TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Tietokonetekniikka

Tutkintotyö

Kari-Matti Mäkelä

KAUKO-OHJATTAVA KOODILUKKO

Työn ohjaaja Yliopettaja Mauri Inha
Työn teettäjä Kari-Matti Mäkelä
Tampere 2008

TAMPEREEN AMMATTIKORKEAKOULU
Tietotekniikka
Tietokonetekniikka
Mäkelä, Kari-Matti
Kauko-ohjattava koodilukko
Tutkintotyö 27 sivua + 16 liitesivua
Työn ohjaaja Yliopettaja Mauri Inha
Työn teettäjä Kari-Matti Mäkelä
Marraskuu 2008
Hakusanat kauko-ohjain, IR, PICmicro, koodilukko, PIC16F690

TIIVISTELMÄ

Tässä työssä suunniteltiin ja toteutettiin kauko-ohjattava koodilukko eli prototyyppi laitteesta, joka tunnistaa kaukosäätimen lähettämiä IR-muotoisia näppäinkomentoja ja tunnistaa niistä neljän esiohjelmoidun näppäimen sarjan, ohjaten siten esimerkiksi sähköistä lukkoa. Työ toteutettiin PIC16F690-mikrokontrollerilla, ja käytettäväksi kaukosäädinprotokollaksi valittiin Philipsin kehittämä RC5. Työn tuloksena saatiin toimiva laite sekä siihen kuuluva ohjelma. Työn tavoitteena oli tutustuttaa tekijä assembly-ohjelmointiin ja toimivan laitteen rakentamiseen.

TAMPERE POLYTECHNIC
Information Technology
Computer Engineering
Mäkelä, Kari-Matti
Remote Controlled Combination Lock
Engineering Thesis 27 pages, 16 appendices
Thesis Supervisor Senior Teacher Mauri Inha
May 2008
Keywords remote control, IR, PICmicro, combination lock, PIC16F690

ABSTRACT

This thesis consisted of designing and constructing a remote controlled combination lock. The prototype device recognizes IR remote control commands send from a remote controller, and from them recognizes a series of four preprogrammed keystrokes. This recognition can for example drive an electric lock. Work was based on a PIC16F690 microcontroller. RC5 remote control protocol by Philips was chosen to be used. Results from this thesis include a functioning device and the software for it. The goal of this thesis was to familiarize oneself to assembly programming and how to construct a functioning device.

Alkusanat

Olen opiskellut assembly-ohjelmointia ja prosessoritekniikkaa useillekin eri prosessoreille sekä ammatti- että ammattikorkeakoulussa, mutta käytännössä koskaan en ole saanut motivaatiota ja tilaisuutta valmistaa kokonaista toimivaa laitetta ja sille ohjelmistoa. Siksi valitsin tämän aiheen, kuitenkin hieman peläten, että työ voisi osoittautua minulle liian haastavaksi. Nyt jälkeenpäin voin todeta, että työn tekeminen on ollut erittäin palkitsevaa. Työn teko on antanut uutta ymmärrystä sulautettujen järjestelmien toimintaan ja myöskin lisämotivaatiota syventää omia tietojani.

Tampereella 6.11.2008

Kari-Matti Mäkelä

1 JOHDANTO

Työn tavoitteena oli tehdä kauko-ohjattava koodilukko eli laite, joka tunnistaa kaukosäätimen IR-signaalina lähettämiä näppäinkomentoja ja tunnistaa niistä neljän esiohjelmoidun näppäimen sarjan, joka ohjaa esim. sähköistä lukkoa. Käytännössä työ toteutettiin PIC-mikrokontrollerilla, ja kaukosäädinprotokollaksi valittiin yleisesti käytössä oleva Philipsin RC5-protokolla. Työn tuloksena saatiin toimiva kytkentä ja siihen kuuluva koodi.

Itse työn aiheella ei välttämättä ole kovin hyödyllisiä käytännön sovelluksia. Pääsyy aiheen valintaan olikin halu tutustua syvemmin assembly-ohjelmointiin käytännön tasolla, sekä suunnitella ja toteuttaa hieman koulun koekytkentöjä monimutkaisempi toimiva laite alusta loppuun. IR-koodien vastaanotto ja tunnistus vaikutti sopivan vaikeustason työltä ilman aiempaa PIC-ohjelmointikokemusta.

Ensimmäiseksi työssä kerrotaan hieman yleisiä pohjatietoja RC5-protokollasta sekä käytetyistä komponenteista. Seuraavaksi käsitellään laitteen suunnitteluun, kytkentään ja ohjelmointiin liittyviä piirteitä. Sen jälkeen tutustutaan tarkemmin itse ohjelmaan, ja sen toimintaperiaatteita ja rakennetta eritellään yksityiskohtaisesti. Lopuksi pohditaan vielä jatkokehitysmahdollisuuksia sekä tehdään loppuyhteenveto työstä.

2 TAUSTATIETOJA

2.1 Infrapuna kauko-ohjauksessa

Infrapunalla (IR) tarkoitetaan ihmissilmälle näkymätöntä sähkömagneettista säteilyä. Infrapuna sijaitsee sähkömagneettisessa spektrissä juuri näkyvän valon alapuolella, ennen punaisen valon aallonpituutta (kuva 1). Kaikki lämmönlähteet, kuten aurinko ja sähkölamput, lähettävät myös IR-säteilyä. Kulutuselektroniikassa, kuten televisioissa ja videoissa, on jo pitkään käytetty infrapunaan perustuvia kaukosäädintekniikoita. Tekniikkaan tarvittavat IR-ledit ja vastaanottimet ovat yksinkertaisia ja halpoja toteuttaa, niiden kantama on sisätiloihin sopiva ja ne myös

kestävät hyvin radiotaajuuksisia häiriöitä. IR-signaali käyttäytyy kuten näkyvä valo ja tarvitsee siksi suhteellisen esteettömän tien lähettimeltä vastaanottimelle.

Kuva 1 IR valospektrissä. /3/

Sovelluksissa, joissa signaalin on kuljettava esimerkiksi toiseen huoneeseen, käytetäänkin usein radiotaajuudella toimivia tekniikoita, esimerkiksi Bluetoothia. IR on kuitenkin ylivoimaisesti suosituin kauko-ohjaustapa. Useimmiten IR-signaalit on moduloitu n. 40 kHz:n taajuudella häiriösietoisuuden parantamiseksi. /1/

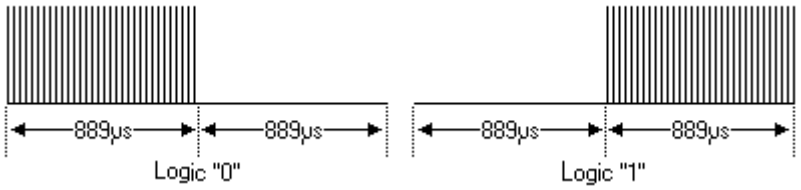
2.2 RC5-protokolla

IR-tekniikkaan perustuvia kaukosäädinprotokollia on useita erilaisia, ja monet valmistajat, kuten Sony, Panasonic, JVC, Sharp, NEC ja Nokia, käyttävätkin omaa protokollaansa. Yksi yleisimmistä on kuitenkin Philipsin kehittämä RC5- ja siihen perustuvat RC5x- ja RC6-protokollat. Näitä käytetään paitsi Philipsin omassa kulutuselektroniikassa, myös monien muiden valmistajien laitteissa.

Valitsin RC5:n käytön, koska se on hyvin dokumentoitu ja siitä löytyy runsaasti tietoa. Vaihtoehtona olisi ollut myös hieman yksinkertaisempi Sonyn protokolla.

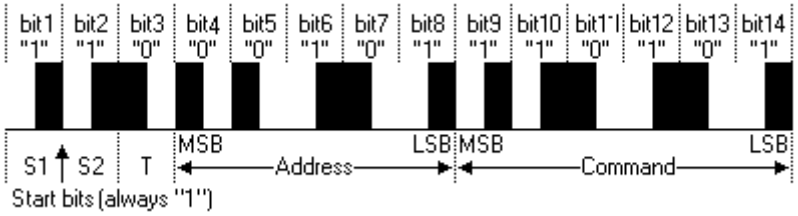
RC5:ssä komennot lähetetään sarjamuotoisena binääridatana. Komennot ovat aina vakion 14 bitin mittaisia. Yksittäiset bitit on koodattu ns. Manchester-koodauksella, jossa loogista 1:tä vastaa signaalin nousu matalasta tilasta korkeaan tilaan, ja loogista 0:aa lasku korkeasta tilasta matalaan tilaan. (Kuva 2.) Korkea tila lähetetään moduloimalla IR-signaalia 36 kHz:n taajuudella; matalan tilan aikana ei lähetetä signaalia ollenkaan. Yhden bitin kesto vastaa 64:ää 36 kHz:n pulssia, eli 64 x 1 / 36 kHz = 1,778 ms. Yhden koodin pituus on tällöin 14 x 1,778 ms = 24,889 ms. Koodia toistettaessa kahden peräkkäisen koodin väliin jäävä aika vastaa

50:tä bitin pituutta eli 88,889 ms, jolloin ensimmäisen koodin alusta seuraavan koodin alkuun kuluu 113,778 ms.



Kuva 2 Manchester-koodaus. /2/

Yksi 14-bittinen RC5-koodi on esitetty kuvassa 3. Se alkaa aina kahdella Start-bitillä, jotka vastaavat loogista 1:tä. Nämä antavat vastaanottimen automaattiselle tason säädölle (AGC) aikaa säätää taso sopivaksi ja osoittavat, että kyseessä on RC5-protokollan - eikä esimerkiksi jonkun toisen protokollan - mukainen signaali. Start-bittien jälkeen lähetetään yksi Toggle-bitti. Toggle-bitti pysyy samana niin kauan kuin kaukosäätimen nappia pidetään pohjassa, ja vaihtaa tilaansa kun näppäin päästetään irti. Tällä viestitetään vastaanottimelle, pidetäänkö näppäintä pohjassa vai painetaanko sitä toistuvasti.



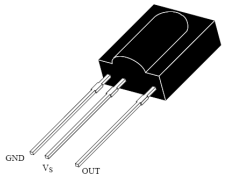
Kuva 3 Esimerkki RC5-koodista. /2/

Seuraavaksi lähetetään 5 Address-bittiä (eniten merkitsevä bitti ensin). Address-bitit määrittävät ohjattavan laiteluokan. Esimerkiksi televisioille, videoille ja vahvistimille on omat laiteosoitteet. Näin voidaan ohjata Philips-merkkistä televisioita ilman että Philips-merkkinen video häiriintyy. Laiteosoitteita on 32 kappaletta. Lopuksi koodissa on 6 Command-bittiä (jälleen eniten merkitsevä bitti ensin), jotka vastaavat laitteelle annettavia komentoja, esimerkiksi "äänenvoimakkuus +", "äänenvoimakkuus –", "numeronäppäin 1" tai "standby". Komentoja voi olla maksimissaan 64 erilaista. Taulukossa 1 on esitetty muutamia komentoja (desimaalinumeroina) laiteosoitteelle 0 (TV).
/2/

Taulukko 1 Esimerkki RC5-komennoista laiteosoitteelle 0 (TV). /3/

0 - 9	numeronäppäimet 0 - 9
12	standby
13	mute
16	äänenvoimakkuus +
17	äänenvoimakkuus -

2.3 IR-demodulaattori TSOP1738

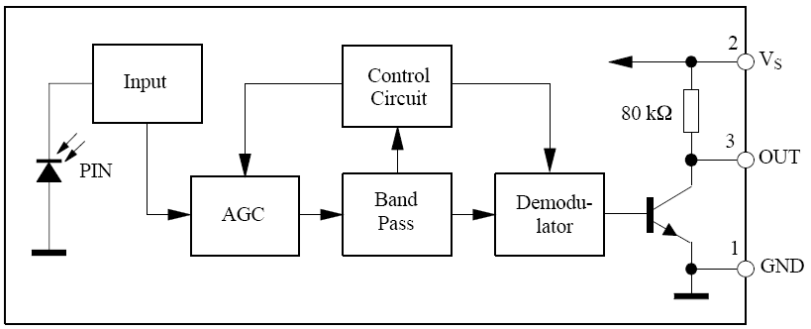


Kuva 4 TSOP1738. /4/

Jotta IR-signaali voitaisiin muuttaa mikrokontrollerille helposti ymmärrettävään muotoon, signaalia tarvitsee muokata moneen kertaan. Signaali täytyy mm. muuttaa sähköiseen muotoon, siinä mahdollisesti olevat häiriöt ja ylimääräiset taajuudet on suodatettava pois, signaali on demoduloitava ja todennäköisesti signaalin tasoa on nostettava. Tätä helpottamaan monet valmistajat tarjoavat piirejä, joihin edellä mainitut toiminnot on integroitu yhteen komponenttiin. /1/

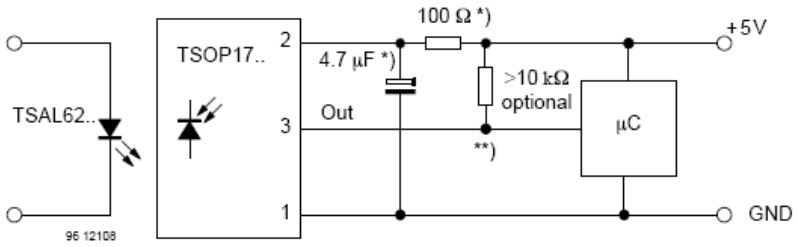
Piirit on yleensä jaoteltu käytetyn modulaatiotaajuuden mukaan. Piiri tulisikin valita mahdollisimman läheltä vastaanotettavan protokollan käyttämää modulaatiotaajuutta, jotta herkkyys olisi paras mahdollinen. Tähän työhön valitsin Vishayn valmistaman TSOP1738-piirin. (Kuva 4.)

