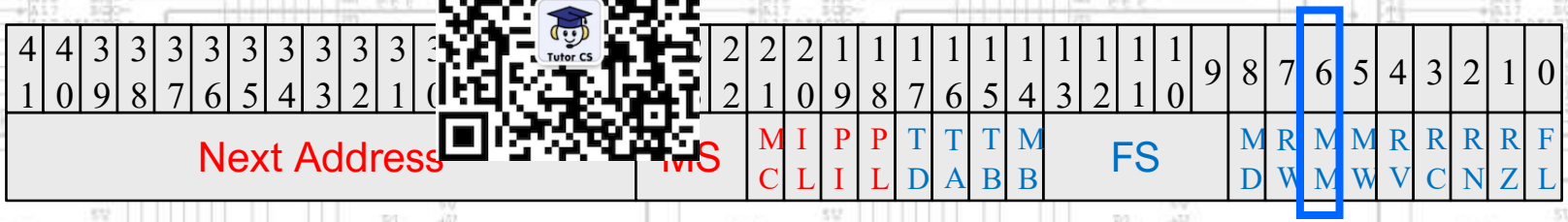
# Multiple-Cycle Design



- The Multiple-Cycle implementation demonstrates the use of a single memory for:
  - Data
  - Instruction
- This design is also used to show the implementation of more complex instructions

