MEMEC SCANDINAVIA AB

A Memec International Components Group Company

MEMEC SCANDINAVIA AB

A Memec International Components Group Company

Förord

Denna nybörjarguide är avsedd att ge en grundläggande kunskap om hur man kan använda och programmera Microchip's PIC-mikrokontroller. För att kunna tillgodose sig denna nybörjarguide bör en grundläggande kunskap om digitalteknik (binära och hexadesimala talsystemet samt logiska grindar etc) redan vara lagd. Microchips PIC-mikrokontroller finns i ett hundratal olika utföranden och har därför också blivit mycket populära. Då en studie på en PIC är avslutad är det mycket lätt att anpassa de program som skrivits till den till en annan PIC. Under utvecklingsarbetet gäller det att hålla nere kostnaderna och en av dessa kostnadsbesparingar kan vara att mjukvaran skall vara lätt att lära. Denna bok lämpar sig ypperlig till att användas i skolor som kursmateriel. Boken får kopieras på det villkoret att kopieringen ej sker på kommersiella grunder och att innehållet i boken ej ändras eller kopieras helt eller delvis till annan publikationsplats. Om kopiering skall ske till anan publikationsplats skall ett särskilt tillstånd från Memec Scandinavia AB finnas.
Observera att det kan förekomma fel i denna skrift och att Memec Scandinavia AB avsvär sig alla tänkbara och otänkbara skadeståndskrav i avseende på dessa felaktigheter. Denna bok är skriven av Niklas Wennerstrand i samarbete med Patrik Särenfors och Johan Wesslén som har hjälp till med korrekturläsning

Microchip är en mikrokontrollertillverkare som har vuxit sig mycket stor på några få år. De började med mikrokontroller så sent som 1989 och är i dags dato världens näst största mikrokontrollertillverkare för 8-bit kontroller. Microchip är även tillverkare av minneskretsar, säkerhetskretsar (Keeloq) för krypterad identifikation och periferikretsar som t.ex. analog till digital omvandlare.

Memec Scandinavia AB

Memec Scandinavia AB bildades 1991 som ett högteknologiskt agenturföretag. Vi förser den nordiska elektronikindustrin med avancerade komponenter, framförallt inom halvledarområdet. Vi är idag etablerade i Stockholm-Gärdet. Vi sätter teknisk kunskap och kvalitet i förgrunden. Därför har vår personal lång erfarenhet när det gäller försäljning av elektronikprodukter. Våra tekniker har utbildning i och erfarenhet från elektronikkonstruktion.
Vi ingår som dotterbolag till Tyska Veba i deras elektronikdivision "Memec International Components Group". Vår koncern omsätter inom elektronikområdet ca: 8 Miljarder kronor och vi är 1600 anställda med agenturverksamhet i Europa, USA och Asien.
I Thame strax nordväst om London, har vi vårt lager på ca: 2500m². Vårt lager och logistikhantering är ISO9002 certifierad.
För våra kunder erbjuder vi:
Alltid tekniskt stöd via telefon.
Program och utvecklingshjälp.
Tekniskt stöd på fältet.
Programmeringscenter med möjlighet till "Tape and Reel"

Memec Scandinavia AB
Sehlstedtsgatan 6
115 28 Stockholm

Tel vx: 08-459 79 00
Tel fax:08-459 79 99
Hemsida: www.memec.se

Vad kan man göra med en PIC?

Det finns PIC'ar som har ett fåtal I/O och PIC'ar som har upp till 50 st I/O. Om ADC (Analog to Digital Conversion) är det som söks så finns det PIC'ar som stöder detta. Det finns även PIC'ar med I2C eller SPI-buss för seriekommunikation.
För att få en inblick i vad respektive PIC kan göra rekommenderar jag att hämta den senaste informationen på ***Microchips hemsida: (http:/www.microchip.com***). Där kan man även hämta hem en "product line card" (produktöversikt) där övergripande information om respektive PIC kan utläsas.
Microchips hemsida är en stor informationskälla för den som vill veta mer. Där finns databöcker, datablad, programvaror, senaste information om nya produkter och intressanta applikationsexempel.
Applikationsexemplen innehåller programkod och förklaringar och är en bra idekälla för vad PIC'en kan användas till.

Vanligt förekommande frågor

Vad är skillnaden mellan en mikroprocessor och en mikrokontroller?
En mikroprocessor måste använda yttre så kallade periferikretsar för att kunna utföra något nyttigt medan en mikrokontroller arbetar ensam (stand alone) och har minne och I/O-portar integrerat i samma kapsel. Detta gör att konstruktionslösningen med en mikrokontroller blir mindre och billigare.

Vad är RISC och CISC?
Microchips PIC-mikrokontroller är av en typ man benämner RISC. RISC står för Reduced Instruction Set Computer (reducerad/förenklad instruktionsuppsättning,dator) (dator benämningen kommer ursprungligen från stora datorsystem). Då man talar om RISC avser man oftast att arkitekturen är av typen Harward arkitektur. Harward akitekturen är en bidragande orsak till att RISC är snabbare än CISC.
CISC betyder Complex Instruction Set Computer (stor/komplex Instruktionsuppsättning, dator) och kännetecknas av att det finns många instruktioner men framför allt har denna typ av mikrokontroller oftast en arkitektur av typen Von Neuman-arkitektur.

Vad innebär Harward/Von Neuman-arkitektur ?
Den traditionella Von Neuman-arkitekturen kännetecknas av att data och program delar samma buss vilket gör bussen långsammare än om man har en separat databuss och en separat instruktion/programsbuss. Harward-arkitekturen har separata bussar vilket gör att exekveringen går mycket snabbare.

Vad är interrupt?
Interrupt är precis vad det låter som, nämligen ett avbrott. Många yttre signaler som kommer till en processor kommer inte alltid med jämna mellanrum och därför kan man inte förutse när de skall komma in i programmflödet. Hårdvarustyrd interrupt är det som man vanligen kallar interrupt, men man kan även använda så kallad polling för att se om något speciellt har inträffat. ”Polling” kan missa den inträffade händelsen om programmet utför något annat under samma tid som händelsen sker. ”Polling” går ut på att du i ditt program t.ex. läser värdet på en I/O pinne då och då. Hårdvarustyrd interrupt avbryter hårdvarumässigt programflödet för att utföra någon speciell åtgärd. Låt oss ta en liknelse: Om du väntar besök kan du då och då öppna ytterdörren för att se om din gäst har kommit för att du skall kunna vidtaga den ceremoniella välkomstritualen (polling). Alternativt kan du installera en dörrklocka så att du hör när din gäst har kommit, varpå du utför den ceremoniella välkomstritualen (hårdvarumässig interrupt). Båda sätten är naturligtvis bra men den hårdvarumässiga metoden är betydligt enklare och kräver mindre processorkraft. Tänk om du i det första fallet inte kollar om din gäst står utanför dörren tillräckligt ofta. Då skulle du kanske missa din gäst och han/hon skulle gå hem utan att du ens vetat om att han/hon faktiskt varit hemma hos dig.

Vad är I/O?
I/O står för Input/Output dvs ingång eller utgång. För att kunna kommunicera med omvärlden behövs det ingångar och utgångar från mikrokontrollern. Exempel på ingångar är anslutningar till tryckknappar. Exempel på utgångar är anslutningar till lysdioder eller styrning av reläer.

Vad är ett register?
Ett register håller ett binärt ord. Ett register är ofta hårdarumässigt knutet till andra delar i CPU’n. Beroende på vad ett register innehåller sätts olika bitar i andra register. Om tex. ett register får värdet noll efter en logisk operation med ett annat register indikeras detta i statusregistret genom att Z biten sätts till en etta. Register kan i vissa fall liknas vid snabba RAM. Till skillnad från externa RAM-minnen ligger registren på samma chip som själva mikrokontrollern och har därmed en mycket snabbare accesstid än externa RAM.

Vad är w-registret?
På alla PIC’ar finns det så kallade ”working-registeret” (arbetsregistret). W-registret är nära knuten till ALU’n (Aritmetic Logic Unit) där beräkningar utförs .För att utföra en beräkning mellan ett register och något annat måste detta andra vara lagrat i W-registret. W-registret är för andra processorfamiljer att liknas vid ackumulatorn. Det som händer i w-registret påverkar statusregistret.
(det som händer i andra register kan också påverka statusregistret.)

Vad är RAM, ROM, EPROM, EEPROM och FLASH?

RAM: Random Access Memory
Detta är ett flyktigt minne som det går att skriva till och läsa från mycket fort.

ROM: Read Only Memory
Detta är ett icke flyktigt minne som tillverkas på fabrik genom så kallad maskning av minnet. Man kan endast läsa från detta minne.

EPROM: Electrical Programable Read Only Memory
Detta minne kan man programmera relativt långsamt med en programmeringsspänning som ligger runt 12V. Om dessa minnen är försedda med ett litet fönster kan man radera dem med UV-ljus.

OTP One Time Programable
OTP är samma sak som EPROM. Denna benämning förekommer ofta i samband med mikrokontroller.

EEPROM: Electrical Erasable Programable Read Only Memory
Fungerar som EPROM men de raderas ej med UV-ljus utan med hjälp av en spänning.

FLASH: FLASH (kort och gott blixt)
Fungerar som EEPROM men raderas i moduler (stora sjok), dvs de raderas mycket snabbare än EEPROM.

Vad är mjukvara?

Mjukvara är direktiv till mikrokontrollern om hur den skall fungera. Mjukvaran programmeras av dig eller någon annan eminent programmerare.

Vad är hårdvara?

Hårdvara är det som du kan få utlopp för din aggression på. T.ex. kasta i golvet eller sparka på.

Vad är en programmerare?

En programmerare kan vara av två typer. Den ena är av kött och blod tillbringande sin tid framför datorer och knappar in program i datorn. Den andra typen är en elektronisk mackapär för inmatning av maskinkod till elektroniska kretsar.

Vad är Source File (källfil)?

Source file eller källfil är den filen som innehåller ditt program och där programmet är skrivet i ett programmeringsspråk som tex assembler, C, Basic eller Pascal. Denna fil kompileras senare till ett maskinkodsformat.

Vad är en kompilator (assembler)?

En kompilator (assembler) assemblerar (kompilerar) källkoden till en hex-fil med sufixet .HEX. Under kompileringen utförs feltester av källkoden. Om det förekommer syntax fel i programmet kommer kompilatorn att upplysa om detta.

Vad är en list file?

En list-file skapas av kompilatorn och innehåller förutom programraderna med kommentarer även den maskinkod som genereras. Maskinkoden presenteras rad för rad och den finns även representerad som hexadecimala tal.

Vad är en multiplexer?

En multiplexer är en typ av elektronisk växel som växlar mellan olika in och utgångar. Genom ett binärt tal anges vilken ingång eller utgång som skall väljas.

Vad är ADC (Analog to Digital Converter)?

En ADC eller analog till digital omvandlare är en modul som kan omvandla en analog spänning till en binär representation. Då en fysisk storhet (höjd, vikt, temperatur, hastighet etc) skall mätas behöver denna fysiska storhet omvandlas till ettor och nollor så att mikrokontrollern kan förstå och bearbeta resultatet.

Vad är brown out?

Brown out är när spänningsmatningen får en spänningssänkning under en kort stund. Detta kan påverka innehåll i register och på så sätt generera fel i programmet. En del PIC'ar har ett skydd mot brown out. Skyddet består i att om en brown out upptäcks utför PIC'en en reset varvid programmet startar om när spänningen återvänder. Efter resetten initieras register och programmet startar om från början. Denna reset kallas för brown out reset (BOR).

Vad är sleep mode?

För att spara in på strömförbrukningen kan vissa delar i mikrokontrollern stängas av då de inte behövs och därmed spara in på strömförbrukningen (ner till ca: 1uA). Detta görs genom att sätta kontrollern i sleep mode. Under sleepmode behåller alla register sina värden och därmed kan programmet fortlöpa vid ett senare tillfälle som om ingenting hade hänt. För att väcka kontrollern används avbrott av olika slag tex interrupt on pin, interrupt on ADC complete etc. Denna möjlighet gör att PIC'en är ideal att använda i batteridrivna applikationer. Ett ytterligare sätt att spara ström är att välja en så låg oscillatorfrekvens som möjligt.

Vad är ESD?

ESD står för Electro-Static-Discharge (elektrostatisk urladdning). ESD kan skada elektronik då den kan uppnå mycket höga spänningar . ESD uppstår t.ex. då två ytor gnids mot varandra och de två materialen kommer från olika spänningsserier (fysik/kemi överkurs). ESD kan uppstå då en tröja gnids mot en människokropp. ESD orsakar fel på elektronik då personer kommer i beröring med elektroniken. För att skydda sig mot ESD skall speciella ESD-armledsband användas. Armledsbandet leder bort laddning till markjord. För att en person skall kunna uppfatta att den är uppladdad måste personen i fråga vara uppladdad till ca 15kV, vilket för länge sedan hade förstört elektroniken om den vidrörts.

Vad är Watchdog timer (WDT)?

Watchdog timer är en elektronisk ”vakthund” som utför en reset på mikrokontrollern då en viss tid har passerat utan att WDT har uppdaterats. WDT uppdateras genom att nolla WDT registret med jämna mellanrum. WDT består i själva verket av ett register som räknas upp med ett visst intervall och när detta register slår om från FFh till 0h genereras en reset. Detta används som ett skydd mot att programmet hänger sig (t.ex. fastnar i en oändlig loop).

Vad är en emulator?

En emulator är en elektronisk enhet som oftast ansluts från en dator till ditt laborationskretskort. Med denna emulator kan en riktig mikrokontroller emuleras (efterliknas). Från emulatorn går det en anslutningskabel som ansluts till den plats där mikrokontrollern skall sitta.
Det som är bra med en emulator är att man får en effektiv överblick på vad som händer i olika register och dylikt då man arbetar med PIC'en i en verklig miljö. I vissa emulatorer kan yttre enheter som samverkar med mikrokontrollern presenteras och tid sparas genom att slippa programmera om mikrokontrollern varje gång som en liten ändring utförs i programmet. Med en emulator kan programmet stegas fram. Programmet kan även köras i realtid, dvs precis som en riktig mikrokontroller skulle ha uppfört sig. En annan stor fördel med en emulator är att det blir lättare att spåra ”buggar” (fel) i programmet.
Genom att sätta vissa villkor som gör att programmet avbryts, t.ex. när programmet passerar en viss programadress kan konstruktören se hur programmet har uppfört sig innan avbrottet uppstod. Därmed kan konstruktören lättare felsöka och testa små delar i sitt program.

Vad är en simulator?

En simulator är ett program som efterliknar en processor. I en simulator kan programflödet simuleras i mikrokontrollern (detta görs helt och hållet i PC-datorn). Diverse adressinnehåll och teoretiska tider på kodexekveringen i olika delar av programmet kan avläsas från PC-datorn etc. En simulator kopplas inte till något kretskort utan ger en helt teoretisk simulering av verkligheten. Beakta att teori inte alltid är så nära verkligheten som man ofta skulle vilja.

Vad är USART?

USART står för (Universal Synchronous Asynchronous Receiver Transmitter). Med en USART kan data skickas och tas emot med hjälp av seriell kommunikation t.ex. RS232. RS232 är den standard som man bl.a. använder för kommunikation mellan datorns com-port och ett modem.
Obs det behövs även lite kringkomponenter för att få de spänningar som RS232 standarden kräver

Vad är I2C?

I2C är en standard för seriell kommunikation som använder sig av två ledare. I2C uppfanns av Philips och användes från början till att kommunicera mellan olika elektronikkomponenter i TV, radio och stereo-apparater. Denna standard passar bra för kommunikation mellan kretsar på ett och samma kretskort. Önskas längre avstånd och högre hastighet bör I2C inte användas.

Vad är SPI?

SPI är en seriell överföringsstandard som använder sig av tre ledare. SPI uppfanns av Motorola.

Litteratur som kan vara värd att nämna.

