Indledning

PIC16F84 er en minimal mikroprocessor, som har sine anvendelser hvor der ikke er brug for det store antal porte og fleksibilitet i programmering som en normal mikroprocessor stiller til rådighed, men hvor en mikroprocessor stadig kan være til gavn.

Jeg har her forsøgt dels at give læseren mulighed for at vurdere om en PIC vil være det rigtige valg til en given opgave, dels at beskrive brugen af PIC16F84 ud fra mine egne erfaringer. Jeg forudsætter at læseren ved lidt om mikroprocessorer i forvejen, fx fra fagene 49116 Digitalteknik og/eller 50250 Mikrodatamatsystemer.

PIC16F84 er ikke den eneste lille single−chip mikroprocessor på markedet. Der findes bl.a. Atmel−serien, Scenix−processorerne og BasicStamp, der er bygget over en PIC. Der findes også både større og mindre PIC−processorer, bl.a. 12C−serien i 8−bens hus og 17C−serien med indbyggede A/D−konvertere og mange andre features. Jeg har ikke selv nogle praktiske erfaringer med disse systemer og kan derfor ikke sammenligne dem, men hvis læseren på forhånd kender et andet system skulle det være muligt at sammenligne det med PIC16F84 ud fra denne manual.

Jeg har vedlagt Microchips datablad for processoren til referencebrug samt et udvalg af application notes fra Parallax, som beskriver nogle praktiske applikationer af PIC'en. Desuden har jeg vedlagt to eksempler på hvad processoren kan bruges til, stort set uden kommentarer. Ellers er PIC'en så udbredt en hobbyprocessor at det er muligt at finde virkelig mange eksempler på brug af den ved en søgning på nettet.

Hvis du finder fejl, har forslag eller spørgsmål, er du velkommen til at kontakte mig.

Niels Holmgård Andersen, december 1999
c973399@student.dtu.dk

Indhold

Lynoversigt til beslutningsbrug

De hårde facts

For de tekniske og dem med hang til tal og akronymer:

- 1 kword Flash−hukommelse til program og data (1k x 14 bits)
- 68 bytes RAM
- 64 bytes EEPROM
- 13 general purpose I/O−ben fordelt på to porte, RA (5 bit) og RB (8 bit). Hvert ben kan source 20 mA eller sinke 25 mA[1]
- En 8 bit timer/counter med 8−bit prescaler
- 2 eksterne og 2 interne interrupts[2]
- Watchdog−timer med intern RC−clock, der deler prescaleren med counter−modulet
- DC til 10 MHz clock−frekvens[3] med krystal eller RC−generator[4]
- En 8 bit akkumulator (Svarende til register A i Intels 8051, AL i 8086, A i Motorolas 6805)
- RISC−arkitektur[5] (kun 35 instruktioner, der alle[6] udføres på 1 instruktionscyklus = 4 clockcykler)
- Harvard−arkitektur med separat instruktions− og databus (6 bit instruktionsbredde, 8 bit data)

Fordele

PIC'en er meget mindre på alle skalaer end et traditionelt mikroprocessorsystem med en CPU, RAM, ROM og I/O−porte. Det er netop dette der er dens force, men for at få lidt tal på:

- PIC'en er nem og overskuelig at programmere, kan programmeres i C[7], egen RISC−assembler[8] eller 8051−assembler[9]
- PIC'en kan programmeres hurtigt og nemt, også i det kredsløb den skal styre, med en PC med RS232 seriel port og et par modstande.[10] Flash−hukommelsen gør at den kan slettes elektrisk og omprogrammeres lige så hurtigt, hvilket gør det nemt lige at teste en lille ændring.
- PIC'en bruger kun omkring 2 mA ved normalt brug[11] og kun omkring 1 uA i standby.
- PIC'en fylder kun en enkelt 18−bens DIL−kreds for 13 klar−til−brug udgange.
- Prisen for en PIC16F84 er kun 30 kroner[12], hvilket man ikke får meget processor med ekstern RAM og ROM for, og da slet ikke hvis man regner det nødvendige dobbeltsidede print og glue logic med. Prisen er faktisk så lav at PIC'en er konkurrencedygtig selv ved bygning af relativt små sekvensmaskiner, som man ellers ville bygge af MSI−kredse eller PAL/GAL.

1 Dog ikke alle sammen samtidig
2 Level−trigget på RB0, kanttrigget på RB4−7, TMR0 overflow og EEPROM write complete.
3 For PIC16F84−10, DC til 4 MHz for PIC16F84−04
4 Begge generatortyper har samtlige aktive komponenter indbygget i PIC'en
5 Reduced Instruction Set Computer, en computer med et lille og simpelt instruktionssæt
6 Bortset fra Jump−instruktioner
7 Ved hjælp af Pavel Baranovs C2C, se denne
8 Ved hjælp af Microchip MPASM, se denne
9 Ved hjælp af Parallax SPASM, se denne
10 Ved hjælp af LudiPipo og PIP02, se denne
11 Egetforbrug @ 5V, 4MHz
12 $103 for 25 stk for 10 MHz versionen jfr. DigiKey september 1999

- PIC'en har alle de "hurtige" signaler internt (ved lave kapaciteter og strømme), hvilket gør støjproblemer og strømforsyningsstøj meget mindre end ved et fuldskala mikroprocessorsystem.
- PIC'en kan køre på spændinger fra 2−6V og er ikke særlig kræsen med forsyningens kvalitet.
- PIC'ens udgange er kraftige nok[13] til uden ekstra transistorer eller buffere at drive lysdioder og lignende belastninger.

Ulemper

PIC'en kan klare imponerende mange opgaver på et lille areal, med lille effektforbrug og for små penge, men det er samtidig dens lidenhed, der sætter grænserne for hvad den kan bruges til. Et oprids:

- PIC'ens ret lille programhukommelse gør det begrænset hvor mange og hvor avancerede funktioner den kan styre.
- Den meget lille RAM – 68 bytes – gør den ubrugelig som datalogger.
- Operationshastigheden er godt nok ganske høj, op til 2,5 MIPS[14] ved 10 MHz, men funktionerne er simple, hvilket gør PIC'en dårlig til matematik. Desuden er dens plads til variable og tabeller meget begrænset, så skal der regnes seriøst på noget, så brug en "rigtig" processor. Er programmet kompliceret, med mange funktioner og eventuelt med rekursion, vil PIC'en ikke kunne udføre det, da RAM− og stakplads er meget begrænset.
- PIC'en har ingen DIV−instruktion – divisioner må man selv programmere.
- Antallet af I/O−porte er ikke overvældende stort. Skal der bruges mange porte – dvs. flere end der umiddelbart kan multiplexes fra PIC'ens 8+5 portben – er et større system indlysende nødvendigt.
- PIC'ens on−chip perifære kredsløb er ikke særlig avancerede. Den har fx ikke større mikrocontrollere/processorers mulighed for at modtage eller sende serielt samtidig med at den laver andre ting. Den kan godt kommunikere serielt, men så er det det eneste den laver.
- PIC'ens timer/counter er kun på 8 bit og har ingen hardware udgange, hvilket gør det til et fuldtidsjob for processoren fx at lave et PWM−signal.

Konklusionen er at PIC'en er nemmere, hurtigere og billigere at bruge til små opgaver end et standard mikroprocessorsystem, men at den ikke kan erstatte en "rigtig" processor til større, RAM− eller regneintensive opgaver eller opgaver der kræver at flere ting sker simultant.

Gode PIC−opgaver

Mindre mikroprocessorstyrede måleinstrumenter med pc−interface.
Coprocessor til styring af periferienheder i et større mikroprocessorsystem, fx stepmotorkontrol via seriel bus eller kontrol af en ultralydsafstandsmåler.
Små selvstændige reguleringsopgaver evt. med styring fra mikroprocessor eller pc, fx at holde temperatur og luftfugtighed konstant eller styre hastigheden af en DC−motor med tilbagekobling.
Modtagelse og dekodning af serielle signaler, fx infrarød kommunikation, kvadratur−

13 Udgangsstrøm max 20 mA pr. pin, max 100 mA i alt, indgangsstrøm max 25 mA pr. pin, max 150 mA i alt
14 Millioner instruktioner pr. sekund

kodning mv.

Som eksempel på hvad PIC−familien kan klare er der blevet implementeret en komplet HTTP/1.0 internetserver på en enkelt PIC[15].

15 http://www−ccs.cs.umass.edu/~shri/iPic.html

Alle disse facts kan være nok så interessante, men for at få PIC'en til at løse en konkret opgave skal man, udover at have designet et kredsløb, hvilket bliver behandlet i afsnittene efter *Basiskredsløb for PIC16F84,* have skrevet et program og have overført det til den. Overførslen bliver beskrevet i afsnittet *Programmering af kredsen*. Programskrivningen bliver behandlet her.