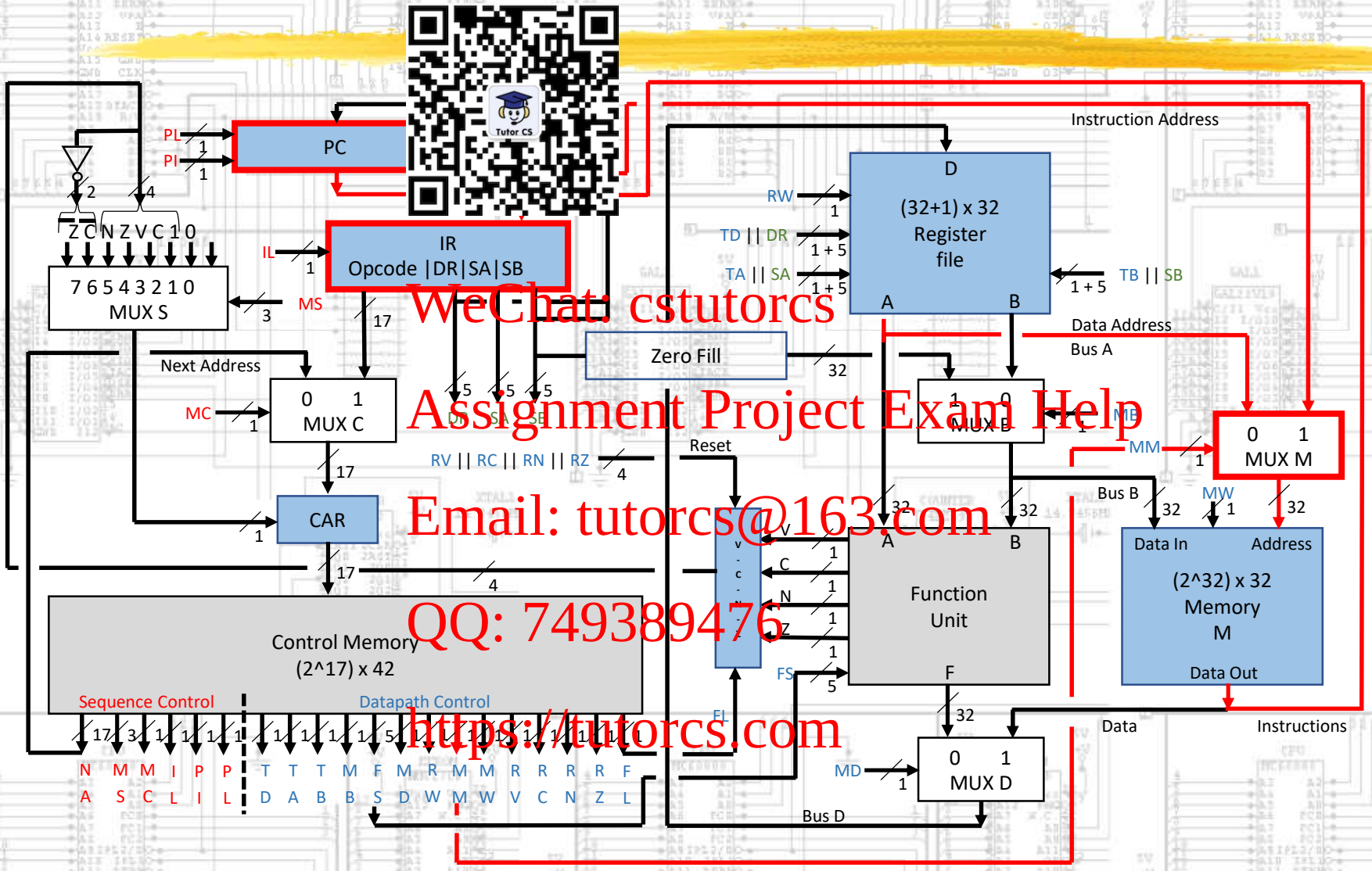
# Memory M Address





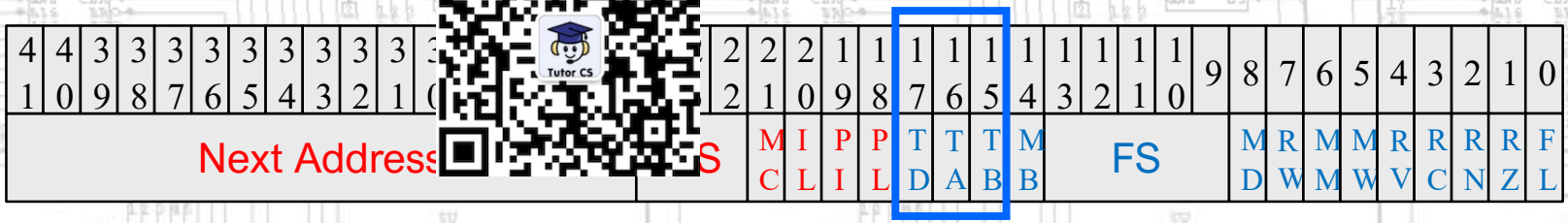

- The following address sources are used to fetch:
  - Instructions -> PC Program Counter Register (32bit)
  - Data -> BusA (32bit)
- MUX M selects between the two address sources through the MM control signal

