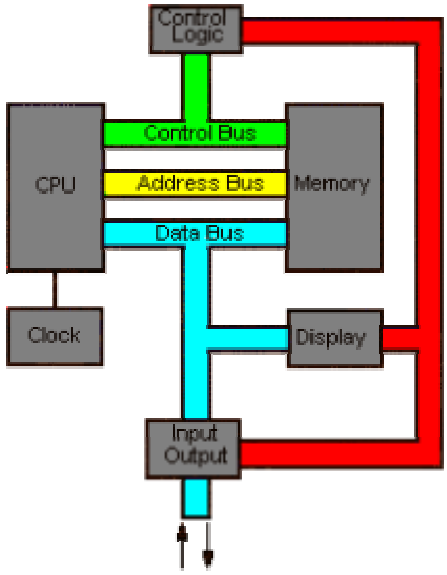
Микропроцесорен контролер

Микропроцесорната техника представлява върхна точка на цифровата система за проектиране. Тя се използва за контрол на почти всички автоматизирани процеси. Микропроцесорните контролери могат да бъдат използвани за извършване на почти всяка функция. Те се малки, сравнително евтини и надеждни. По-важното е, че при замяната им с подобна система, изградена от дискретни компоненти, би ни въвело в много трудна дизайнерска задача, невероятно сложна, свързана с голяма и ненадеждна електронна система, която изисква много поддръжка. При всички микропроцесорни цифрови системи се наблюдава тенденция към увеличаване на сложностния дизайн, с по-голяма скорост на работа на всеки един процесор и на намаляването на физическите му размери и цена. Микропроцесорът е контролирането на компютър, който използваме в момента, но той също е в центъра на **PIC (**Програмируеми Логически Контролери) или по-скоро **PIC (** програмируеми контролери интерфейс), който са широко използван в промишлената автоматизация и контрол.

- **Основната структура и функциониране на микропроцесорните системи**

Двата основни елемента на микропроцесорната система са процесора или централно звено за обработка на неговата памет. Взаимовръзките са известни като „автобуси", защото те съдържат голям брой паралелни свързващи кабели.



Трите групи „автобуси", свързви тези два блока са:

- **The data bus - Данните автобус**
- **The address bus - Адресът автобус**
- **The control bus - Контролът автобус**

The data bus - осъществява данни бивани обработвани и в двете посоки. Еднопосочният по направление **The address bus** доставя на паметта адреси. **The control bus** се използва, за да се провери дали всичко работи в правилната последователност чрез изпращане и получаване на временни сигнали.
Там също трябва да бъдат някои средства за комуникация между процесора и външния свят. Това се постига чрез използването на периферни устройства като клавиатури, мишки и VDU на които изпращат (вход), или получават (изход) информация, чрез входно / изходни портове.

Понякога тази информация е цифрово кодирана и трябва да бъде декодирана преди процесора да може да я поеме. Друг път тя може да бъде генерирана от аналогови устройства, които трябва да я преобразуват в цифров сигнал (или обратното), използвайки АЦ (аналогово към цифрово) или ЦА (цифров в аналогов) конвертори

преди употреба.

Всички микропроцесорни системи имат някаква памет. Тя може да бъде трайна, неизменчива памет (ROM) или временна - изменчива памет (RAM).

- **RAM** е гъвкава за четене и запис в паметта. Повечето АД и някои PIC имат някакви разширения в RAM, чрез които ще се съхраняват програми написани от потребителя. Когато казваме, че е RAM имаме предвид, че тя не може да се използва за съхранение на данни, докато PIC е изключен, освен ако към RAM е подкрепена с батерия (има батерия трайно прикрепена за подържане на данните в паметта). CMOS RAM обикновено се използва при подкрепителната система, защото използва толкова малка мощност и работи на по-широки напрежения отколкото традиционните TTL техника.

- **DRAM** е ефективно състояща се от кондензатор, който трябва да бъде поддържан на постоянно ниво на зареждане, ако има информация за съхраняване. За тази цел опресняването на устройствата редовно допълва зареждането на всяка клетка. Динамична памет е много по-малка, отколкото еквивалентната статична RAM.
- **ROM** се програмира по време на самото производство. Ние казваме, че е не-променлива, защото нейните данни са изградени и не се губи мощност, ако се премахнат. ROM често се използва за подържане на операционната система и други определени данни, изисквани от микропроцесорната система.

