UPORABA MIKROPROCESORJEV
Upravljanje stojala za teleskop z analognim krmilnikom

Jure Varlec, 28030404

19. september 2010

1 Opis naloge

Sodobno stojalo za teleskop je motorizirano in vsebuje procesno enoto, ki upravlja z motorji. Ta procesna enota je povezana z zunanjim krmilnikom, ki predstavlja uporabniški vmesnik. Od njega prejema ukaze, kot so "pojdi levo", "povečaj hitrost", "obrni se na tele koordinate". Vmesnik podpira tudi naprednejše funkcije stojala, saj zna to, med drugim, opraviti kalibracijo položaja, kompenzirati mehanske posebnosti konkretnega stojala, in vsebuje katalog nebesnih objektov. Izkaže pa se, da je priloženi krmilnik nekoliko neroden kot vmesnik za ročno usmerjanje teleskopa. To se še poslabša z leti, ko se gumbi na njem obrabijo.

Stojalo omogoča tudi nadzor z osebnim računalnikom. V ta namen lahko prejema ukaze preko vmesnika RS-232 po protokolu, ki je odvisen od proizvajalca in modela. To je nadvse primerno za izvajanje kompleksnejših nalog, denimo avtomatizacijo astronomskih opazovanj. Kot nadomestek za preprosto ročno usmerjanje teleskopa pa je uporaba računalnika pretiravanje.

Naloga je torej izdelati krmilnik, ki bo znal pošiljati ukaze stojalu preko vmesnika RS-232. Smer in hitrost naj določa položaj analogne paličice: večji kot je odklon paličice v dani smeri, večja je hitrost. Naloga je namenjena spoznavanju zbirniškega jezika. Program naj bo zasnovan tako, da ga je moč prilagoditi različnim tipom stojal.

2 Elektronsko vezje

Shema prototipnega vezja je priložena poročilu. Vezje ima v grobem šest delov:

- mikrokontroler PIC 16F877 podjetja Microchip, h kateremu spadata tudi kristal s frekvenco 12MHz in tipka za reset čipa;
- LED za komunkacijo z uporabnikom;
- tipka za dajanje ukazov mikrokontrolerju;
- ICSP konektor za programiranje mikrokontrolerja;
- kontrolna paličica, predstavljena z dvema potenciometroma, h kateri spadajo tudi tranzistorski ojačevalniki;
- MAX232 gonilnik, ki TTL napetostne nivoje prevaja na RS-232 nivoje.

Kontrolna paličica se giblje po dveh pravokotnih oseh, ki vrtita vsaka svoj potenciometer. Ta imata razpon upornosti do 10kΩ. To je primerljivo z največjo priporočeno vhodno impedanco analogno-digitalnega pretvornika v mikrokontrolerju, zaradi česar ima vsak izhod potenciometrov tranzistorski sledilnik napetosti. Napetost, ki jo dobimo na izhodu sledilnika, ima nelinearno odvisnost od upornosti.

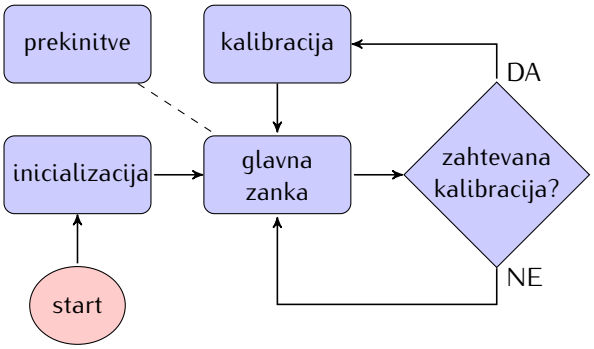
3 Oris rešitve

Osnovna ideja je sila preprosta: na določen interval, denimo na desetinko sekunde, preberemo odmik kontrolne paličice. Na podlagi tega določimo smer in hitrost gibanja teleskopa. Generiramo ukaze in jih pošljemo preko serijske povezave. Ponavljamo. Tej glavni zanki dodamo še rutino za kalibracijo paličice in prekitnitveno rutino. Slednja je potrebna, ker pošiljanje podatkov po RS-232 protokolu poteka asinhrono. Diagram poteka je na sliki 1.

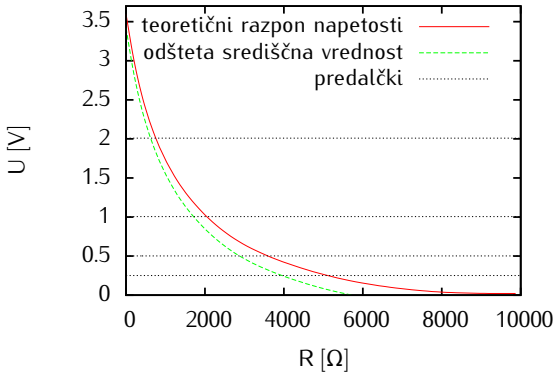
4 Kalibracija

Ta rutina je najpreprostejši del programa, saj je kratka, deluje z izklopljenimi prekinitvami in uvede delo s podatkovnim pomnilnikom ter analogno-digitalnim pretvornikom.

Analogno-digitalni pretvornik (ADC) na uporabljenem mikrokontrolerju je 10-biten, podatkovni registri pa so 8-bitni. Ker 10-bitne natančnosti ne potrebujemo, se splača



Slika 1: Diagram poteka programa.



Slika 2: Odvisnost izmerjene napetosti od upornosti potenciometra. Pri maksimalnem odklonu paličice izmerimo vrednost približno 110, pri ničelnem odklonu (ko je paličica na sredini) pa približno 10 od maksimalne možne vrednosti 255, katera predstavlja referenčno napetost 5V. Ko to središčno vrednost odštejemo (zelena krivulja) nam ostane merljivo območje od 0 do 100, torej do približno dveh voltov na grafu. To območje razdelimo na predalčke, ki so predstavljeni s črtkanimi črtami.

manj pomembna bita zanemariti, saj si olajšamo delo. Tedaj lahko izmerimo 256 različnih vrednosti. ADC omogoča, da mu podamo referenčno napetost. Zaradi enostavnosti ga nastavimo tako, da uporablja v ta namen kar napajalno napetost. Izkaže se, da ima paličica tedaj od središčne do skrajne lege razpon do približno 110 oziroma dobra dva volta od petih.

Vezje je sestavljeno tako, da za vsako os paličice odčitamo dve vrednosti. Pri odklonu v neko smer se ena od teh vrednosti veča, druga pa manjša. Na ta način ugotovimo, v katero smer je paličica odklonjena po dani osi. Odvisnost napetosti od upornosti je prikazana na sliki 2.

Stojalo teleskopa omogoča neko število diskretnih hitrosti premikanja. Območje, v katerem se gibljejo možni izmerki napetosti, razdelimo na predalčke. Med izvajanjem glavne zanke bomo hitrost izbrali glede na to, v kateri predalček bo spadal izmerek odklona. Kako naj bodo razporejeni predalčki je stvar okusa in udobja, kar se pokaže šele z daljšo uporabo krnilnika. Ker te ne moremo biti deležni vnaprej, razporedimo predalčke tako, da bodo spremembe hitrosti po odmiku razporejene približno enakomerno. Nelinearno krivuljo s slike 2 lahko približno oponašamo z eksponentno funkcijo. Ker naš mikrokontroler nima inštrukcij za deljenje in množenje in ker oblika eksponentne funkcije ni posebej pomembna, si izberimo osnovo 2, saj množenje in deljenje s tem številom implementiramo zelo enostavno z zamikanjem bitov v registru.

Postopek kalibracije je naslednji:

1. Signal uporabniku, naj izmeri središčni položaj.
2. Merjenje središčnega položaja. Ker paličica ni nikoli čisto na sredini, izmerimo vse štiri kanale ADC-ja in jih povprečimo.
3. Signal uporabniku, naj izmeri skrajni položaj.
4. Merjenje skrajne lege. Paličica naj bo odklonjena po eni od osi, ne po diagonali. Izmerimo vse štiri kanale ADC-ja in izberemo največjo vrednost.
5. Od izmerjene skrajne lege odštejemo središčno lego. Rezultat štirikrat delimo z 2, vrednosti shranimo v temu namenjeno polje.
6. Zaključimo s kalibracijo.

Komunikacija z uporabnikom teče preko LED in tipke. Z diodo signaliziramo pripravljenost na meritev, uporabnik pa s pritiskom na tipko pove, da je paličica v primerni legi za meritev. Ker je med pritiskom na tipko vrednost na digitalnem vhodu, na katerem je tipka, slabo definirana, pritisk registriramo šele, ko je tipka pritisnjena več kot 50ms. Poleg tega je med fazami kalibracije še premor dolžine 300ms, da ima uporabnik čas spustiti tipko.

Paličica ni nikoli čisto na sredini, temveč je rahlo odklonjena. Vrednosti, ki jih izmerimo na štirih kanalih ADC-ja, so paroma korelirane: ko je ena večja, je druga manjša. Ker pa se ne razlikujejo veliko si lahko privoščimo določiti eno samo vrednost, ki bo dobra kot središčna vrednost za vse izmerke. Dobimo jo s povprečenjem izmerkov.

Ne glede na to, v katero od štirih z osema vzporednih smeri je odklonjena paličica, odčitamo vedno enako mejno vrednost, zato je dovolj, da odčitamo eno, vsaj dokler potenciometra še nista obrabljena. Ne smemo pa meriti diagonale, kot je to v navadi pri igralnih palicah za računalnike, saj je konkretna paličica, ki je uporabljena v tem vezju, fizično omejena na gibanje po krogu namesto po kvadratu. Lahko bi sicer izračunali velikost odmika, vendar je merjenje odmika vzdolž osi lažje in zanesljivejše. K temu se bomo še vrnili v razdelku 6.2.

V podatkovnem pomnilniku mikrokontrolerja si pripravimo polje dolžine petih bajtov, v katerega bomo vpisali meje predalčkov. Zanka, ki te meje izračuna in vpiše v polje nudi priložnost, da si ogledamo, kako poteka posredno

naslavljanje pomnilnika v PIC mikrokontrolerjih. Za razliko od neposrednega naslavljanja, pri katerem je naslov registra, na katerega deluje inštrukcija, vdelan v samo inštrukcijo, je pri posrednem naslavljanju naslov vpisan v register z oznako FSR. Ta služi kot kazalec. Ko nato izvedemo operacijo na registru z oznako INDF, se ta operacija izvede na registru, katerega naslov je vpisan v FSR.

