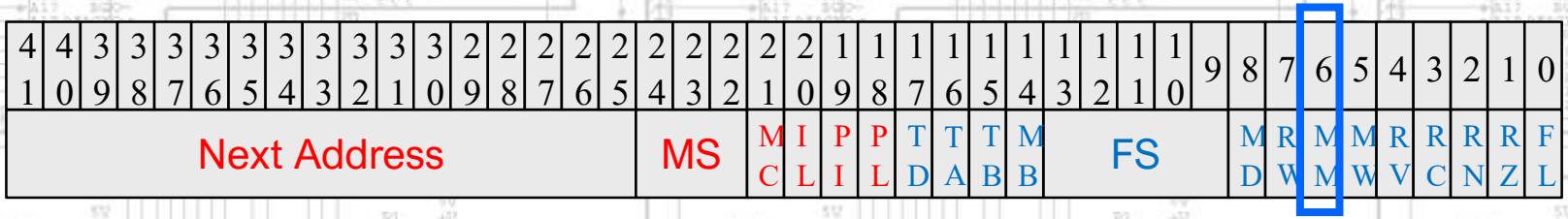
Multiple-Cycle Design

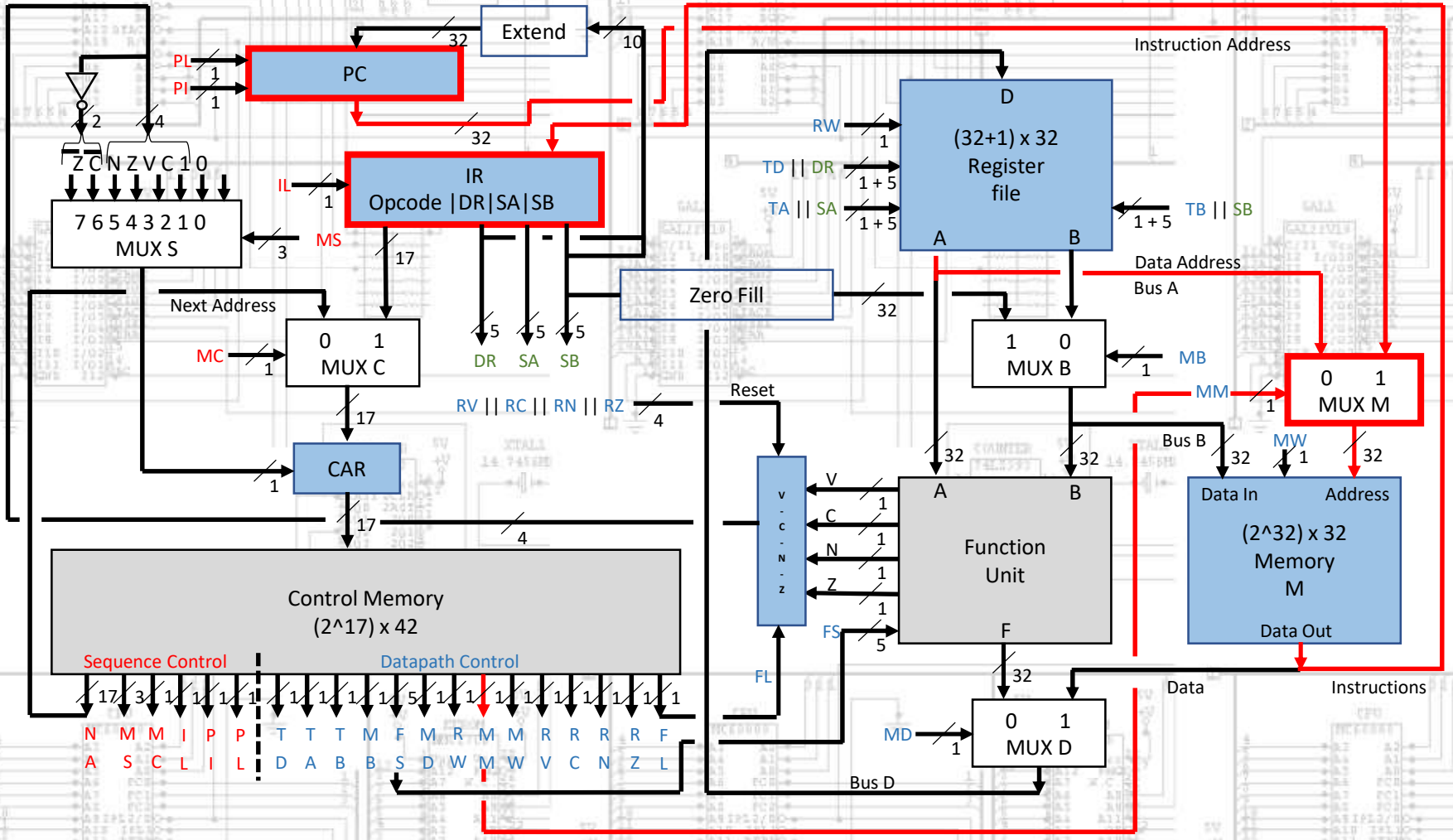
- The Multiple-Cycle Implementation demonstrates the use of a single memory for:
  - Data
  - Instruction
- This design is also used to show the implementation of more complex instructions