PICmicro Mid-Range MCU Family Reference Manual

Denna bok finns gratis på Microchips hemsida och ger mycket bra beskrivningar på hur de olika delarna i en PIC fungerar. Boken ger förutom beskrivningar även programexempel. Indelningen av boken består av separata kapitel om t.ex. Interrupt, I/O portar, minne, ADC, USART, timers, PWM, CCP och mycket annat matnyttigt. Efter varje kapitel finns det hänvisningar till applikationsexempel som relaterar till de behandlade ämnena.

Embedded control hanbook

Denna bok innehåller applikationsexempel. Dessa applikationsexempel kan hämtas gratis på Microchips hemsida.

Databok på respektive PIC

Databok på den PIC som du tänker använda är förstås A och O. Databöcker finns att hämta hem gratis från Microchips hemsida.

MPLAB IDE User's Guide

Detta är manualen som beskriver hur MPLAB fungerar. Denna manual finns med i PICSTART PLUS paketet och den senaste versionen finns även att hämta gratis på Microchips hemsida.

MPASM Assembler User's Guide

Denna manual behandlar hur assemblern fungerar. Denna manual finns med i PICSTART PLUS paketet och den senaste versionen finns även att hämta gratis på Microchips hemsida.

In-Circuit Serial Programming Guide (ICSP).

Denna manual beskriver hur man programmerar PIC'en "In Circuit" (direkt på kretskortet). Denna manual finns att hämta gratis på Microchips hemsida.

Errata dokument

På vissa PIC'ar kan det förekomma fel i kislet eller i databoken för respektive PIC. Errata beskriver dessa fel och hur man kommer förbi dem. Errata finns att hämta på Microchips hemsida. Du bör alltid titta på errata-blad innan du börjar din konstruktion.

Mikrodatorteknik (riskfylld elektronik)

Denna bok är skriven på svenska av Lars Bengtsson. Boken innehåller allt från det mest enkla programmet till mer avancerade program som använder ADC, USART timers och LCD displayer kopplade till PIC'en. Den del som beskriver själva MPLAB är dock ej aktuell idag men boken lämpar sig ändå till undervisningsmaterial då programexemplen är mycket gångbara. Boken säljs av bl.a. Memec Scandinavia AB.

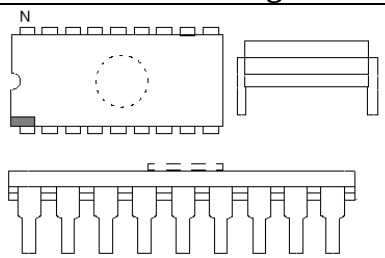
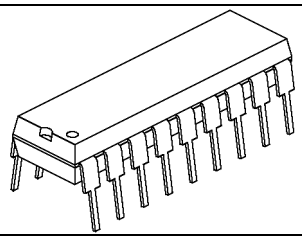
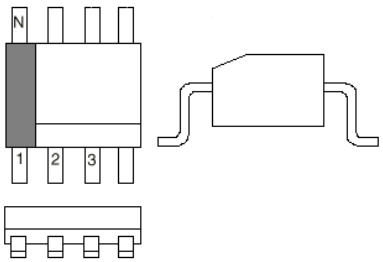
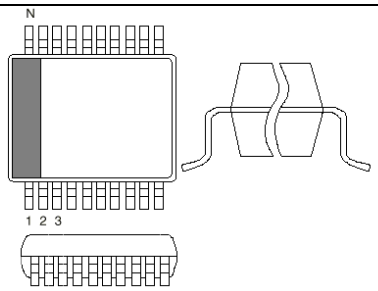
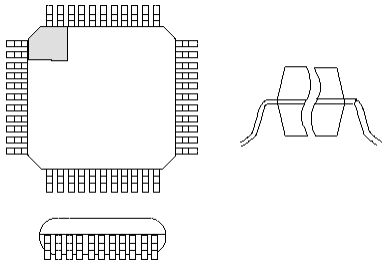
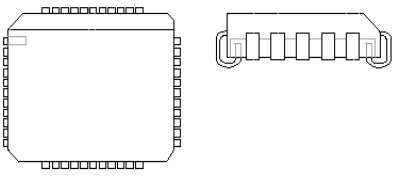
A Beginners Guide To The Microchip PIC

En bok skriven av Nigel Gardner som innehåller många grundläggande programexempel. Säljs av bl.a. Memec Scandinavia AB. Boken är på engelska.

MEMEC SCANDINAVIA AB

A Memec International Components Group Company

Lathund för Microchip kapslingar

Vanlig benämning	Microchip benämning	Bild beskrivning
Windowed ceramic dual in line WDIL	JW	
Plastic Dual In Line PDIL	P	
Plastic Surface Mount SOIC	SO	
Small outline SOIC SSOP	SS	
Plastic Quad Flatpack PQFP Plastic Thin Quad Flatpack TQFP	PQ PT eller TQ	
Plastic Leaded Chip Carrier PLCC	L	

Vi studerar PIC16F84

PIC16F84 är den PIC som vi ska bekanta oss med. Denna PIC får man med då man köper PICSTART PLUS, utvecklingsprogrammerare. PIC16F84 är elektriskt omprogrammerbar (EEPROM) vilket gör den ideal att använda i utbildningssammanhang eller labbändamål.
PIC16F84 har 1024x14 ord minne (flashminne), 64 bytes stort EEPROM (för lagring av ej flyktig data), 68 byte registerminne, maxhastigheten på oscilatorn är 20MHz, 13 I/O portar, timer plus en watchdogtimer och PIC16F84 har även interruptmöjligheter. Det finns möjlighet att programmera den på kretskortet (In Circuit Serial Programming, ICSP).
När en konstruktion med en ny PIC startar bör konstruktören titta på ***Microchips hemsida: (http:/www.microchip.com***) för att hämta hem datablad på PIC'en. Detta råder jag Er att göra. Det finns även errata på en del PIC'ar. Errata beskriver en del problem och hur man undviker dessa problem.
Ett frågetecken som brukar dyka upp är varför minnet är benämnt som 1024x14 när PIC16F84 är en 8 bit kontroller. Detta beror på att de övre mest signifikanta bitarna anger kommandot och det åtta minst signifikanta bitarna anger data till kommandot. Detta är i motsats till andra mikrokontroller där man är van vid att kommandot tar en eller flera byte och att data tar en byte. Detta betyder att det går åt fler ord/bytes i en vanlig kontroller än vad som går åt i en PIC16F84. Att kommando och data finns i samma ord innebär att de hämtas samtidigt in till ALU'n vilket naturligtvis snabbar upp exekveringen. Om man skall jämföra en PIC med ett medeltal av andra mikrokontroller då PIC'en har 1024x14 word motsvaras detta av 1792 bytes minne hos andra mikrokontroller, dvs PIC'en har mer minne än vad man först tror. Den andra minnesbenämning brukar också vara nämnd i "product line card" (produktöversikten). Denna kodkompakthet åstadkommes med den arkitektur som PICen är uppbyggd med (Harward arkitektur).

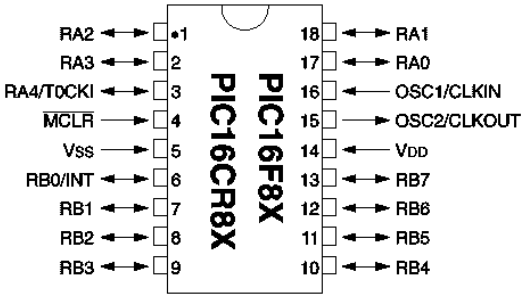


Fig 1.1 Pinkonfigurationen för PIC16F84

PIC16F84 har två portar: port A och port B. Port A har fem I/O-pinnar och port B har åtta I/O-pinnar. Pilarna i fig 1.1 visar riktningarna på respektive pinne. MCLR(master clear) är detsamma som RESET på andra processorfamiljer. Om MCLR sätts låg utförs en reset på processorn. T0CKI används som pulsräknare, dvs ett internt register (T0) som räknas upp med ett. Ur T0 kan sedan utläsas hur många yttre pulser som har registrerats på ingången. Till Vss kopplas jord och till Vdd kopplas t.ex. +5V. INT är en ingång som kan konfigureras som interrupt på positiv flank eller interrupt på negativ flank.
Utgångarna kan driva en last på 20mA och sänka 25mA. Men kom ihåg att inte lasta mer än PIC'ens maximala strömförbrukning på respektive port eller totala strömförbrukning. Data finns listade under absolute maximum rating i respektive databok.

Port B

På PIC16F84 kan port B konfigureras så att interna pull-up resistorer kopplas till ingångarna. Detta är mycket praktiskt för att eliminera behovet av yttre resistorer. Anledningen till att använda pull-up resistorer på ingångarna är att man vill ha ett säkert värde på ingångarna. Om pull-up resistorer ej skulle finnas skulle ingångarna ligga och "fladdra". Denna inställning görs i option-registret. I databöcker kallas denna resistor för "weak pull-up resistor". Med weak(svag) avses att det går en svag ström genom resistorn vilket i sin tur innebär att resistorn är relativt högohmig.

Port A

Pinne RA4 på port A kan när den är konfigurerad som ingång användas som en pulsräknaringång till T0CKL. Den är utförd med schmitt trigger ingång. Om RA4 är konfigurerad som utgång måste den strömmatas med tex en pull-up resistor. Detta beror på att RA4 internt är kopplad som open collector.

> Då man utför en "read modify write" **på en port** skall man lägga in en nop-instruktion mellan de två instruktionerna om instruktionerna ligger i direkt följd efter varandra.
> Read modify write instruktioner är instruktioner som på något sätt använder en port.
> Tex : bcf, bsf, btfsc eller btfss

Reset

En PIC16F84 skiljer mellan olika typer av reset nämligen:
Power-on Reset (POR).
MCLR' reset vid normal uppstart.
MCLR' reset under sleep-mode.
Watchdog timer (WDT) reset under normal gång.
Watchdog timer (WDT) reset vid uppvakning från sleep.

Olika resetter påverkar GPR-registren (generella register) på olika sätt. Vid POR vet man ej vad GPR-registren kommer att innehålla. Vid övriga resetter behåller GPR-registren sina värden. Vid olika resetter påverkas statusregistret bit TO' och PD' som kan användas för att utföra nödvändiga åtgärder i programvaran (tex felmeddelanden).

Power-on Reset (POR)

En power-on reset genereras i chippet då en detektering av att Vdd stiger i området 1.2V till 1.7V. För att utnyttja POR, anslut bara MCLR' direkt eller via en resistor till Vdd. Detta eliminerar behovet av en yttre RC komponent som vanligtvis behövs för att skapa en POR. Det krävs dock en viss stigtid på Vdd för att detta skall fungera som tänkt (se databok för elektriska specifikationer)

En varning: om konstruktionen skall förberedas för programmering direkt på kretskortet ICSP (In Circuit Serial Programming) bör en skyddsdiod kopplas på Vpp/MCLR'-ingången. Detta beror på att programmeringsspänningen är upp emot 13V och kan därför förstöra kringkomponenter.

När mikrokontrollern startar normalt efter en reset, måste spänning och oscillatorfrekvens vara stabila för att garantera att mikrokontrollern startar upp som den ska. Om dessa villkor ej är uppfyllda skall resetten ligga kvar längre så att spänningsstabilisering kan uppnås. För ytterligare information (se applikationsnote **AN607**) "Power-up Trouble Shooting".
POR genererar ej någon reset om Vdd bara skulle minska en del under en kort tid. (Brow-out reset är däremot lämpad för det.)

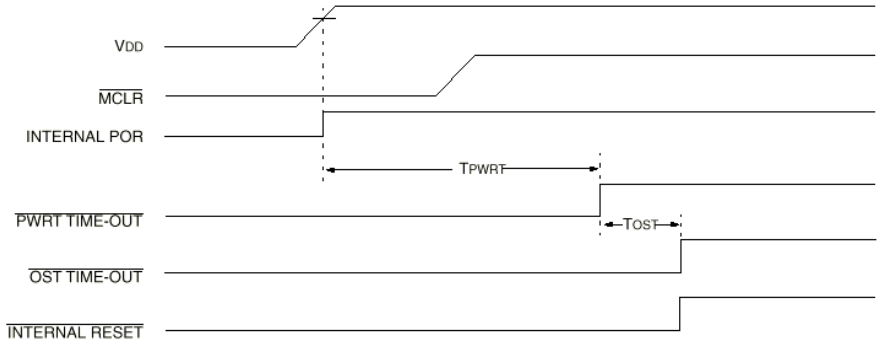
Power-up Timer (PWRT)

Power-up timer (PWRT) genererar en 72 ms nominell tidsfördröjning eller time-out (Tpwrt) från det att POR har genererats (se graferna på nästa sida). PWRT styrs av en inre RC-oscillator och chippet ligger i reset så länge som PWRT är aktiv. Detta ger att Vdd får tillräckligt god tid på sig att stiga till sitt nominella värde (möjligt undantag synes i sista figuren på nästa sida).
En konfigurationsbit PWRTE kan slå av och på funktionen med PWRT (se figur 1 och 2 på nästa sida för exempel på de olika alternativen). Tpwrt(Tiden på power up timern) varierar beroende på Vdd, temperatur och tillverkningsvariationer.

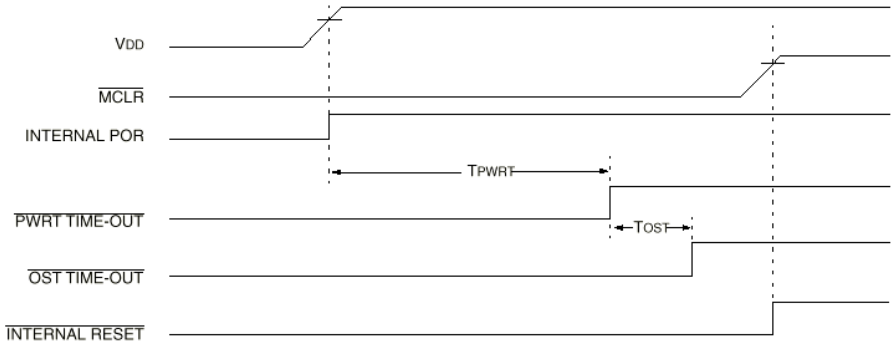
Oscillator Start-up Timer (OST)

Oscilator start-up timer tillhandahåller en fördröjning som är 1024st godkända klockpulser lång (från OSC1 ingången). Denna fördröjning startar efter det att PWRT har avslutats (se exempel 1 och 2 på nästa sida). Detta garanterar att kristalloscillatorn hinner att stabilisera sig innan kontrollern går igång. OST finns tillgänglig endast för XT, LP och HS oscillator-mode och endast vid en POR eller uppvaknande från sleep.
OmVdd stiger mycket långsamt, finns en möjlighet att Tpwrt time out och Tost time out avslutas innan Vdd har uppnått sitt slutliga värde. I detta fall (se sista exempel 4 på nästa sida) behövs en yttre power-on reset krets anslutas till PIC'en (se nedan)

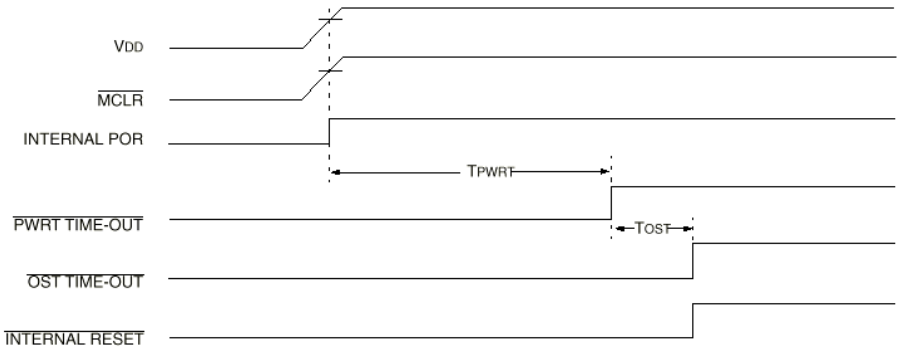
Time-out sekvens vid power-up (MCLR' ej ansluten till Vdd):exempel 1



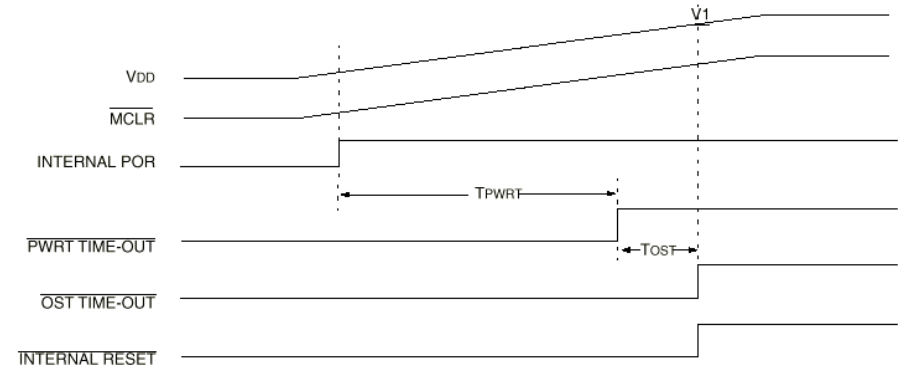
Time-out sekvens vid power-up (MCLR' ej ansluten till Vdd):exempel 2



Time-out sekvens vid power-up (MCLR' ansluten till Vdd) snabb Vdd stigtid: exempel 3



Time-out sekvens vid power-up (MCLR' ansluten till Vdd) långsam Vdd stigtid: exempel 4



När Vdd stiger mycket långsamt, finns en möjlighet att Tpwrt time-out och Tost time-out inträffar innan Vdd har uppnått sitt slutliga värde. I detta exempel kommer PIC'en att utföra en korrekt reset men endast om V1≥Vdd min.