# PC - Bus A - MM



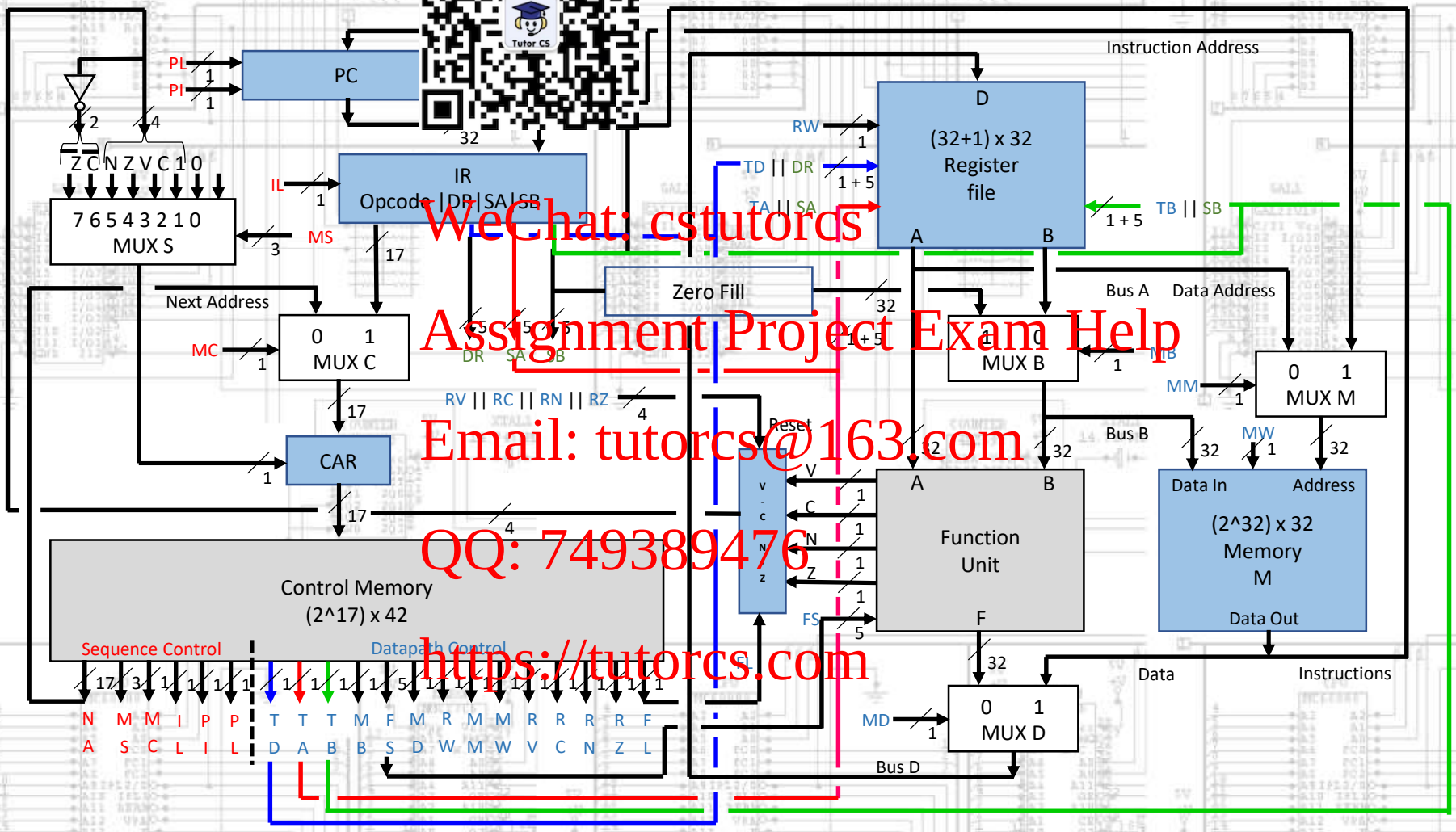
# Temp Register





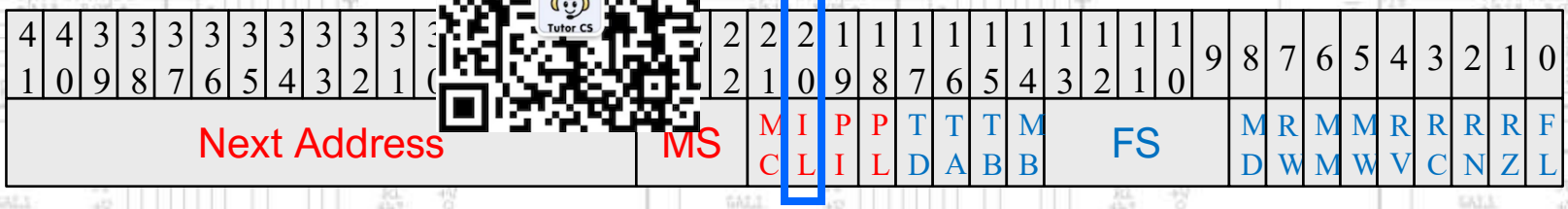

- Instructions are executed over multiple clock cycles
- This requires an additional register
  - R32 for temporary storage
- This register should be selected through an additional bit control signals:
  - TD, TA, TB
- The overwrite:
  - SA, SB, DR

