa** łączy rejestry ***PC*** i ***DPTR*** z wewnętrzną pamięcią programu (ROM) lub / oraz z pamięcią zewnętrzną przez porty ***P0*** i ***P2***,

i) **rejestr rozkazów** i **układ sterowania (sterujący) (**część składowa mikroprocesora – p. 4**) -** do tego rejestru doprowadzane są z pamięci programu kody rozkazów (instrukcji), które zostają następnie „odczytywane" przez układ sterowania, nadzorujący także wykonanie tychże instrukcji.

j) wymienione już wcześniej porty **P0**, **P1**, **P2** i **P3**,

k) rejestry związane z **poszczególnymi blokami funkcjonalnymi** wspomnianymi wcześniej:

- systemem przerwań (**IE**, **IP**),
- układem czasowo-licznikowym (**TMOD**, **TCON**, itd.),
- portem szeregowym.

Niektóre z nich zostaną przedstawione w dalszej części opracowania.

Uwaga: Prawie wszystkie wymienione w punkcie 8.3 rejestry znajdują się w obszarze rejestrów specjalnych *SFR*, który zostanie omówiony szczegółowo w dalszej części opracowania.

*Dalsze omawianie mikrokontrolera będzie prowadzone w taki sposób, aby przedstawić demonstrowany na wykładzie program **przes_a** realizujący świecenie i „przesuw" diod świecących oraz kolejne wersje tego programu.*

9. UKŁAD POŁĄCZEŃ do realizacji programu ***przes_a*** realizującego „przesuw" diod świecących

Na wykładzie zademonstrowano m. in. układ z wykorzystaniem mikrokontrolera rodziny MCS51, który w zależności od stanów dwóch wejść:

- ***W1*** – załącz / wyłącz przesuw,
- ***W2*** – przesuw prawo / lewo,

steruje sześcioma diodami świecącymi w jeden z następujących sposobów:

- stabilne świecenie dwóch diod, pozostałe zgaszone,
- świecenie jednej lub dwu diod z przesuwem w prawo,
- świecenie jednej lub dwu diod z przesuwem w lewo.

Na rysunku 9.1 przedstawiono sposób podłączenia sygnałów wejściowych oraz diod do mikrokontrolera.

a) wyprowadzenia **P3.4** i **P3.5** służą jako ***wejścia cyfrowe***:

- styk zamknięty - stan *L* (*0*), bo wejście jest zwarte do masy przez ten styk,
- styk otwarty - stan *H* (*1*), bo na wejściu pojawia się napięcie +5V (przez wewnętrzny rezystor podciągający).

Wejścia *P3.4* i *P3.5* to odpowiednio czwarty i piąty bit portu *P3* (czwarta i piąta **linia portu** *P3*). Są dostępne na wyprowadzeniach układu 8051 o numerach odpowiednio 14 i 15 (rys. 8.2b).

b) wyprowadzenia **P1.1** ÷ **P1.6** służą jako ***wyjścia cyfrowe***:

- anody diod *D1* ÷ *D6* są zasilane przez bufory odwracające,
- stan diod jest następujący:
  - wyjście mikrokontrolera wyzerowane (stan *0*) – dioda świeci,
  - wyjście mikrokontrolera ustawione (stan *1*) – dioda zgaszona.

