Opgaven

en uitwerkingen bij het boek Digitale Techniek

Jesse op den Brouw

Deel 3

Versie: 0.99pl3
Datum: 1 juni 2018

DE HAAGSE HOGESCHOOL

Suggesties en/of opmerkingen over dit boek kunnen worden gestuurd naar:
J.E.J.opdenBrouw@hhs.nl.

Inhoudsopgave

11

11.1. Teken het toestandsdiagram van een JK-flipflop.

11.2. Teken het toestandsdiagram van een synchrone 6-teller.

11.3. Gegeven een RG-machine. De machine heeft twee ingangen A en B en twee uitgangen R en G. De machine begint met de uitgangen R en G allebei logisch 0 en wacht op een verandering op A en/of B. Als A logisch 1 wordt, moet de machine achtereenvolgens eerst R logisch 1 maken en daarna zowel R als G logisch 1 maken. Daarna moet de machine weer wachten op een verandering van A en/of B. Als B logisch 1 wordt, moet de machine achtereenvolgens eerst G logisch 1 maken en daarna zowel R als G logisch 1 maken. Daarna moet de machine weer wachten op een verandering van A en/of B. Teken het toestandsdiagram van de RG-machine.

11.4. Stel de toestandstabel van de JK-flipflop op.

11.5. Stel de toestandstabel van de synchrone 6-teller op.

11.6. Stel de toestandstabel van de RG-machine op.

11.7. Is het mogelijk om een Moore-machine als een Mealy-machine te tekenen? En andersom? Motiveer het antwoord.

11.8. Gegeven een machine met drie toestanden en twee toestandsbits. Bepaal het aantal unieke codecombinaties.

11.9. Bepaal het aantal mogelijke codecombinaties voor een machine met tien toestanden en een minimum aan toestandsbits.

11.10. Geef de toestands- en uitgangsfuncties van de JK-flipflop. Gebruik de binaire telcode voor toestandscodering.

11.11. Geef de toestands- en uitgangsfuncties van de synchrone 6-teller. Gebruik de binaire telcode voor toestandscodering.

11.12. Geef de toestands- en uitgangsfuncties van de RG-machine op. Gebruik de binaire telcode voor toestandscodering.

11.13. Geef de toestands- en uitgangsfuncties van de RG-machine op. Gebruik de one-hot-codering voor toestandscodering.

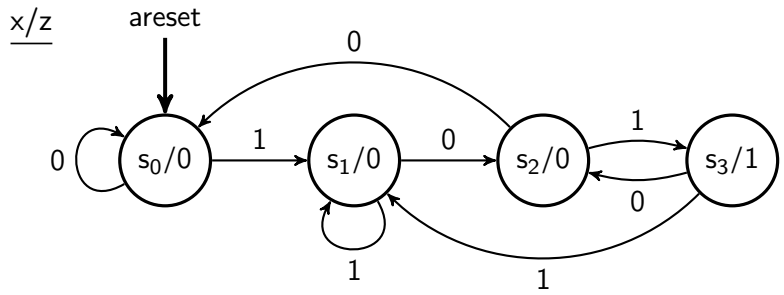
11.14. Ontwerp het toestandsdiagram van een machine voor het herkennen van het patroon (of reeks) 1001. De machine geeft een 1 af en stopt met herkennen.

11.15. Ontwerp het toestandsdiagram van een machine voor het doorlopend herkennen van het patroon (of reeks) 1001. De machine moet bij herkenning een 1 afgeven. Ontwerp zowel een Moore- als een Mealy-machine.

11.16. Ontwerp het toestandsdiagram van een Mealy-machine voor het doorlopend herkennen van het patroon (of reeks) 1001. Overlappingen zijn mogelijk. De machine moet bij herkenning een 1 afgeven. Geef de vergelijkingen voor de opvolgertoestand en de uitgang als gekozen wordt voor de binaire telcode als toestandscodering.

11.17. Ontwerp het toestandsdiagram van een Mealy-machine voor het het doorlopend herkennen van het patroon (of reeks) 100 met een minimaal aantal toestanden.

11.18. Gegeven onderstaande machine voor het overlappend herkennen van 101 (figuur P11.1). Geef de toestandsfuncties indien one-hot codering wordt gebruikt.

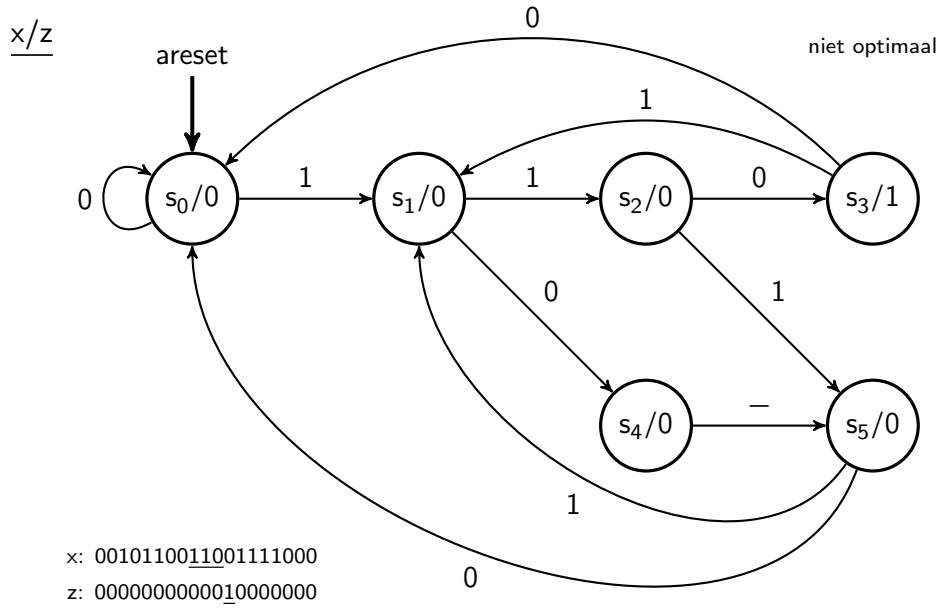


Figuur P11.1: *Herkenner voor het overlappend herkennen van het patroon 101 (Moore).*

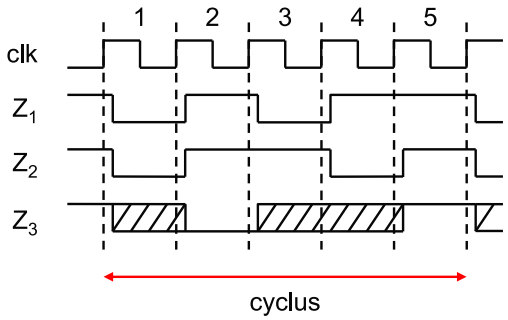
11.19. In figuur P11.2 is het toestandsdiagram van een machine te zien voor het bloksgewijs herkennen van het patroon (of reeks) 110. Een studente beweert dat het toestandsdiagram van deze machine compacter kan worden getekend omdat toestand s_0 en s_5 identiek zijn en samengenomen kunnen worden. Klopt deze bewering? Motiveer het antwoord.

11.20. Ontwerp een Mealy-machine voor het het bloksgewijs herkennen van het patroon (of reeks) 110 met een minimaal aantal toestanden. De machine blijft wachten zolang een 0 wordt aangeboden en gaat pas beginnen met herkennen mij een 1, inclusief de eerste 1.

11.21. In figuur P11.3 is een machine gespecificeerd d.m.v. een timingdiagram. Geef de functies voor de opvolgertoestandslogica en de uitgangslogica, de toestandscodering ligt gedeeltelijk vast.

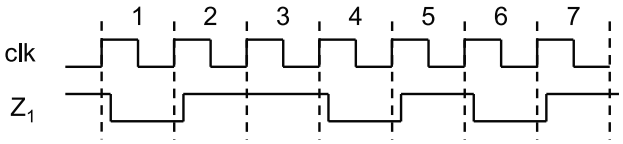


Figuur P11.2: *Herkenner voor het bloksgewijs herkennen van het patroon 110 (Moore).*



Figuur P11.3: *Timingdiagram van een toestandsmachine.*

11.22. Gegeven het timingdiagram in figuur P11.4. Ontwerp een machine die het timingdiagram uitvoert met gebruikmaking van extra flipflops en zonder uitgangslogica.



Figuur P11.4: *Timingdiagram van een toestandsmachine.*

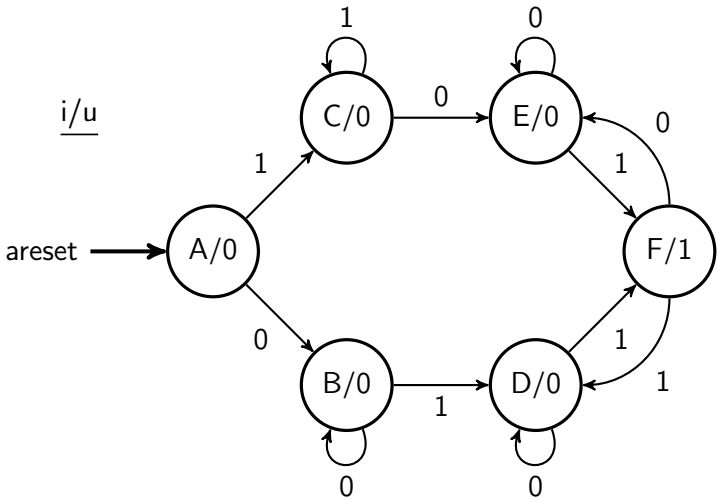
11.23. Ontwerp een toestandsdiagram voor een snoepmachine die 35 cent moet herkennen. Toegestane munten zijn 5 cent (S) en 10 cent (D). Bij dit bedrag moet snoepgoed worden geretourneerd. Bij te hoog bedrag moet een signalering geactiveerd worden.

11.24. Wat is het hoogste bedrag dat in de machine kan worden ingeworpen (dat is niet het bedrag dat de machine uiteindelijk moet herkennen).

11.25. Hoeveel unieke toestanden heeft bovenstaande machine?

11.26. Stel dat een snoepautomaat munten van 5 cent (S), 10 cent (D) en 20 cent (T) accepteert. Wat is het hoogste bedrag dat kan worden ingeworpen? Hoeveel unieke toestanden heeft deze snoepautomaat?

11.27. Minimaliseer het aantal toestanden van de machine in figuur P11.5.



Figuur P11.5: *Toestandsdiagram van een Moore-machine.*

11.28. Gegeven een machine die beschreven wordt door onderstaande *next state equations* en *output equation*:

$$\begin{aligned} Q_1^{n+1} &= Q_1^n \cdot X^n + Q_0^n \cdot X^n \\ Q_0^{n+1} &= Q_1^n \cdot X^n + \overline{Q_0^n} \cdot X^n \\ Z^n &= Q_1^n + Q_0^n \end{aligned} \tag{P11.1}$$

(a) Stel de waarheidstabel op voor de functies Q_1^{n+1}, Q_0^{n+1} en Z^n.

(b) Stel het toestandsdiagram op voor deze machine. Gebruik hiervoor de waarheidstabel uit (a).