- **EPROM** е вид ROM, която може да бъде програмирана от електрически пулсации и след това заличавани при излагане на ултра-виолетова светлина през кварцови обективи монтирани в началото на всяко устройство. Веднъж програмирани лещите се покриват с непрозрачна пластмаса или хартия стикер. В това състояние EPROM провежда, тези които не са променливи ROM.

- **EEPROM** е подобна на EPROM, но изтрита е като програмирано чрез електрически импулси. Тя има гъвкавостта на подкрепително зареждащата CMOS RAM, но не изисква предварително зареждане. Също съхранява данни за неопределено време, докато се репрограмира. EEPROM се използва главно в повечето съвременни АД и PIC и главно поради тази причина, но писане на данни в EEPROM не отнема повече време , отколкото в RAM. За повечето приложения обаче това не е проблем.

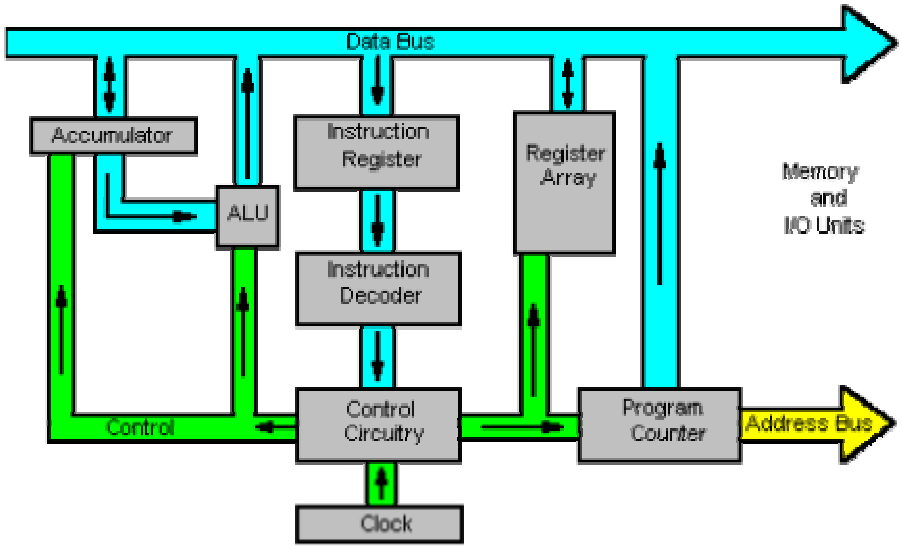
Сегашните развития в основните технологии на страните е стъпка към миниатюризациите което означава, че в съвременното PIC най-много от всичко паметта се изисква да е поставена на един единствен чип.

„CPU” - то В CPU

„CPU” е микропроцесорният "мозък" - какъвто можем да създадем да зависи до голяма степен от набор от инструкции (наречен набор инструкции), който е предназначен за рабора с тях.

Обикновено един CPU се състои от 3 основни части:

- The ALU
- The Registers - Регистър
- The Control Unit - Контролното звено



ALU или *Логическото устройство* изпълнява всички аритметични или логически функции.

Регистрите са местещи се регистри, където се обработват данни, за да могат да се съхраняват във временното съхранение. Броят и целта на регистрите варира от един микропроцесор до друг, но микропроцесор от тези местещи регистри се използва като *броячна* програма за предоставяне на адресите, захранени с адрес автобуса. Друг регистър се използва като *акумулатор,* който съдържа данните обработващи се във всеки един момент.

Контролният елемент е свързан с часовник, който генерира импулси с честота измервани в MHz. Неговата цел е контролиране и синхронизиране на начина, по който данните се обработват.

Инструкционният регистър притежава всяка инструкция, докато се декодира от *Инструкционния декодер,* който след това изпраща сигнал до контролната техника да се даде възможност на инструкцията да бъде отстранена.

Като пример нека погледнем само един машинен цикъл. В първия цикъл на програмата.