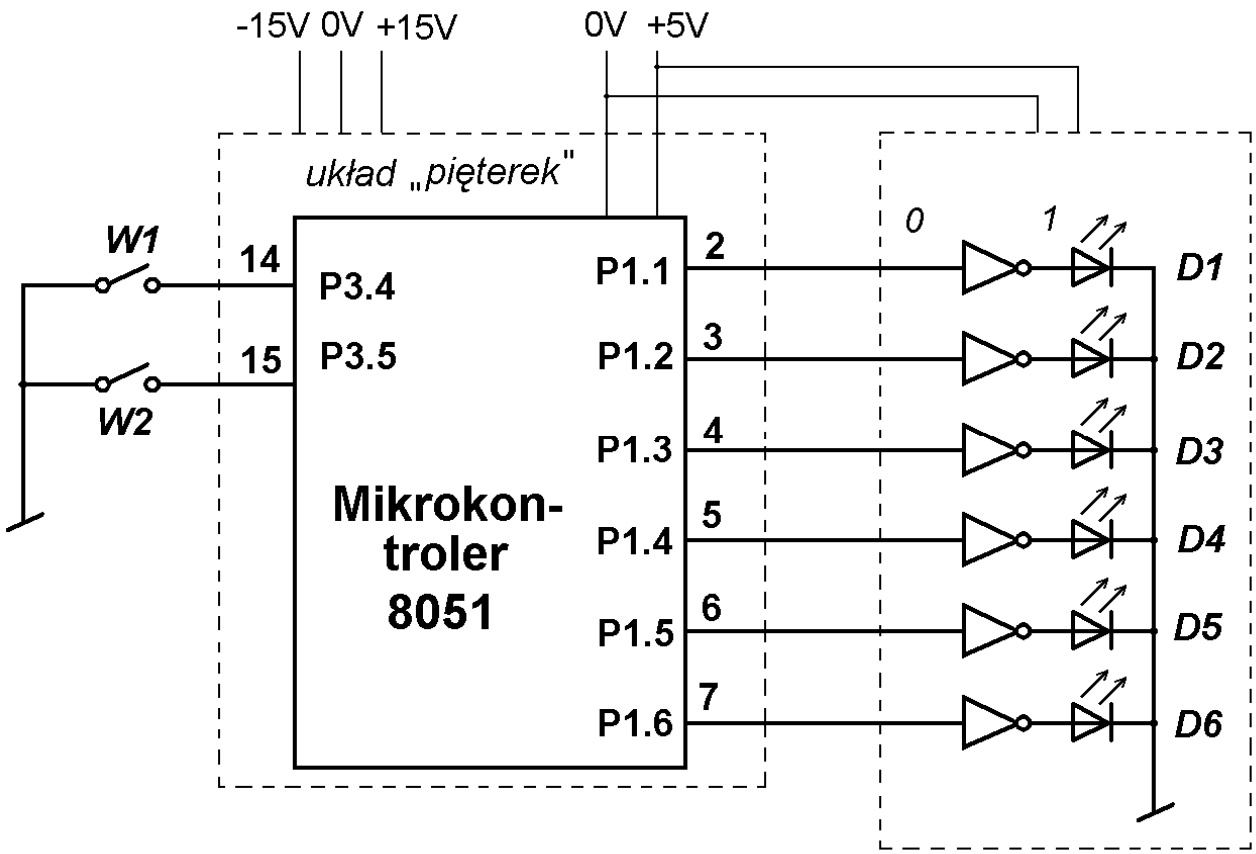
Wyjścia *P1.1* ÷ *P1.6* to odpowiednio pierwsza do szóstej linie portu *P1*. Są dostępne na wyprowadzeniach układu o numerach odpowiednio od 2 do 7 (rys. 8.2b).

Uwaga: Linie *P1.0* i *P1.7* nie są wykorzystane.

c) rola **buforów odwracających**:

- są elementami pośredniczącymi między mikrokontrolerem, a diodami (dodatkowe zabezpie-czenie wyjść mikrokontrolera),
- powodują odwrócenie polaryzacji - dioda świeci przy wyzerowanym wyjściu. Jest to o tyle istotne, że po starcie mikrokontrolera wszystkie linie portów są **ustawione** (przyjmują stan *1*) i przy przyjętej tutaj polaryzacji odbiorniki (u nas diody) są **wyłączone**.

Poniżej omówione są dokładniej porty mikrokontrolera.



Rys.9.1. Schemat układu do „przesuwu” diod

Wejścia:

- przycisk zamknięty – stan *0*,
- przycisk otwarty – stan *1*.

Diody świecące:

- linia portu P1.x wyzerowana – dioda świeci,
- linia portu P1.x ustawiona – dioda zgaszona.

10. PORTY MIKROKONTROLERA

Porty są to specjalne rejestry pośredniczące między urządzeniem zewnętrznym, a szyną systemową mikroprocesora.

W tym punkcie podano podstawowe informacje o budowie portów wejścia / wyjścia.

a) mikrokontroler ma **4 porty wejścia / wyjścia ogólnego przeznaczenia**, każdy o 8 wyprowadzeniach (**liniach**) razem są więc **32** wyprowadzenia (32 linie portów) – rys. 8.1, rys. 8.2,

b) każda linia portu to oddzielna „nóżka" mikrokontrolera – rys. 8.2b),

c) każda linia portu może być wykorzystana jako standardowe wejście **lub** wyjście cyfrowe – decyduje o tym projektant (programista),

d) każdemu portowi przyporządkowany jest **rejestr** w obszarze rejestrów specjalnych ***SFR***, przy czym nazwa rejestru jest taka sama, jak nazwa portu. Ten rejestr można od strony programowej traktować tak samo, jak inne rejestry mikrokontrolera (np. wspomniany wyżej rejestr *PSW*). Poniżej podany jest sposób uszeregowania bitów w rejestrze *P1* odpowiadającemu portowi *P1*:

90H	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0	**P1**
	(MSB)							(LSB)	

Każdy bit rejestru odpowiada jednej linii portu (jednej „nóżce" mikrokontrolera): bit *P1.0* odpowiada linii *P1.0* („nóżka" 1 mikrokontrolera), bit *P1.1* – linii *P1.1* („nóżka" 2), itd. (rys. 8.2b).