(c) Het toestandsdiagram kan vereenvoudigd worden, d.w.z. er kunnen toestanden samengenomen worden zonder dat het gedrag van de machine verandert. Zulke toestanden worden *equivalent* genoemd. Laat zien dat het toestandsdiagram uit (b) ook met twee toestanden getekend kan worden zonder dat de werking van de machine verandert.

12

12.1. Register A is $2n$ bits breed. Ontwerp een derde versie van de vermenigvuldiger waarbij voor A slechts n bits nodig zijn. Hint: P schuiven in plaats van A.

12.2. De opteller is $2n$ bits breed. Ontwerp een vierde versie van de vermenigvuldiger waarbij voor de opteller minder bits nodig zijn.

12.3. Een nadeel van de ontworpen besturing is dat P wordt gewist in de rusttoestand. Pas het toestandsdiagram aan zodat P behouden blijft.

12.4. Herontwerp de toestandsmachine voor de tweede versie van de vermenigvuldiger als een Mealy-machine.

12.5. Toon aan dat een vermenigvuldiging van twee n-bits getallen altijd past in $2n$ bits.

12.6. Een slimme student opperde dat signaal b_0 direct verbonden kan worden aan signaal $loadp$. Werkt de vermenigvuldiger dan nog correct? Motiveer het antwoord.

12.7. Ontwerp een systeem dat van de inhoud van een 8-bits register het aantal enen bepaalt dat in het register is opgeslagen. Ontwerp zowel het datapad als de besturing.

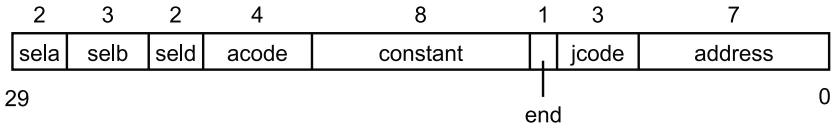
13

13.1. Beschrijf waarom de ALU-operatie `nop` handig is.

13.2. Beschrijf waarom de ALU-operatie `pass` handig is.

13.3. Zijn er nog meer (eenvoudige) logische operaties te bedenken? Zijn deze operaties ook met de huidige verzameling van logische operaties te realiseren (want dan zijn de nieuwe operaties overbodig ...)?

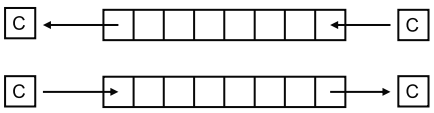
13.4. Geef het bitpatroon dat ervoor zorgt dat de EXOR van R2 en R3 in R1 terecht komt (snelschrift: **`exor`** `R1,R2,R3`). Het instructieformaat is te zien in figuur P13.1.



Figuur P13.1: *Het instructieformaat van de vierde versie van de eenvoudige processor.*

13.5. Beschrijf de werkingen van **`and`** `R1,R2,R3` en **`and`** `R1,R3,R2`.

13.6. Eerder zijn de schuifoperaties aan bod geweest. Hierbij werd het meest significante of minst significante bit in de carry flag geplaatst en werd een logische 0 ingeschoven. In plaats van deze 0 zou ook de carry flag kunnen worden ingeschoven. Beschrijf het effect van deze nieuwe operatie. Merk op dat alle registers en flags flankgestuurd zijn, dus de huidige waarde van de carry flag wordt ingeschoven en de huidige meest significante of minst significante bit van het registers wordt uitgeschoven. Zie ook figuur P13.2.



Figuur P13.2: *Naar links en naar rechts roteren met behulp van de carry flag.*

13.7. Stel dat het voorbeeldprogramma in listing P13.1 niet vanaf adres 0 geplaatst is, maar vanaf adres 5 (het programma zelf blijft ongewijzigd). Kan het programma daar zomaar "neergezet" worden? Motiveer het antwoord.

```
 0:   ld   R0,#13      ; load multiplicand (constant 13)
 1:   ld   R1,#11      ; load multiplier (constant 11)
 2:   ld   R2,#0       ; clear result registers
 3:   ld   R3,#0       ; or use ld R3,R2
 4:   cmp  R1,#0       ; is multiplier already 0 ?
 5:   je   @11         ; yes it is, so jump!
 6:   ld   F,#0        ; clear the flags (clears carry)
 7:   add  R2,R2,R0    ; add multiplicant (low byte)
 8:   add  R3,R3,#0    ; process carry in high byte
 9:   sub  R1,R1,#1    ; multiplier one off
10:   jmp  @4          ; and again
11:   ld   R0,R2       ; result to output*
12:   jmp  @12         ; hold your horses!
```

Listing P13.1: *Code voor het vermenigvuldigen van 11 met 13.*

13.8. In het programma in listing P13.1 wordt direct na de twee **add**-instructies een **sub**-instructie uitgevoerd. Maar de **add**- en **sub**-instructies verwerken ook de carry-flag. Moet dan vóér de **sub**-instructie de carry flag niet gewist worden?

13.9. De instructie **not** `Rx,#const` "bestaat" niet. Is deze instructie wel te maken? Motiveer het antwoord.

13.10. Stel dat er helemaal geen **not**-instructie zou bestaan. Kan dan toch de NOT-operatie worden uitgevoerd/verwerkt? (ja dat kan). Motiveer het antwoord.

13.11. Ontwerp een programma dat de input eenmaal inleest en het aantal enen bepaalt in de ingelezen waarde. De pseudo-code is gegeven in listing P13.2.

```
R3 := 0;                    // result
R1 := 8;                    // counter
R2 := input;                // read input once
do                          // loop
    R2 := shr(R2);          // shift right R2, low bit in carry
    R3 := R3 + 0 + carry;   // add carry
    carry := 0              // clear the carry
    R1 := R1 - 1 - carry;   // decrement counter
while (R1 <> 0);            // as long as R1 not equal to 0
```

Listing P13.2: *Code voor het tellen van het aantal enen in de input.*

13.12. Het ontwikkelde programma uit opgave 13.11 kent een lus die een aantal klokpulsen duurt. Hoeveel klokpulsen duurt het uitvoeren van de lus?

13.13. Gegeven de pseudo-code in listing P13.3. Vertaal deze code naar assembler. Gebruik hierbij vergelijk- en spronginstructies.

```
if R1 = 5 then
    R2 := 8;
else
    R2 := 10;
end if;
```

Listing P13.3: *Code voor een beslissing.*

13.14. Een ontwerper wil een wachtlus beschrijven waarbij op basis van de zero flag wordt getest of de registers allemaal op 0 staan. Zie listing P13.4. Werkt deze opzet correct?

```
        ld   R1,#<value1>   ; load value 1
        ld   R2,#<value2>   ; load value 2
        ld   R3,#<value3}   ; load value 3
        ld   F,#0           ; clear flags
delay:  sub  R1,R1,#1       ; subtract 1 from low byte
        sub  R2,R2,#0       ; subtract carry from middle byte
        sub  R3,R3,#0       ; subtract carry from high byte
        jne  delay          ; jump if not equal
```

Listing P13.4: *Code voor wachtlus gebaseerd op de zero flag.*

13.15. Gegeven de wachtlus gebaseerd op de carry flag, te zien in listing P13.5. Verder is gegeven dat de processor loopt op 50 MHz en dat de wachttijd 2/3 seconden bedraagt. Bereken de waarden waarmee de registers geladen moeten worden.

```
        ld   R1,#<value1>   ; load value 1
        ld   R2,#<value2>   ; load value 2
        ld   R3,#<value3}   ; load value 3
        ld   F,#0           ; clear flags
delay:  sub  R1,R1,#1       ; subtract 1 from low byte
        sub  R2,R2,#0       ; subtract carry from middle byte
        sub  R3,R3,#0       ; subtract carry from high byte
        jnc  delay          ; jump if no carry
```

Listing P13.5: *Code voor wachtlus gebaseerd op de carry flag.*

13.16. Gegeven de code in listing P13.6. Wat zijn de waarden (of inhouden) van $R1$ en $R2$ na uitvoeren van de code?

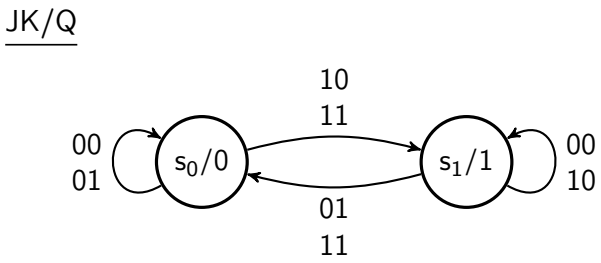
```
1    ld   R1,#13
2    ld   R2,#45
3    exor R1,R1,R2
4    exor R2,R2,R1
5    exor R1,R1,R2
```

Listing P13.6: *Code.*

A

Uitwerkingen

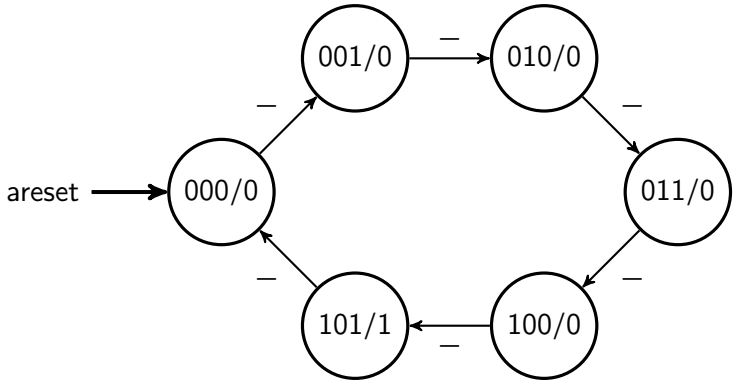
Uitwerking opgave 11.1.
Het toestandsdiagram van de JK-flipflop is te vinden in figuur A.1. De flipflop moet in toestand s_0 blijven als het moet onthouden ($JK = 00$) moet gereset wordt ($JK = 01$). De flipflop gaat naar toestand s_1 als er geset wordt ($JK = 10$) of getoggeld wordt ($JK = 11$). De flipflop moet in s_1 blijven als er onthouden moet worden ($JK = 00$ of geset wordt ($JK = 10$). De flipflop gaat naar toestand s_0 als er gereset wordt ($JK = 01$) of getoggeld wordt ($JK = 11$).



Figuur A.1: *Toestandsdiagram van een JK-flipflop.*

Uitwerking opgave 11.2.
In figuur A.2 is het toestandsdiagram van de 6-teller te zien. De teller telt continue van 0 tot en met 5, er is geen enable aanwezig. Sterker nog, er is niet eens een (stuur-)ingang. We hebben hier gekozen om de codering in de toestanden te plaatsen en niet om symbolische namen te gebruiken. Zo zien we direct de uitgangswaarden van de teller. De uitgang wordt in de hoogste stand (101) een klokcyclus 1. Dit komt overeen met de terminal count van de teller.