Piirin taajuus on 38 kHz, mikä eroaa hieman RC5-protokollan käyttämästä 36 kHz:stä. 36 kHz:n piiriä ei kuitenkaan hankintahetkellä ollut saatavissa, ja käytäntö osoitti 38 kHz:n olevan riittävän herkkä.



Kuva 5 TSOP1738 sisäinen kytkentä. /4/

Kuvassa 5 on esitetty piirin sisäinen kytkentä. Piirin kotelo toimii optisena suodattimena. Lisäksi siinä on 38 kHz:n kaistanpäästösuodin, automaattinen tasonsäätö ja demodulaattori. Piirissä on vain kolme jalkaa, käyttöjännite, maa ja ulostulo. On huomattava, että ulostulo on "active-low" -tyyppinen, eli looginen 0 vastaa korkeaa jännitettä ja looginen 1 matalaa jännitettä. Ulostulon jännitteet ovat TTL-tasoisia, joten ne ovat helposti tulkittavissa mikrokontrollerilla. Kuvassa 6 on esitetty mallikytkentä. Piirin värähtelyn estämiseksi on käyttöjännitteen ja maan väliin kytkettävä 4,7 µF:n kondensaattori sekä mahdollinen ylösvetovastus. /4/



Kuva 6 TSOP1738 esimerkkikytkentä. /4/

2.4 PIC-mikrokontrollerit

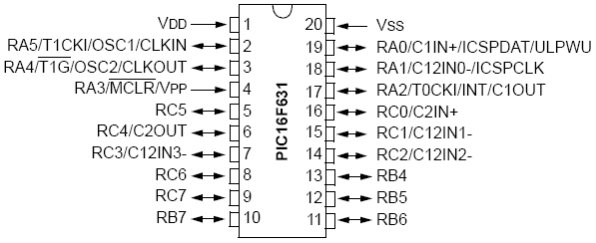
Mikrokontrollerilla tarkoitetaan mikroprosessoria, jossa on yhdelle piirille integroitu sekä ohjelma- että käyttömuisti (RAM) ja useita eri oheislaitteita, kuten sarjaliikenneohjaimia tai vaikkapa USB-ohjaimia. Mikrokontrollerit usein tarvitsevat hyvin vähän ulkoisia piirejä toimiakseen. Niitä käytetäänkin laajalti monissa sulautetuissa järjestelmissä niiden monipuolisuuden ja halpuuden takia. /5/

PIC, oikealta nimeltään PICmicro, on yleisnimitys amerikkalaisen Microchip Technologyn valmistamalle mikrokontrolleriperheelle. Valtaosa PIC:eistä on yleensä 8-bittisiä, eli dataa luetaan niissä aina 8-bittiä kerrallaan. PIC:in prosessorit ovat yleensä RISC-tyyppisiä (Reduced Instruction Set Computer), eli käskyt ovat yksinkertaisia ja niitä on vähän. Tästä on hyötynä nopea ja selkeä käskyjen suorittaminen, koska useimmat käskyt kestävät vakioajan suorittaa. Käskyt on myös helppo opetella ja ohjelmointi voidaan suorittaa suoraan konekielellä, jolloin koodista saadaan tehokasta. Varjopuolena on se, että monimutkaisemmat operaatiot vaativat huomattavasti enemmän käskyjä kuin CISC-tyyppisissä (Complex Instruction Set Computer) prosessoreissa, joissa on enemmän erikoistuneita komentoja kuin RISC-prosessoreissa.
/6/

PICmicrot noudattavat ns. Harvard-arkkitehtuuria, eli niissä on erilliset muistiavaruudet ohjelmalle ja datalle. Tämä mahdollistaa käskyjen nopean suorituksen, koska osoite- ja dataväyliä voidaan käyttää samanaikaisesti sen sijaan että ohjelma- ja datamuisti käyttäisivät samaa väylää tiedonsiirrossa prosessorille. 8-bittiset PIC:it jaotellaan yleensä kolmeen tuoteperheeseen sen perusteella, kuinka pitkää käskysanaa niissä käytetään (eli kuinka monta bittiä on varattu yhdelle käskyriville). Perustason (Baseline) PIC:eissä on 12 bitin mittaiset käskysanat, keskitason (Midrange) PIC:eissä 14 bitin ja ylemmän tason (High-end) PIC:eissä 16 bitin käskysanat. Lisäksi Microchip valmistaa myös 16-bittisiä PIC:ejä kuten PIC24, mutta ne eroavat olennaisesti 8-bitin PIC:eistä. Kaikki 8-bittiset PIC:it ovat keskenään hyvin samanlaisia ja eroavat lähinnä oheislaitteiltaan ja IO-pinnien määrältään. Sama ohjelma voidaan usein pienin muutoksin ajaa suuressa osassa eri PIC-malleja.
/5, 6, 7/

Seuraavassa luvussa kerrotaan hieman tarkemmin tässä työssä käytetystä PIC16F690-mikrokontrollerista.

2.5 PIC16F690



Kuva 7 PIC16F690 /8/

PIC16F690 on keskitason (Midrange) 8-bittinen mikrokontrolleri. 20 pinnistä 17:ää voidaan käyttää IO-pinneinä (kuva 7). Kellotaajuus voi olla mikä tahansa väliltä 0 - 20 MHz, ja kellopulssin lähteenä voidaan käyttää sekä ulkoisia että sisäisiä oskillaattoreita. Sisäisesti voidaan luoda 1 % tarkkuudella 32 kHz:n - 8 MHz:n kellosignaalia. Käyttöjännite saa olla 2,0 V:sta 5,5 V:iin, ja piirissä on myös monipuoliset virransäästöominaisuudet. Ohjelmamuisti on 100 000 kirjoitusta kestävä 4096 sanan flash-muisti. Käyttömuistia ja EEPROM-muistia on 256 tavua. Muihin ominaisuuksiin kuuluvat mm. 8 tasoinen laitteistopohjaisen pinomuisti paluukäskyille, 10-bittinen 12-kanavainen AD-muunnin, yksi 16-bittinen ja kaksi 8-bittistä ajastinta sekä USART-moduuli sarjamuotoiseen liikenteeseen.
/8/

2.5.1 Muistin rakenne

Käyttömuistia eli RAM:ia kutsutaan rekistereiksi. Koska käskyt, jotka käsittelevät rekistereitä käyttävät siihen 7 bittiä, vain 128 rekisteriä voidaan osoittaa. Tämän rajoituksen poistamiseksi muisti on järjestetty neljään pankkiin, jolloin käytettävä pankki on valittava STATUS-rekisterin RP0- ja RP1-bittejä käyttämällä. Ensimmäiset 32 osoitetta jokaisessa pankissa on varattu SFR-rekistereille (Special Function Register) eli rekistereille, joilla PIC käyttää oheislaitteitaan. Esim. IO-pinnejä RA0-RA5 käsitellään PORTA- ja TRISA-rekisterien kautta. SFR-rekisterien jälkeen on tilaa GPR-rekistereille (General Purpose Register), joihin tallennetaan ohjelman muuttujat. Joitakin tärkeitä SFR-rekistereitä, kuten STATUS

ja INTCON, voidaan käyttää mistä tahansa pankista, kuten myös GPR-muistialuetta välillä 70h - 7Fh.
/7/

Ohjelmamuistia käsitellään 13-bittisen ohjelmalaskurin välityksellä. Alimmat 8 bittiä sijaitsevat PCL-rekisterissä ja ylimmät PCLATH-rekisterissä. Nämä määräävät ohjelman teoreettiseksi maksimikooksi 8192 kpl 14-bittisiä käskyjä. Näistä PIC16F690:ssä on kuitenkin käytännössä toteutettu vain ensimmäiset 4096 osoitetta.

Lisäksi PIC sisältää 256 tavua sisältönsä myös jännitteettömässä tilassa säilyttävää EEPROM-muistia, johon voidaan kirjoittaa ohjelman suorituksen aikana.
/8/

2.5.2 Käskykanta

Käskyjä on yhteensä 35. Kaikkien käskyjen suorittamiseen kuluu saman verran aikaa: 4 kellopulssin eli yhden käskysyklin verran. Tässä työssä PIC on asetettu käyttämään sisäistä 4 MHz:n kelloa, jolloin yhden käskyn suorittamiseen menee aikaa 1 µs. Poikkeuksena ovat ehdolliset ja haarautumiskäskyt, jotka kestävät kahden käskysyklin ajan.
/8/

Käskyt voidaan jakaa kolmeen ryhmään: tavuihin liittyvät operaatiot, bitteihin liittyvät operaatiot ja vakioihin liittyvät sekä ohjausoperaatiot. Tavuihin kohdistuvat operaatiot sisältävät 6-bittisen operandin, 7-bittisen muistiosoitteen sekä kohdebitin d. Toisena operandina on W-rekisteri eli työrekisteri. Operaation tulos talletetaan kohdebitin mukaan joko muistiosoitteeseen (d = 1) tai W-rekisteriin (d = 0). Assembly-kääntäjä osaa myös tulkita kohdebitin kirjaimista f (File Register, d = 1) ja w (d = 0). Esimerkiksi käsky

INCF data, w

kasvattaa rekisterin "data" sisältöä yhdellä ja tallettaa sen W-rekisteriin.
/7/

Bitteihin liittyvät operaatiot kohdistuvat aina jonkun rekisterin tiettyyn bittiin. Ne sisältävät 7-bittisen muistiosoitteen, 4-bittisen operandin ja 3-bittisen numeron. Käskyllä joko asetetaan tai nollataan haluttu bitti. Näihin kuuluvat myös käskyt, joilla testataan tietyn bitin arvo. Esimerkiksi käsky

BTFSS STATUS, C

lukee STATUS-rekisterin C-bitin ja hyppää seuraavan käskyn yli jos C = 1. Tämän käskyn suorittaminen vie kahden käskysyklin verran aikaa.
/7/

Vakioihin liittyvissä operaatioissa vakiodata on yksi operandi ja toisena operandina on aina W. Myös hyppykäskyt kuten CALL tai GOTO kuuluvat tämän tyyppisiin käskyihin. Niissä vakiona on kohdeosoite. Esimerkkinä käsky

XORLW, b'00000001'

tekee XOR-operaation W-rekisterin ja binäärisen luvun 0000 0001 välillä ja tallettaa tuloksen W-rekisteriin. On myös muutamia PIC:n toimintaan liittyviä käskyjä, kuten SLEEP, joka ohjaa PIC:in virransäästötilaan.
/7/

Koko käskykanta selityksineen löytyy liitteestä 3.

2.5.3 Tärkeimmät rekisterit

Seuraavassa käsitellään lyhyesti muutamaa tässä työssä käytettyä SFR-rekisteriä.

Taulukko 2 STATUS-rekisteri.

IRP	RP1	RP0	TO	PD	Z	DC	C
bitti 7							bitti 0

-RP0 ja RP1-biteillä valitaan käytettävä muistipankki.
-Z-bitti on 1 silloin kun edellisen operaation tulos oli 0.
-C-bitti toimii Carry-bittinä, esim RLF-käskyssä siirretään rekisterin bittejä vasemmalle ja yli menevä bitti siirretään C:hen.
/8/

Taulukko 3 OPTION_REG-rekisteri.

RABPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bitti 7							bitti 0

-T0CS:ista valitaan Timer0:n kellolähde, 1 = ulkoinen, 0 = sisäinen.

-PSA määrittää, käytetäänkö prescaleria eli jakajaa Watchdog Timer:issa (1) vai Timer0:ssa (0).

