Mechatronica
Inleiding tot de PIC microcontroller

Kristof Goris
VUB-MECH-R&MM

30 januari 2006

Inhoudsopgave

Hoofdstuk 1

Inleiding

1.1 Wat is Mechatronica?

De term mechatronica is een combinatie van de termen mechanica en elektronica. Mechatronica is een synergistische aanpak bij het integraal en optimaal ontwerpen van een mechanisch systeem en het bijbehorende regelsysteem. Hedendaagse machines en huishoudapparaten zijn aangedreven door motoren, of worden geactueerd door actuatoren die geregeld worden via elektronische circuits. Meestal gebeurt de controle van deze systemen door programmeerbare componenten zoals embedded systems, PLC's (Programmable Logic Controller), microcontrollers of zelfs rechtstreeks via PC's. Door het tegelijkertijd ontwerpen van de mechanische constructie en het bijbehorende regelsysteem kunnen mechanische constructies superieure eigenschappen krijgen en tegelijkertijd goedkoper en meer flexibel worden. Voorbeelden van dergelijke toepassingen zijn GSM's, recente vaatwasmachines, programmeerbare hifi-ketens, digitale uurwerken,... Al deze huishoudelijke toepassingen bevatten microcontrollers waarbij het programma in een onuitwisbaar geheugen is opgeslagen.

1.2 Wat is een Microcontroller?

Een microcontroller (μC) is een single chip computer. Dit wil zeggen dat de CPU (Central Processing Unit), geheugen voor het programma (ROM), geheugen voor data (RAM), databus, Input/Output poorten voor communicatie met randapparatuur en eventueel D/A- en A/D convertors geïntegreerd zijn in één enkele chip. Een microcontroller is via software te programmeren, waardoor het een flexibele component is. De microcontroller maakt het mogelijk om signalen binnen gekregen via sensoren te verwerken, waarna de μC het signaal kan onthouden of verwerken om tot de gewenste actie te komen. Daarin verschilt een μC van een gewone processor, die allerlei externe componenten nodig heeft voor communicatie en geheugenfuncties. Microcontrollers

kunnen algemeen toepasbaar zijn, of gespecialiseerd voor bepaalde functies, bijvoorbeeld Digitale Signaal Processors (DSP's). De toepassingsgebieden zijn zoals reeds vermeld zeer uitgebreid.

1.3 Doelstelling

Het doel van dit practicum is de student aan de hand van een praktisch voorbeeld een duidelijk beeld te geven van wat de mogelijkheden en toepassingsgebieden zijn van microcontrollers. Dit project zal bestaan uit een gedeelte programmeren, een gedeelte elektronica en een gedeelte mechanica/pneumatica en uiteraard de link ertussen. Dit practicum heeft niet tot doel de student tot een volwaardig programmeur om te scholen, wel zullen de basisinstructies bekeken worden en daarna zal de student deze zelfstandig toepassen in een praktisch project. Als uiteindelijk resultaat wordt er van de student verwacht een website te maken die de verschillende aspecten van het uitgevoerde project in detail beschrijft en een mondelinge presentatie van het uitgevoerde project. De voornaamste eigenschappen van een goed ingenieur waartoe dit tracht bij te dragen is inventiviteit en zelfstandig werken.

Hoofdstuk 2

Basisbegrippen

2.1 Bits en Bytes

Een *bit* stelt een binaire waarde *0* of *1* voor. Een *byte* bestaat uit 8 bits welke in één keer gelezen of geschreven kunnen worden. Als we spreken over bit 0 bedoelen we de LSB (Least Significant Bit). Wanneer we werken met een 8 bits microcontroller komt bit 7 overeen met de MSB (Most Significant Bit). Met 8 bits kan men 256 verschillende waarden voorstellen. Het aantal bytes dat geadresseerd kan worden verschilt van μC tot μC. Zo kan bijvoorbeeld de *Motorola 68HC11* 64k bytes (65556 bytes) adresseren, hiervoor zijn 16 bits nodig (ook *word* genoemd). Een adres betekent een plaats in het geheugen met een *naam* of *adres* van 16 bits waarin een waarde van 8 bits kan bewaard worden.

Getallen kunnen o.a. decimaal, binair of hexadecimaal voorgesteld worden. In assemblercode worden voor decimale, binaire en hexadecimale getallen respectievelijk een *d*, een *b* en een *h* gezet en gevolgd door de getalwaarde tussen *'enkele aanhalingstekens'*. Voorbeelden van de verschillende notaties worden weergegeven in Tabel 2.1. Merk op dat de hexadecimale en de binaire voorstelling snel uit het hoofd om te zetten zijn. Per groep van 4 bits komt één hexadecimaal symbool overeen.