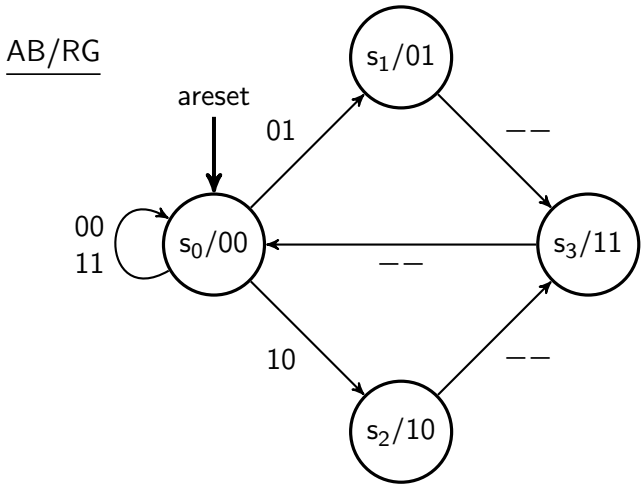
Uitwerking opgave 11.3.
De opgave begint al gelijk met een onduidelijkheid. Er staat dat als de ingang $A = 1$ de



Figuur A.2: *Toestandsdiagram van een 6-teller.*

machine een bepaalde toestandcyclus moet doorlopen, maar dat staat er ook als $B = 1$. Wat moet er nu gebeuren als zowel $A = 1$ als $B = 1$? Dat staat dus niet vermeld. Als ontwerper moeten we deze onduidelijkheid wegnemen. Meestal volgt een gesprek met de opdrachtgever om het probleem op te lossen. We kiezen er hier voor om de machine in dezelfde toestand te laten als $AB = 11$.

In figuur A.3 is het toestandsdiagram getekend. Alleen in toestand s_0 worden de signalen A en B bekeken. In alle andere toestanden wordt in feite niet naar de ingangen gekeken. Zoals gezegd laten we de machine in toestand s_0 als $AB = 11$. Dat geldt ook voor $AB = 00$, geen van de ingangen is dan immers 1. Alleen als $AB = 01$ of $AB = 10$ worden de andere toestanden doorlopen. Na een opvolgertoestand s_1 of s_2 worden volgt toestand s_2 en gaat de machine weer naar toestand s_0



Figuur A.3: *Toestandsdiagram van de RG-machine.*

Uitwerking opgave 11.4.

De toestandtabel is te vinden in tabel A.1. Als de flipflop in s_0 staat blijft de flipflop in deze toestand als er moet worden onthouden ($JK = 00$) of als er gereset wordt ($JK = 01$). Met een set-actie of als er getoggled moet worden, gaat de flipflop naar toestand s_1. Als de flipflop in s_1 staat, blijft de flipflop in deze toestand als er moet worden onthouden

($JK = 00$) of als er geset wordt ($JK = 10$). De flipflop gaat naar toestand s_0 als er gereset moet worden ($JK = 01$) of als er getoggled moet worden $JK = 11$). Merk op dat we hier spreken over een flipflop en niet over een machine. Dat maakt in feite niet uit; elke flipflop is ook een toestandsmachine.

Tabel A.1: *Toestandstabel van de JK-flipflop.*

toestand	opvolger				uitgang
	00	01	10	11	
s_0	s_0	s_0	s_1	s_1	0
s_1	s_1	s_0	s_1	s_0	1

Uitwerking opgave 11.5.

De teller heeft geen ingangen en gaat bij elke actieve klokflank naar de volgende toestand. Dat maakt het opstellen van een toestandstabel erg makkelijk. Omdat de toestand van de teller ook de uitgangswaarden representeren gebruiken we nu bitcombinaties en niet de bekende symbolische benaming. We hebben echter drie flipflops nodig om de telstand (toestand) weer te geven en dat betekent dat er twee toestanden niet gebruikt worden. We geven de opvolger dan ook aan met don't cares. Zie tabel A.2.

Tabel A.2: *Toestandstabel van de synchrone 6-teller.*

toestand	opvolger
000	001
001	010
010	011
011	100
100	101
101	000
110	−−−
111	−−−

Uitwerking opgave 11.6.

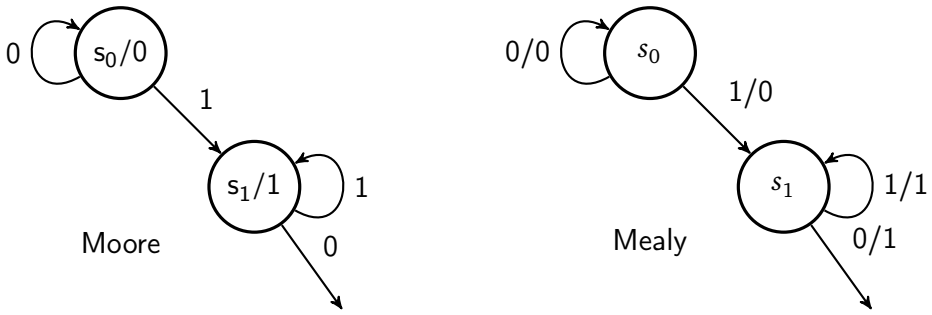
In tabel A.3 is de toestandstabel van de RG-machine te zien. In toestand s_0 wordt gewacht zolang $RG = 00$ of $RG = 11$ is. Als $RG = 01$ is gaat de machine naar toestand s_1. Bij een ingangscombinatie van $RG = 10$ gaat de machine naar toestand s_2. Vanuit toestand s_1 en s_2 gaat de machine onder elke conditie naar toestand s_3. Daarna gaat de machine altijd naar s_0. Dat betekent dat we alleen bij de regel van toestand s_0 even goed moeten kijken wat de opvolgertoestand is.

Uitwerking opgave 11.7.

Het is altijd mogelijk om een toestandsdiagram van een Moore-machine als een Mealy-machine te tekenen. Een Moore-machine is namelijk een vereenvoudiging van een Mealy-machine. Bij een toestand in het toestandsdiagram wordt dan bij alle uitgaande pijlen dezelfde uitgangswaarde geschreven. Een voorbeeld is te zien in figuur A.4.

Tabel A.3: *Toestandstabel van de RG-machine.*

toestand	opvolger				uitgangen
	00	01	10	11	
s_0	s_0	s_1	s_2	s_0	00
s_1	s_3	s_3	s_3	s_3	01
s_2	s_3	s_3	s_3	s_3	10
s_3	s_0	s_0	s_0	s_0	11



Figuur A.4: *Voorbeeld van Moore-toestanden getekend als Mealy-toestanden.*

Het is niet mogelijk om een Mealy-machine te tekenen als een Moore-machine omdat de uitgangswaarden ook afhankelijk zijn van de ingangswaarden.

Uitwerking opgave 11.8.

Het aantal toestanden is drie en het aantal toestandsbit is twee. We kunnen het aantal unieke codecombinaties uitrekenen:

$$N_D = \frac{2^k!}{(2^k-n)!k!} = \frac{2^2!}{(2^2-3)!2!} = \frac{4!}{1!2!} = \frac{24}{2} = 12 \tag{A.1}$$

Uitwerking opgave 11.9.

Het aantal toestanden is tien. Daaruit volgt dat er minimaal vier toestandsbits nodig zijn om elke toestand uniek te coderen. We kunnen het aantal unieke codecombinaties uitrekenen:

$$N_D = \frac{2^k!}{(2^k-n)!k!} = \frac{2^4!}{(2^4-10)!4!} = \frac{16!}{6!4!} = 1\,210\,809\,600 \tag{A.2}$$

Uitwerking opgave 11.10.

Eerst moet aan de toestanden een codering worden toegekend. De meest voor de hand liggende codering is $s_0 = 0$ en $s_1 = 1$. Er is dus maar één flipflop nodig. De JK-flipflop heeft twee ingangen zodat de waarheidstabel voor de toestandsfunctie en uitgang acht regels bevat. Zie tabel A.4.

Tabel A.4: *Waarheidstabel met toestandsfunctie en uitgangsfunctie voor de JK-flipflop.*

Q^n	J^n	K^n	Q^{n+1}	D^n	u^n
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	0	1	1

Met behulp van een Karnaughdiagram vinden we voor de JK-flipflop de functies:

$$\begin{aligned} Q^{n+1} &= J^n \cdot \overline{Q^n} + \overline{K^n} \cdot Q^n \\ u^n &= Q^n \end{aligned} \tag{A.3}$$

De uitgang u^n is hetzelfde als de uitgang van de flipflop. Strikt genomen is dit een Medvedev-machine.

Uitwerking opgave 11.11.

In tabel A.5 is de waarheidstabel van de opvolgerfuncties van de 6-teller te zien. We gebruiken hier de binaire telcode omdat de uitgangen van de teller dan direct van de uitgangswaarden van de flipflops kan worden afgetapt. Twee toestanden, te weten $Q_2Q_1Q_0 = 110$ en $Q_2Q_1Q_0 = 111$, worden niet gebruikt en de opvolgers worden als don't care opgegeven.

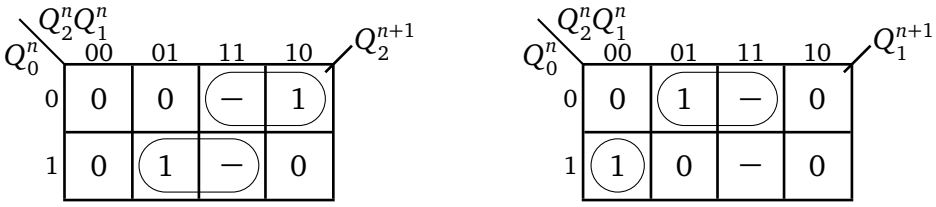
Tabel A.5: *Waarheidstabel met toestandsfuncties voor de 6-teller.*

Q_2^n	Q_1^n	Q_0^n	Q_2^{n+1}	Q_1^{n+1}	Q_0^{n+1}
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	1	0	–	–	–
1	1	1	–	–	–

De functies van de flipflops zijn met behulp van Karnaughdiagrammen te vinden, zie figuur A.5. De functie voor Q_0^{n+1} is erg eenvoudig, die laten we achterwege.

Hieronder zijn ze de functies gegeven:

$$\begin{aligned} Q_2^{n+1} &= Q_2^n \cdot \overline{Q_0^n} + Q_1^n \cdot Q_0^n \\ Q_1^{n+1} &= Q_1^n \cdot \overline{Q_0^n} + \overline{Q_2^n} \cdot \overline{Q_1^n} \cdot Q_0^n \\ Q_0^{n+1} &= \overline{Q_0^n} \end{aligned} \tag{A.4}$$



Figuur A.5: *Toestandsfuncties van de 6-teller.*

Uitwerking opgave 11.12.

Voor de toestanden van de RG-machine gebruiken we de binaire telcode: $s_0 = 00$, $s_1 = 01$, $s_2 = 10$ en $s_3 = 11$. We stellen een waarheidstabel op met alle mogelijke combinaties van toestandsbits en ingangswaarden, zie tabel A.6. We noteren de opvolgertoestanden en uitgangswaarden als bitcombinaties. We hebben de kolommen met D_1^n en D_0^n weggelaten omdat we uitgaan van D-flipflops en daarvoor geldt $Q^{n+1} = D^n$.