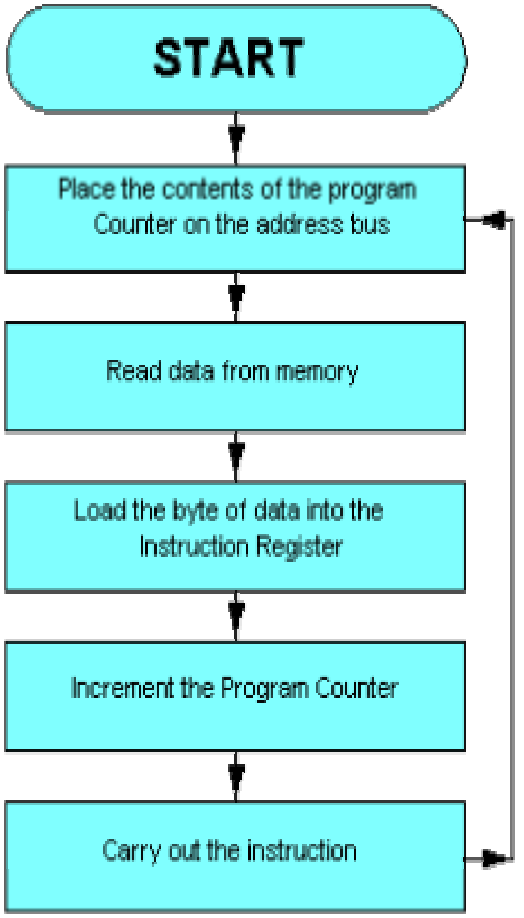
Съдържанието на програмата брои първоначално 0000 0000 е предаден на адрес автобуса.

Инструкциите в този първи адрес е "чети от паметта" и изпрати на данни автобус.

Инструкцията се поема от инструкционния регистър, като същевременно тя е декодирана в сигнала, който отива в контролната техника. Тогава се извършва подготовка.

Нека предположим, че указанието е да се зареди определен номер съхранен в паметта на акумулатора.

Програмният брояч увеличава на 0000 0001 съдържанието на този адрес който поставен в данните автобус и контролната техника зарежда този в акумулатора.



Ако искаме да добавим данните, съхранени в акумулатора към друг номер, съхранени на друг адрес, програмата трябва да даде необходимите инструкции за смяна на първия номер от акумулатора в един от вътрешните регистри и след това да прочетете втория номер и мястото му в акумулатора.

В инструкциите за въвеждане на съдържание в съдържанието на акумулатора със съдържанието на вътрешен регистър могат да бъдат изпратени на ALU и резултатът може да се съхранява в акумулатор за трансфер на единица продукция.

Бройни системи

Писане на определени групи от цифри в двоичен формат, като например 1001 1110 могат да бъдат тромави и объркващи ни, затова двоичните числа често се превръщат в десетичен (base10), осмичен (основа 8), шестнадесетичен (основа 16) или в двоично - десетични числа.

Двоична система

Двоичната система използва два вида цифри (1 и 0). Ние пишем един двоичен номер с *неговия най-значителен бит (MSB) най-далеч* от ляво и най -*малко значителен бит (LSB) най-далеч* в дясно, така да се организира във възходящ ред от 2 започвайки с най-малко значимия бит.

Долната диаграма обяснява конверсията от двоичен към десетичния код, съвсем ясно.

Bit 7 (MSB) Малко 7 (MSB)							Bit 1 (LSB) Малко 1 (LSB)
1 1	**0 0**	**1 1**	**1 1**	**0 0**	**1 1**	**0 0**	**1 1**
128 128	64 64	32 32	16 16	8 8	4 4	2 2	0 0
2^{7} ^{2} 7	2^{6} ^{2} 6	2^{5} ^{2} 5	2^{4} ^{2} 4	2^{3} ^{2} 3	2^{2} ^{2} 2	2^{1} ^{2} 1	2^{0} ^{2} 0

Двоична **10110101** = (1x128) + (0x64) + (1x32) + (1x16) + (0x8) + (1x4) + (0x2) + (1x1)

Осмична система

Осмичната (основа 8) бройна система използва набор от осем цифри (0 до 7).. Осмичните номера са организирани във възходящ нарастващ ред до 8. Например:

Осмичното число 321 = $(3x8^{2})+(2x8^{1})+(1x8^{0})$ = (3x64)+(2x16)+(1x1) = 209 в десетично.

За да се превърне двоична система в осмична е нужно да се раздели на групи от три бита. Това е така, защото осмичните номера от 0 до 7 представляват 3-битови двоични номера 000 до 111.

