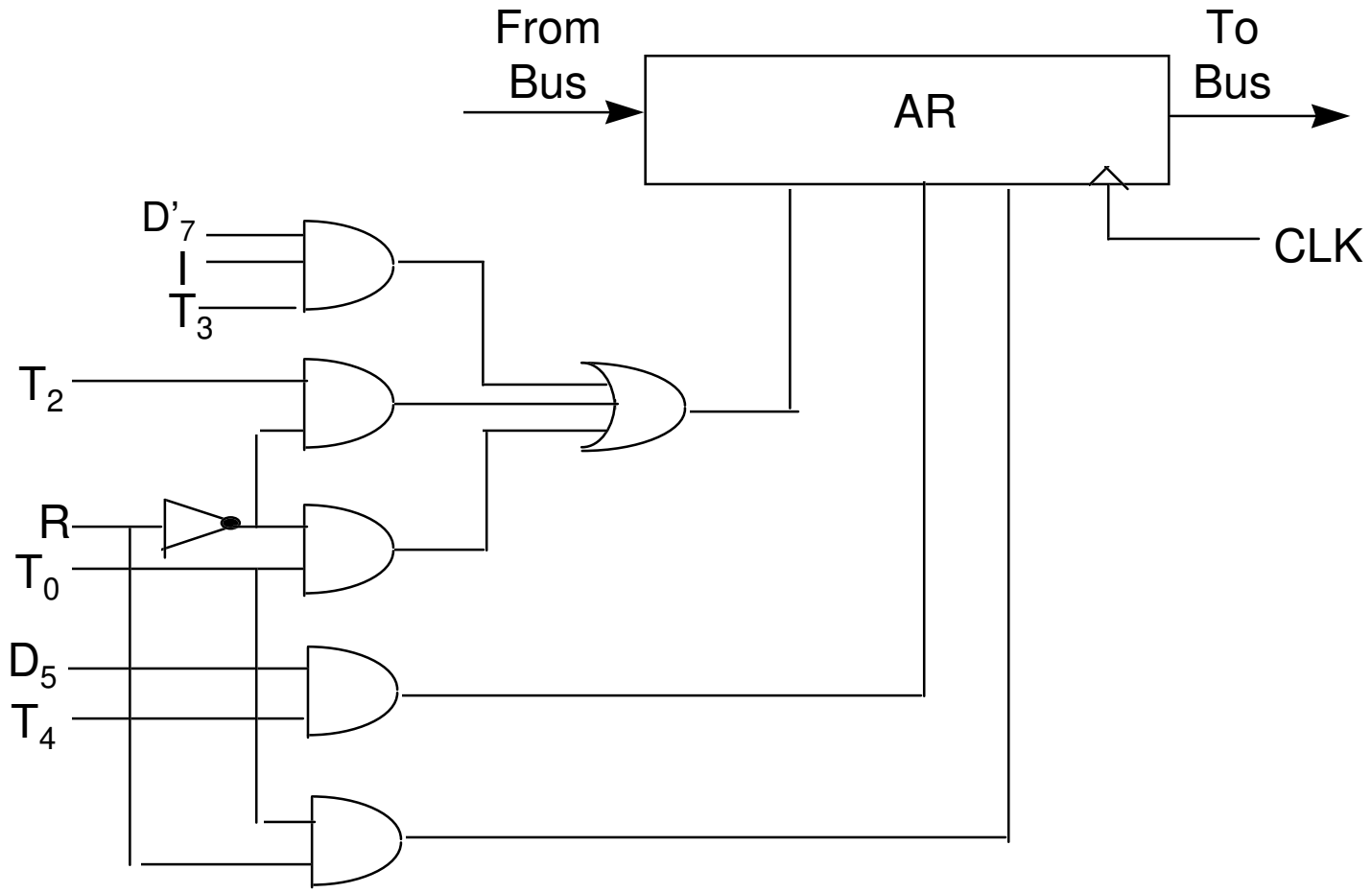
Announcements

- Homework #6 is due in class today
- Midterm #2 is Thursday November 9

Control Logic for AR

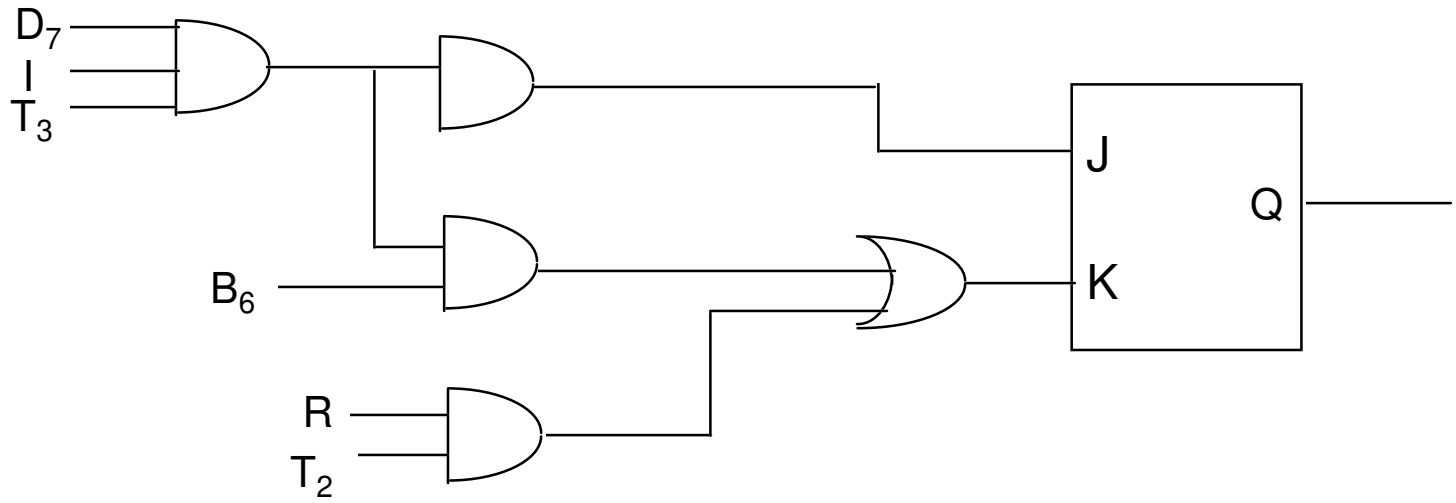
- RTL Statements that change AR:
  - $R' T_0$: AR <- PC
  - $R' T_2$: AR <- IR(0-11)
  - $D'_7 I T_3$: AR <- M[AR]
  - $R T_0$: AR <- 0
  - $D_5 T_4$: AR <- AR + 1
- Now compute control lines:
  - $LD_{AR} = R' T_0 + R' T_2 + D'_7 I T_3$
  - $CLR_{AR} = RT_0$
  - $INR_{AR} = D_5 T_4$

Circuit to Control AR



Control For A Single F.F.

- Consider IEN:
  - $D_7IT_3B_7$: IEN <- 1
  - $D_7IT_3B_6$: IEN <- 0
  - R T_2: IEN <- 0
- Circuit looks like:



Consider the Design of the ALU

- Collect RTL Statements:

  D_0T_5: AC <- AC ^ DR

  D_1T_5: AC <- AC + DR

  D_2T_5: AC <- DR

  pB_{11}: AC(0-7) <- INPR

  rB_9: AC <- AC'

