A3 Computer Architecture

Engineering Science

3rd year A3 Lectures

Prof David Murray

david.murray@eng.ox.ac.uk
www.robots.ox.ac.uk/∼dwm/Courses/3CO

Michaelmas 2000

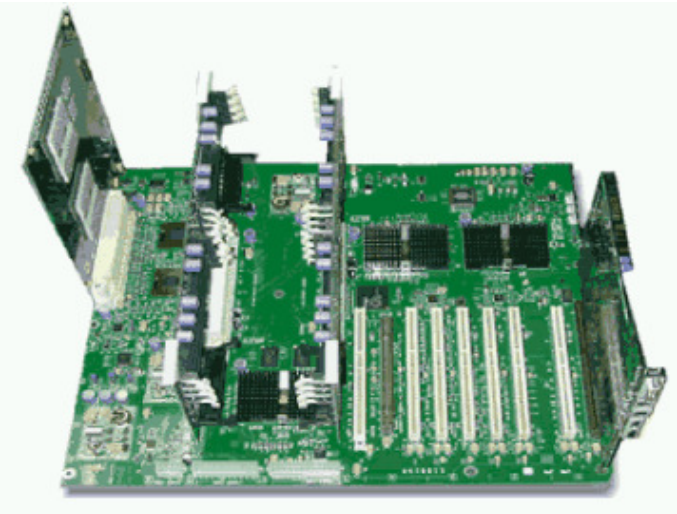
Computer Architecture

3A3 Michaelmas 2000

Prof D W Murray

Overview

- The development of the the digital computer must rank as one of the two engineering achievement which have has most impact on the progress of this century.
- Open up the box on any PC, and one is likely to be (i) impressed and (ii) daunted by the sheer complexity — even of what one can see. One is looking at the cumulative wizardry of, say, 10^6 engineer-years of development.



- Not surprising if you personally could not reproduce it tomorrow.
- BUT — you already know all you need to build a computer.

After the 1st year you know ...

- ... how to build combinational logic including larger elements such as ROMs, PLAs
- ... how to design sequential logic
- ... how to design machines with a finite number of identifiable states.
- **But you have probably/certainly forgotten it all ...**

In these lectures ...

- You are going to exploit what you know to discover how the most sophisticated sequential machine — **the general purpose computer** — works.
- The emphasis is on computer **architecture** rather than on computer **buildings**.
- Computer designs are diverse, often creating confusion.
  However, as with conventional architecture, although the detailed shape and plumbing is different, the underyling principles are obeyed ...
  ... and the differences are the result of engineering design considerations.
- We will establish a **Bog Standard Architecture** to expose the necessities —
  providing a basis for understanding more complex designs.

Lecture Content

- Lecture 1 revises Sequential Logic and introduces RTL.
- Lectures 2–5 deal with the basic operation of the core of the computer.
- Lectures 6–8 extend the picture to include the hardware and software functionality of the typical machine.

1. Revision and RTL
2. Overall structure of CPU and Memory. Instruction Fetching.
3. Control 1: The random logic controller.
4. ALU and Memory hardware. The Status word.
5. Control 2: Microprogrammed controller
6. The macro level. Memory addressing and stacks.
7. I/O, programmed and interrupt-driven.
8. Memory hierarchies ++

A hierarchy of languages

You will see ermerging a hierarchy of representations of machine operation.

- The operational description at the lowest (micro-) level will be in terms of RTL
- We will introduce Assembler Language as a way of describing several RTL statements (the macro-level)
- We shall look at how certain aspects of languages at the high level get compiled.

Books ...

- Hill and Peterson, “Digital Logic and Microprocessors”, John Wiley and Sons, New York, ISBN 0-471-82979-X
  - **Essential reading** up to and including Chapter 10 on RTL.
  - Chapter 11- on small computer organization is showing its age.
- Clements, “The Principles of Computer Hardware” 2nd ed, Oxford University Press, ISBN 0-19-853764-6
  - Early chapters cover basic combinational and sequential logic
  - Chapter 5 onwards are **essential reading**
- Tanenbaum, Structured Computer Organization (1998 edition)
  - focusses on overall operation of computer systems
  - stresses the hierarchy of levels of detail at which the operation can be viewed
  - More emphasis is given to the higher levels than other texts.
  - Introduces description based on virtual machine and Java.
- **YOU MUST READ!** Don’t merely rely on the notes.

1: Revision and Register Transfer Language

3A3 Michaelmas 2000

State machines using flip-flops: revision

Problem: Design using JK flip flops a modulo-4 counter that if x=0 would count repeatedly from 0,1,2,3,... and if x=1 would cound 3,2,1,0,...

Solution. If we designate the output bits as the outputs Q_0 and Q_1 of the JKs, then there are 4 states, for which we require 2 flip-flops. The truth table and transition list of the JK are

J	K	Q^{k+1}
0	0	Q^k
0	1	0
1	0	1
1	1	$\overline{Q^k}$

J	K	$Q^k \to Q^{k+1}$
0	X	$0 \to 0$
1	X	$0 \to 1$
X	0	$1 \to 1$
X	1	$1 \to 0$

and hence

x=0 O_1	O_0	J_1	K_1	J_0	K_0
0	0	0			
0	1	1			
1	0	X			
1	1	X			

x=1 O_1	O_0	J_1	K_1	J_0	K_0
0	0	1			
1	1	X			
1	0	X			
0	1	0			

To repeat ...

x=0 O_1	O_0	J_1	K_1	J_0	K_0
0	0	0			
0	1	1			
1	0	X			
1	1	X			

x=1 O_1	O_0	J_1	K_1	J_0	K_0
0	0	1			
0	1	X			
1	0	X			
1	1	0			

Thus J_1 is a function of O_1, O_0 and x, and its Karnaugh map is

O_1	x / O_0	0	1
0	0	0	1
0	1	1	0
1	1	X	X
1	0	X	X

whence $J_1 = O_1.\bar{x} + \overline{O_0}.x$ and similarly for K_1, J_0, and K_0.

