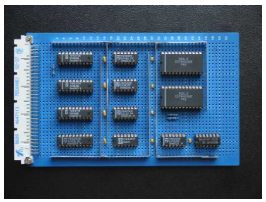
Digital- och datorteknik



Föreläsning #14

Biträdande professor Jan Jonsson

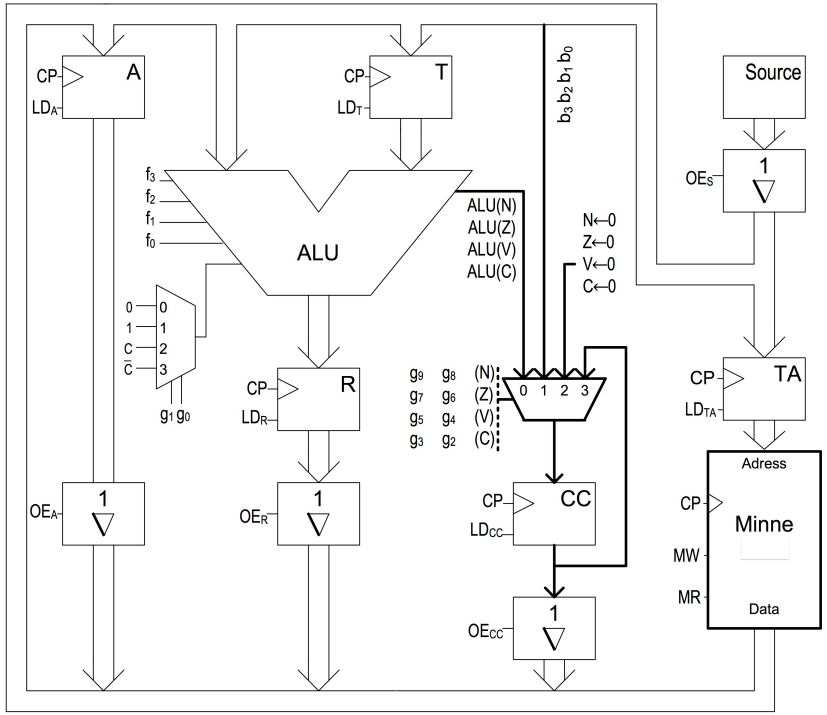
Institutionen för data- och informationsteknik
Chalmers tekniska högskola

Vår dator

Vad vi har åstadkommit hittills:

Med hjälp av kombinatoriska nät och sekvensnät har vi byggt upp en dataväg innehållande

- ett källregister (”source”)
- ett fåtal dataregister
- en ALU
- ett primärminne med 256 minnesregister

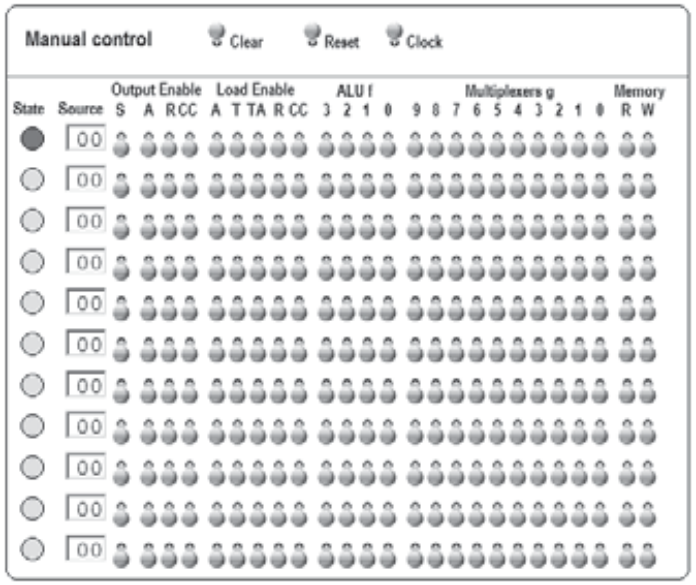


Vår dator

Vad vi har åstadkommit hittills:

Med hjälp av omkopplare kan vi manuellt styra datavägens komponenter och åstadkomma enkla RTN-operationer:

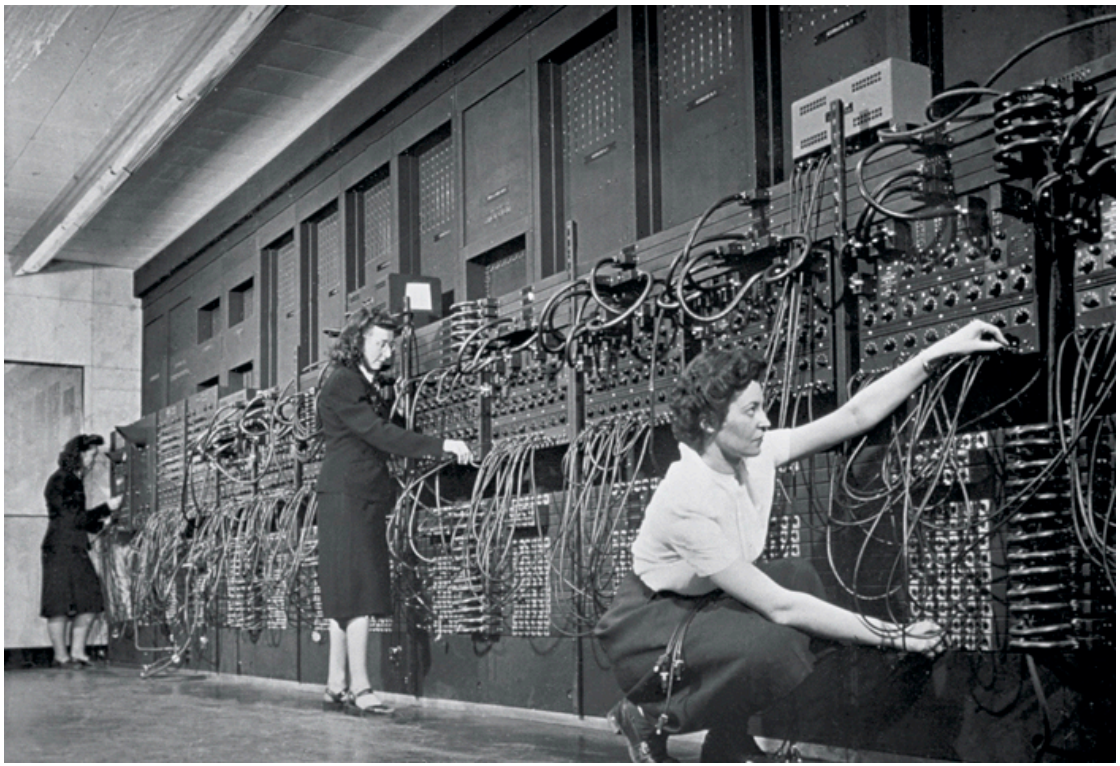
- flytta data mellan två register
- flytta data mellan ett register och en cell i primärminnet
- utföra aritmetiska och logiska operationer på data.



Med hjälp av flera uppsättningar omkopplare kan vi programmera datorn att utföra kortare sekvenser av enkla RTN-operationer.

Vår dator

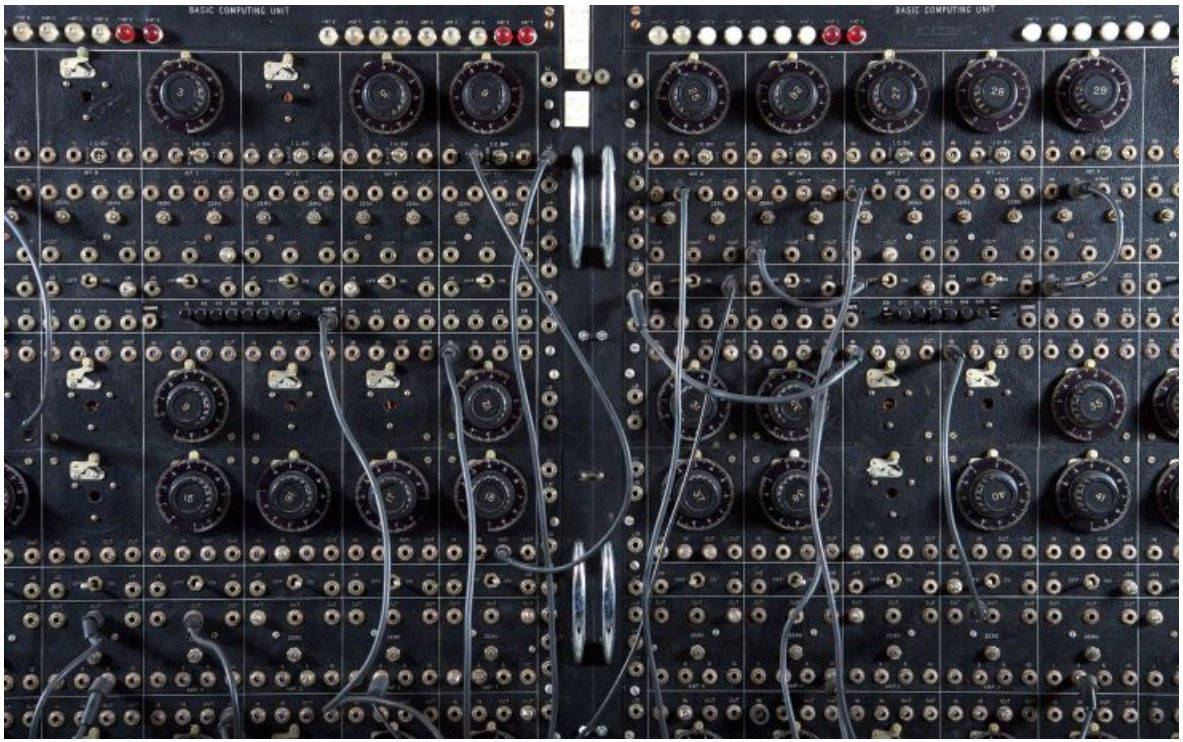
Detta liknar väldigt mycket vad som fanns på 40-talet:



Programmering av ENIAC, 1946.

Vår dator

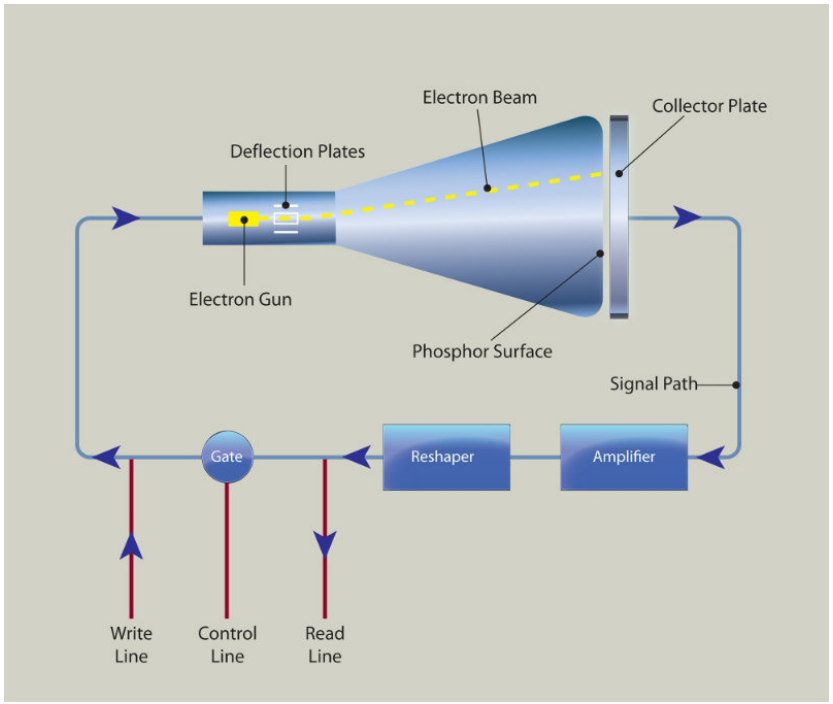
Detta liknar väldigt mycket vad som fanns på 40-talet:

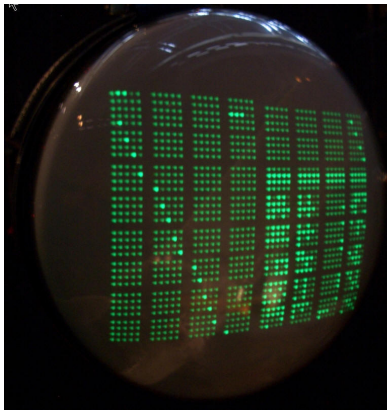


Programmeringsmodul för ENIAC.

Vår dator

Detta liknar väldigt mycket vad som fanns på 40-talet:





Minnesmodul motsvarande 256 bytes (Williams-Kilburn tube, 1947)

Vår dator

Hur kan datorn förbättras?

Att manuellt ge datorn instruktioner via RTN-operationer ger full kontroll över hur datavägen används. Det är dock väldigt lätt att göra misstag i sekvensen av styrsignaler.

Den som programmerar datorn behöver därför en mer abstrakt (= mindre detaljrik) modell av datavägen att arbeta med.