Decimaal	Binair	Hexadecimaal
d‘0’	b‘00000000’	h‘00’
d‘1’	b‘00000001’	h‘01’
d‘2’	b‘00000010’	h‘02’
d‘3’	b‘00000011’	h‘03’
d‘4’	b‘00000100’	h‘04’
d‘5’	b‘00000101’	h‘05’
d‘6’	b‘00000110’	h‘06’
d‘7’	b‘00000111’	h‘07’
d‘8’	b‘00001000’	h‘08’
d‘9’	b‘00001001’	h‘09’
d‘10’	b‘00001010’	h‘0A’
d‘11’	b‘00001011’	h‘0B’
d‘12’	b‘00001100’	h‘0C’
d‘13’	b‘00001101’	h‘0D’
d‘14’	b‘00001110’	h‘0E’
d‘15’	b‘00001111’	h‘0F’
d‘16’	b‘00010000’	h‘10’
d‘32’	b‘00100000’	h‘20’
d‘64’	b‘01000000’	h‘40’
d‘128’	b‘10000000’	h‘80’
d‘255’	b‘11111111’	h‘FF’
d‘256’	b‘0000000100000000’	h‘0100’
d‘4096’	b‘0001000000000000’	h‘1000’
d‘32768’	b‘1000000000000000’	h‘8000’
d‘65535’	b‘1111111111111111’	h‘FFFF’

Tabel 2.1: Weergave getallen

2.2 Geheugens

Naargelang het geheugen al dan niet vluchtig is en de (her)programmeerbaarheid ervan kunnen we volgende geheugens onderscheiden:

RAM (Random Access Memory): geheugen dat snel gelezen en gescheven kan worden. Meestal wordt dit gebruikt voor data. Indien de spanning wegvalt wordt het geheugen gewist.

ROM (Read Only Memory): geheugen dat voorgeprogrammeerd is en enkel kan gelezen worden.

PROM (Programmable ROM): ROM geheugen dat één keer kan geschreven worden.

EPROM (Erasable PROM): ROM geheugen dat door UV-licht gewist kan worden (neemt veel tijd) en daarna kan geherprogrammeerd worden.

EEPROM (Electrical EPROM): ROM geheugen dat elektrisch kan gewist en herschreven worden, byte per byte (veel trager dan RAM).

Flash EEPROM: EEPROM geheugen dan enkel in blokken gewist kan worden (goedkoper).

2.3 Programmeertalen

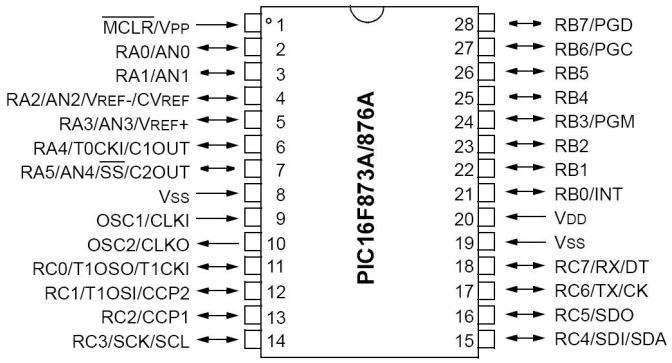
Microcontrollers kunnen op verschillende manieren worden geprogrammeerd. Net zoals C, C++, Java, Basic,... is ook Assembler een programmeertaal. Toch onderscheidt Assembler zich iets van de andere *hogere* programmertalen zoals C. Assembler noemt men een *lagere* programmeertaal omdat deze dichter bij de hardware staat. Je bedient vrijwel direct de hardware met Assembler codes. Bij C is dit niet het geval. Een hogere programmeertaal moet worden *gecompileerd*. Dit proces maakt van bepaalde stukken C-code assembler instructies. En deze Assembler instructies worden dan weer *geassembleerd* tot machinecode. Het gaat de processor tenslotte enkel om de machinecode omdat dat het enigste is wat die begrijpt. Machinecode bestaat uit een hexadecimale getallenrij, iets wat voor de meeste onder ons onleesbaar is. Om dit probleem te verhelpen heeft men lagere programmeertalen ontwikkeld. Zo kan men via een soort leesbare instructies toch een programma schrijven in een taal die redelijk begrijpbaar is voor de mens. Door de steeds groter wordende programma's en complexiteit kwam er de behoefte voor een taal die nog makkelijker was en dichter bij de menselijk taal stond. Hierdoor werd er een hogere programmeertaal ontwikkelt, zoals C, C++,... Algemeen kan men stellen hoe dichter bij de processortaal (hardware), hoe minder duidelijk voor de mens!

