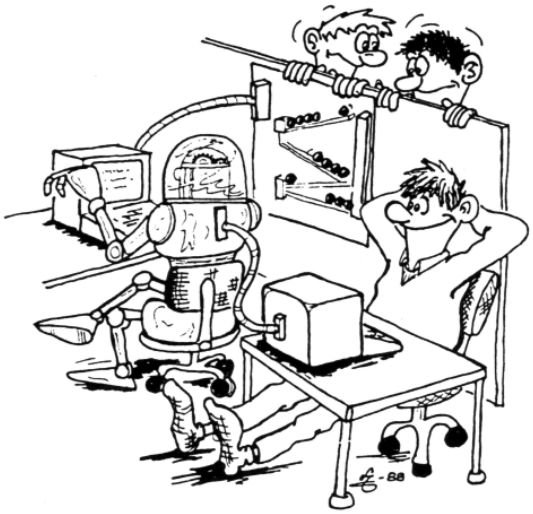
Industriautomation

Tentamen SSY 066, fredag 15/01-2016, fm, Lindholmen
Lärare: Kristofer Bengtsson, 0768 979561

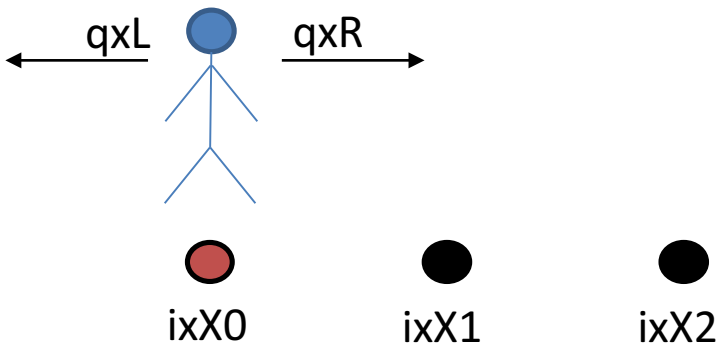


Fullständig lösning ska lämnas på samtliga uppgifter. I förekommande fall av tvetydigt formulerade tentamensuppgifter ska den föreslagna lösningen och eventuella antaganden motiveras. Examinator förbehåller sig rätten att godkänna rimligheten i antaganden och motiveringar.

Totalt omfattar tentamen 15 poäng. För betygen tre, fyra och fem krävs 7, 10 resp. 13 poäng. Lösningar anslås direkt efter tentatillfället på kursens hemsida i Pingpong. Granskning av rättningen sker 1 februari kl. 12:30 – 13:30 på institutionen.

OBS. Inga hjälpmedel är tillåtna.

Uppgift 1



I denna uppgift skall du implementera 2 rungs (network i codesys) i ladder logic så att gubben kan gå mellan positionerna X0, X1 och X2:

- En rung som styr utgång qxL
- En rung som styr qxR

Beroende på insignalerna toX0, toX1 och toX2 (som en annan kod sätter), skall din kod styra gubben så att den går till rätt position och sedan stannar där. När gubben befinner sig på en position är respektive sensor, ixX0, ixX1 eller ixX2, true.

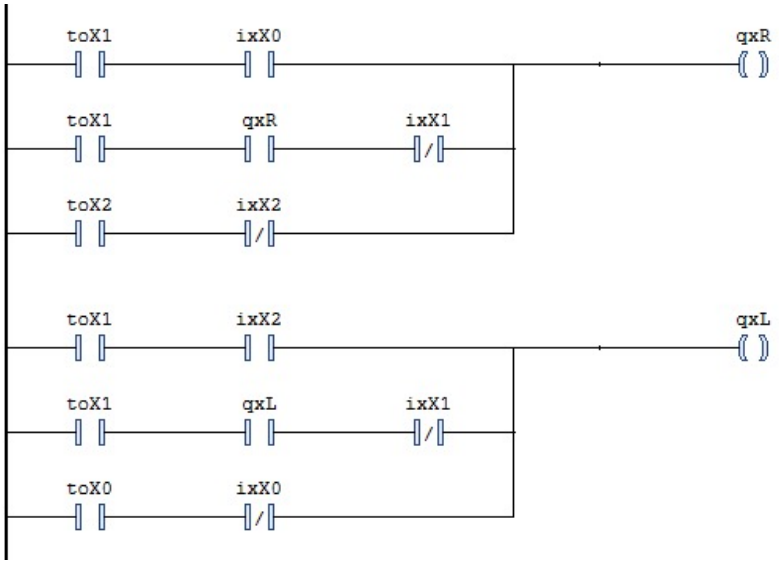
Endast en av insignalerna toX0, toX1 och toX2 kommer vara sann åt gången och de ändras endast till true när gubben befinner sig i en av positionerna. Sedan är denna insignal sann ända tills en ny insignal blir satt sann.