State machines using ROMs

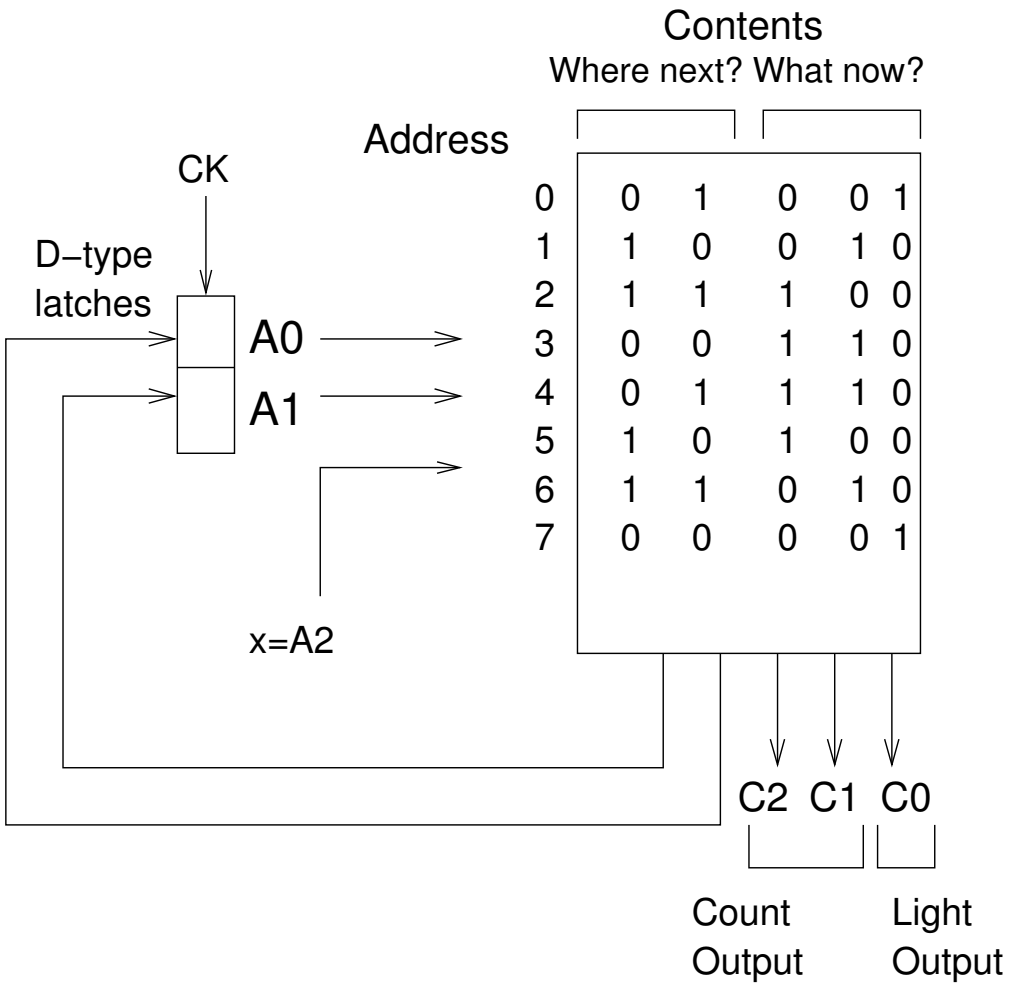
You also saw in Year 1 that an easy method of building a sequence state machine was to use a ROM, where:

- one part of the contents at a particular address tells you **what to do now**, and
- the other part tells you the address of **where to go next**

Problem. Design using a ROM a modulo-4 counter that if x=0 would count repeatedly from 0,1,2,3,... and if x=1 would count 3,2,1,0,... Whenever the output is 0, a light should light.

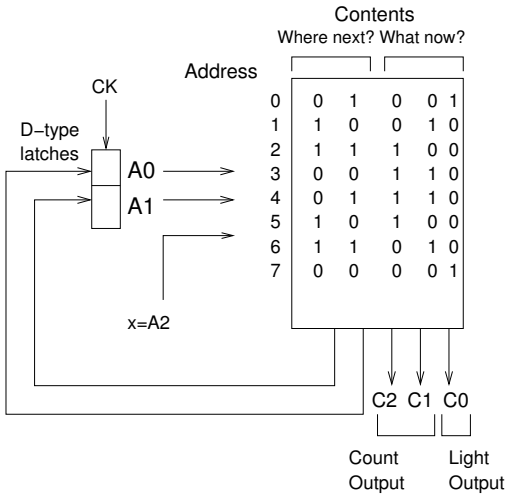
One possible solution

- let x become the high bit of the ROM address, as in



Should ROM solutions be “clever”?

- Tempting to ask whether there is not a more efficient solution than



- Eg, here the light output could be made by ANDing the inverses of C_2 and C_1. For another, by reorganizing the “where next” bits of addresses 4 to 7, we could make the content bits C_2 and C_1 the same as A_1 and A_0, and therefore redundant.
- Such questions miss the main points of using a ROM:
  - to provide a solution with minimal thought and
  - to provide a solution which is alterable merely by re-programming the ROM.

Algorithmic State Machine Charts

- These two methods of designing sequential circuits are specific hardware solutions,
  but they do not address a particular area of the design problem:

- the translation of the **written description**
  of what a machine should do into a
  formal description of the state machine.

- One way of doing this is to use an ASM chart, similar to flow diagrams used in software design.

Algorithmic State Machine Charts

- There are three symbols in such diagrams:
  - **State definition (rectangles):** There is one per state. It gives the name of the state, the binary flip-flop values that define the state, and a list of outputs.
  - **Decision (diamonds):** There are any number of these for each state. Each senses an input condition or flag, and takes a binary decision.
  - **Conditional outputs (curvy boxes):** These occur at the exit lines of decision diamonds, and describe outputs that become true only when a condition is satisfied. Note that these outputs cannot "change" the outputs given in the state box. If you need to do that, you need an extra state.

Vending machine example

Problem. A vending machine holds items which cost 20 cents, and accepts 5 cent (nickel), 10 cent (dime) and 25 cent (quarter) coins. Sensors in the coin shute cause two flags to be set as follows