Przykładowo założono, że po załączeniu lub wyzerowaniu mikrokontrolera przy otwartym *W1* (rys.9.1) świecą diody *D1* i *D4*, zaś pozostałe są zgaszone. Linie portu *P1.1* (*D1*) i *P1.4* (*D4*) muszą więc być wyzerowane, a pozostałe winny być ustawione. Stan bitów w rejestrze *P1* powinien wobec tego być następujący (bity *P1.0* i *P1.7* nie są wykorzystane, więc mogą być ustawione, jak poniżej, lub też wyzerowane):

90H	1	1	1	0	1	1	0	1	**P1**
		D6	*D5*	*D4*	*D3*	*D2*	*D1*		

Zawartość rejestru *P1* powinna więc wynosić *1110 1101B* (binarnie) = *EDH* (heksadecymalnie) i taką wartość należy do rejestru wpisać. Wpisywanie wartości *0* (**zerowanie bitu**) lub *1* (**ustawienie bitu**) do poszczególnych bitów rejestru dokonywane jest programowo i jest opisane w dalszej części opracowania.

e) struktura linii portu (rys.10.1)

Przedstawione na rys. 10.1 schematy dotyczą **pojedynczej linii portu** – każdy port zawiera więc po 8 takich układów.

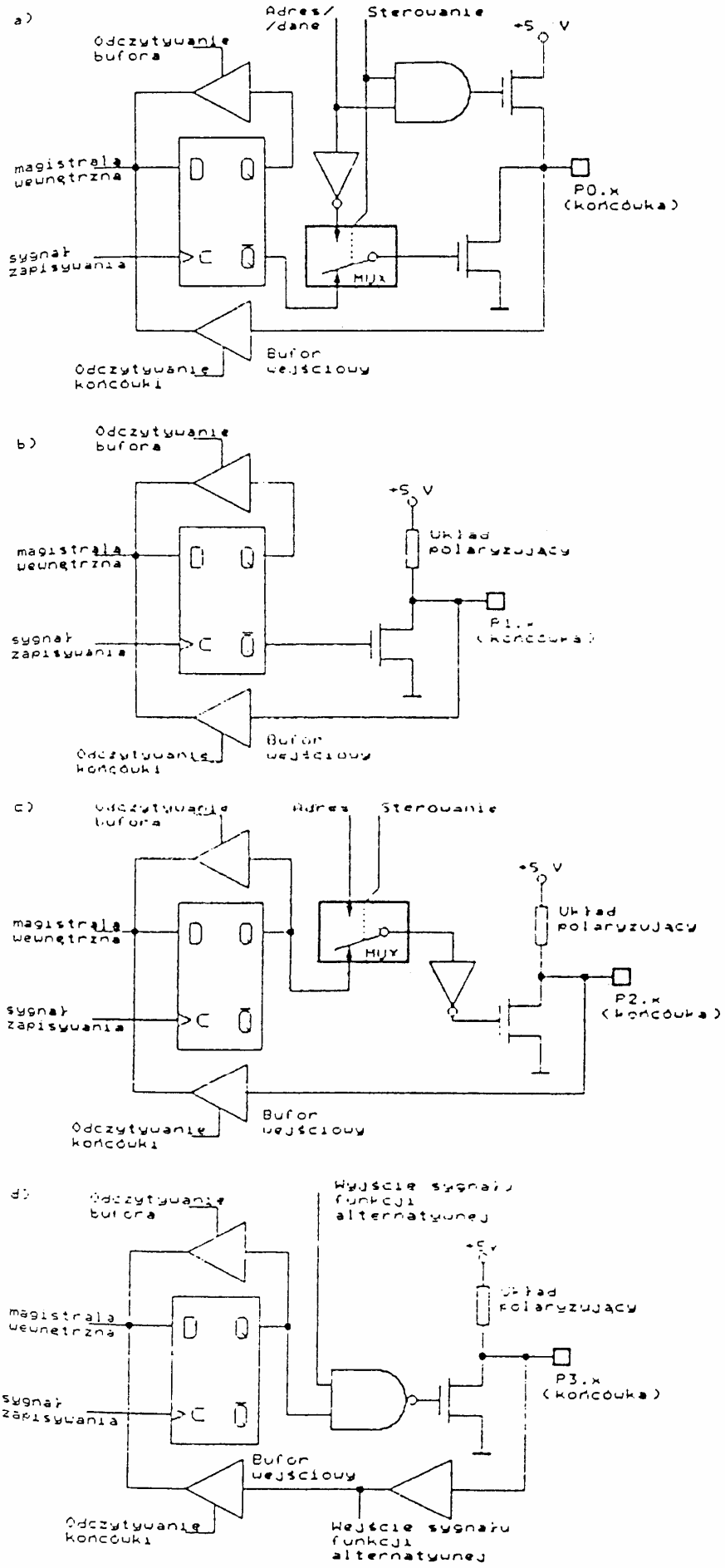
Schemat ideowy linii każdego portu jest **nieco inny** - najprostszy jest schemat linii portu *P1* (rys. 10.1 b) i on zostanie opisany poniżej.