Exempel: Om toX1 blir sann skall din kod sätta qxR till true om gubben befinner sig i X0 eller qxL till true om gubben befinner sig i X2. När gubben är framme och sensorn ixX1 blir sann skall styrsignalen bli false.

Observera att du endast får implementera två rungs för att lösa uppgiften!

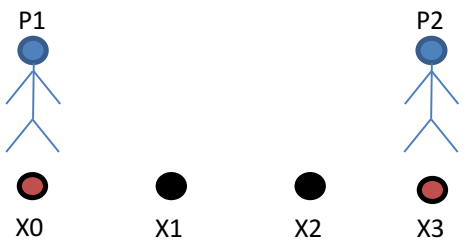
(5 poäng)

Svar uppgift 1



Uppgift 2

I denna uppgift skall två gubbar, P1 och P2, utföra uppdrag vid de olika positionerna. P1 kan flyttas runt med följande operationer, p1ToX0, p1ToX1 och p1ToX2, och P2 med p2ToX1, p2ToX2 och p2ToX3. Dessan operationer ser till att respektive gubbe tar sig till angiven position oberoende av var de befinner sig när de startar.

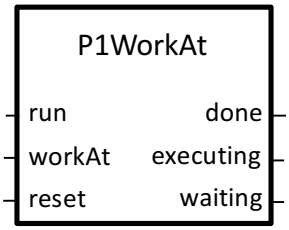


a) En begränsning i systemet är att P1 och P2 inte kan passera varandra utan att de krockar. Definiera conditions för de 6 operationerna ovan, så att gubbarna inte kolliderar. De skall dock kunna röra sig samtidigt när det är tillåtet. Inför gärna egna variabler som du kan använda i dina guards och actions. Observera att gubbarna inte kan stanna mellan positionerna, så när de väl har börjat röra sig måste de kunna komma fram till rätt position. Gubbarna kan inte komma ikapp varandra när de går åt samma håll.

(2 poäng)

b) Implementera operationen P1WorkAt med SFC med in och utgångarna enlikt funktionsblocket nedan. Koden skall styra P1 med hjälp av operationerna ovan. Dessa operationer är redan implementerade och startas med tex. p1ToX1.run och är färdig när p1ToX1.done blir sann. p1ToX1.done går låg när run går låg, så run skall vara sann under hela operationen.

Arbetet vid positionen sker med operationen p1Work och startas och är färdig på samma sätt som operationerna ovan. Observera att du själv måste hålla kolla på att P1 inte krockar med P2. Definiera egna in och utgångar i funktionsblocket för att hantera dina villkor från del a. (P2 kommer styras av ett liknande block som du inte behöver implementera). Det är tillåtet med tilldelning direkt i ett stegs action-ruta.




```
VAR_INPUT
        run: bool // startar uppgiften vid true.
        workAt: Int // 0 till 2. Säger var P1 skall arbeta
        reset: bool // sätt sann när operationen skall återgå till init
END_VAR;
VAR_OUTPUT
        done: bool // sätts true när operationen är i sitt finished tillstånd
        executing: bool // sätts sann när operationen utförs
        waiting: bool // sätts sann när operatören måste vänta på den andra operatören
END_VAR;
```

(3 poäng)