Programmering af PIC16F84 kan som på enhver anden mikroprocessor gøres i flere sprog. Der er tre umiddelbart tilgængelige sprog. PIC'ens modersmål er dens egen RISC−assemblersprog, der er på kun 35 instruktioner og kompileres til maskinkode med Microchips egen MPASM. Dette sprogs mnemonics er meget forskellige fra både Motorolas og Intels varianter, der vel er dem de fleste kender. En anden assemblermulighed er at bruge Parallax' SPASM, der ligner Intel 8051−sproget til forveksling og virker ved simpelthen at oversætte Intels mnemonics til mellem 1 og 4 af de tilsvarende Microchip−maskinkodeinstruktioner afhængig af om instruktionen umiddelbart kan udføres af PIC'en eller om den må udføres i flere trin. Ved begge disse sprog får man fordelene af at kunne klemme den sidste smule ydelse ud af processoren og ved at kende den præcise udførselshastighed ved at tælle maskincycles. Hvis den ydeevne og kontrol ikke er nødvendig kan PIC'en programmeres i C, Pascal, Basic, Forth og diverse andre højniveausprog, hvoraf jeg har valgt C som standardsprog. Der findes flere C−kompilere til den, her har jeg valgt en freeware demoudgave af et kommercielt produkt, som i sig selv er fuldt funktionsdygtig.

Programmering i C med C2C

Den C−kompiler der ligger vedlagt er Pavel Baranovs C2C[16], der oversætter en C−fil til en assembler−fil, som er spiselig for MPASM. Baranov sælger også en kommerciel udgave med flere faciliteter, men freeware−versionen er næsten ikke begrænset og er uden videre brugbar. Jeg forudsætter at læseren er bekendt med C og vil derfor ikke forsøge at skrive en manual i C−programmering, men nøjes med at beskrive forskellene til standard C.

C2C implementerer ikke hele ANSI−C, men nøjes med følgende delmængde:

Kodestrukturer:

if, else; switch, case, default;	Betingelser
while, for, break, continue;	Løkker
goto, labels;	Spring
char funktion1 (char i, short p) , return;	Funktioner med returtypen void eller char, med enhver parameter
extern, asm { og *};*	Ekstern eller indbygget assembler

C2C implementerer alle standard kodestrukturer.

Operatorer og datatyper:

~, ++, −−, +, −, <, <=, >, >=, ==, !=, =, !, &, |, ^, &=, |=, ^=, &&, ||, *, /, %, <<, >>, <<=, >>=

char, short, void	På henholdsvis 8 bit unsigned, 8 bit signed, ingenting
int, long	16 bit signed, virker ikke i den gratis version

16 http://www.geocities.com/SiliconValley/Network/3656/c2c/c.html

char[], short[]	En−dimensionelle arrays
const	Konstanter (og konstante arrays)
*const **	Konstante pointere

C2C mangler muligheden for pointeraritmetik, da det ikke er muligt at lave variable pointere. Også flerdimensionelle arrays mangler, ligesom floating point ikke eksisterer i sproget. Det ville også fylde meget af den begrænsede kodeplads i PIC'en at implementere det, så det er ikke nogen alvorlig mangel.

Kompilerdirektiver:

#include	Henter biblioteker ind
#define, #undef	Definerer kompilervariable
#ifdef, #ifndef, #else, #endif	Reagerer på kompilervariable
@	Placerer en variabel eller funktion på en absolut adresse
#pragma	Sætter parametre bl.a. for de indbyggede RS232−funktioner

Konstanter kan udover standard decimal (124) og hexadecimal (0x7C) angives binært med notationen 01111100b (altså et lille b efter et binært tal).

Da C2C er specielt lavet til at programmere PIC− og lignende processorer ligger der udover standardsproget også nogle funktioner til at styre de enheder der er i en PIC. Disse burde ifølge ANSI−C standarden egentlig ligge i et library for PIC'en, men de er altså indbygget i sproget og gør det lettere for programmøren at skrive hardwarenære programmer:

clear_wdt, enable_interrupt, disable_interrupt, set_mode, set_option, set_tris_a, set_tris_b, set_tris_c, output_port_a, output_port_b, output_port_c, output_high_port_a, output_high_port_b, output_high_port_c, output_low_port_a, output_low_port_b, output_low_port_c, input_port_a, input_port_b, input_port_c, input_pin_port_a, input_pin_port_b, input_pin_port_c, sleep, nop, set_bit, clear_bit, putchar, getchar, delay_s, delay_ms, delay_us, char_to_bcd, bcd_to_char

Der er også flere reserverede funktionsnavne i C2C, fx `void main (void)` og `void interrupt (void)` der er henholdsvis startfunktionen (som i standard C) og interrupthandleren. Desuden er alle hardwareregistrenes adresser lagt som navngivne konstanter i C2C. For yderligere information henvises til den dokumentation der følger med compileren og som ligger vedlagt på disketten.

Den kommercielle version udvider den gratis med bl.a. bedre optimering valgfrit mod mindre kodestørrelse eller større hastighed, 16−bit matematik samt en mulighed for at indsætte C−kildekoden i den genererede assemblerfil til brug ved manuel optimering eller debugging.

Eksempel

Der er virkelig mange steder på nettet som omhandler PIC16F84[17] eller dens næsten−tvilling, PIC16C84. C−verisonen er kodekompatibel med F−versionen, men har en smule mindre RAM og har EEPROM i stedet for FLASH−ROM. Der er eksempler på stort set hvad som helst. Et godt udgangspunkt er David Taits PIC−links[18] eller Tomi Engdahls mere generelle elektronikside[19].

Jeg har valgt at gengive et program, der er et eksempel på hvordan en PIC kan bruges som computerinterface i et instrument. Opgaven er at tælle impulser fra en geigertæller og udlæse dem på en meget simpel parallel bus, med andre ord at opføre sig som en stor adresserbar tæller. Den kan desuden styre nogle få ting i geigertælleren ud fra samme kommandobus. Diagrammer til elektronikken i tælleren er vedlagt som bilaget *En indbygget PIC*.

PIC'ens port B er delt op i en 4 bit kommandobus, der styres af en printerport på en pc, og en 4 bit databus, der driver indgangene på samme printerport. Kommunikationen er ikke clocket og der er ingen handshaking i implementationen bortset fra en særlig test−kommando, som pc−softwaren bruger til at finde ud af om instrumentet er tilsluttet. Primitivt, men det virker, og det er et eksempel på hvor lidt der skal til at implementere et simpelt computerinterface fra et instrument med en PIC. PIC'ens pin A4 bliver brugt som indgang til tælleren TMR0 for at tælle pulser fra et geigerrør, pin A3 som udgang til at kortslutte detektoren og dermed stoppe instrumentet. Der er dermed 3 pins i overskud til udbygninger.

PIC'en reagerer på 15 kommandoer fra pc'en og implementerer følgende protokol:

00	NOP	Ingen kommando
01	LL	Udlæs laveste bytes laveste 4 bits
02	LH	Udlæs laveste bytes højeste 4 bits
03	ML	Udlæs midterste bytes laveste 4 bits
04	MH	Udlæs midterste bytes højeste 4 bits
05	HL	Udlæs højeste bytes laveste 4 bits
06	HH	Udlæs højeste bytes højeste 4 bits
07		Ikke brugt
08		Ikke brugt
09	PNG	Ping PIC – en kontrol af at der er kontakt. Blinker med porten (0000−1111)
10	ON	Start tællere ved at slippe kortslutningen af detektoren
11	OFF	Stop tællere ved at kortslutte nederste modstand i detektorens spændingsdeler.
12	RST	Reset tællere
13	LAP	Lap (mellemtid). Gem tællere i midlertidige variable før udlæsning
14	DRD	Direct Read. Sæt udlæsning til at foregå direkte fra tællerregistrene.
15	STA	Udlæs status

Koden er skrevet i C2C og dokumenteret på en mærkelig blanding af dansk og engelsk.

17 En søgning på Altavista giver følgende antal dokumenter: PIC16F84: 4071; PIC16C84: 6885; PIC: 1283569

18 http://www.labyrinth.net.au/~donmck/dtait/piclinks.html. Dette er et mirror af det oprindelige site, der er blevet nedlagt. Der mangler dog stadig en klar overtager, så i det mindste nogle måneder endnu (fra december 1999) vil denne side være det bedste udgangspunkt. Ellers søg selv – og du skal finde. Der er nok af det.