-PS2-, PS1- ja PS0-biteillä valitaan jakaja.

/8/

Taulukko 4 INTCON-rekisteri.

GIE	PEIE	T0IE	INTE	RABIE	T0IF	INTF	RABIF
bitti 7							bitti 0

-GIE määrittää, sallitaanko erikseen kaikki sallitut keskeytykset (1 = sallitaan, 0 = ei sallita).

-T0IE määrittää, sallitaanko Timer0-keskeytykset (1 = sallitaan, 0 = ei sallita).

-RABIE määrittää, sallitaanko PORTA/B muutoksesta johtuvat keskeytykset (1 = sallitaan, 0 = ei sallita).

-T0IF ilmoittaa, onko Timer0-keskeytys tapahtunut (1= keskeytys tapahtunut, 0 = ei keskeytystä, nollattava ohjelmallisesti).

-RABIF ilmoittaa, onko PORTA/B muutoksesta johtuva keskeytys tapahtunut (1 = keskeytys tapahtunut, 0 = ei keskeytystä, nollattava ohjelmallisesti).

/8/

Muita tärkeitä rekisterejä ovat mm. seuraavat: PORTA (6 bit), PORTB (4 bit) ja PORTC (8 bit) rekisterit ovat kaksisuuntaisia IO-rekistereitä. Niiden suunta määrätään bitti kerrallaan vastaavilla TRISA-, TRISB- ja TRISC-rekistereillä (0 = ulostulo, 1 = sisäänmeno). OSCCON-rekisterillä säädetään PIC:n sisäistä oskillaattoria. Oletusarvoisesti se on 4 MHz. IOCA- ja IOCB-rekistereissä sallitaan bitti kerrallaan PORTA- ja PORTB-muutoksesta johtuvat keskeytykset (1 = sallitaan keskeytykset, 0 = ei sallita). TMR0-rekisteri kasvattaa itseään jokaisella

käskysyklillä (jos jakaja on 0) tai jakajan määrittämällä nopeudella. Rekisteri pyörii myös silloin, kun Timer0-keskeytykset eivät ole sallittuja.
/8/

2.6 PICkit 2 Starter Kit

Flash-muistinsa ansiosta PIC:it on helppo ohjelmoida tietokoneella. PIC:n ohjelmointiin käytettiin Microchipin valmistamaa USB-liitäntäistä PICkit 2 -ohjelmoijaa. Halvimman Starter Kitin mukana tulee pieni demoalusta sekä sille tehty 12 esimerkkiohjelmaa käsittävä kurssi. Demoalusta (Low Pin Count Demo Board) sisältää liittimen PICmicrolle, 4 lediä, potentiometrin ja painonapin sekä vapaata tilaa kytkentöjen tekemiseen. PIC voidaan ohjelmoida PICkitin mukana tulevalla ohjelmalla (PICkit Programmer) valmiiksi käännetyllä *.hex-tiedostolla, tai suoraan MPLAB-ohjelmistolla.

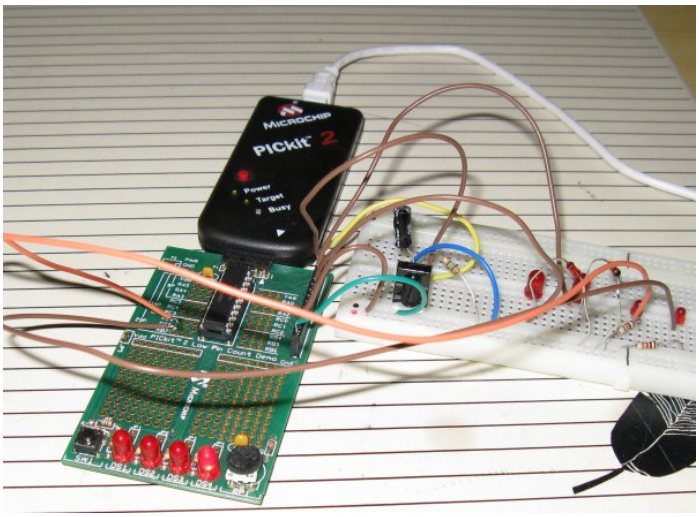
PICkit 2:lla voidaan ohjelmoida useita eri Microchipin flash-muistilla varustettuja tuotteita, esimerkiksi 8- 14- ja 20-pinnisiä PICmicroja ja sarja-EEPROM-muisteja.

3 TOTEUTUS

3.1 Kytkentä

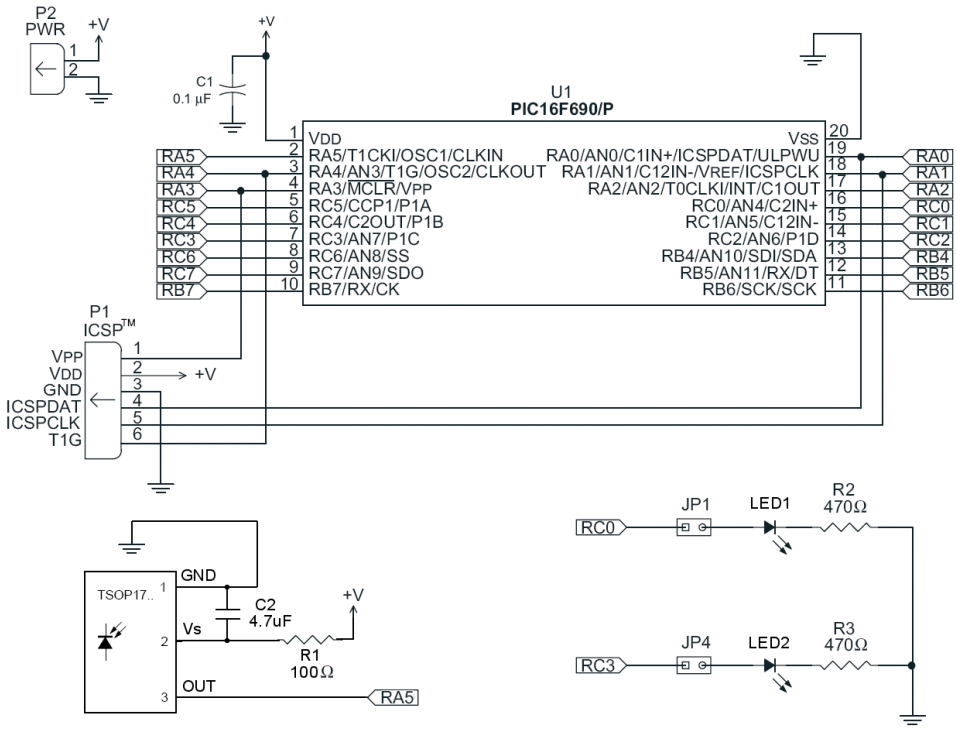
Kytkentä rakennettiin PICkit 2 Starter Kit -demoalustan pohjalta, koska piiri voidaan siinä helposti ohjelmoida ilman PIC:in irroittamista kytkennästä. Demoalustassa on myös riittävästi liittimiä, joten kytkentöjen tekeminen erilliseen koekytkentäalustaan ja niiden muuttaminen on vaivatonta.

Ohessa on kuva testikytkennästä (kuva 8). Kytkentä toimii 5 V:n käyttöjännitteellä, joka otetaan suoraan USB-väylästä.



Kuva 8 Testikytkentä.

Kuvassa 9 on esitetty kytkentäkaavio. Kytkentä sisältää PIC:in, IR-signaalin vastaanottopiirin sekä kaksi lediä. Demoduloitu IR-signaali on kytketty PIC:n RA5-pinniin, ledi 1 RC0- ja ledi 2 RC3-pinniin.



Kuva 9 Kytkentäkaavio. /osittain 4,7/

Käytännössä kytkentä toimii niin, että ledi 1 vaihtaa tilaansa aina sen merkiksi, että on vastaanotettu oikea neljän numeronäppäimen painalluksen sarja kaukosäätimeltä. Ledi 1:n tilalla voitaisiin käyttää kytkentää, joka ohjaisi vaikkapa sähköistä lukkoa. Jos on vastaanotettu virheellinen IR-koodi, sytytetään ledi 2 hetkeksi.

3.2 Kehitysympäristö

Ohjelmointi ja testaus tehtiin MPLAB-nimisellä ohjelmistolla. MPLAB on Microchipin tarjoama ilmainen integroitu kehitysympäristö (IDE) Microchipin tuotteille. Se tarjoaa mm. koodieditorin, assembler-kääntäjän sekä kattavat simulointimahdollisuudet. MPLAB on myös saumattomasti integroitu PICkit 2:n kanssa, joten kääntämisen jälkeen koodi voidaan helposti ja nopeasti ohjelmoida piirille suoraan MPLAB-ohjelmasta.

Vaikka MPLAB:in simulointiominaisuudet ovat hyvät, käytännössä helpoimmaksi tavaksi löytää virheet ohjelmassa osoittautui kuitenkin ohjelman pilkkominen pieniksi osiksi, jolloin niitä voitiin testata erikseen suoraan piirille ohjelmoimalla.

3.3 WinLIRC

Suunnitteluvaiheessa oli tarpeen varmistaa, että käytettävissä olevan Philipsin television kaukosäädin todella tuottaisi RC5-muotoista signaalia, ja myös tutkia, kuinka hyvin todellinen signaali vastaisi teoriaa. Lisäksi oli varmistettava, että IR-demodulaattori toimisi oikein. Vaikka digitaalinen oskilloskooppi olisi paras väline demoduloidun IR-signaalin tarkasteluun, kotioloihin löytyi myös muita tapoja tämän tekemiseen. Oskilloskoopin korvikkeena käytettiin WinLIRC-nimistä Windows-ohjelmaa. WinLIRC ottaa vastaan ja lähettää IR-komentoja tietokoneen sarjaporttiin liitetyn yksinkertaisen kytkennän kautta. Ohjelman ominaisuuksiin kuuluu useiden eri kaukosäädinten koodien oppimistoiminto, ja ohjelma mahdollistaakin sitä tukevien muiden Windows-ohjelmien ohjauksen minkä tahansa kaukosäätimen avulla. Tämän oppimistoiminnon avulla voitiin myös

selvittää vastaanotetun IR-signaalin muoto. Seuraavassa on ote oppimistoiminnon tekemästä konfiguraatiotiedostosta. Siitä nähdään numeronäppäimen "1" painalluksesta mitatut tiedot. Numerot vastaavat vuorotellen loogisten 1:ten ja 0:ien kestoja mikrosekunteina ("Active-high") Ensimmäinen luku tarkoittaa siis sitä, että signaali oli ylhäällä 950 µs, alhaalla 799 µs ja taas ylhäällä 1845 µs jne.

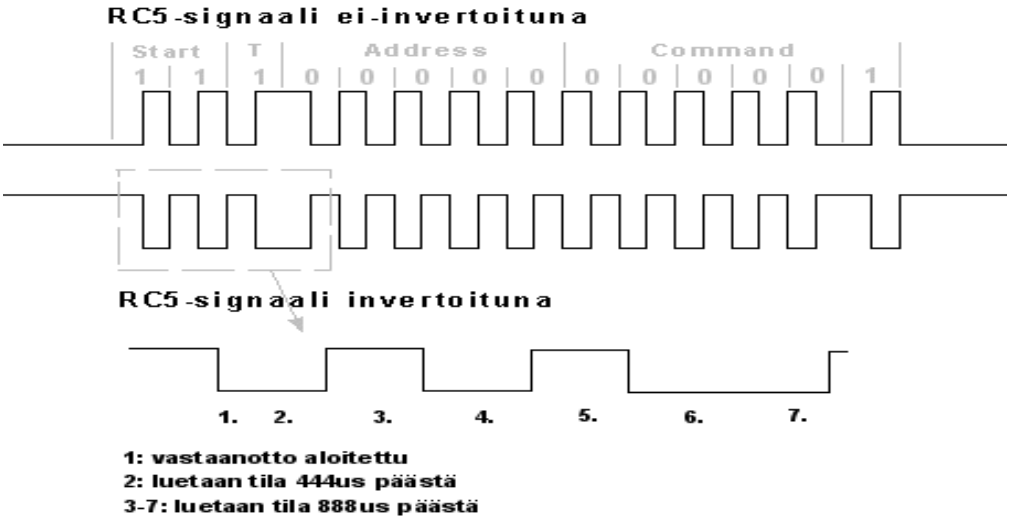
950	799	1845	797	959	800
962	792	961	803	946	798
961	802	960	800	962	796
960	800	960	802	953	1686
958					

Muuttamalla tämä graafiseen muotoon ja siitä biteiksi voidaan todeta, että koodi todella vastaa taulukon 1 mukaista numeronäppäimen "1" koodia. Mittauksesta nähdään myös, että vaikka yksittäisen bitin kesto pysyy vakiona, signaalin aika ylhäällä ei ole saman pituinen kuin sen aika alhaalla. Signaali on ylhäällä n. 960 µs ja alhaalla n. 800 µs. Seuraavaksi käsiteltävän ohjelman toimintaan sillä ei kuitenkaan ole merkitystä.