Tabel A.6: *Waarheidstabel met toestandsfuncties en uitgangsfuncties voor de RG-machine.*

Q_1^n	Q_0^n	A^n	B^n	Q_1^{n+1}	Q_0^{n+1}	R^n	G^n
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0
0	0	1	0	1	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	1	1	0	1
0	1	0	1	1	1	0	1
0	1	1	0	1	1	0	1
0	1	1	1	1	1	0	1
1	0	0	0	1	1	1	0
1	0	0	1	1	1	1	0
1	0	1	0	1	1	1	0
1	0	1	1	1	1	1	0
1	1	0	0	0	0	1	1
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	1
1	1	1	1	0	0	1	1

De functies zijn eenvoudig van aard, deze zijn op te lossen zonder Karnaughdiagrammen te gebruiken:

$$\begin{aligned} Q_1^{n+1} &= \overline{Q_1^n}\cdot Q_0^n + Q_1^n\cdot\overline{Q_0^n} + \overline{Q_1^n}\cdot A^n\cdot B^n \\ Q_0^{n+1} &= \overline{Q_1^n}\cdot Q_0^n + Q_1^n\cdot\overline{Q_0^n} + \overline{Q_1^n}\cdot\overline{A^n}\cdot B^n \\ R^n &= Q_1^n \\ G^n &= Q_0^n \end{aligned} \tag{A.5}$$

Te zien is dat de uitgangswaarden direct van de flipflops kan worden afgetapt.

Uitwerking opgave 11.13.

Voor de toestandscodering gebruiken we de one-hot codering: $s_0 = 0001$, $s_1 = 0010$, $s_2 = 0100$ en $s_3 = 1000$. De toestandsfunctie van een flipflop is op te schrijven door alle inkomende pijlen te bekijken. Elke inkomende pijl vertegenwoordigd één productterm van de toestandsfunctie. Bekijken we toestand s_0, dan zien we twee inkomende pijlen waarbij één pijl twee overgangsvoorwaarden heeft. Hier horen dan twee producttermen bij. Zo blijft de machine in toestand s_0 als R en G beide 0 of beide 1 zijn. Daar horen de producttermen $Q_0^n \cdot \overline{A^n} \cdot \overline{B^n}$ en $Q_0^n \cdot A^n \cdot B^n$. De machine gaat altijd vanuit toestand s_3 naar toestand s_0. Daar hoort de *gedegenereerde* productterm Q_n^3 bij. Een gedegenereerde productterm bestaat alleen uit de waarde van de flipflop; er zijn geen overgangsvoorwaarden opgegeven dus alleen de waarde van de flipflop is nodig. Toestand s_1 wordt alleen bereikt als de machine is toestand s_0 staat én $RG = 01$. Daar hoort de productterm $Q_0^n \cdot \overline{A^n} \cdot B^n$ bij. Iets dergelijks geldt ook voor toestand s_2. Daar hoort de productterm $Q_0^n \cdot A^n \cdot \overline{B^n}$ bij. Toestand s_3 wordt (altijd) bereikt als de machine in toestand s_1 of toestand s_2 staat. Daar hoort de term $Q_1^n + Q_2^n$ bij. Dit is een samenstelling (somterm) van de gedegenereerde producttermen Q_1^n en Q_2^n. Alle functies zijn als volgt:

$$\begin{aligned}
Q_0^{n+1} &= Q_0^n \cdot \overline{A^n} \cdot \overline{B^n} + Q_0^n \cdot A^n \cdot B^n + Q_3^n \\
Q_1^{n+1} &= Q_0^n \cdot \overline{A^n} \cdot B^n \\
Q_2^{n+1} &= Q_0^n \cdot A^n \cdot \overline{B^n} \\
Q_3^{n+1} &= Q_1^n + Q_2^n \\
R^n &= Q_2^n + Q_3^n \\
G^n &= Q_1^n + Q_3^n
\end{aligned} \tag{A.6}$$

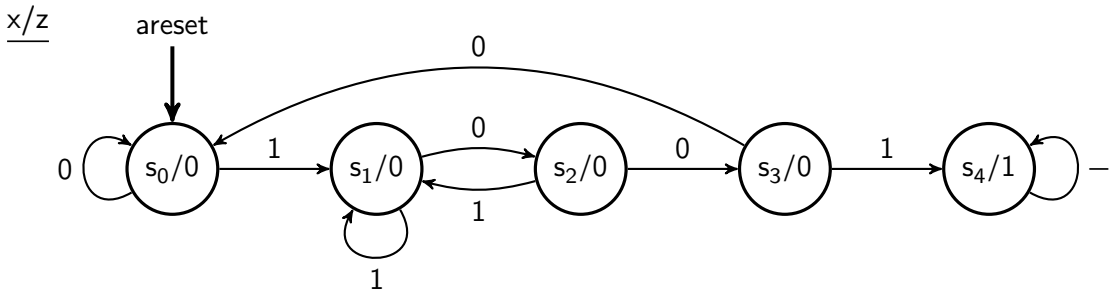
Uitwerking opgave 11.14.

In figuur A.6 is het toestandsdiagram getekend. Het begint met het opstellen van de vijf toestanden die nodig zijn voor het herkennen van 1001. Daarna volgende de uitzonderingen. Zo blijft de machine in toestand s_0 zolang er een 0 wordt gedetecteerd, bijvoorbeeld bij het patroon 000.. De machine blijft in toestand s_1 als na een 1 nog een 1 (of meerdere) wordt gedetecteerd, bijvoorbeeld bij het patroon 111.. Als na het (deel-)patroon 10 weer een 1 wordt gedetecteerd gaat de machine van toestand s_2 terug naar toestand $_1$ bijvoorbeeld bij het patroon 101.. De machine gaat van toestand s_3 terug naar toestand s_0 als na het (deel-)patroon 100 weer een 0 wordt herkend, bijvoorbeeld bij het patroon 1000.. De machine blijft na herkennen in toestand s_4 hangen. De uitgangswaarde blijft dan 1. Eventueel kan de machine uitgebreid worden met een extra toestand waar de machine dan naar toegaat en de uitgangswaarde 0 is zodat er slechts één 1 wordt afgegeven.

Uitwerking opgave 11.15.

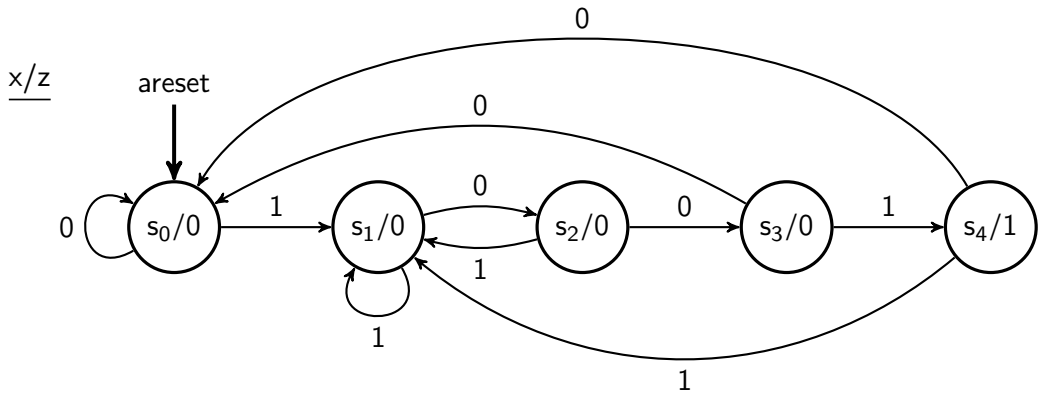
Voor de Moore-variant kunnen we uitgaan van de machine in de vorige opgave. Alleen de overgang bij de laatste toestand moet veranderd worden. Zie figuur A.7.

De Mealy-machine heeft maar vier toestanden. Bij de laatste toestand wordt bekeken of de uitgangswaarde 1 is en dan wordt een 1 afgegeven (bij de overgang uiteraard). Merk op dat de machine onder beide voorwaarden naar toestand s_0 gaat. Zie figuur A.8.



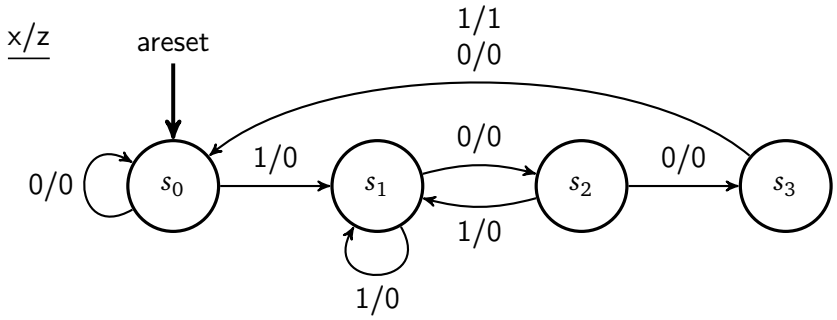
x: 00101100111001111000
z: 00000000011111111111

Figuur A.6: *Herkenner voor het eenmalig herkennen van het patroon 1001 (Moore).*



x: 00101100111001111000
z: 00000000010001000000

Figuur A.7: *Herkenner voor het doorlopend herkennen van het patroon 1001 (Moore).*



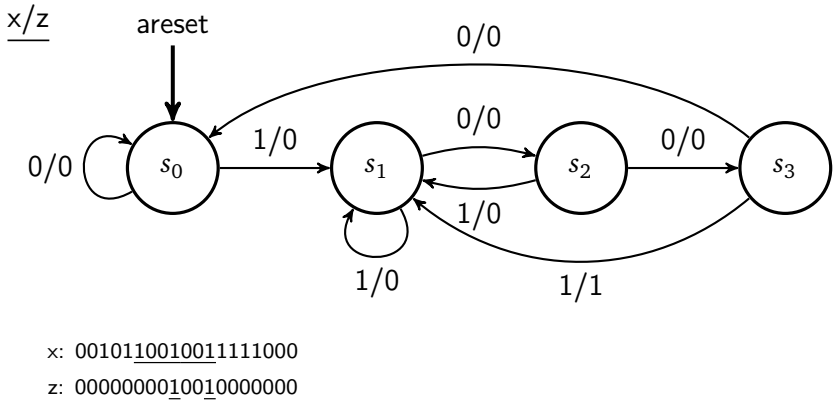
x: 00101100111001111000
z: 00000000010001000000

Figuur A.8: *Herkenner voor het doorlopend herkennen van het patroon 1001 (Mealy).*

Uitwerking opgave 11.16.

Ook deze machine is weer een variatie op het thema. Het gaat weer om toestand s_3.

Slecht één overgang is anders dan de eerder ontwikkelde machine. In toestand s_3 gaat de machine bij een 1 naar toestand s_2. Overlappingen als ..1001001.. worden nu herkend, maar ook doorlopende patronen als ..10011001.. worden hekend.



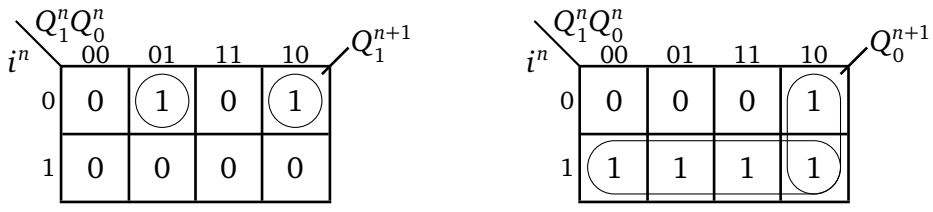
Figuur A.9: *Herkenner voor het doorlopend overlappend herkennen van het patroon 1001 (Mealy).*

We ontwikkelden de hardware-implementatie met behulp van de binaire telcode als toestandscodering: $s_0 = 00$, $s_1 = 01$, $s_2 = 10$ en $s_3 = 11$. Hiervoor zijn twee flipflops nodig. De machine heeft maar één ingang zodat het aantal ingangen voor de opvolgertoestandslogica en uitgangslogica drie bedraagt. Zie tabel A.7.