19 http://www.hut.fi/Misc/Electronics/

Princippet i programmet er en uendelig løkke der kontinuert checker kommandoen fra computeren på de lave bit af port B og ændrer PIC'ens outputs i overensstemmelse med kommandoen. Tællevirksomheden, der måske umiddelbart synes at burde optage det meste af programmet, ligger dels i hardware og dels i interruptrutinen. Den interne tæller, der har en størrelse på 8 bit, tæller opad på eksterne pulser fra detektorkredsløbet. Når den løber over giver den et interrupt, der tæller en variabel i software op. Hvis denne variabel er 255 er der tale om et overflow, og næste variabel tælles en op. Alt i alt klarer 3 linier kode altså at opdatere en 24 bit tæller. Der går så nogle linier med at sætte interrupts op og erklære variable, men det er stadig en ret nemt bygget tæller.

```
char med_byte, high_byte; //counter bytes
char b_TMR0, b_med_byte, b_high_byte; //lap counter bytes
char command, status, lap;
char pinged; //temporær variabel til at blinke med porten

//TMR0 Overflow handler
//Denne funktion tager sig af selve tællingen. Kaldes når TMR0 løber over.
void tmrHandler( void )
{
     if (med_byte == 255) {
          high_byte++;
     }
     med_byte++;
}

void count(void)
{
     //reserveret til fremtidige udvidelser – kaldes hele tiden.
}

//Communicate klarer kommunikationen med computeren.
void communicate(void)
{
     //Fetch and decode command
     command = input_port_b();
     command = command >> 4; //Da kommandoen ligger på de 4 MSB på port B
                             //skiftes den ned med Shift Left til 0..16
     //NOP – no operation
     if (command == 0) {
          //tæl videre
     }
     //isolate wanted nibble and return –
     if (command == 1) { //læs laveste 4 bits fra tæller ud
          if (lap == 1) output_port_b(b_TMR0); //buffered
          else output_port_b(TMR0); //direkte – vælges alt efter LAP
     }
     if (command == 2) { //H T'ller – 16–128
          if (lap == 1) output_port_b(b_TMR0 >> 4);
          else output_port_b(TMR0 >> 4);
     }

     //osv. – samme kodestruktur

     if (command == 9) { //Ping PIC
          if (pinged == 1) {
               output_port_b(255);
               pinged = 0;
          } else {
               output_port_b(0);
               pinged = 1;
          }
     }
     if (command == 10) { //Lap count (store and return stored from now on)
          lap = 1;
          b_high_byte = high_byte; //Store all counters in variables
          b_med_byte = med_byte;
          b_TMR0 = TMR0;
     }
     if (command == 11) { //Lap count off
          lap = 0;
     }
     if (command == 12) { //Reset Counter
          output_port_b(0);
          high_byte = 0;
          med_byte = 0;
```

```
          TMR0 = 0; //will probably be run several times by host computer
                    //due to the speed of the PIC (1 MIPS) – thus rendering
                    //unnessesary the double reset approach to prevent drip–
over
     }
     if (command == 13) { //Run on – StartCount
          output_port_a(0);    //unjam detector & set statusbit
     }
     if (command == 14) { //Run off – StopCount
          output_port_a(255);  //jam detector by shorting lower resistor
     }
     if (command == 15) { //Statusbits – Reset, Run mv.
          output_port_b(status);
     }
}

//This is the interrupt handler
//Denne funktion kaldes automatisk når der sker et interrupt. Den undersøger
hvilket
//interrupt der er tale om og kalder den tilsvarende funktion.
void interrupt( void )
{
     if (INTCON & 4) //TMR0 Overflow
     {
          clear_bit(INTCON, 2); //Dette interrupt er nu behandlet
          tmrHandler();
     }
}

//This is the main function which is called
//after processor power–up or reset
void main( void )
{
     //Hardwareinitialisering
     set_bit( STATUS, RP0 ); //Select the Register bank 1
     OPTION_REG = 233; //Configure the OPTION register – sætter prescaler
til 1 og                          starter counteren i interrupt–mode
     clear_bit( STATUS, RP0 ); //Select the Register bank 0
     INTCON = 160; //Configure the INTCON register – enabler counter–
interrupt

     //Initialisering af porte
     set_tris_a(16); //Configure the Port A to inputs except RA4, the
counter
     output_port_a(255);
     set_tris_b(240); //Configure the Port B to lower 4 bits outputs, rest
inputs
     output_port_b(255);

     //reset variable
     high_byte = 0;
     med_byte = 0;
     TMR0 = 0;

     //Start endless loop
     while( 1 )
     {
          //reserveret til instrumentets fremtidige funktioner
          count();
          //kommuniker med computeren
          communicate();
     }
  //selve tællingen sker i 'baggrunden' vha interrupts, der har højeste
prioritet
}
```

Et relativt kort program og en PIC udfører dermed en opgave der ellers skulle en dusin chips og en hel del designarbejde til at klare. Det allerbedste ved denne løsning ligger dog (i overført betydning) i rutinen `void count (void)`, der er reserveret til fremtidige udvidelser. Programmet fylder i sin nuværende form kun 369 ord ud af 1024 og det er meget lidt processorintensivt. PIC'en kunne nemt lave meget mere – den spilder langt det meste af sin tid med at vente på pc'en og næste impuls fra geigertælleren. Den kunne fx holde styr på strålingens trend integreret over de sidste par minutter og udlæse det, så man kunne lege tampen brænder med instrumentet eller det kunne omforme de enkeltklik som kommer fra detektoren i instrumentet til en tone proportional med intensiteten og dermed lave en mere umiddelbar udlæsning. Det kunne også interface til

et IIC−display og vise samtlige nævnte informationer i klartekst.

Generering af kode

Når programmet er skrevet skal det kompileres og assembleres, hvilket som beskrevet i starten er en totrinsproces. C2C genererer ud fra .c−koden en .asm−fil, der indeholder MicroChip−assembler. Denne kode assembleres til en .hex−fil, der indeholder de bytekoder der skal lægges i PIC'ens programhukommelse. Penslet helt ud sker der (i dos−versionen) følgende:

```
c2c geiger.c
```

C2C producerer følgende output:

```
C2C−pro 2.23 beta C−compiler.  Copyright(c) 1998−99 by Pavel Baranov.
pbaranov@spb.lucent.com
Free copy

geiger.c

Finished.
```

Der kan (og vil sandsynligvis) også komme en masse fejlmeddelelser, men C2C fortæller pænt hvilket linienummer der gav anledning til fejlen og hvilken fejltype det var. Når alle syntaksfejl er luget ud vil C2C generere en fil indeholdende MPASM−assemblerkode, der skal assembleres med Microchips assembler. Det gøres ved kommandoen:

```
mpasm geiger.asm
```

Ved assemblering med MPASM skal assembleren have at vide hvilken processor den skal generere kode til. Det kan gøres direkte i programmet, men da menuerne er temmelig besværligt opbygget, gøres det nemmest ved før MPASM startes at indsætte assemblerdirektivet LIST øverst i filen. Eksempel:

```
; This file was generated by C2C−pro compiler version 2.23 beta

     include "p16c84.inc"
     title "Variables ****************************************"
     __int_save_cont_W              equ 0x0c
     osv...
```

Ændres til:

```
; This file was generated by C2C−pro compiler version 2.23 beta

     list      p=16f84
     include "p16c84.inc"
     title "Variables ****************************************"
     __int_save_cont_W              equ 0x0c
```

Dermed slipper man for en hel del bladren i MPASM's liste over processorer. Include−direktivet henter en hel del fastlagte variabelnavne, og her er 16C84 er lige så godt som 16F84, da de er kodekompatible. Der er ingen speciel fil for F−versionen. MPASM kan også godt finde på at give fejlmeddelelser, men hvis koden er fra C2C får man oftest følgende besked eller en der ligner til forveksling:

```
Checking C:\PROJEKT\GEIGER.ASM for symbols...
Assembling...
GEIGER.ASM   425
Building files...
```

```
Errors          :     0
Warnings        :     0 reported,     0 suppressed
Messages        :     5 reported,     0 suppressed
Lines Assembled :   558
```

Når MPASM har tygget sig igennem koden står man med en færdig .hex fil, i dette tilfælde geiger.hex, der skal lægges ned i PIC'en. Dette beskrives i afsnittet programmering. .hex filen kan også bruges til at simulere det færdige kredsløb med ved hjælp af programmet MPSIM, der kan fås hos MicroChip[20] uden beregning.

Tips

C2C kræver en 386 eller bedre, da det er 32−bit program.

Der findes et program på disketten, C2G også af Pavel Baranov, der er en starthjælp til at få sat registrene rigtigt til det brug og det antal features man har brug for. Det er et windowsprogram, der i en række dialogbokse spørger en om hvilke features man vil bruge og derefter udfører arbejdet med at slå bitmønstre op i databladet for en. Hvis man fx skal bruge Watchdogtimeren med en prescaler på en faktor 64, gerne vil have interrupts ved overløb af tælleren og ved skift på port B og så videre, er det bare at svare på spørgsmålene, så kommer de rigtige registerværdier ud i den anden ende, endda skrevet i færdig MPASM eller C2C−kode.

Programmering i assembler med MPASM/SPASM

PIC16F84 er som nævnt en RISC−processor med sit eget assemblersprog, så dens assemblersprog ligner ikke noget andet. En komplet oversigt over sprogets mnemonics kan findes i bilag sammen med en oversigt over Parallax' SPASM, der har 8051−agtige mnemonics. Jeg vil henvise til dette for mere information.

Dette eksempel er taget fra Parallax' gode samling af application notes[21] og implementerer en modtager til seriel data fra en RS232−forbindelse. Den komplette tekst og diagrammer kan findes som eksempel 2 i deres samling. Programmet er skrevet i 8051−assembler og skal derfor assembleres med SPASM.

```
; PROGRAM: RCV232.SRC
; Taken from Parallax PIC Application Note: Receiving RS−232 Serial Data
; July 15, 1993

; This program receives a byte of serial data and displays it on eight LEDs
; connected to port rb. The receiving baud rate is determined by the value of
; the constant bit_K and the clock speed of the PIC. See the table in the
; application note (above) for values of bit_K. For example, with the clock
; running at 4 MHz and a desired receiving rate of 4800 baud, make bit_K 50.

bit_K     =    24        ;Change this value for desired baudrate
half_bit  =    bit_K/2   ;as shown in table.
serial_in =    ra.2
data_out  =    rb

; Variable storage above special−purpose registers.

          org  8

delay_cntr ds   1         ;Counter for serial delay routines
bit_cntr  ds   1          ;Number of received bits
```

20 http://www.microchip.com. MPSIM ligger også på den vedlagte cdrom.