# TD||DR - TA||SA - TB||SB



# IR Instruction Register



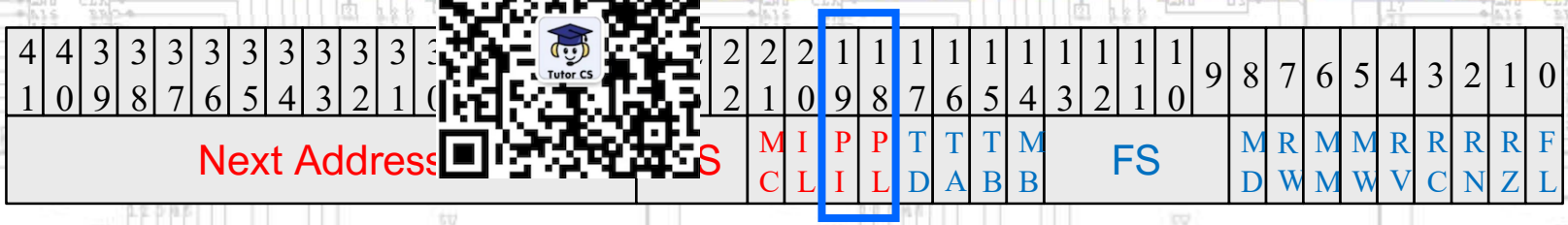





- Instructions must be held in a register during the execution of multiple micro-ops
- The IR is only loaded if an instruction is fetched from memory M
  - The IR has a load enable control signal IL
  - This signal is part of the control word