Octal Осмична	7 7	6 6	5 5	4 4	3 3	2 2	1 1	0 0
Binary Двоична	111 111	110 110	101 101	100 100	011 011	010 010	001 001	000 000

Диаграма по-долу показва как се конвертира осем битов номер в двоичен номер, за да е осмичен и тогава в десетичен:

8 битов в двоичен номер 11111101

Разделяне на групи от три бита - 011 111 101

Осмичен номер - 3 5 7

Десетично число $3x8^{2} + 7x8^{1} + 5x8^{0} = 192 + 56 + 5 = 253$ $5x8^{0} = 192 + 56 + 5 = 253$

Шестнадесетична система

В шестнадесетична цифрова система (base16) се използва набор от шестнадесет цифри (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Буквите от A до F представляват номера от 10 до 15. Шестнадесетичните номера са организирани във възходящ ред до 16.

Конвертирайки числото 321 използвано в предишния пример от шестнадесетичен към десетичен код е

$= (3x16^{2}) + (2x16^{1}) + (1x16^{0}) = (3x256) + (2x16) + (1x1) = 768 + 32 + 1 = 801.$

За да се превърне двоично число в шестнадесетично е нужно да се раздели броя на групи на четири бита. Това е така, защото шестнадесетични цифри от 0 до F

16 -ичен	F F	E E	D D	C C	B B	A Ед-ин	9 9	8 8	7 7	6 6	5 5	4 4	3 3	2 2	1 1	0 0
2 - ичен	1111 1111	1110 1110	1101 1101	1100 1100	1011 1011	1010 1010	1001 1001	1000 1000	0111 0111	0110 0110	0101 0101	0100 0100	0011 0011	0010 0010	0001 0001	0000 0000

представляват четири битови числа в диапазона от 0000 до 1111.

Диаграма по-долу е показано как се конвертира от осем битов кодв двоичен код в шестнадесетичен и тогава в десетичен:

8 битов двоичен номер 10101100

Разделяне на групи от четири бита 1010 1100

Шестнадесетичен номер A C

Десетично число A (10) $x16^{1} + C (12) x16^{0} = 10x16 + 12x1 = 172$

Двоично - кодирана десетична номерация

Двоична кодирана десетична (BCD) номерация са десетично кодирани числа в 4-битов двоичен номер. Цифрите от 0 до 9 са представени от номерата 0000 до 1001.

BCD	9 9	8 8	7 7	6 6	5 5	4 4	3 3	2 2	1 1	0 0
Binary Двоична	1001 1001	1000 1000	0111 0111	0110 0110	0101 0101	0100 0100	0011 0011	0010 0010	0001 0001	0000 0000

Диаграма по-долу показва как се конвертира десетичен номер в BCD.

Десетично число 954

Двоична кодирана Десетична номерация 1001 0101 0100

- **Програмиране**

Всички системи, които са създадени или изследвани чрез дискретни компоненти имат свои функции, определени от връзките им направени между различните компоненти. Те са известни като посветени системи, защото те са изградени да правят само едно нещо. Например виждали сте как D тип флип-флопи могат да бъдат използвани за изграждане на четири битов двоичен брояч, в капаче или щифт регистър, в зависимост от това как се свързват. Веднъж създадени, те ще изпълняват само една от тези задачи. Ние казваме, че са били твърдо свързани. Можем да промените тяхната функция като ги пресвържете.

Програмируеми логически контролери (PLC's)

PLC's се използват за автоматично управление с Асемблер. PLC обикновено съдържа микропроцесор, памет и входни / изходни портове в една стабилна единица. Входовете и изходите определят терминалите, така че входните и изходни устройства могат да бъдат свързани директно с всеки терминал. Тези входови и на изходни връзки се наричат портове. Портовете са разпределени чрез номера така че те да могат да бъдат еднозначно идентифицирани.

С помощта на PLC's написването на програма е подобно на чертаене на електрическа включваща верига. Включващата верига е съставена по стълбовиден диаграмен формат и е известна като реле стълбовидно- логическо програмиране. Ladder logic е език за високо ниво на програмиране ,не е труден, но е доста различен от по-често срещани езици за програмиране, като BASIC или FORTRAN. Ladder logic използва условни твърдения, становища и на FOR NEXT loops, но има някои много съществени разлики.

Обикновено при изграждане на основа или друго високо ниво на програма нещата се случват в ред. Въпреки че, програмата може да съдържа условни твърдения, становища и FOR NEXT loops, всеки команден ред се изпълнява последователно, докато не стигне до при Ladder logic. Няколко неща се случват по едно и също време. Ladder - диаграмите трябва да отговарят на редица правила.