A oto podstawowe właściwości portu P1:

- z każdą linią portu związany jest **przerzutnik** typu D, którego wyjście *Q* to odpowiedni bit rejestru odpowiadającego danemu portowi. Z każdym portem związane jest więc 8 przerzutników i przykładowo, w przypadku portu P1, wyjście *Q* każdego z nich odpowiada jednej z linii P1.0, P1.1, ... , P1.7.
- jeśli linia portu pracuje jako wyjściowa, stan wyjść przerzutnika decyduje o stanie końcówki (czyli wyprowadzenia linii portu na zewnątrz mikrokontrolera) poprzez tranzystor,
- porty *P1÷ P3* mają **rezystor podciągający** końcówkę do „+" (układ polaryzujący), co wymusza stan wysoki na końcówce, gdy tranzystor nie jest wysterowany (pod warunkiem, że nie jest wymuszony stan niski z zewnątrz – rys. 10.1 b),
  - warunkiem **ustawienia** linii portu jest wpisanie *1* do przerzutnika, co oznacza, że wyjście *Q* przyjmie stan 1, zaś wyjście $\bar{Q}$ stan 0; wtedy tranzystor nie będzie wysterowany (rys.10.1b) i na końcówce pojawi się stan wysoki poprzez układ polaryzujący (o ile nie zostanie wymuszony stan niski z zewnątrz),
  - warunkiem **wyzerowania** linii portu jest wpisanie *0* do przerzutnika, co oznacza, że wyjście *Q* przyjmie stan 0, zaś wyjście $\bar{Q}$ stan 1; wtedy tranzystor zostanie wysterowany (rys.10.1 b) i na końcówce pojawi się stan niski.

Uwaga: z powyższego opisu wynika, że **stan przerzutnika** (a dokładnie jego wyjścia *Q*) **odpowiada stanowi końcówki** i poza szczególnymi przypadkami tak właśnie jest.

- port *P0* nie posiada takiego układ polaryzującego – linie tego portu mogą być użyte jako **wejścia o wysokiej impedancji** (stanowią linie szyny danych przy współpracy z pamięcią zewnętrzną); w razie potrzeby można także podłączyć odpowiedni zewnętrzny rezystor podciągający do linii tego portu,
- porty ***P0*** i ***P2*** są wykorzystywane przy współpracy z zewnętrzną pamięcią programu i (lub) zewnętrzną pamięcią danych i wtedy nie można już ich dodatkowo wykorzystać jako „zwykłych" wejść \ wyjść cyfrowych,



Rys. 10.1. Sruktura linii portu: a) P0; b) P1; c) P2; d) P3 [2]

- także linie portu ***P3*** mogą pełnić **dodatkowe funkcje** i jeśli linia tę funkcję pełni, też nie można jej dodatkowo wykorzystać jako zwykłego (standardowego) wejścia \ wyjścia cyfrowego. Jeśli natomiast linię wykorzystano jako zwykłe wejście \ wyjście cyfrowe, nie może ona równocześnie pełnić swej dodatkowej funkcji.
- **odczytywany** może być zarówno **stan końcówki**, jak i **stan przerzutnika** (w niektórych przypadkach mogą się one różnić, np. przy bezpośrednim podłączeniu diody lub bazy dodatkowego tranzystora do wyjścia); w przypadku każdej instrukcji odczytującej stan linii portu jest ściśle określone, czy odczytuje ona stan końcówki, czy stan przerzutnika,
- jeśli linia ma być **wejściem**, to do przerzutnika musi być **wpisana *1***, gdyż tranzystor musi być wyłączony; taki stan występuje po załączeniu mikrokontrolera lub po jego wyzerowaniu: wtedy do wszystkich portów wpisywane są same **„jedynki”**, czyli zawartość rejestru każdego portu wynosi *1111 1111B = FFH.*

Uwagi dodatkowe:

Ustawianie i zerowanie poszczególnych linii portu dokonywane jest przy wykorzystaniu odpowiednich instrukcji, co jest szczegółowo opisane w p. 12 opracowania.