Att manuellt mata in RTN-operationer via en mängd omkopplare är dessutom ett mycket tidsödande sätt att beskriva uppgifter av mer omfattande och komplicerad natur.

Den som programmerar datorn behöver därför tillgång till en funktion som automatiskt genererar sekvenser av styrsignaler.

Vår dator

Programmerarens modell av datorn

I en modern dator beskrivs operationer på datavägen med hjälp av ett maskinspråk, vilket är en uppsättning maskininstruktioner (binära kodord) som är avsedda att avkodas av datorn. Den, för människan, läsbara representationen av ett maskinspråk kallas assemblerspråk.

Den som programmerar datorn med hjälp av maskinspråk kan nu bara direkt referera till en delmängd av datavägens register.

Den som programmerar datorn med hjälp av maskinspråk kan nu inte heller direkt ge styrsignaler till datavägen, ALU eller primärminne. Istället genereras dessa signaler i samband med att maskininstruktioner avkodas av datorn.

Vår dator

Programmerarens modell av datorn

De register som programmeraren behöver referera till beror på vilka typer av operationer som maskinspråket ger stöd för. En dator har generellt sett behov av följande typer av operationer:

- Kopiering av data mellan primärminne och dataregister
- Uttrycksevaluering (aritmetiska och logiska operationer)
- Villkorliga och ovillkorliga ändringar av programflödet

För vår dator innebär det att vi behöver kunna referera till register A och register CC samt primärminnets register.

Vi kommer snart att se att det också finns ett behov av register som ”pekar på” viktiga dataareor i primärminnet, samt register som håller reda på var i programflödet datorn befinner sig.

Vår dator

Instruktionsformat

En maskininstruktion består av en operationskod (1 byte) samt operandinformation (0 eller 1 byte).

Operationskoden kan ses som etiketten på en viss fördefinierad sekvens av RTN-operationer på datavägen.

Exempel:

Operationskod: 05

Assemblerkod: CLRA

Beskrivning: Nollställ register A (0 → A)

Steg	RTN-beskrivning	Aktiva styrsignaler
1	0 → R; ALU(N,V,Z,C) → CC	LD_R; LD_{CC}
2	R → A	OE_R; LD_A

Vår dator

Instruktionsformat

Till varje operationskod hör också följande information:

- Antal bytes: hur många bytes som operationskoden och dess operandinformation totalt tar i anspråk.
- Antal klockcykler: 1 + det antal steg som motsvarande fördefinierade sekvens av RTN-operationer omfattar.
- Adresseringsmetod: om sekvensen av RTN-operationer kräver några operander, och var man i så fall hittar dessa.
- Flaggpåverkan: om sekvensen av RTN-operationer påverkar datorns flaggbitar, och i så fall på vilket sätt.

Se sidan 5 i ”Instruktionslista för FLISP”

Instruktion	Adressering				Operation	Flaggor			
	metod	OP	#	~		N	Z	V	C
CLRA	Inherent	05	1	3	0 → A	0	1	0	0

Vår dator

Var lagrar vi instruktionerna?

Vi kommer snart att konstruera en styrenhet som automatiskt genererar unika sekvenser av RTN-operationer (styrsignaler) baserat på vilken operationskod som maskininstruktionen har.

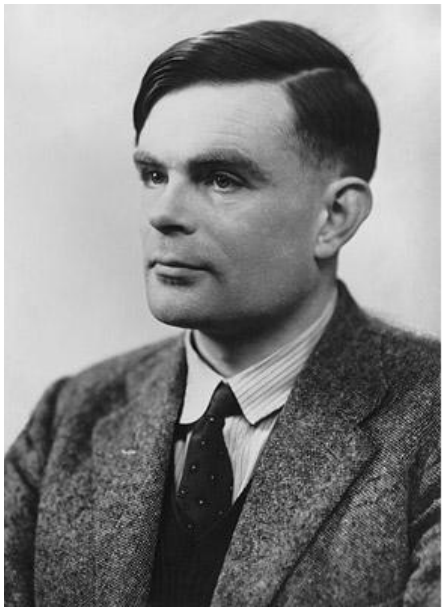
Datavägen utrustas därför med ett instruktionsregister I, som lagrar operationskoden för den aktuella maskininstruktionen. Innehållet i register I avkodas därefter av styrenheten för att generera styrsignaler.

Maskininstruktionerna lagras dock med fördel i primärminnet, då det ger flera fördelar jämfört med att använda omkopplare eller hålkort för att programmera datorn.

Denna metod kallas för ”det lagrade programmets princip”.