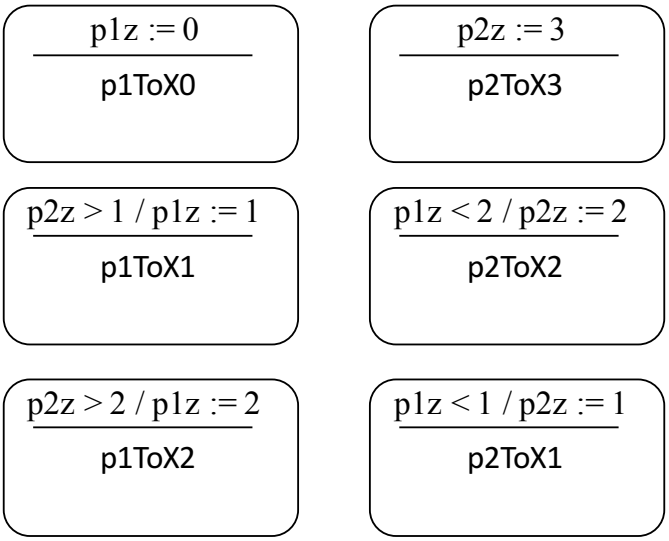
Svar Uppgift 2a

Inför två variabler som definierar var varje gubbe befinner sig (det går också lösa med en variabel eller med en variabel för x1 och en för x2):

p1z: INT := 0
p2z: INT := 3

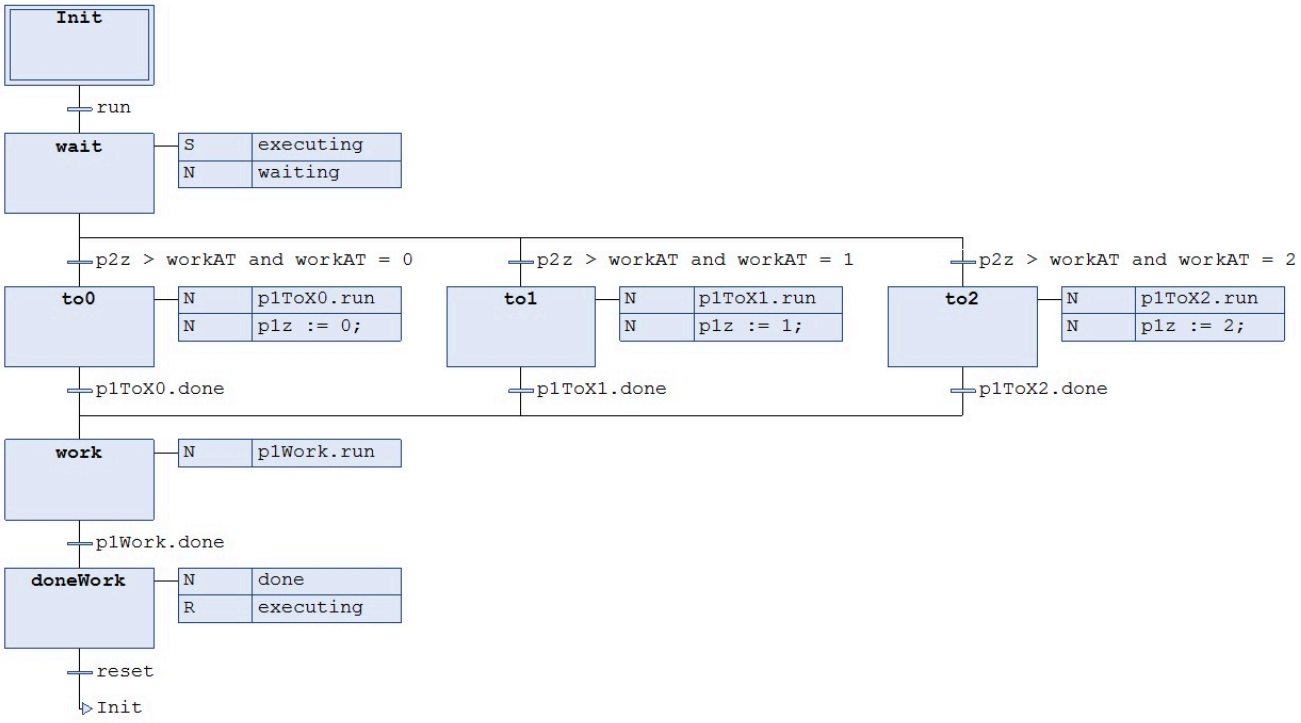
För att de inte skall krocka måste p1z < p2z.