Jeżeli linia portu jest wykorzystywana jako wyjście, to po jej ustawieniu lub wyzerowaniu jej stan (poprzez przerzutnik związany z tą linią) pozostaje niezmieniony do chwili wpisania nowego stanu do przerzutnika.

11. ALGORYTM programu *przes_a* realizującego „przesuw" diod świecących.

Mikrokontroler działa według odpowiedniego **programu**, który musi zostać napisany przez programistę. Przed przystąpieniem do pisania programu należy mieć **bardzo precyzyjny** obraz czynności, które powinien wykonywać mikrokontroler. W tym bardzo pomocny jest algorytm, który funkcje wykonywane przez mikrokontroler przedstawia w prosty i czytelny sposób.

Ogólnie **algorytmem** nazywa się ściśle określony ***sposób postępowania***, doprowadzający do rozwiązania zadania. Operacje realizowane kolejno w czasie nazywa się **krokami** algorytmu.

Podstawowymi elementami algorytmu są:

- **sekwencja operacji** – zbiór operacji realizowanych w określonej kolejności na określonych danych,
- **przełącznik** – element badający spełnienie określonego warunku i realizujący skok w różne miejsca w programie zależnie od tego, czy warunek jest spełniony, czy też nie,
- **pętla –** element umożliwiający kolejną wielokrotną realizację określonych operacji.

Poniżej zostanie opisany sposób tworzenia algorytmu, na podstawie którego zostanie następnie opracowany program ***przes_a*** realizujący „przesuw" diod świecących. Schemat układu do realizacji „przesuwu" diod przedstawiony jest na rys. 9.1.

11.1. Działanie programu

Poniżej jeszcze raz przedstawiono działanie programu, uwzględniając tym razem przyporządkowanie styków i diod do odpowiednich wyprowadzeń mikrokontrolera.

Program ***przes_a*** (co oznacza *„przesuw – wersja a"*) w zależności od stanów dwóch wejść:

- ***W1*** (*P3.4*) – załącz / wyłącz przesuw,
- ***W2*** (*P3.5*) – przesuw prawo / lewo,

realizuje określone świecenie diod ***D1*** ÷ ***D6*** podłączonych do linii portów odpowiednio *P1.1* ÷ *P1.6*. W szczegółach wygląda to następująco:

a) jeśli styk *W1* jest otwarty (*P3.4* = 1):

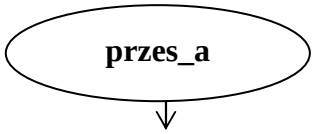
- świecą diody *D1* i *D4*,
- pozostałe diody są zgaszone,

b) jeśli styk *W1* jest zamknięty (*P3.4* = 0) następuje cykliczna zmiana stanu diod, co jest widoczne jako „przesuw" świecących diod. Kierunek „przesuwu" zależy od stanu styku *W2*:

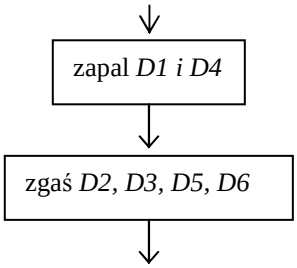
- jeśli *W2* jest otwarty (*P3.5* = 1) następuje „przesuw" w lewo,
- jeśli *W2* jest zamknięty (*P3.5* = 0) następuje „przesuw" w prawo.

11.2. Tworzenie algorytmu

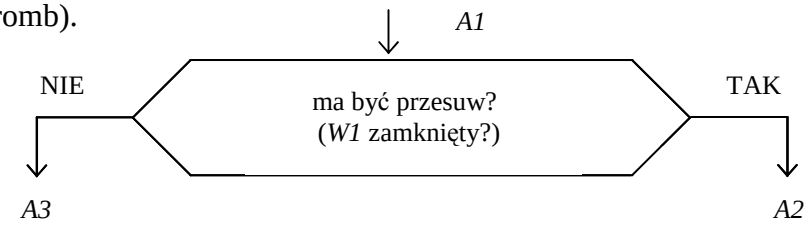
a) zaczyna się od narysowania nagłówka (w kształcie prostokąta lub elipsy), w którym umieszczamy słowo *START* lub podajemy nazwę programu, tu: ***przes_a,*** a następnie rysujemy strzałkę prowadzącą do kolejnego bloczka algorytmu,



b) następnie rysujemy bloczki w kształcie **prostokątów** zawierające instrukcje początkowe (inicjalizacyjne) – **sekwencja operacji**. W tym przypadku będzie to zapalenie diod *D1* i *D4* oraz zgaszenie pozostałych diod,