21 http://www.parallaxinc.com. Samlingen er vedlagt som picapps.pdf og som bilag. Parallax laver små controllere med andres processorer, bl.a. til undervisningsbrug. De har bl.a. brugt PIC til at lave BasicStamp, en lille Basic−programmerbar controller bygget over en PIC15C56 og en lille seriel EEPROM og med et elegant pc−interface, som de lever af.

```
rcv_byte   ds   1          ;The received byte

; Org 0 sets ROM origin to beginning for program.

           org  0

; Remember to change device info if programming a different PIC. Do not use
RC
; devices. Their clock speed is not sufficiently accurate or stable for
serial
; communication.

           device    pic16c54,xt_osc,wdt_off,protect_off
           reset     begin

; Set up I/O ports.

begin      mov  !ra, #00000100b ;Use ra.2 for serial input.
           mov  !rb, #0         ;Output to LEDs.

:start_bit snb  serial_in  ;Detect start bit. Change to
                           ;sb serial_in if using 22k resistor
                           ;input.

           jmp  :start_bit ;No start bit yet? Keep watching.
           call start_delay    ;Wait one-half bit time to the middle
                           ;of the start bit.

           jb   Serial_in,:start_bit

                           ;If the start bit is still good,
                           ;continue. Otherwise, resume waiting.
                           ;Change to jnb Serial_in, :start_bit
                           ;if using 22k resistor input.

           mov  bit_cntr, #8   ;Set the counter to receive 8 data bits
           clr  rcv_byte  ;Clear the receive byte to get ready
                           ;for new data.

:receive   call bit_delay ;Wait one bit time.

           movb c,Serial_in    ;Put the data bit into carry.
                           ;Change to movb c,/Serial_in if using
                           ;22k resistor input.

           rr   rcv_byte  ;Rotate the carry bit into the receive
                           ;byte.

           djnz bit_cntr,:receive
                           ;Not eight bits yet? Get next bit.

           call bit_delay ;Wait for stop bit.

           mov  data_out, rcv_byte
                           ;Display data on LEDs.

           goto begin:start_bit ;Receive next byte.

; This delay loop takes four instruction cycles per loop, plus eight
; instruction cycles for other operations (call, mov, the final djnz, and
ret).
; These extra cycles become significant at higher baud rates. The values for
; bit_K in the table take the time required for additional instructions into
; account.

bit_delay  mov  delay_cntr,#bit_K
:loop      nop
           djnz delay_cntr, :loop
           ret

; This delay loop is identical to bit_delay above, but provides half the
delay
; time.

start_delay     mov  delay_cntr,#half_bit
:loop      nop
           djnz delay_cntr, :loop
           ret
```

22k resistor input refererer til muligheden for at benytte en modstand til

strømbegrænsning og beskyttelsesdioderne, der er indbygget i PIC'en, til spændingskonvertering i stedet for en MAX232 eller anden konverter mellem RS232 og TTL−levels. Læg mærke til at seriel kommunikation er indbygget i C2C som kun en instruktion – dette eksempel ville i C kun fylde en 3−4 liniers reel kode og omkring 10 liniers opsætning.

Som det ses er det meget mere omstændeligt at skrive i assembler, så i de fleste applikationer vil C være at foretrække. Læg mærke til at C2C kan lave inline assembler, så tidskritiske rutiner om nødvendigt kan skrives direkte i et større program, dog kun i MPASM−dialekt.

Tips

SPASM producerer sin objektkode i en fil med efternavnet .obj i stedet for .hex. Indholdet er dog lige godt – og i samme format, så brændersoftwaren kan ikke se forskel.

Det kan ved brug af MPASM godt betale sig at indsætte assemblerdirektivet LIST P=16F84 som beskrevet i afsnittet om C−programmering. SPASM bruger linien `device pic16c84, xt_osc, wdt_off, protect_off`, til samme formål samt til at sætte processoren op. I dette tilfælde skal PIC'en benytte en krystaloscillator, men slå både watchdogtimer og kodebeskyttelse fra. SPASM 4.7, som ligger vedlagt, den version kender ikke PIC16F84, men 16C84 er som sagt kodekompatibel.

Programmering med RS−232 og PIP−02

Uanset hvordan man får fremstillet sin objektkode skal den jo lægges ned i en PIC på et tidspunkt, og måden man gør det på er en af processorens helt store styrker. Den kan både slettes og programmeres elektrisk og uden de høje spændinger som skulle bruges i gamle dage, hvilket gør UV−lys og separate brændere overflødige. PIC'en sættes i programmeringsmode ved at få en høj spænding (ca. 12V) på sit resetben. Derved sætter den alle porte i tristate−tilstand og kan kommunikere med fx en pc via RB6 og RB7, de to mest betydende bit i port B, med en seriel protokol.

Programmeringssystemet styrer clocken, SCL for synchronous clock og kan læse og skrive til PIC'en via det andet ben, SDA, serial synchronous data. Disse to ben (og ground) skal forbindes til en pc eller et andet system der kan programmere chippen. Begge ben kører med standard TTL−levels og kan derfor drives af næsten hvad som helst. Der kan selvfølgelig købes færdige Programmere[22] fra Microchip og 3.parts leverandører, men når protokollen er så simpel som den er, er der ingen grund til det.

Hardware

Masser af mennesker har lavet den smule hardware og software der skal til, som oftest til parallelporten. Deres Programmere kan findes i dusinvis af versioner på nettet. Jeg vil foreslå at benytte et lidt utraditionelt design, LudiPipo[23], der benytter serielporten både til at forsyne PIC'en med 5V, 12V og programmeringssignaler. LudiPipo er designet af Ludwig Catta og har tre meget tiltrækkende egenskaber. Den behøver ingen

22 En dims der kan programmere chippen – der findes vist desværre ikke et godt dansk ord. Brænder er ikke god mere.
23 Al information i download−pakken er vedlagt som bilag.

ekstern 12V strømforsyning, det er meget simpelt og dermed meget billigt. Derudover følges det med udmærket software til selve programmeringen.

I originaludgaven bruges der 4 modstande, en almindelig− og en zenerdiode samt en lille elektrolytkondensator. Ideen er at give PIC'en de 12V fra serielporten direkte, generere de 5V almindelig forsyningsspænding med en formodstand, en zenerdiode og en ensretter og bruge de tre resterende modstande til at begrænse strømmen i PIC'ens indbyggede beskyttelsesdioder, så serielportens normalt 12V begrænses til et sikkert niveau. Dermed kan både kommunikation og strømforsyning komme via en RS232−serielport fra den tilsluttede pc. Var det lidt uklart, så gør diagrammet det nok betydelig lettere at se for sig: (kommer snart, kan ses i bilag under Ludipipo)

I og med at Programmeren er så billig kan den uden videre bygges ind i det instrument eller apparat som PIC'en skal sidde i. Da instrumentet typisk har sin egen strømforsyning kræver det kun tre modstande og et SUBD 9F stik[24] at bygge programmeringsmuligheden ind i apparatet og slippe for separate Programmere. Det gør udviklingen nemmere, da chippen ikke skal flyttes så meget rundt. Kredsløbsdiagrammet for en sådan onboard Programmer kan både ses i bilaget *LudiPipo* og i de to vedlagte eksempler *Et basalt kredsløb* og *En indbygget PIC.*

Brinck Elektronik[25] laver også en Programmer, deres byggesæt Br870, der er udmærket. Det er bare ganske dyrt at købe de nævnte 7 komponenter, et stik og en sokkel for 155 kroner, når man kan lave det selv for en tyver og en time

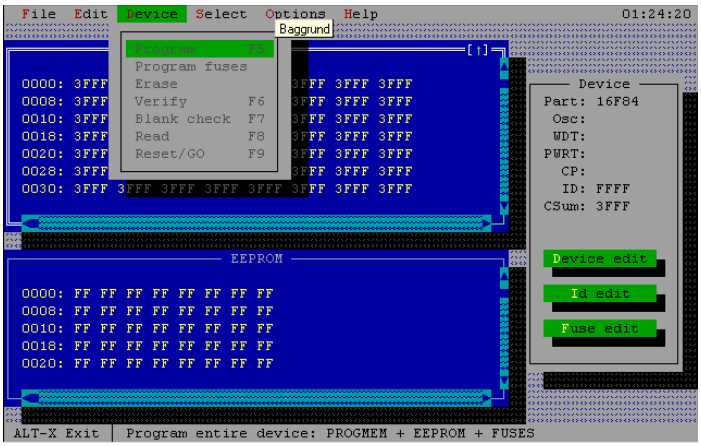
Uanset hvordan man vælger at bygge LudiPipo−kredsløbet kan programmet PIP−02 bruges til at lægge den færdige objektfil ind i PIC'ens hukommelse. Der følger en hel del drivere med til programmet. Den der passer til seriel kommunikation hedder COM84 og skal kaldes med den serielport som Programmeren sidder på som argument før PIP02 startes, fx med

```
>COM84 COM2
```

for en Programmer koblet til serielport 2. Herefter kan PIP02 startes. Programmet er DOS−menubaseret og er generelt velopdragent og brugervenligt. Hent .hex filen ind, sæt de tre option−bits i panelet til højre og vælg Program i Device−menuen. Det kan anbefales at teste om programmet er lagt ordentligt i PIC'en ved at bruge Verify umiddelbart efter.

24 Et standard 9−pin serielt hunstik.
25 http://www.brinck.com.

PIP02 ser ud som følger:



Generelt skulle der ikke være nogen problemer. Læs dog nedenstående tips – der er visse mærkværdigheder ved brug af LudiPipo, som det godt kan betale sig at lægge mærke til.