Програмируемата система, като микропроцесор, обаче, може да има своята променлива функция, без да променя която и да е от нейните връзки. За да общуваме с микропроцесора ние трябва да го инструктираме (наречено програмиране), така че той да знае какво трябва да прави и как да го направи. Програмата показва CPU реда, по който тя трябва да изпълнява разнообразните операции, предвидени от неговите инструкции. Различни задачи изискват тези операции да бъдат извършвани по различни начини. Всяка задача ще има по-различна програма написана за нея.

Най-основно ниво на програмиране може да се прави в машинен код. Това е поредица от 0-ли и 1 - ци с които микропроцесора трябва да работи. Пишейки програмите по този начин обаче е дълъг, труден и неизбежно води до грешки, които създава проваляне на програмата. За да се направи програмирането по-разбираемо са разработени програмни езици, трябва да се развиват все повече за да олеснят задачите за написването на програмата и тя да стане по - лесна.

Съществуват два основни вида:

- Ниско ниво или асемблер.
- Високо ниво на езика за програмиране

Ниско ниво на програмиране или асемблерни езици

Програмите за тези езици са написани на мнемоника (памет СПИН). Те са предвидени в набор инструкции, които трябва да се спазват. Мнемониникът за "зареждане на съдържанието на мемори адреса в акумулатор" може да бъде LDA и двоичен код за операцията 1001 1110. Това е много по-удобно написано в шестнадесетичен код (който е 9 E).Преобразуване на тези програми в машинен код се прави от програмата съдържаща се в ROM наричена - *Асемблер.*

Високо ниво на езици за програмиране

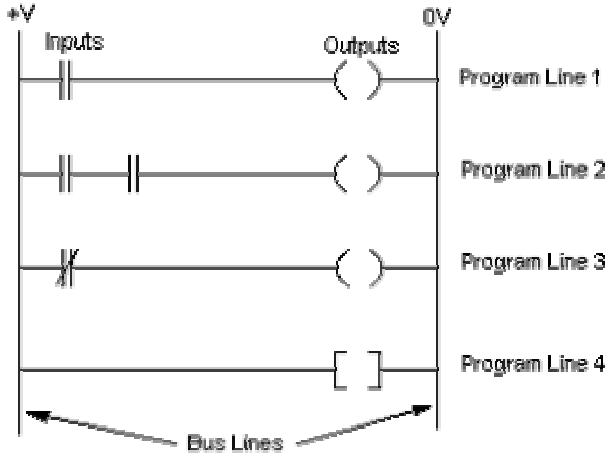
Програмите за тези езици са написани използвайки командите, които са много по-лесни и за разберане и запомняне, тъй като те използват термини като **Load, Add , Compare, Build** и др. Голямият надостатък е, че те се нуждаят от повече памет и компютърно време, защото всяко твърдение трябва да бъде декодирано с номерата на машинния код - инструкции. А компилатор или преводач (подобно на асемблер програмата) прави превода на машиния код. BASIC, LOGO и PASCAL и всички са примери за високо ниво на програмиране.

Схемите са подредени като серия от хоризонтални редове, съдържащи двата входа

(отнасящи се като контакти), и изходи (отнасящи се като намотки).

- Входовете винаги трябва да предхождат изходите и да са под формата на нормално отворени ⊣⊢ и нормално затворени ⊣/⊢ контакти.
- Там винаги трябва да бъде най-малко един изход за всяка линия. Символът за изхода е ( )
- Крайно стъпало по стълбата винаги е крайното твърдение [ ]

Логическият стълбов член е използван.

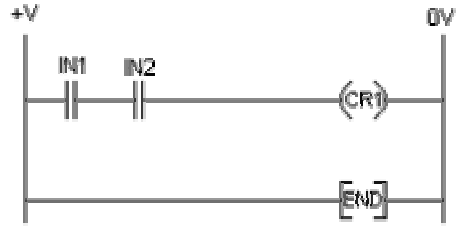


При стъпбовото логическо програмиране всяка инструкция е поставена в отделно памет - място (адрес). В програмата на процесора се съдържа регистър, който сочи как следващата инструкция да бъде заредена от паметта. Когато инструкция е получена от процесора, тя се поставя в инструкционния регистър за декодиране.