Tabel A.7: *Waarheidstabel met toestandsfuncties en uitgangsfuncties voor de patroonherkenner.*

Q_1^n	Q_0^n	i^n	Q_1^{n+1}	Q_0^{n+1}	z^n
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	1	1

In figuur A.10 zijn de functies voor Q_1^{n+1} en Q_0^{n+1} in Karnaughdiagrammen ingevuld. De functie van z^n is erg eenvoudig, dat kan zonder een Karnaughdiagram.

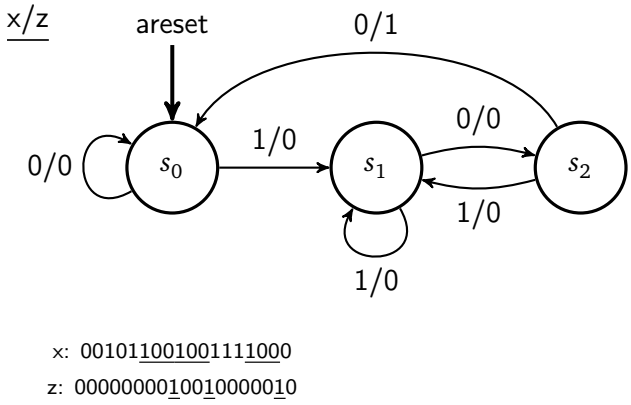


Figuur A.10: *Toestandsfuncties van de patroonherkenner.*

We vinden voor de toestandsfuncties en uitgangsfunctie het volgende:

$$\begin{aligned} Q_1^{n+1} &= \overline{Q_1^n} \cdot Q_0^n \cdot \overline{i^n} + Q_1^n \cdot \overline{Q_0^n} \cdot \overline{i^n} \\ Q_0^{n+1} &= i^n + Q_1^n \cdot \overline{Q_0^n} \\ z^n &= Q_1^n \cdot Q_0^n \cdot i^n \end{aligned} \tag{A.7}$$

Uitwerking opgave 11.17.
De machine moet een patroon van drie bits kunnen detecteren. Dat houdt in dat de Mealy-machine uit slechts drie toestanden bestaat. Het toestandsdiagram is gegeven in figuur A.11.



Figuur A.11: *Herkenner voor het doorlopend herkennen van het patroon 100 (Mealy).*

Uitwerking opgave 11.18.
Bij een one-hot codering wordt voor elke toestand één flipflop gebruikt die dan ook 1 is als de machine in de betreffende toestand staat. De andere flipflops zijn dan 0. Aldus gebruiken we de codering $s_0 = 0001$, $s_1 = 0010$, $s_2 = 0100$ en $s_3 = 1000$. We zoeken weer naar de pijlen die naar een toestand toewijzen, elke pijl levert één productterm uit de functie van de flipflop. We vinden:

$$\begin{aligned} Q_0^{n+1} &= Q_0^n \cdot \overline{x^n} + Q_2^n \cdot \overline{x^n} = (Q_0^n + Q_2^n) \cdot \overline{x^n} \\ Q_1^{n+1} &= Q_0^n \cdot x^n + Q_1^n \cdot x^n + Q_3^n \cdot x^n = (Q_0^n + Q_1^n + Q_3^n) \cdot x^n \\ Q_2^{n+1} &= Q^1 \cdot x^n \end{aligned} \tag{A.8}$$

Zoals te zien is, kunnen twee van de drie functies het beste in de POS-vorm genoteerd worden, dat enkele poorten winst op.

Uitwerking opgave 11.19.
De bewering van de studente klopt: toestand s_0 en s_5 zijn *equivalent* en kunnen samengenomen worden tot één toestand. We kunnen dit demonstreren met behulp van een toestandstabel voor de machine, zie tabel A.8.

Tabel A.8: *Toestandstabel van de bloksgewijze herkenner voor het patroon 110.*

toestand	opvolger		uitgang
	0	1	
s_0	s_0	s_1	0
s_1	s_4	s_2	0
s_2	s_3	s_5	0
s_3	s_0	s_1	1
s_4	s_5	s_5	0
s_5	s_0	s_1	0

Als we kijken naar de toestanden s_0 en s_5 dan zien we dat beide dezelfde opvolgertoestanden hebben onder besturing van de ingang én beide dezelfde uitgangswaarde hebben. Dat betekent dat het voor een buitenstaander niet te zien is of de machine in toestand s_0 of toestand s_5 staat; in beide gevallen is de uitgangsreactie hetzelfde. Als de machine in toestand s_0 staat en de ingangswaarde is 0, dan blijft (of gaat) de machine naar toestand s_0. Is de ingangswaarde een 1, dan gaat de machine naar s_1. De uitgangswaarde is 0. Dat gebeurt ook als de machine is s_5 staat. We kunnen toestand s_5 schrappen en een nieuwe toestandstabel opstellen, zie tabel A.9.

Tabel A.9: *Gereduceerde toestandstabel van de bloksgewijze herkenner voor het patroon 110.*

toestand	opvolger		uitgang
	0	1	
s_0	s_0	s_1	0
s_1	s_4	s_2	0
s_2	s_3	s_0	0
s_3	s_0	s_1	1
s_4	s_0	s_0	0

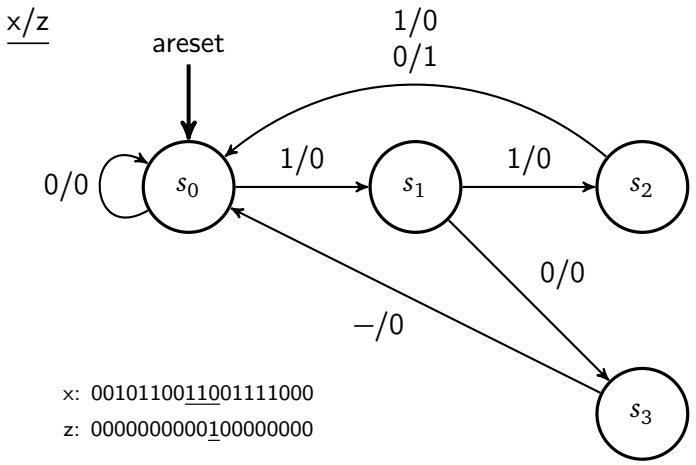
De oplettende lezer ziet dat ook toestand s_3 dezelfde opvolgers heeft maar hier is de uitgangswaarde anders dat bij toestand s_0/s_5.

Uitwerking opgave 11.20.

Het toestandsdiagram van de Mealy-machine voor het bloksgewijs herkennen van 110, beginnend met een 1 is te zien in figuur A.12. Er zijn slechts vier toestanden nodig. De machine wacht in toestand s_0 zoals de ingangswaarde 0 is. Als de ingangswaarde 1 is, worden drie bits “afgetikt“. Als na de eerste 1 en 0 volgt kan het patroon niet meer voorkomen en gaat de machine via toestand s_3 terug naar toestand s_0. Volgt na de eerste 1 nog een 1 dan gaat de machine naar toestand s_2. In deze toestand wordt dan bekeken of vervolgens een 0 of een 1 wordt gedetecteerd. De uitgangswaarde is dan naar gelang de ingangswaarde een 0 of een 1.

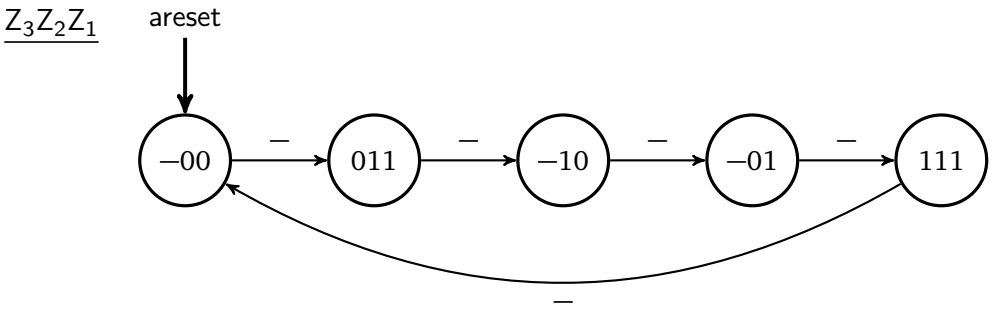
Uitwerking opgave 11.21.

We hebben in figuur A.13 het toestandsdiagram getekend met in de toestanden de uitgangswaarden voor Z_3, Z_2 en Z_1. Merk op dat bij drie van de vijf toestanden de waarde



Figuur A.12: *Herkenner voor het bloksgewijs herkennen van het patroon 110 (Mealy).*

van Z_3 niet belangrijk is. We hebben ze op don't care gesteld. De meest linkse toestand (−00) wordt dus door *twee* toestandscoderingen gekenmerkt, te weten 000 en 100. Iets dergelijks geldt ook bij de toestanden −10 en −01. We kunnen van deze don't cares gebruik maken bij het opstellen van de waarheidstabel voor de opvolgertoestand (de uitgangswaarden komen direct van Z_2 en Z_1). Zo kan de opvolger van toestand 011 gespecificeerd worden als −10; het maakt dus niet uit wat de waarde van Z_3 wordt. Merk op dat de don't cares alleen bij specificatie bestaan. Bij uitwerken van de functies worden de don't cares omgezet naar nullen en enen.



Figuur A.13: *Toestandsdiagram van machine op basis van een timingdiagram.*

In tabel A.10 is de waarheidstabel voor de toestandsfuncties te zien. Merk op dat de uitgangen direct van de flipflops komen; er is dus geen uitgangslogica. In een aantal gevallen kan de waarde van Z_3 als don't care worden gespecificeerd. Zo is de opvolger van toestand 011 de toestand −10; het maakt niet uit wat de waarde van Z_3 is. Dat betekent echter wel dat voor beide codecombinaties (010 en 110) de juiste opvolger moet worden opgegeven. Noot: in tegenstelling tot gebruikelijk gebruiken we hier Z i.p.v. Q als functievariabelen.

Tabel A.10: *Waarheidstabel met toestandsfuncties voor machine.*

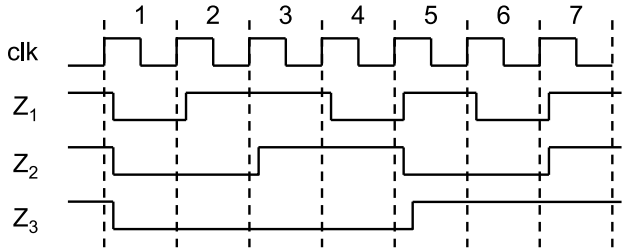
Z_3^n	Z_2^n	Z_1^n	Z_3^{n+1}	Z_2^{n+1}	Z_1^{n+1}
0	0	0	0	1	1
0	0	1	1	1	1
0	1	0	–	0	1
0	1	1	–	1	0
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	–	0	1
1	1	1	–	0	0

De toestandsfuncties zijn eenvoudig van aard:

$$\begin{aligned} Z_3^{n+1} &= Z_0^n \\ Z_2^{n+1} &= \overline{Z_1^n} + \overline{Z_2^n} \cdot Z_0^n \\ Z_1^{n+1} &= \overline{Z_1^n} + \overline{Z_0^n} \end{aligned} \tag{A.9}$$

Uitwerking opgave 11.22.