4 OHJELMAN TOIMINTA

4.1 Yleiskuva toiminnasta

Kuva 10 selittää lyhyesti ohjelman toimintaperiaatteen. Kuten aiemmin tuli esille, RC5-koodin bitit on muodostettu Manchester-koodauksella, jossa "1" bittiä vastaa nousu 0→1 ja "0" bittiä lasku 1→0. Tässä työssä käytetään näistä muutoksista nimitystä bittipari (01 ja 10).



Kuva 10 Ohjelman toimintaperiaate.

Yhden bitin (eli bittiparin) molemmat puoliskot kestävät noin 888 µs. Jotta jokainen tila tulisi mahdollisimman luotettavasti luettua, on parasta lukea aina tilan keskikohdasta. On myös muistettava, että IR-demodulaattori invertoi signaalin.

Alkutilanteessa odotetaan RA5-pinnin putoamista alas. Tämä viestittää ohjelmalle, että RC5-koodin vastaanotto on alkanut. Koska RC5-koodin ensimmäinen bittipari on aina "01" eli invertoituna "10", RA5-pinnin pudotessa alas puolet bittiparista on jo mennyt (888 µs). Jälkimmäisen osan tila on luettava sen puolesta välistä eli 444 µs:n kuluttua. Jos tila on 1, tiedetään että vastaanotettiin virheellinen koodi, ja voidaan palata alkutilanteeseen odottamaan uuden koodin alkua. Jos tila on 0, kyseessä on laillinen RC5-koodi, ja ensimmäinen bitti on vastaanotettu ja voidaan tallettaa. Seuraavaksi luetaan bittiparien tilat aina 888 µs:n välein ja tarkistetaan

niiden laillisuus. Parit ovat laillisia, jos ne ovat aina toistensa vastakohtia (01 tai 10). Jos näin ei ole, on saatu virheellinen koodi ja palataan taas alkutilanteeseen. Muuten talletetaan saadut bitit (ne päätellään bittiparista) kunnes on saatu 14 bittiä eli kokonainen koodi. Kokonainen koodi talletetaan sen jälkeen VastaanotetutKoodit-listaan, jossa on aina 4 viimeisintä koodia. Lopuksi näitä verrataan haluttuun koodisarjaan, joka sijaitsee VaaditutKoodit-listassa, ja vastaavuuden sattuessa vaihdetaan ledin 1 tilaa. Sen jälkeen palataan taas alkutilaan odottamaan uutta koodia. Ohjelman toimintaa kuvaava vuokaavio löytyy liitteestä 1 ja lähdekoodi on kokonaisuudessaan liitteessä 2.

Seuraavissa luvuissa käydään lähdekoodi läpi yksityiskohtaisemmin vaihe vaiheelta. Selkeyden vuoksi väliotsikot on jaoteltu samalla lailla kuin lähdekoodiin moduulit.

4.2 Alkutoimenpiteet

Alussa määritellään prosessorin konfiguraatioasetukset ja varataan muuttujille tilaa. Ohjelma käyttää keskeytyksiä. Sen vuoksi tärkeistä rekistereistä, jotka saattavat muuttua keskeytyspalvelun sisällä, on tehtävä kopiot. Tässä ohjelmassa tehdään kopiot W- ja STATUS-rekistereistä. Näille on varattu tilaa yhteiseltä muistialueelta 70h - 7Fh, jolloin niihin pääsee suoraan käsiksi mistä tahansa muistipankista ilman STATUS-rekisterin RP0- ja RP1-bittien asettamista.

4.3 ISR

Seuraavaksi määritellään org 0 -käskyllä ohjelmakoodin alkamisosoitteeksi 0h, josta PIC aloittaa koodin suorittamisen resetin jälkeen. Keskeytyksen sattuessa hypätään aina kohtaan 0004h, joten keskeytyspalvelu (Interrupt Service Routine, ISR) on sijoitettava tähän kohtaan. (Normaalisti ohjelman suorituksen alkaessa keskeytyspalvelun yli hypätään goto Start -käskyllä.) Tässä ohjelmassa on käytössä 2 eri keskeytyslähdettä: Timer0-ajastimen ylivuodosta aiheutuva T0IF-lippu ja PORTA bitin 5 muutoksesta aiheutuva RABIF-lippu (jotka molemmat sijaitsevat

INTCON-rekisterissä). Tässä ohjelmassa keskeytyspalvelun tehtävänä on selvittää, mistä lähteestä keskeytys tuli ja viestittää sitten pääohjelmalle, että keskeytys on tapahtunut asettamalla PortALippu tai Timer0Lippu. Lisäksi keskeytyspalvelussa on muistettava nollata keskeytyksen aiheuttanut RABIF- tai T0IF-lippu. Keskeytyspalvelun ensimmäisinä käskyinä talletetaan W- ja STATUS-rekisterit apumuistipaikkoihin. Nämä palautetaan viimeiseksi palvelusta poistuttaessa.

4.4 Start

Start-kohdasta alkaen suoritetaan alustustoimenpiteitä. Ensiksi käytetyt muistipaikat alustetaan, ja ANSEL- ja ANSELH-rekisterit nollataan eli käytetään digitaalisia sisäänmenoja. Koska IR-signaali tulee PIC:in RA5-pinniin, PORTA-rekisteri konfiguroidaan TRISA-rekisterillä sisäänmenoksi. Ledit on yhdistetty RC0- ja RC3-pinneihin, joten PORTC on asetettava ulostuloksi TRISC-rekisterillä. Lisäksi vielä talletetaan neljään peräkkäiseen muistipaikkaan tunnistettava koodisarja. Nämä kohdat suoritetaan vain kerran ohjelman suorituksen alussa.

4.5 MainLoop

MainLoop-kohdasta alkaa pääohjelman suoritus. Tästä kohdasta alkaa aina uuden kaukosäädinkoodin vastaanottaminen. Ensiksi alustetaan käytettävät laskurit. Koska RC5-koodissa on aina 14-bittiä, asetetaan bittilaskurin sisällöksi 14. Ohjelmassa kokonainen vastaanotettu RC5-koodi talletetaan koodi_high- ja koodi_low-muistipaikkoihin siten, että koodi_high sisältää 6 eniten merkitsevää bittiä ja koodi_low 8 vähiten merkitsevää bittiä. Siksi asetetaan myös koodi_high_laskuri arvoon 6. Tästä päätellään, kumpaan muistipaikkaan saadut bitit tallennetaan.

Tässä vaiheessa on myös syytä antaa IR-demodulaattorille n. 200 ms aikaa nostaa ulostulonsa ylös, jotta vältyttäisiin virhetilanteilta. 200 ms saadaan aikaan yksinkertaisella viiveloopilla kutsumalla viive200ms-aliohjelmaa. Viive myös poistaa tahattomat koodin kahteen kertaan vastaanotot, koska käytetty kaukosäädin

helposti toistaa koodin kerran yhdellä napinpainalluksella. IR-demodulaattori on invertoiva, joten kun IR-signaalia ei vastaanoteta, sen ulostulo on korkeassa tilassa. Jotta voitaisiin todeta, milloin signaali putoaa alas ensimmäisen kerran, voidaan käyttää Portin A muutoskeskeytystä. PIC päättelee, onko portissa tapahtunut muutos vertaamalla portin tilaa sen lukkopiirissä olevaan tilaan. Siksi ensimmäiseksi on luettava kertaalleen Port A bitin 5 (RA5) tila, jotta se jää lukkopiiriin. Tämän jälkeen asetetaan INTCON-rekisteristä RABIE-bitti, jolla sallitaan keskeytykset tästä lähteestä. Myös yleinen keskeytysten sallimisbitti GIE asetetaan. Lisäksi on vielä erikseen sallittava Portin A bitin 5 keskeytykset IOCA-rekisteristä.

4.6 OdotaEkaa

Seuraavaksi jäädään odottamaan ensimmäistä muutosta RA5-tilassa. Muutoksen tapahtuessa hypätään keskeytyspalveluun ISR (kuvaus kohdassa 4.2), jossa PortALippu asetetaan. Kun pääohjelmassa on havaittu, että lippu on asetettu, se nollataan ja Timer0 asetetaan keskeyttämään 444 µs:n kuluttua. TMR0-rekisteri on 8-bittinen ja inkrementoituu 1 µs välein. Kun OPTION_REG-rekisteristä asetetaan jakajaksi (Prescaler) 4 ja TMR0:n sisällöksi 145, saadaan keskeytys aina 444 µs:n välein (444 µs = 111 µs x 4, 256 - 111 = 145). Lopuksi vielä sallitaan Timer0:n ylivuodosta aiheutuvat keskeytykset asettamalla INTCON-rekisterin bitti T0IE, ja estetään Portin A muutoskeskeytykset koodin luvun ajaksi nollaamalla bitti RABIE.

4.7 LueEka

Nyt odotetaan Timer0-keskeytystä. Keskeytyksen tapahtuessa keskeytyspalvelu viestittää pääohjelmaa asettamalla Timer0Lipun. Kun pääohjelmassa on havaittu, että lippu on asetettu, se nollataan ja Timer0 asetetaan nyt keskeyttämään 880 µs:n kuluttua (ajastin asetetaan 880 µs:iin 888 µs:n sijasta, koska ajastimen uudelleen virittämiseen kuluu aina ylimääräistä aikaa). 880 µs saadaan asettamalla jakaksi 8 ja TMR0:n sisällöksi 146 (880 µs = 110 µs x 8, 256 - 110 = 146). Seuraavaksi

luetaan ensimmäisen bittiparin jälkimmäinen osa RA5-pinnistä. Jos tila on 1, ollaan saatu virheellinen koodi, ja mennään Virhe-tilaan (kohta 4.11). Muuten talletetaan ensimmäinen saatu bitti, eli 1, koodi_high-muistipaikkaan. Tänne talletetaan 6 eniten merkitsevää bittiä siten, että muistipaikan bittejä siirretään aina oikealle ennen uuden bitin tallettamista muistipaikan vähiten merkitsevälle paikalle. Samalla vähennetään koodi_high_laskuria. Lopuksi vähennetään vielä bittilaskuria, koska ensimmäinen bitti on menestyksellisesti vastaanotettu.

4.8 Lue1, Lue2

Nyt luetaan jäljelle jääneet 13 bittiä bittipari kerrallaan. Odotetaan jälleen ensin Timer0-keskeytystä tarkistamalla toistuvasti Timer0Lippua. Kun lippu on asetettu, se nollataan ja Timer0 viritetään jälleen keskeyttämään 880 µs:n kuluttua. Sen jälkeen luetaan bittiparin ensimmäinen puolisko RA5-pinnistä bitti1-muistipaikkaan. Seuraavaksi odotetaan taas Timer0-keskeytystä, ja sen tapahduttua nollataan jälleen lippu ja viritetään Timer0 keskeyttämään 880 µs:n kuluttua. Sitten luetaan bittiparin jälkimmäinen puolisko bitti2-muistipaikkaan.

Nyt on tarkistettava, onko kyseessä laillinen pari eli ovatko bitti1 ja bitti2 toistensa vastakohtia. Tämä tehdään suorittamalla niiden välillä XOR-operaatio. XOR:n tulos on 1 ainoastaan silloin, kun bitti1 ja bitti2 eivät ole samat. Jos kyseessä ei ole laillinen pari, hypätään Virhe-tilaan. Muuten talletetaan saatu bitti kutsumalla TalletaKoodi-aliohjelmaa (kohta 4.9). Lopuksi nollataan bitti1-, bitti2- ja OnkoLaillinenPari-muistipaikat ja vähennetään Bittilaskuria. Jos Bittilaskuri on 0, ollaan saatu kokonainen RC5-koodi ja hypätään KoodinTarkistus-kohtaan (kohta 4.10). Muuten hypätään takaisin Lue1-kohtaan aloittamaan seuraavan bittiparin lukua.

4.9 TalletaKoodi

Tässä aliohjelmassa talletetaan bittiparin muistipaikkojen bitti1 ja bitti2 tarkoittama RC5-bitti koodi_high- tai koodi_low-muistipaikkaan. Koska kyseessä on invertoitu RC5-koodi, bitti1 vastaa suoraan RC5-bittiä.

Ensiksi tarkistetaan, onko koodi_high_laskuri mennyt nolliin. Jos se ei ole nolla, vähennetään sitä yhdellä ja talletetaan bitti1 siirtämällä ensin koodi_high-muistipaikan bittejä vasemmalle ja asettamalla sitten vähiten merkitsevä bitti bitti1-muistipaikan mukaisesti.