Memory M Address

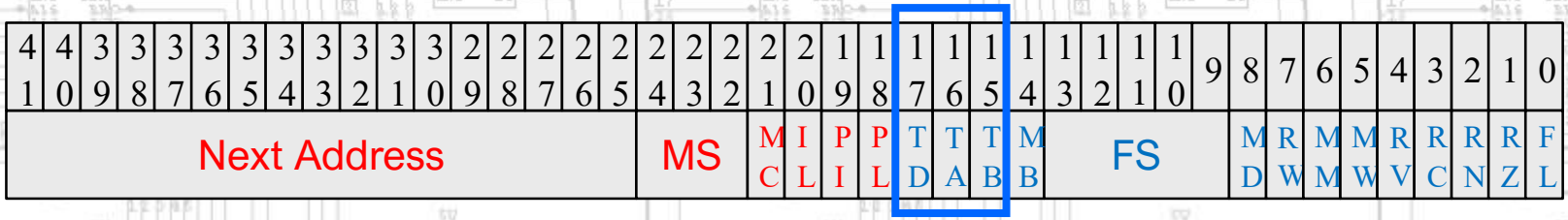


- The following address sources are used to fetch:
  - Instructions -> PC Program Counter Register (32bit)
  - Data -> Bus A (32bit)
- MUX M selects between the two address sources through the MM control signal

