NTNU/IES, RKS

Mikroprosessorsystemer
Labøving 11: Assembler

Denne øvingen skal gjøre dere kjent med de forskjellige assembly instruksjonene som finns for flytting av data i Atmel AVR.

Atmel Studio assembly prosjekt

a) Start opp Atmel Studio.
b) Lag et nytt prosjekt. Velg AVR Assembler (ikke AVR GCC)
c) Gi prosjektet et navn, f.eks. labovingX_DittNavn. Pass på at navnet ikke inneholder noen spesialtegn eller norske bokstaver. UNC nettverksstier fungerer ikke (dvs. :\\dinkatalog\..), men f.eks. J:\brukernavn\Mine Dokumenter\uPsys kan fungere så lenge det ikke er ulumskheter som nevn ovenfor. Prosjektet kan også ligge på c:\temp\ på den lokale maskinen.
d) Trykk "next" og velg ATmega328P som device.
e) Trykk "finish".

Struktur assemblerprogram.

Atmel Studio lager nå automatisk et skjelett for assemblerprogrammet. Et assembler program inneholder på samme måte som C en initial konfigurering og en uendelig løkke:

```
//     initialization

//     eternal loop
loop:
//     application code
    rjmp loop
```

Assemblerprogram bruker ofte semikolon ; for å starte kommentarer, men AVR assembleren takler fint C-type kommentarer med `//` og `/**/`

Databladet viser strukturen for AVR kjernen og hvilke assemblerinstruksjoner den aktuelle kretsen støtter og «AVR instructionset» beskriver all instruksjoner.

Dataoverføringsfunksjoner.

I assemblerprogrammering blir det mye henting og flytting av data. All data som skal regnes på/endres må inn i ett av de 32 generelle registrene, og eventuelt resultat skrives ut til i/o-registre/minne.

Den enkleste instruksjonen er såkalt immediate adressering, ldi. For å laste inn for eksempel 99 inn i register 16 skriver vi:

```
    ldi r16, 99
```

Legg merke til at destinasjon kommer først og kilde til sist. Dette gjelder alle instruksjoner.

En programfil trenger ikke inneholde mere enn dette, men legg inn en header / introduksjon til filen først som er en kommentar med info om hva den er, dato og hvem som har skrevet koden på samme måte som i c-programmering.

Vi skal nå simulere programmet. Pass på at programmet slutter med en uendelig løkke:

```
loop:
//     application code
    rjmp loop
```

Dette gjør vi for at simulering av foregående skal gå greit, ellers mister simulatoren synk med programmet vårt og vi får ikke oppdatering av verdier i simulatoren.

a) Start simulering: Debug -> Start Debugging and Break.
b) Kjør ldi instruksjonen: Debug -> Step Into.
c) Se på innholdet i r16 vha. Processor View – utvid registre.
d) Verifiser at innholdet er som forventet.

Memory 1 vil gi et vindu hvor dere kan se innholdet i de forskjellige minnene.

For å kunne skrive register og bitnavn må vi inkludere en definisjonsfil:

```
#include <m328pdef.inc>
```

Atmel Studio gjør nå dette automatisk, men ved assemblering (kompilering) ser vi at filen bli inkludert i prosjektet.

Oppgave 1 - dataoverføring

Skriv assemblykode for følgende adresseringsmåter, simuler og verifiser at resultatene er som forventet. Register R00-R31 finnes i «Processor status»-vinduet. Innholdet i SRAM finner du under data IRAM i et «Memory»-vindu.

a) Immediate, last verdien 3 inn i R17 og 5 inn i r18
b) Register Direct. Kopier innholdet i R17 til R0.
c) I/O Direct. Kopier innholdet i Statusregisteret til R0.
d) I/O Direct. Kopier innholdet i R17 til Statusregisteret.
e) Data Direct. Lagre verdien i R17 til første minneposisjon i SRAM.
f) Data Direct. Last verdien i første minneposisjon i SRAM inn i R0.
g) Data Indirect. Lagre verdien i R16 på adressen til Z-pekeren med postinkrement.
h) Data Indirect. Last verdien i minneposisjonen Z peker på inn i R0.

Oppgave 2 – addisjon.

For å legge sammen to tall må de først inn i to generelle registre, dvs. ldi. Oversikt over aritmetisk-logiske instruksjoner finner du i databladet og i «AVR instrucion set»

Skriv assemblykode for følgende adresseringsmåter, simuler og verifiser at resultatene er som forventet.

127+129
80 + 48
128 + 40

Addisjon av større tall enn 8-bit må gjøres på samme måte som ved manuell regning. Man må ta med mente (carry) videre i beregningene. Eksempel 25+56:

```
  0 1 0 (mente)
```

```
     2 5
+    5 6
=    8 1
```

Det finns to addisjonsinstruksjoner, ADC og ADD siden mente blir spart fra forrige aritmetisk-logiske operasjon og man ikke vil ha den med i første addisjon. Alternativt må mente slettes før man legger sammen.

Legg til kode som beregner 8166 + 840. Siden tallene er større enn 8 bit må de lagres som en 16-bits verdi i to 8-bits registre. For å laste inn high/low-byte så kan dere bruke assembler-funksjonene high() og low():

```
ldi  r16,low(8166)
```

Se også AVR Studio assembler help under «Expressions» for mere forklaring og oversikt over funksjoner.

Oppgave 3 – løkker og forsinkelse.

I denne oppgaven skal vi implementere en lesing av en inngangspinne, en forsinkelse og toggling av utgangspinne i assembly. Programmet skal gjøre følgende:

a) Vente på at pinne PB0 (eller PB7 på Xplained Mini) går lav.
b) Deretter gå inn en uendelig løkke som toggler pinne PB5 ca. hvert sekund.

For å sjekke i/o register må vi enten lese det inn i et av de generelle registrene, eller bruke en instruksjon som kan sjekke bit i i/o registre f.eks: `sbic/sbis`

For å sette bit i registre kan vi (eventuelt lese først og så) skrive til dem med out/sts, eller bruke `sbi`

Forsinkelse kan implementeres ved å lage en programløkke som tar et antall klokkesykluser. Eksempel på løkke som tar 1000 sykler, dvs. 1ms @ 1MHz:

```
      ldi r16,250     // 1 klokkesyklus
delay_loop:
      dec r16         // 1 klokkesyklus
      nop             // 1 klokkesyklus
      brne delay_loop // 2 klokkesykler nå hoppet tas.
```

Vi skal i denne oppgaven implementere en løkke som tar ca. 1 sekund, dvs. ca 1 000 000 klokkesykluser ved en klokkefrekvens på 1 MHz eller 16 000 000 @ 16MHz. Dette kan realiseres ved nøstede løkker, dvs. flere løkker utenpå hverandre, eller ved å utvide tellervariabelen til mer enn 8 bit.

Simuler oppgaven og sjekk at programmet virker som forventet. Forsinkelse tar lang tid i simulator så man bør redusere verdiene for sjekke den. Test det deretter på Arduino UNO. Husk å konfigurere PB5 som utgang.

Lever den oversiktlige assemblyfilen i It's Learning. Ignorer om det kommer en feilmelding.