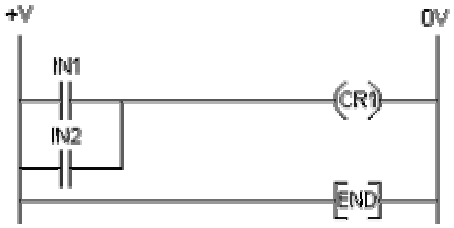
Декодирането и може да доведе до други инструкции четящи се от паметта, или продукция контролища се от процесора.

Входовете обикновено са презададени по X, резултатите обикновено са презададени по Y.

Стълбовидната програма може да бъде преведена на програма, състояща се от инструкции и данни. Някои прости примери са показани по-долу.



Address Адрес	Instruction Инструкция	Data Данни
0 0	LOAD - ТОВАР	IN1 IN1
1 1	AND /И	IN2 IN2
2 2	OUT	CR1 CR1
3 3	END-КРАЙ	



Address Адрес	Instruction Инструкция	Data Данни
0 0	LOAD - ТОВАР	IN1 IN1
1 1	OR- ИЛИ	IN2 IN2
2 2	OUT	CR1 CR1
3 3	END -КРАЙ	

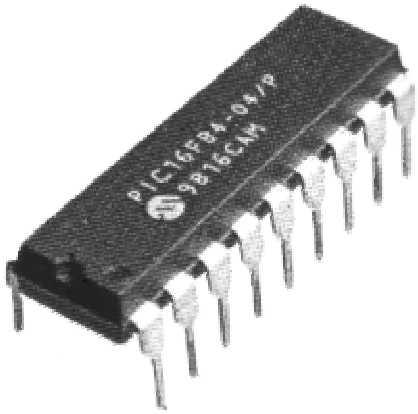
Стълбовите вериги се състоят само от входове и изходи, които са ограничени само до смяна на операциите. За да се позволи на PLC's да се справя с по-усъвършенствани задачи по контрола на производителите се включва набор от специални функции. Тези специални функции се различават за всеки отделен производител, но включват общи опции като броячи, таймери и ключалки.

Програмируеми логически контролери са доста обемисти и скъпи за използване в училище и затова много по-добро решение е да си разработите свой собствен контролер на базата на PIC's .

Програмируеми интерфейс контролери

Програмируем интерфейс контролер или **PIC,** обединява един микропроцесор и ROM.

Те могат да бъдат получени в 8, 18 и 28 щифтови конфигурации, които осигуряват най-различни изходни и цифрови и аналогови суровини.



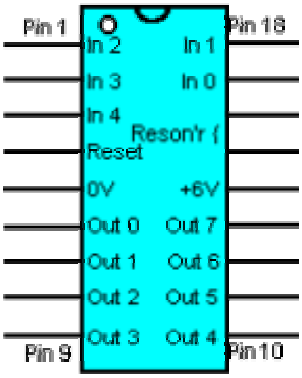
Чиповете използват препрограмирана "флаш памет", която може да бъде писана и пренаписвана с лекота.

Изграждане на работен контролер включва просто свързване на чипа с ход и взаимодействие на входните и на изходните компоненти и добавяне на кондензатор, резонатор и нулиране на лампите.

Най-често използваният PIC 16F84 е показан в дясно. Той е с 18 щифтово устройство, което има 8 изхода и 5 входа.