Polje je iz petih zaporednih podatkovnih registrov. Naj se prvi imenuje thresh, drugi bins, ostali trije pa so anonimni (slika 3). Naj bo v registru binmax največja izmerjena vrednost, v registru center pa središčna vrednosti. Tedaj se izračun predalčkov v psevdokodi zapiše

```
FSR = addr(thresh) + 4
INDF = binmax - center
do
   shift_right(INDF)
   FSR = FSR - 1
until FSR == addr(thresh)
```

Koda v zbirniku je seveda primerno daljša. Preverjanje pogoja `FSR == addr(thresh)`, denimo, sestoji iz treh inštrukcij: nalaganja naslova thresh v delovni register, XOR operacije s FSR registrom in preverjanja STATUS registra, če je rezultat operacije nič.

V zgornji kodi je operator addr izmišljen. Ko nastopa register brez operatorja, je uporabljena njegova vsebina, z operatorjem addr pa njegov naslov. Slednje izvedemo tako, da uporabljamo inštrukcije, ki za argument jemljejo dobesedne številke, zbirniku pa ukažemo, da vanje vpiše ustrezne naslove.

V polju dolžine 4, ki se začne s registrom bins, so zdaj meje predalčkov, ki jih lahko uporabi glavna zanka za določanje hitrosti. V registru thresh pa je vrednost, ki je še pol nižja od najnižjega predalčka. To uporabimo kot mejo za mrtvo cono premikov paličice. Več o tem v razdelku 6.

Slika 3: Polje z mejami predalčkov.

5 Prekinitvena rutina

Ta rutina je zadolžena za dvoje: prestreza pritisk uporabnikove tipke in nalaga podatke za pošiljanje na serijska vrata. Ob resetu mikrokontroler začne z izvajanjem na programskem naslovu 0. Tja vpišemo skok na glavni del kode. Prekinitvena rutina se nahaja na naslovu 4.

Podatkovni pomnilnik mikrokontrolerja je razdeljen na splošne registre in posebne registre. Prvi služijo za hrambo podatkov, drugi pa za nadzor mikrokontrolerja. Nekateri od teh so posebej odlikovani, saj hranijo kontekst izvajanja programa. Preden v prekinitveni rutini naredimo karkoli, kar bi lahko vplivalo na te registre, jih moramo shraniti. Ti registri so: delovni register W, STATUS, PCLATH, FSR. Register STATUS vsebuje nastavitve dostopa do pomnilnika, dodatne rezultate aritmetično-logičnih operacij (rezultat je nič, rezultat presega register), tip reseta, če je do njega prišlo. Register PCLATH vsebuje visoke bite za naslavljanje programskega pomnilnika, kar si bomo ogledali v razdelku 7.1. Nekatere inštrukcije imajo stranske učinke na W, STATUS in PCLATH. Ker pri vstopu v prekinitveno rutino ne vemo, kako je nastavljen dostop do podatkovnega pomnilnika, se morajo registri, kamor bomo shranjevali vrednosti posebnih registrov, nahajati v odlikovanjem delu podatkovnega pomnilnika.

Pričujoči mikrokontroler ima podatkovni pomnilnik razdeljen na štiri *banke*, vsaka obsega 128 registrov. Razlog je v tem, da je v posamezni inštrukciji prostora zgolj za 7 bitov naslova. Nekaj tega naslovnega prostora zasedajo posebni registri. V vsaki banki je 96 splošnih registrov, v bankah 2 in 3 pa še dodatnih 16 registrov. Od tistih 96 registrov jih je prvih 80 za vsako banko drugačnih, zadnjih 16 pa si banke delijo. Zaradi tega so ti registri posebej primerni za shranjevanje podatkov, ki morajo biti dostopni ne glede na to, katera od bank je trenutno v uporabi. Banko namreč izbiramo z nastavljanjem dveh bitov v registru STATUS.

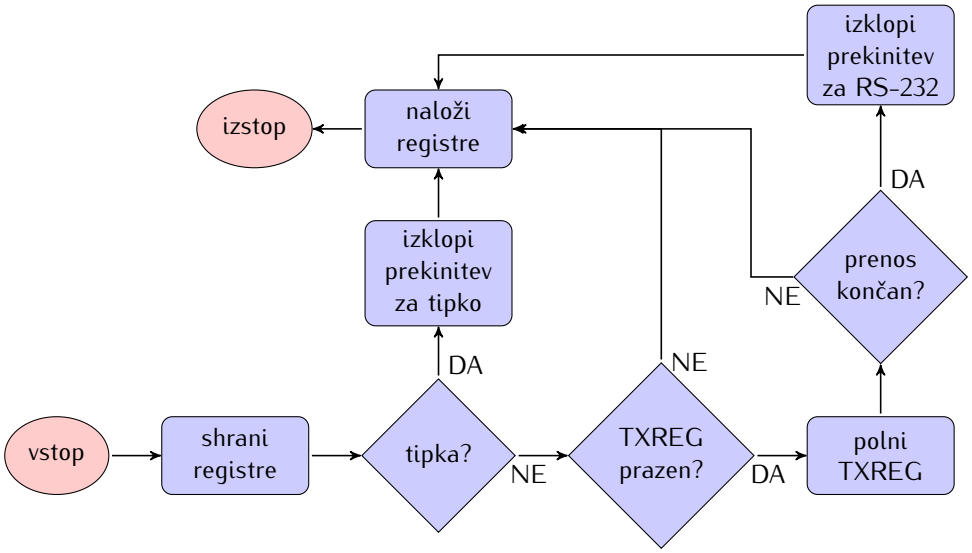
5.1 Uporabnikova tipka

Uporabnikova tipka je vezana na nožico RB0, ki je odlikovana s tem, da ima čisto svojo prekinitveno zastavico. Za prekinitve lahko sicer uporabimo tudi nekatere druge nožice v vhodno-izhodnih vratih B (pričujoči mikrokontroler ima vrata od A do E), vendar imajo te skupno prekinitveno zastavico. Na tipko sta vezani dve funkciji. Pri vklopu mikrokontrolerja ta čaka dve sekundi. Če v tem času uporabnik pritisne tipko, se namesto v običajno glavno zanko izvajanje preseli v diagnostično zanko, ki namesto ukazov za stojalo teleskopa izpisuje izmerjene in izračunane vrednosti.

Druga funkcija, ki je na voljo med izvajanjem glavne zanke, je proženje kalibracije. To je izvedeno tako, da ob pritisku na tipko prekinitvena rutina samo izklopi prekinitveno tipko. Ob koncu iteracije glavne zanke ta preveri, ali je tipka izklopljena, in v tem primeru požene kalibracijo. Za ponoven vklop prekinitvene tipke je zadolžena kalibracijska rutina. Bit, ki vklaplja prekinitveno tipko, torej služi tudi kot zastavica za zahtevo kalibracije.

5.2 RS-232 prenos

Mikrokontroler omogoča asinhron prenos podatkov. Izveden je tako, da v poseben register TXREG programsko naložimo en bajt podatkov. Ti podatki se prenesejo v oddajno strojno opremo, ki poskrbi za prenos. Medtem ko prenos enega bajta teče, je TXREG že prazen, tako da ga lahko takoj ponovno napolnimo z naslednjim bajtom. Tako dosežemo neprekinjen prenos podatkov.



Slika 4: Prekinitvena rutina.

Vsakič, ko se `TXREG` izprazni, se sproži prekinitev. Prekinitvene zastavice ne moremo izklopiti neposredno (kot lahko to naredimo za zastavico tipke), pač pa se izklopi avtomatsko, ko v `TXREG` naložimo nov bajt. Če je bil naložen zadnji bajt, moramo prekinitev izklopiti, saj drugače prekinitvene rutine ne moremo zapustiti. Tako bit, ki vklaplja prekinitev, služi tudi kot zastavica za zaključen prenos.

Podatki, ki jih pošiljamo, so organizirani takole. V banki 0 sta registra `strpos` in `strlen`; prvi vsebuje naslov trenutnega bajta, ki ga želimo poslati, drugi pa število vseh bajtov. Sami podatki se nahajajo v banki 1; prvi razpoložljivi register naj bo označen s `str`. Tako imamo zanje na voljo 80 prostih registrov. Prenos začnemo tako, da glavna zanka naloži podatke v banko 1, v `strpos` naloži naslov prvega bajta, v `strlen` naloži število vseh bajtov in vklopi prekinitev. Takrat izvajanje takoj skoči v prekinitveno rutino. Pogoj za končan prenos je `strpos == addr(str) + strlen`.

Spet uporabljamo posredno naslavljanje preko `FSR` registra. Ker imamo za naslov na razpolago 8 bitov lahko naslovimo banki 0 in 1 hkrati.

6 Glavna zanka

Zasnovo glavne zanke določata dva cilja: prvič, ista koda mora podpirati več tipov stojal; drugič, ker teleskop omogoča le diskretna stanja smeri in hitrosti naj se ukazi pošljejo šele ob spremembi stanja. Prvi cilj zadovoljimo s tem, da ima vsako stojalo svoje rutine, ki generirajo ukaze. Uporabljeno elektronsko vezje sicer nima stikal za izbiro, vseeno pa naj koda omogoča preprosto nadgradnjo. K tem rutinam se bomo vrnili v razdelku 7. Na tem mestu je pomembno le to, da glavna zanka izračuna smer in hitrost ter jima priredi stanja, ki so oštevilčena. Iz teh številk se ukazi generirajo v drugih rutinah, te pa kličemo na način, soroden funkcijskim kazalcem v jeziku C.

00	NW	02	SW	10	W	(x1)	12	N	(y1)
01	NE	03	SE	11	E	(x2)	13	S	(y2)

Tabela 1: Oštevilčenje stanj smeri. Številke stanj so v štiriškem sistemu. Za stojala v ekvatorialni postavitvi je v navadi, da osi imenujemo glede na smeri neba. Pri štirih osnovnih smereh je naveden še register, v katerem je pripadajoči izmerek.

Smiselno število stanj hitrosti je 4, stanj smeri pa 8. En register je dovolj, da opišemo vsa možna stanja. Označimo ga s `spdir`; njegovi spodnji biti določajo hitrost, zgornji pa smer. Hitrost je številka od 0 do 3, iz registra jo dobimo z operacijo AND s številom 0x0f. Enako velja za smer, le da za kopiranje vrednosti iz registra `spdir` v delovni register uporabimo namesto inštrukcije `movf spdir, W` inštrukcijo `swapf spdir, W` ker slednja zamenja zgornjo in spodnjo polovico bajta.