Coin	I1	I2
No Coin	0	0
25	0	1
5	1	0
10	1	1

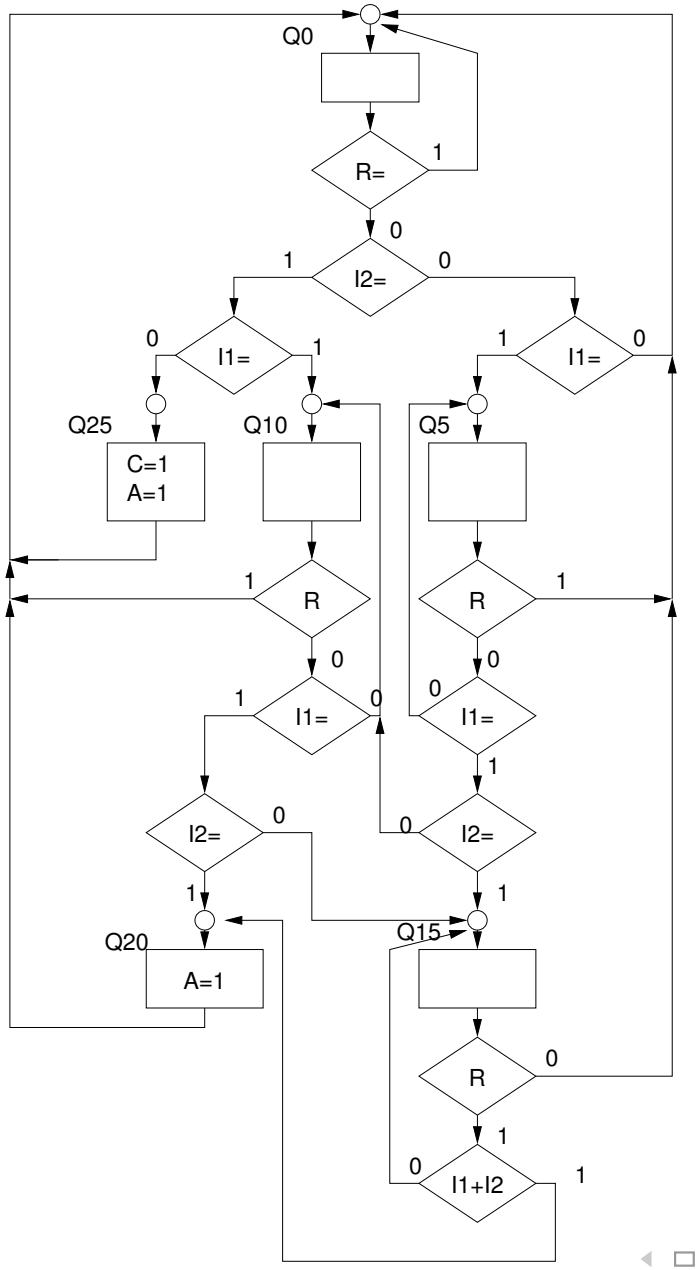
When coins to the value of 20 cents are deposited, the activate signal A is set to 1 for one clock cycle. If a 25 cent coin is deposited, a change signal C is set to one. Flag R is set when the user wishes to have all money returned.

Vending machine example

Example Solution. A possible solution is shown in Figure **??**. Any press of the Return button and every completed transaction puts the system in the Q0 state. From there one enters Q25 if a 25 cent coin is deposited, Q10 if a 10 cent coin and Q5 if a 5 cent. Notice that in this solution if in Q5, Q10 or Q15 when a 25 coin is deposited, the coin is ignored. However, also notice that in the solution given when the machine is in state Q15, *any* coin will be accepted and no change given.

There are 6 states, so at least 3 flip-flops are required. There is no method to determine how to assign the configuration of flip-flop outputs to the particular states, but the choice will alter the complexity of the resulting circuit. Thus, the choice of Q0=000 and Q5=100 etc is not arbitrary, but nonetheless there are no formal rules to obtain the optimal assignment.

ASM Chart


Q0
R=
1
0
I2=
1
0
I1=
0
1
I1=
1
0
Q25
C=1
A=1
Q10
Q5
R
1
R
1
0
0
I1=
1
0
I1=
0
1
I2=
0
I2=
0
1
1
Q20
A=1
Q15
R
0
1
I1+I2
0
1

Separating Data and Control

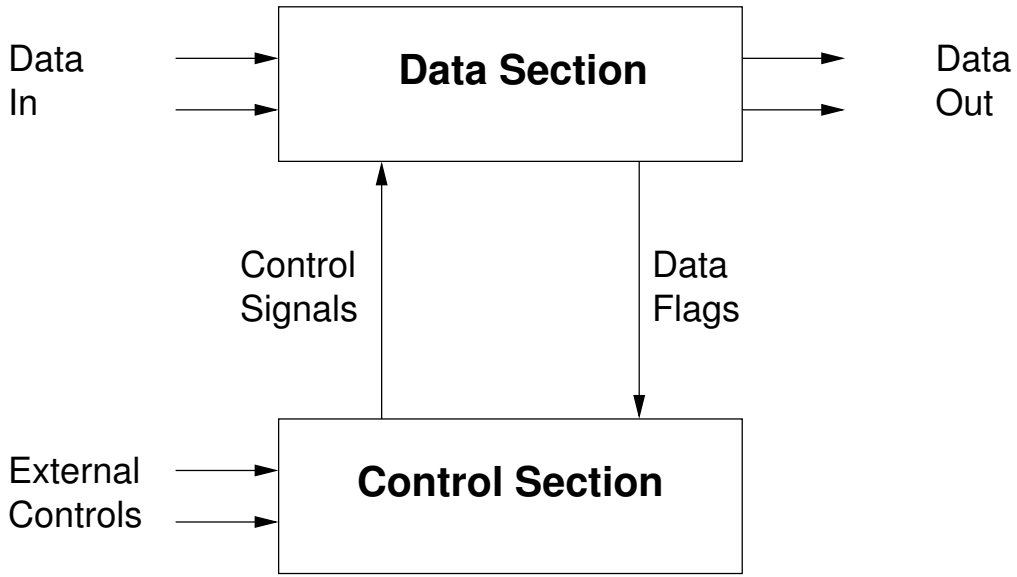
- More economical graphical methods can be found to represent state machines.
- BUT, whatever graphical method is used, there is an obvious difficulty using a method which represents every state.

  Given n flip-flops we require 2^n states
  which gets very large very quickly.

- Now, the large numbers of flip-flops are often storing *data* upon which the machine operates in largely the same way irrespective of what the data is exactly.
- Our understanding then of how the machine functions depends on those control operations, not on the particular data. It makes sense then to separate the representation of control from that of data.

Separating Data from Control

- The control section will need at times to know something about the data — for example, to take a different action if a datum is zero rather than positive — so there will be two way communication between the control and data sections
- BUT, only decision flags *about a datum*, not the *datum itself* need be communicated.



Register Transfer Language

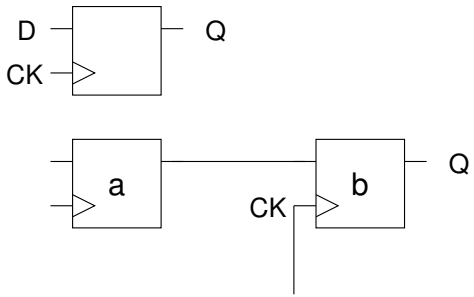
- It is of course possible to describe the control section of such a machine using ASM charts
- But the language does not have the syntax to deal with data elements.
- Instead we can use a hardware description language called

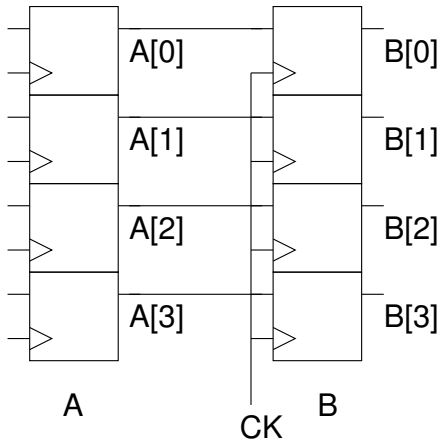
  Register Transfer Language.