Свързваща енергия с PIC 16F84

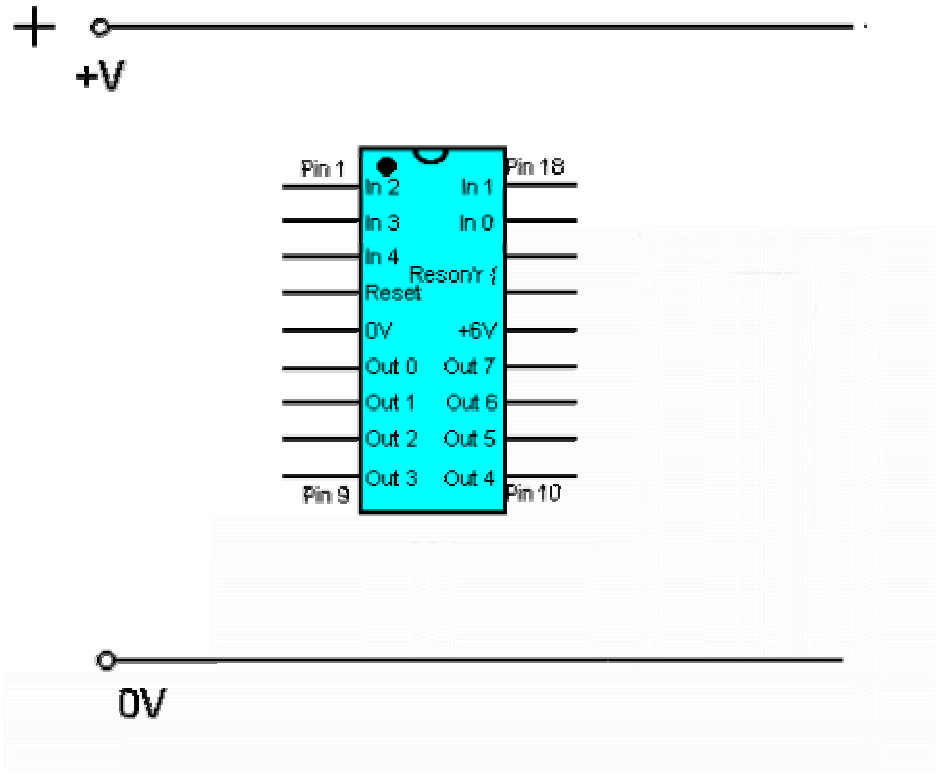
Външно щифтовата диаграма за 16F84 е показана в дясно. 16F84 изисква доставка на 6V напрежение. Това може да бъде предоставено от 4 x AA клетки.



4 MHz - ов керамичен резонатор също трябва да бъде свързан, както е показано по-долу. 16F84 провежда вътрешен кръгов импулс. Резонаторът се използва за регулиране на скоростта на импулса в кръга (4MHz).

Щифт 4 (нулиране), трябва да бъде свързан чрез 4k7 резистор до + V.

А нулиране на съоръжението може да се добави чрез добавяне на бутон между щифт 4 и 0V.

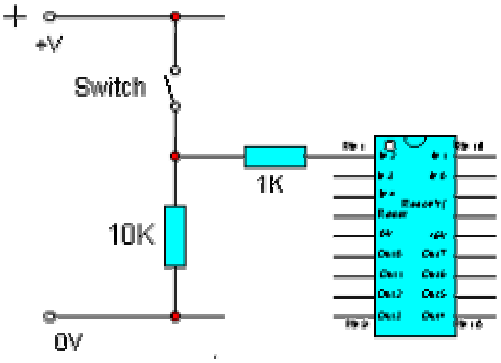

+V
Pin 1
In 2
In 1
Pin 18
In 3
In 0
In 4
Reset
0V
+6V
Out 0
Out 7
Out 1
Out 6
Out 2
Out 5
Out 3
Out 4
Pin 9
Pin 10
0V

Взаимодействие на входа и на изхода на устройството

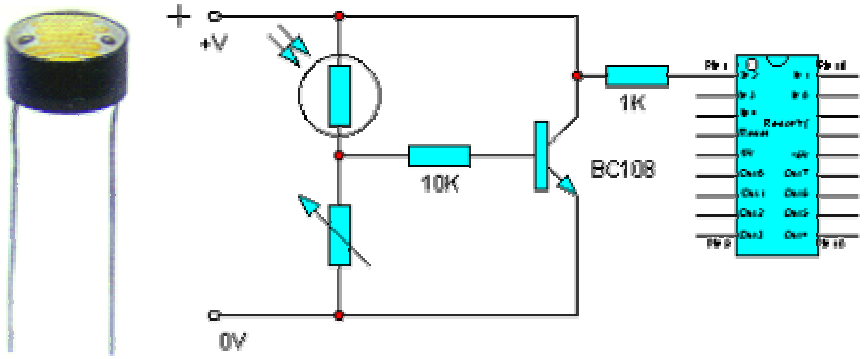
Условно внимание се отделя, съответно на напрежението и настоящите нива, цифрови сензори или преобразуватели, можещи да бъдат пряко свързани с въвеждане и извеждане на портовете.

Най-често дигитални сензори са превключвателите.