Tips

LudiPipo−princippet kan også udmærket benyttes i et større system selv om RB6 og 7 bliver brugt til andre ting. Apparatet skal køre på sin egen spændingsforsyning medmindre strømforbruget er meget lille, så serielporten ikke skal trække hele opstillingen. Spændingerne på benene vil på grund af de interne dioder ikke overstige ca. Vcc + 0,6V, hvilket ikke ødelægger noget. Der må ikke trækkes strøm fra portene mens der programmeres, da det logiske niveau så meget nemt kan blive for lavt. Med andre ord er CMOS−kredse OK, men lysdioder og transistorer må afkobles med en jumper eller lignende. Man må også være opmærksom på at de enheder der normalt er koblet til benene vil opfatte signalerne som logiske niveauer, hvilket sammen med at de resterende ben er tri−statede vil kunne give uforudsigelige resultater hvis der ikke er sørget for veldefinerede (og inaktive) signaler på enable−ben og lignende.

Når PIC'en forsynes fra egen strømforsyning er den meget kræsen med spændingsniveauerne. Man kan – især når pull−up modstanden på MCLR/Reset−benet er lille – komme ud for at PIC'en nægter at lade sig programmere. Problemet kan løses ved at sænke seriemodstanden til Vpp[26], gøre pull−up modstanden større eller ved at sænke PIC'ens Vcc[27] ved indkoble en diode i lederetningen før PIC'ens Vcc−ben, da chippen afgør hvorvidt den skal programmeres eller ej ved at sammenligne Vpp med Vcc. En lavere Vpp, som fx ved en svag serielport, vil give en langsommere programmering, men virker, selv om det helt klart er et design der ikke er efter bogen og dermed ikke kan garanteres at virke for alle chips.

Vær opmærksom på en enkelt mærkværdighed hvis du går ud fra et program designet til PIC16C84, men bruger 'F84 – det eneste der er er anderledes i F84−versionen i forhold til C84−programmer er at dens bit til watchdogtimeren er inverteret! Det kan give sig udslag i meget underlige fejl, da processoren hele tiden vil blive resat. Hvis man benytter PIP02 og ønsker at bruge watchdogtimeren skal den altså slås fra i

26 Programmeringsspændingen, typisk 12V
27 Forsyningsspændingen, typisk 5V

programmet og vice versa.

Basiskredsløb for PIC16F84

I og med at PIC'en har indbygget RAM og ROM bliver minimumskredsløbet meget lille. Der skal faktisk kun en clock og en spændingsforsyning til. Selv clockkredsløbet kan minimeres, da det er muligt at clocke PIC'en på ikke mindre end 5 forskellige måder: Ekstern clock, krystal− eller keramisk oscillator, low power krystal, high speed (overtone) krystal og endelig RC−clock, hvor PIC'en udnytter et RC−led til at generere sine egne clockpulser. Hvorvidt man skal vælge RC− eller krystaller til clockgeneration kommer an på den ønskede nøjagtighed og hastighed. RC er upræcis og ustabil[28], men meget billig. Krystallerne er meget mere præcise, og her afhænger valget udelukkende af den ønskede hastighed (og dermed af krystaltypen). Jeg vil anbefale at benytte krystaller, medmindre hastigheden er fuldstændig ukritisk.

Et eksempel på et basalt system med Programmer er vist i bilaget *Et basalt kredsløb*.

Tips

Et par 74HC− eller AC573'ere eller lignende kan hurtigt give en PIC et væld af udgange. Tilsvarende kan 244'ere klare at multiplexe port B til op til 32 (port A er på 5 bit) bidirektionelle 8 bit porte. Et eksempel kunne være at bruge 3 bit fra port A til at drive en 74HC138 3−til−8 aktivt lav−dekoder/demultiplexer med udgangene brugt som chip select på hver sin periferienhed. Linie 0 må reserveres, da PIC'en skal have tid til at konfigurere sit data direction register for port B for hver læsning, men der er stadig syv 8 bit porte med valgfri dataretning som resultat – og der er oven i købet 2 ben til overs på port A til seriel kommunikation.

28 Chipnøjagtighed –28% med store, men uspecificerede variationer med temperatur og fugtighed. Max. 4 MHz

Konklusion

PIC16F84 er en minimal mikroprocessor, der er velegnet til små styrings− eller interfacingopgaver. Dens lille pris og forholdsvis enkle programmering og brug gør den værd at overveje til små projekter.

Her er forsøgt en kort oversigt over processorens egenskaber og en form for brugervejledning til at hjælpe folk, der i forvejen kender noget til mikroprocessorer og elektronik, i gang med PIC'en. Desuden er vedlagt nogle testede eksempler på hvad PIC'en kan bruges til samt software til at programmere den.

Bilag

Links

Det bedste og oftest opdaterede sted at finde links er nu på nettet selv.

Producenter

http://www.microchip.com
Producenten af PIC−familien.

http://www.parallaxinc.com
Parallax laver bl.a. BasicStamp, der består af en PIC med basicfortolker og en eeprom. De står også bag SPASM, der er et assemblerværktøj til PIC'en

Software

http://www.geocities.com/SiliconValley/Network/3656/c2c/c.html
Hjemmeside for C2C
http://
Download−sted for PIP−02

Projekter

http://www.student.dtu.dk/~c973357
Madam Skrald I – en robot bygget til Robocup 99 baseret på en PIC16F84

http://www−ccs.cs.umass.edu/~shri/iPic.html
En internetserver på en enkelt PIC

Metalinks/samlinger

http://www.labyrinth.net.au/~donmck/dtait/piclinks.html
David Taits Piclinks

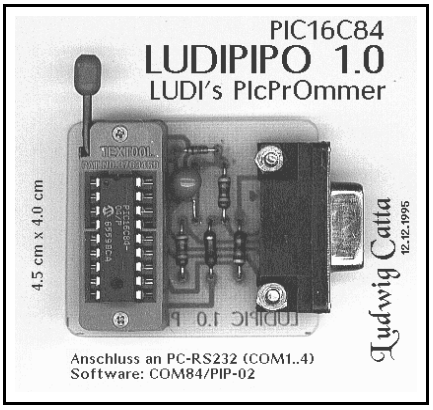
http://www.hut.fi/Misc/Electronics/
Det finske tekniske universitets elektroniklinks

Et basalt kredsløb

Helt nede på jorden. Kan downloades som gif−filer fra www.student.dtu.dk/~c973357/madam1.html

En indbygget PIC

Er endnu ikke tilgængelige på World Wide Wait, men vil nok komme på et tidspunkt et eller andet sted på www.student.dtu.dk/~c973357




```
****************************************
  LUDIPIPO 1.0 Prommer fuer PIC16C84
  ------  LUDI's PIcPrOmmer  ------
  fuer die serielle Schnittstelle
  - direkter Anschluss, 4.5cm x 4.0 cm
  Ludwig Catta :-)        12.12.1995
****************************************
```

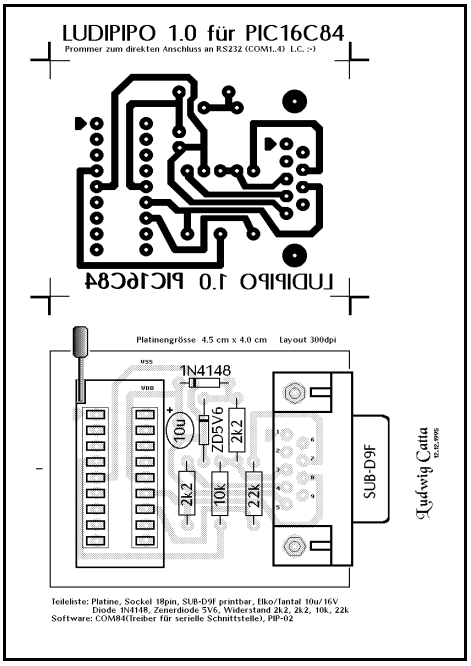
Oinkidoinki, Freunde!

Naja, mal ne kleine Bastelarbeit: ein Miniprommer, der ausgezeichnet funzt - und (fast) nix kostet. Der LUDIPIPO loescht/programmiert PIC16C84 innert wenicher Sekunden :-). Gut geeignet fuer Schmartkards zum SAT-TV decrypten.

Das File LUDIPIPO.ZIP enthaelt:

```
LUDIPIPO PCX        54.966 12.12.95    1:00  Layout, Bestueckung
LUDIPIPF PCX        78.412 12.12.95    1:00  Foto
FILE_ID  DIZ           201 17.12.95    3:25
COM84    EXE         4.571 02.08.95   10:33  serieller Treiber
COM84    TXT           742 05.12.95   19:12
PIP-02   EXE        88.385 22.10.95   23:31  Proggy zum PIC-Proggymieren
PIP-02   HLP        31.753 22.10.95   23:31
PIP-02   TXT           465 05.12.95   19:18
LUDIPIPO TXT         3.848 17.12.95    3:01  dieser Text
```

Die Idee stammt von Erik Hermann ... ooooops, wo iss denn bloss der Schaltplan ... achja .. hier iss er ja:




```
     TxD ---*-------------------------------
     (3)    I                               I
            ---                             ---
            I I                             I I
            I I 2k2                         I I 10k
            ---                             ---
             I       I\I                     I
             *-------I-I-----*---------      I
             I       I/I     I         I     I
            ----\            I +       I     I
             /\  \           ---     14 I    I 4
            /  \    5V6      ---    ---------------
            ----          10u  I     I  Vdd Vpp    I
              I                  I     I             I
              I                  I  5  I             I
     GND ---*----------------*----I Vss             I
     (5)           22k             I               I
                   -----       12  I               I
     RTS ----------I   I-----------IRB6 (clock)    I
     (7)           -----           I               I
                   -----       13  I               I
     DTR ----------I   I-----*-----IRB7 (data)     I
     (4)           -----     I     I               I
                    2k2      I     I  PIC 16C84    I
     CTS ------------------I     I-----------------I
     (8)
```

In Klammern die Pins fuer SUB-D9F. Beachte: keine zusaetzliche Versorgungs-spannung, wie sonst bei den -zigmilliarden PIC-Prommers notwendich. Auch kein doppelter Boden, kein Werbeangebot, kein Nepp von Echodepp ....