Hoofdstuk 3

PIC microcontroller

3.1 Overzicht

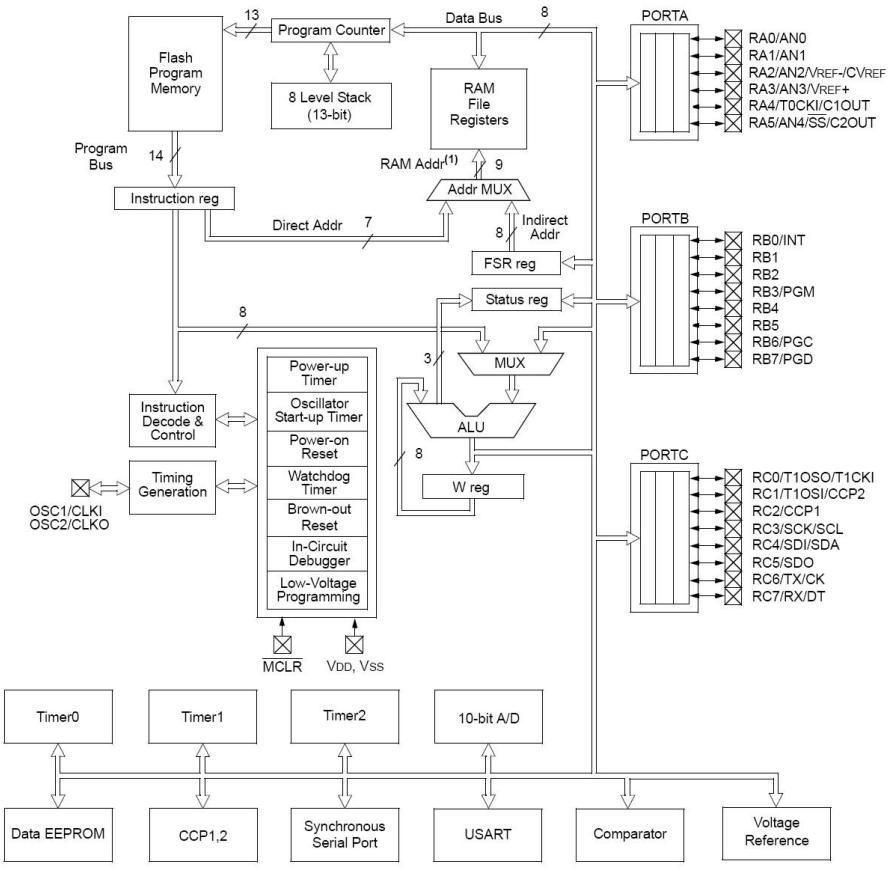
Er bestaan veel verschillende types μC's, elk met hun eigen karakteristieken en functionaliteiten. Veel types kun je onderbrengen in een groep omdat ze dezelfde functies bevatten, deze groep noemen ze dan een familie. Elke chip in zo'n familie heeft dan bepaalde eigenschappen gemeen met de rest. Het verschil zit hem dan in de extra functies en opties die sommige μC's al dan niet bevatten. Zo kan bijvoorbeeld binnen een groep alleen het beschikbaar geheugen verschillen. Tabel 3.1 geeft de specificaties weer van 2 μC's van de *PIC16F87XA* familie van de fabrikant *Microchip*. Zodra je een μC onder de knie hebt, zal je weinig moeite ondervinden bij het leren van een andere, aangezien de techniek en werking van de meeste types op elkaar lijkt. We zullen nu de *PIC16F87XA* μC inleiden. Voor een gedetailleerde beschrijving van deze controller wordt naar de datasheets verwezen. Deze zijn beschikbaar op *www.microchip.be*. Figuur 3.1 geeft de verschillende pinnen weer van de *PIC16F876A*. Het blok diagramma van de componenten intern aanwezig in de controller wordt weergegeven in Figuur 3.2.



Figuur 3.1: PIC16F876A 28-Pin uitvoering

	PIC16F876A	**PIC16F877A**
Operating Frequency DC	DC-20 MHz	DC-20 MHz
Resets (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
Flash Program Memory (14-bit words)	8K	8K
Data Memory (bytes)	368	368
EEPROM Data Memory (bytes)	256	256
Interrupts	14	15
I/O Ports Ports	Ports A, B, C	Ports A, B, C, D, E
Timers	3	3
Capture/Compare/PWM modules	2	2
Serial Communications	MSSP, USART	MSSP, USART
Parallel Communications		PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels
Analog Comparators	2	2
Instruction Set	35 Instructions	35 Instructions

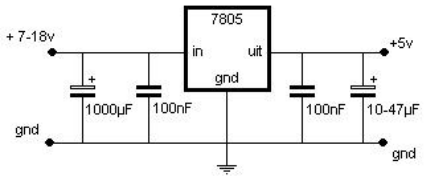
Tabel 3.1: PIC16F87XA familie



Figuur 3.2: PIC16F876A intern

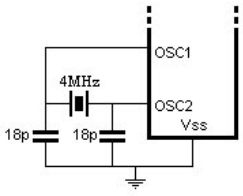
3.2 Minimale Hardware

Voor dat de microcontroller daadwerkelijk iets doet, dienen er enkele externe hardware componenten worden ingesteld. Om te beginnen moet de PIC gevoed worden. De μC heeft een gelijkspanning nodig tussen de 2V en de 5,5V. Een externe (regelbare) voedingsbron, of batterijen kunnen hiervoor gebruikt worden. Wanneer je digitale IC's aan de microcontroller wil koppelen raadt men aan om een gestabiliseerde spanning van 5V te gebruiken. Deze spanning komt namelijk bij de meeste IC's overeen met een logische 1. De *PIC16F876A* wordt bijvoorbeeld gevoed door pin 20 (Vdd) met 5V en pin 8 én pin 19 (Vss) met de ground (0V) te verbinden. Heb je geen gestabiliseerde 5V dan kun je de schakeling weergegeven in Figuur 3.3 gebruiken om een spanning tussen de 18V en 7V om te zetten naar een gestabiliseerde 5V. Deze schakeling maakt gebruik van een spanningsregulator (LM7805). Voor meer details wordt verwezen naar de datasheets.



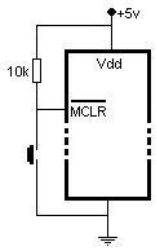
Figuur 3.3: Schakeling gestabiliseerde 5V

Een μC loopt net zoals een pc op een bepaalde frequentie, de klokfrequentie. Alle bewerkingen van de μC lopen op dat signaal. Om het juiste kloksignaal aan te bieden kun je bijvoorbeeld een oscillator (kristal) aansluiten zoals in Figuur 3.4. Elke instructie (op enkele instructies na) heeft 4 klokcycli nodig. De reden hiervoor is dat de interne processor een instructie eerst moet ophalen en decoderen voordat deze het daadwerkelijk kan gaan uitvoeren. Zou je dus een externe clock van 4MHz aansluiten dan worden de instructies uitgevoerd met 1MHz, en dat is dus 1 instructie per 1μs. Enkele instructies vergen dubbel zolang. In de datasheet kun je meer informatie vinden over de waarde van de condensatoren, kristallen en de overeenstemmende te configureren modes.