In het timingsdiagram is te zien dat uitgang Z_1 per cyclus drie keer 0 is en vier keer 1. Er zijn dan minstens twee (extra) flipflops nodig om de een toestandsmachine te ontwerpen die deze cyclus uitvoert. Het volledige timingsdiagram is te vinden in figuur A.14.



Figuur A.14: *Timingdiagram van een toestandsmachine.*

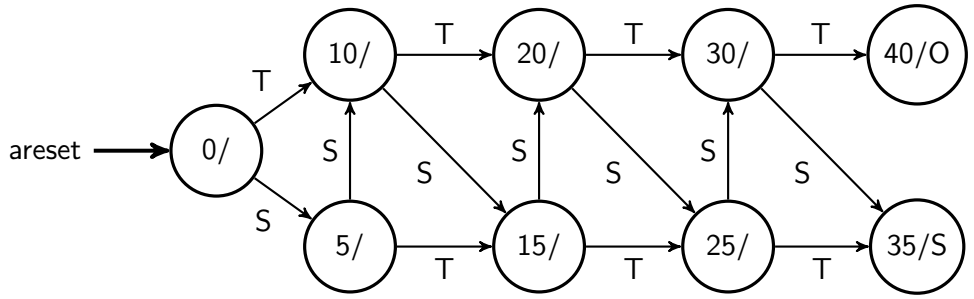
De machine doorloopt de cyclus: 000-001-011-010-101-100-111. Toestand 100 komt niet voor.

Uitwerking opgave 11.23.

In figuur A.15 is het gestileerde toestandsdiagram te zien van een herkenner voor 35 cent. Geaccepteerde muntstukken zijn 5 cent (S) en 10 cent (T). Er zijn in totaal negen toestanden. Het is mogelijk om het bedrag geheel in 5 cent muntstukken in te werpen, dus elke toestand vertegenwoordigt een veelvoud van 5 cent.

Uitwerking opgave 11.24.

Het hoogste bedrag dan kan worden ingeworpen is 40 cent. Het kan als volgt beredeneerd worden. Het gewenste bedrag is 35 cent. De kleinste bedrag dat kan worden ingeworpen



Figuur A.15: *Herkenner voor het herkennen van 35 cent (Moore).*

is 5 cent. Het bedrag dat het dichts bij 35 cent ligt is dus 30 cent, namelijk gewenst bedrag minus kleinste munt. In deze situatie kan nu de grootste munt worden ingeworpen, dus 10 cent. Dat betekent dat er nooit meer dan 40 cent kan worden ingeworpen. In formulevorm:

$$hb = gb - k + g \tag{A.10}$$

Hierin is hb het hoogste bedrag, gb is gewenste bedrag, k is kleinste munt en g is grootste munt. Deze formule geldt alleen als de waarden van de toegestane munten een tweevoud zijn.

Uitwerking opgave 11.25.

De machine heeft negen unieke toestanden. Dat is als volgt uit te rekenen. Het hoogste bedrag is 40 cent. De kleinste munt is 5 cent. Dat betekent dat als er alleen maar muntstukken van 5 cent in worden geworpen de machine in de toestanden 5 cent, 10 cent, 15 cent, ..., 35 cent (40 cent kan op deze manier niet worden gehaald). Het aantal unieke toestanden is dus 40 cent gedeeld door 5 cent plus 1 want 0 cent is ook een toestand. In formulecvorm:

$$t = \frac{hb}{k} + 1 = \frac{gb - k + g}{k} + 1 = \frac{gb + g}{k} \tag{A.11}$$

Hierin is t het aantal unieke toestanden en de overige variabelen hebben dezelfde betekenis als in de vorige opgave. Ook deze formule geldt alleen als de waarde van de muntstukken een tweevoud is.

Uitwerking opgave 11.26.

De munten zijn allemaal een tweevoud van elkaar, dus kunnen de formules uit de vorige opgave gebruikt worden. Het hoogste bedrag is:

$$hb = gb - k + g = 35 - 5 + 20 = 50 \tag{A.12}$$

dus 50 cent. Het aantal unieke toestanden is:

$$t = \frac{gb + g}{k} = \frac{35 + 20}{5} = 11 \tag{A.13}$$

Er zijn dus 11 unieke toestanden.

Uitwerking opgave 11.27.

Zo op het eerste gezicht is er geen vereenvoudiging mogelijk. Geen van de toestanden heeft een identiek opvolger en uitgangswaarde. Zie hiervoor tabel A.11.

Tabel A.11: *Toestandstabel voor machine.*

toestand	opvolger		uitgang
	0	1	
A	B	C	0
B	B	D	0
C	D	C	0
D	D	F	0
E	E	F	0
F	D	E	1

We zien echter het volgende bij toestand D en E. Als de machine in D staat en er wordt een 0 gedetecteerd, dan blijft de machine in D. Bij een 1 gaat de machine naar F. Dat geldt ook voor toestand E. De machine blijft in toestand E als een 0 wordt gedetecteerd. Bij een 1 gaat de machine naar toestand F. In toestand F gaat de machine naar D bij een 0 en naar E bij een 1.

Als de machine nu de volgende toestanden doorloopt: $D-D-D-D-F-D-D$ dan is de uitgangsreactie $0-0-0-0-1-0-0$. Diezelfde uitgangsreactie is ook te zien als de machine de toestanden $E-E-E-E-F-E-E$ doorloopt. Het is voor een toeschouwer niet te achterhalen of de machine nu in toestand D of E staat (als de machine niet in F staat natuurlijk). Ergo, toestanden D en E zijn identiek en kunnen samengenomen worden. Zie tabel A.12.

Tabel A.12: *Toestandstabel voor machine.*

toestand	opvolger		uitgang
	0	1	
A	B	C	0
B	B	D/E	0
C	D/E	C	0
D/E	D/E	F	0
F	D/E	D/E	1

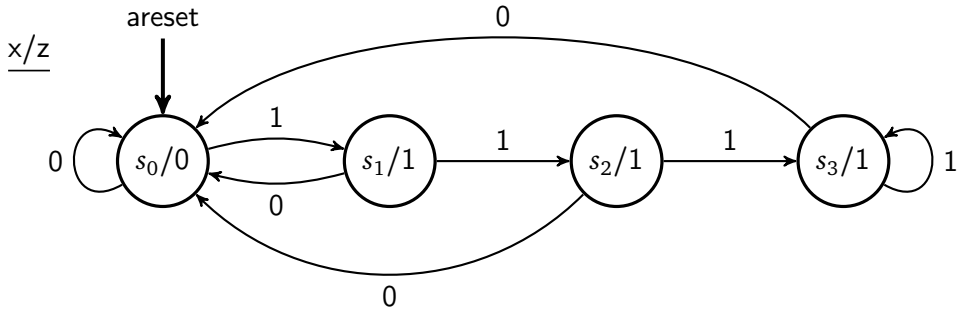
Uitwerking opgave 11.28.

We stellen een waarheidstabel op uit de functies. Merk trouwens op dat de nieuwe waarden van de Q's logisch 0 zijn als X logisch 0 is. Hieronder de tabel na invullen. Zie tabel A.13.

Uit de waarheidstabel is het volgende toestandsdiagram te tekenen, waarbij de volgende toestandscodering wordt gebruikt: $s_0 = 00$, $s_1 = 01$, $s_2 = 10$ en $s_3 = 11$. Zie figuur A.16.

Tabel A.13: *Waarheidstabel bij toestandsfuncties van opgave 11.28.*

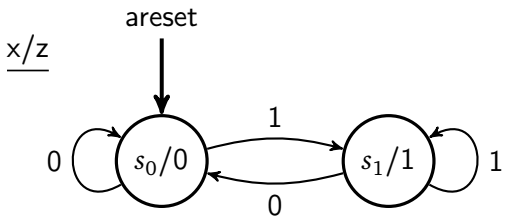
Q_1^n	Q_0^n	X^n	Q_1^{n+1}	Q_0^{n+1}	Z^n
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	0	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	1	1	1



Figuur A.16: *Toestandsdiagram machine opgave 11.28.*

Het is mogelijk dit toestandsdiagram te vereenvoudigen. Vereenvoudigen houdt in dat twee (of meer) toestanden te combineren zijn (zijn dan *identiek*) en te vervangen zijn door één toestand. Twee toestanden zijn identiek als ze onder dezelfde invoer(reeks) dezelfde uitvoer(reeks) genereren ongeacht welke van de twee toestanden de begintoestand is.

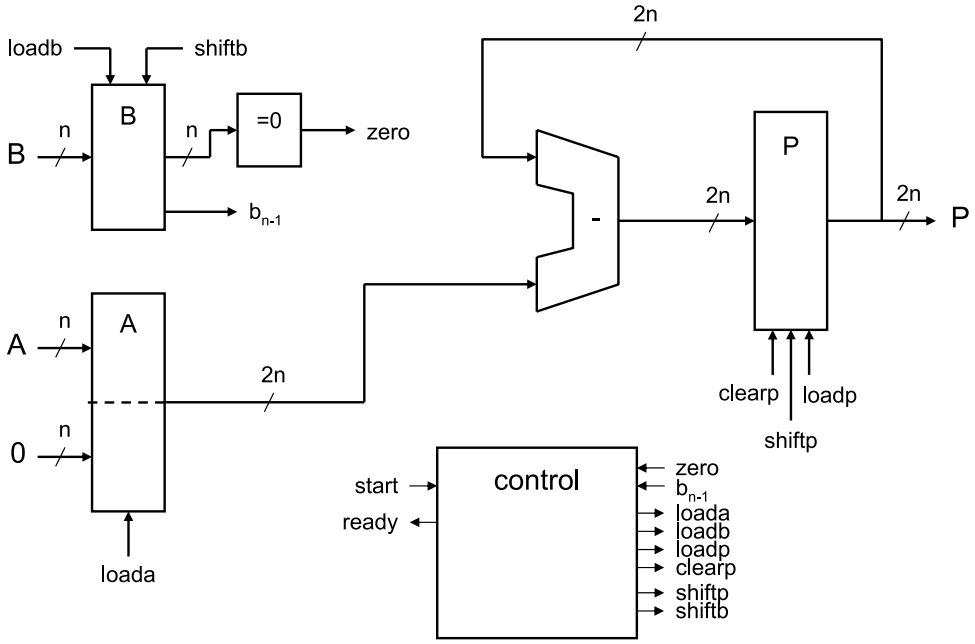
Bekijk toestand s_2 en s_3. Als vanuit s_2 of s_3 een aantal logische 1-en óf een logische 0 wordt verwerkt, is de uitvoer hetzelfde. Toestand s_2 en s_3 zijn dus identiek en kunnen worden gecombineerd. Dat geldt ook voor toestand s_1 en s_2. Dus toestand s_1 en s_2 zijn identiek én toestand s_2 is identiek aan s_3. De toestanden s_1, s_2 en s_3 zijn dus identiek en kunnen worden gecombineerd. In figuur A.17 is het geminimaliseerde toestandsdiagram getekend. Merk op dat dit het toestandsdiagram van een D-flipflop is.