Operationerna får följande preconditions (det behövs inga postconditions eftersom ”bokningen” av positionen måste ske innan en gubbe börjar gå annars kan de krocka)

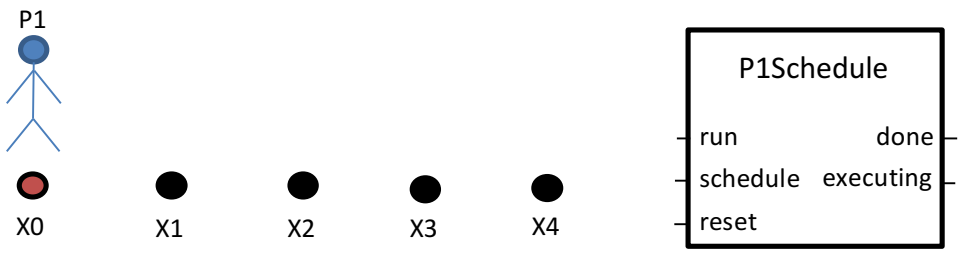


Svar Uppgift 2b

```
1   PROGRAM P1WorkAt
2   VAR_INPUT
3       run : BOOL ;
4       workAT : INT ;
5       reset : BOOL ;
6       p2z : INT ;
7   END_VAR
8   VAR_OUTPUT
9       done : BOOL ;
10      executing : BOOL ;
11      waiting : BOOL ;
12      p1z : INT := 0 ;
13  END_VAR
14
```



Uppgift 3



I denna uppgift skall du implementera blocket P1Schedule med hjälp av ladder för att styra P1s arbete med ett schema. Schemat säger i vilka positioner P1 skall arbeta och i vilken ordning. Till exempel kan ett schema säga att P1 skall arbeta enligt <1, 2, 1, 3, 4>, vilket innebär att P1 utför arbeten i X1, X2, X1, X3 och X4.

Schemat är definierat i ”strukten” Schedule nedan, som består av en array, workAT, vilket beskriver en sekvens av arbetspositioner (första positionen finns i workAT[0]) och en variabel noOfWorks som definierar hur många arbetspositioner schemat innehåller.

När ingången run blir sann skall din kod säga till P1 att arbeta i första arbetspositionen i schemat. För att styra P1 använder du P1WorkAt från föregående uppgift (du behöver inte ta med dina egna in och utgångar i denna uppgift). När det är klart fortsätter du styra P1 tills alla arbetspositioner är utförda (definieras av noOfWorks och inte 100, dock kan det vara upp till 100 arbeten). När alla uppdrag är gjorda skall du sätta done och vänta på reset-signalen så att du kan sätta done false och vara bered på ett nytt schema.

Observera att du måste vänta minst en scancykel från att du säger till P1 att arbeta tills du reagerar på PIWorkAT.done-signalen då done kan råka vara sann när du startar P1WorkAT.run.

Tips: Rita upp en graf med möjliga tillstånd som hjälp vid implementeringen På nästa sida finns några exempel på FBs som du kan använda.

```
TYPE Schedule :
STRUCT
        workAT: ARRAY[0..100] OF INT; // Sekvense av arbetsordrar
        noOfWorks: INT; // antal arbetsorder som arrayen inkluderar
END_STRUCT
END_TYPE

VAR_INPUT
        run: bool // startar uppgiften på positiv flank om alla utgångarna är false.
        schedule: Schedule // Schemat som skall utföras
        reset: bool // sätt sann när operationen skall återgå till init
END_VAR;
VAR_OUTPUT
        done: bool // sätts true när operationen är i sitt finished tillstånd
        executing: bool // sätts sann när operationen utförs
END_VAR;
```

(5 poäng)

Svar Uppgift 3

```
1   PROGRAM P1Schedule
2   VAR_INPUT
3       run : BOOL ;
4       schedule : Schedule ;
5       reset : BOOL ;
6   END_VAR ;
7
8   VAR_OUTPUT
9       done : BOOL ;
10      executing : BOOL ;
11  END_VAR ;
12
13  VAR
14      init : BOOL := TRUE ;
15      eval : BOOL ;
16      work : BOOL ;
17      doneWork : BOOL ;
18      current : INT := 0 ;
19  END_VAR
20
```