Figuur 3.4: Schakeling kloksignaal

Als de controller niet meer goed reageert of volledig gestopt is, kan de μC gereset worden via de MCLR pin. Door te resetten komt de μC in zijn beginstand, worden alle instellingen van de controller weer in de default waarde gezet en begint het geprogrammeerde programma opnieuw vanaf het begin. In tegenstelling tot de meeste normale digitale logica waar er iets gebeurt zodra je er een logische 1 op zet, wordt de μC gereset als je een logische 0 op de MCLR pin zet. Wil je dat de μC normaal zijn werk doet, moet je dus zorgen dat die pin altijd een hoog staat. De pin MCLR is een *active low* pin, en wordt in de datasheet aangegeven met een streepje boven de naam van de pin. Een voorbeeld van hoe je de MCLR pin kan aansluiten zodanig dat de μC blijft werken en manueel gereset kan worden wordt weergegeven in Figuur 3.5. Door de MCLR pin te verbinden met de voeding (via een weerstand om de stroom te beperken) ziet de MCLR pin een logische 1. Door een schakelaar te plaatsen tussen de MCLR pin en de ground, kunnen we de μC manueel resetten.

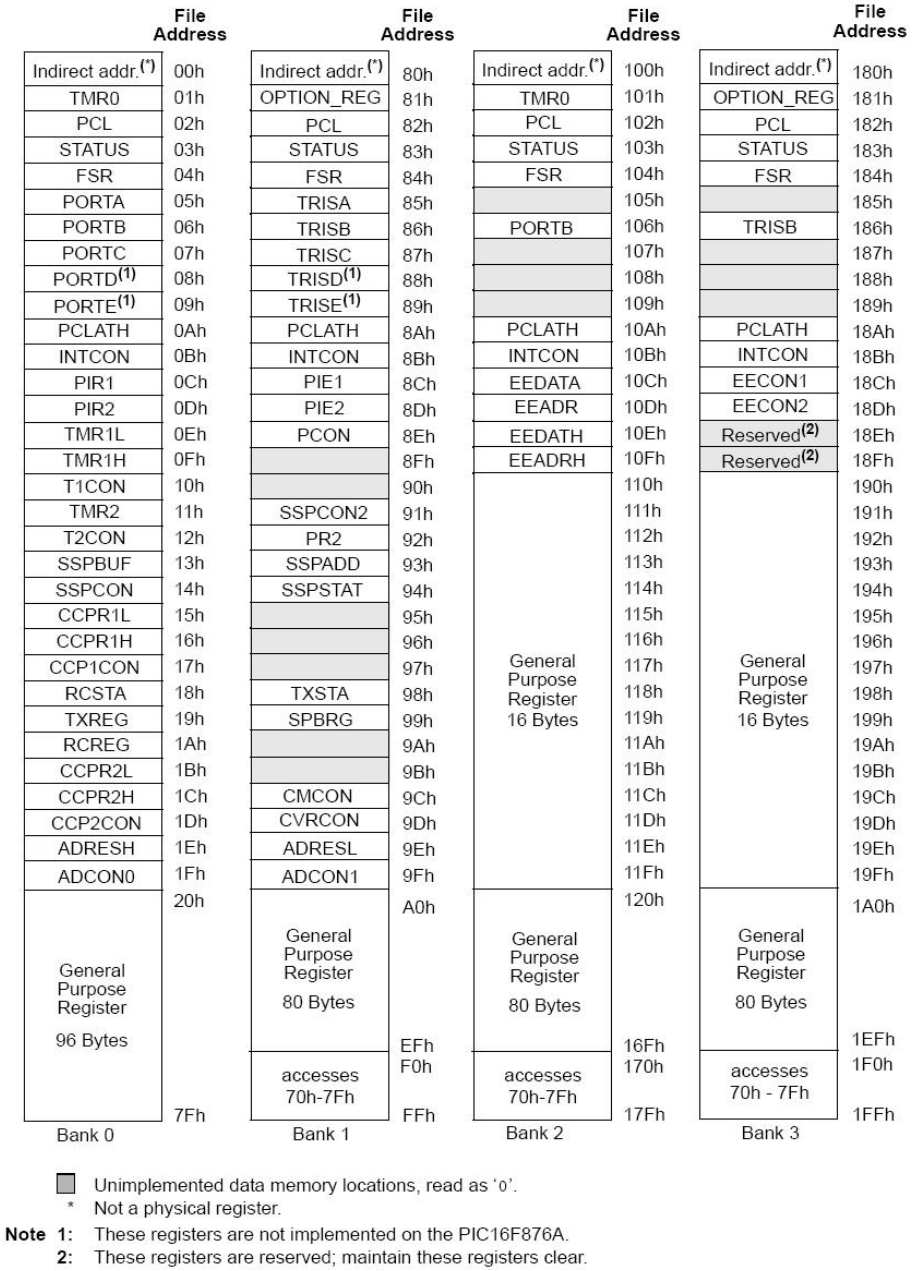


Figuur 3.5: Schakeling reset

3.3 Intern Geheugen

Figuur 3.2 geeft zoals reeds gezegd de interne structuur van de microcontroller weer. Merk in het midden het blokje ALU (*Arithmetic Logic Unit*) op. De ALU is het eigenlijke rekenwonder van de μC en voert de berekeningen uit. Om dat te doen moet er data naar deze unit worden getransporteerd. Dit gebeurt door gebruik te maken van een stuk geheugen ook wel *werkregister* genoemd. De PIC16F87XA heeft maar één *werkregister W* van 8 bits breed. Het aantal werkregisters en de breedte ervan verschilt van μC tot μC. Zo heeft bijvoorbeeld de *Motorola 68HC11* 4 registers, waarvan 2 van 8 bits en 2 van 16 bits breed.