# PC Program Counter Register



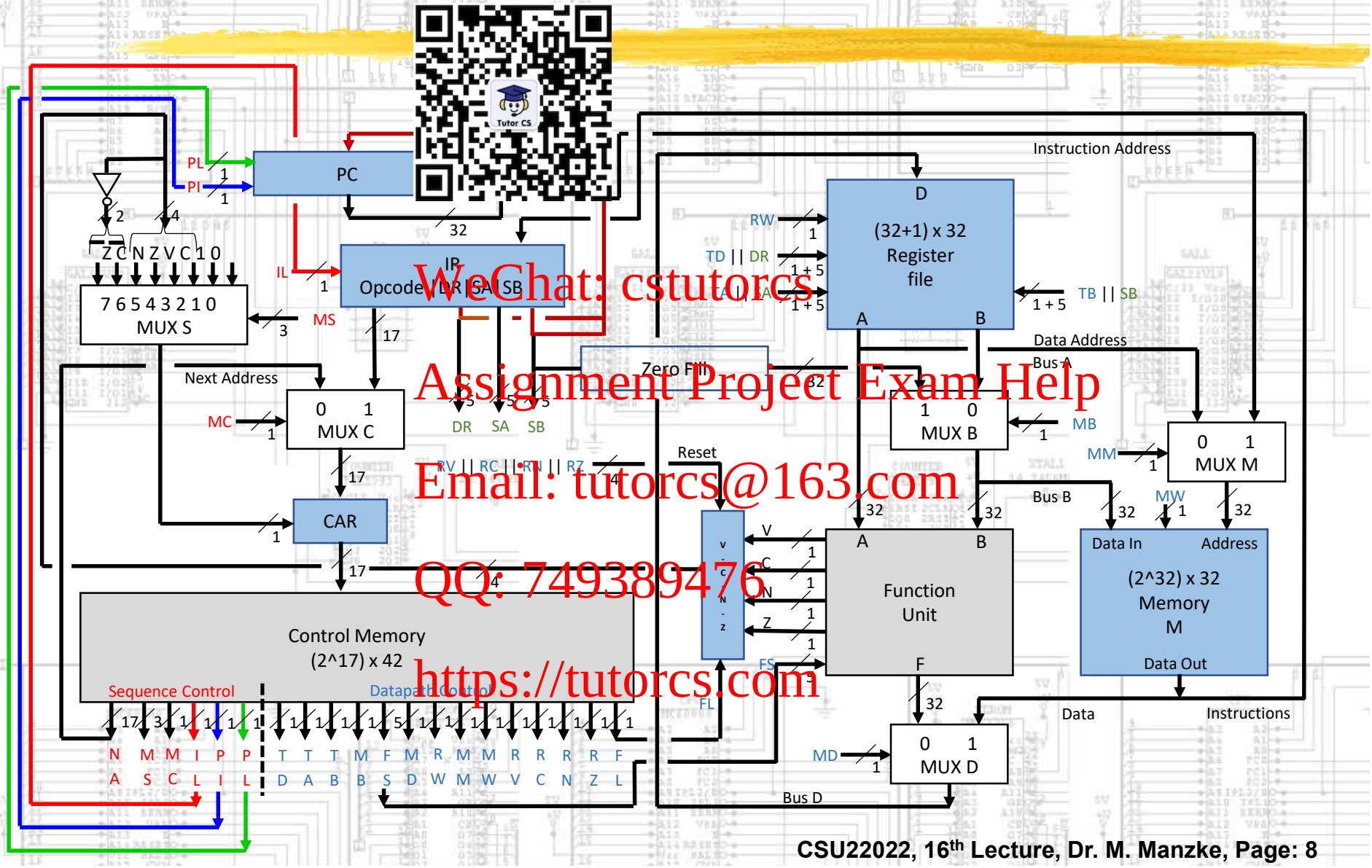





- The PC only increments if an instruction is fetched from memory M
- The control word has two bits that determine the PC modifications:
  - PI - increment enable signal
    - PC ← PC + 1
  - PL – PC load signal
    - PC ← PC + se AD

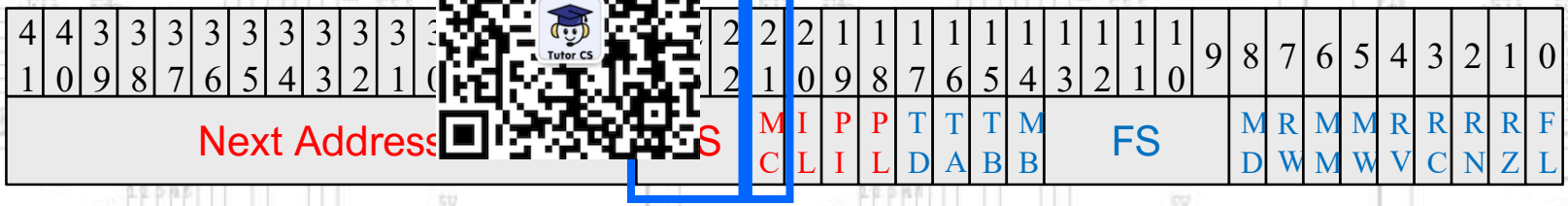
# IR - PC - PL





# Next Address Logic








- The CAR Control Address Register selects the control word in the 256 x 42 control memory



- The next logic (MUX S) determines whether CAR is incremented on loaded.