6.1 Določanje smeri

Smeri so določene s številkami od 0 do 7. Preslikava med številkami in stanji je pomembna tako za glavno zanko kot za rutino, ki številke prevaja v ukaze za stojalo. Videli bomo, da je prikladna razporeditev, navedena v tabeli 1.

Izmerki so spravljeni v registrih `x1`, `y1`, `x2` in `y2`. Spravljene imamo tudi stare izmerke. Po meritvi vseh štirih kanalov najprej pogledamo, ali je sprememba na katerekoli večja od vrednosti v registru `thresh`. Če takšne ni, po premoru meritev ponovimo. Na ta način premikom paličice dodamo mrtvo cono, s čimer so prehodi med stanji bolje definirani.

V nadaljevanju izvedemo naslednje:

```
spdir<4,5> = 0
if x2 >= x1
   spdir<4> = 1
   swap(x1, x2)
endif
if y2 >= y1
   spdir<5> = 1
   swap(y1, y2)
endif
```

Zagotovili smo, da velja $\{x, y\}_1 >= \{x, y\}_2$, s čimer se nadaljnja koda poenostavi. Bistvo pa je v tem, da bita `spdir<4,5>` zdaj vsebujeta informacijo o tem, v katero od diagonalnih smeri naj gre teleskop. Zdaj vidimo, zakaj je bil v tabeli 1 primeren četrtiški zapis številk stanj: med diagonalnimi in z osema vzporednimi smermi bo odločal bit `spdir<6>`. Tega nastavimo, če je odklon v kateri od obeh smeri manjši od mejne vrednosti, spravljene v registru `bins`:

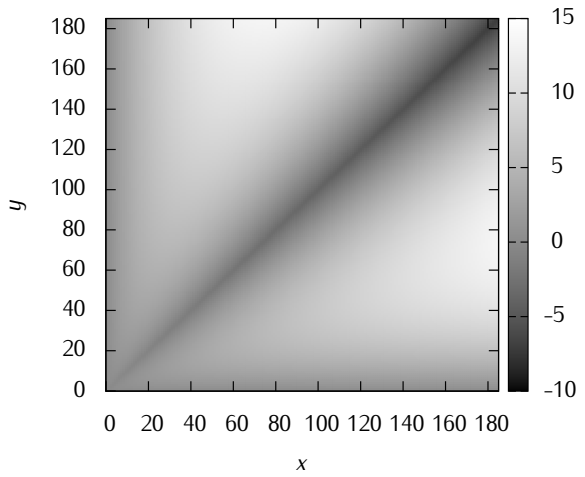
```
if y1 <= bins[0]
   spdir<6> = 1
   spdir<5> = 0
else if x1 <= bins[0]
   spdir<6> = 1
   spdir<4> = spdir<5>
   spdir<5> = 1
endif
```

Zdaj zgornja polovica registra `spdir` vsebuje številko smeri v skladu s tabelo 1 in klic ustrezne rutine bo dal pravi ukaz stojalu teleskopa.

6.2 Določanje hitrosti

Hitrost določa velikost odmika paličice od središčne lege. To običajno izračunamo kot $\sqrt{x^2 + y^2}$. Tak račun pa je na procesorju, ki nima strojnega niti množenja niti deljenja, počasen. Pravzaprav natančnega rezultata niti ne potrebujemo, zato se zadovoljimo s približkom, ki se ga da hitro izračunati. Po Robertsonu je najenostavnejši približek $\max(x, y) + k \min(x, y)$, kjer je k neka konstanta. To konstanto izberemo tako, da je napaka čim manjša. Zadovoljiv rezultat za ne prevelike x oziroma y da, denimo, že $k = 1/2$, ki podaljša račun le za dve inštrukciji. Lahko pa uporabimo $k = 3/8$, ki da mnogo manjšo napako, množenje števila s to konstanto pa še vedno traja le 9 inštrukcij, saj ga implementiramo s tremi zamiki bitov in dvema seštevanjema. Maxsimalno vrednost x oziroma y omejimo na 185, ker je pri večjih rezultat večji od registra. Na tem območju je absolutna napaka povsod manjša od 14 (slika 5).

Ko imamo oceno velikosti odklona, enostavno z odštevanjem pogledamo, v katerega od predalčkov spada. Zaporedna številka predalčka je kar nastavitev hitrosti, ki jo podamo ustrezni rutini, da generira ukaze za stojalo. V register `spdir` jo dodamo z operacijo IOR.



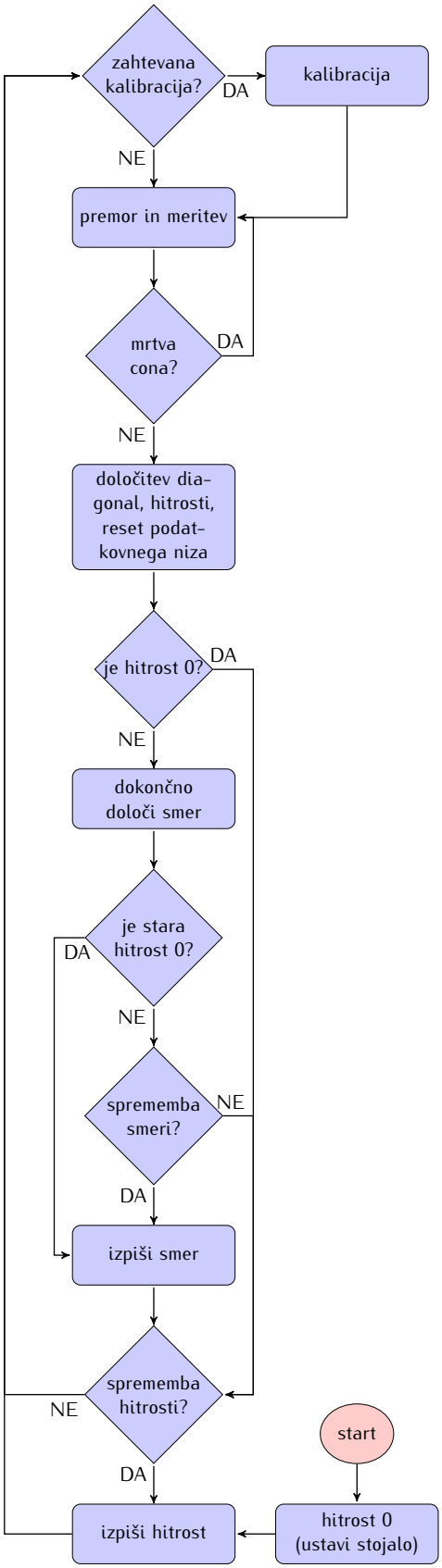
Slika 5: Absolutna napaka ocene velikosti 2-vektorja po Robertsonovi formuli s $k = 3/8$. Napaka ni večja od 14.

6.3 RS-232 prenos

Uporabljamo v razdelku 5.2 opisani mehanizem. Ker je prenos asinhron, ga začnemo čimprej, torej takoj, ko so je na voljo prvi ukaz. Nadaljuje se tudi v naslednjo iteracijo zanke. Denimo, da smo v trenutni iteraciji pripravljeni na generiranje ukazov. Najprej moramo počakati, da se stari odpošljejo. Res je sicer, da je zaradi premora na začetku zanke stari prenos gotovo končan in da tako ali tako nima smisla ponorelo pošiljati ukazov enega za drugim. Vendar pa je treba vedno zagotoviti pravilnost izvajanja, zato najprej počakamo, da se proženje prekinitve zaradi prenosa podatkov izključi.

Ko je stari prenos zagotovo končan, lahko nastavimo kazalec `strpos` nazaj na začetek (t.j. na naslov `str`) in dolžino niza `strlen` na 0. Na `str` naslovimo tudi `FSR`. Rutine za generiranje ukazov so, kot si bomo ogledali v razdelku 7, take, da od mesta, kamor kaže `FSR`, nadaljujejo s pisanjem znakov. Po končanem pisanju vrnejo število zapisanih znakov. V glavni zanki dobljeno število prištejemo `strlen`.

Ker se glavna zanka in prekinitvena rutina ne moreta izvajati hkrati lahko vrednost v registru `strlen` v glavni zanki mirno povečamo tudi medtem, ko prenos teče. Tako lahko najprej pokličemo rutino, ki generira ukaz za spremembo hitrosti, in začnemo prenos tako, da vklopimo prekinitev. Medtem ko prenos teče, pokličemo rutino, ki generira ukaz za spremembo smeri. Ta piše naprej od tam, kjer je prejšnja končala. Ko zaključi, povečamo `strlen`. Če je bila rutina hitra, se prenos še ni končal in se samo nadaljuje. Sicer pa je potrebno le ponovno vklopiti prekinitev.



Slika 6: Potek glavne zanke.

6.4 Potek izvajanja

Zavoljo preprostosti razlage sem v prejšnjih podrazdelkih skoke izvajanja v glavni zanki povečini zamolčal. Ukaze pošiljamo samo takrat, ko pride do spremembe stanja, zato je potek izvajanja nekoliko bolj razvejan. Prikazan je na sliki 6.

6.5 Diagnostična zanka

Kot opisano v razdelku 5.1 lahko ob vklopu krmilnika izberemo namesto običajne glavne zanke diagnostično zanko. Ta je namenjena izpisu izmerkov vseh štirih kanalov ADC in rezultatov kalibracije. Poleg tega preizkusi vse tri LED, ki so na vezju. Tako lahko preizkusimo pravilno delovanje vseh delov krmilnika. Seveda je pravilno delovanje gonilnika MAX232 pogoj za izpis izmerjenih vrednosti.

7 Generiranje ukazov za stojalo

Ukazi so v splošnem nizi bajtov. Odvisno od tipa stojala lahko predstavljajo znake po ASCII tabeli, ni pa nujno. Za razvoj in testiranje programa je bolje, da za začetek ukaze zapišemo v človeku berljivi obliki. Način generiranja nizov pa mora biti tak, da lahko tipe stojal, kar vključuje tudi človeka, preprosto menjavamo.