Het principe werkt als volgt: data uit een register wordt in W geplaatst en wordt vervolgens naar de ALU gebracht, waar een bewerking wordt uitgevoerd. Het resultaat wordt opnieuw in een register geplaatst. Ook om data onderling te verplaatsen maakt men gebruik van W. Het aantal beschikbare registers om data te stockeren is beperkt. Figuur 3.6 geeft de registers en hun adres weer voor de PIC16F876A/877A.



Figuur 3.6: PIC16F876A/877A registers en hun adres

Merk op dat het geheugen opgesplitst is in 4 verschillende delen (bank 1 - 4). De vrij te gebruiken registers worden weergegeven onder *Genaral Purpose Registers*. De registers met een specifieke naam zijn niet vrij te gebruiken, maar voorbehouden voor speciale instellingen.

3.4 I/O Poorten

De μC heeft verschillende in- en uitgangen. Deze laten toe verschillende componenten zoals bijvoorbeeld sensoren, motorcontrollers, LCD schermen,... te laten communiceren met de μC. Een *poort* is een reeks gegroepeerde pinnen. Zo heeft bijvoorbeeld de PIC16F876A 3 poorten: een A-poort, een B-poort en een C-poort, met respectievelijk 6, 8 en 8 pinnen. Deze pinnen kunnen afzonderlijk als in- of uitgang geconfigureerd worden. Sommige pinnen worden naast in- of uitgang ook nog gebruikt voor andere speciale toepassingen zoals A/D omzetters, In-Circuit Serial Programming (ICSP), Serial Communication Interface (SCI), Pulse Width Modulation (PWM),...

Het configureren van een poort gebeurt door in een speciaal voorbehouden register een aantal 0'en en 1'en te zetten. Voor de A-poort is dit het register TRISA. Wanneer er in TRISA een bit hoog gezet wordt, komt de overeenstemmende pin van de A-poort (register PORTA) overeen met een ingang. Een bit laag zetten komt overeen met een uitgang. Wanneer de pinnen alternatieve functies hebben dienen er soms nog andere registers geconfigureerd te worden. Zo moet bijvoorbeeld bij de A-poort ook het ADCON1 register van de juiste waarde voorzien worden, omdat de pinnen van de A-poort ook gebruikt kunnen worden als A/D omzetters. Merk op dat TRISA en PORTA in verschillende banken zitten. Voor meer details wordt verwezen naar de datasheets.

Ter verduidelijking een voorbeeldje. In onderstaand geval werken we met de PIC16F876A zoals weergegeven in Figuur 3.1. We wensen pin 21 (RBO), 23 (RB2), 24 (RB3) en 27 (RB6) van de B-poort als ingang, en de ander pinnen van de poort als uitgang. De registers met betekking tot de B-poort worden weergegeven in Figuur 3.7. Om aan onze doelstelling te beantwoorden, moeten we dus de waarde '01001101' in het register TRISB stockeren. Het overeenstemmende stukje assembler code wordt weergegeven in Figuur 3.8.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
06h, 106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
86h, 186h	TRISB	PORTB Data Direction Register								1111 1111	1111 1111
81h, 181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged. Shaded cells are not used by PORTB.

Figuur 3.7: B-poort registers

```
BSF     STATUS, RP0     ; Change to bank1
MOVLW   b'01001101'     ; Put a value in W
                        ; (1's are inputus, 0's outputs)
MOVWF   TRISB           ; Put W into TRISB register
BCF     STATUS, RP0     ; Back to bank0
```

Figuur 3.8: Voorbeeld configuratie B-poort PIC16F876A

3.5 Programma Architectuur

De microcontroller heeft een permanent programmageheugen waarin de geprogrammeerde instructies sequentieel, dit wil zeggen in de volgerde waarin ze geschreven zijn, uitgevoerd worden. Intern heeft de controller een teller (*Program Counter*) die bijhoudt welke instructies uitgevoerd zijn, in de loop van het programma. Helemaal in het begin is deze Program Counter 0000, wat dus inhoudt dat die de instructie op adresgeheugen nummer 0000 moet gaan uitvoeren. Deze Counter wijst altijd naar de volgende uit te voeren instructie. Omdat het wijst naar 0000 wordt dit geheugenadres gelezen en wordt de instructie opgehaald. Vervolgens wordt de Program Counter verhoogd met 1 zodat die weer wijst naar de volgende uit te voeren instructie. Na het ophalen van een instructie wordt deze vertaald zodat de ALU weet wat die ermee moet gaan doen. Zodra die gereed is met het uitvoeren van zijn instructie kan die zijn volgende instructie gaan ophalen, vertalen en uitvoeren. Dit proces wordt herhaald tot de microcontroller gereset wordt. Het programma dient dus in een lus geschreven te worden. De μC stopt niet met het uitvoeren van instructies!