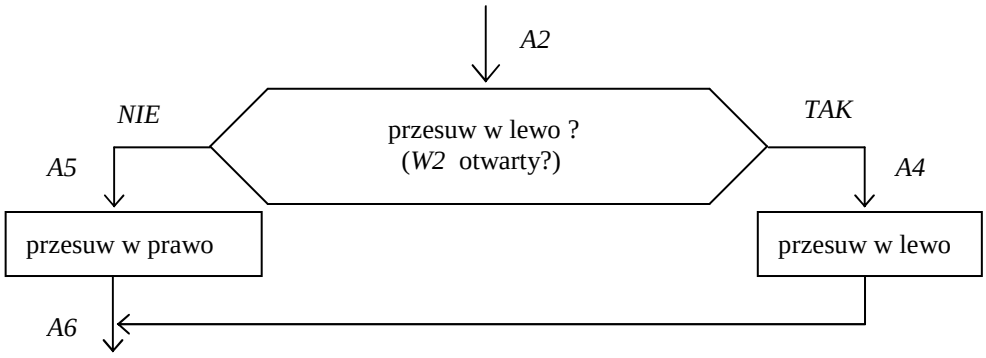
c) teraz trzeba sprawdzić stan styku *W1* i albo zapewnić stan diod jak wyżej (jeśli *W1* jest otwarty – świecą dalej diody *D1* i *D4*, pozostałe zgaszone) albo realizować „przesuw" (jeśli *W1* jest zamknięty). Jak widać z powyższego, konieczne jest wprowadzenie **rozgałęzienia** w programie, co realizuje **przełącznik** („bloczek decyzyjny") w kształcie ***sześciokąta*** (czasem rysuje się go jako romb).



W zależności od stanu styku *W1* należy przejść do jednego z dwu miejsc w programie, oznaczonych symbolami *A2* i *A3* (zamiast *A2* i *A3* można tu wprowadzić inne nazwy, mogą to być prawie dowolne wyrazy). Taką symboliczną nazwę oznaczającą miejsce w programie, będziemy nazywać **etykietą**.

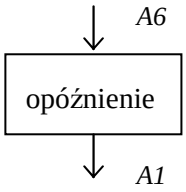
Powyżej wprowadzono także etykietę *A1* oznaczającą to miejsce w programie, w którym testowany jest stan styku *W1*. Ta etykieta jest konieczna, gdyż w to miejsce trzeba będzie wracać.

d) jeśli należy zrealizować „przesuw”, trzeba w podobny sposób sprawdzić stan styku *W2*, a następnie dokonać „przesuwu” w prawo lub w lewo. Zgodnie z oznaczeniami podanymi powyżej, ten fragment programu zaczyna się od etykiety *A2*.



Po wykonaniu „przesuwu” w wymaganym kierunku następuje przejście do następnego miejsca w programie, który tu oznaczono etykietą *A6*.