Poglejmo kot primer funkcijo za izbiro hitrosti. Za argument dobi številko, ki pove hitrost. V višjem programskem jeziku bi uporabili pogojno strukturo, kot je `switch` oziroma `case`. V zbirniku pa uporabimo preračunane skoke (computed goto). Denimo, da je številka hitrosti spravljena v delovnem registru `W`. Tedaj naredimo preračunan skok kot

```
   addwf   PCL, F
   goto    speed0
   goto    speed1
   goto    speed2
   goto    speed3

speed0:
   wrchar  's'
   wrchar  'p'
   wrchar  'd'
   wrchar  '0'
   .
   .
   .
```

Recimo, da je "spd0" ukaz za ustavitev stojala.

Najprej pojasnimo `wrchar`. Dasiravno izgleda kot zbirniška inštrukcija, je v resnici makro, zapisan takole:

```
wrchar    MACRO   char
   movlw  char
   movwf  INDF
   incf   FSR, F
   ENDM
```

Makro je struktura, s katero elegantneje zapišemo ponavljajoče se vzorce inštrukcij. Konkretni primerek je namenjen vpisu znaka na mesto, kamor kaže FSR, o čemer smo že govorili. V kodi, ki jo zaobjema, nadomesti simbol `char` z argumentom, ki ga je prejel ob klicu. Nekaj odstavkov kasneje bomo spoznali, da ima zbirnik svoje spremenljivke; pravzaprav imena rutin (kot je zgoraj speed0), niso nič drugega kot simboli za številke, ki predstavljajo naslove programskega pomnilnika. Ker se makro ob prevajanju "vstavi" na mesto, od koder je bil klican, je treba paziti na to, kateri simboli so lokalni in kateri ne. To velja tudi za simbole naslovov inštrukcij programa. V preprostem primeru, kot je `wrchar`, te skrbi ni.

Inštrukcija `addwf PCL, F` prišteje vrednost v registru `W` registru `PCL`. Ta operacija povzroči skok za toliko inštrukcij naprej, kolikor smo prišteli. Če je bila v registru `W` ničla, se izvede naslednja inštrukcija. Številka hitrosti se v niz, ki predstavlja ukaz, prevede tako, da jo prevedemo v skok na ustrezno rutino. Ker pa lahko s to metodo naslovimo le zaporedne naslove programskega pomnnilnika, na ta mesta nastavimo `goto` inštrukcije in s tem podrutine naslovimo po imenih.

7.1 Naslavljanje programskega pomnilnika

Pričujoči mikrokontroler ima 13 bitov široke naslove programskega pomnilnika, vsaka beseda pa je široka 14 bitov. Naslovi lahko torej 8K besed in toliko pomnilnika tudi ima. Podobno kot pri podatkovnem pomnilniku pa ga ni mogoče nasloviti v enem kosu.

Programski števec (PC), ki vsebuje naslov trenutno izvajane inštrukcije, je torej širok 13-bitov. Razdeljen je na dva registra: `PCL`, ki vsebuje spodnje bite, in `PCLATH`, ki vsebuje zgornje bite. Ko se inštrukcije izvajajo ena za drugo se PC z vsako inštrukcijo poveča za ena, kar se odraža tudi v registru `PCL`. Register `PCLATH`, po drugi strani, pa je namenjen samo pisanju. Zgornjih bitov trenutnega naslova iz njega ne moremo prebrati. Zapisani biti se tudi ne upoštevajo takoj, ko jih tja vpišemo. Uporabijo se šele, ko spremenimo register `PCL` oziroma uporabimo inštrukciji `goto` ali `call`.

V inštrukcijah `goto` in `call` je prostora za 11 bitov programskega naslova. To pomeni, da mora biti programski pomnilnik razdeljen na štiri *strani*. Ob klicu rutine ali skoku se celoten naslov sestavi iz 11 bitov v inštrukciji in najvišjih dveh bitov registra `PCLATH`. Inštrukcija `call` nudi pri tem dodatno pomoč: ob klicu rutine se naslov, od koder je klicana, naloži na sklad, od koder se potem ob inštrukciji `return` včita nazaj v PC. Tako rutini ni treba posebej povedati, od kod je bila klicana. Pri skokih z inštrukcijo `goto` te pomoči seveda nimamo.

Preračunani skoki so seveda še bolj omejeni, saj je register `PCL` širok le 8 bitov. Skok se izvede takoj, ko ga spremenimo, poleg tega pa visokih bitov PC-ja ne moremo prebrati. To pomeni, da dvobajtnega seštevanja ne moremo uporabiti. Najpametneje je, da pazimo na položaj preračunanih skokov v programskem pomnilniku. Zbirnik seveda ve, kam v pomnilniku bodo inštrukcije prišle, zato lahko preverimo, ali smo prekoračili mejo med dvema 256-besednima segmentoma pomnilnika. Zgoraj navedeni preračunani skok okrasimo takole:

```
check:
   addwf   PCL, F
   goto    speed0
   goto    speed1
   goto    speed2
   goto    speed3
   IF (HIGH($) != HIGH(check))
      ERROR "Jump crosses 256-word boundary!"
   ENDIF
```

Simbol $ predstavlja naslov trenutne vrstice. Če se med vrsticama, označenima s `check` in $, spremenijo visoki biti naslova, bo zbirnik javil napako.

7.2 Dolgi skok

Zaradi enostavnosti klicev rutin je lepo, če so vse na prvi strani programskega pomnilnika, skupaj z glavno zanko. Za manjše programe to velja samo po sebi. V našem primeru pa so rutine za generiranje ukazov zaradi izvedbe z makrojem `wrchar` hitro precej velike, sploh če bi imeli večje število implementiranih naborov ukazov. Zbirniku lahko z direktivo `ORG` ukažemo, kam v pomnilnik naj postavlja inštrukcije. Za lažje nadgrajevanje programa zato postavimo te velike rutine na drugo stran programskega pomnilnika. Na ta način ima preostanek programa še nekaj prostora na prvi strani, kamor se lahko širi.

Ker želimo, da bi lahko med delovanjem naprave izbirali, katere rutine uporabljamo za generiranje ukazov, jih moramo klicati s preračunanimi skoki. Pred vstopom v glavno zanko izberemo rutino in njen naslov shranimo. Glavna zanka ta naslov uporabi za skok. Težava je v tem, da mora rutina vedeti, kam se vrniti, saj preračunani skoki ne uporabljajo sklada; uporabljeni mikrokontroler ne dovoljuje dostopa do njega. Zato je klic take rutine nekoliko bolj kompliciran.

Za izvedbo skoka si v podatkovnem pomnilniku pripravimo štiri registre (pravzaprav šest, če štejemo še tista dva, v katera smo shranili naslov izbranega generatorja ukazov): `addrhigh` in `addrlow` uporabimo za shranjevanje naslova za vrnitev, `argval` in `retval` pa uporabimo za podajanje argumenta in vračanje vrednosti. Makro za dalgi skok je potem

```
longjmp    MACRO   addrh, addrl
   LOCAL   here
   movlw   HIGH(here)
   movwf   addrhigh
   movlw   LOW(here)
   movwf   addrlow
   movf    addrh, W
   movwf   PCLATH
   movf    addrl, W
```

```
   movwf   PCL
here:
   ENDM
```

Ker zbirnik ve naslove vseh inštrukcij jih lahko vpisujemo v registre kot dobesedne vrednosti. Naslov, kjer se bo po vrnitvi nadaljevalo izvajanje, označimo s `here`. Na začetku makroja povemo, da je ta vrednost lokalna temu makroju. Simbol se bo zato evalviral v naslov šele, ko ga bo zbirnik ta makro vstavil tja, od koder ga bomo klicali. Ta naslov razbijemo na visoke in nizke bite in ga shranimo v oba pripravljena registra. Oba dela naslova rutine, ki jo kličemo, sta shranjena v dveh registrih katerih naslova podamo makroju kot argumenta. Vrednosti v njiju se vpišeta se v `PCLATH` in `PCL`, s čimer izvedemo skok.

Ko rutina konča, se izvajanje vrne nazaj z uporabo makroja

```
retjmp    MACRO
   movf   addrhigh, W
   movwf  PCLATH
   movf   addrlow, W
   movwf  PCL
   ENDM
```

Ta samo prepiše shranjeni naslov v `PCLATH` in `PCL`.

Ker za shranjevanje naslova uporabljamo samo dva registra (nismo implementirali sklada) seveda takšne rutine ne morejo klicati ena druge, odpade tudi rekurzija. Za nas to ni tako pomembno.

7.3 Uporaba tabel

Tipično se za hrambo podatkov uporablja tabela, implementirana kot rutina, ki za argument jemlje indeks elementa v registru `W`. Je oblike

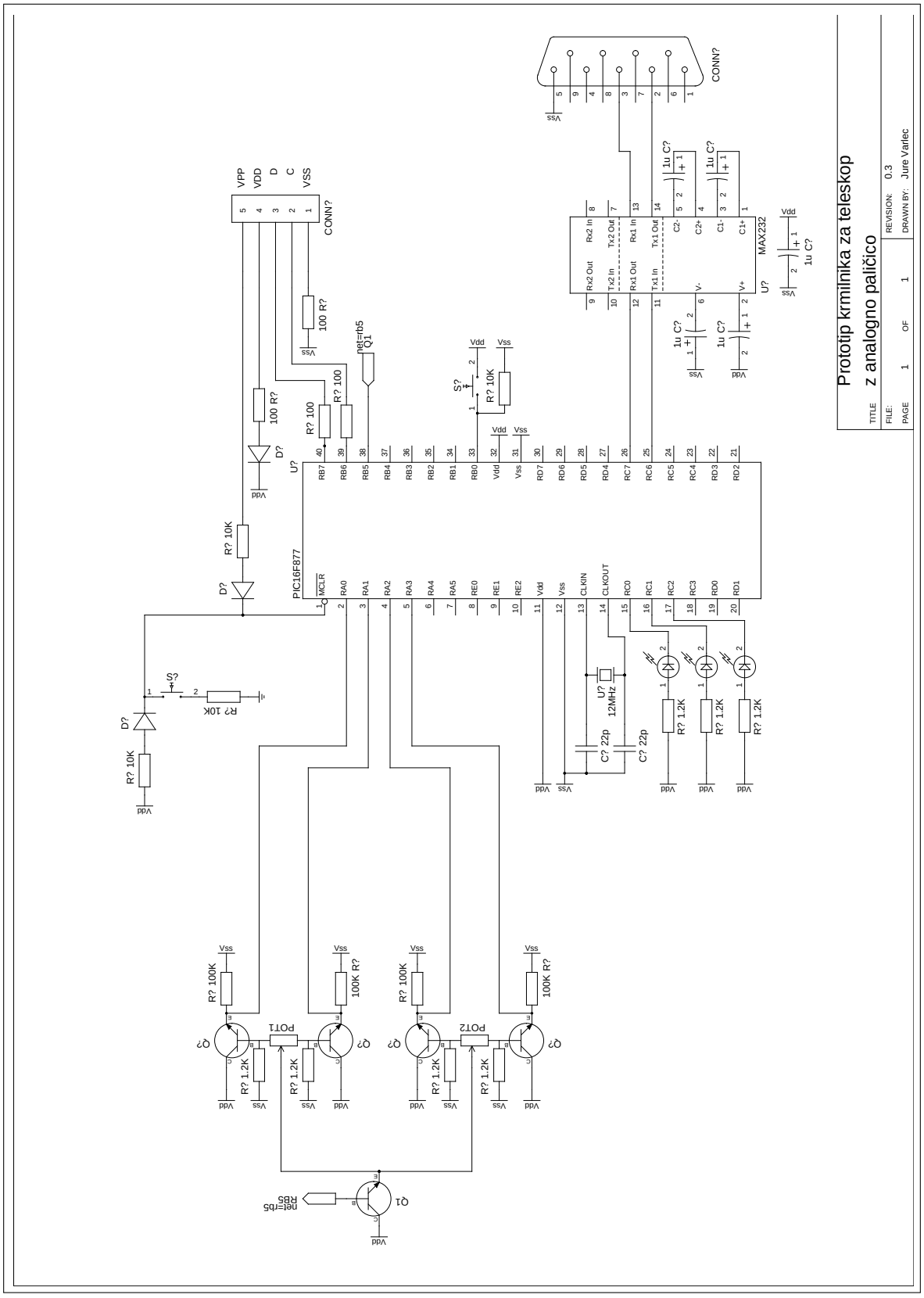
```
table:
   addwf   PCL, F
   retlw   'a'
   retlw   'b'
   retlw   'c'
   .
   .
   .
```