”Det lagrade programmets princip”

Alan Turing



John von Neumann

Alan Turing skrev redan 1936 om en hypotetisk datamaskin, ”universal Turing machine”, med ett oändligt minne som innehöll såväl data som instruktioner.

John von Neumann är dock den som associeras med principen via en rapport som skrevs 1945 i samband med utvecklandet av datorn EDVAC.

Vår dator

Egenskaper hos ”det lagrade programmets princip”

Att lagra programmet (maskininstruktionerna) i ett läs- och skrivbart minne erbjuder några intressanta möjligheter:

- Datoranvändaren kan relativt enkelt modifiera ett lagrat program.
- Datoranvändaren kan relativt enkelt byta ut ett lagrat program mot ett annat.
- Datorn kan köra program, exempelvis kompilatorer och assemblerare, som i sin tur genererar program som kan lagras i minnet.

Vår dator

Egenskaper hos ”det lagrade programmets princip”

Att lagra program tillsammans med data i ett läs- och skrivbart minne kan dock ge upphov till vissa problem:

- Om datorns dataväg bara har en databuss kommer data och instruktioner inte att kunna behandlas parallellt. Bussen kan därmed bli en flaskhals som påverkar datorns prestanda.

  Detta problem kan reduceras genom att man använder s k cacheminnen för att lagra instruktioner som används ofta.

- Programmet kan manipuleras, antingen medvetet eller av misstag, så att det inte längre fungerar som avsett.

  Detta problem kan elimineras genom att man använder s k minnesskydd på de delar av minnet där programmet lagras.

Vår dator

Egenskaper hos ”det lagrade programmets princip”

Att lagra program tillsammans med data i ett läs- och skrivbart minne kan dock ge upphov till vissa problem:

- Då både data och instruktioner lagras som bitsträngar kan datorn inte avgöra vilken tolkning en bitsträng skall ges.

  Den som programmerar datorn måste därför ange var i minnet programmets första maskininstruktion är lagrad.
  I vår dator används en resetvektor (minnesadress FF_{16}) för att lagra denna information.

  Genom att utrusta datavägen med ett speciellt register, kallat programräknare (PC), kan datorn själv hålla reda på var de efterföljande maskininstruktionerna är lagrade.

Vår dator

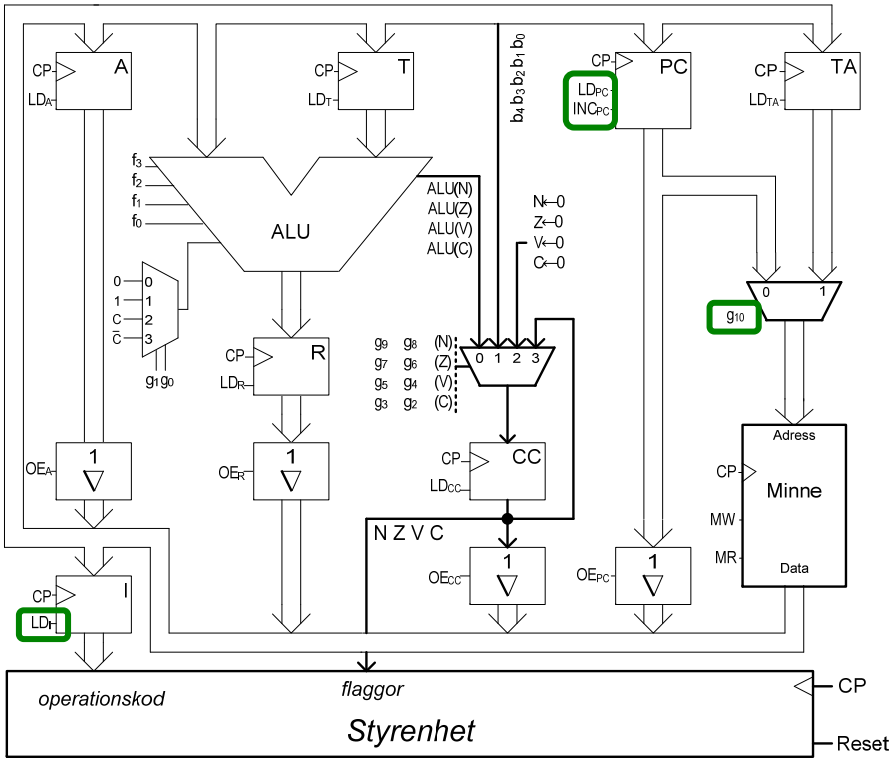
Dataväg med programräknare och instruktionsregister:

I vår sjätte version av datavägen har vi fått en programräknare, PC, som håller reda på var i minnet nästa maskininstruktion är lagrad.