  - Controlled with MS



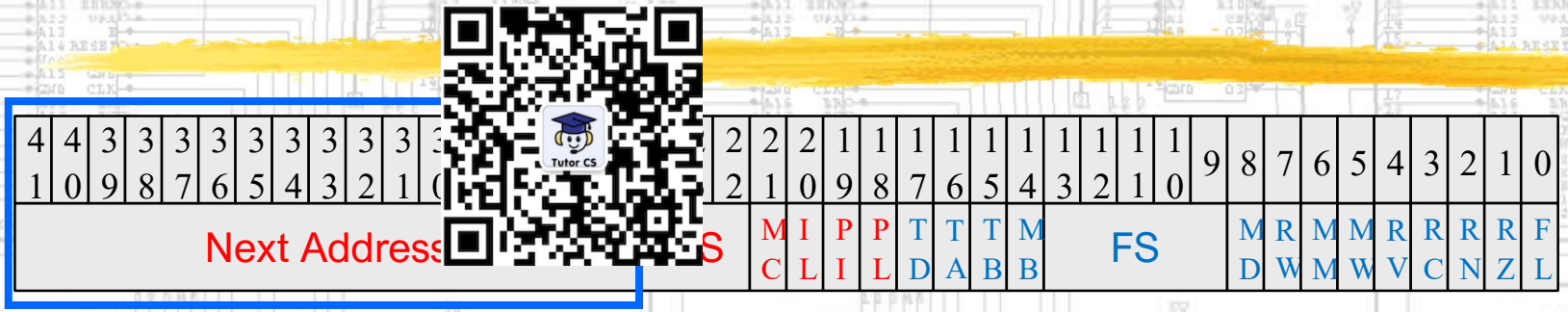
- The source (Opcode or NA) of the loaded address is determined by MUX C