Ob klicu takšna rutina v registru `W` vrne vrednost z ustreznega indeksa. Namesto makroja `wrchar`, ki se ob vsakem klicu razpihne na tri inštrukcije, bi lahko generirali ukaze z zanko, ki bi prepisovala vrednosti iz tabele. Vendar pa ima uporabljeni mikrokontroler več kot dovolj programskega pomnilnika, da si lahko mirno privoščimo uporabo makroja. Preračunani skoki so tako omejeni na začetek rutine, zaradi česar nimamo težav zaradi morebitnih prekoračitev 256-besednih segmentov.

8 Zaključek

Trenutna različica programa implenmentira poleg človeku berljivih ukazov tudi ukaze po protokolu LX200. Preizkušena je bila na ustreznem stojalu teleskopa. Deluje zadovoljivo. Potrebna je le optimizacija predalčkov, ki določajo meje hitrosti, kajti čeprav je krmilnik že uporaben, je njegov namen vendarle večje udobje in natančnost usmerjanja teleskopa. Končni izdelek mora seveda podpirati tudi varčevanje z energijo in biti sposoben delovati na baterije. Vendar pa elektronsko vezje v pričujoči izvedbi tega ne podpira.

Pričujoče poročilo seveda ne vsebuje vseh pomožnih rutin in makrojev, ki jih program uporablja. Ti so opisani v izvorni kodi programa.