Jos koodi_high_laskuri on nolla, ollaan saatu jo tallennettua ensimmäiset 6 eniten merkitsevää bittiä koodi_high-muistipaikkaan, ja voidaan alkaa täyttämään koodi_low-muistipaikkaa 8:lla vähiten merkitsevällä bitillä. Niiden täyttäminen tapahtuu samaan tapaan kuin edellä eli siirtämällä muistipaikan bittejä vasemmalle, ja asettamalla vähiten merkitsevä bitti bitti1-muistipaikan mukaisesti. Muita tarkistuslaskureita ei tarvita, koska tätä aliohjelmaa kutsutaan juuri niin monta kertaa, että kaikki bitit saadaan tallennettua.

4.10 KoodinTarkistus

Tässä kohtaa tarkistetaan vastaako viimeisen 4 tallennetun koodin sarja vaadittua sarjaa. Tänne tullaan aina, kun on vastaanotettu kokonainen laillinen RC5-koodi. Koodi on tallennettu koodi_high- ja koodi_low-muistipaikkoihin mutta tässä ohjelmassa tarkistamiseen käytetään vain koodi_low-muistipaikassa olevaa 8:aa alinta bittiä. Tämä siksi, että 8:aan alimpaan bittiin mahtuvat kaikki komentobitit. Ylimpiä laiteosoite-bittejä ei huomioida, koska ne ovat yhdessä kaukosäätimessä aina samat.

Ensiksi estetään Timer0-keskeytykset nollaamalla INTCON-rekisterin bitti T0IE. Sitten saatu koodi tallennetaan 4 tavun mittaiseen VastaanotetutKoodi-listaan siten, että viimeisin koodi on aina ylimmäisenä. Seuraavaksi tarkistetaan tavu kerrallaan, vastaavatko ne VaaditutKoodit-listan koodeja. Jos vastaavuutta ei löydy, nollataan

koodi_high- ja koodi_low-muistipaikat ja palataan pääohjelman alkuun MainLoop-kohtaan odottamaan seuraavan koodin vastaanottoa. Jos vastaavuus löytyy, nollataan koodi_high- ja koodi_low-muistipaikat, ja vaihdetaan ledin 1 tilaa XOR-komentoa apuna käyttäen. Ledi 1 on liitetty RC0-pinniin, eli PORTC bittiin 0. Tämän jälkeen palataan MainLoopin alkuun.

4.11 Virhe

Jos jossain vaiheessa koodin vastaanottoa löydetään virhe, hypätään tänne. Ensin nollataan muistipaikat koodi_high, koodi_low, bitti1, bitti2 ja OnkoLaillinenPari. Sitten Timer0-keskeytykset estetään nollaamalla INTCON-rekisterin T0IE-bitti, ja virhetilasta ilmoitetaan käyttäjälle sytyttämällä ledi 2 noin 0,4 sekunniksi. Ledi 2 on kytketty RC3-pinniin eli PORTC bittiin 3. Lopuksi palataan MainLoopin alkuun seuraavaa koodia odottamaan. Virhetilan voi aiheuttaa esimerkiksi jonkun toisen kaukosäätimen signaali tai loisteputkilampusta tulevat IR-häiriöt.

5 JATKOKEHITYS

Työtä kirjoittaessa tuli esille useita eri jatkokehitysmahdollisuuksia. Esimerkiksi nyt kaukosäätimen numeronäppäimen pitäminen pohjassa liian pitkään saa laitteen luulemaan, että nappia on painettu toistuvasti, jolloin laite ei tunnista oikeaa neljän koodin sarjaa. Tämä voitaisiin korjata tarkistamalla toggle-bitin tila kahden peräkkäisin vastaanotetun koodin välillä. Jos toggle-bitti pysyy samana, voidaan todeta, että nappia on painettu vain yhden kerran.

Toinen jatkokehitysmahdollisuus liittyy siihen, että tällä hetkellä uuden tunnistettavan koodisarjan muuttaminen vaatii PIC:n uudelleenohjelmoimista. Ohjelmaa voitaisiin muuttaa siten, että sarjaa pystyisi muuttamaan myös laitteen ollessa päällä; esim. demoalustan nappia painamalla mentäisiin opetustilaan jossa uusi sarja vastaanotettaisiin. Sarja voitaisiin tallettaa PIC:n EEPROM-muistiin, jolloin se säilyisi muistissa myös laitteen ollessa sammutettuna.

Tällä hetkellä laite vaatii ulkoisen 5 V:n virtalähteen. Jos laitetta haluttaisiin käyttää paristoilla, olisi tehtävä joitakin muutoksia. 5 V:n jännitteen saamiseksi tarvittaisiin vähintään neljä 1,5 V:n paristoa ja 5 V:n regulaattori, jotka veisivät paljon tilaa. Niiden sijaan voitaisiin käyttää ns. Charge Pump -piiriä, joka pystyy nostamaan jännitettä. Tällöin selvittäisiin vain kahdella 1,5 V:n paristolla. Virrankulutuksen minimoimiseksi ohjelman toimintaa voisi myös muuttaa niin, että koko koodin vastaanotto tapahtuisi keskeytyspalvelussa, ja pääohjelmassa PIC ohjattaisiin virransäästötilaan SLEEP-käskyllä.

Koska laite tunnistaa ja muuttaa biteiksi kaikki RC5-muotoiset käskyt, sitä voitaisiin muuttaa siten, että pystyttäisiin myös lähettämään RC5-käskyjä. PIC:in nopeus riittäisi hyvin ohjelmistopohjaiseen modulointiin 36 kHz:llä, ja siinä on myös erillinen PWM-moduuli (Pulse Width Modulation), jolloin modulaatio voitaisiin tehdä laitteistopohjaisena. Näin voitaisiin kommunikoida langattomasti esim. kahden eri PIC:n välillä. Laite voitaisiin muuttaa tunnistamaan ja lähettämään myös muiden protokollien mukaisia koodeja, jolloin PIC voisi ohjata montaa laitetta samalla kertaa. Esim. yhdellä napin painalluksella voitaisiin käynnistää televisio, DVD-soitin ja vahvistin.

6 LOPPUPÄÄTELMIÄ

Työ auttoi ymmärtämään paremmin etenkin assembly-ohjelmointia ja siinä käytettäviä toimintatapoja, ja myöskin toimivan laitteen toteutukseen liittyviä piirteitä. PICmicro osoittautuikin erittäin hyväksi aloittelijalle sopivaksi mikrokontrolleri-perheeksi. PIC-ohjelmoinnin opettelu madaltaa myös kynnystä muiden monimutkaisempien mikrokontrollerien ohjelmointiin tutustumiseen.

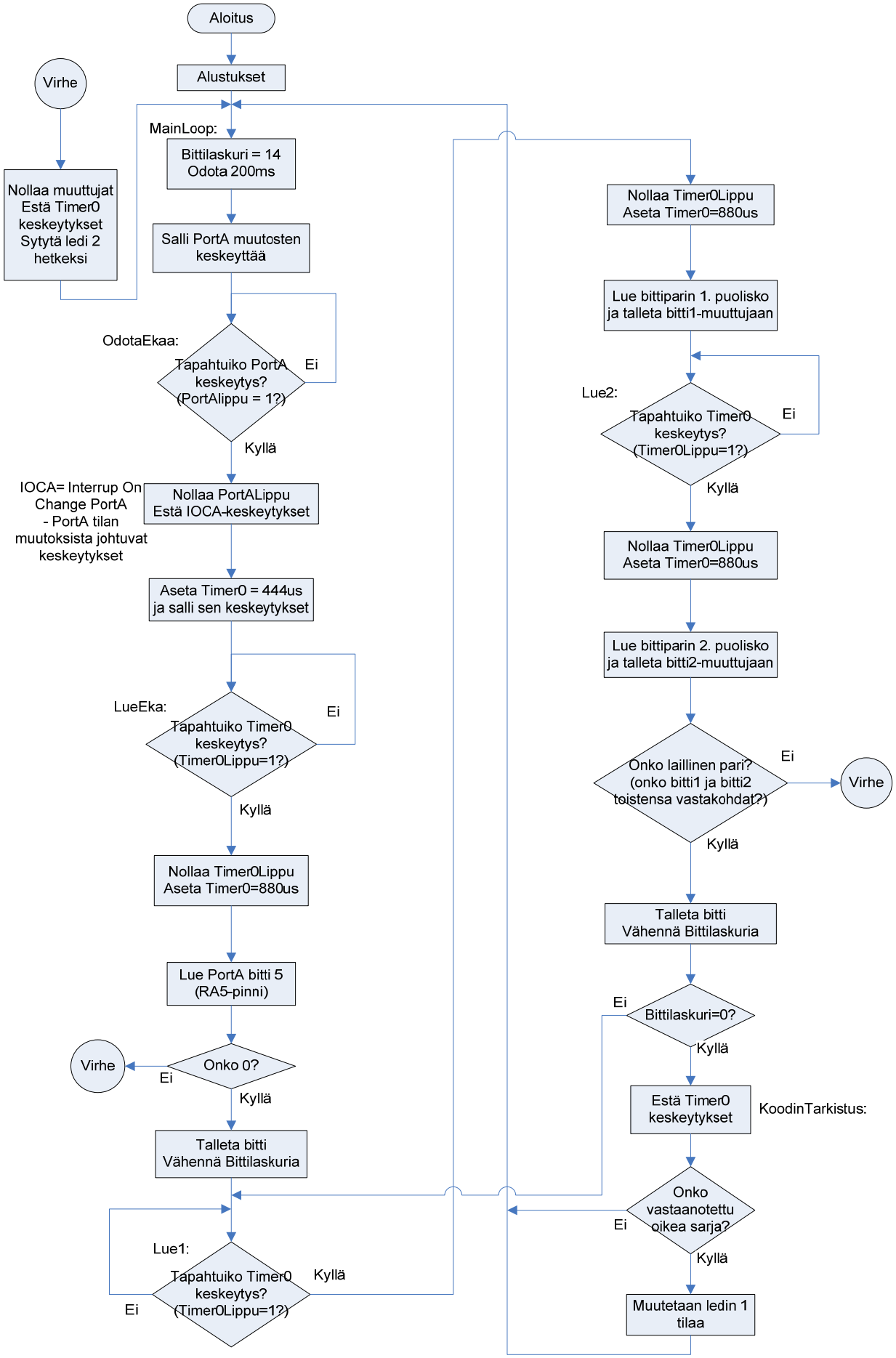
Aikaa työn tekemiseen kului yhteensä noin 24 päivää, joista suunnitteluun ja PIC-ohjelmointiin tutustumiseen meni suurin osa ajasta, n. 13 päivää. Itse ohjelmointiin kului n. 7 päivää ja työn kirjoittamiseen n. 4 päivää.