- Why **Register Transfer?**
  The transfer of data between two storage registers is the principal activity that occurs in the data section of the machine, and as we will see in Lecture 2 is the bread and butter operation in computers.

Register Transfers

- A register is a D-type latch which transfers input to output on a receipt of a clock pulse: that is $Q^{t+1} = D^t$.
- In the example on the leftBy clocking register b, the transfer $b \leftarrow a$ is effected.
- Most often the register has several bits. In the right hand example $B \leftarrow A$ means that $B[i] \leftarrow A[i]$ for $i = 0, 1, 2, 3$.
- **Note carefully** that the **receiving register is clocked, NOT the transmitting register.**





Syntax of RTL by example

Module: Datamover
Memory: $A[2]$;$B[2]$;$C[2]$;S
Inputs: $X[2]$.
Ouputs: $Z[2]$; P.

1 $A \leftarrow X$
2 $C \leftarrow \overline{A}$; $S \leftarrow A[0]$
3 $B \leftarrow C$
4 $C \leftarrow A \vee B$
5 $Z = C$; $S \leftarrow 0$; $\rightarrow 1$

ENDSEQUENCE
ControlReset(1); $P = S$.
END

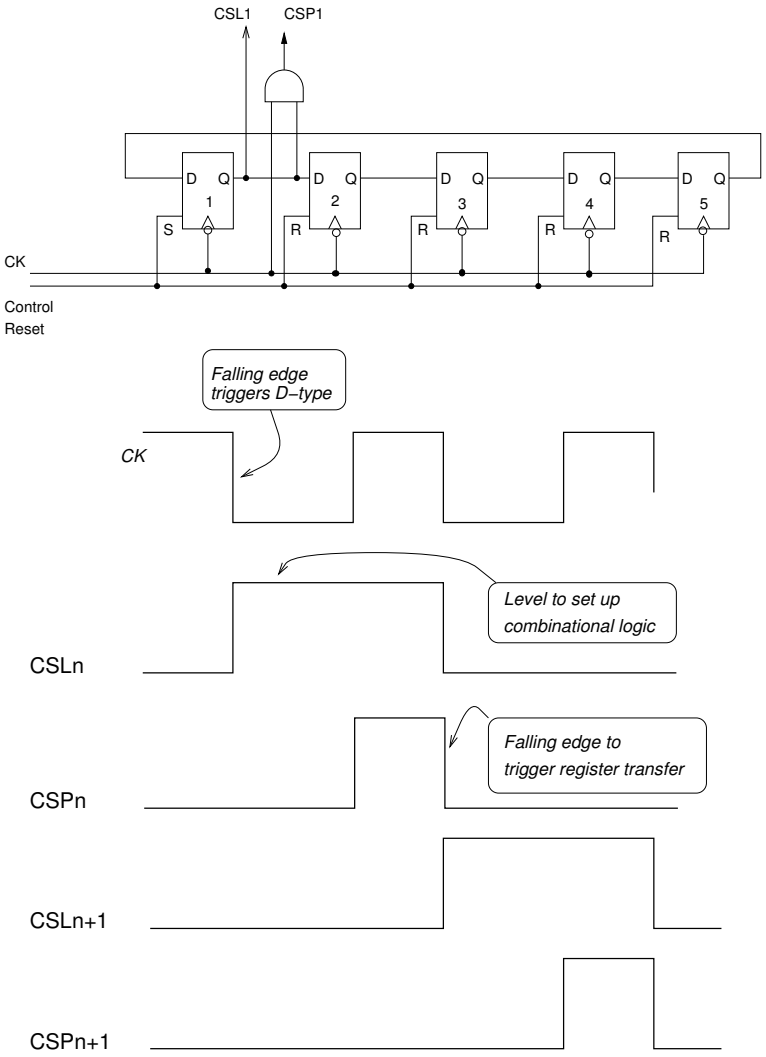
Control section: generating CSL and CSP signals

- To get the data section to work may require on each line
  - CSL — a level signal, typically to set up data pathways between registers
  - CSP — an edge or pulse to fire the register transfers.

- These are provided by the control section.

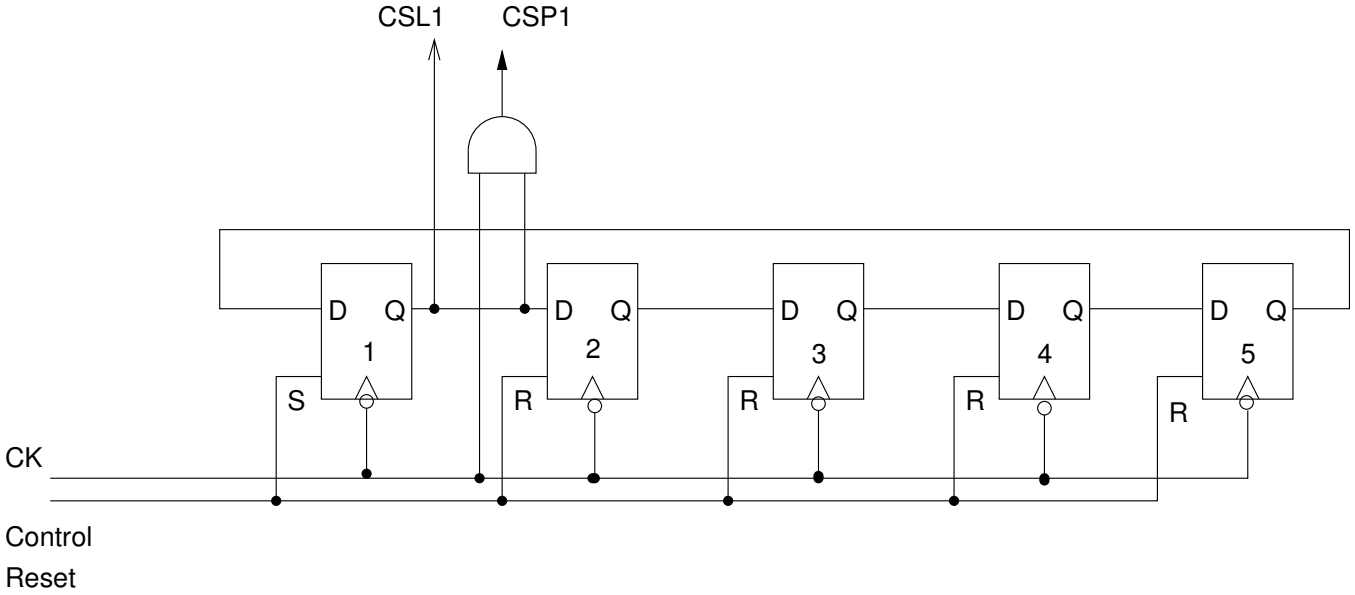
- To realize the control section, you
  - count the number of lines of RTL (here 5) In our example flows uninterrupted from line 1 to 5, where there is an “unconditional goto” $\rightarrow(1)$.
    (We shall see a conditional goto later on.)
  - study the flow from one line to another.