Vi har också fått ett register I som lagrar operationskoden för den aktuella maskininstruktionen.

Nya styrsignaler som läggs till:

- LD_I för att välja om register I skall uppdateras eller ej.
- LD_{PC} för att välja om register PC skall uppdateras eller ej.
- INC_{PC} för att välja om värdet i PC skall ökas med 1 eller ej.
- g_{10} för att välja om PC eller TA adresserar minnet.



Vår dator

Den automatiska styrenheten:

Vi har identifierat tre faser i den automatiska styrenhetens sätt att arbeta:

- Återställningsfas (RESET): Uppstart av styrenheten.

  Platsen för programmets första maskininstruktion tas fram.
- Hämtfas (FETCH): Hämta en ny maskininstruktion.

  Operationskoden för maskininstruktionen lagras i register I.
- Utförandefas (EXECUTE): Utför den nya instruktionen.

  Innehållet i register I (operationskoden) avkodas och den till operationskoden tillhörande sekvensen av RTN-operationer genomlöps.

Vår dator

Den automatiska styrenheten:

RTN-operationerna för RESET och FETCH är:

Återställningsfas (RESET)

Steg	RTN-beskrivning	Aktiva styrsignaler
0	$FF_{16} \rightarrow R;$	$f_1; f_0; LD_R$
1	$R \rightarrow TA$	$OE_R; LD_{TA}$
2	$M(TA) \rightarrow PC$	$MR; g_{10}; LD_{PC}$

Hämtfas (FETCH)

Steg	RTN-beskrivning	Aktiva styrsignaler
3	$M(PC) \rightarrow I;\ PC + 1 \rightarrow PC$	$MR; LD_I; INC_{PC}$

Vår dator

Den automatiska styrenheten:

För att styrenheten skall vara helt automatisk måste den få en indikation om att utförandefasen för aktuell maskininstruktion är avslutad, och därmed kunna återgå till FETCH-fasen.

Vi lägger därför till en styrsignal NF (NewFetch) som skall aktiveras i slutet av EXECUTE-fasen.

Exempel:

Utförandefas (EXECUTE) för CLRA

Steg	RTN-beskrivning	Aktiva styrsignaler
4	$0 \rightarrow R$; ALU(N,V,Z,C) $\rightarrow$ CC	LD_R; LD_{CC}
5	$R \rightarrow A$	OE_R; LD_A; NF

Vår dator

Den automatiska styrenheten:

Vi kan nu konstruera det sekvensnät som skall realisera styrenhetens tre faser (uppgift 13.7 i Arbetsboken):

- Räknesekvensen är 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0,...

  Tillstånd 0, 1 och 2 motsvarar RESET

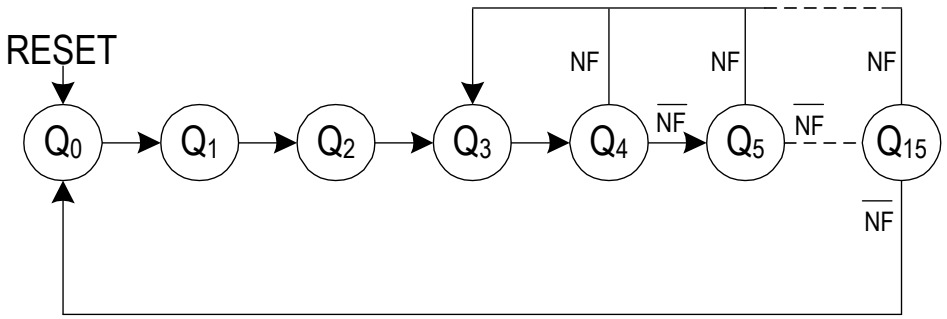
  Tillstånd 3 motsvarar FETCH

  Tillstånd 4 … 15 reserveras för de sekvenser av RTN-operationer som utförs i EXECUTE-fasen för aktuell maskininstruktion
- Signalen RESET = 1 tvingar maskinen till tillstånd 0.
- Signalen NF = 1 tvingar maskinen till tillstånd 3.

Vår dator

Den automatiska styrenheten:

Vi kan nu konstruera det sekvensnät som skall realisera styrenhetens tre faser (uppgift 13.7 i Arbetsboken):



I början av nästa läsperiod kommer vi att konstruera det kombinatoriska nätet för styrenheten.