Slika 7: Shema elektronskega vezja.

```
        list p=16f877

        include "p16f877.inc"

        __CONFIG _BODEN_OFF & _WDT_OFF & _HS_OSC & _PWRTE_ON

;;; CONSTANTS
;;; ==========================================================================

        CONSTANT PORTC_STAT = b'00000111' ; Settings of PORTC

;;; MEMORY LAYOUT
;;; ==========================================================================

        ;; Shared memory region, 16 bytes long
        CBLOCK  0x70
        W_TEMP                  ; Context switch during interrupt
        STATUS_TEMP
        PCLATH_TEMP
        FSR_TEMP
        addrhigh                ; Address for longjmp
        addrlow
        argval                  ; Used to pass arguments to longjmp
        retval                  ; Used to pass return value from longjmp
        flags
        ENDC

        ;; Description of flag bits:
        ;;   0 - unused
        ;;   1 - transmission in progress
        ;;   2 - diagnostic mode
        ;;   3 - speed is 0
        ;;   4 - old speed was 0
        ;;   5 - speed has changed
        ;;   6 - deadzone
        ;;   7 - unused

        ;; Start of BANK0 general purpose memory
        CBLOCK  0x20
        ;; Delay macro
        cnt0
        cnt1
        cnt2
        ;; Blinking routine
        noblinks
        nocyc
        ledidx
        ;; Magnitude approximation
        mag
        ;; Speed and direction
        spdir                   ; high bits are direction, low bits are speed
        spdir_old
        spdaddr                 ; Addresses of the routine that determines
        diraddr                 ; the strings for the chosen mount
        spdaddrh
        diraddrh
        ;; Measurements
        x1
        x1_raw
        x2
```



```
        x2_raw
        y1
        y1_raw
        y2
        y2_raw
        center
        tmp                     ; Never rely on this after a subroutine call
        thresh                  ; Deadzone threshold
        bins:4                  ; Array for binning of measurements
        binmax
        ;; BIN2BCD
        BIN
        count
        huns
        tens
        ones
        ;; STRINGS:
        strpos                  ; Pointer to the current character in str
        strlen                  ; Length of the current string in str
        ENDC

        ;; Start of BANK1 general purpose memory
        CBLOCK  0xa0
        str                     ; Buffer for longer strings, 80 bytes free
        ENDC


;;; MACROS which are needed in the main body of code
;;; (Other macros are defined where they are needed)
;;; ==========================================================================

;;; Delay macro for 12MHz.
;;; Arguments: delay in microseconds; maximum value 2^24-1, minimum value 5.
;;; Uses registers cnt0, cnt1, cnt2.

delay           MACRO   musec
        LOCAL   musec2 = musec
musec2  -=      0x5
        movlw   (musec2 / 0x10000 + 0x1) % 0x100
        movwf   cnt2
musec2  %=      0x10000
        movlw   (musec2 / 0x100 + 0x1) % 0x100
        movwf   cnt1
musec2  %=      0x100
        movlw   (musec2 + 0x1) % 0x100
        movwf   cnt0

        decfsz  cnt0, F
        goto    $-1
        nop
        decfsz  cnt1, F
        goto    $-4
        nop
        decfsz  cnt2, F
        goto    $-7
        nop

        ENDM

;;; ==========================================================================
```

```
;;; Shortcut macro for measuring each axis.
;;; Argument: name of axis.

        ;; Adjust these constants to match actual wiring
        CONSTANT        X1=1, Y1=0, X2=3, Y2=2

adcmsr          MACRO   chan
        bsf     PORTB, 5        ; Enable current through the potentiometer
        IF (chan == 0)
        bcf     ADCON1, CHS0
        bcf     ADCON1, CHS1
        ELSE
        IF (chan == 1)
        bsf     ADCON1, CHS0
        bcf     ADCON1, CHS1
        ELSE
        IF (chan == 2)
        bcf     ADCON1, CHS0
        bsf     ADCON1, CHS1
        ELSE
        IF (chan == 3)
        bsf     ADCON1, CHS0
        bsf     ADCON1, CHS1
        ELSE
        ERROR   "Wrong channel selected!"
        ENDIF
        ENDIF
        ENDIF
        ENDIF
        call    adc_measure
        bcf     PORTB, 5
        ENDM

;;; =====================================================================

;;; Subtract one register from the other and put absolute value in W.
;;; Arguments: register addresses.

subabs          MACRO   reg1, reg2
        movf    reg1, W
        subwf   reg2, W
        btfsc   STATUS, C       ; skip the goto if reg2 < reg1
        goto    $+3
        xorlw   0xff
        addlw   0x1
        ENDM

;;; =====================================================================

;;; Toggle a LED in PORTC
;;; Argument: LED number

tglled          MACRO   led
        movf    PORTC, W
        xorlw   led
        movwf   PORTC
        ENDM

;;; =====================================================================

;;; Jump to a computed location, anywhere in memory.
```



```
;;; Arguments: registers that store the jump destination.
;;; Origin is stored in addrhigh and addrlow, so these calls don't stack.

longjmp         MACRO   addrh, addrl
        LOCAL   here
        movlw   HIGH(here)
        movwf   addrhigh
        movlw   LOW(here)
        movwf   addrlow
        movf    addrh, W
        movwf   PCLATH
        movf    addrl, W
        movwf   PCL
here:
        ENDM

;;; Return from a long jump

retjmp          MACRO
        movf    addrhigh, W
        movwf   PCLATH
        movf    addrlow, W
        movwf   PCL
        ENDM

;;; =====================================================================

;;; Copy a string. Assumes FSR points to the destination.
;;; Arguments: pos is the beginning of the string, len is its length.
;;; Puts len in W.

memcpy          MACRO   pos, len
        LOCAL   pos2 = pos
        WHILE   (pos2 < pos + len)
        movf    pos2, W
        movwf   INDF
        incf    FSR, F
pos2    +=      0x1
        ENDW
        movlw   len
        ENDM

;;; A bunch of macros to insert certain strings. They assume FSR points to
;;; the destination. They put the number of inserted characters in W.

;;; End-of-line
ins_eol         MACRO
        movlw   0xd
        movwf   INDF
        incf    FSR, F
        movlw   0xa
        movwf   INDF
        incf    FSR, F
        movlw   0x2
        ENDM

;;; Space
ins_spc         MACRO
        movlw   0x20
        movwf   INDF
        incf    FSR, F
```

```
        movlw   0x1
        ENDM

;;; Comma and space
ins_cmsp        MACRO
        movlw   0x2c
        movwf   INDF
        incf    FSR, F
        movlw   0x20
        movwf   INDF
        incf    FSR, F
        movlw   0x2
        ENDM

;;; Space-dash-space
ins_dashtab     MACRO
        movlw   0x20
        movwf   INDF
        incf    FSR, F
        movlw   0x2d
        movwf   INDF
        incf    FSR, F
        movlw   0x20
        movwf   INDF
        incf    FSR, F
        movlw   0x3
        ENDM

;;; Horizontal tab
ins_tab         MACRO
        movlw   0x9
        movwf   INDF
        incf    FSR, F
        movlw   0x1
        ENDM

;;; ========================================================================

;;; A fast way to calculate approximate magnitude of a 2-vector.
;;; Arguments: registers holding absolute values of vector components.
;;; Magnitude is returned in mag.
;;; Maximum possible value for the arguments is 185, maximum error is 14.
;;; It uses the Robertson formula with magic constant of 3/8, which is
;;; calculated by a separate routine.

magaprx         MACRO   mag1, mag2
        movf    mag1, W         ; Find the maximum value
        subwf   mag2, W
        btfss   STATUS, C
        goto    $+5
        movf    mag1, W         ; mag2 >= mag1
        call    magaprx_3_8
        addwf   mag2, W
        goto    $+4
        movf    mag2, W         ; mag1 > mag2
        call    magaprx_3_8
        addwf   mag1, W
        movwf   mag
        ENDM

;;; ========================================================================
```



```

;;; The following macro is used in applying deadzone to the measurement.
;;; It takes raw measurement in W and writes it to arg.
;;; If the change between W and the old arg_raw is larger than thresh,
;;; it sets the appropriate flag.

deadzn          MACRO   arg
        movwf   arg
        subabs  arg, arg+1
        subwf   thresh, W
        btfss   STATUS, C
        bsf     flags, 6
        ENDM

;;; A shortcut to shuffle things around after checking for deadzone.
;;; Use this to copy raw measurements from arg to arg_raw and
;;; subtract the center offset.

ddznshf         MACRO   arg
        movf    arg, W
        movwf   arg+1
        subabs  arg, center
        movwf   arg
        ENDM

;;; ========================================================================

;;; Swap two registers without temporary storage

swapff          MACRO   reg1, reg2
        movf    reg2, W
        xorwf   reg1, F
        xorwf   reg1, W
        movwf   reg2
        xorwf   reg1, F
        ENDM

;;; CODE
;;; ========================================================================

        ORG     0x0
        goto    start

;;; INTERRUPTS
        ORG     0x4
irq:
        ;; Save state on interrupt
        movwf   W_TEMP
        swapf   STATUS, W
        clrf    STATUS
        movwf   STATUS_TEMP
        movf    PCLATH, W
        movwf   PCLATH_TEMP
        clrf    PCLATH
        movf    FSR, W
        movwf   FSR_TEMP

        ;; Button press? Disable the interrupt, which will start calibration.
        btfss   INTCON, INTF
        goto    txirq
        bcf     INTCON, INTF
```

```
        bcf     INTCON, INTE
        bcf     INTCON, INTF
        goto    endirq


;;; USART code. It fils TXREG from strpos, which should be in BANK1 general
;;; purpose area, i.e. str < strpos < str+0x80. It stops sending when it
;;; reaches str+strlen.
txirq:
        ;; Transmit register empty?
        bsf     STATUS, RP0
        btfss   PIR1, TXIF
        goto    endirq
        bcf     STATUS, RP0

        movf    strpos, W
        movwf   FSR
        movf    INDF, W
        movwf   TXREG
        incf    strpos, F

        ;; Stop transmitting when finished
        movlw   str
        addwf   strlen, W
        xorwf   strpos, W
        btfss   STATUS, Z
        goto    endirq
        bsf     STATUS, RP0
        bcf     PIE1, TXIE

endirq:
        ;; Restore state before returning from interrupt
        movf    FSR_TEMP, W
        movwf   FSR
        movf    PCLATH_TEMP, W
        movwf   PCLATH
        swapf   STATUS_TEMP, W
        movwf   STATUS
        swapf   W_TEMP, F
        swapf   W_TEMP, W
        retfie

;;; BODY
;;; =====================================================================

start:
        bcf     STATUS, RP0
        bcf     STATUS, RP1

        bsf     STATUS, RP0
        movlw   0x02            ; RA0-RA3, RA5 as analog
        movwf   ADCON1
        movlw   b'11011111'     ; RB5 as output
        movwf   TRISB
        movlw   b'11111000'     ; Set output pins for the LEDs
        movwf   TRISC
        bcf     STATUS, RP0
        movlw   PORTC_STAT      ; Enable RC0-RC2 (turn LED off)
        movwf   PORTC
        clrf    PORTB           ; Disable the joystick
        movlw   0x80            ; ADC clock
```



```
        movwf   ADCON0

        ;; Setup serial port
        bsf     STATUS, RP0
        bcf     TXSTA, SYNC
        bsf     TXSTA, BRGH
        bcf     TXSTA, TX9
        bsf     TXSTA, TXEN
        ;movlw  d'12'            ; Baud rate 57.6k at 12MHz, precision 0.1%
        movlw   d'77'           ; Baud rate 9.6k at 12MHz, precision 0.16%
        movwf   SPBRG
        bcf     STATUS, RP0
        bsf     RCSTA, SPEN

        clrf    flags

        ;; Enable interrupts
        bsf     INTCON, INTE
        bsf     INTCON, PEIE
        bsf     INTCON, GIE

        ;; Signal powerup
        movlw   0x2
        call    blink

        ;; Enter diagnostic mode if INT was pressed during powerup signal
        btfsc   INTCON, INTE
        goto    wait_calib
        bsf     flags, 2
        delay   d'300'
        bsf     INTCON, INTE

wait_calib:
        ;; Wait for calibration
        btfsc   flags, 0
        goto    done_calib
        btfss   INTCON, INTE
        call    calibrate
        tglled  0x1
        delay   d'500000'
        goto    wait_calib

done_calib:
        ;; Clear LEDs
        movlw   PORTC_STAT
        movwf   PORTC

        ;; Decide what to do next
        btfsc   flags, 2
        goto    diagnostic_main

        ;; Choose the address of the routine that will generate strings
        ;; to control the mount. Will use a switch to decide between them.
        ;; Symbols chosen_{dir,spd} have to be declared as variables because
        ;; the addresses they store become defined in second pass af assembly.

        VARIABLE        chosen_dir = human_readable_dir
        VARIABLE        chosen_spd = human_readable_spd
        ;VARIABLE       chosen_dir = lx200_dir
        ;VARIABLE       chosen_spd = lx200_spd
        movlw   LOW(chosen_dir)
```

```
        movwf   diraddr
        movlw   HIGH(chosen_dir)
        movwf   diraddrh
        movlw   LOW(chosen_spd)
        movwf   spdaddr
        movlw   HIGH(chosen_spd)
        movwf   spdaddrh

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

;;; MAIN LOOP

        clrf    spdir
        clrf    x1_raw
        clrf    x2_raw
        clrf    y1_raw
        clrf    y2_raw

        ;; First, jump right ahead and order the mount to stop.
        clrf    strlen
        movlw   str
        movwf   strpos
        movwf   FSR
        bsf     flags, 5
        goto    main_printspd

main:
        btfss   INTCON, INTE
        call    calibrate

        movf    spdir, W
        movwf   spdir_old
        clrf    spdir