Hardware realization of Control Section



- Transition from one line to the next occurs on the falling edge of a clock pulse.
- The clock is ANDed with the CSL to provide a pulse CSP which is used to fire the register transfers and the transition to the next line of RTL.
- So CSP1 only occurs at the end of line 1 of RTL.

Other observations ...

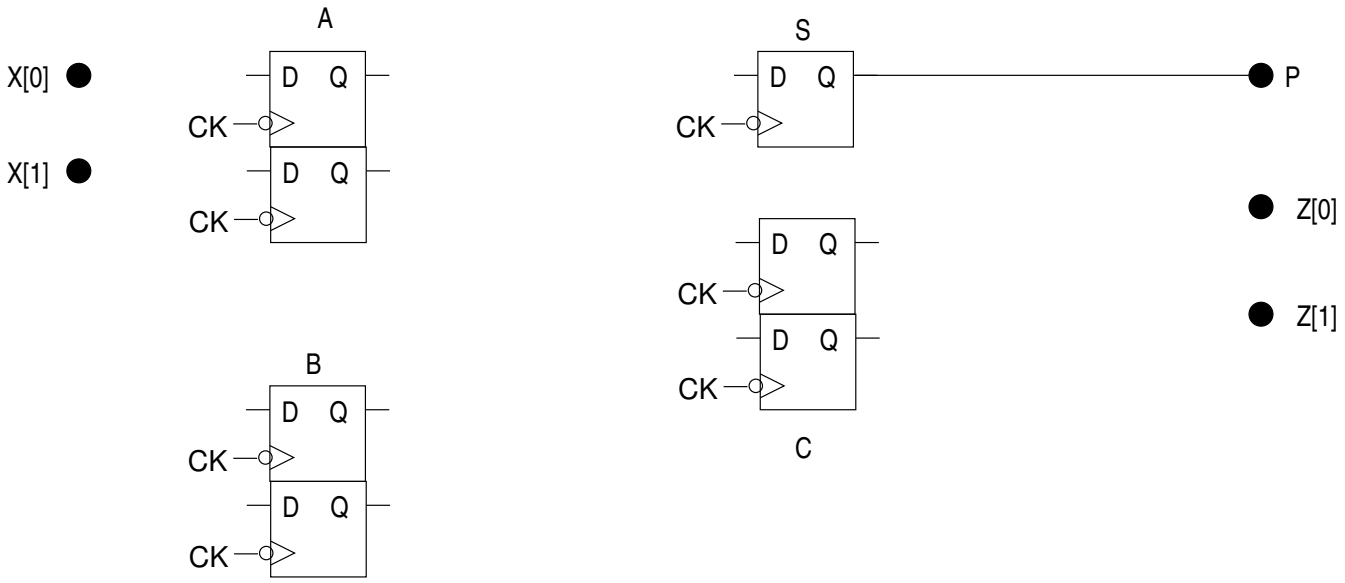


- The latch corresponding to the active line has output 1 — the remainder have output 0.
- The $\rightarrow(1)$ is achieved by looping the output of latch 5 into the input of latch 1.
- Notice too that the reset line sets flip-flop 1 high, and the rest low.
- The level output from a latch supplies the CSL signal, so CSL1 is high during line 1, CSL2 is high during line 2, and so on.

Data section: using the CSL and CSP signals

- The register hardware required for the data section is defined in the first blocks of the module.

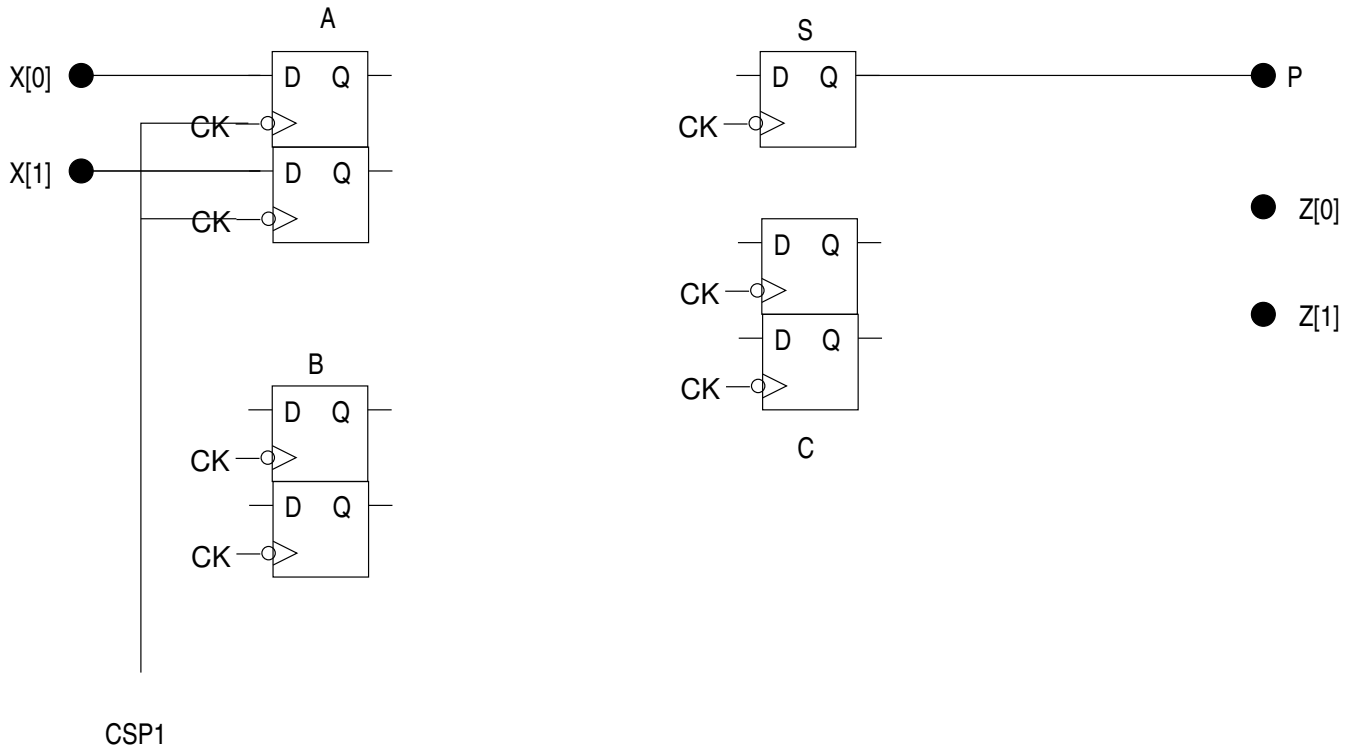
  MEMORY: A[2]*; B*[2]*; C*[2]*; S*
  INPUTS: X[2]*.*
  OUTPUTS: Z[2]*; P.*