  rB_7: AC <- shr AC, AC(15) <- E

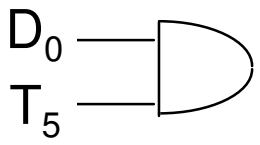
  rB_6: AC <- shl AC, AC(0) <- E

  rB_{11}: AC <- 0

  rB_5: AC <- AC + 1

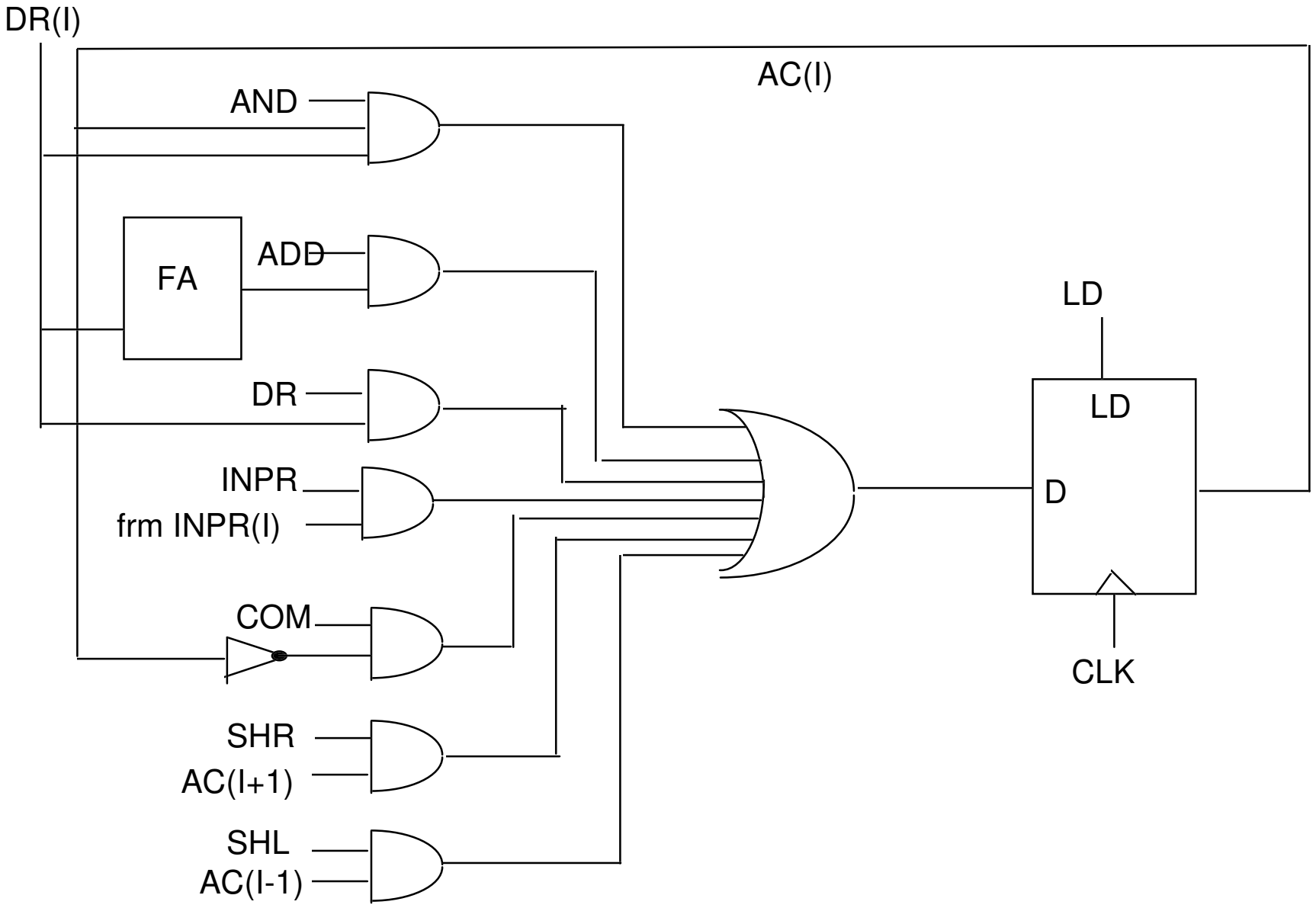
ALU (cont)

- Consider a single bit position
  - Use D F.F. with load for each bit
  - build one large OR gate for functionality
  - use AND gates for each RTL statement
    - D_0T_5 : AND



    - OR together to form load line
  - Use OR gate to combine different RTL statements

Implementing the ALU (cont)



Programming the Stored Program Computer

- Writing programs for the Mano computer is tedious
  - figure out what instruction is required
  - figure out the memory location
  - write the instruction in hex
- When we add a new statement to a program
  - changes the address of every instruction after it
    - have to re-compute addresses of all of the rest of the statements after it
  - would like a more symbolic way to describe a program

Assembly Lanaguage

- Provides a (slightly) higher level of abstraction
  - Programs are still written one instruction at a time.
  - provides a more symbolic way to write a program.
- Assembly Language Statement has:
  - label - symbolic names
    - for data and instructions
  - instruction field
    - opcode (e.g. ADD)
    - operand
      - can be a label
      - can be a number
    - indirect flag (I)
  - comment fields
    - a “/”