Graferna på föregående sida visar :
PWRT startas efter det att POR har avslutats. Efter detta startar OST. Den totala time out tiden kommer att variera beroende på oscillator, konfiguration och PWRTE konfigurations-bit-status. Till exempel vid RC-mode är PWRT ej inkopplad.

Time out och power down status bitar (TO'/PD').

TO', Time Out biten anger om watchdog timern har nått en time out. En nolla indikerar att WDT har nått en time out.
PD', Power Down biten anger hur PIC'en har har utfört en power down (se statusregisteret).

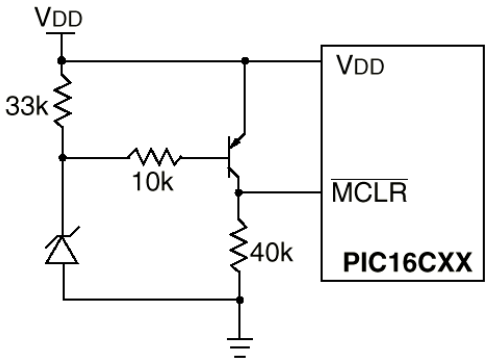
Tabellen nedan visar olika kombinationer på TO' och PD' bitarna i status registret:

TO'	PD'	Resulterande inställning
1	1	Power-on Reset
0	X	Förbjudet, TO' sättes vid POR'
X	0	Förbjudet, PD' sättes vid POR'
0	1	WDT reset under normal gång
0	0	WDT uppvaknande
1	1	MCLR' reset under normal gång
1	0	MCLR' reset under sleep eller interrupt uppvaknande från sleep

Reset vid brown out.

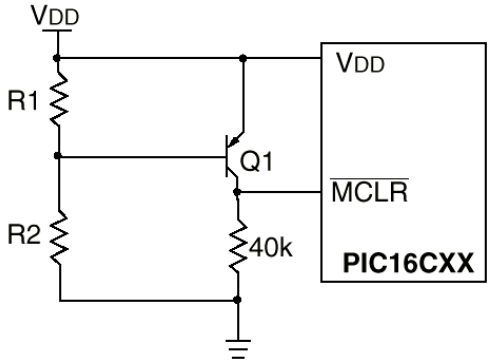
"Brown out" är ett tillstånd då Vdd av någon orsak under en kort stund sjunker under kretsens specificerade minspänning av matningsspänning Detta kan orsaka fel i registren men att kretsen återhämtar sig utan att utföra POR.
För att eliminera att en "brown out" orsakar fel i kretsen kan en brown-out krets lösa detta enligt nedan.
(vissa PIC'ar har en "brownout krets" inbyggd i sig, dock ej 16F84'an)

Brown out krets 1



Denna krets kommer att aktiveras då Vdd går under (Vz+0.7V) där Vz är Zenerdiod spänningen.

Brown out krets 2



Denna krets är billigare men mindre noggrann än krets 1. Transistor Q1 stryper då Vdd är under en specificerad nivå som avgörs av nedanstående ekvation.

Vdd*(R1/(R1+R2))=0.7V

Watchdog Timer (WDT)

Watchdog timern går av sig själv med hjälp av en intern RC oscillator som är separerad från RC oscillatorn på OSC1/CLKIN pinnen. Detta innebär att WDT går även om processorn är försatt i sleep mode. WDT time-out genererar en reset. Om processorn är satt i sleep mode orsakar en time out på WDT att processorn vaknar upp, i annat fall orsakar en time out på WDT att PIC'en utför en reset. WDT kan väljas bort vid programmeringstillfället av PIC'en med hjälp av konfigurationsbitarna.

WDT har en nominell time out period på 18ms (utan prescaler(se optionsregistret)). Time out tiden varierar beroende på olika temperaturer, spänningsmatning och tillverkningsserier av PIC'en.

Om en större time out önskas för längre tider kan prescalern i optionsregistret användas. Prescalern kan dela ner klockningen med en viss faktor. Detta ställs in i options-registret. Om prescaler är satt för att generera förhållandet 1:128 ger detta en period på ca 2.5 sekunder.
CLRWDT och SLEEP instruktionerna nollställer WDT och prescaler (om den är inställd till WDT). Detta förhindrar att det inträffar en WDT interrupt (varvid en reset genereras).

Vid en WDT time out kommer bit TO' i statusregistret att nollas.

Power-down mode (SLEEP)

PIC'ens strömförbrukning kan minskas genom att den sätts i sleep mode varvid vissa delar i PIC'en stängs av. Senare kan PIC'en vakna upp från sleep mode med bibehållen information i registren för fortsatt exekvering av programmet.
För att sätta kontrollern i sleep-mode används kommandot SLEEP. Om WDT är vald nollställs WDT vid en SLEEP instruktion men fortsätter att gå. PD' biten (STATUS<3>) nollställs, TO' biten (STATUS<4>) sätts och oscillatorn stängs av. Alla I/O pinnar behåller sin status under sleep mode som de hade innan SLEEP instruktionen utfördes. För att få så låg strömförbrukning som möjligt under sleep mode bör så många I/O anslutningar som möjligt konfigureras till ingångar. Anslut antingen Vdd eller Vss till ingångarna för att förhindra att de ligger och flyter. Kontrollera att ingen extern krets drar ström från I/O pinnarna. Något som ytterligare minskar strömförbrukningen är om du stänger av yttre klockpulser. Tänk också på de interna pull-up resistorerna som kan konfigureras på port B. Även dessa konsumerar ström.
Uppvakning från sleep kan ske på följande sätt:

Extern reset på MCLR' pinnen.
WDT uppvaknande (om WDT är förvalt i konfigurationsbitarna).
Interrupt från RB0/INT pinnen.
Interrupt från förändring på port B.
Interrupt från EEPROM skrivning färdig.
(**För mer information se databok)**

In-Circuit Serial Programming (ICSP=programmering direkt på kretskort)

PIC16F84 kan programmeras direkt på kretskortet vilket gör att konstruktionen ej behöver programmeras förrän precis innan leverans. Detta innebär att det är lätt att kundanpassa en konstruktion samt att utföra uppgradering på gammalt lagermateriel.
För att läsa mer om hur man utför detta hänvisar jag till databok samt **In-Circuit Serial Programming Guide** dessa böcker hittar du på Microchips hemsida.

Register uppsättningen i PIC16F84'an

	Bank0	Bank1	
00h	Indirect addr*	Indirect addr*	80h
01h	TMR0	OPTION	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h			87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2*	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	68st Allmänna Register (SRAM)	Mappad (åtkomst) i Bank 0	8Ch
4Fh			CFh
50h			D0h
7Fh			FFh

Grå rutor är oimplementerat minne och läses som noll

* = ej ett fysiskt register

Till vänster ser Ni en minneskarta på olika register och deras adresser. Jag kommer inte att förklara alla register, utan hänvisar till den databok som finns på Microchips hemsida.

Indirekt addr:	Används för indirekt adressering
TMR0:	Timer register som räknas upp av pulser.
PCL:	Programräknare för låga delen av PC (8bit).
OPTION:	Register som hanterar vissa inställningar se nedan.
STATUS:	Register som visar olika status samt hanterar bank select. (se nedan).
FSR:	File Select Register (används vid indirekt adressering).
PORTA:	I/O port A
PORTB:	I/O port B
TRISA:	Register som anger I/O A portens riktning
TRISB:	Register som anger I/O B portens riktning
EEDATA:	Register som hanterar EEPROM minnet på 16F84.
EEADR:	Register som hanterar EEPROM minnet på 16F84.
EECON1:	Register som hanterar EEPROM minnet på 16F84.
EECON2:	Register som hanterar EEPROM minnet på 16F84.
PCLATH:	Programräknaren den höga Byten för adressering större än 255 adresser.
INTCON:	Interrupt-hanterings-kontroll-register.

Status registret (adress 03h, 83h)

bit7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
IRP	RP1	RP0	TO'	PD'	Z	DC	C

Bit7: IRP:Register Bank Select bit (används för indirekt adressering)
0=Bank 0, 1 (00h-FFh)
1=Bank 2, 3 (100h-1FFh)
IRP biten används inte på PIC16F8X och bör vara satt till noll för att garantera framåtkompatibilitet med större PIC'ar.

Bit5-6 RP1:RP0: Register Bank Select bit (andvänds för direkt adressering).
00=Bank 0 (00j-7Fh)
01=Bank 1 (80h-FFh)
10=Bank 2 (100h-17Fh)
11=Bank 3 (180h-1FFh)
Varje bank innehåller 128 bytes. Endast RP0 används på PIC16F8X. RP1 bör vara satt till noll för att garantera framåtkompatibilitet med större PIC'ar.

Bit4: TO': Time out bit
1=Efter påslag av kretsen, efter en CLRWDT instruktion eller SLEEP instruktion.
0=En Watch Dog Timer time-out har inträffat.

Bit3: PD': Power-down bit
1=Efter en power-up eller då instruktionen CLWDT har utförts.
0=Då en SLEEP instruktion har utförts

Bit2: Z: Zero bit
1=Resultatet av en räkneoperation eller en logisk operation har blivit noll.
0=Resultatet av en räkneoperation eller en logisk operation inte har blivit noll.

Bit1: DC: Digit carry/borrow' bit (för ADDWF och ADDLW instruktioner)(För borrow' är det tvärt om)
1=En carry bit från fjärde biten från msb har inträffat.
0=ingen carry bit från fjärde biten från msb har inträffat.

Bit0: C: Carry/borrow bit (för ADDWF och ADDLW instruktioner)
1=En carry bit har ramlat ut från msb av resultatbyten
0=Ingen carry bit har ramlat ut från msb av resultatbyten
OBS när det gäller borrow(lånebit) är förhållandena omkastade

Option registret (Adress 81h)

bit7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
RBPU'	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

Bit7: RBPU': PORTB Pull-up inställnings bit (man kan konfigurera så att ett svagt pull-up resistor kopplas internt).
1=PORTB pull-up är bortagen.
0=PORTB pull-up är aktiverad.

Bit6: INTEDG: Med denna bit ställerman in om interrupten på INT pinnen skall ske på positiv eller negativ flank.
1=Interup sker på positiv flank RB0/INT.
0= Interup sker på negativ flank RB0/INT.

Bit5 T0CS: TMR0 inställnings bit som anger om uppräkning skall ske med intern klocka eller extern puls.
1=Uppräkning genom extern puls på RA4/T0CKI.
0=Intern uppräkning från interna klockan.

Bit4: T0SE: TMR0 flank inställnings bit.
1=Öka TMR0 med ett då RA4/T0CKI pin går från hög till låg.
0=Öka TMR0 med ett då RA4/T0CKI pin går från låg till hög.

Bit3: PSA: Prescaler Assignment inställning av prescaler för endera WDT eller TMR0
1=Prescaler tilldelad för Watch dog timer.
0= Prescaler tilldelad för TMR0

Bit2-0: PS2:PS0 anger med vilken faktor/prescaler man skall dela ner uppräkningen av TMR0. Då WDT avses heter det postscaler.

Bit värden PS2 PS1 PS0	TMR0 prescaler hastighet	WDT postscaler hastighet
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

INTCON registret (Adress 0Bh, 8Bh)

bit7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Bit7: GIE: Global Interrupt möjliggörande bit
1=möjliggör interrupter som man har ställt in i de andra bitarna nedan.
0=kopplar bort interruptmöjligheten.

Bit6: EEIE: Skrivning till EEPROM-minne interrupt inställning.
1=möjliggör EE skrivnings interrupt.
0=kopplar bort EE skrivningsinterrupt.

Bit5: T0IE: TMR0 owerflow interrupt inställning.
1=Möjliggör TMR0 interrupt
0=kopplar bort TMR0 interrupt

Bit4: INTE: RB0/INT interrupt inställning
1=möjliggör RB0/INT interrupt.
0=kopplar bort RB0/INT interrupt

Bit3: RBIE: interrupt om ändring på port B.
1=möjliggör interrupten
0=kopplar bort interrupten

Bit2: T0IF: TMR0 owerflow interruptflagga.
1=TMR0 har fått owerflow (måste nollas i mjukvara)
0=ingen owerflow har skett.

Bit1: INTF: RB0/INT interrupt flagg-bit.
1=Interrupt på RB0/INT har skett.
0=Ingen interrupt på RB0/INT har skett

Bit0: RBIF: Interruptflagga för port B som anger om en pin har ändrat värde där.
1=minst en ändring på port B RB7:RB4 har skett (denna etta måste nollas i mjukvara).
0=ingen interrupt har genererats genom port B RB7:RB4.