Data section/ continued

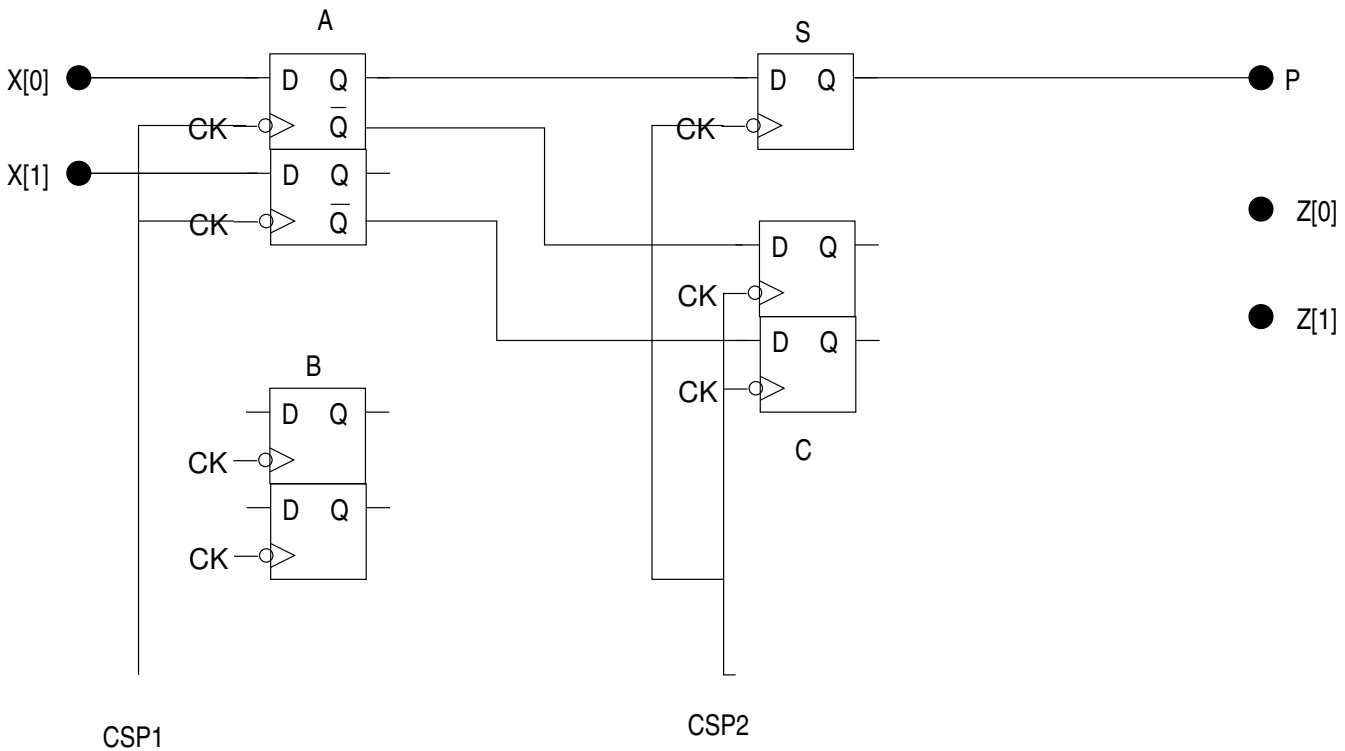
- Line 1 of sequence indicates that inputs *X* are connected to the input of register *A*, which is clocked by CSP1.