Het normale verloop van een programma wordt dus geregeld door de Program Counter. Er zijn situaties die vereisen dat er onmiddellijk op gereageerd wordt. Hiervoor is de microcontroller voorzien van *interrups*. Laat ons het begrip proberen duidelijk te maken aan de hand van een voorbeeld uit het dagelijks leven. Bijvoorbeeld als de telefoon gaat, onderbreek je (interrupt) je gewone werk, handelt de telefoon af en gaat daarna verder met het werk. Deze interrupt wordt veroorzaakt door een externe bron. Als het tijd is om te gaan lunchen onderbreekt je ook het werk. De bron van deze interrupt is de tijd. De interrupts van een μC werken gelijkaardig. Als een interruptsignaal binnenkomt (van eender welke bron), wordt het programma dat sequentieel doorlopen werd, onderbroken, en zal de Program Counter eerst de instructies binnen in de interrupt routine uitvoeren, en vervolgens terug keren naar het oorspronkelijke programma. Een gouden regel bij interrupts: maak de interrupt routine zo kort mogelijk. Dit zal de hinder voor het hoofdprogramma, de interferentie met andere interrupts en de performantie ten goede komen.

3.6 Instructielijst

Om de PIC microcontroller te programmeren heeft men 35 instructies. We kunnen de instructies onderverdelen in 3 groepen: instructies gericht op *bytes*, *bits* of *literals*. We zullen de verschillende instructies even verduidelijken. Voor meer details wordt verwezen naar de datasheets. Alle instructies die rechtsteeks met bytes werken worden weergegeven in onderstaande Tabel 3.2. De eerste kolom geeft de assembler code van de instructie weer, de tweede kolom beschrijft zijn parameters en de derde kolom de instructie zelf. Een *f* stelt een *file register* voor, en een *d* wordt gebruikt om de bestemming (*destination*) weer te geven. Wanneer we d de waarde 0 geven wordt het resultaat van de instructie in W geschreven, met d de waarde 1 wordt het resultaat in f geschreven. Een voorbeeldje: wensen we de waarde in register 20h met 1 te verhogen en het resultaat in register W te stockeren dan luidt de instructie: *INCF 0x20,1*. Merk op dat de waarde in adres 20h niet verandert is!

Instructie	**Parameters**	**Betekenis**
ADDWF	f, d	Add W and f
ANDWF	f, d	AND W with f
CLRF	f	Clear f
CLRW		Clear W
COMF	f, d	Complement f
DECF	f, d	Decrement f
DECFSZ	f, d	Decrement f, Skip if 0
INCF	f, d	Increment f
INCFSZ	f, d	Increment f, Skip if 0
IORWF	f, d	Inclusive OR W with f
MOVF	f, d	Move f
MOVWF	f	Move W to f
RLF	f, d	Rotate Left through Carry
RRF	f, d	Rotate Right through Carry
SUBWF	f, d	Subtract W from f
SWAPF	f, d	Swap nibbles in f
XORWF	f, d	Exclusive OR W with f

Tabel 3.2: Instructies met betrekking tot bytes

Zoals reeds vermeld wordt het W register gebruikt om data te verplaatsen. Er kunnen slechts complete bytes worden verplaatst. Met de instructies gericht op bits kunnen we in een register een bit aanpassen. Wanneer we een bit hoog maken dan noemt dit *setten*, een bit laag maken noemt men *clearen*. Volgende instructies in Tabel 3.3 setten, clearen of testen de *bit b* in register f.

Instructie	**Parameters**	**Betekenis**
BCF	f, b	Bit clear f
BSF	f, b	Bit set f
BTFSC	f, b	Bit test f, Skip if clear
BTFSS	f, b	Bit test f, Skip if set

Tabel 3.3: Instructies met betrekking tot bites

De laatste groep instructies hebben betrekking tot getallen of lettercombinaties, of zijn controle operatoren. Ze worden weergegeven in Tabel 3.4. De parameter k geeft hier de lettercombinaties (*literal*) of het getal weer.

Instructie	**Parameters**	**Betekenis**
ADDLW	k	Add literal and W
ANDLW	k	AND literal with W
CALL	k	Call subroutine
CLRWDT		Clear Watchdog Timer
GOTO	k	Go to address
IORLW	k	Inclusive OR literal with W
MOVLW	k	Move literal to W
NOP		No operation
RETFIE		Return from interrupt
RETLW	k	Return with literal in W
RETURN		Return from subroutine
SLEEP		Go into standby mode
SUBLW	k	Subtract W from literal
XORLW	k	Exclusive OR literal with W