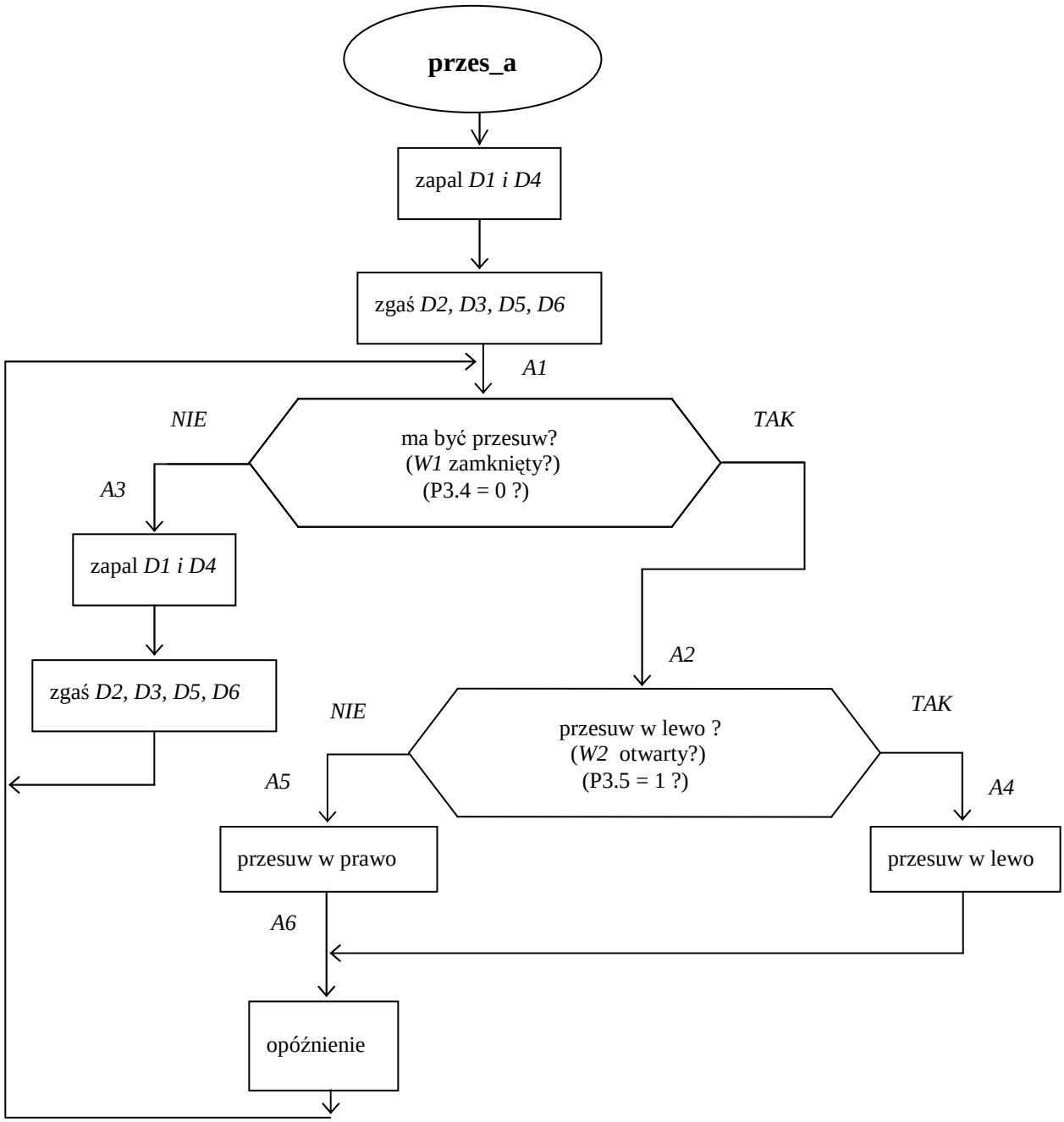
e) Po wykonaniu powyższych czynności można by już przejść do ponownego sprawdzania stanu styku *W1* (etykieta *A1*), zamykając w ten sposób **pętlę programową** (program musi działać w pętli, aby możliwa była zmiana sposobu świecenia diod podczas pracy układu). Jednak w przypadku wyboru „przesuwu” zmiana stanu diod następowałaby tak szybko (co ok. 10 μs), że zapalanie i gaszenie diod w ogóle nie byłoby zauważalne. Dlatego konieczne jest wprowadzenie opóźnienia rzędu części sekundy lub nawet sekund i dopiero po upływie tego czasu opóźnienia można wrócić na początek pętli (etykieta *A1*).



f) Można już teraz podać pełny algorytm programu – rys. 11.1.

- Dołożono w nim jeszcze do przełączników dodatkowe zapytania o spełnienie warunku. W przełączniku pierwszym zapada decyzja o tym, czy ma być przesuw, czy nie i takie zapytanie jest w nim jasno sformułowane. Dodatkowo w nawiasach podano równoważne zapytania (*czy W1 jest zamknięty?* oraz *czy bit P3.4 jest wyzerowany?),* które są przydatne dla programisty podczas pisania programu na podstawie algorytmu.
- Może powstać pytanie, dlaczego we fragmencie programu zaczynającym się od etykiety *A3* ponownie zapalamy i gasimy diody. Przecież po załączeniu układu te diody są już odpowiednio zapalone lub zgaszone. Po co więc powtarzać te czynności? Odpowiedź jest prosta. Do miejsca w programie oznaczonego etykietą *A3* możemy przejść także w sytuacji,

gdy przedtem był realizowany „przesuw” i stan diod może być bardzo różny (np. mogą być zapalone diody *D3* i *D6*). Tak więc ponowne podanie tu instrukcji zapalenia i zgaszenia diod jest konieczne.



Rys. 11.1. Algorytm programu ***przes_a***

Pytanie: Co się stanie, jeśli dioda jest zapalona, a my damy rozkaz jej zapalenia?
Odp: Nic się nie zmieni, dioda pozostanie zapalona.