1 $A \leftarrow X$



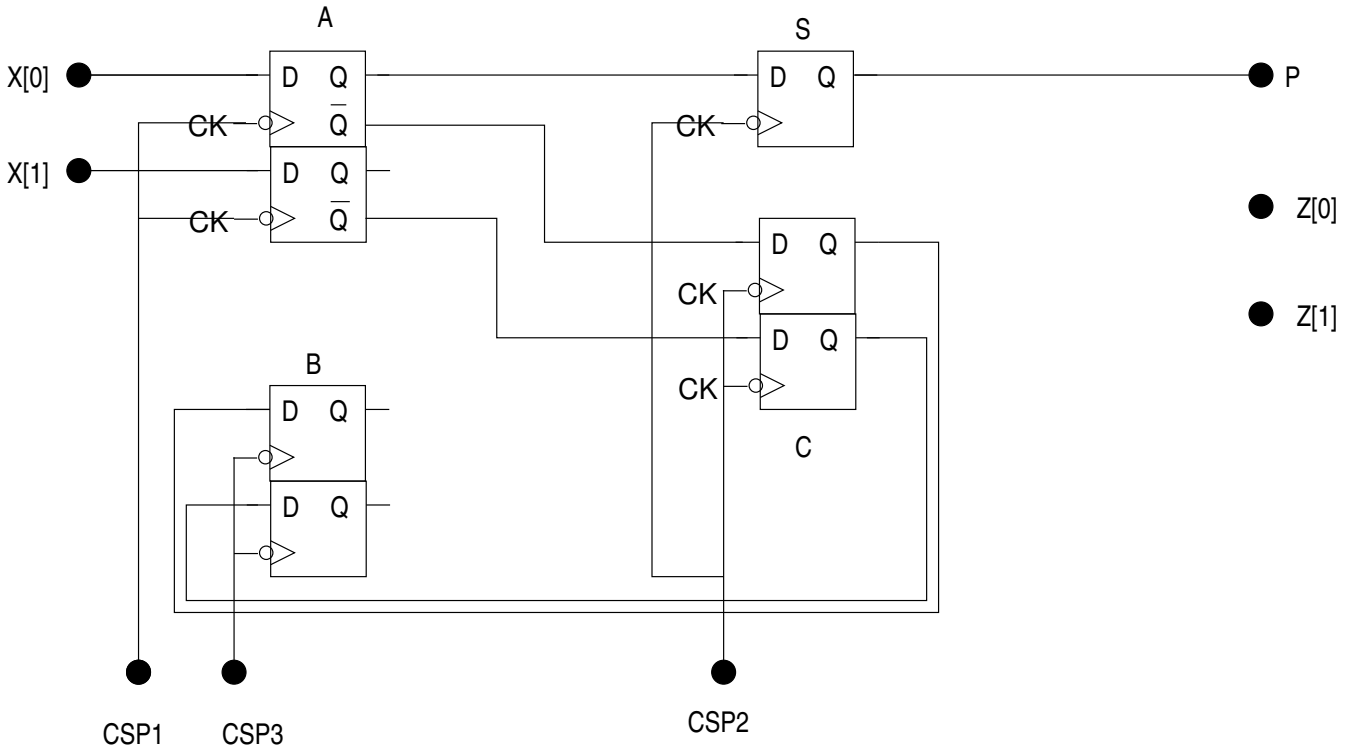
Line 2

- $2 \quad C \leftarrow \overline{A}; \quad S \leftarrow A[0]$
- Line 2 indicates that the inverted outputs of A are to be connected to C and that the output $A[0]$ should be connected to the input of S. CSP2 should be attached to the CK input of C and S.