        bcf     flags, 3
        bcf     flags, 4
        bcf     flags, 5
        bcf     flags, 6

main_measureloop:
        btfss   INTCON, INTE
        call    calibrate

        delay   d'100000'
        adcmsr  X1
        deadzn  x1
        adcmsr  X2
        deadzn  x2
        adcmsr  Y1
        deadzn  y1
        adcmsr  Y2
        deadzn  y2

        ;; See if any measurement is outside deadzone threshold.
        ;; If not, measure again. Else, backup the raw values and
        ;; subtract center offset.
        btfss   flags, 6
        goto    main_measureloop
        ddznshf x1
        ddznshf x2
        ddznshf y1
        ddznshf y2
```



```

        ;; Find the largest measurement for each axis; they determine direction
        ;; Intermediate result stored in spdir<4,5>
        movf    x2, W
        subwf   x1, W
        btfss   STATUS, C
        bsf     spdir, 4        ; Mark x2 >= x1
        movf    y2, W
        subwf   y1, W
        btfss   STATUS, C
        bsf     spdir, 5        ; Mark y2 >= y1

        ;; Make sure that {x,y}1 >= {x,y}2
        btfss   spdir, 4
        goto    $+6             ; swapff is a 5-instruction macro
        swapff  x1, x2
        btfss   spdir, 5
        goto    $+6
        swapff  y1, y2

        ;; Determine the speed using the bins array
        magaprx x1, y1
        call    binning
        iorwf   spdir, F

        ;; Did the speed change?
        movf    spdir, W
        andlw   0xf
        btfsc   STATUS, Z
        bsf     flags, 3        ; Mark that speed is zero
        movwf   tmp
        movf    spdir_old, W
        andlw   0xf
        btfsc   STATUS, Z
        bsf     flags, 4        ; Mark that old speed was zero
        xorwf   tmp, W
        btfss   STATUS, Z
        bsf     flags, 5

        ;; Prepare to overwrite the string
        call    wait_tx
        clrf    strlen
        movlw   str
        movwf   strpos
        movwf   FSR

        ;; Determine direction.
        ;; For possible values of spdir<4-7> see the description of
        ;; string routines at the end of this file.
        btfsc   flags, 3
        goto    main_printspd   ; If speed is zero skip this

        movf    y1, W
        movwf   mag
        call    binning
        andlw   0xff
        btfss   STATUS, Z
        goto    $+4
        bsf     spdir, 6        ; If y = 0 we have E or W
        bcf     spdir, 5
        goto    main_printdir
```

```
611
612         movf    x1, W
613         movwf   mag
614         call    binning
615         andlw   0xff
616         btfss   STATUS, Z
617         goto    $+6
618         bsf     spdir, 6        ; If x = 0 we have N or S
619         bcf     spdir, 4
620         btfsc   spdir, 5
621         bsf     spdir, 4
622         bsf     spdir, 5
623
624         ;; Did the direction change?
625 main_printdir:
626         btfsc   flags, 4        ; If old speed was zero always print direction
627         goto    main_printdir2
628         movf    spdir, W
629         andlw   0xf0
630         movwf   tmp
631         movf    spdir_old, W
632         andlw   0xf0
633         xorwf   tmp, W
634         btfsc   STATUS, Z
635         goto    main_printspd
636
637         ;; Print direction.
638 main_printdir2:
639         swapf   spdir, W
640         andlw   0xf
641         movwf   argval
642         longjmp diraddrh, diraddr
643         movf    retval, W
644         addwf   strlen, F
645         call    start_tx
646
647         ;; Print speed
648 main_printspd:
649         btfss   flags, 5
650         goto    main
651         movf    spdir, W
652         andlw   0xf
653         movwf   argval
654         longjmp spdaddrh, spdaddr
655         movf    retval, W
656         addwf   strlen, F
657         call    start_tx
658
659         goto    main
660
661 ;;; ------------------------------------------------------------------------
662
663 ;;; Diagnostic part. It prints measured values from the potentiometer with
664 ;;; calibration center subtracted, approximate magnitudes as calculated
665 ;;; by magapprox, measured center and extreme position from the calibration
666 ;;; routine.
667 ;;; OUTPUT FORMAT:
668 ;;; x1, x2 - y1, y2    abs(x1^2+y1^2), abs(x2^2+y2^2)    cntr, extrm    bins
669
670 diagnostic_main:
671         movlw   str
```



```
672         movwf   strpos
673         movwf   FSR
674         clrf    strlen
675
676         ;; x1, x2 - y1, y2
677         adcmsr  X1
678         movwf   x1
679         subabs  x1, center
680         movwf   x1
681         movwf   BIN
682         call    BIN2BCD
683
684         memcpy  huns, 0x3
685         addwf   strlen, F
686         ins_cmsp
687         addwf   strlen, F
688
689         adcmsr  X2
690         movwf   x2
691         subabs  x2, center
692         movwf   x2
693         movwf   BIN
694         call    BIN2BCD
695
696         memcpy  huns, 0x3
697         addwf   strlen, F
698         ins_dashtab
699         addwf   strlen, F
700
701         adcmsr  Y1
702         movwf   y1
703         subabs  y1, center
704         movwf   y1
705         movwf   BIN
706         call    BIN2BCD
707
708         memcpy  huns, 0x3
709         addwf   strlen, F
710         ins_cmsp
711         addwf   strlen, F
712
713         adcmsr  Y2
714         movwf   y2
715         subabs  y2, center
716         movwf   y2
717         movwf   BIN
718         call    BIN2BCD
719
720         memcpy  huns, 0x3
721         addwf   strlen, F
722         ins_tab
723         addwf   strlen, F
724
725         ;; abs(x1^2+y1^2), abs(x2^2+y2^2)
726         magaprx x1, y1
727         movwf   BIN
728         call    BIN2BCD
729         memcpy  huns, 0x3
730         addwf   strlen, F
731         ins_cmsp
732         addwf   strlen, F
```

```
733
734         magaprx x2, y2
735         movwf   BIN
736         call    BIN2BCD
737         memcpy  huns, 0x3
738         addwf   strlen, F
739         ins_tab
740         addwf   strlen, F
741
742         ;; cntr, extrm
743         movf    center, W
744         movwf   BIN
745         call    BIN2BCD
746
747         memcpy  huns, 0x3
748         addwf   strlen, F
749         ins_cmsp
750         addwf   strlen, F
751
752         movf    binmax, W
753         movwf   BIN
754         call    BIN2BCD
755
756         memcpy  huns, 0x3
757         addwf   strlen, F
758         ins_tab
759         addwf   strlen, F
760
761         clrf    tmp
762
763         ;; bins
764 diagnostic_bins:
765         movf    tmp, W
766         addlw   bins
767         movwf   FSR
768         movf    INDF, W
769         movwf   BIN
770         incf    tmp, F
771
772         call    BIN2BCD
773         movf    strlen, W
774         addlw   str
775         movwf   FSR
776         memcpy  huns, 0x3
777         addwf   strlen, F
778         ins_spc
779         addwf   strlen, F
780
781         movlw   0x4
782         xorwf   tmp, W
783         btfss   STATUS, Z
784         goto    diagnostic_bins
785
786         ins_eol
787         addwf   strlen, F
788
789         call    start_tx
790
791         movlw   0x1
792         call    blink
793
```



```
794         call    wait_tx
795         btfss   INTCON, INTE
796         call    calibrate
797
798         goto    diagnostic_main
799
800
801 ;;; ROUTINES
802 ;;; =====================================================================
803
804 ;;; Wait for USART transmission to complete, as reported by the TXIE flag.
805 ;;; Does NOT check for physical completion (the TRMT flag).
806
807 wait_tx:
808         bsf     STATUS, RP0
809         btfsc   PIE1, TXIE
810         goto    $-1
811         bcf     STATUS, RP0
812         return
813
814 ;;; Start transmission of a string.
815 ;;; Arguments: strpos is the first character, strlen is the string length.
816 ;;; Transmission is handled via interrupts.
817
818 start_tx:
819         bsf     STATUS, RP0
820         bsf     PIE1, TXIE
821         bcf     STATUS, RP0
822         return
823
824 ;;; =====================================================================
825
826 ;;; ADC measurement
827 ;;; Select AD channel before calling
828 ;;; Assumes BANK 0, left justified ADRES
829 ;;; Returns with MSB in W
830 ;;; Wait at least 3 instructions (at 12MHz) before calling again
831
832 adc_measure:
833         bsf     ADCON0, ADON
834         delay   d'20'           ; Minimum wait calculated from datasheet
835         bsf     ADCON0, GO      ; Start conversion
836         btfsc   ADCON0, GO      ; Conversion takes minimum 32usec at 12MHz
837         goto    $-1
838         movf    ADRESH, W
839         bcf     ADCON0, ADON
840         return
841
842 ;;; =====================================================================
843
844 ;;; Poll RB0 with 50ms debounce
845
846 rb0deb:
847         btfss   PORTB, 0
848         goto    $-1
849         delay   d'50000'
850         btfss   PORTB, 0
851         goto    rb0deb
852         return
853
854 ;;; Calibration
```

```
855
856 calibrate:
857         movlw   PORTC_STAT
858         movwf   PORTC
859
860         ;; Use the red LED and ask the user to first measure the center and
861         ;; then one of the extremal positions (up, down, left or right).
862         ;; He orders the measurement using RB0.
863
864         ;; First, measure the center values and average them
865         clrf    center
866         clrf    tmp
867         delay   d'300000'
868         tglled  0x2
869         call    rb0deb
870         adcmsr  X1
871         addwf   center, F
872         btfsc   STATUS, C
873         incf    tmp, F
874         adcmsr  X2
875         addwf   center, F
876         btfsc   STATUS, C
877         incf    tmp, F
878         adcmsr  Y1
879         addwf   center, F
880         btfsc   STATUS, C
881         incf    tmp, F
882         adcmsr  Y2
883         addwf   center, F
884         btfsc   STATUS, C
885         incf    tmp, F
886         rrf     tmp, F          ; Carry is used in rrf automatically
887         rrf     center, F
888         rrf     tmp, F
889         rrf     center, F
890
891         ;; Do the measurement and find the maximum value
892         delay   d'300000'
893         tglled  0x2
894         call    rb0deb
895         adcmsr  X1
896         movwf   binmax
897         adcmsr  X2
898         movwf   tmp
899         subwf   binmax, W
900         btfsc   STATUS, C
901         goto    $+3
902         movf    tmp, W
903         movwf   binmax
904         adcmsr  Y1
905         movwf   tmp
906         subwf   binmax, W
907         btfsc   STATUS, C
908         goto    $+3
909         movf    tmp, W
910         movwf   binmax
911         adcmsr  Y2
912         movwf   tmp
913         subwf   binmax, W
914         btfsc   STATUS, C
915         goto    $+3
```



```
916         movf    tmp, W
917         movwf   binmax
918
919         ;; Calculate exponential function to bin measurements
920         movlw   bins+3
921         movwf   FSR
922         subabs  binmax, center
923         movwf   INDF
924         bcf     STATUS, C       ; Carry is used in rrf, so zero it out
925         rrf     INDF, W
926         decf    FSR, F
927         movwf   INDF
928         movlw   thresh          ; thresh is placed before bins
929         xorwf   FSR, W
930         btfss   STATUS, Z
931         goto    $-7
932
933         delay   d'1000000'
934         bsf     flags, 0
935         bcf     INTCON, INTF
936         bsf     INTCON, INTE
937         return
938
939 ;;; ========================================================================
940
941 ;;; Blinking routine. It cycles three LEDs.
942 ;;; Arguments: W is the number of LED cycles
943
944 blink:
945         movwf   nocyc
946         clrf    ledidx
947         bsf     STATUS, C       ; Used by rlf, will start cycle with LED 0
948
949 blink_cyc:
950         movlw   0x6             ; Cycle length
951         movwf   noblinks
952
953 blink_reloop:
954         delay   d'166666'
955
956         ;; Toggle the chosen LED
957         movlw   0x1
958         rlf     ledidx, F
959         btfsc   ledidx, 3
960         movwf   ledidx
961         movf    PORTC, W
962         xorwf   ledidx, W
963         movwf   PORTC
964
965         decfsz  noblinks, F
966         goto    blink_reloop
967         decfsz  nocyc, F
968         goto    blink_cyc
969
970         return
971
972 ;;; ========================================================================
973
974 ;;; A routine to multiply a number by 3/8, used by magaprx.
975
976 magaprx_3_8:
```

```
        movwf   mag
        bcf     STATUS, C       ; Divide by 8
        rrf     mag, F
        bcf     STATUS, C
        rrf     mag, F
        bcf     STATUS, C
        rrf     mag, F
        movf    mag, W          ; Multiply by 3
        addwf   mag, W
        addwf   mag, W
        return

;;; ========================================================================

;;; Find which bin the given value falls in.
;;; Arguments: mag is the value to be compared with the bins array.
;;; Result is given in W.

binning:
        VARIABLE        binidx = 0
        WHILE (binidx < 4)
        movf    mag, W
        subwf   bins+binidx, W
        btfsc   STATUS, C
        retlw   binidx
binidx  +=      1
        ENDW
        retlw   0x3

;;; ========================================================================

;;; Convert 8-bit binary to BCD
;;; Adapted from:
;;;    http://www.piclist.com/techref/microchip/math/radix/b2a-8b3d-ab.htm
;;; Argument: BIN
;;; Returns: huns, tens, ones
;;; uses ADD-3 algoerthm

BIN2BCD:
        movlw d'8'
        movwf count
        clrf huns
        clrf tens
        clrf ones

BCDADD3:
        movlw d'5'
        subwf huns, 0
        btfsc STATUS, C
        CALL ADD3HUNS

        movlw d'5'
        subwf tens, 0
        btfsc STATUS, C
        CALL ADD3TENS

        movlw d'5'
        subwf ones, 0
        btfsc STATUS, C
        CALL ADD3ONES