Instruktioner till PIC16F84

ADDWF	f,d	Addera w med fil	(w)+(f)→(dest)
ANDWF	f,d	OCH mellan w och fil	(w).AND.(f) →(dest)
CLRF	F	Rensa fil	00h→(f)
CLRW	-	Rensa w-reg	00h→(w)
COMF	f,d	komplementet på filen	(f)' →(dest)
DECF	f,d	Minska fil med ett	(f)-1→(dest)
DECFSZ	f,d	Minska fil med ett och hoppa över nästa instruktion om fil=0	(f)-1→(dest);hoppa om f=0
INCF	f,d	Öka fil med ett	(f)+1→(dest)
INCFSZ	f,d	Öka fil med ett och hoppa över nästa instruktion om fil=0	(f)+1→(dest);hoppa om f=0
IORWF	f,d	"Inclusive OR" ELLER mellan w och fil	(w).OR.(f) →(dest)
MOVF	f,d	Flytta innehållet till fil eller w	(f) →(dest)
MOVWF	f	Flyttar innehållet i w till fil	(w) →(f)
NOP	-	Ingen instruktion utförs under denna klocksykel	-
RLF	f,d	Roterar binära innehållet i en fil ett steg åt vänster	Carry till lsb
RRF	f,d	Roterar binära innehållet i en fil ett steg åt höger	Carry till msb
SUBWF	f,d	Subtrahera w från fil	(f)-(w) →(dest)
SWAPF	f,d	Skifta bit 3 till 0 med bit 7 till 4	(f<3:0>)→(dest<7:4>)
XORWF	f,d	Exklusiv ELLER mellan w och fil	(w).OR.(f) →(dest)

BCF	f,b	Nolla bit b i fil f	0→(f<b>)
BSF	f,b	Ettställ bit b i fil f	1→(f<b>)
BTFSC	f,b	Bit test på fil f bit b om biten är noll hoppa över nästa instr	Skip if (f<b>)=0
BTFSS	f,b	Bit test på fil f bit b om biten är satt hoppa över nästa instr	Skip if (f<b>)=1

ADDLW	k	Addera konstant med w	(w)+k→(w)
ANDLW	k	Utför OCH mellan w och konstant	(w).AND.(k) →(w)
CALL	k	Hoppa till en subrutin k [label]	Se databok
CLRWDT	-	Rensa watchdog timer registret	00h→WDT
GOTO	k	Hoppa till k [label]	Se databok
IORLW	k	"Inclusive OR" ELLER mellan konstant och w	(w).OR.(k) →(w)
MOVLW	k	Flytta konstant till w	k→(w)
RETEFIE	-	Hoppa tillbaka från interrupt subrutin	Se databok
RETLW	k	Returnera en konstant till w (används i lookup tabeller)	Se databok
RETURN	-	Hoppar tillbaka från en subrutin	TOS→(PC)
SLEEP	-	Sätter processorn i standbyläge (sleep mode)	Se databok
SUBLW	k	Subtraherar w från konstant	k-(w) →(w)
XORLW	k	Exklusiv ELLER mellan konstant och w	(w).XOR. k→(w)

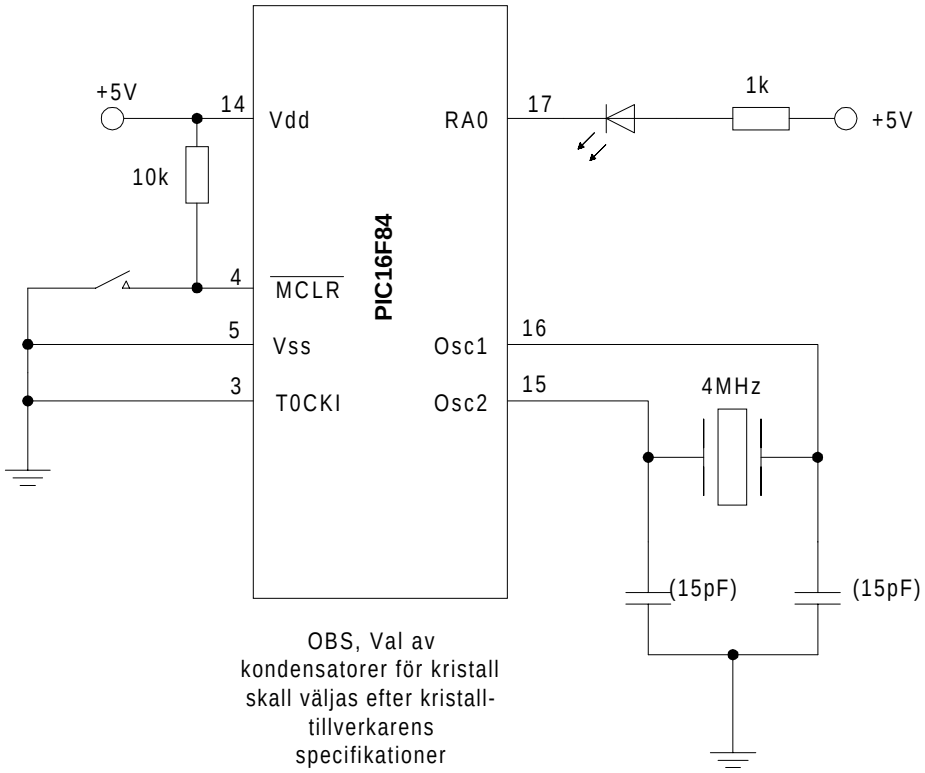
k = konstant (litteral)
f = fil = register
b = bit
w = w-registret
d= destinationsbit

Destinationsbiten anger vart resultatet av en operation skall läggas. Destinationsbiten kan endast ha två värden.

d=0→resultat läggs i w-registret.

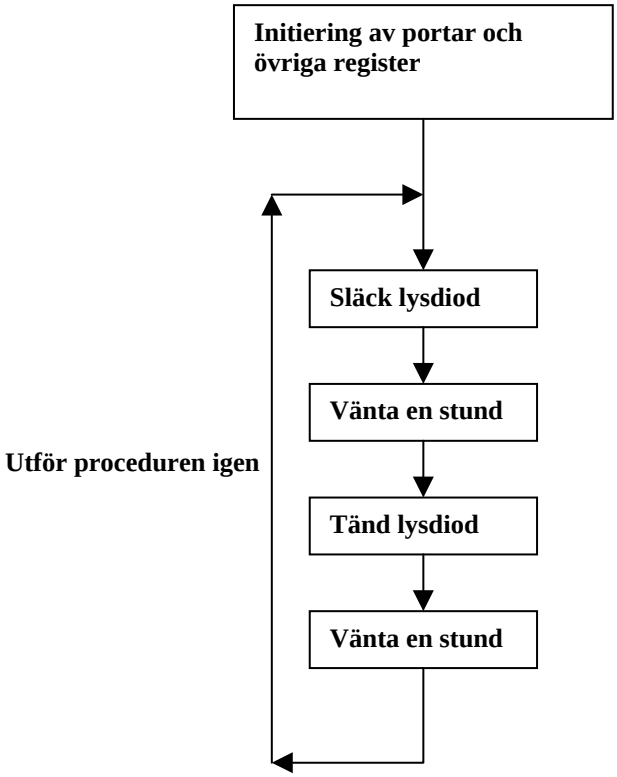
d=1→resultat läggs i den andra filen (registret) f.

En uppkoppling för blinkande LED.



Ett enkelt Program.

Nu tänkte jag gå igenom ett enkelt program som skall få en lysdiod att blinka. Det första man bör göra när man skriver ett program är att ta fram ett enkelt flödesschema. Flödesschemat visar hur programmet skall exekveras. Det allra viktigaste innan du skriver programmet är förstås att du vet vad programmet skall utföra och därmed vad du ska använda för hårdvara.



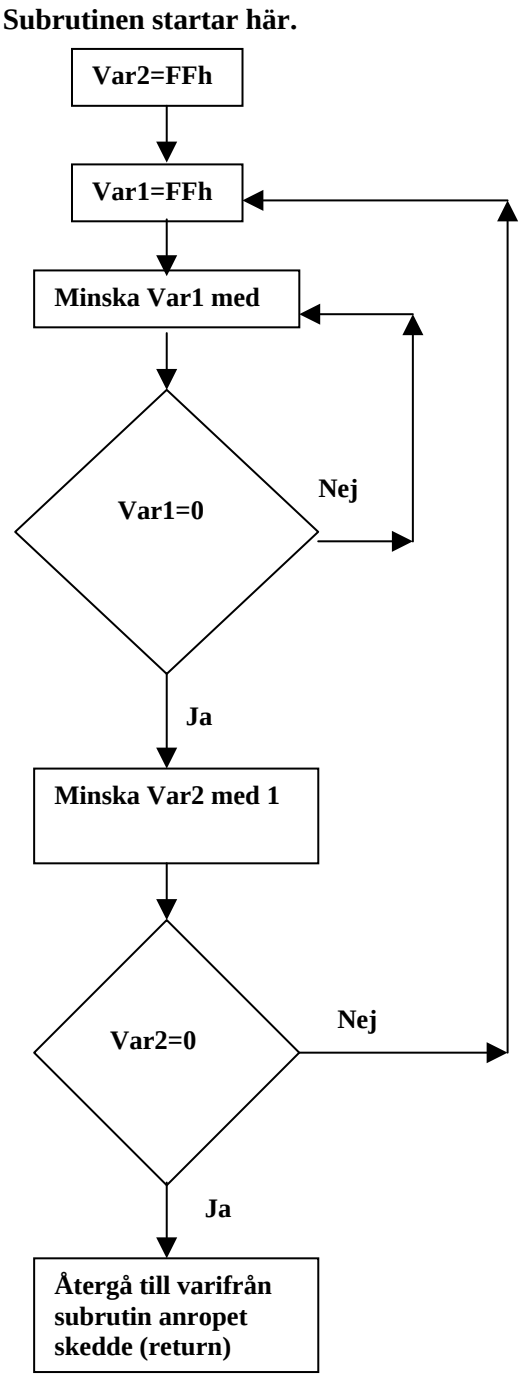
Här ovan ser vi ett flödesschema på ett program som ska få en lysdiod att blinka. Det är viktigt att flödesschemat blir så enkelt som möjligt i början. Använd stora byggblock som ovan. Under utvecklingsfasens gång kan de stora byggblocken brytas ner i mindre beståndsdelar. Till slut kan byggblocken i extremfall vara så små att de endast består av en enda instruktion. I detta exempel kan det vara en ide att bryta ner vänta-blocket i ett ytterligare ett detaljerat flödesschema.

Vänta-blocket eller vänta-rutinen är här ett återkommande block som finns på fler än ett ställe. I detta fall kan en subrutin med fördel användas. Subrutinen kan åberopas då den behövs. På så vis behöver man endast skriva in detta block en gång. Detta gör att programmutrymme sparas

Anledningen till att det behövs ett vänta-block är att människans ögon eller rättare sagt hjärna inte skulle hinna registrera att lysdioden blinkar. Händelseförloppet skulle uppfattas som om att lysdioden alltid var tänd.

Vänta-subrutinen

Denna subrutin åberopas genom att använda call-instruktionen, tex: ***call wait***.
När subrutinen avslutas med instruktionen return, hoppar programmet tillbaka från det ställe som subrutinen anropades plus en programrad.



Förklaring av programmet (se programmet på nästa sida).

När ett program skrivs är det viktigt att programmeraren skriver in kommentarer och upplysningar för sig själv och för andra. Detta görs genom att skriva in ett semikolon innan kommentaren, tex ***;här är en kommentar***. Kommentarer kan läggas in över allt i programmet och läses ej in av kompilatorn utan ignoreras helt.
Om kommentarer ej läggs in i programmet kommer du snart att märka att du inte har en aning om vad det är du gör.

Först i programmet skrivs generella kommentarer som talar om hur programmet fungerar.
Sedan kommer ett preprocessordirektiv (list p=16f84) som anger för kompilatorn hur den ska kompilera programmet. Här anges t.ex. att det är ett program för PIC16F84 som skall kompileras.

Variabeltilldelningar tilldelar en siffra ett mer lättförståeligt namn (i detta fall tilldelas variabelnamnet en siffra för en specifik registeradress) . I stället för att komma ihåg siffror kan ett mer lättförståeligt variabelnamn användas. Det som händer vid kompileringen är att dessa namn ersätts av den siffra som är tilldelad namnet. Dessa tilldelande programrader är endast synlig för kompilatorn och kommer ej att bidra till någon maskinkod. EQU preprocessordirektivet gör precis detta.

Instruktionen org är ett preprocessordirektiv som anger för kompilatorn vart nästföljande instruktion skall läggas in i programminnet i PIC'en. I detta fall skall en goto instruktion läggas i reset vektorn 0x00(se databok för minnes karta för PIC16F84). Detta gör att när en reset utförs på PIC'en så kommer goto init instruktionen utföras varvid initiering sker och programflödet fortlöper som det ska.

Instruktionen goto utför ett ovillkorligt hopp till den adress som angivits. I stället för att hålla reda på vilken adress som finns vart i programmet använder man så kallade "labels" (etiketter) . I detta fall är "labeln" init som finns längre ner i programmet. Programmet kommer alltså att hoppa till den adress som init symboliserar.

Subrutinen wait är den subrutin som vi tidigare har nämnt och skapat ett flödesschema på. För att call instruktionen skall veta var subrutinen finns anges en label där denna programslinga finns. I detta fall är labeln wait. Denna subrutin utförs ej innan en call instruktion har utförts. För en närmareförklaring av wait rutinen se kommentarer i programkoden.

Initiering av portar

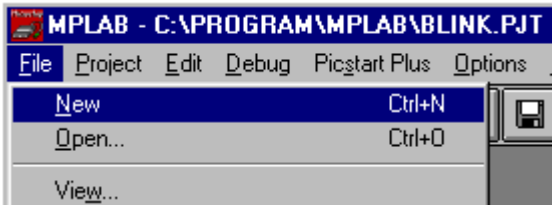
I/O anslutningarna på en PIC kan konfigureras som utgångar respektive ingångar. Detta görs genom att modifiera tris-registret för respektive port. En minnes regel för att hålla reda på vad som definierar en utgång respektive ingång är att tänka:**1**=**I***nput* samt **0**=**O***utput*. Tris-registret modifieras genom att lagra det värde som finns i w-registret til tris-registret. Det finns i detta fall två tris register ett för port A och ett för port B. Den minst signifikanta biten i trisa motsvarar RA0 på port A och biten till vänster om denna motsvaras av RA1 osv. I detta fall sätts alla bitar till nollor. Detta ger att alla pinnar på port A är utgångar. Tris står för tri state dvs tre lägen ett, noll samt högimpedivt.

Nu kommer själva huvudprogrammet som har "labeln" main. Huvudprogrammet är mycket enkelt och består endast av ett fåtal instruktioner. De bitmanipulationer som utförs direkt på portarna påverkar inte tris-registret utan sänder helt enkelt en logisk etta eller nolla till respektive I/O-pin. Main är skapad som en slinga och kommer att upprepa sig gång på gång då goto hela tiden hoppar tillbaka till main. Se kommentarer i programmet.

```
;*****************************************************************
;Skriven av                  :Niklas Wennerstrand
;Datum                       :1999-02-08
;Filnamn                     :diodblik.asm
;Programmets syfte           :Få en lysdiod att blinka på port A utgång RA0
;Processor                   :16F84
;Clock                       :XT 4MHz
;Watchdog timer              :off
;Code protect                :off
;*****************************************************************
                list        p=16F84     ;processorangivelse för kompilatorn
;********************** Variabeltilldelningar ************************
STATUS          equ         0x03        ;Adress för status register
porta           equ         0x05        ;Adress för port A
var1            equ         0x0c        ;RAM-adress för Var1
var2            equ         0x0d        ;RAM-adress för Var2
trisa           equ         0x85        ;Adress för trisa registret
RP0             equ         0x05        ;bit 5 i statusregistret heter RP0.
                                        ;Med denna bit väljer man vilken bank som avses i minnet.
;*****************************************************************

                org         0x00
                goto        init

;************************Subrutiner*********************************
wait            movlw       0xff        ;Lägger ff i w-registret
                movwf       var2        ;Lägger innehållet i w-reg i fil var2
loop2           movlw       0xff        ;Lägger ff i w-registret
                movwf       var1        ;Lägger innehållet i w-reg i fil var1
loop1           decfsz      var1,1      ;minska  var1 med 1 hoppa över nästa instr om noll
                goto        loop1       ;
                decfsz      var2,1      ;minska  var2 med 1 hoppa över nästa instr om noll
                goto        loop2       ;
                return                  ;återgår från subrutin
;*****************************************************************

;************************Initiering av portar*************************
init            clrw                    ;Rensar w-registret dvs fyller det med nollor.

                movwf       porta       ;Förbered port A att sättas till noll
                                        ;för att undvika ofrivillig drivning av kretsen
                                        ;då tris registret sätts.

                bsf         STATUS,RP0  ;väljer bank 1 där trisa registret finns
                movwf       trisa       ;sätter I/O kontrollregistret tris för port A, alla I/O=utgång
                                        ;baserat på vad som finns i W-reg.
                bcf         STATUS,RP0  ;väljer bank 0 där porta registret finns


;*****************************************************************
main            bsf         porta,0     ;Set fil porta bit 0 släcker lysdiod RA0=1
                call        wait        ;väntar en stund
                bcf         porta,0     ;Clear fil porta bit 0tänder lysdiod RA=0
                call        wait        ;väntar en stund
                goto        main        ;

                end                     ;Direktiv som talar om för kompilatorn avslut på programmet
```

Skriv in och behandla program med hjälp av MPLAB.