Einfach schenial.

Die Widerstandswerte sind nich kritisch, ich hab statt 2k2,2k2,10k,22k die Werte 1k4,1k4,10k,16k genommen. Im Originalplan iss ein Kondensator mit 100u :-[, den hab ich durch einen 10u/16V Tantal ersetzt. Iss kleiner und lag gerade rum. Probs sollte es keine geben: nich viel zu loeten, nich viel zu testen. Falls du beim Loeten ne Tasse Cappuccino schluerfst und Schokokuchen verdrueckt, pass auf, dass du nich aus Versehen ne Diode verschluckst. Sind recht klein die Dinger. Kosten DM 0.02, lohnt trotzdem nich, gleich 100.000 Stueck davon zu kaufen! Und lass dich nich von Maedels ablenken, sonst loetest du sie verkehrtrum ein! Die Dioden, mein ich.

Achja: als Sockel hab ich nen Textool aus dem Ramschladen fuer zwomark-fuffzich drauf. Andere Nullkraftsockel haben u.U. keinen 3/10"-Abstand,

sondern die Beine weiter auseinander ;−] gegebenenfalls also Sockelpins links weiter rausziehen.

Nochwas: PIP−02 proggymiert auch serielle EEPROMs, na, die im 8−pin−Gehaeuse. Man kann also noch nen 8−pin−Sockel dazuknallen − Platine wird halt nen Centimeter laenger. Ich war zu faul dazu, vielleicht beim nexten Mal. Ausserdem komm ich dann nich mehr ins Buch der Rekorde mit dem weltkleinsten Prommer.

Nochmal nochwas: wenns nich funzt − ey, gibts doch gar nich − dann kontrollieren. Als ich den LUDIPIPO ansteckte, kamen bloss Nullen (naja, eben wie in der Politik...). Verdammt, Ludi, haste Mist gebaut, dachte ich. Nach dem Kontrollieren fand sich der Fehler (auch kleine Meister machen grosse Fehler ...): ich hatte den SUB−D9F mit zwo 3mm−Schrauben sauber festgeschraubt − aber keinen der 9 Pins angeloetet :−(. Nuja, wenns ein Verhueterli zerreisst, iss es schlimmer.

Judihui kannste dann bloss sagen .. oder judibummmmmms

Ludwig

Noch ein Vorteil:
Geht beim LUDIPIPO etwas schief, musste deswegen nich gleich heiraten :−|

D I R E C T I V E S

	Directive		Description
	INCLUDE	'filename'	Include file 'filename' in assembly (may precede device directive)
	DEVICE	type,setting,setting...	Establish device type and settings (must precede other directives/instructions)
	ID	word <or> 'text'	Set id to word ('5x) or 4-character text string ('xx)
label	=	value	Equate label to value
label	EQU	value	Equate label to value
	ORG	address	Set origin to address
	DS	locations	Define space: origin=origin+locations
	RESET	address	Assemble 'jmp address' at last location for reset ('5x only)
	EEORG	address	Set data eeprom origin to address ('84 only)
	EEDATA	data,data...	Preload data eeprom with data ('84 only)
	END		End assembly (not needed)

P A R A L L A X I N S T R U C T I O N S E T

Parallax Instruction	Words	Flags	W used	Description	Microchip Equivalent
				========== byte-oriented operations ==========	
MOV	W,#lit	1	-	-	Move literal into W	MOVLW lit
MOV	W,fr	1	Z	-	Move fr into W	MOVF fr,0
MOV	fr,W	1	-	-	Move W into fr	MOVWF fr
MOV	fr,#lit	2	-	W	Move literal into fr	MOVLW lit; MOVWF fr
MOV	fr1,fr2	2	Z	W	Move fr2 into fr1	MOVF fr2,0; MOVWF fr1
ADD	W,#lit	1	C,DC,Z	-	Add literal into W	ADDLW lit ('xx only)
ADD	W,fr	1	C,DC,Z	-	Add fr into W	ADDWF fr,0