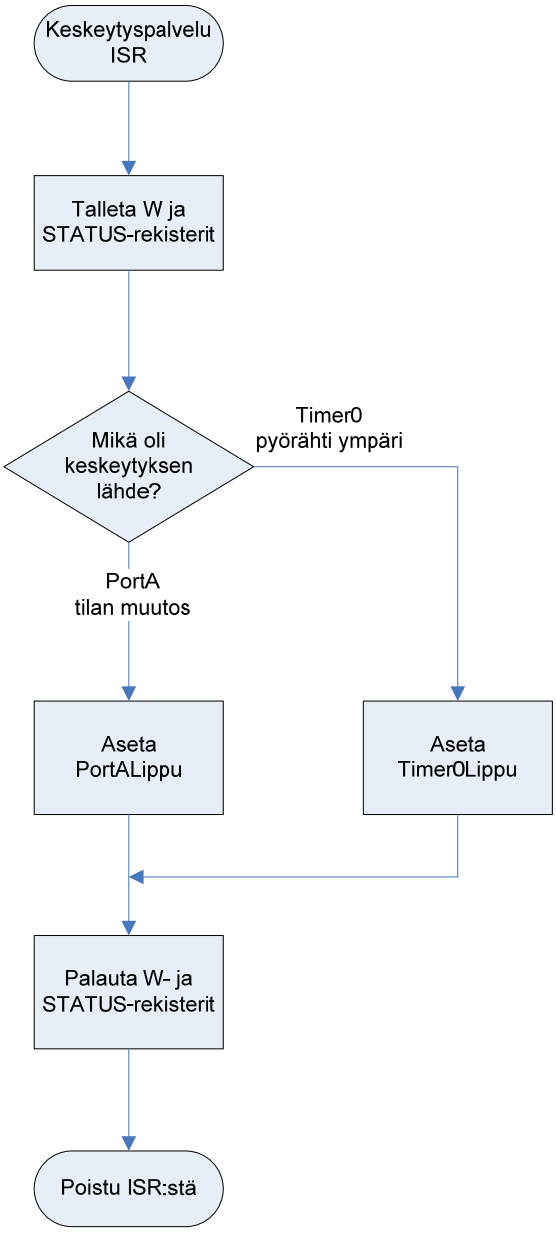
LÄHDELUETTELO

1. San Bergmans, SB-Projects: IR remote control. [www-sivu]. [viitattu 4.11.2008] Saatavissa: http://www.sbprojects.com/knowledge/ir/ir.htm
2. San Bergmans, SB-Projects: IR remote control: Philips RC-5. [www-sivu]. [viitattu 4.11.2008] Saatavissa: http://www.sbprojects.com/knowledge/ir/rc5.htm
3. Infrared. [www-sivu]. [viitattu 6.11.2008] Saatavissa http://www.ustr.net/infrared/infrared1.shtml
4. Vishay, TSOP1738 datakirja [sähköinen dokumentti]. [viitattu 6.11.2008] Saatavissa: http://www.datasheetcatalog.org/datasheets/208/301092_DS.pdf
5. Gooligum Electronics, Introduction to PIC Microcontrollers. [sähköinen dokumentti]. [viitattu 6.11.2008] Saatavissa: http://www.gooligum.com.au/tutorials/PIC_Intro_0.pdf
6. mikroElektronika, Introduction: World of microcontrollers. [www-sivu]. [viitattu 30.4.2008] Saatavissa: http://www.mikroe.com/en/books/picmcubook/ch0/
7. Microchip, Low Pin Count User Guide [sähköinen dokumentti]. [viitattu 30.4.2008] Saatavissa: http://www.microchip.com/Microchip.WWW.SecureSoftwareList/secsoftwaredownload.aspx?device=en023805&lang=en&ReturnURL=http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en023805#
8. Microchip, PIC16F690 datakirja. [sähköinen dokumentti]. [viitattu 29.4.2008] Saatavissa: http://ww1.microchip.com/downloads/en/DeviceDoc/41262E.pdf

LIITTEET

1. Vuokaavio (2 sivua)
2. Ohjelmalistaus (5 sivua)
3. PIC16F690 käskykanta (9 sivua)


Aloitus
Alustukset
Virhe
Nollaa muuttujat
Estä Timer0 keskeytykset
Sytytä ledi 2 hetkeksi
MainLoop:
Bittilaskuri = 14
Odota 200ms
Salli PortA muutosten keskeyttää
OdotaEkaa:
Tapahtuiko PortA keskeytys? (PortAlippu = 1?)
Ei
Kyllä
IOCA= Interrup On Change PortA - PortA tilan muutoksista johtuvat keskeytykset
Nollaa PortALippu
Estä IOCA-keskeytykset
Aseta Timer0 = 444us
ja salli sen keskeytykset
LueEka:
Tapahtuiko Timer0 keskeytys? (Timer0Lippu=1?)
Ei
Kyllä
Nollaa Timer0Lippu
Aseta Timer0=880us
Lue PortA bitti 5
(RA5-pinni)
Onko 0?
Virhe
Ei
Kyllä
Talleta bitti
Vähennä Bittilaskuria
Lue1:
Tapahtuiko Timer0 keskeytys? (Timer0Lippu=1?)
Kyllä
Ei
Nollaa Timer0Lippu
Aseta Timer0=880us
Lue bittiparin 1. puolisko
ja talleta bitti1-muuttujaan
Lue2:
Tapahtuiko Timer0 keskeytys? (Timer0Lippu=1?)
Ei
Kyllä
Nollaa Timer0Lippu
Aseta Timer0=880us
Lue bittiparin 2. puolisko
ja talleta bitti2-muuttujaan
Onko laillinen pari? (onko bitti1 ja bitti2 toistensa vastakohdat?)
Ei
Virhe
Kyllä
Talleta bitti
Vähennä Bittilaskuria
Bittilaskuri=0?
Ei
Kyllä
Estä Timer0 keskeytykset
KoodinTarkistus:
Onko vastaanotettu oikea sarja?
Ei
Kyllä
Muutetaan ledin 1 tilaa


Keskeytyspalvelu ISR
Talleta W ja STATUS-rekisterit
Mikä oli keskeytyksen lähde?
Timer0 pyörähti ympäri
PortA tilan muutos
Aseta PortALippu
Aseta Timer0Lippu
Palauta W- ja STATUS-rekisterit
Poistu ISR:stä

```
;*************************************************************************************************
; KAUKO-OHJATTAVA KOODILUKKO
;
; (c) Kari-Matti Mäkelä
;
; TAMPEREEN AMMATTIKORKEAKOULU
; Tietotekniikan koulutusohjelma
; Tietokonetekniikka
;
; Tampere 2008
;
;
; PICmicro: PIC16F690
;
; Ohjelma vastaanottaa IR-demodulaattorilta TTL-tasoista IR-signaalia PIC:in RA5-pinniin ja
; tunnistaa siitä RC5-kaukosäädinprotokollan mukaiset koodit (esim. Philipsin TV:n kaukosäätimen
; näppäinten painallukset). Jos vastaanotetaan virheellinen koodi (esim. häiriöpulssi tai jonkin
; muun IR-protokollan mukainen koodi), sytytetään hetkeksi ledi 4. Kun neljän peräkkäisen
; esiohjelmoidun koodin (tässä tapauksessa numeronäppäinten) sarja on tunnistettu, vaihdetaan ledin
; 1 tilaa (ledin tilalla voisi olla esimerkiksi rele, joka ohjaa lukkoa).
;
;*************************************************************************************************


#include <p16F690.inc>
     __config (_INTRC_OSC_NOCLKOUT & _WDT_OFF & _PWRTE_OFF & _MCLRE_OFF & _CP_OFF & _BOR_OFF &
_IESO_OFF & _FCMEN_OFF)

     cblock     0x20
viive1                          ; Muuttujat
viive2
bitti1
bitti2
koodi_high
koodi_low
koodi_high_laskuri
OnkoLaillinenPari
PortALippu
Timer0Lippu
viive144
Bittilaskuri
VastaanotetutKoodit:4           ; 4 tavun mittainen, tänne talletetaan 4 viimeisintä vast.otettua
                                ; laillista koodia.
VaaditutKoodit:4                ; Täällä tunnistettava 4 koodin sarja
     endc

     cblock 0x70                ; Käytettävissä mistä tahansa bankista.
W_Save
STATUS_Save
     endc

;-------------------------------------------------------------------------------------------------

     org 0
     goto      Start
     nop
     nop
     nop
ISR:                            ; Interrupt Service Routine, keskeytyspalvelu, alkaa kohdasta
                                ; 0004h. Tänne tullaan keskeytyksen tapahtuessa (jos sallittu)

     movwf     W_Save           ; Tärkeiden rekisterien talletus jottei keskeytyspalvelu pääse
     movf      STATUS,w         ; muuttamaan niitä.
     movwf     STATUS_Save
                                ; selvitetään mistä lähteestä keskeytys tuli:
     btfsc     INTCON,RABIF     ; kun RA5:n tila vaihtuu, RABIF-bitti --> "1" --> keskeytys
     goto      ServicePortA
     btfsc     INTCON,T0IF      ; kun TMR0 kiepsahtaa ympäri, T0IF-bitti --> "1" --> keskeytys
     goto      ServiceTimer0
     goto      ExitISR

ServicePortA:
     bcf       INTCON,RABIF     ; nollataan keskeytyslippu RABIF
     btfss     PORTA,5          ; luetaan RA5 tila, jotta se jää portin lukkopiiriin (latch):
     nop                        ; tähän pic vertaa seuraavaa tilan vaihtoa.
     bsf       PortALippu,0     ; viesti pääohjelmalle, että muutos tapahtunut RA5 pinnissä
     goto      ExitISR
```

```
ServiceTimer0:
     bcf       INTCON, T0IF        ; nollataan keskeytyslippu T0IF
     bsf       Timer0Lippu,0       ; viesti pääohjelmalle, että ajastinkeskeytys tapahtunut
     goto      ExitISR

ExitISR:                           ; poistutaan keskeytyspalvelusta
     movf      STATUS_Save,w       ; rekistereiden palautus
     movwf     STATUS
     swapf     W_Save,f            ; swapf-käskyä käytetään koska se ei vaikuta STATUS-rekisterin
     swapf     W_Save,w            ; lippuihin (Z,C,ym.).
     retfie

;-------------------------------------------------------------------------------------------------

Start:                             ; pääohjelma alkaa tästä
     banksel   ANSEL
     clrf      ANSEL               ; ANSEL ja ANSELH nollaus, jotta simulaattori toimii oikein!
     clrf      ANSELH              ; 0=digi, 1=ana
     banksel   TRISA

     movlw     0xFF
     movwf     TRISA               ; PortA sisäänmenoksi (1)
     clrf      TRISC               ; PortC ulostuloksi (0)
     banksel   PORTC
     clrf      PORTC               ; alustuksia
     clrf      viive1
     clrf      viive2
     clrf      PortALippu
     clrf      bitti1
     clrf      bitti2
     clrf      koodi_high
     clrf      koodi_low
     clrf      OnkoLaillinenPari
     clrf      Timer0Lippu

     ; tähän talletetaan tunnistettava koodisarja, nyt napit 1 2 3 4
     movlw     b'00000001'         ; nappi 1
     movwf     VaaditutKoodit
     movlw     b'00000010'         ; nappi 2
     movwf     VaaditutKoodit+1
     movlw     b'00000011'         ; nappi 3
     movwf     VaaditutKoodit+2
     movlw     b'00000100'         ; nappi 4
     movwf     VaaditutKoodit+3

     clrf      VastaanotetutKoodit
     clrf      VastaanotetutKoodit+1
     clrf      VastaanotetutKoodit+2
     clrf      VastaanotetutKoodit+3

     banksel   OPTION_REG
     movlw     b'00000001'         ; Timer0 asetukset: prescaler=4, sisäinen kello
     movwf     OPTION_REG
     banksel   PORTA

;-------------------------------------------------------------------------------------------------

MainLoop:
     movlw     D'6'
     movwf     koodi_high_laskuri  ; laskuri jota käytetään 6:n eniten merkitsevän bitin tallet.
     movlw     D'14'               ; RC5-koodissa 14 bittiä: asetetaan se Bittilaskurin alkuarvoksi
     movwf     Bittilaskuri

     call      viive200ms          ; odotellaan alussa 200ms jotta ir-demodulaattorin ulostulo
                                   ; kerkiää asettumaan. Koodin vastaanoton jälkeen myös odotetaan
                                   ; jotta saadaan tuplat pois.

     btfss     PORTA,5             ; luetaan RA5 tila, jotta se jää portin lukkopiiriin (latch).
     nop                           ; tähän pic vertaa seuraavaa tilan vaihtoa.
     clrf      INTCON
     bsf       INTCON, RABIE       ; Interrupt-On-Change (IOC) sallinta PortA/B
     banksel   IOCA
     bsf       IOCA,5              ; RA5-bitin sallitaan keskeyttää
     banksel   INTCON
     bsf       INTCON, GIE         ; lopuksi sallitaan globaalit keskeytykset



;-------------------------------------------------------------------------------------------------
```

```
OdotaEkaa:                          ; tässä odotetaan RC5-koodin ensimmäistä bittiä
     btfss     PortALippu,0         ; tapahtuiko muutos RA5? (Eli käytännössä putosiko RA5 alas?)
     goto      OdotaEkaa            ; ei, pyöritään tässä
     bcf       PortALippu,0         ; kyllä, nollataan lippu, asetetaan TMR0=444us ja sallitaan sen
                                    ; keskeytykset
     movlw     D'145'               ; 444 / 4 = 111. TMR0 pyörii ylöspäin joten se on asetettava
                                    ; 256-111=145.
     movwf     TMR0                 ; TMR0:aan kirjoittaminen nollaa prescalerin! Eli on asetettava
     banksel   OPTION_REG           ; uudestaan.
     movlw     b'00000001'          ; Timer0 asetukset: prescaler=4, sisäinen kello
     movwf     OPTION_REG
     banksel   INTCON
     bsf       INTCON, T0IE         ; sallitaan TMR0-keskeytykset

     bcf       INTCON, RABIE        ; estetään IOC-keskeytykset, sitä tarvitaan vain koodin
     banksel   IOCA                 ; alkamisajan määrittämiseen.
     clrf      IOCA
     banksel   PORTA

;-----------------------------------------------------------------------------------------------

LueEka:                             ; Ensimmäisen RC5-bittiparin toisen bitin luku 444us päästä siitä
                                    ; kun RA5 putoaa alas
     btfss     Timer0Lippu,0        ; pyöritään tässä kunnes 444us on tullut täyteen
     goto      LueEka
     bcf       Timer0Lippu,0        ; nollataan lippu

     movlw     D'146'               ; Viritetään jo kello odottamaan 888us (880us kompensoimaan
                                    ; muista käskyistä aiheutuvia viiveitä), asetus
     movwf     TMR0                 ; mahdollisimman lähellä edellistä TMR0 keskeytystä.
     banksel   OPTION_REG           ; 880 / 8 = 110, 256-110=146
     movlw     b'00000010'          ; Timer0 asetukset: prescaler=8, sisäinen kello
     movwf     OPTION_REG
     banksel   INTCON


                                    ; Nyt luetaan 1. bitti (oikeasti 1. bittiparin jälkimmäinen).
     btfsc     PORTA,5              ; Jos on 0, ensimmäinen pari on laillinen ja vastaa RC5-
                                    ; protokollassa 1:stä.
     goto      Virhe                ; Ei laillinen RC5-koodi, mennään Virhe-tilaan

     decf      koodi_high_laskuri,f   ; alussa 6.
     rlf       koodi_high,f         ; tilaa seuraavalle bitille
     bsf       koodi_high,0         ; koodi_high sisältää: start-bitit (2 bittiä), toggle (1) ja
                                    ; address-biteistä 3, yht. 6

     decf      Bittilaskuri,f       ; yksi RC5-bitti vastaanotettu, vähennetään laskuria


;-----------------------------------------------------------------------------------------------
```

```
Lue1:                           ; Luetaan muut bitit bittipari kerrallaan.
     btfss     Timer0Lippu,0    ; Odotetaan taas että 880us tulee täyteen.
     goto      Lue1
     bcf       Timer0Lippu,0
     movlw     D'146'           ; Viritetään ajastin valmiiksi bittiparin toisen bitin lukua
     movwf     TMR0             ; varten (880us).
     banksel   OPTION_REG
     movlw     b'00000010'
     movwf     OPTION_REG
     banksel   INTCON

     btfsc     PORTA,5          ; Luetaan bittiparin ensimmäinen bitti.
     bsf       bitti1,0
Lue2:                           ; Luetaan toinen bitti bittiparista:
     btfss     Timer0Lippu,0    ; odotetaan 880us täyttymistä
     goto      Lue2
     bcf       Timer0Lippu,0
     movlw     D'146'           ; viritetään ajastin 880us
     movwf     TMR0
     banksel   OPTION_REG
     movlw     b'00000010'
     movwf     OPTION_REG
     banksel   INTCON

     btfsc     PORTA,5          ; Luetaan bittiparin toinen bitti.
     bsf       bitti2,0

     movf      bitti1,w         ; Tarkistetaan laillisuus, bittien on oltava toistensa
     xorwf     bitti2,w         ; vastakohtia.

     movwf     OnkoLaillinenPari   ; jos on 1, on laillinen pari
     btfss     OnkoLaillinenPari,0
     goto      Virhe            ; Ei ollut laillinen bittipari, mennään Virhe-tilaan.
     call      TalletaKoodi     ; On laillinen pari, jatketaan tallettamalla saatu bittiparin
                                ; osoittama bitti

     clrf      bitti1           ; nollauksia
     clrf      bitti2
     clrf      OnkoLaillinenPari
     decfsz    Bittilaskuri,f   ; Vähennetään bittilaskuria, kun nolla:
     goto      Lue1
     goto      KoodinTarkistus  ; mennään tarkistamaan saatu koodi.

;-------------------------------------------------------------------------------------------------

TalletaKoodi:                   ; Talletetaan saadun 14-bittisen koodin eniten merkitsevät 6
                                ; bittiä koodi_high -muuttujaan ja 8 vähiten merkitsevää
                                ; koodi_low -muuttujaan.
     bcf       STATUS,Z         ; <-- tämä sen takia, että koodi_high_laskuri ei pääsisi
                                ; kiepsahtamaan ympäri.
     movf      koodi_high_laskuri,f     ; Laskuri osoittaa, milloin ollaan saatu 6 ylintä bittiä, ja
     btfsc     STATUS,Z                 ; aletaan täyttämään loppuja 8 bittiä koodi_low –muuttujaan.
     goto      KoodiLow
     decf      koodi_high_laskuri,f

     rlf       koodi_high,f     ; Siirretään edellinen bitti pois tieltä.
     btfsc     bitti1,0         ; bitti1 vastaa RC5 koodin bittiparin tarkoittamaa bittiä.
     bsf       koodi_high,0
     return
KoodiLow:
     rlf       koodi_low,f
     btfsc     bitti1,0
     bsf       koodi_low,0
     return


;-------------------------------------------------------------------------------------------------
```

```
KoodinTarkistus:                ; tänne tullaan vain, kun saatu yksi kokonainen ja laillinen RC5-
                                ; koodi.
     bcf       INTCON, T0IE     ; Estetään TMR0-keskeytykset koska nyt ei enää lueta koodia.

     ; Talletetaan ensin juuri saatu koodi VastaanotetutKoodit-listaan:

     movf      VastaanotetutKoodit+2,w
     movwf     VastaanotetutKoodit+3
     movf      VastaanotetutKoodit+1,w
     movwf     VastaanotetutKoodit+2
     movf      VastaanotetutKoodit,w
     movwf     VastaanotetutKoodit+1
     movf      koodi_low,w           ; Uusin koodi on aina listassa ylinnä, vanhin kirjoitetaan
     movwf     VastaanotetutKoodit   ; yli.

     ; Tarkistetaan vastaavatko viimeisimmät 4 talletettua koodia vaadittua koodisarjaa (alussa
     ; kaikki nollattu):

     movf      VaaditutKoodit+3,w
     xorwf     VastaanotetutKoodit,w ; luku1 XOR luku2 = 0 ainoastaan silloin kun molemmat luvut
                                     ; ovat samoja.
     btfss     STATUS,Z              ; jos Z=1, saatiin ensimmäinen vastaavuus
     goto      EiKuuluSarjaan        ; jos Z=0, koodi ei kuulu sarjaan

     movf      VaaditutKoodit+2,w
     xorwf     VastaanotetutKoodit+1,w
     btfss     STATUS,Z
     goto      EiKuuluSarjaan

     movf      VaaditutKoodit+1,w
     xorwf     VastaanotetutKoodit+2,w
     btfss     STATUS,Z
     goto      EiKuuluSarjaan

     movf      VaaditutKoodit,w
     xorwf     VastaanotetutKoodit+3,w
     btfss     STATUS,Z
     goto      EiKuuluSarjaan

     clrf      koodi_low
     clrf      koodi_high
     movlw     b'00000001'
     xorwf     PORTC,f             ; Oikea sarja vastaanotettu. Vaihdetaan ledin 1 tilaa.
     goto      MainLoop

EiKuuluSarjaan:                  ; Koodi ei kuulu sarjaan. Palataan MainLooppiin odottamaan
                                 ; seuraavaa koodia.
     clrf      koodi_low         ; nollauksia
     clrf      koodi_high
     goto      MainLoop

;-----------------------------------------------------------------------------------------------

Virhe:                           ; Tänne tullaan, jos vastaanotettu koodi ei ole RC5-protokollan
                                 ; mukainen.
     clrf      koodi_low         ; nollauksia
     clrf      koodi_high
     clrf      OnkoLaillinenPari
     clrf      bitti1
     clrf      bitti2
     bcf       INTCON, T0IE
     bsf       PORTC,3           ; Virheellinen koodi vastaanotettu, pidetään lediä 4 päällä
     call      viive200ms        ; hetken aikaa.
     call      viive200ms
     bcf       PORTC,3
     goto      MainLoop          ; palataan odottamaan seuraavaa koodia

;-----------------------------------------------------------------------------------------------

viive200ms:
     decfsz    viive1,f          ; Sisempi kierros: 3 käskyä *  256 kierrosta = 768 käskyä/us
     goto      viive200ms
     decfsz    viive2,f          ; Ulompi kierros: (3 + 768 käskyä) * 256 kierrosta = 197376
     goto      viive200ms        ; käskyä/us = 0,197s eli noin 200ms (PICkit2 Lesson 2)
     return


     end                         ; endin jälkeen mitään ei käännetä!
```