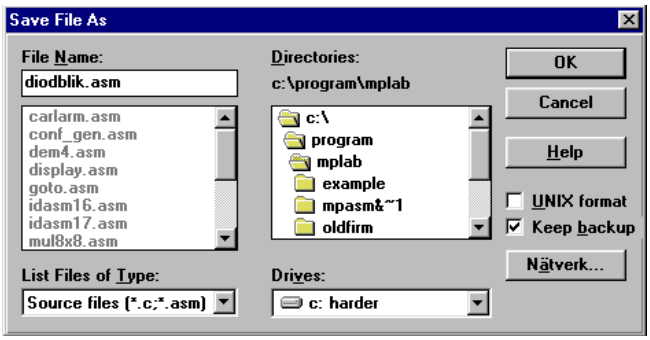
Det första som skall göras är att skiva in ”sourcefilen” (källfilen/källkoden). ”Sourcefilen” är den fil som innehåller assemblerprogrammet, dvs det språk som vi kommer att använda oss av. Ofta pratar man om lågnivåspråk och högnivåspråk. Med högnivåspråk avses att språket ligger på en mer lättförståelig nivå för programmeraren än vad ren maskinkod skulle göra. Maskinkod består endast av nollor och ettor och skulle för en vanlig dödlig bli totalt obegripligt. Assembler räknas dock till lågnivåspråk beroende på att det ligger nära ren maskinkod. (Till högnivåspråk räknas t.ex. Pascal och Basic som är långt ifrån maskinkod men närmre människans språk.)



För att skapa en ny sourcefile markerars *File* och sedan *New*.

Ett tomt ark med namnet *Untitled*.öppnas
På detta ark skall assemblerprogrammet skrivas in. För att programmet inte skall gå förlorat så måste sourcefilen namnges och **du måste spara filen så ofta som möjligt**. Detta beror på att om du har knappat in kod och ett strömavbrott skulle inträffa eller att din katt hoppar upp på tangentbordet och förstör din kod leder detta annars till stor frustration.

Välj *File→Save As…* så att nedanstående fönster öppnas.

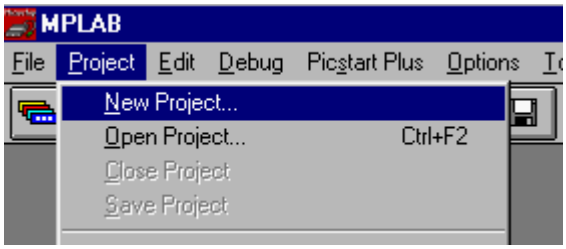


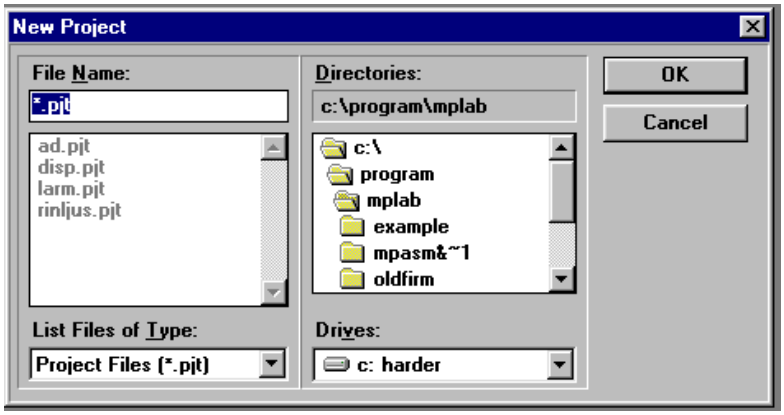
Skriv in filnamnet diodblik under *File Name* med sufixet ***.asm***. Detta med sufixet .asm är mycket viktig och får ej glömmas. Programmet skall ligga under MPLAB-mappen. Tryck därefter [OK].
I MPLAB får filnamnet max vara 8 tecken långt, plus suffixet som endast får ha tre tecken.

Nu är det dags att skriva in assemblerprogrammet. Skriv in programmet på föregående sida så noggrant som möjligt för att få så få felmeddelande som möjligt vid kompileringen. Var inte orolig om det blir fel. Det är däremot mycket ovanligt att inskrivningen blir rätt första gången men skam den som ger sig.

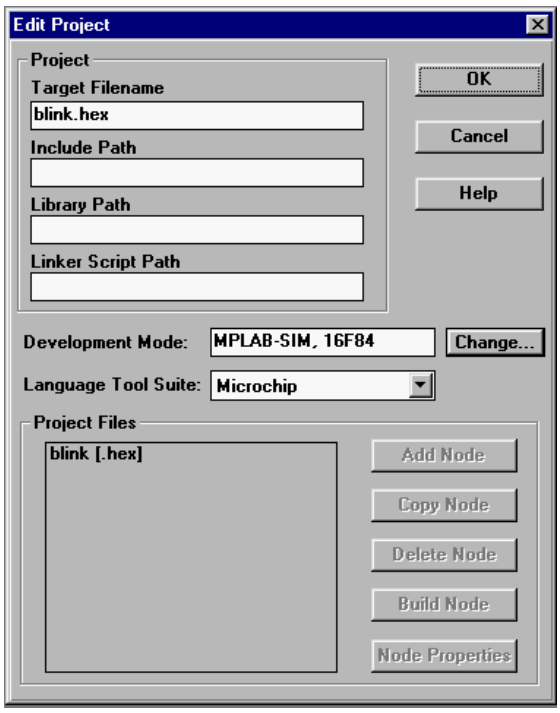
Skapa projekt.

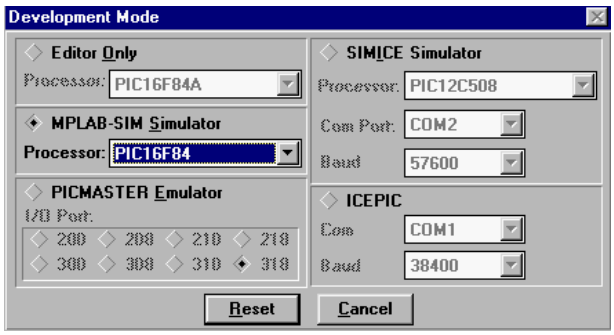
För att kunna kompilera ”sourcekoden” måste ett projekt skapas. Projektet skapar de filer som hör ihop med programmet och länkar olika filer mellan varandra.
I vårt fall skall vi endast använda oss av en ”sourcefile” (källfil). Detta för att allt skall bli så enkelt som möjligt.
Klicka med vänster musknapp på menyvalet *Project* och markera sedan *New Project*. Skriv in ett projektnamn under *File Name* med sufixet **.pjt** och tryck därefter [OK]. För att underlätta det hela, se till att projektet ligger under mplab-mappen.



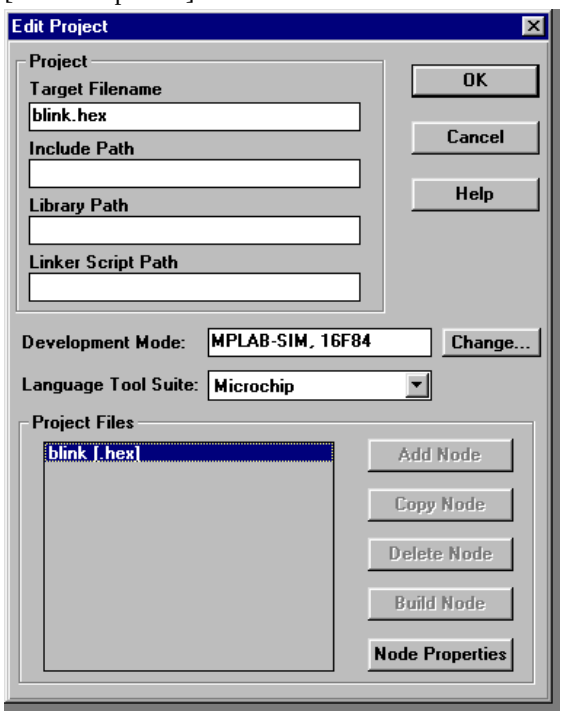


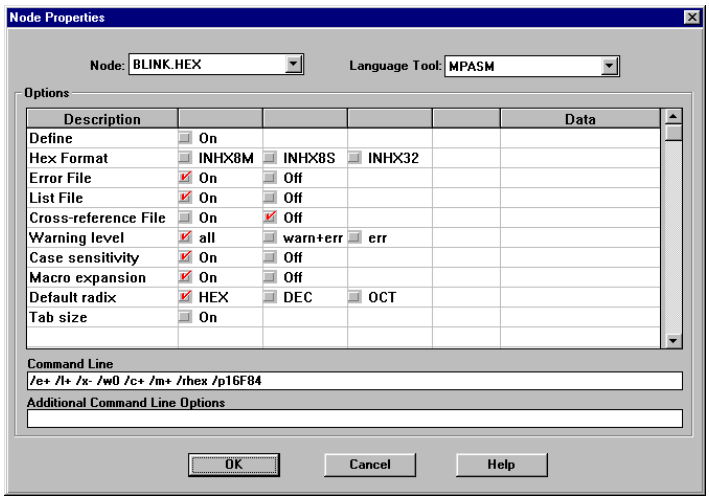
Ange den information som står i rutan *development mode* till vad som anges här. För att göra detta tryck på [Change] och då öppnas fönstret till höger.
Detta fönster skiljer sig antagligen lite från Ert men det huvudsakliga är att Ni markerar MPLAB-SIM och ställer in processor PIC16F84. Tryck sedan [Reset].





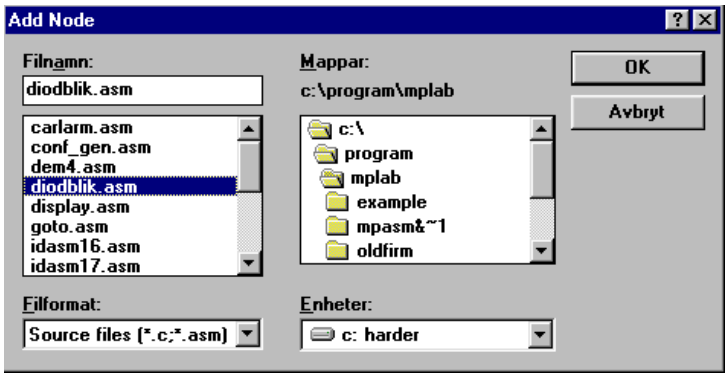
Markera blink[.hex] i *Project Files* och tryck därefter på [Node Properties].





Nästa fönster som öppnar sig är det ovan. Där finns ett antal förkryssade alternativ. För att gå vidare så skall du avmarkera alla bockar vilket sätter inställningarna till ”default”. Markera MPASM i *Language Tool* så som bilden ovan visar. Tryck därefter [OK].

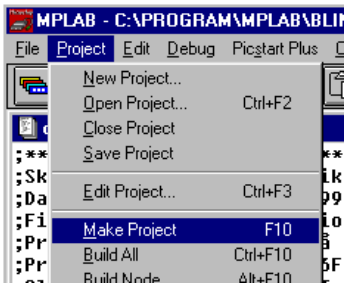
Tryck på [Add Node], varvid följande fönster öppnas.



Här markerar du *diodblik.asm* och trycker på [OK].
Efter detta öppnas *Edit Project*- Fönstret, tryck då på [OK] varvid projektet är skapat.
Nu är vi redo att kompilera source-filen.

Kompilering av program

Välj
Projekt→Make Project
Eller *Build all.*
Build all är dock att föredra då den skapar och uppdaterar alla tillhörande filer.
Build all kan dock ta längre tid vid stora projekt med flera assemblerfiler.



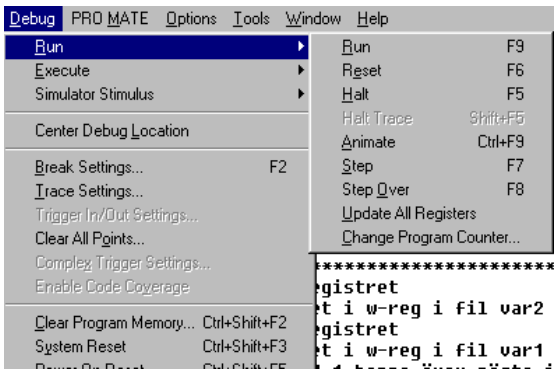
När kompileringen är färdig öppnas ett fönster där det förhoppningsvis står:

```
Building DIODBLIK.HEX...

Compiling DIODBLIK.ASM:
Command line: "F:\PROGRAM\MPLAB\MPASMWIN.EXE /p16F84 /q
F:\PROGRAM\MPLAB\DIODBLIK.ASM"
Message[302] F:\PROGRAM\MPLAB\DIODBLIK.ASM 46 : Register in operand not in bank 0.  Ensure that
bank bits are correct.

Build completed successfully.
```

Om du skulle få ett felfritt resultat som ovan, redan första gången skulle jag bli mycket imponerad. Faktum är att det troligtvis kommer att stå ”error” på många rader. Det är bara att dubbelklicka på felraden och därmed hoppar du automatiskt till den felrad som indikerats.
De vanligaste felen som förekommer är att man har glömt semikolon framför kommentarer, förväxling mellan noll och bokstaven o, använt variabelnamn som ej är deklarerade, glömt end i slutet på programmet men kanske framför allt stavat fel på instruktioner eller variabler. Ett annat vanligt fel är att man skriver kommandon, labels eller data i fel kolumn.
I exemplet ovan har kompileringen genererat ett meddelande ”Message” Detta är inte så mycket att bry sig om utan är endast till för att upplysa programmeraren om att tänka till lite extra när och vart dessa instruktioner används.
Det kan även förekomma varningar ”Warning” och det har ungefär samma innebörd som ”Message”.
Dessa varningar och meddelanden går att stänga av genom att ”editera” projektet och ändra inställningarna för projektet under *Node Properties*

När man rättar felen som man har gjort behöver felet inte nödvändigtvis ligga på den raden som felet indikerats på. Felet kan bero på något innan själva felraden. Om man har deklarerat en variabel som heter Var1 och sedan använder en variabel senare i programmet som heter var1, får man ett felmeddelande att denna variabel ej är deklarerad. Här såg vi även ett annat mycket vanligt fel nämligen att kompilatorn gör skillnad på versaler och gemener. Kompilatorn kan dock ställas in under *Node Properties* för att ignorera skillnader mellan gemener och versaler.

Kontroll av programmet i simulatorn.
Då programmet är kompilerat och det ej finns några kompileringsfel behöver man oftast kunna se om programmet uppför sig som man tänkt sig. Detta utför man i simulatorn. I simulatorn kan man se vad som händer i olika register, mäta exekveringstider samt se hur programflödet fortlöper i programmet.
För att vara säker på att simuleringsmiljön är på kan du ställa in detta i menyn *Options→Development Mode*.
Ange även rätt processor. Om du har följt instruktionerna tidigare skall detta redan vara inställt.
Om man har tillgång till en emulator kan man göra liknande tester med den skillnaden att testerna utförs i verkligheten (se rubrik: vad är en emulator).



Till vänster ser du vilken meny som du skall använda dig utav för att simulera programmet välj debug-menyn.
Ett tips är att memorera eller skriva små fusklappar på vilka funktionsknappar som gör vad. Det blir så omständligt att gå in i dessa menyer så fort man tex. vill hoppa fram en programrad i programmet.

Run[F9]: Kör programmet så fort som datorn tillåter (Här ser man inte vad som sker i olika register.).

Reset[F6]: Simulerar en reset på processorn. Detta skall alltid göras innan man börjar köra programmet.

Halt[F5]: Stannar upp programmet om man har kört run. Nu kan man avläsa de olika registervärdena.

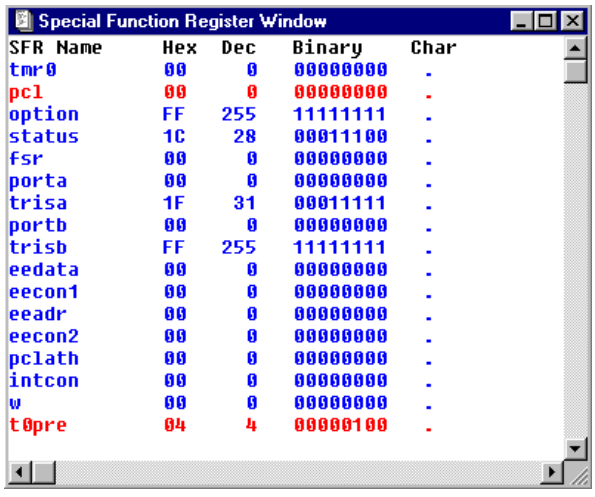
Animate: [Ctr]+[F9] Detta kör programmet långsamt steg för steg och här kan du se de olika registren uppdateras.