Figuur A.17: *Toestandsdiagram machine opgave* **????**.

Uitwerking opgave 12.1.

In figuur A.18 is de derde versie van de vermenigvuldiger te zien. We gaan nu P naar links schuiven. We schuiven A niet. P krijg dus een extra stuursignaal $shiftp$ en A heeft slechts alleen nog een load-signaal. Verder is te zien dat de optelacties afhankelijk zijn van het meest significante bit van B. Dat komt omdat we P nu steeds gaan links schuiven. Zo worden na elke optelling ($P = P_A$) de meer significante bits netjes bij P opgeteld.



Figuur A.18: *Derde versie van de sequentiële vermenigvuldiger. Hierbij wordt P geschoven.*

Het toestandsdiagram, te zien in figuur A.19 wordt echter uitgebreid met één toestand omdat het laden van P en het schuiven van P niet tegelijkertijd kan plaatsvinden. Verder is te zien dat het optellen van A bij P alleen mag gebeuren als er nog enen in B zijn, anders wordt P één keer teveel geschoven.

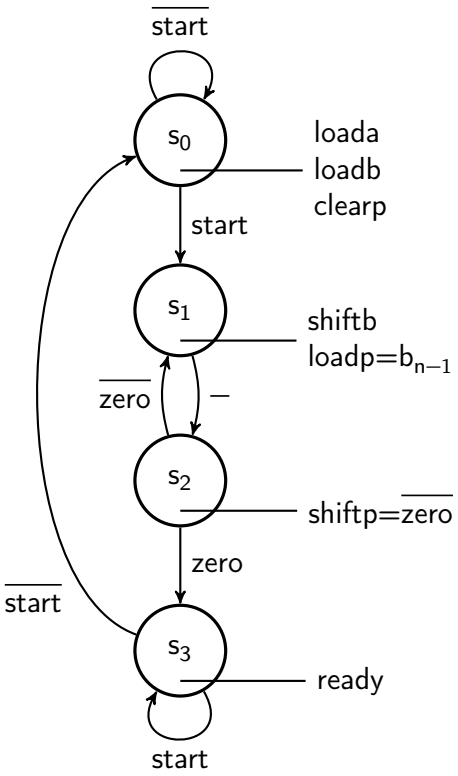
Uitwerking opgave 12.2.

P.M.

Uitwerking opgave 12.3.

Het toestandsdiagram moet worden aangepast zodat de inhoud van P bewaard blijft als de machine in de rusttoestand is (hier s_0). We kunnen dit aanpassen door het toevoegen van een extra toestand zoals te zien is in figuur A.20. In toestand s_0 wordt alleen gewacht totdat $start$ geactiveerd wordt. Pas in toestand s_1 worden diverse registers geladen of op gewist.

Dit is niet de enige mogelijkheid. Het is ook mogelijk om het laden van A en B en het wissen van P in toestand s_0 afhankelijk te maken van de waarde van $start$. Als $start$ geactiveerd is worden de registers aangepast. Als $start$ niet geactiveerd is (en de machine blijft dan in s_0) dan worden de inhouden van A, B en P niet aangepast. Dit is het gedrag (van de uitgangen) van een Mealy-machine.



Figuur A.19: *Het toestandsdiagram van de besturing van de derde versie vermenigvuldiger.*

Uitwerking opgave 12.4.
P.M.

Uitwerking opgave 12.5.
We kunnen de vermenigvuldiging van de twee 4-bits getallen A en B als volgt noteren:

$$A \times B = (2^n - 1) \times (2^n - 1) = 2^{2n} - 2 \cdot 2^n + 1 = 2^{2n} - 2^{n+1} + 1 \tag{A.14}$$

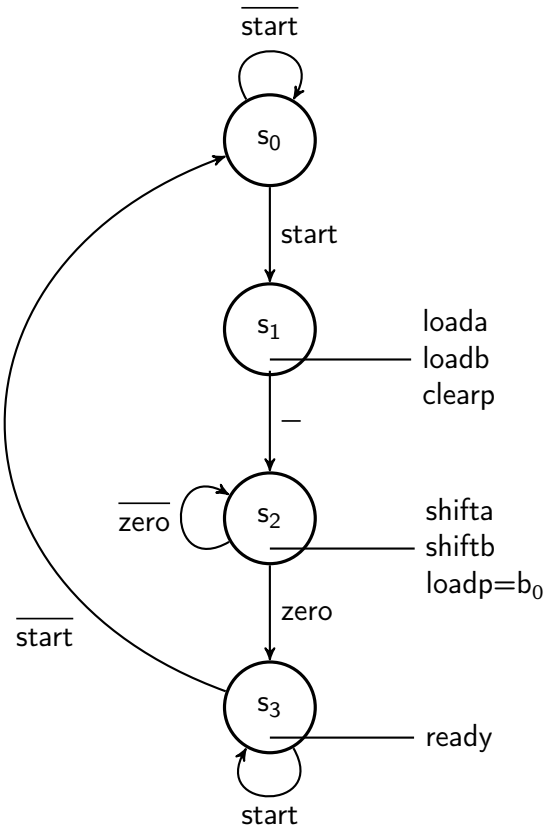
Verder geldt dat een $2n$-bits getal de maximale waarde heeft van $2^{2n} - 1$. We kunnen dus stellen dat:

$$2^{2n} - 1 > 2^{2n} - 2^{n+1} + 1 \tag{A.15}$$

Dit geldt voor elke $n > 0$.

Uitwerking opgave 12.6.
Als we signaal $loadp$ continue verbinden met signaal b_0, dan kunnen we het toestandsdiagram weergeven zoals is te zien in figuur A.21. De vraag is of dit toestandsdiagram een goedwerkende machine oplevert. We kunnen in ieder geval zeggen dat er voor toestand s_1 niets veranderd. We moeten alleen de toestanden s_0 en s_2 bekijken.

Te zien is dat in toestand s_0 zowel $clearp$ als $loadp$ geactiveerd worden. Als we ervoor zorgen dat $clearp$ overheersend is t.o.v. $loadp$ dan werkt de machine in deze toestand



Figuur A.20: *Het toestandsdiagram van de besturing van de twee versie van de sequentiële vermenigvuldiger met extra toestand zodat de inhoud van P bewaard blijft in toestand s_0.*

correct. In toestand s_2 is de waarde van b_0 altijd 0. Dat komt omdat de machine vanuit s_1 naar s_2 gaat als alle bits van B 0 zijn. Ook in deze toestand werkt de machine correct.

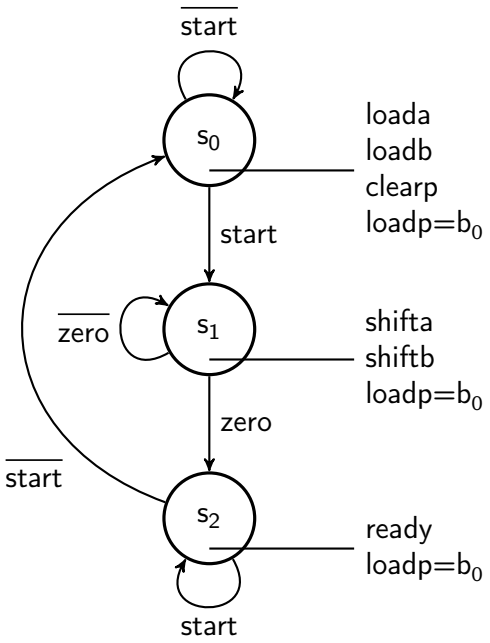
Resumerend kunnen we zeggen dat de machine correct werkt als signaal $clearp$ overheersend is t.o.v. signaal $loadp$.

Uitwerking opgave 12.7.

In listing A.1 is de pseudo-code voor de enenteller te zien. Er zijn twee variabelen (of registers) nodig. Register A is een teller en register B is een schuifregister dat naar rechts kan schuiven.

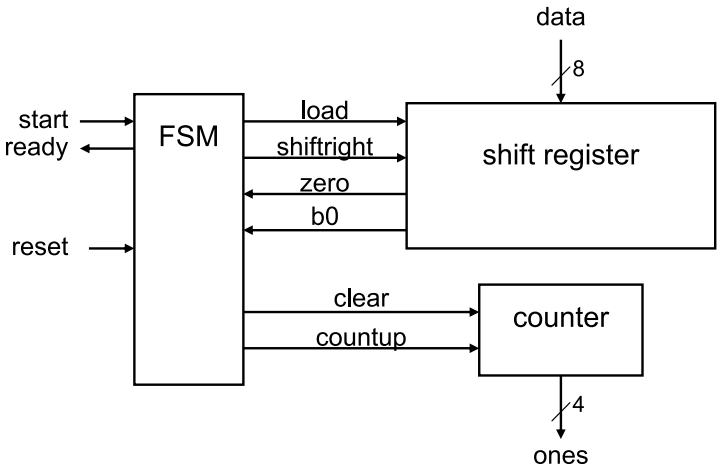
```
A := 0;
B := input;
while B <> 0 do
    if b0 = 1 then
        A = A+1;
    end if;
    right-shift B;
end while;
```

Listing A.1: *Pseudo-code enenteller.*



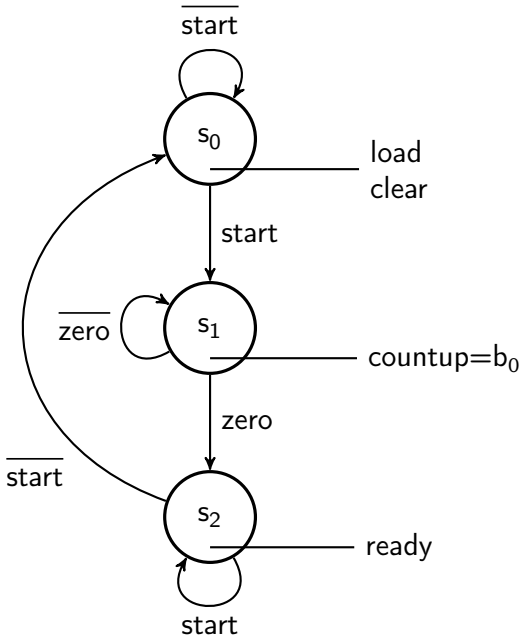
Figuur A.21: *Het toestandsdiagram van de besturing van de twee versie van de sequentiële vermenigvuldiger met b_0 verbonden aan $loadp$.*

In figuur A.22 is het datapad van de enenteller te zien. *A* is uitgevoerd als een teller die op 0 gezet kan worden middels een $clear$-ingang en met één verhoogd kan worden middels een $countup$-ingang. De telstand wordt via signaal *ones* naar buiten uitgevoerd. Het schuifregister heeft een $load$-ingang om een 8-bits waarde te laden en een $shift right$-ingang om de inhoud één plaats naar rechts te schuiven. Er wordt een 0 ingeschoven (niet getekend). Het schuifregister geeft twee statussignalen terug, te weten het signaal $zero$ dat logisch 1 wordt als alle bits in het schuifregister 0 zijn en een signaal $b0$ dat het minst significante bit uit het schuifregister aan de toestandsmachine (FSM) aanbiedt. De FSM bestuurd het datapad. Het heeft een $start$-ingang waarmee een startopdracht kan worden gegeven. Als de machine klaar is wordt signaal $ready$ geactiveerd.