  - Selected by MC

# Next Address Field





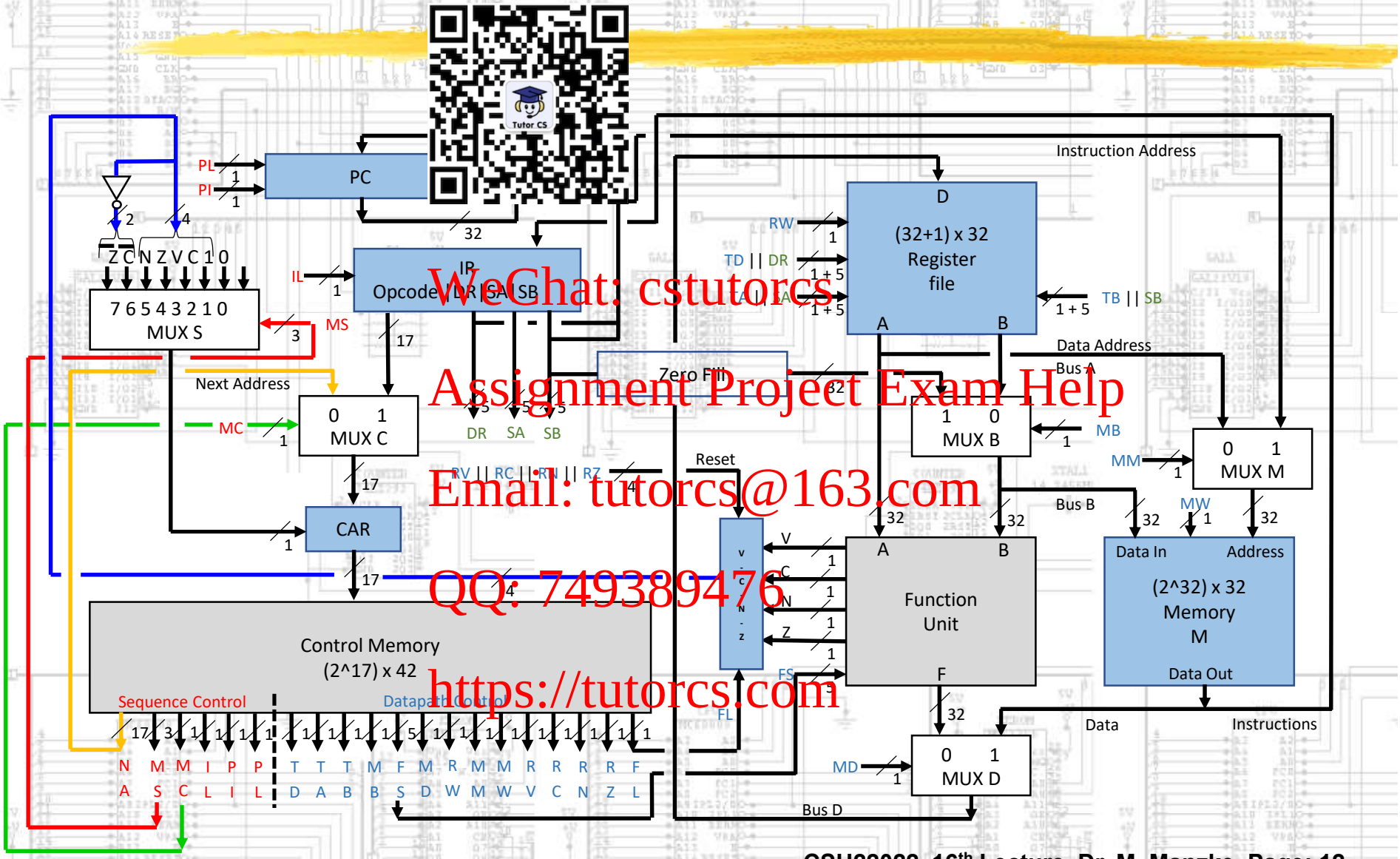



- The sources for the multiplexer can be:
  - Contents of the 17 bit NA Next Address field
  - 17 bit from the opcode field in the IR
- An opcode loaded into the CAR points to:
  - Microprogram in Control Memory
  - This program implements the instruction through the execution of a sequence of micro-operations
- MUX S determines whether the CAR is:
  - Incremented
  - Loaded