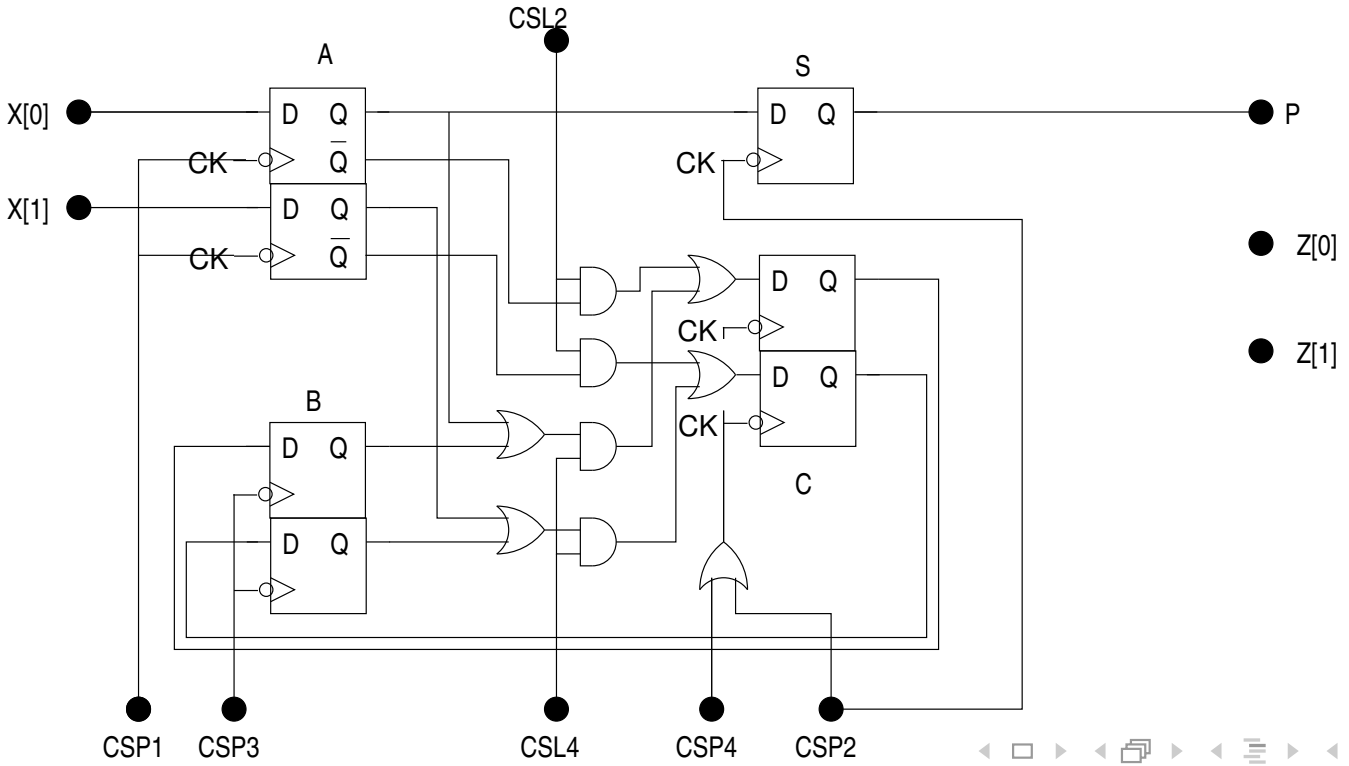
Line 3

- *3 B←C*



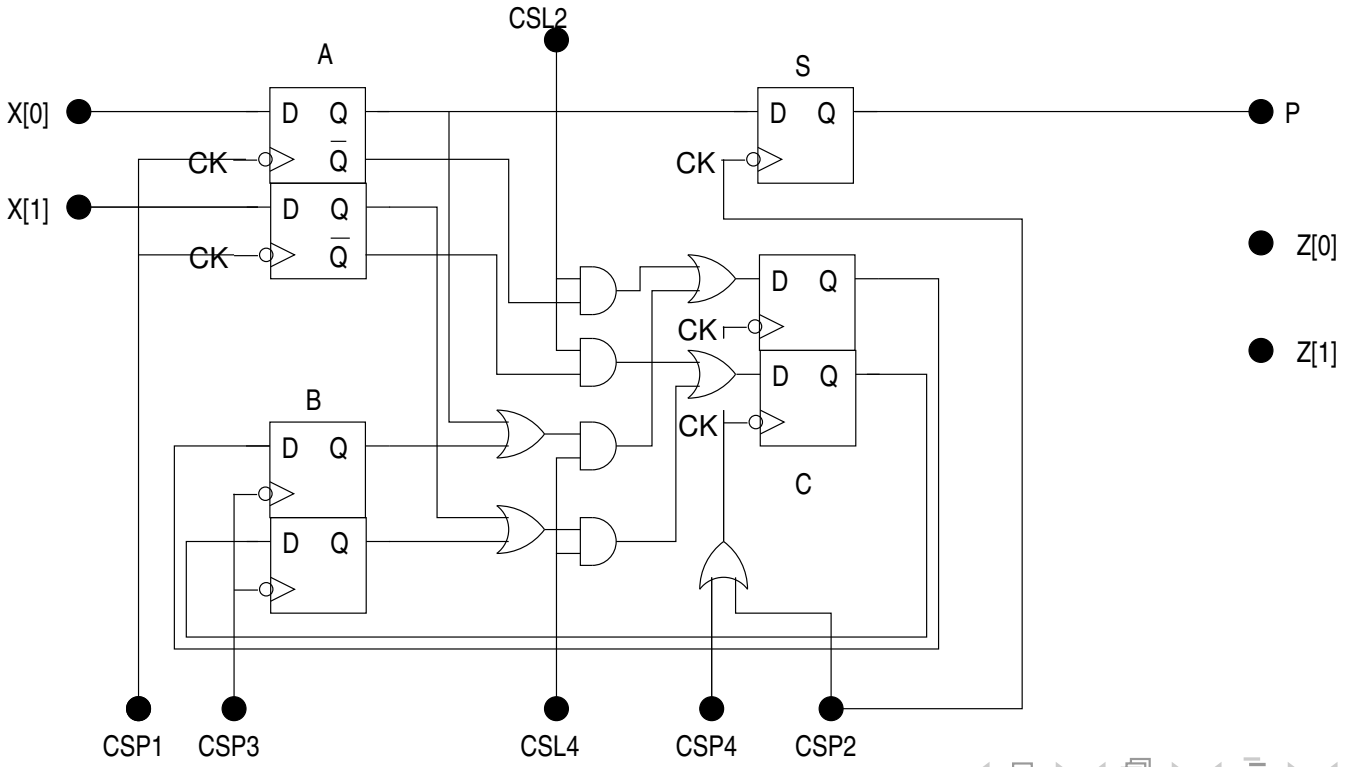
Line 4

- $4 \quad C \leftarrow A \vee B$
- For Line 4, our design need modifying. Notice that C is clocked on line 4 as well as line 2, so C's clock input must be CSP2$\vee$CPS4. But on line 2, the inputs to C are connected to $\overline{A}$ and on line 4 they are connected to $A \vee B$. We need to insert AND gates, ANDing with CSL2 and CSL4 respectively, and then OR the inputs into C.



Line 5

- $5 \quad Z = C; S{\leftarrow}0; {\rightarrow}1$
- At line 5, $Z = C$ means that outputs Z should be connected to the output of C for the entire line — so we must AND the outputs of C with CSP5. In addition, we have to make alterations to the input of S and to its clock input.
(**Incomplete! See Question sheet.**)



Using data to control flow

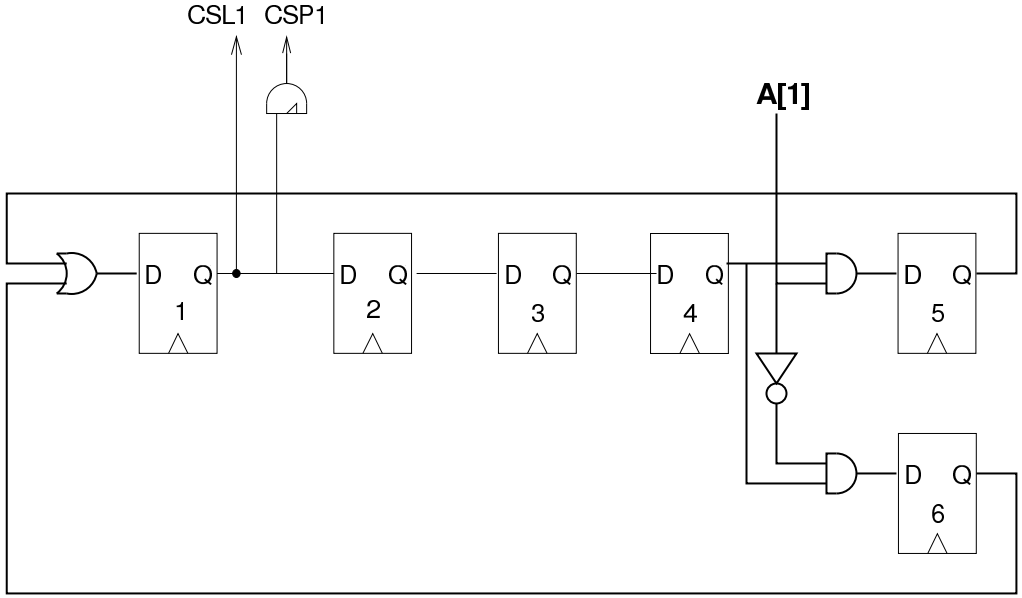
MODULE: DATAMOVER
MEMORY: $A[2]$*;* $B[2]$*;* $C[2]$*;* S
INPUTS: $X[2]$*.*
OUTPUTS: $Z[2]$*;* P*.*

1 $A \leftarrow X$
2 $C \leftarrow \overline{A}$*;* $S \leftarrow A[0]$
3 $B \leftarrow C$
4 $C \leftarrow A \vee B$*;* $\rightarrow (A[1], \overline{A[1]})/(5, 6)$
5 $Z = C$*;* $S \leftarrow 1$*;* $\rightarrow 1$
6 $Z = \overline{C}$*;* $S \leftarrow 0$*;* $\rightarrow 1$

ENDSEQUENCE
ControlReset(1); $P = S$*.*
END

... using a de-multiplexer!

- *A*[1] provides the selection line in a multiplexer.
- During Line 4 CSL4 is high, and if *A*[1] is high the input to latch 5 is high, but if *A*[1] is low the input to latch 6 is high. After the next clock pulse, CSL5 or CSL6 respectively will go high.
  Note that because there are multiple ways of arriving back at Line 1, the input to latch 1 requires an OR gate.



- There are obviously some alterations required to the Data Section for the new line 6. You are asked to do these in the Problem Sheets.

Timing

Let us consider the various register outputs as a function of time for particular inputs X. Note that X can change asynchronously.

1	$A \leftarrow X$
2	$C \leftarrow \overline{A}$; $S \leftarrow A[0]$
3	$B \leftarrow C$
4	$C \leftarrow A \vee B$; $\rightarrow (A[1], \overline{A[1]})/(5, 6)$
5	$Z = C$; $S \leftarrow 1$; $\rightarrow 1$
6	$Z = \overline{C}$; $S \leftarrow 0$; $\rightarrow 1$