Step[F7]: Denna funktion stegar fram programmet en rad i taget. Registren går att tyda mycket bra.

Step Over[F8]: Hoppar över nästkommande programrad.

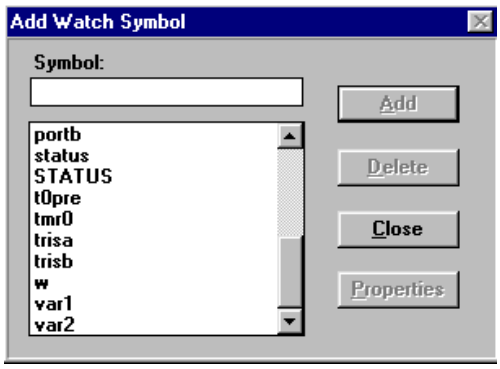
Öppna ett watch window för att se registren.

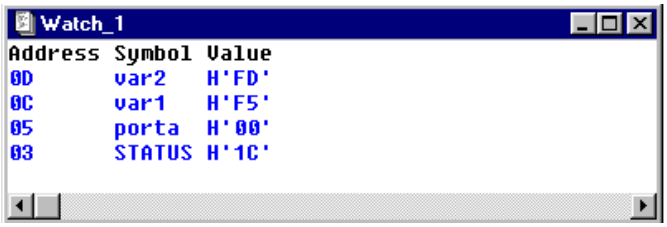
För att kunna se vad olika register innehåller, välj: *Window→Special Function Registers* , varvid nedanstående fönster öppnas.



Här kan man se innehållet i de vanligaste registren.
För att se egendefinierade variabler väljer du *Window→New Watch Window.*

Välj *Window→New Watch Window* så öppnas nedanstående fönster där man kan markera vilken variabel som man vill titta på. Markera den eller de register som du vill titta på och tryck sedan [Add].

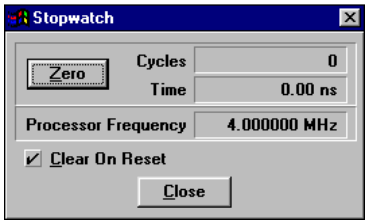


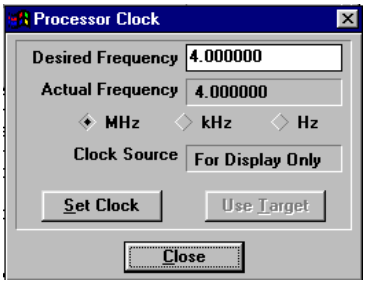


Här ett exempel på ett eget watch window.
För att lägga till eller ta bort register från watch window använd [Insert] eller [Delete] på tangentbordet.
Hur värdet skall presenteras ställs in under [properties].

Ta tid på kodexekvering.

För att på ett lätt och smidigt sätt se hur lång tid olika delar i programmet tar kan man använda ett stoppur. Detta görs genom att välja *window→Stopwatch*, varvid nedanstående fönster öppnas.





För att välja klockhastighet välj *Options→Processor→Setup→Clock Frequency*, så öppnas fönstret till höger ovan.

Break-point

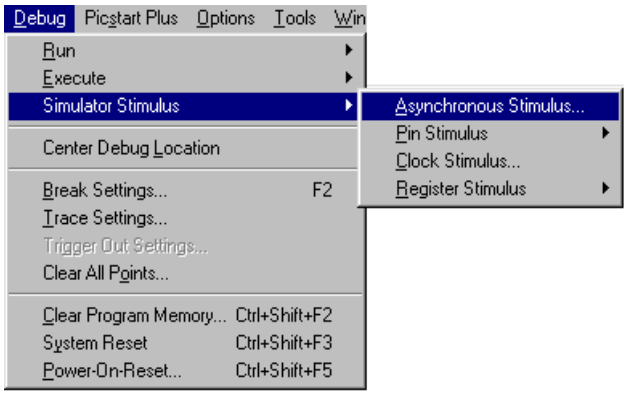
Ibland vill man låta programmet köra av sig självt till en viss punkt i programmet. Då behöver man en så kallad ”break-point”. När programmet stöter på denna ”break-point” stannar exekveringen, varvid man tex kan läsa av stoppuret eller interna register.

Gör så här:
Markera den programrad där du vill lägga din break-point och klicka sedan med vänstra musknappen.
Klicka sedan med den högra musknappen så att fönstret till vänster öppnas.
Välj sedan *Break Point(s)* varvid din programrad ändrar färg.
För att ta bort markering utför samma procedur som ovan

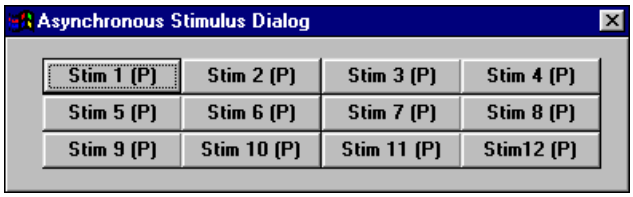
Ändring av insignaler på portarna.

Då man debuggar (felsöker) sitt program vill man ofta kunna testa olika insignaler till kretsen som i sin tur skall bestämma själva programmets gång. För att åstadkomma detta i MPLAB kan man välja att använda antingen en stimulus fil eller i det enklare alternativet: ”*asynchronous stimulus*”. Det senare alternativet är det som jag kommer att beskriva. För att använda en ”stimulus-fil” hänvisar jag till MPLAB-manualen.

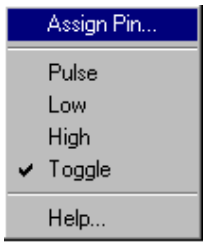
Välj: *Debug→Simulator Stimulus→Asynchronous Stimulus.*



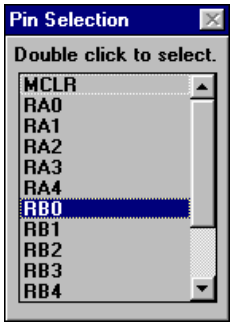
Efter detta öppnas följande fönster där du markerar en knapp med högra musknappen (i vårt fall stim1 knappen).



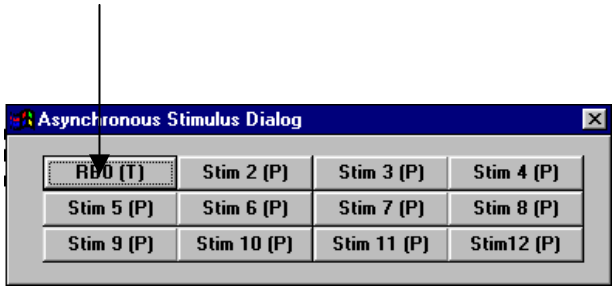
Välj sedan vad du vill skall hända då du trycker på knappen. Detta gör du genom att bocka för det alternativ som du vill använda dig av. Jag har här valt att varannan gång som knappen trycks skall portens insignal skifta från 1 till noll eller 0 till 1 (”toggla”).
Efter detta tilldelar vi en pinne på PIC’en till just denna knapp genom att välja *Assign Pin*.



Härnäst öppnas nedanstående fönster där vi väljer RB0 till den knapp som ska konfigureras.
Detta görs genom att dubbelklicka på RB0.



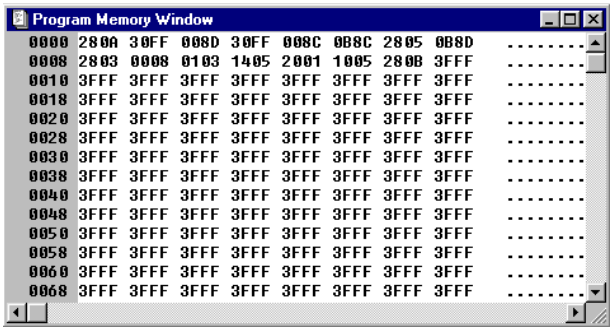
Nu har vi lyckats att tilldela en funktion som togglar RB0 då man trycker på denna knapp.



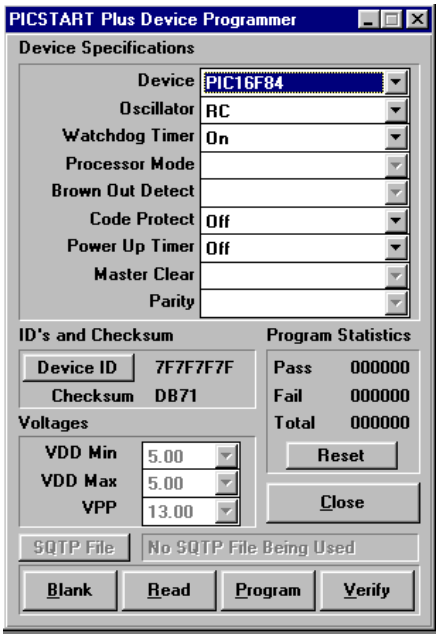
Programmering av själva kretsen

För att kunna programmera själva kretsen behöver vi en programmerare, förslagsvis en PICSTART PLUS. Anslut PICSTART PLUS till com-porten och anslut matningsspänningen. (Det finns även andra programmerare för PIC'ar bl.a. PROMATE II). Anslut därefter själva PIC'en i för PIC'en avsedd sockel. Det finns markerat på PICSTART PLUS hur kretsen skall vara applicerad. På de flesta datorer finns det två eller flera commportar och du måste ange för MPLAB till vilken port du har anslutit PICSTART PLUS till. Detta gör du genom att välja: *Options→Programmer Options→Communications Port Setup*, där du kan välja com-port.
(PICSTART PLUS är en utvecklingsprogrammerare och kontrollerar ej max och minspänningar. Det gör däremot PROMATE II)
Nästa steg är att aktivera Picstart Plus. Detta görs genom att man väljer *Picstart Plus→Enable Programmer*.

Efter detta öppnas följande fönster:



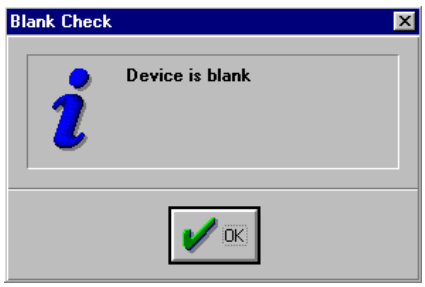
Fönstret ovan visar innehållet i datorns programminnesbuffer. Detta minne skall senare föras över till PIC'en.
I fönstret till höger ställer man in konfigurationsbitarna för PIC'en



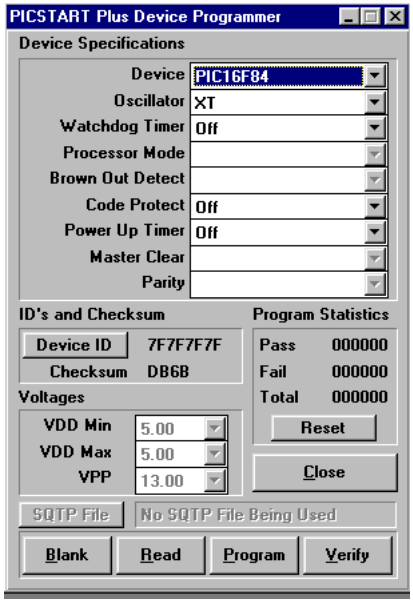
För att kunna programera PIC'en måste vi vara säkra på att den är raderad. Detta gör vi genom att välja:

Picstart Plus→Erase Program Memory.
Picstart Plus→Erase Configuration Bits.

När detta är gjort kommer Program Memory Window att vara fyllt med 3FFF. Detta hexadecimala tal motsvaras av att minnet är fyllt med ettor (programminnet är tomt).
Tryck sedan på [Program] varvid processorn raderas.
För att vara på den säkra sidan, tryck [Blank] för att testa om PIC'en är tom.



För att ladda ned program i PIC'en behöver vi återigen fylla programminnesbufferten. Detta görs genom att välja *Project→Build All* varvid program memory window åter får ett program i sitt minne.
Ställ nu in konfigurationsbitarna enligt bilden nedan.



När konfigurationsbitarna är inställda är det dags att bränna in programmet i kretsen.
Detta görs genom att trycka på [Program].
Ett fönster öppnas där du kan se att programmet laddats ned adress för adress.

För att vara säker på att programmet har laddats ner korrekt i PIC'en kan du verifiera programminnets program med det som nu finns i själva kretsen.
Detta görs genom att trycka [Verify].
(Tänk på att du inte kan läsa in minnet från en PIC som har "code protect" satt.)

<u>Nu är PIC'en redo för att testas i din uppkoppling.</u>

Minnet i mid-range PIC'ar.

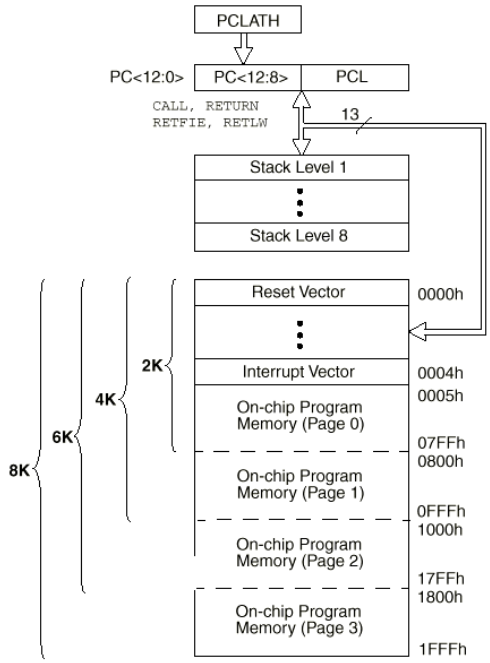
Introduktion

Det finns två olika minnesblock som är uppdelade i programminne och dataminne. Var och en av dessa minnesgrupper har sin egen buss så att åtkomst kan ske samtidigt på en och samma klockcykel.
Dataminnet kan delas upp i ytterligare två delar. Generella Register(General Purpose RAM register, GPR) och Speciella FunktionsRegister (Special Function Register, SFR).

Programminnets uppbyggnad.

Mid-Range MCU har en 13 bitars programräknare (Program Counter, PC) som kan adressera 8K x 14 i programminnesområdet. Bredden på programminnet (instruktionsbredden) är 14 bitar. Då alla instruktioner endast upptar ett ord betyder detta att 8K x 14 programminne kan innehålla 8K instruktioner.
Programminnet är uppdelad i fyra olika block (pages) med vardera 2K ord (0h – 7FFh, 800h – FFFh, 100h – 17FFh och 1800h – 1FFFh). Figuren nedan visar programminneskartan samt den 8 nivåer djupa hårdvarustacken. Beroende på vilken PIC du använder har du olika stor tillgång till detta minnesområde. Se databok för just din PIC.
För att hoppa mellan de olika minnesblocken skall du modifiera de högsta bitarna i programräknaren (PC). Detta utför du genom att skriva in önskat värde i ett speciellt funktionsregister (SFR) som heter Program Counter Latch High (PCLATH). PC räknas automatiskt upp med ett för varje programradsexekvering och räknar dessutom upp de höga bitarna i PC, vilket bidrar till att PC passerar minnessidorna (blocken) vid automatisk uppräkning av PC. Detta sker utan din inblandning. För PIC'ar med mindre än 8K ord bidrar denna uppräkning av PC till att om adresseringen blir högre än 8K slår räknaren runt och börjar från början. Om man i en PIC som har 4K ord adresserar 17FFh blir den verkliga adressen 7FFh. En PIC med 2K-ord eller mindre behöver ingen paging (blockhoppning).