Figuur A.22: *Het datapad voor het bepalen van het aantal enen in een register.*

In figuur A.23 is het toestandsdiagram van de machine te zien. Het begint met toestand s_0 waar wordt gewacht totdat $start$ is geactiveerd. In deze toestand wordt het schuifregister geladen en de teller op 0 gezet (clear). In toestand s_1 wordt gewacht zolang het schuifregister niet 0 is. De $countup$-ingang wordt geactiveerd als b_0 logisch 1 is. Dit is dus een Mealy-uitgang. Als het schuifregister 0 is wordt naar toestand s_2 gegaan. In deze toestand wordt de $ready$-uitgang geactiveerd.



Figuur A.23: *Het toestandsdiagram van de besturing van de enenteller.*

Uitwerking opgave 13.1.

De `nop`-operatie is handig als de ALU "niets" moet doen. Te denken valt aan een sprong-instructie waarbij de PC geladen worden met een nieuw adres. De ALU moet dan in ieder geval de flags niet aanpassen en hoeft ook geen bewerking te doen.

Uitwerking opgave 13.2.

De `pass`-operatie is handig als de inhoud van een register of een constante ongewijzigd in een (ander) register moet worden geladen. De flags worden aangepast.

Uitwerking opgave 13.3.

Er zijn zeker meer logische operaties te verzinnen. Te denken valt aan NOR, NAND en EXNOR. Alleen zijn deze operaties uit te voeren door een OR-, AND- of EXOR-operatie uit te voeren gevolgd door een NOT-operatie. Ze zijn dus niet echt nodig in een processor.

Uitwerking opgave 13.4.

We laten eerst de bitpatronen met onderdeel van de processor zien. Waarden die niet van toepassing zijn, zijn gecodeerd als don't care.

$sela = 10$
$selb = 011$
$seld = 01$
$acode = 010$
$constant = --------$
$end = 1$
$jcode = 000$
$address = --------$

Het gehele bitpatroon is dus:

```
1001101010--------1000-------
```

Uitwerking opgave 13.5.
De instructie **`and`** `R1,R2,R3` zorgt ervoor dat de bit-voor-bit AND van *R*2 en *R*3 wordt geladen in *R*1, d.w.z. dat de statement *R*1 := *R*2 and *R*3 wordt uitgevoerd. De instructie **`and`** `R1,R3,R2` doet precies hetzelfde alleen zijn nu de bronregisters omgedraaid maar dat maakt voor de AND niet uit. Er zijn dus twee instructies beschikbaar voor het realiseren van de AND. Hetzelfde geldt voor de OR en de EXOR.

Uitwerking opgave 13.6.
Bij linksom roteren wordt de huidige waarde van de carry flag in het minst significante bit geschoven. Het meest significante bit wordt in de carry flag geschoven. Dit gebeurt allemaal op een actieve klokflank. Bij rechtsom roteren wordt de huidige waarde van de carry flag in het meest significante bit geschoven en wordt het minst significante bit in de carry flag geschoven. Ook dit gebeurt op een actieve klokflank. Goed beschouwd kunnen we zeggen dat roteren over 9 bits gebeurt. Nog een opmerking: de waarde van de carry flag vóór roteren kan een andere waarde hebben dat de carry flag ná roteren.

Uitwerking opgave 13.7.
Het programma kan niet zomaar verplaatst worden naar een ander (begin-)adres. Dat komt omdat de sprongadressen zijn berekend ten opzichte van het beginadres (hier dus adres 0). Als we het programma willen verplaatsen, dan moeten de sprongadressen opnieuw berekend worden. Dit wordt het *relocatieprobleem* genoemd. Veel moderne processoren zijn daarom uitgerust met een *relatieve spronginstructie* waarbij gesprongen worden naar een locatie relatief ten opzichte van de Program Counter. Een programma kan dan zonder veel problemen verplaatst worden naar een ander deel van het programmageheugen. Noot: eenzelfde probleem doet zich voor als een processor uit RAM-geheugen informatie ophaalt of wegschrijft.

Uitwerking opgave 13.8.
Voor het uitvoeren van de **`sub`**-instructie hoeft de carry niet expliciet op 0 gezet worden. De optelling in de **`add`**-instructie direct voor de **`sub`**-instructie zorgt er namelijk voor dat de uitgaande carry altijd 0 is omdat een 8x8-bits vermenigvuldiging een 16-bits resultaat

oplevert. We kunnen dit wiskundig aantonen:

$$P_{max} = A_{max} \times B_{max} = (2^n - 1) \times (2^n - 1) = 2^{2n} - 2^{n+1} + 1 \tag{A.16}$$

De uitkomst is altijd kleiner dan of gelijk aan $2^{2n}-1$, de maximale waarde van een 16-bits getal verdeeld over twee registers. Er treedt dus nooit een uitgaande carry op.

Uitwerking opgave 13.9.

De instructie **not** `Rx,#const` doet niets anders dan de inverse waarde van de constante laden in een register. Natuurlijk kan deze instructie uitgevoerd worden ook al lijkt ze onzinnig. We kunnen echter ook zelf de inverse berekenen en als constante opgeven.

Uitwerking opgave 13.10.

De NOT-operatie kan ook worden uitgevoerd door een EXOR-operatie met allemaal enen. We kunnen dus ook schrijven: **exor** `Rx,Rx,255` waarbij *Rx* één van de registers voorstelt. De **not**-instructie is dus in wezen overbodig.

Uitwerking opgave 13.11.

Het assembler-programma voor het bepalen van het aantal enen in de input is te zien in listing A.2. Het begint met het laden van een aantal registers. Register *R*3 wordt gebruikt voor het tellen van de enen. In register *R*1 wordt bijgehouden hoeveel bits zijn verwerkt. Register *R*2 bevat een kopie van de ingangen.

```
1  0:    ld   R3,#0       ; holds counted 1s
2  1:    ld   R1,#8       ; holds counter
3  2:    ld   R2,IN       ; reads inputs
4  3:    shr  R2,R2       ; right shift, low bit in carry flag
5  4:    add  R3,R3,#0    ; add carry flag
6  5:    ld   F,#0        ; clear carry
7  6:    sub  R1,R1,#1    ; decrement counter
8  7:    jne  @3          ; and repeat
9  8:    ld   R0,R3       ; count to output
10 9:    jmp  @9          ; endless loop
```

Listing A.2: *Assembler-code voor het tellen van enen in de input.*

Vervolgens wordt register *R*2 eenmaal naar rechts geschoven waardoor het minst significante bit in de carry flag wordt geplaatst. We tellen vervolgens 0 bij register *R*3 waarbij ook de carry flag wordt opgeteld. Daarna zetten we de carry flag op 0. We verlagen register *R*1 met één. Merk op dat de carry flag dus eerst op 0 gezet moet worden voordat we de aftrekoperatie uitvoeren omdat die ook de carry flag meeneemt. De aftrekoperatie past ook de zero flag aan. Als de uitkomst dus 0 is wordt de zero flag op 1 gezet (en anders op 0). We hoeven dus niet expliciet te testen of register *R*1 gelijk aan 0 is, dat wordt bij de aftrekoperatie al automatisch gedaan. Als register *R*1 ongelijk aan 0 is wordt opnieuw naar de schuifoperatie gesprongen. Zo niet, dan zorgt de laatst sprongopdracht voor een eeuwig durende lus.

Uitwerking opgave 13.12.
De lus begint bij de instructie op adres 3 en loopt tot en met de instructie op adres 7. In totaal duurt het uitvoeren van de lus dus 5 klokpulsen.

Uitwerking opgave 13.13.
De uitwerking is te zien in listing A.3. Eerst wordt $R1$ vergeleken met 5. Als dat niet het geval is wordt gesprongen naar adres 4 en wordt $R2$ geladen met 10. Als $R1$ gelijk is aan 5, wordt er niet gesprongen en wordt de volgende instructie uitgevoerd, het laden van $R2$ met 8. Daarna wordt gesprongen naar adres 5 zodat de `ld`-instructie op adres 4 niet wordt uitgevoerd.

```
0:      cmp  R1,#5        ; Compare with 5
1:      jne  @4           ; If not true, jump
2:      ld   R2,#8        ; R2 := 8
3:      jmp  @5           ; and jump ...
4:      ld   R2,#10       ; R2 := 10
5:      ...
```

Listing A.3: *Assembler-code voor een beslissing.*

Uitwerking opgave 13.14.
Deze opzet werkt niet correct omdat alle `sub`-instructies de zero flag aanpassen, dus ook als register $R3$ van 1 naar 0 gaat. Neem als voorbeeld de het getal 010000_{16} (met hexadecimale notatie is het makkelijk te zien). Na een doorloop van de lus is de teller op $00ffff_{16}$ aangekomen en is de zero flag op 1 gezet omdat register $R3$ van 0 naar 1 is gegaan.

Uitwerking opgave 13.15.
We kunnen een aantal doorgangen van de lus berekenen met de formule:

$$\text{aantal keer lus doorlopen} = \frac{f_{proc} \cdot \text{wachttijd}}{\text{klokpulsen per lusdoorloop}} \tag{A.17}$$

Na invullen van de gegevens volgt dat:

$$\text{aantal keer lus doorlopen} = \frac{50000000 \cdot 0{,}666666\ldots}{4} = 8333333{,}333\ldots \tag{A.18}$$

Het antwoord is niet een geheel getal. Dat betekent dat we de wachttijd niet exact kunnen realiseren. We ronden het aantal af op 83333333. Hierdoor komen we twee klokpulsen te kort. We kunnen de code eventueel aanvullen met twee `nop`-instructies. Merk op dat het getal 8333333 met twee verlaagd moet worden vanwege het laden van de registers en de manier waar de op de aftrekking en testen op de carry flag werkt.

Uitwerking opgave 13.16.
Na het uitvoeren van deze vijf instructies zijn de waarden (of inhouden) van registers $R1$ en $R2$ verwisseld. Dit wordt in de literatuur de *exor swap* genoemd. Zie ook

https://en.wikipedia.org/wiki/XOR_swap_algorithm. Het voordeel van deze manier van verwisselen van de waarden is dat er geen tijdelijke variabele (of register) nodig is. In listing A.4 zijn de waarden van $R1$ en $R2$ als commentaar aan de code toegevoegd.

```
1    ld   R1,#13      ; R1 = 0000.1101
2    ld   R2,#45      ; R2 = 0010.1101
3    exor R1,R1,R2    ; R1 = 0010.0000
4    exor R2,R2,R1    ; R2 = 0000.1101
5    exor R1,R1,R2    ; R1 = 0010.1101
```

Listing A.4: *Code exor swap.*