ADD	fr,W		1	C,DC,Z	–	Add W into fr	ADDWF fr,1
ADD	fr,#lit		2	C,DC,Z	W	Add literal into fr	MOVLW lit; ADDWF fr,1
ADD	fr1,fr2		2	C,DC,Z	W	Add fr2 into fr1	MOVF fr2,0; ADDWF fr1,1
MOV	W,#lit–W	1	C,DC,Z	–	Move literal–W into W	SUBLW lit (’xx only)	
MOV	W,fr–W		1	C,DC,Z	–	Move fr–W into W	SUBWF fr,0
SUB	fr,W		1	C,DC,Z	–	Subtract W from fr	SUBWF fr,1
SUB	fr,#lit		2	C,DC,Z	W	Subtract literal from fr	MOVLW lit; SUBWF fr,1
SUB	fr1,fr2		2	C,DC,Z	W	Subtract fr2 from fr1	MOVF fr2,0; SUBWF fr1,1
AND	W,#lit		1	Z	–	AND literal into W	ANDLW lit
AND	W,fr		1	Z	–	AND fr into W	ANDWF fr,0
AND	fr,W		1	Z	–	AND W into fr	ANDWF fr,1
AND	fr,#lit		2	Z	W	AND literal into fr	MOVLW lit; ANDWF fr,1
AND	fr1,fr2		2	Z	W	AND fr2 into fr1	MOVF fr2,0; ANDWF fr,1
OR	W,#lit		1	Z	–	OR literal into W	IORLW lit
OR	W,fr		1	Z	–	OR fr into W	IORWF fr,0
OR	fr,W		1	Z	–	OR W into fr	IORWF fr,1
OR	fr,#lit		2	Z	W	OR literal into fr	MOVLW lit; IORWF fr,1
OR	fr1,fr2		2	Z	W	OR fr2 into fr1	MOVF fr2,0; IORWF fr1,1
XOR	W,#lit		1	Z	–	XOR literal into W	XORLW lit
XOR	W,fr		1	Z	–	XOR fr into W	XORWF fr,0
XOR	fr,W		1	Z	–	XOR W into fr	XORWF fr,1
XOR	fr,#lit		2	Z	W	XOR literal into fr	MOVLW lit; XORWF fr,1
XOR	fr1,fr2		2	Z	W	XOR fr2 into fr1	MOVF fr2,0; XORWF fr1,1
CLR	W		1	Z	–	Clear W	CLRW
CLR	fr		1	Z	–	Clear fr	CLRF fr
MOV	W,++fr		1	Z	–	Move fr+1 into W	INCF fr,0
INC	fr		1	Z	–	Increment fr	INCF fr,1
MOV	W,--fr		1	Z	–	Move fr–1 into W	DECF fr,0
DEC	fr		1	Z	–	Decrement fr	DECF fr,1
MOV	W,>>fr		1	C	–	Move right–rotated fr into W	RRF fr,0
RR	fr		1	C	–	Rotate right fr	RRF fr,1
MOV	W,<<fr		1	C	–	Move left–rotated fr into WRLF fr,0	
RL	fr		1	C	–	Rotate left fr	RLF fr,1
MOV	W,<>fr		1	–	–	Move nibble–swapped fr into W	SWAPF fr,0
SWAP	fr		1	–	–	Swap nibbles in fr	SWAPF fr,1
NOT	W		1	Z	–	Perform not on W	XORLW 0FFh
MOV	W,/fr		1	Z	–	Move not’d fr into W	COMF fr,0
NOT	fr		1	Z	–	Perform not on fr	COMF fr,1
TEST	W		1	Z	–	Test W for zero	IORLW 0
TEST	fr		1	Z	–	Test fr for zero	MOVF fr,1

```
                                                    ========================
                                                    bit-oriented operations
                                                    ========================

CLRB        bit                     1       -       -       Clear bit                                   BCF bit
CLC                                 1       C       -       Clear carry                                 BCF 3,0
CLZ                                 1       Z       -       Clear zero                                  BCF 3,2

SETB        bit                     1       -       -       Set bit                                                 BSF bit
STC                                 1       C       -       Set carry                                   BSF 3,0
STZ                                 1       Z       -       Set zero                                    BSF 3,2

ADDB        fr,bit                  2       Z       -       Add bit into fr                             BTFSC bit; INCF fr,1
ADDB        fr,/bit                 2       Z       -       Add not bit into fr                         BTFSS bit; INCF fr,1
SUBB        fr,bit                  2       Z       -       Subtract bit from fr                        BTFSC bit; DECF fr,1
SUBB        fr,/bit                 2       Z       -       Subtract not bit from fr    BTFSS bit; DECF fr,1

MOVB        bit1,bit2   4           -       -       Move bit2 into bit1                         BTFSS bit2; BCF bit1; BTFSC bit2; BSF bit1
MOVB        bit1,/bit2  4           -       -       Move not bit2 into bit1     BTFSC bit2; BCF bit1; BTFSS bit2; BSF bit1

                                                    ============================
                                                    inc/dec-conditional branches
                                                    ============================

MOVSZ       W,++fr                  1       -       -       Move fr+1 into W, skip if zero          INCFSZ fr,0
INCSZ       fr                      1       -       -       Increment fr, skip if zero INCFSZ fr,1
IJNZ        fr,addr                 2       -       -       Increment fr, jump if not zero          INCFSZ fr,1; GOTO addr

MOVSZ       W,--fr                  1       -       -       Move fr-1 into W, skip if zero          DECFSZ fr,0
DECSZ       fr                      1       -       -       Decrement fr, skip if zero DECFSZ fr,1
DJNZ        fr,addr                 2       -       -       Decrement fr, jump if not zero          DECFSZ fr,1; GOTO addr

                                                    ============================
                                                    compare-conditional branches
                                                    ============================

CSA         fr,#lit                 3       C,DC,Z  W       Compare, skip if above                  MOVLW /lit; ADDWF fr,0; BTFSS
3,0
CSA         fr1,fr2                 3       C,DC,Z  W       Compare, skip if above                  MOVF fr1,0; SUBWF fr2,0;
BTFSC 3,0
CSAE        fr,#lit                 3       C,DC,Z  W       Compare, skip if above or equal         MOVLW lit; SUBWF fr,0; BTFSS
3,0
CSAE        fr1,fr2                 3       C,DC,Z  W       Compare, skip if above or equal         MOVF fr2,0; SUBWF fr1,0;
BTFSS 3,0
CSB         fr,#lit                 3       C,DC,Z  W       Compare, skip if below                  MOVLW lit; SUBWF fr,0; BTFSC
3,0
CSB         fr1,fr2                 3       C,DC,Z  W       Compare, skip if below                  MOVF fr2,0; SUBWF fr1,0;
BTFSC 3,0
CSBE        fr,#lit                 3       C,DC,Z  W       Compare, skip if below or equal         MOVLW /lit; ADDWF fr,0; BTFSC
3,0
CSBE        fr1,fr2                 3       C,DC,Z  W       Compare, skip if below or equal         MOVF fr1,0; SUBWF fr2,0;
BTFSS 3,0
CSE         fr,#lit                 3       C,DC,Z  W       Compare, skip if equal                  MOVLW lit; SUBWF fr,0; BTFSS
```

```
3,2
CSE             fr1,fr2                 3           C,DC,Z      W           Compare, skip if equal                  MOVF fr2,0; SUBWF fr1,0;
BTFSS 3,2
CSNE            fr,#lit                 3           C,DC,Z      W           Compare, skip if not equal MOVLW lit; SUBWF fr,0; BTFSC 3,2
CSNE            fr1,fr2                 3           C,DC,Z      W           Compare, skip if not equal MOVF fr2,0; SUBWF fr1,0; BTFSC 3,2

CJA             fr,#lit,addr  4         C,DC,Z      W           Compare, jump if above                  MOVLW /lit; ADDWF fr,0; BTFSC 3,0; GOTO
addr
CJA             fr1,fr2,addr  4         C,DC,Z      W           Compare, jump if above                  MOVF fr1,0; SUBWF fr2,0; BTFSS 3,0; GOTO
addr
CJAE            fr,#lit,addr  4         C,DC,Z      W           Compare, jump if above or equal         MOVLW lit; SUBWF fr,0; BTFSC 3,0; GOTO addr
CJAE            fr1,fr2,addr  4         C,DC,Z      W           Compare, jump if above or equal         MOVF fr2,0; SUBWF fr1,0; BTFSC 3,0; GOTO
addr
CJB             fr,#lit,addr  4         C,DC,Z      W           Compare, jump if below                  MOVLW lit; SUBWF fr,0; BTFSS 3,0; GOTO addr
CJB             fr1,fr2,addr  4         C,DC,Z      W           Compare, jump if below                  MOVF fr2,0; SUBWF fr1,0; BTFSS 3,0; GOTO
addr
CJBE            fr,#lit,addr  4         C,DC,Z      W           Compare, jump if below or equal         MOVLW /lit; ADDWF fr,0; BTFSS 3,0; GOTO
addr
CJBE            fr1,fr2,addr  4         C,DC,Z      W           Compare, jump if below or equal         MOVF fr1,0; SUBWF fr2,0; BTFSC 3,0; GOTO
addr
CJE             fr,#lit,addr  4         C,DC,Z      W           Compare, jump if equal                  MOVLW lit; SUBWF fr,0; BTFSC 3,2; GOTO addr
CJE             fr1,fr2,addr  4         C,DC,Z      W           Compare, jump if equal                  MOVF fr2,0; SUBWF fr1,0; BTFSC 3,2; GOTO
addr
CJNE            fr,#lit,addr  4         C,DC,Z      W           Compare, jump if not equal MOVLW lit; SUBWF fr,0; BTFSS 3,2; GOTO addr
CJNE            fr1,fr2,addr  4         C,DC,Z      W           Compare, jump if not equal MOVF fr2,0; SUBWF fr1,0; BTFSS 3,2; GOTO addr

                                                                ========================
                                                                bit-conditional branches
                                                                ========================

SB              bit                     1           -           -           Skip if bit                             BTFSS bit
SC                                      1           -           -           Skip if carry                           BTFSS 3,0
SZ                                      1           -           -           Skip if zero                            BTFSS 3,2

SNB             bit                     1           -           -           Skip if not bit                         BTFSC bit
SNC                                     1           -           -           Skip if not carry                       BTFSC 3,0
SNZ                                     1           -           -           Skip if not zero                        BTFSC 3,2

JB              bit,addr      2         -           -           Jump to address if bit                  BTFSC bit; GOTO addr
JC              addr                    2           -           -           Jump to address if carry    BTFSC 3,0; GOTO addr
JZ              addr                    2           -           -           Jump to address if zero     BTFSC 3,2; GOTO addr

JNB             bit,addr      2         -           -           Jump to address if not bit BTFSS bit; GOTO addr
JNC             addr                    2           -           -           Jump to address if not carry            BTFSS 3,0; GOTO addr
JNZ             addr                    2           -           -           Jump to address if not zeroBTFSS 3,2; GOTO addr

                                                                ======================
                                                                unconditional branches
                                                                ======================

SKIP                                    1           -           -           Skip next instruction word '5x: BTFSS 4,7 / 'xx: BTFSC 10,7

JMP             addr                    1           -           -           Jump to address                         GOTO addr
```

```
JMP          PC+W                    1           C,DC,Z      –           Add W into PC(L)                        ADDWF 2,1
JMP          W                       1           –           –           Move W into PC(L)                       MOVWF 2

CALL         addr                    1           –           –           Call to address                         CALL addr
RET                                  1           –           –           Return from call                        ’5x: RETLW 0 / ’xx: RETURN
RETW         lit,lit...    1         –           –           Return from call, literal in W          RETLW lit; RETLW lit...

RETI                                 1           –           –           Return from interrupt                   RETFIE (’xx only)

LSET         addr                    0+          –           –           Point to 512/2048-word page BCF/BSF (’5x: 3,5+)/(’xx: 10,3+)
LJMP         addr                    1+          –           –           Point to page and jump to addr          BCF/BSF (’5x: 3,5+)/(’xx:
10,3+); GOTO addr
LCALL        addr                    1+          –           –           Point to page and call to addr          BCF/BSF (’5x: 3,5+)/(’xx:
10,3+); CALL addr

                                     Note: Skips should be followed only by single-word instructions.

                                                                          ==========================
                                                                          i/o and control operations
                                                                          ==========================

MOV          !port,W                 1           –           –           Move W into port’s TRIS    TRIS port (port=5 to 7)
MOV          !port,#lit    2         –           W           Move literal into port’s TRIS          MOVLW lit; TRIS port (port=5 to 7)
MOV          !port,fr      2         Z           W           Move fr into port’s TRIS   MOVF fr,0; TRIS port (port=5 to 7)

MOV          !OPTION,W     1         –           –           Move W into OPTION                     OPTION
MOV          !OPTION,#lit  2         –           W           Move literal into OPTION   MOVLW lit; OPTION
MOV          !OPTION,fr    2         Z           W           Move fr into OPTION                    MOVF fr,0; OPTION

CLR          WDT                     1           TO,PD       –           Clear WDT and prescaler    CLRWDT
SLEEP                                1           TO,PD       –           Clear WDT and enter sleep mode          SLEEP

NOP                                  1           –           –           No operation                            NOP


                                                 M I C R O C H I P   I N S T R U C T I O N   S E T

-------------------------------------------------------------------------------------------------------------------------------
Microchip Instruction      Words        Flags       W used      Description                            Parallax Equivalent
-------------------------------------------------------------------------------------------------------------------------------

                                                                          ======================================
                                     byte-oriented file register operations
                                                                          ======================================

ADDWF        fr,d          *           1           C,DC,Z      –           Add W and fr
                           ADD W,fr / ADD fr,W
ANDWF        fr,d          *           1           Z           –           And W and fr                           AND W,fr / AND fr,W
CLRF         fr                        1           Z           –           Clear fr                               CLR fr
CLRW                                   1           Z           –           Clear W                                          CLR W
COMF         fr,d          *           1           Z           –           Complement fr                          MOV W,/fr / NOT fr
```