Tabel 3.4: Instructies met betrekking tot literals

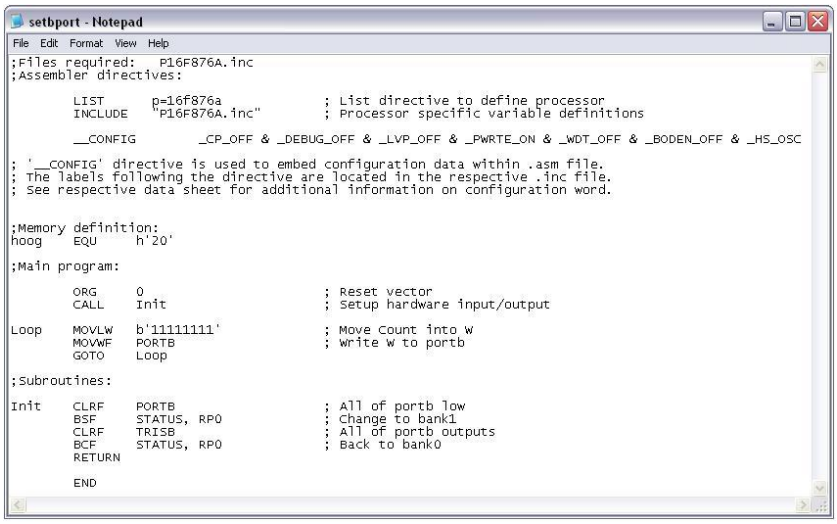
Sommige instructies beïnvloeden het *STATUS* register. Het al dan niet setten van de *ZERO FLAG Z*, *CARRY FLAG C* of *DIGIT CARRY FLAG DC* in het STATUS register kan gebruikt worden in de programma's om bijvoorbeeld keuzes te maken, instructies over te slaan, of submodules op te roepen. Zodra er een berekening of bewerking wordt uitgevoerd en de uitkomst gelijk is aan 0, dan set de ALU bit 2 (Z) van het STATUS register. Als er een berekening wordt uitgevoerd die een uitkomst heeft die groter is dan de 8 bits die ter beschikking staan dan geeft bit 0 (C) van het STATUS register dat aan. Dus zou je de binaire waarde 10000000 optellen bij 10000000 dan wordt de uitkomst 100000000. Maar aangezien de ALU maar met 8 bits werkt blijft er als resultaat 00000000 over. Om te melden dat de berekening buiten het bereik viel wordt bit C geset. In de datasheets vind je welke instructie welk bits in het statusregister beïnvloed.

Hoofdstuk 4

Programmeren, Assembleren en Inladen

4.1 Programmeren

Nu je kennis hebt van de controller, zijn mogelijkheden en de te gebruiken instructies, kan je aan de slag en je eigen programma schrijven. De programma's worden in Assembler geschreven, en vervolgens geassembleerd tot machinetaal. Een programma wordt in *Notepad* of een andere teksteditor geschreven. Je slaagt het bestand als volgt op **.asm* met * de door jou gekozen naam. Enkele spelregels en niet besproken instructies dienen nog uitgelegd te worden. Ze zullen verduidelijkt worden in onderstaand voorbeeld (Figuur 4.1).



Figuur 4.1: Voorbeeldprogramma in assembler geschreven in Notepad

Alle tekens achter een ; -teken worden niet gelezen door de Assembler. Vandaar de commentaar na een ; die noodzakelijk is om het programma begrijpbaar te maken.

Allereerst vertellen we met de instructie *LIST* de Assembler met welke processor we werken.

Om het ons te vereenvoudigen werden door de fabrikant een aantal registers en/of bits ervan reeds benoemd. Een lijst hiervan wordt weergegeven in een **.INC* bestand, dat gedownload kan worden op de website van de fabrikant *www.microchip.be*. Bijvoorbeeld adres '0006' noemt men *PORTB* in de file *P16F876A.INC*. Willen we in ons programma gebruik maken van de reeds benoemde registers dan moeten we verwijzen naar het **.INC* bestand. We doen dit door de instructie *INCLUDE*.

Vervolgens worden er met de instructie *__CONFIG* enkele specifieke toepassingen van de μC geconfigureerd. Bijvoorbeeld, wanneer we een externe oscillator (kristal) van 20MHz gebruiken, dient er *__CONFIG _HS_OSC* te staan.

De instructie *EQU* wordt gebruikt om net zoals in het *.INC bestand registers te benoemen. Hierna kan je in het programma verwijzen naar het register door de gekozen naam op te roepen in plaats van het adres.

Met de instructie *ORG* wordt de Assembler verteld waar de Program Counter moet starten (h'0000').

Merk de structuur (layout) van de tekst op. Instructies, hun parameters en eventuele commentaar worden gescheiden door *tab's* om het geheel overzichtelijker te maken. Voor de instructie dient men ook een tab te plaatsen met daarvoor eventueel een *label*. Bijvoorbeelden *Loop* en *Init* zijn labels. Labels maken het mogelijk om over een reeks instructies te springen.

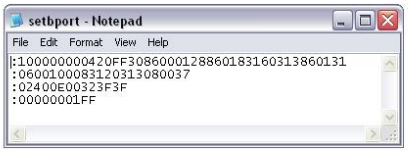
Tenslotte moet de Assembler verteld worden wanneer het programma beeindigd is. Dit gebeurd door de instructie *END*.

4.2 Assembleren

Het programma is nu min of meer leesbaar voor ons, en de Assembler, maar moet nog omgezet worden naar machinetaal. Dit gebeurt met de software getoond in Figuur 4.2. De software is te downloaden op *www.velleman.be*.

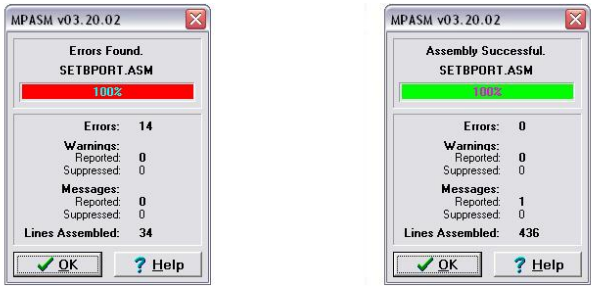
Figuur 4.2: MPASM Programma Assembler

Tijdens het assembleren worden er 4 bestanden gemaakt. Een eerste bestand heeft een extentie **.HEX*. Hier is het de microcontroller allemaal om te doen, dit is het enige bestand dat de μC begrijpt. Het bevat het programma in machinetaal. Figuur 4.3 geeft het voorbeeldprogramma weer in machinetaal. Merk op dat dit een reeks van hexadecimale getallen is, vandaar de extentie *.HEX.