```



```
        decf count, 1
        bcf STATUS, C
        rlf BIN, 1
        rlf ones, 1
        btfsc ones,4 ;
        CALL CARRYONES
        rlf tens, 1

        btfsc tens,4 ;
        CALL CARRYTENS
        rlf huns,1
        bcf STATUS, C

        movf count, 0
        btfss STATUS, Z
        GOTO BCDADD3

        movf huns, 0 ; add ASCII Offset
        addlw h'30'
        movwf huns

        movf tens, 0 ; add ASCII Offset
        addlw h'30'
        movwf tens

        movf ones, 0 ; add ASCII Offset
        addlw h'30'
        movwf ones

        RETURN

ADD3HUNS:
        movlw d'3'
        addwf huns,1

        RETURN

ADD3TENS:
        movlw d'3'
        addwf tens,1

        RETURN

ADD3ONES:
        movlw d'3'
        addwf ones,1

        RETURN

CARRYONES:
        bcf ones, 4
        bsf STATUS, C
        RETURN

CARRYTENS:
        bcf tens, 4
        bsf STATUS, C
        RETURN

;;; ========================================================================

```

```
;;; Check whether we have crossed the page boundary.

        IF ($ >= 0x800)
            ERROR "Core routines larger than a page!"
        ENDIF

;;; ========================================================================

;;; SPEED AND DIRECTION STRINGS
;;; These routines are computed GOTOs that append the requested string to
;;; wherever FSR points. They return the number of written bytes in retval.
;;; Because of their size, these routines are placed in PAGE 1 so that they
;;; don't cross the page boundary. Also, they are 256-word aligned to make
;;; computed GOTOs easier.

;;; Speed strings are indexed in ascending order.

;;; Direction strings are indexed according to the following table,
;;; with indices in 4-base:
;;; 00 - NW    02 - SW   10 - W    12 - N
;;; 01 - NE    03 - SE   11 - E    13 - S

;;; Call these subroutines with index in argval.
;;; They return the number of characters in retval.

;;; ========================================================================

;;; Macro that writes a single character

wrchar          MACRO   char
        movlw   char
        movwf   INDF
        incf    FSR, F
        ENDM

;;; ========================================================================

;;; Human readable

        ORG     0x800
human_readable_dir:
        wrchar  'D'
        wrchar  'i'
        wrchar  'r'
        wrchar  'e'
        wrchar  'c'
        wrchar  't'
        wrchar  'i'
        wrchar  'o'
        wrchar  'n'
        wrchar  ' '

        movf    argval, W
        addwf   PCL, F
        goto    human_readable_dir_NW
        goto    human_readable_dir_NE
        goto    human_readable_dir_SW
        goto    human_readable_dir_SE
        goto    human_readable_dir_W
        goto    human_readable_dir_E
        goto    human_readable_dir_N
```



```
        goto    human_readable_dir_S

human_readable_dir_NW:
        wrchar  'N'
        wrchar  'W'
        ins_eol
        movlw   0xc+0x2
        movwf   retval
        retjmp

human_readable_dir_NE:
        wrchar  'N'
        wrchar  'E'
        ins_eol
        movlw   0xc+0x2
        movwf   retval
        retjmp

human_readable_dir_SW:
        wrchar  'S'
        wrchar  'W'
        ins_eol
        movlw   0xc+0x2
        movwf   retval
        retjmp

human_readable_dir_SE:
        wrchar  'S'
        wrchar  'E'
        ins_eol
        movlw   0xc+0x2
        movwf   retval
        retjmp

human_readable_dir_E:
        wrchar  'E'
        ins_eol
        movlw   0xc+0x1
        movwf   retval
        retjmp

human_readable_dir_W:
        wrchar  'W'
        ins_eol
        movlw   0xc+0x1
        movwf   retval
        retjmp

human_readable_dir_S:
        wrchar  'S'
        ins_eol
        movlw   0xc+0x1
        movwf   retval
        retjmp

human_readable_dir_N:
        wrchar  'N'
        ins_eol
        movlw   0xc+0x1
        movwf   retval
        retjmp
```

```
1221
1222 human_readable_spd:
1223         wrchar  'S'
1224         wrchar  'p'
1225         wrchar  'e'
1226         wrchar  'e'
1227         wrchar  'd'
1228         wrchar  ' '
1229
1230         movf    argval, W
1231         addlw   0x30
1232         movwf   INDF
1233         incf    FSR, F
1234         ins_eol
1235         movlw   0x9
1236         movwf   retval
1237         retjmp
1238
1239         ;; Sanity check
1240         IF (HIGH($) != HIGH(human_readable_dir))
1241             ERROR "Table crosses 256-word boundary!"
1242         ENDIF
1243
1244 ;;; ==========================================================================
1245
1246 ;;; LX200 protocol
1247 ;;; This one crosses 256-word boundary, but the both computed gotos are
1248 ;;; at the beginning, so it's OK.
1249
1250         ORG     0x900
1251 lx200_spd:
1252         wrchar  ':'
1253         movf    argval, W
1254         addwf   PCL, F
1255         goto    lx200_spd0
1256         goto    lx200_spd1
1257         goto    lx200_spd2
1258         goto    lx200_spd3
1259
1260 lx200_dir:
1261         movf    argval, W
1262         addwf   PCL, F
1263         goto    lx200_dir_nw
1264         goto    lx200_dir_ne
1265         goto    lx200_dir_sw
1266         goto    lx200_dir_se
1267         goto    lx200_dir_w
1268         goto    lx200_dir_e
1269         goto    lx200_dir_n
1270         goto    lx200_dir_s
1271
1272 lx200_dir_e:
1273         wrchar  ':'
1274         wrchar  'Q'
1275         wrchar  'w'
1276         wrchar  '#'
1277         wrchar  ':'
1278         wrchar  'Q'
1279         wrchar  'n'
1280         wrchar  '#'
1281         wrchar  ':'
```



```
1282         wrchar  'Q'
1283         wrchar  's'
1284         wrchar  '#'
1285         wrchar  ':'
1286         wrchar  'M'
1287         wrchar  'e'
1288         wrchar  '#'
1289         movlw   0x10
1290         goto    lx200_dir_end
1291 lx200_dir_w:
1292         wrchar  ':'
1293         wrchar  'Q'
1294         wrchar  'e'
1295         wrchar  '#'
1296         wrchar  ':'
1297         wrchar  'Q'
1298         wrchar  'n'
1299         wrchar  '#'
1300         wrchar  ':'
1301         wrchar  'Q'
1302         wrchar  's'
1303         wrchar  '#'
1304         wrchar  ':'
1305         wrchar  'M'
1306         wrchar  'w'
1307         wrchar  '#'
1308         movlw   0x10
1309         goto    lx200_dir_end
1310 lx200_dir_n:
1311         wrchar  ':'
1312         wrchar  'Q'
1313         wrchar  'w'
1314         wrchar  '#'
1315         wrchar  ':'
1316         wrchar  'Q'
1317         wrchar  'e'
1318         wrchar  '#'
1319         wrchar  ':'
1320         wrchar  'Q'
1321         wrchar  's'
1322         wrchar  '#'
1323         wrchar  ':'
1324         wrchar  'M'
1325         wrchar  'n'
1326         wrchar  '#'
1327         movlw   0x10
1328         goto    lx200_dir_end
1329 lx200_dir_s:
1330         wrchar  ':'
1331         wrchar  'Q'
1332         wrchar  'w'
1333         wrchar  '#'
1334         wrchar  ':'
1335         wrchar  'Q'
1336         wrchar  'n'
1337         wrchar  '#'
1338         wrchar  ':'
1339         wrchar  'Q'
1340         wrchar  'e'
1341         wrchar  '#'
1342         wrchar  ':'
```

```
        wrchar  'M'
        wrchar  's'
        wrchar  '#'
        movlw   0x10
        goto    lx200_dir_end
lx200_dir_nw:
        wrchar  ':'
        wrchar  'Q'
        wrchar  'e'
        wrchar  '#'
        wrchar  ':'
        wrchar  'Q'
        wrchar  's'
        wrchar  '#'
        wrchar  ':'
        wrchar  'M'
        wrchar  'n'
        wrchar  '#'
        wrchar  ':'
        wrchar  'M'
        wrchar  'w'
        wrchar  '#'
        movlw   0x10
        goto    lx200_dir_end
lx200_dir_ne:
        wrchar  ':'
        wrchar  'Q'
        wrchar  'w'
        wrchar  '#'
        wrchar  ':'
        wrchar  'Q'
        wrchar  's'
        wrchar  '#'
        wrchar  ':'
        wrchar  'M'
        wrchar  'n'
        wrchar  '#'
        wrchar  ':'
        wrchar  'M'
        wrchar  'e'
        wrchar  '#'
        movlw   0x10
        goto    lx200_dir_end
lx200_dir_sw:
        wrchar  ':'
        wrchar  'Q'
        wrchar  'e'
        wrchar  '#'
        wrchar  ':'
        wrchar  'Q'
        wrchar  'n'
        wrchar  '#'
        wrchar  ':'
        wrchar  'M'
        wrchar  's'
        wrchar  '#'
        wrchar  ':'
        wrchar  'M'
        wrchar  'w'
        wrchar  '#'
        movlw   0x10
```



```
        goto    lx200_dir_end
lx200_dir_se:
        wrchar  ':'
        wrchar  'Q'
        wrchar  'w'
        wrchar  '#'
        wrchar  ':'
        wrchar  'Q'
        wrchar  'n'
        wrchar  '#'
        wrchar  ':'
        wrchar  'M'
        wrchar  's'
        wrchar  '#'
        wrchar  ':'
        wrchar  'M'
        wrchar  'e'
        wrchar  '#'
        movlw   0x10
lx200_dir_end:
        movwf   retval
        retjmp

lx200_spd0:
        wrchar  'Q'
        wrchar  'e'
        wrchar  '#'
        wrchar  ':'
        wrchar  'Q'
        wrchar  'w'
        wrchar  '#'
        wrchar  ':'
        wrchar  'Q'
        wrchar  'n'
        wrchar  '#'
        wrchar  ':'
        wrchar  'Q'
        wrchar  's'
        wrchar  '#'
        wrchar  ':'
        wrchar  'Q'
        wrchar  '#'
        movlw   0x13
        goto    lx200_spd_end
lx200_spd1:
        wrchar  'R'
        wrchar  'C'
        wrchar  '#'
        movlw   0x4
        goto    lx200_spd_end
lx200_spd2:
        wrchar  'R'
        wrchar  'M'
        wrchar  '#'
        movlw   0x4
        goto    lx200_spd_end
lx200_spd3:
        wrchar  'R'
        wrchar  'S'
        wrchar  '#'
        movlw   0x4
```

```
lx200_spd_end:
        movwf   retval
        retjmp


        end
```