DECF	fr,d	*	1	Z	–	Decrement fr		MOV W,--fr / DEC fr
DECFSZ	fr,d	*	1	–	–	Decrement fr, skip if 0	MOVSZ W,--fr / DECSZ fr	
INCF	fr,d	*	1	Z	–	Increment fr		MOV W,++fr / INC fr
INCFSZ	fr,d	*	1	–	–	Increment fr, skip if 0	MOVSZ W,++fr / INCSZ fr	
IORWF	fr,d	*	1	Z	–	Or W and fr		OR W,fr / OR fr,W
MOVF	fr,d	*	1	Z	–	Move fr		MOV W,fr / TEST fr
MOVWF	fr		1	–	–	Move W to fr		MOV fr,W
NOP			1	–	–	No operation		NOP
RLF	fr,d	*	1	–	–	Rotate fr left through carry		MOV W,<<fr / RL fr
RRF	fr,d	*	1	–	–	Rotate fr right through carry		MOV W,>>fr / RR fr
SUBWF	fr,d	*	1	C,DC,Z	–	Subtract W from fr		MOV W,fr-W / SUB fr,W
SWAPF	fr,d	*	1	Z	–	Swap nibbles of fr		MOV W,<>fr / SWAP fr
XORWF	fr,d	*	1	Z	–	Xor W and fr		XOR W,fr / XOR fr,W
						==========		
						bit-oriented file register operations		
						==========		
BCF	fr,b	**	1	–	–	Bit clear fr		CLRB bit
BSF	fr,b	**	1	–	–	Bit set fr		SETB bit
BTFSC	fr,b	**	1	–	–	Bit test fr, skip if clear	SNB bit	
BTFSS	fr,b	**	1	–	–	Bit test fr, skip if set	SB bit	
						==========		
						literal and control operations		
						==========		
ADDLW	lit		1	C,DC,Z	–	Add literal into W		ADD W,#lit ('xx only)
ANDLW	lit		1	Z	–	And literal into W		AND W,#lit
CALL	addr		1	–	–	Call to address		CALL addr
CLRWDT			1	TO,PD	–	Clear WDT and prescaler	CLR WDT	
GOTO	addr		1	–	–	Go to address		JMP addr
IORLW	lit		1	Z	–	Or literal into W		OR W,#lit
MOVLW	lit		1	–	–	Move literal into W		MOV W,#lit
OPTION			1	–	–	Move W into OPTION		MOV !OPTION,W
RETFIE			1	–	–	Return from interrupt		RETI ('xx only)
RETLW	lit		1	–	–	Return with literal in W	RETW lit / RET ('5x)	
RETURN			1	–	–	Return from subroutine		RET ('xx only)
SLEEP			1	TO,PD	–	Clear WDT and enter sleep mode		SLEEP
SUBLW	lit		1	C,DC,Z	–	Subtract W from literal	MOV W,#lit-W ('xx only)	
TRIS	port		1	–	–	Move W into port's TRIS	MOV !port,W (port=5 to 7)	
XORLW	lit		1	Z	–	Xor literal into W		XOR W,#lit

CLRC			1	–	–	Clear carry	CLC
SETC			1	–	–	Set carry	STC
CLRDC			1	–	–	Clear digit carry	CLRB DC
SETDC			1	–	–	Set digit carry	SETB DC
CLRZ			1	–	–	Clear zero	CLZ
SETZ			1	–	–	Set zero	STZ
SKPC			1	–	–	Skip if carry	SC
SKPNC			1	–	–	Skip if not carry	SNC
SKPDC			1	–	–	Skip if digit carry	SB DC
SKPNDC			1	–	–	Skip if not digit carry	SNB DC
SKPZ			1	–	–	Skip if zero	SZ
SKPNZ			1	–	–	Skip if not zero	SNZ
TSTF	fr		1	Z	–	Test fr	TEST fr
MOVFW	fr		1	Z	–	Move fr into W	MOV W,fr
NEGF	fr,d	*	1	Z	–	Negate fr	NOT fr; INC fr
ADDCF	fr,d	*	1	Z	–	Add carry to fr	ADDB fr,C
SUBCF	fr,d	*	1	Z	–	Subtract carry from fr	SUBB fr,C
ADDDCF	fr,d	*	1	Z	–	Add digit carry to fr	ADDB fr,DC
SUBDCF	fr,d	*	1	Z	–	Subtract digit carry from fr	SUBB fr,DC
B	addr		1	–	–	Branch to address	JMP addr
BC	addr		1	–	–	Branch if carry	JC addr
BNC	addr		1	–	–	Branch if not carry	JNC addr
BDC	addr		1	–	–	Branch if digit carry	JB DC,addr
BNDC	addr		1	–	–	Branch if not digit carry	JNB DC,addr
BZ	addr		1	–	–	Branch if zero	JZ addr
BNZ	addr		1	–	–	Branch if not zero	JNZ addr

* alternate operands to fr,d

fr	= fr,1
fr,W	= fr,0

** alternate operand to fr,b

bit	= fr.b

```
------------------------------------------------------------------------------------------------------------------------
Pre-Defined Symbols
------------------------------------------------------------------------------------------------------------------------
******** Dynamic Equates (always reflect current values)
$                           =             Current origin
%                           =             Current data EEPROM origin ('84 only)


******** DEVICE directive equates - establish device type
    Example:  DEVICE        PIC16C71,XT_OSC,WDT_ON,PWRT_OFF,PROTECT_ON


******** PIC16C84 Equates - enabled by DEVICE PIC16C84
                            Specific DEVICE equates

LP_OSC                      =             0FFFCh                    Oscillator (select one)
XT_OSC                      =             0FFFDh
HS_OSC                      =             0FFFEh
RC_OSC                      =             0FFFFh

WDT_OFF                     =             0FFFBh                    Watchdog timer (select one)
WDT_ON                      =             0FFFFh

PWRT_OFF                    =             0FFF7h                    Power-up timer (select one)
PWRT_ON                     =             0FFFFh

PROTECT_OFF                 =             0FFFFh                    Code protection (select one)
PROTECT_ON                  =             0FFEFh

                            Register and bit equates
                                                                    page
INDF                        =             00h                       0/1           Indirect addressing register

TMR0                        =             01h                       0             Timer 0 register

PCL                         =             02h                       0/1           Program counter low-byte register

STATUS                      =             03h                       0/1           Status register
C                           =             STATUS.0      0/1         Carry bit
DC                          =             STATUS.1      0/1         Digit carry bit
Z                           =             STATUS.2      0/1         Zero bit
PD                          =             STATUS.3      0/1         Power-down bit
TO                          =             STATUS.4      0/1         Time-out bit
RP0                         =             STATUS.5      0/1         Register page bit 0
RP1                         =             STATUS.6      0/1         Register page bit 1
IRP                         =             STATUS.7      0/1         Indirect register page bit

FSR                         =             04h                       0/1           File select register

PORTA                       =             05h                       0             RA i/o register
RA                          =             05h                       0             RA i/o register

PORTB                       =             06h                       0             RB i/o register
```

```
RB                      =       06h                     0       RB i/o register

EEDATA                  =       08h                     0       EEPROM data register
EEADR                   =       09h                     0       EEPROM address register

PCLATH                  =       0Ah                     0/1     Program counter high-byte register

INTCON                  =       0Bh                     0/1     Interrupt control register
RBIF                    =       INTCON.0        0/1     RB4-RB7 change interrupt flag bit
INTF                    =       INTCON.1        0/1     RB0/INT interrupt flag bit
T0IF                    =       INTCON.2        0/1     TMR0 overflow interrupt flag bit
RBIE                    =       INTCON.3        0/1     RB4-RB7 change interrupt enable bit
INTE                    =       INTCON.4        0/1     RB0/INT interrupt enable bit
T0IE                    =       INTCON.5        0/1     TMR0 overflow interrupt enable bit
EEIE                    =       INTCON.6        0/1     EEPROM interrupt enable bit
GIE                     =       INTCON.7        0/1     Global interrupt enable bit

OPTION                  =       01h                     1       OPTION register
PS0                     =       OPTION.0        1       Prescaler bit 0
PS1                     =       OPTION.1        1       Prescaler bit 1
PS2                     =       OPTION.2        1       Prescaler bit 2
PSA                     =       OPTION.3        1       Prescaler assignment bit
T0SE                    =       OPTION.4        1       TMR0 source edge bit
T0CS                    =       OPTION.5        1       TMR0 clock select bit
INTEDG                  =       OPTION.6        1       RB0/INT edge select bit
RBPU                    =       OPTION.7        1       RB weak pull-up enable bit

TRISA                   =       05h                     1       RA tristate control register

TRISB                   =       06h                     1       RB tristate control register

EECON1                  =       08h                     1       EEPROM control register 1
RD                      =       EECON1.0        1       EEPROM read control bit
WR                      =       EECON1.1        1       EEPROM write control bit
WREN                    =       EECON1.2        1       EEPROM write enable bit
WRERR                   =       EECON1.3        1       EEPROM write error flag bit
EEIF                    =       EECON1.4        1       EEPROM interrupt flag bit
EECON2                  =       09h                     1       EEPROM control register
```