# Sequencer Control Fields

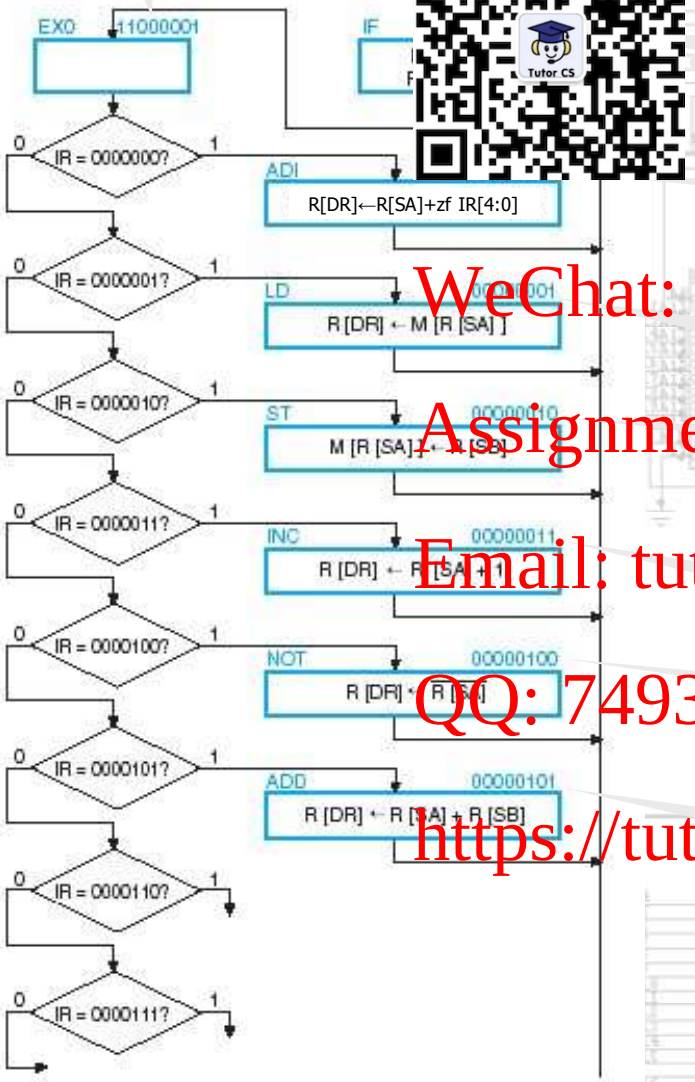


| MS | | | IL | | | PI | | PL | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Action | Symbolic Notation | Code | Action | | | Symbolic Notation | Action | Symbolic Notation | Action | Symbolic Notation | Code |
| Increment *CAR* | CNT | 000 | NA | NXA | No load | NLI | No load | NLP | No load | NLP | 0 |
| Load *CAR* | NXT | 001 | Opcode | OPC | Load instr. | LDI | Increment PC | INP | Load PC | LDP | 1 |
| If $C = 1$, load *CAR*; else increment *CAR* | BC | 010 | | | | | | | | | |
| If $V = 1$, load *CAR*; else increment CAR | BV | 011 | | | | | | | | | |
| If $Z = 1$, load *CAR*; else increment *CAR* | BZ | 100 | | | | | | | | | |
| If $N = 1$, load *CAR*; else increment *CAR* | BN | 101 | | | | | | | | | |
| If $C = 0$, load *CAR*; else increment *CAR* | BNC | 110 | | | | | | | | | |
| If $Z = 0$, load *CAR*, else increment *CAR* | BNZ | 111 | | | | | | | | | |



# Microprogram ASM






$C1_{16}$

$C0_{16}$

$00_{16}$

$01_{16}$

$02_{16}$

$03_{16}$

$04_{16}$

$05_{16}$



Implementation on page14

# Microprogram in Control Memory



? = $0_2$ or ? = $1_2$

```
-- |41          25|24|23|22|21|20|19|18|17|16|15|14|13  9|8|7|6|5|4|3|2|1|0|
-- | Next Address  | MS| | | | | | | T| T| T| M|  FS |M|R|M|M|R|R|R|R|F|
-- | Next Address  | MS| | | | | | | D| A| B| B|  FS |D|W|M|W|V|C|N|Z|L|

-- ADI                R[DR]←R[SA]+IR[4:0]
  "00000000????????? ???  ?  ?  ?  ?  ?  ?  ?  ? ????? ? ? ? ? ? ? ? ?",-- 00
-- LD                 R[DR]←M[R[SA]]
  "00000000????????? ???  ?  ?  ?  ?  ?  ?  ?  ? ????? ? ? ? ? ? ? ? ?",-- 01
-- ST                 M[R[SA]]←R[SB]
  "00000000????????? ???  ?  ?  ?  ?  ?  ?  ?  ? ????? ? ? ? ? ? ? ? ?",-- 02
-- INC                R[DR]←R[SA]+1
  "00000000????????? ???  ?  ?  ?  ?  ?  ?  ?  ? ????? ? ? ? ? ? ? ? ?",-- 03
-- NOT                R[DR]←NOT[R[SA]]
  "00000000????????? ???  ?  ?  ?  ?  ?  ?  ?  ? ????? ? ? ? ? ? ? ? ?",-- 04
-- ADD                R[DR]←R[SA]+R[SB]
  "00000000????????? ???  ?  ?  ?  ?  ?  ?  ?  ? ????? ? ? ? ? ? ? ? ?",-- 05

                                    .
                                    .
                                    .
-- IF                 IR←M[PC], PC←PC+1
  "00000000????????? ???  ?  ?  ?  ?  ?  ?  ?  ? ????? ? ? ? ? ? ? ? ?",-- C0
-- EX0                CAR←IR[31:15]
  "00000000????????? ???  ?  ?  ?  ?  ?  ?  ?  ? ????? ? ? ? ? ? ? ? ?",-- C1
                                    :
                                    :
                                   );

        variable addr : integer;
        variable control_out : std_logic_vector(41 downto 0);
```