Arkitekturen på minneskarta samt stacken



OBS **Programminnet är olika stort på olika PIC'ar. Se datablad för respektive PIC. Kalibreringsdata kan programmeras in i programminnet.**

Reset-vektor

På alla PIC'ar bidrar en reset till att programräknaren (PC) adresserar adress 0h. Denna adress kallar vi för "reset vektor adress" därför att det är till denna adress som programexekveringen kommer att hoppa till när man utför en hård reset på PIC'en.
Alla resetter nollställer även innehållet i PCLATH-registret. Detta betyder att vid reset kommer minnesbank PAGE0 automatiskt att vara vald.

Interrupt-vektor

När en interrupt/avbrott har inträffat ändras PC till adress 0004h. Denna adress kallar vi för interruptvektoradressen. Då en interrupt bidrar till att PC adresserar interruptvektorn, påverkas inte PCLATH. Detta betyder att innan man skriver något till PC skall PCLATH specificeras för vilket minnesblock(page) som skall användas. Innan man ändrar innehållet i PCLATH kan det vara en fördel om man dessförinnan har sparat undan det gamla innehållet i PCLATH för senare användning vid återhopp från din interrupt rutin. Innan man hoppar ur sin interruptrutin bör gamla PCLATH värdet återställas så att programmet vet var det befann sig innan interrupten.

Kalibreringsinformation

En del av PIC'arna har kalibreringsdata lagrat i programminnet då de kommer från fabrik. Denna kalibreringsinformation programmeras av Microchip under slutfasen av framställningen av PIC'en. Dessa kalibreringsvärden används för kalibrering av interna RC-oscillatorn till ADC'n .
Kalibreringsvärdena för t.ex. PIC12C67X läggs i slutet på programminnet och hämtas med hjälp av RETLW instruktionen. (Se respektive databok för hur just din PIC lagrar eventuella kalibreringsvärden.)

> OBS När det gäller fönsterkretsar så skall kalibreringsvärdet läsas av för de PIC'ar som har intern oscillator. Skriv ner detta värde, helst på själva kretsen innan radering av kretsen sker.

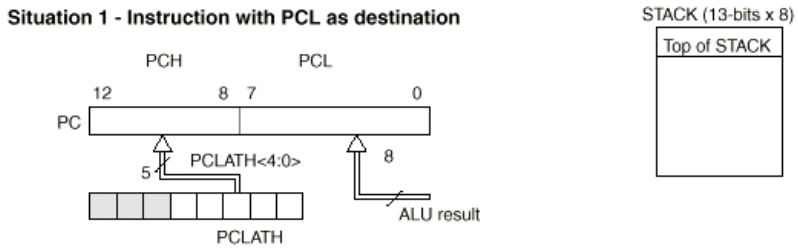
Varning när det gäller code protect på fönsterkretsar.

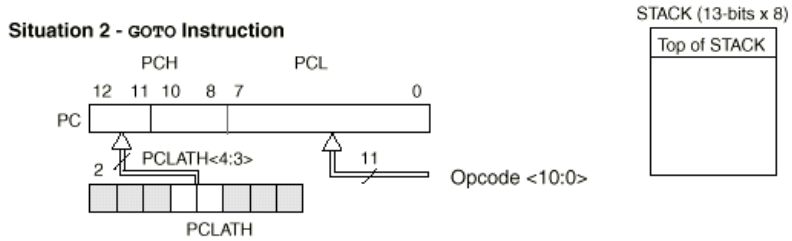
> Varning: Om man sätter CP (Code Protect biten) på en fönsterkrets kan man inte programmera om den, trots att den har raderats med UV-ljus.

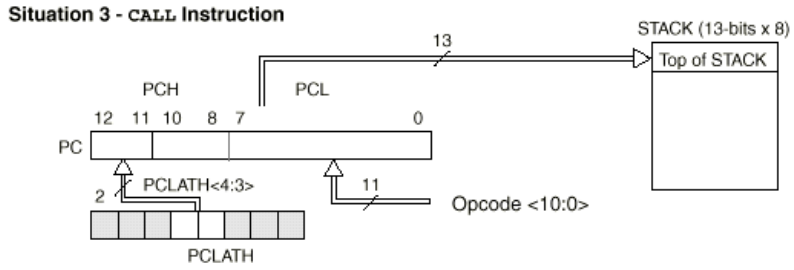
Programräknaren (PC)

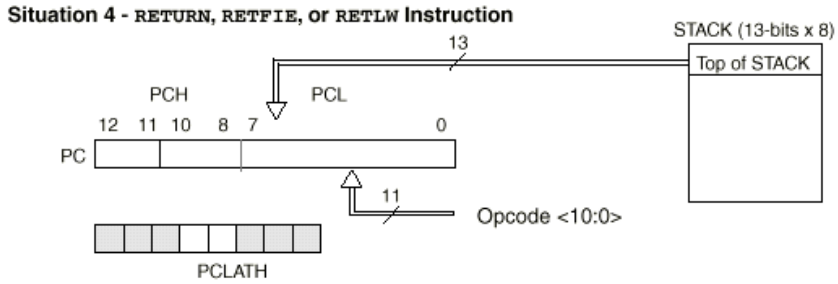
Programräknaren specificerar vilken adress nästkommande instruktion skall hämtas ifrån. Programräknaren (PC) är 13 bitar bred. De 8 minst signifikanta bitarna kallas PCL-registret. Detta register är läs och skrivbart. De mest signifikanta bitarna i PC kallas för PCH-registret. Detta register innehåller bitarna PC<12:8> och går inte att läsa eller skriva till direkt. PCH skriver och läser man till genom ändringar av PCLATH-registret.
Figuren på nästa sida visar olika händelser och modifiering av PC. Situation 1 visar hur PC modifieras under en skrivning till PCL (PCLATH<4:0>→PCH). Situation 2 visar hur PC uppdateras under en *goto* instruktion (PCLATH<4:3>→PCH). Situation 3 visar hur PC laddas under en *call* instruktion (PCLATH<4:3>→PCH), med PC laddad (PUSHas) till toppen på stacken. Situation 4 visar hur PC laddas under en *return* instruktion varvid PC laddas (POPar) sitt värde från stacken.

Uppdatering av PC under olika förhållanden.









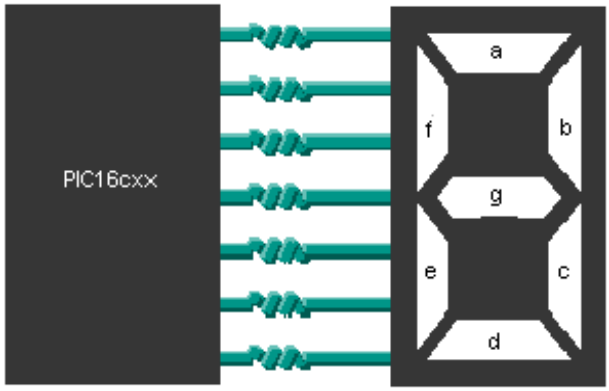
(**Notera att PCLATH aldrig uppdateras med innehållet i PCH**)

Beräknad GOTO / Lookup tabeller

Genom att addera en offset till programräknaren(PC) kan man få en beräknad goto att inträffa (ADDWF PCL). Denna metod används för så kallade lookup-tabeller för omvandling av data. När denna typ av metod används skall man iaktta att den maximala längden på lookup-tabellen begränsas av PCL (om vardera 256 byte).

> Notera att då man skriver till PCL kommer de fem minst signifikanta bitarna i PCLATH att laddas till PCH.

Se **AN556** för ytterligare beskrivning.

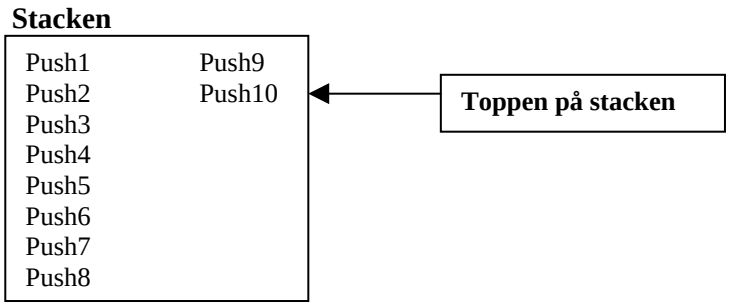


Här är ett exempel på hur man kan använda en lookup tabell för att visa ett decimalt tal på en sjusegmentdsdisplay.
Om VisaSiffra variabeln innehåller det binära talet b'00000101' visar displayen talet 5.

```
                Org         0x10
                Clrf        PCLATH
                Movf        VisaSiffra,W
                Call        SjuSegmentDecode
                Movwf       PORTB
                Goto        Fortsatt
SjuSegmentDecode
                Addwf       PCL                             ;addera PCL med innehåll i W-reg
                            ; '_gfedcba'
                Retlw       b'00111111'   ;decode 0         ;lägger värdet i w-reg och hoppar tillbaka till
                Retlw       b'00000110'   ;decode 1         ;Call instruktionen plus en programrad.
                Retlw       b'01011011'   ;decode 2
                Retlw       b'01001111'   ;decode 3
                Retlw       b'01100110'   ;decode 4
                Retlw       b'01101101'   ;decode 5
                Retlw       b'01111101'   ;decode 6
                Retlw       b'00000111'   ;decode 7
                Retlw       b'01111111'   ;decode 8
                Retlw       b'01101111'   ;decode 9
```

Stacken

Stacken möjliggör upp till 8 stycken i varandra liggande subrutiner (adresser varifrån subrutiner anropats.). Stacken innehåller returadressen varifrån en subrutin eller interruptsubrutin anropades plus en adress. Mid-range PIC'ar har en 8 djup och 13bit bred hårdvarustack. Stackens område finns varken i programminnet eller dataminnet och det går inte att direkt skriva eller läsa från stacken. Programräknaren "PUSHas" till stacken när en *call* instruktion utförs eller att en interrupt orsakar ett hopp. Värdet på stacken "POPas", lyfts ut till PC då en *RETURN, RETLW* eller en *RETEFI* instruktion inträffar. PCLATH modifieras inte när stacken "PUSHas" eller "POPas".
Efter det att stacken har "PUSHats" 8 ggr så kommer den nionde "PUSHen" att skriva över det första värdet som var lagrat från den första "PUSHningen". Den tionde "PUSHen" skriver över det andra värdet som lagrades under den andra "PUSHningen" osv. Se exemplet nedan.



OBS:	Det finns ingen statusbit som indikerar att stackens djup har överskridits.
OBS:	Det finns inga instruktioner som kallas POP eller PUSH. Detta är en åtgärd som inträffar då en *CALL, RETURN, RETLW* eller *RETEFIE* instruktion har utförts, samt att interruptvektor-avbrott har inträffat

Programminnets block-hoppning (paging).

En del PIC'ar har ett programminne som är större än 2K. En *CALL eller en GOTO* instruktion har endast en 11 bitars adresseringslängd. De 11 bitarna kan adressera 2K programminne per block (page).
För att *CALL* och *GOTO* instruktioner skall kunna adressera hela programminnet behövs det ytterligare två bitar för att definiera resterande programminne Dessa bitar kommer från PCLATH<4:3>. När en *CALL* eller en *GOTO* instruktion utförs måste användaren själv se till att page bitarna i PCLATH<4:3> är satt så att den önskade minnesblocket adresseras. När en *RETURN* instruktion utförs tas hela det 13 bitars ordet från stacken. Därför behövs det ej utföras någon manipulation av PCLATH<4:3> under en *RETURN* instruktion.

OBS:	PIC'ar med 2K ord minne eller mindre, behöver inte page bitarna i PCLATH<4:3>. Man bör ej använda PCLATH<4:3> för generella bitlagringspositioner då kompatibilitet för framtida högre nivåer av PIC'ar ej kommer att fungera. Vid programmering av PIC'ar med storleksordningen 2K till 4K ord kan paging bit PCLATH<4> ignoreras då denna bit används för access till page 2 och 3(1000h-1FFFh). Man bör ej använda PCLATH<4> för generella bitlagringspositioner då kompatibilitet för framtida högre nivåer av PIC'ar ej kommer att fungera.

Exemplet nedan visar ett subrutinanrop i page 1 i programminnet. I detta exempel antar vi att PCLATH är undansparad och återskapad i interrupthanteringsrutinen (om en interrupt används).

Subrutinanrop till page1 från page0

```
        ORG     0x500
        BSF     PCLATH,3    ;välj page 1 (800h-FFFh)
        CALL    SUB1_P1     ;anropa subrutin i page1(800h-FFFh)
        :                   ;
        :                   ;
        ORG     0x900       ;
SUB_P1  :                   ;Anropad subrutin page1 (800h-FFFh)
        :                   ;
        RETURN              ;återhopp till anrop av subrutin i pag1 (00h-7FFh)
                            :
```

Dataminnets struktur.

Dataminnet är uppdelat i olika områden: speciella funktionsregister (SFR) och generella register (GPR). De speciella funktionsregistren är till för att kontrollera själva hårdvarans uppträdande hos PIC'en. Generella registren är till för att t.ex. lagra delresultat i ett program.
Dataminnet ligger i olika banker. SFR-register och GPR-register ligger uppdelade i olika banker för att man skall få tillgång till mer än 96 byte generella RAM-register. Då man hoppar från olika banker behöver man använda kontrollbitar för att välja bank. Dessa kontrollbitar finns i status-registret STATUS<7:5>. Bilden på nästa sida visar en minneskarta med olika banker och minnesområden.

För att flytta ett värde från ett register måste detta värde passera w-registret. Detta betyder att det går åt 2 klockcykler för att flytta ett registervärde till ett annat register

Åtkomsten till dataminnet kan ske på två sätt: direkt eller indirekt adressering. Vid direkt adressering kan man ibland behöva använda RP1:RP0 bitarna i statusregistret. Vid indirekt adressering använder man sig av File select registret (FSR). Indirektadressering använder sig av en indirekt registerpekare(IRP). IRP-bitarna i statusregistret väljer bank0, bank1eler bank2 och bank3s minnesarea.

Generella register (GPR).

En del av mid-range PIC'arna har minnesindelning (bankning) för GPR arean. GPR initieras ej vid uppstart av processorn. Under alla resetter utom power on reset (POR) behåller PIC'en sina GPR-registervärden.
Dessa register kan nås antingen via direkt eller indirekt adressering då File Select Register (FSR) används.
En del PIC'ar har minnesareor som delas över de olika bankerna så att en läsning eller skrivning sker i samtliga banker oavsett vilken bank du använder. Detta refererar vi till som allmänna RAM.

Speciella funktionsregister (SFR).

SFR används för att styra och indikera hårdvaruhändelser i och utanför PIC'en. Dessa register är att betrakta som statiska RAM-register.

De speciella funktionsregistren kan delas upp i två grupper. Den första gruppen av SFR har hand om kärnan och den andra SFR-gruppen innehåller periferifunktioner. Här beskrivs de SFR som har att göra med själva kärnan.

Alla mid-range PIC'ar har "bankat" minne för SFR. Att byta mellan dessa banker kräver att man manipulerar PR0:PR1 bitarna i statusregistret. En del SFR initieras vid power on reset (POR) medan andra SFR är opåverkade.

OBS:	De speciella funktionsregistren (FSR) minnesarea kan ha generella register (GPR) mappade på samma minnesarea.

Man kan nå registren på två sätt, antingen direkt eller indirekt då FSR används (File Select Register).