15.0 INSTRUCTION SET SUMMARY

The PIC16F690 instruction set is highly orthogonal and is comprised of three basic categories:

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal and control** operations

Each PIC16 instruction is a 14-bit word divided into an **opcode,** which specifies the instruction type and one or more **operands,** which further specify the operation of the instruction. The formats for each of the categories is presented in Figure 15-1, while the various opcode fields are summarized in Table 15-1.

Table 15-2 lists the instructions recognized by the MPASM™ assembler.

For **byte-oriented** instructions, 'f' represents a file register designator and 'd' represents a destination designator. The file register designator specifies which file register is to be used by the instruction.

The destination designator specifies where the result of the operation is to be placed. If 'd' is zero, the result is placed in the W register. If 'd' is one, the result is placed in the file register specified in the instruction.

For **bit-oriented** instructions, 'b' represents a bit field designator, which selects the bit affected by the operation, while 'f' represents the address of the file in which the bit is located.

For **literal and control** operations, 'k' represents an 8-bit or 11-bit constant, or literal value.

One instruction cycle consists of four oscillator periods; for an oscillator frequency of 4 MHz, this gives a normal instruction execution time of 1 µs. All instructions are executed within a single instruction cycle, unless a conditional test is true, or the program counter is changed as a result of an instruction. When this occurs, the execution takes two instruction cycles, with the second cycle executed as a `NOP`.

All instruction examples use the format '`0xhh`' to represent a hexadecimal number, where '`h`' signifies a hexadecimal digit.

15.1 Read-Modify-Write Operations

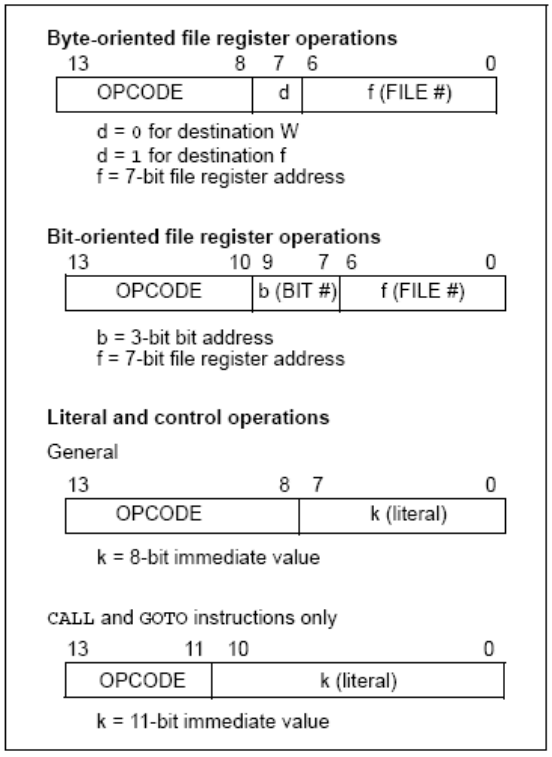
Any instruction that specifies a file register as part of the instruction performs a Read-Modify-Write (RMW) operation. The register is read, the data is modified, and the result is stored according to either the instruction, or the destination designator 'd'. A read operation is performed on a register even if the instruction writes to that register.

For example, a `CLRF PORTA` instruction will read PORTA, clear all the data bits, then write the result back to PORTA. This example would have the unintended consequence of clearing the condition that set the RAIF flag.

TABLE 15-1: OPCODE FIELD DESCRIPTIONS

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= `0` or `1`). The assembler will generate code with x = `0`. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = `0`: store result in W, d = `1`: store result in file register f. Default is d = `1`.
PC	Program Counter
$\overline{TO}$	Time-out bit
C	Carry bit
DC	Digit carry bit
Z	Zero bit
$\overline{PD}$	Power-down bit

FIGURE 15-1: GENERAL FORMAT FOR INSTRUCTIONS



© 2008 Microchip Technology Inc.

PIC16F631/677/685/687/689/690

TABLE 15-2: PIC16F684 INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C, DC, Z	1, 2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1, 2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	–	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1, 2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1, 2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1, 2, 3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1, 2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1, 2, 3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1, 2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1, 2
MOVWF	f	Move W to f	1	00	0000	lfff	ffff		
NOP	–	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1, 2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1, 2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C, DC, Z	1, 2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1, 2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1, 2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1, 2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1, 2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C, DC, Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call Subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	–	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$, $\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	–	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	–	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	–	Go into Standby mode	1	00	0000	0110	0011	$\overline{TO}$, $\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C, DC, Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF GPIO, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

3: If the Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

© 2008 Microchip Technology Inc.