W przedstawionym algorytmie testuje się, czy styk *W1* jest zamknięty oraz czy styk *W2* jest otwarty. Oczywiście można sprawdzać stan styków na odwrót (czyli sprawdzać, czy styk *W2* jest zamknięty, a *W1* – otwarty). Można też sprawdzać, czy oba styki są zamknięte lub otwarte.

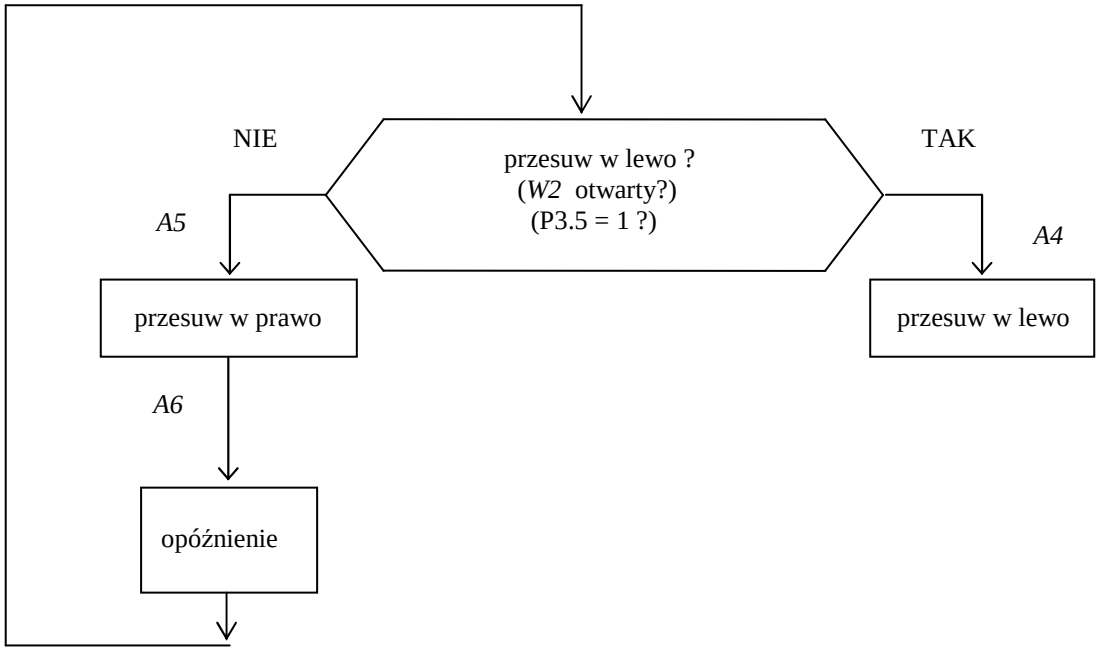
Przyjęte tu sprawdzanie, czy jeden ze styków jest otwarty, a drugi zamknięty, umożliwi poznanie dwu różnych instrukcji, co będzie omówione w p. 12.

Uwaga 1:

Algorytm powinien zawierać przede wszystkim sformułowania zrozumiałe dla każdego użytkownika, umożliwiające zrozumienie działania programu. Gdyby w algorytmie z rys. 11.1 podano np. tylko zapytanie czy *P3.4 = 0?*, byłoby to wprawdzie przydatne dla programisty, ale nie dostarczałoby żadnej informacji o tym, co ten program realizuje. Powyższą uwagę można uogólnić: w algorytmie należy podawać konkretne informacje, np. testowanie stanu styku, wczytanie liczby binarnej z portu, odczyt wejścia analogowego, sprawdzanie znaku różnicy dwu wielkości, jeśli taką liczymy, a dopiero ewentualnie na drugim miejscu informacje szczegółowe, jak stan bitu, stan znacznika przeniesienia *CY*, zawartość akumulatora *ACC*, itp.

Uwaga 2:

Należy zawsze zwrócić uwagę, aby **program działał w pętli**. Nawet, jeśli pozornie nic się nie dzieje, mikrokontroler podczas pracy musi bez przerwy wykonywać jakieś instrukcje. Nie może być tak, że po wykonaniu pewnych instrukcji program „idzie do nikąd". Nieprawidłowy więc jest poniższy fragment algorytmu, w którym nie jest określone, co ma być realizowane po wykonaniu przesuwu w lewo.



Rys. 11. 2. Nieprawidłowy fragment algorytmu programu

W następnych punktach zostaną przedstawione instrukcje realizujące zapalanie i gaszenie diod (zerowanie i ustawianie bitów) oraz sprawdzanie stanu styków W1 i W2 (testowanie stanu wejść cyfrowych).