Minneskarta över filregistren

Bank0	Adress	Bank1	Adress	Bank2(5)	Adress	Bank3(5)	Adress
INDF	00h	**INDF**	80h	**INDF**	100h	**INDF**	180h
TMR0	01h	**OPTION_REG**	81h	**TMR0**	101h	**OPTION_REG**	182h
PCL	02h	**PCL**	82h	**PCL**	102h	**PCL**	182h
STATUS	03h	**STATUS**	83h	**STATUS**	103h	**STATUS**	183h
FSR	04h	**FSR**	84h	**FSR**	104h	**FSR**	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h	PORTF	107h	TRISF	187h
PORTD	08h	TRISD	88h	PORTG	108h	TRISG	188h
PORTE	09h	TRISE	89h		109h		189h
PCLATH	0Ah	**PCLATH**	8Ah	**PCLATH**	10Ah	**PCLATH**	18Ah
INTCON	0Bh	**INTCON**	8Bh	**INTCON**	10Bh	**INTCON**	18Bh
PIR1	0Ch	**PIE1**	8Ch		10Ch		18Ch
PIR2	0Dh	PIE2	8Dh		10Dh		18Dh
TMR1L	0Eh	**PCON**	8Eh		10Eh		18Eh
TMR1H	0Fh	OSCCAL	8Fh		10Fh		18Fh
T1CON	10h		90h		1110h		190h
TMR2	11h		91h		111h		191h
T2CON	12h	PR2	02h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPATAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h		116h		196h
CCP1CON	17h		97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
XTREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch		9Ch		11Ch		19Ch
CCP2CON	1Dh		9Dh		11Dh		19Dh
ADRES	1Eh		9Eh		11Eh		19Eh
ADCCON0	1Fh	ADCON	9Fh		11Fh		19Fh
Generella Register(2)	20h – 7Fh	Generella Register(3)	A0h – EFh	Generella Register(3)	120h – 16Fh	Generella Register(3)	1A0h – 1Efh
		Mappad i Bank0 70h-7Fh(4)	F0h – FFh	Mappad i Bank0 70h-7Fh(4)	170h – 17Fh	Mappad i Bank0 70h-7Fh(4)	1F0h – 1FFh

Notera:

1: Fetstilade register finns i alla mid-range PIC'ar.
2: Det är inte alla av dessa register som är implementerade och kommer därför att läsas som '0'.
3: Dessa register kan ibland vara oimplementerade beroende på vilken PIC du använder. Se databok för din specifika PIC.
4: En del PIC'ar mappar inte dessa register i bank0. I de PIC'ar som mappar dessa register i bank0 kommer dessa register att betraktas som vanligt RAM
5: En del PIC'ar använder ej dessa register och i de fallen kommer dessa register att läsas som'0'.
6: Generella register (GPR) kan vara belägna i SFR arean

Bankning

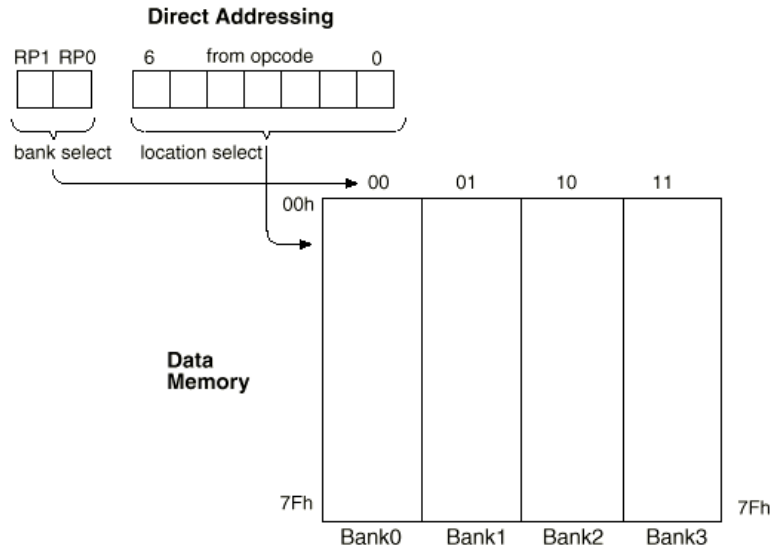
Dataminnet är indelat i fyra olika banker. Varje bank innehåller GPR och SFR. För att växla mellan de olika bankerna krävs det att man manipulerar RP0 och RP1 bitarna i statusregistret (vid direkt adressering). IRP bitarna i statusregistret är till för indirekt adressering.

Direkt och indirekt adressering av bankerna.

Vald bank	Direkt Adressering (RP1:RP0)	Indirekt Adressering (IRP)
0	00	0
1	01	
2	10	1
3	11	

Var och en av bankerna sträcker sig till 7Fh (128 byte). Den lägre delen av bankerna är reserverade för SFR. Ovanför FSR finns GPR. All data finns lagrat i statiska RAM. Alla banker kan innehålla SFR. En del av SFR i bank0 är speglad vidare till andra banker. Detta för att underlätta programmeringen samt för att reducera accesstiderna i programmet (se tabell på föregående sida).

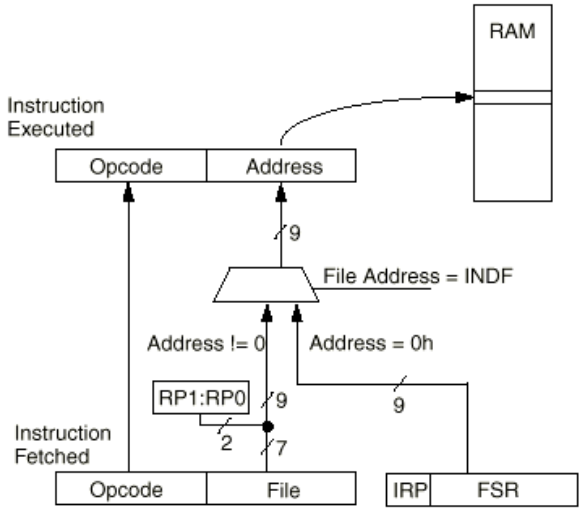
Det finns en variation mellan olika PIC'ar med avseende på minneslayouten på dataminnet. Den dataminneskonfiguration som är planerad att bli en standard för mid range PIC'ar kan du se på föregående sida. Denna minneskarta har de sista 16 byten mappad genom alla minnesbanker. Detta för att reducera programhopp.

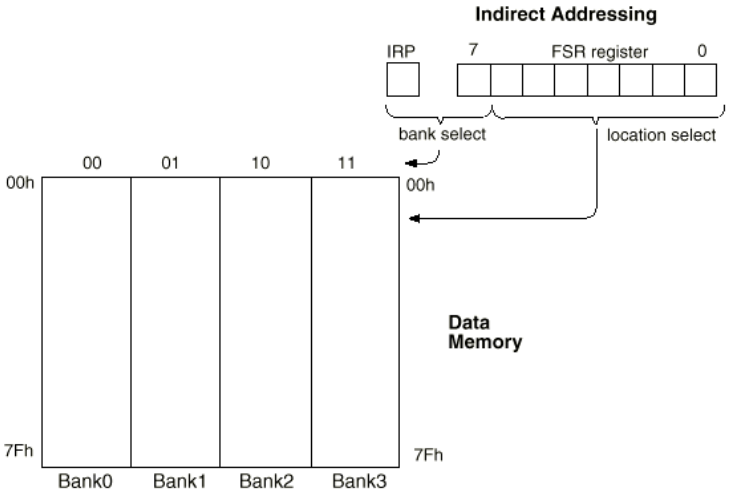


Indirekt adressering, INDF och FSR registren.

Indirekt adressering används till att adressera dataminne som har sin början på variabla platser i dataminnet. File Select Registret FSR används som en pekare till var dataminnet för läsning eller skrivning är belägen. Detta kan vara bra att ha när man vill ha datatabeller i dataminnet. Datatabellerna kan via Indirekt adressering med lätthet skickas till t.ex. en USART. Figuren nedan visar hur ett värde flyttas till dataminnets adress som specificeras med hjälp av FSR registret.
Indirekt adressering använder INDF registret. Alla instruktioner som använder INDF registret når registerpekaren genom File Select Register FSR. Om man läser själva INDF registret kommer indirekt (FSR=’0’) att läsas som 00h. Om man indirekt skriver till INDF registret resulterar detta i en no-operation (men statusbitar kan dock påverkas). En 9-bitars adress genereras då sammanslagning mellan IPR bit(STATUS<7>) och det 8-bit långa FSR registret görs (se bilden nederst på denna sida.).
Om IRPbiten (STATUS<7>)=0 adresseras bank 0 och bank 1 (00h-FFh).
Om IRPbiten (STATUS<7>)=1 adresseras bank 2 och bank 3 (100h-1FFh).
(IRP biten används inte på PIC16F8X och bör vara satt till noll)

Inderekt Adressering





Exempel på indirekt adressering.

Detta exempel använder sig av indirekt adressering för att initiera RAM adresser (belägna mellan 20h-2Fh) och med ett minimum av instruktioner. En liknande metod kan användas till att definiera ett block med data till USART's sändningsregister (TXREG). Startadressen i blocket med data som skall sändas kan enkelt ändras av programmet.

```
        BCF     STATUS, IRP     ;Indirekt adressering Bank0/1.
        MOVLW   0x20            ;Initiering av RAM pekare.
        MOVF    FSR             ;
NEXT    CLRF    INDF            ;Nolla INDF registret.
        INCF    FSR,F           ;Öka pekaren med ett.
        BTFSS   FSR,4           ;Färdigt?
        GOTO    NEXT            ;NEJ, hoppa till NEXT.
CONTINUE
        :                       ;JA, Fortsätt.
        :
```

Initiering med direkt adressering.

Detta exempel visar hur man byter banker vid direkt adressering.

```
        CLRF    STATUS          ;Nolla STATUS register (Bank 0)
        :
        BSF     STATUS, RP0     ;Bank1
        :
        BCF     STATUS, RP0     ;Bank0
        :
        MOVLW   0x60            ;Sätt RP0 och RP1 bitarna i
        XORWF   STATUS, F       ; STATUS-registret och övriga
        :                       ; bitar opåverkade (Bank3)
        BCF     STATUS, RP0     ;Bank2
        :
        BCF     STATUS, RP1     ;Bank0
```

Noll initiering av GPR adresser med hjälp av FSR.

```
                CLRF        STATUS      ;Nolla STATUS register.
                MOVLW       0x20        ;Första adressen i GPR arean (i banken).
                MOVWF       FSR         ;Flytta föregående adress till det indirekta adressregisteret.
Bank0_LP
                CLRF        INDF0       ;Nolla GPR på den adress som anges av FSR.
                INCF        FSR         ;Följande GPR (RAM) adress.
                BTFSS       FSR,7       ;Slutet på denna bank? (FSR=80h, C=0)
                GOTO        Bank0_LP    ;Nej, nolla nästa GPR adress.
;
;Nästa bank (Bank1)
;(** Behövs endast om PIC'en har BANK1 **)
;
                MOVLW       0xA0        ;Första adressen i GPR arean (i banken).
                MOVWF       FSR         ;Flytta föregående adress till det indirekta adressregisteret.
Bank1_LP
                CLRF        INDF0       ;Nolla GPR på den adress som anges av FSR.
                INCF        FSR         ;Följande GPR (RAM) adress.
                BTFSS       STATUS,C    ;Slutet på denna bank? (FSR=00h, C=1)
                GOTO        Bank1_LP    ;Nej, nolla nästa GPR adress.
;
;Nästa bank (Bank2)
;(** Behövs endast om PIC'en har BANK2 **)
;
                BSF         STATUS,IPR  ;Välj Bank2 och bank 3 för indirekt adressering.

                MOVLW       0x20        ;Första adressen i GPR arean (i banken).
                MOVWF       FSR         ;Flytta föregående adress till det indirekta adressregisteret.
Bank2_LP
                CLRF        INDF0       ;Nolla GPR på den adress som anges av FSR.
                INCF        FSR         ;Följande GPR (RAM) adress.
                BTFSS       FSR,7       ;Slutet på denna bank? (FSR=80h, C=0)
                GOTO        Bank2_LP    ;Nej, nolla nästa GPR adress.
;
;Nästa bank (Bank3)
;(** Behövs endast om PIC'en har BANK3 **)
;
                MOVLW       0xA0        ;Första adressen i GPR arean (i banken).
                MOVWF       FSR         ;Flytta föregående adress till det indirekta adressregisteret.
Bank3_LP
                CLRF        INDF0       ;Nolla GPR på den adress som anges av FSR.
                INCF        FSR         ;Följande GPR (RAM) adress.
                BTFSS       STATUS,C    ;Slutet på denna bank? (FSR=00h, C=1)
                GOTO        Bank3_LP    ;Nej, nolla nästa GPR adress.
                :                       ;Ja alla GPR (RAM) är Nollade.
```

Tips.
När du programmerar en PIC som har större programminne än 2K ord kan du få konstiga adresshopp då du ropar på en subrutin med call-instruktionen. Detta beror på att PCLATH måste laddas med rätt adressvärde innan en call eller goto instruktion utförs. Detta behövs för att specificera den rätta minnessidan (page) som subrutinen befinner sig i.
Följande programmsnutt visar hur detta kan gå till.:

```
            MOVLW     HIGH (SUB_1) ;Välj vilken minnessida som subrutinen är belägen i.
            MOVWF     PCLATH       ;Uppdatera PCLATH med denna info.
            CALL      SUB_1        ;Ropa på den önskade subrutinen.
            :
            :
            :
SUB_1       :                      ;Subrutinens start (belägen på sida ??? ”page”)
            :
            RETURN                 ;Återvänd från subrutin.
```

Förslag på grundmall till programuppställning.
Under MPLAB\TEMPLATE\CODE\….. finns det färdiga templates (huvuddrag av program skrivna för att komma igång med just den PIC som du har valt att arbeta med)
För en PIC16C84 heter denna template f84temp.asm.

Se följande sidor för att se hur en template kan se ut:

```
;*****************************************************************************
;   This file is a basic code template for assembly code generation
;   on the PICmicro PIC16F84. This file contains the basic code
;   building blocks to build upon.
;
;   If interrupts are not used all code presented between the ORG
;   0x004 directive and the label main can be removed. In addition
;   the variable assignments for 'w_temp' and 'status_temp' can
;   be removed.
;
;   Refer to the MPASM User's Guide for additional information on
;   features of the assembler (Document DS33014).
;
;   Refer to the respective PICmicro data sheet for additional
;   information on the instruction set.
;
;   Template file assembled with MPLAB V3.99.18 and MPASM V2.15.06.
;
;*****************************************************************************
;
;   Filename:                                   xxx.asm
;   Date:
;   File Version:
;
;   Author:
;   Company:
;
;
;*****************************************************************************
;
;   Files required:
;
;
;
;*****************************************************************************
;
;   Notes:
;
;
;
;
;*****************************************************************************
```

```
        list      p=16F84           ; list directive to define processor
        #include <p16F84.inc>   ; processor specific variable definitions

        __CONFIG   _CP_OFF & _WDT_ON & _PWRTE_ON & _RC_OSC

; '__CONFIG' directive is used to embed configuration data within .asm file.
; The lables following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.



;***** VARIABLE DEFINITIONS
w_temp        EQU     0x0C       ; variable used for context saving
status_temp   EQU     0x0D       ; variable used for context saving


;**********************************************************************
        ORG     0x000           ; processor reset vector
        goto    main            ; go to beginning of program


        ORG     0x004           ; interrupt vector location
        movwf   w_temp          ; save off current W register contents
        movf    STATUS,w        ; move status register into W register
        movwf   status_temp     ; save off contents of STATUS register

; isr code can go here or be located as a call subroutine elsewhere


        movf    status_temp,w   ; retrieve copy of STATUS register
        movwf   STATUS          ; restore pre-isr STATUS register contents
        swapf   w_temp,f
        swapf   w_temp,w        ; restore pre-isr W register contents
        retfie                  ; return from interrupt


main

; remaining code goes here


        END                     ; directive 'end of program'
```

Referenslitteratur.

PICmicro Mid-Range MCU Family Reference Manual	DS330023A
Embedded control handbook	DS000992D
MPLAB IDE User's Guide	DS51025
MPASM Assembler User's Guide	DS33014F
IN-Circuit Serial Programming Guide (ICSP)	DS30277B
Mikrodatorteknik	ISBN 91-634-1593-2
A Beginners Guide To The Microchip PIC	ISBN 1-899013-01-6

Inrapportering av fel i denna guide.

De fel som lärare och elever upptäcker i denna guide kan meddelas via post till Memec Scandinavia AB.
Märk kuvertet med "PIC-Guiden".
Vi välkomnar eventuell synpunkter och tillägg

Memec Scandinavia AB
Sehlstedtsgatan 6
115 28 Stockholm

Version 1.2
1999-02-17