15.2 Instruction Descriptions

ADDLW	Add literal and W
Syntax:	[*label*] ADDLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) + k → (W)
Status Affected:	C, DC, Z
Description:	The contents of the W register are added to the eight-bit literal 'k' and the result is placed in the W register.

ADDWF	Add W and f
Syntax:	[*label*] ADDWF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(W) + (f) → (destination)
Status Affected:	C, DC, Z
Description:	Add the contents of the W register with register 'f'. If 'd' is '0', the result is stored in the W register. If 'd' is '1', the result is stored back in register 'f'.

ANDLW	AND literal with W
Syntax:	[*label*] ANDLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) .AND. (k) → (W)
Status Affected:	Z
Description:	The contents of W register are AND'ed with the eight-bit literal 'k'. The result is placed in the W register.

ANDWF	AND W with f
Syntax:	[*label*] ANDWF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(W) .AND. (f) → (destination)
Status Affected:	Z
Description:	AND the W register with register 'f'. If 'd' is '0', the result is stored in the W register. If 'd' is '1', the result is stored back in register 'f'.

BCF	Bit Clear f
Syntax:	[*label*] BCF f,b
Operands:	0 ≤ f ≤ 127 0 ≤ b ≤ 7
Operation:	0 → (f<b>)
Status Affected:	None
Description:	Bit 'b' in register 'f' is cleared.

BSF	Bit Set f
Syntax:	[*label*] BSF f,b
Operands:	0 ≤ f ≤ 127 0 ≤ b ≤ 7
Operation:	1 → (f<b>)
Status Affected:	None
Description:	Bit 'b' in register 'f' is set.

BTFSC	Bit Test f, Skip if Clear
Syntax:	[*label*] BTFSC f,b
Operands:	0 ≤ f ≤ 127 0 ≤ b ≤ 7
Operation:	skip if (f<b>) = 0
Status Affected:	None
Description:	If bit 'b' in register 'f' is '1', the next instruction is executed. If bit 'b' in register 'f' is '0', the next instruction is discarded, and a NOP is executed instead, making this a two-cycle instruction.

© 2008 Microchip Technology Inc.

BTFSS	Bit Test f, Skip if Set
Syntax:	[*label*] BTFSS f,b
Operands:	0 ≤ f ≤ 127 0 ≤ b < 7
Operation:	skip if (f<b>) = 1
Status Affected:	None
Description:	If bit 'b' in register 'f' is '0', the next instruction is executed. If bit 'b' is '1', then the next instruction is discarded and a NOP is executed instead, making this a two-cycle instruction.

CALL	Call Subroutine
Syntax:	[*label*] CALL k
Operands:	0 ≤ k ≤ 2047
Operation:	(PC)+ 1→ TOS, k → PC<10:0>, (PCLATH<4:3>) → PC<12:11>
Status Affected:	None
Description:	Call Subroutine. First, return address (PC + 1) is pushed onto the stack. The eleven-bit immediate address is loaded into PC bits <10:0>. The upper bits of the PC are loaded from PCLATH. CALL is a two-cycle instruction.

CLRF	Clear f
Syntax:	[*label*] CLRF f
Operands:	0 ≤ f ≤ 127
Operation:	00h → (f) 1 → Z
Status Affected:	Z
Description:	The contents of register 'f' are cleared and the Z bit is set.

CLRW	Clear W
Syntax:	[*label*] CLRW
Operands:	None
Operation:	00h → (W) 1 → Z
Status Affected:	Z
Description:	W register is cleared. Zero bit (Z) is set.

CLRWDT	Clear Watchdog Timer
Syntax:	[*label*] CLRWDT
Operands:	None
Operation:	00h → WDT 0 → WDT prescaler, 1 → $\overline{TO}$ 1 → $\overline{PD}$
Status Affected:	$\overline{TO}$, $\overline{PD}$
Description:	CLRWDT instruction resets the Watchdog Timer. It also resets the prescaler of the WDT. Status bits $\overline{TO}$ and $\overline{PD}$ are set.

COMF	Complement f
Syntax:	[*label*] COMF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	($\bar{f}$) → (destination)
Status Affected:	Z
Description:	The contents of register 'f' are complemented. If 'd' is '0', the result is stored in W. If 'd' is '1', the result is stored back in register 'f'.

DECF	Decrement f
Syntax:	[*label*] DECF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) - 1 → (destination)
Status Affected:	Z
Description:	Decrement register 'f'. If 'd' is '0', the result is stored in the W register. If 'd' is '1', the result is stored back in register 'f'.

© 2008 Microchip Technology Inc.

DECFSZ	Decrement f, Skip if 0
Syntax:	[*label*] DECFSZ f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) - 1 → (destination); skip if result = 0
Status Affected:	None
Description:	The contents of register 'f' are decremented. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed back in register 'f'. If the result is '1', the next instruction is executed. If the result is '0', then a NOP is executed instead, making it a two-cycle instruction.

GOTO	Unconditional Branch
Syntax:	[*label*] GOTO k
Operands:	0 ≤ k ≤ 2047
Operation:	k → PC<10:0> PCLATH<4:3> → PC<12:11>
Status Affected:	None
Description:	GOTO is an unconditional branch. The eleven-bit immediate value is loaded into PC bits <10:0>. The upper bits of PC are loaded from PCLATH<4:3>. GOTO is a two-cycle instruction.

INCF	Increment f
Syntax:	[*label*] INCF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) + 1 → (destination)
Status Affected:	Z
Description:	The contents of register 'f' are incremented. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed back in register 'f'.

INCFSZ	Increment f, Skip if 0
Syntax:	[*label*] INCFSZ f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) + 1 → (destination), skip if result = 0
Status Affected:	None
Description:	The contents of register 'f' are incremented. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed back in register 'f'. If the result is '1', the next instruction is executed. If the result is '0', a NOP is executed instead, making it a two-cycle instruction.

IORLW	Inclusive OR literal with W
Syntax:	[*label*] IORLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) .OR. k → (W)
Status Affected:	Z
Description:	The contents of the W register are OR'ed with the eight-bit literal 'k'. The result is placed in the W register.

IORWF	Inclusive OR W with f
Syntax:	[*label*] IORWF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(W) .OR. (f) → (destination)
Status Affected:	Z
Description:	Inclusive OR the W register with register 'f'. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed back in register 'f'.

© 2008 Microchip Technology Inc.

MOVF	Move f
Syntax:	[*label*] MOVF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) → (dest)
Status Affected:	Z
Description:	The contents of register 'f' is moved to a destination dependent upon the status of 'd'. If d = 0, destination is W register. If d = 1, the destination is file register 'f' itself. d = 1 is useful to test a file register since status flag Z is affected.
Words:	1
Cycles:	1
Example:	MOVF FSR, 0 After Instruction W = value in FSR register Z = 1

MOVWF	Move W to f
Syntax:	[*label*] MOVWF f
Operands:	0 ≤ f ≤ 127
Operation:	(W) → (f)
Status Affected:	None
Description:	Move data from W register to register 'f'.
Words:	1
Cycles:	1
Example:	MOVW OPTION F Before Instruction OPTION = 0xFF W = 0x4F After Instruction OPTION = 0x4F W = 0x4F

MOVLW	Move literal to W
Syntax:	[*label*] MOVLW k
Operands:	0 ≤ k ≤ 255
Operation:	k → (W)
Status Affected:	None
Description:	The eight-bit literal 'k' is loaded into W register. The "don't cares" will assemble as '0's.
Words:	1
Cycles:	1
Example:	MOVLW 0x5A After Instruction W = 0x5A

NOP	No Operation
Syntax:	[*label*] NOP
Operands:	None
Operation:	No operation
Status Affected:	None
Description:	No operation.
Words:	1
Cycles:	1
Example:	NOP

© 2008 Microchip Technology Inc.

RETFIE	Return from Interrupt
Syntax:	[*label*] RETFIE
Operands:	None
Operation:	TOS → PC, 1 → GIE
Status Affected:	None
Description:	Return from Interrupt. Stack is POPed and Top-of-Stack (TOS) is loaded in the PC. Interrupts are enabled by setting Global Interrupt Enable bit, GIE (INTCON<7>). This is a two-cycle instruction.
Words:	1
Cycles:	2
Example:	`RETFIE` After Interrupt PC = TOS GIE = 1

RETLW	Return with literal in W
Syntax:	[*label*] RETLW k
Operands:	0 ≤ k ≤ 255
Operation:	k → (W); TOS → PC
Status Affected:	None
Description:	The W register is loaded with the eight-bit literal 'k'. The program counter is loaded from the top of the stack (the return address). This is a two-cycle instruction.
Words:	1
Cycles:	2

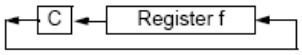
Example:	`CALL TABLE;W contains table`
	`;offset value`
	`TABLE ;W now has`
	`;table value`
	•
	•
	`ADDWF PC ;W = offset`
	`RETLW k1 ;Begin table`
	`RETLW k2 ;`
	•
	•
	•
	`RETLW kn ;End of table`
	
	Before Instruction
	W = 0x07
	After Instruction
	W = value of k8

RETURN	Return from Subroutine
Syntax:	[*label*] RETURN
Operands:	None
Operation:	TOS → PC
Status Affected:	None
Description:	Return from subroutine. The stack is POPed and the top of the stack (TOS) is loaded into the program counter. This is a two-cycle instruction.

© 2008 Microchip Technology Inc.

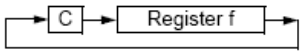
PIC16F631/677/685/687/689/690

RLF	Rotate Left f through Carry
Syntax:	[*label*] RLF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	See description below
Status Affected:	C
Description:	The contents of register 'f' are rotated one bit to the left through the Carry flag. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is stored back in register 'f'.
Words:	1
Cycles:	1
Example:	RLF REG1,0 Before Instruction REG1 = 1110 0110 C = 0 After Instruction REG1 = 1110 0110 W = 1100 1100 C = 1



SLEEP	Enter Sleep mode
Syntax:	[*label*] SLEEP
Operands:	None
Operation:	00h → WDT, 0 → WDT prescaler, 1 → $\overline{TO}$, 0 → $\overline{PD}$
Status Affected:	$\overline{TO}$, $\overline{PD}$
Description:	The power-down Status bit, $\overline{PD}$ is cleared. Time-out Status bit, $\overline{TO}$ is set. Watchdog Timer and its prescaler are cleared. The processor is put into Sleep mode with the oscillator stopped.

RRF	Rotate Right f through Carry
Syntax:	[*label*] RRF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	See description below
Status Affected:	C
Description:	The contents of register 'f' are rotated one bit to the right through the Carry flag. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed back in register 'f'.



SUBLW	Subtract W from literal
Syntax:	[*label*] SUBLW k
Operands:	0 ≤ k ≤ 255
Operation:	k - (W) → (W)
Status Affected:	C, DC, Z
Description:	The W register is subtracted (2's complement method) from the eight-bit literal 'k'. The result is placed in the W register.

C = 0	W > k
C = 1	W ≤ k
DC = 0	W<3:0> > k<3:0>
DC = 1	W<3:0> ≤ k<3:0>

© 2008 Microchip Technology Inc.

SUBWF **Subtract W from f**

Syntax: [*label*] SUBWF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f) - (W) → (destination)

Status Affected: C, DC, Z

Description: Subtract (2's complement method) W register from register 'f'. If 'd' is '0', the result is stored in the W register. If 'd' is '1', the result is stored back in register 'f'.

C = 0	W > f
C = 1	W ≤ f
DC = 0	W<3:0> > f<3:0>
DC = 1	W<3:0> ≤ f<3:0>

SWAPF **Swap Nibbles in f**

Syntax: [*label*] SWAPF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (f<3:0>) → (destination<7:4>),
(f<7:4>) → (destination<3:0>)

Status Affected: None

Description: The upper and lower nibbles of register 'f' are exchanged. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed in register 'f'.

XORLW **Exclusive OR literal with W**

Syntax: [*label*] XORLW k

Operands: $0 \le k \le 255$

Operation: (W) .XOR. k → (W)

Status Affected: Z

Description: The contents of the W register are XOR'ed with the eight-bit literal 'k'. The result is placed in the W register.

XORWF **Exclusive OR W with f**

Syntax: [*label*] XORWF f,d

Operands: $0 \le f \le 127$
$d \in [0,1]$

Operation: (W) .XOR. (f) → (destination)

Status Affected: Z

Description: Exclusive OR the contents of the W register with register 'f'. If 'd' is '0', the result is stored in the W register. If 'd' is '1', the result is stored back in register 'f'.

© 2008 Microchip Technology Inc.