Figuur 4.3: Voorbeeldprogramma in machinetaal (*.HEX)

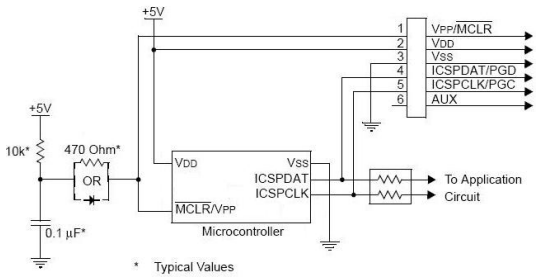
De andere drie bestanden: **.COD*, **.ERR* en **.LST* bevatten respectievelijk de programmacode, een foutenlijst en een volledig verslag van het assembleerproces. Het laatste bestand (*.LST) geeft weer waar er iets mis gelopen is (*error*), of waar er aandacht moet geschonken worden (*warning*). Alle bestanden kunnen steeds geopent worden in Notepad. Het al dan niet slagen van het assembleren wordt ook weergegeven op het scherm zoals in Figuur 4.4. Links is er iets misgelopen, rechts was het assembleren succesvol.



Figuur 4.4: Verslag van het assembleerproces

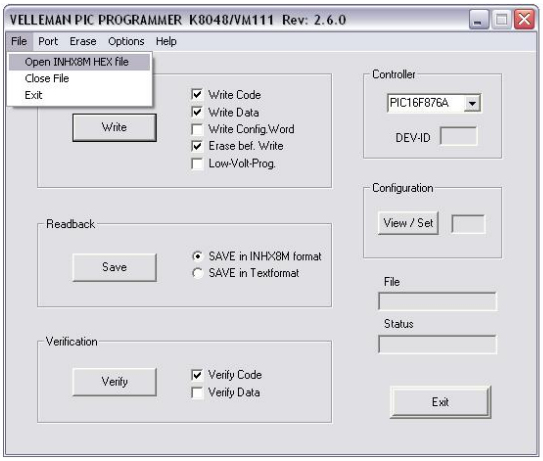
4.3 Programma Inladen

Wanneer het programma juist geassembleerd is, rest er ons enkel nog het programma in te laden in de microcontroller. De PIC16F87XA kan geprogrammeerd worden wanneer de chip in de gewenste toepassing zit. Dit noemt men het *In-Circuit Serial Programming* (ICSP). Figuur 4.5 geeft weer hoe we de pinnen die we gebruiken om het programma in te laden beschikbaar maken. Deze pinnen kunnen we nu verbinden met hardware (beschikbaar in het labo) die het mogelijk maakt om een programma in de μC in te laden.



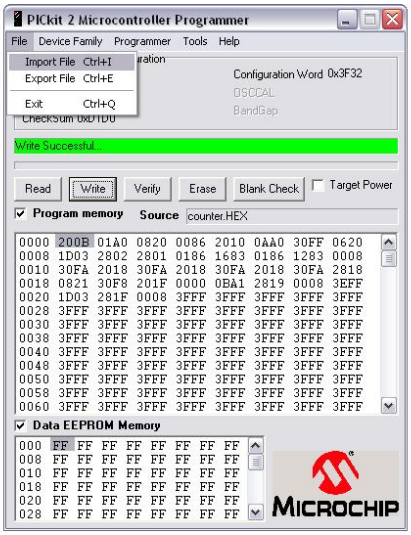
Figuur 4.5: In-Circuit Serial Programming

Twee verschillende programma laders worden kort toelichten. De eerste maakt gebruik van een seriële poort (COM). Het *.HEX bestand dient ingeladen te worden zoals weergegeven in Figuur 4.6. De keuze van de COM-poort, snelheid van inladen, PIC-instelling en spanning waarmee het programma geladen wordt kunnen o.a. voor problemen zorgen. Stel de juiste COM-poort in (onder het tabblad *port* kan je de verschillende poorten kiezen). Onder het tabblad *options/speed* kan je de snelheid waarmee het programma ingeladen wordt veranderen. Meestal werkt de laagste snelheid. Vergeet onder *controller* niet de juiste gebruikte controller in te stellen. Soms kan het zijn dat de spanning zakt tijdens het programmeren. Het is aangeraden pin *PGD* en pin *PGC* te ontkoppelen van de toepassing tijdens het programmeren.



Figuur 4.6: Velleman programma lader

De tweede programma lader maakt gebruik van een USB poort. Het *.HEX bestand dient ingeladen te worden zoals weergegeven in Figuur 4.7. Onder het tabblad *Device Family* dien je *midrange* aan te vinken wanneer je werkt met de PIC16F87XA. De gebruikte μC wordt dan door de *Pickit 2 Programmer* automatisch herkend. Vervolgens kan het programma door op *Write* te klikken ingeladen worden.



Figuur 4.7: Pickit 2 programma lader

Hoofdstuk 5

Voorbeelden

In dit hoofdstuk worden enkele programma's weergegeven.