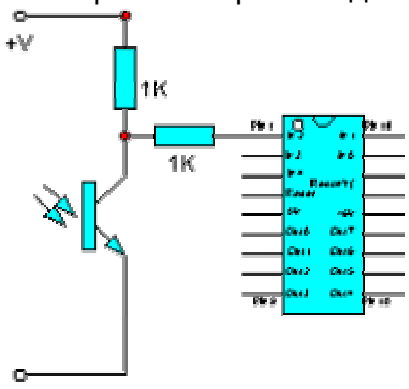
Микро-превключватели, превключватели Рийд, Tilt превключватели и Push ключове и превключватели - всички те могат да бъдат пряко свързани с въвеждане на всеки щифт код, както е показано. 10 k резистор предпазва кратка електрическа верига и резистор 1k защитава въвежден щифт. В тези случаи цифроя вход ще се премести от 0 до 1, когато ключа е натиснат.



Въпреки че, 16F84 не разполага с аналогов щифтов вход, един аналогов сензор може да се използва, ако сензора е свързан за потенциален делител и транзистор, както е показано.



Фототрансмитер може да се използва за преминаване към директно въвеждане , като



в двата случая, входовете ще се преместят от 1 до 0, когато смяното ниво е достигнато.

Повечето съвременни microcontrollers използват това, което се нарича **"флаш" програмиране** чрез EEPROM (ел. изтриваеми програмируема памет само за четене). Това означава, че устройството трябва да бъде в състояние да бъде повторно програмирано над 10000 пъти.

Това е важно, защото вие ще получите рядка микроконтролерна програма използваема за първо време и вие неизбежно ще искате да я промените. Ако все пак направите грешка, PIC може просто да бъдат препрограмиран с нов код. Програмата ще остане в паметта на PIC, докато не бъде повторно препрограмиран.

. Микроконтролерите използват указанията **в** двоичен (серия от 0 и 1's) често се наричан **машинен код.** Този "код" не означава нищо, но не и за квалифициран програмист.

В промишлеността микроконтролерите се програмират използвайки "компилатор" или "асемблер". Това е програмен език, който е малко по-лесен за разбиране, но и безполезен ако не сте взели А- ниво на компютърно програмиране за да можете да го разберете. За повечето приложения в училище, сложността на тези езици прави невъзможно използването им (или най-малкото нереалистично), за повечето приложения.

В KS4, че може да преподава разработване на програмите език, наречен **BASIC,** като по-голямата част от учениците се справят с доста добре с него.. За KS3, обаче , дори и изучаването на BASIC е нереаслистично във времето в което живеем.

За повечето ученици много по-прост и по-лесен за разбиране и програмиране на техника е да се използва **програмен редактор,** като например този, снабдени с технологията „Крокодил". Повечето програмни редактори използват **flowchart** - подходящ за дизайнерските програми. Редактиращият софтуер превръща „flowchart" в BASIC или Асемблер преди микроконтролера да е програмиран.

PIC based

PIC Logicator например изисква програматор, в който е поставен микроконтролер за програмиране.

PICAXE система използва специални микроконтролери с малка програма, наречена „bootstrap „програма- постоянно записана в неговата памет, която позволява да се програмира микроконтролера, докато той се инсталиран във вашия проект.

Chip factory система е с ниска струваща PIC програмна единица, която работи без компютър. Тя използва своя основен език за програмиране, и е

преносима.

PIC Logicator и Chip factory и двете обхващат диапазон от 28 и 18 щифтови микроконтолера. В PICAXE система осигурява подкрепа за тяхните собствени 8, 18 и 28 щифтови чипа. Всичко това го прави много трудно да обясня точно как всеки микроконтролер може да бъде използван.
Няколко нови Микроконтролера наскоро бяха реализирани. Те са много по-евтини, отколкото по-старите устройства, които позволят допълнителни функции на по-ниска цена. Така например 18-щифтов 16F627 е много по-евтин от стария 16F84A, разполагащ с два аналогови входа, както и въвеждане на две допълнителни щифта,а също има вътрешен резонатор.

Най-често използваните типове са:

- 8 PIN - 12F629 (2 входа, 4 изхода)
- 18 PIN - 16F627(4 входа, 2 аналогови, 8 изхода, 1 звук)
- 18 PIN - 16F84A (4 входа, 8 изхода, 1 звук)
- 18 PIN - 16F818 (2 входа, 2 аналогови, 8 изхода, 1 звук)
- 28 PIN - 16F872 (8 входа, 4 аналогови, 8 изхода, 1 звук)