PC - Bus A - MM

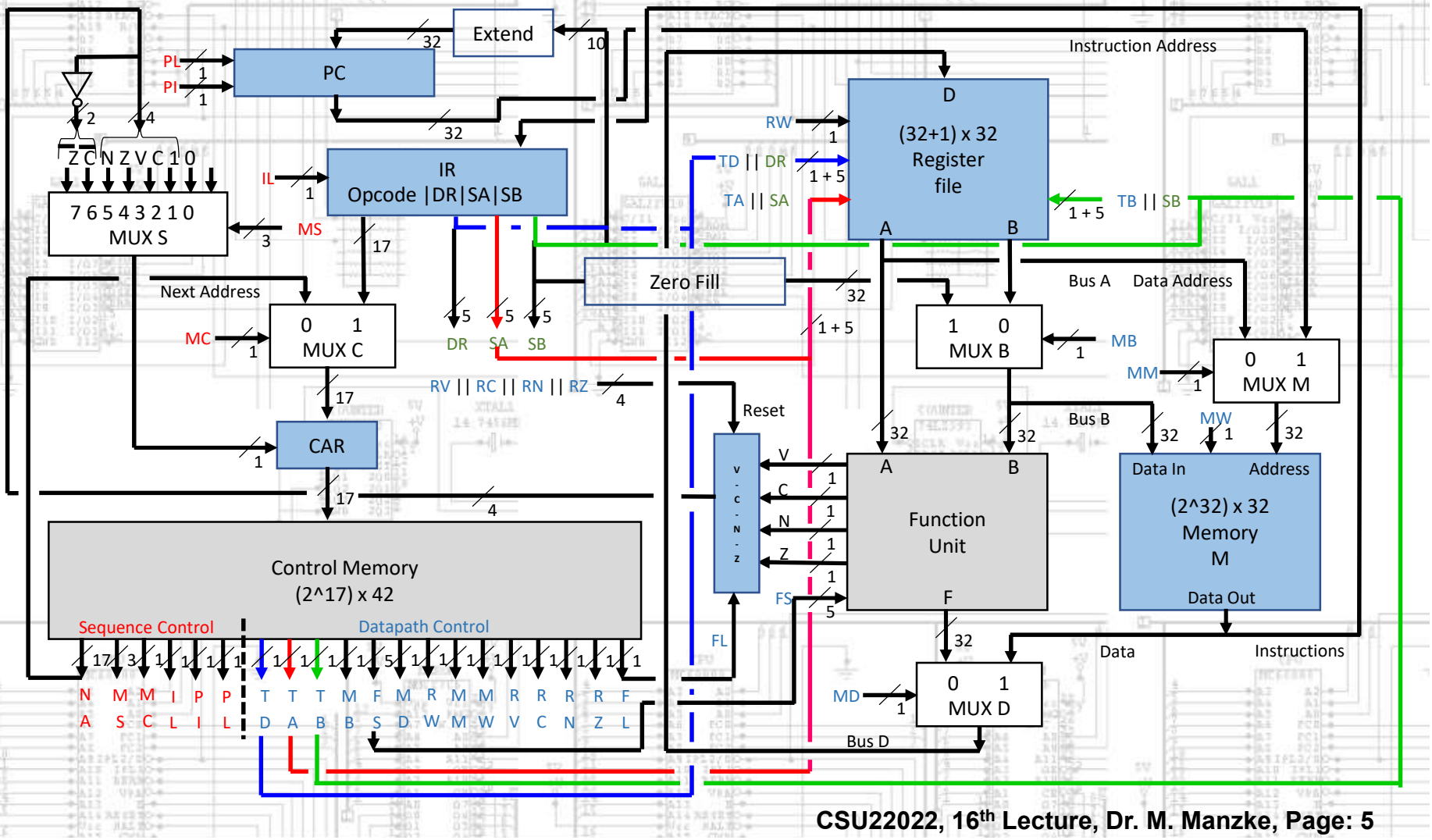

Extend
32
10
PL
PI
1
1
PC
32
Instruction Address
D
(32+1) x 32
Register
file
A
B
RW
1
TD || DR
1 + 5
TA || SA
1 + 5
TB || SB
1 + 5
Z C N Z V C 1 0
2
4
IL
1
IR
Opcode |DR|SA|SB
7 6 5 4 3 2 1 0
MUX S
3
MS
17
Zero Fill
32
Data Address
Bus A
Next Address
MC
1
0 1
MUX C
5
5
5
DR
SA
SB
1 0
MUX B
MB
1
MM
1
0 1
MUX M
RV || RC || RN || RZ
4
Reset
17
CAR
1
Bus B
32
32
32
MW
1
32
V
C
N
Z
1
1
1
1
A
B
Function
Unit
F
Data In
Address
(2^32) x 32
Memory
M
Data Out
17
4
Control Memory
(2^17) x 42
Sequence Control
Datapath Control
FS
5
FL
32
Data
Instructions
MD
1
0 1
MUX D
Bus D
N A
M S
M C
I L
P I
P L
T D
T A
T B
M B
F S
M D
R W
M M
M W
R V
R C
R N
R Z
F L

Temp Register

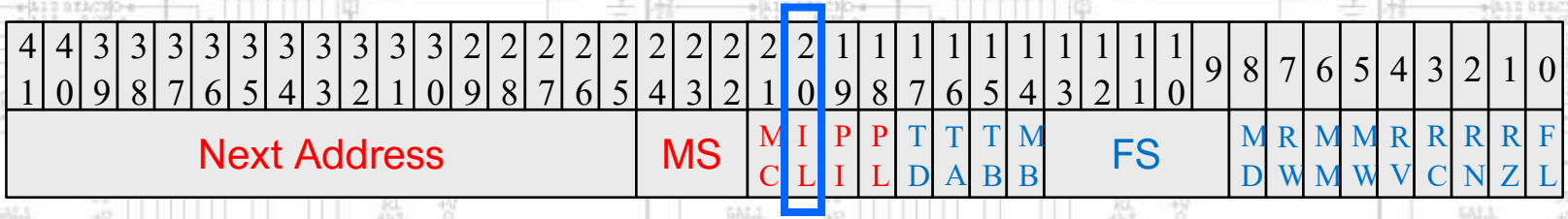


- Instructions are executed over multiple clock cycles
- This requires an additional register
  - R32 for temporary storage
- This register should be selected through an additional bit control signals:
  - TD, TA, TB
- The overwrite:
  - SA, SB, DR

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

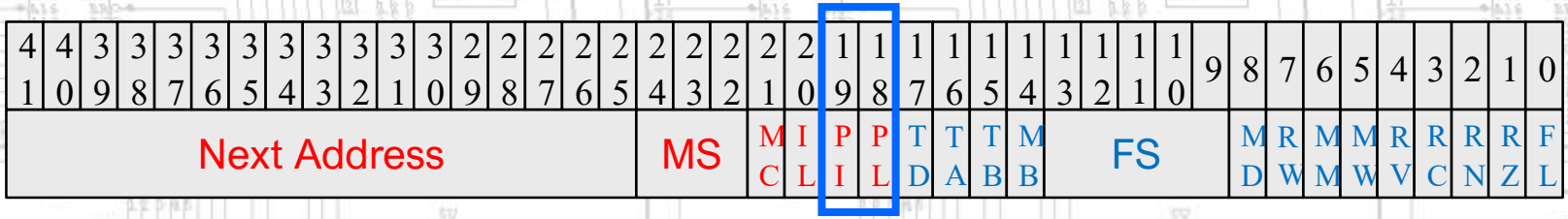


IR Instruction Register



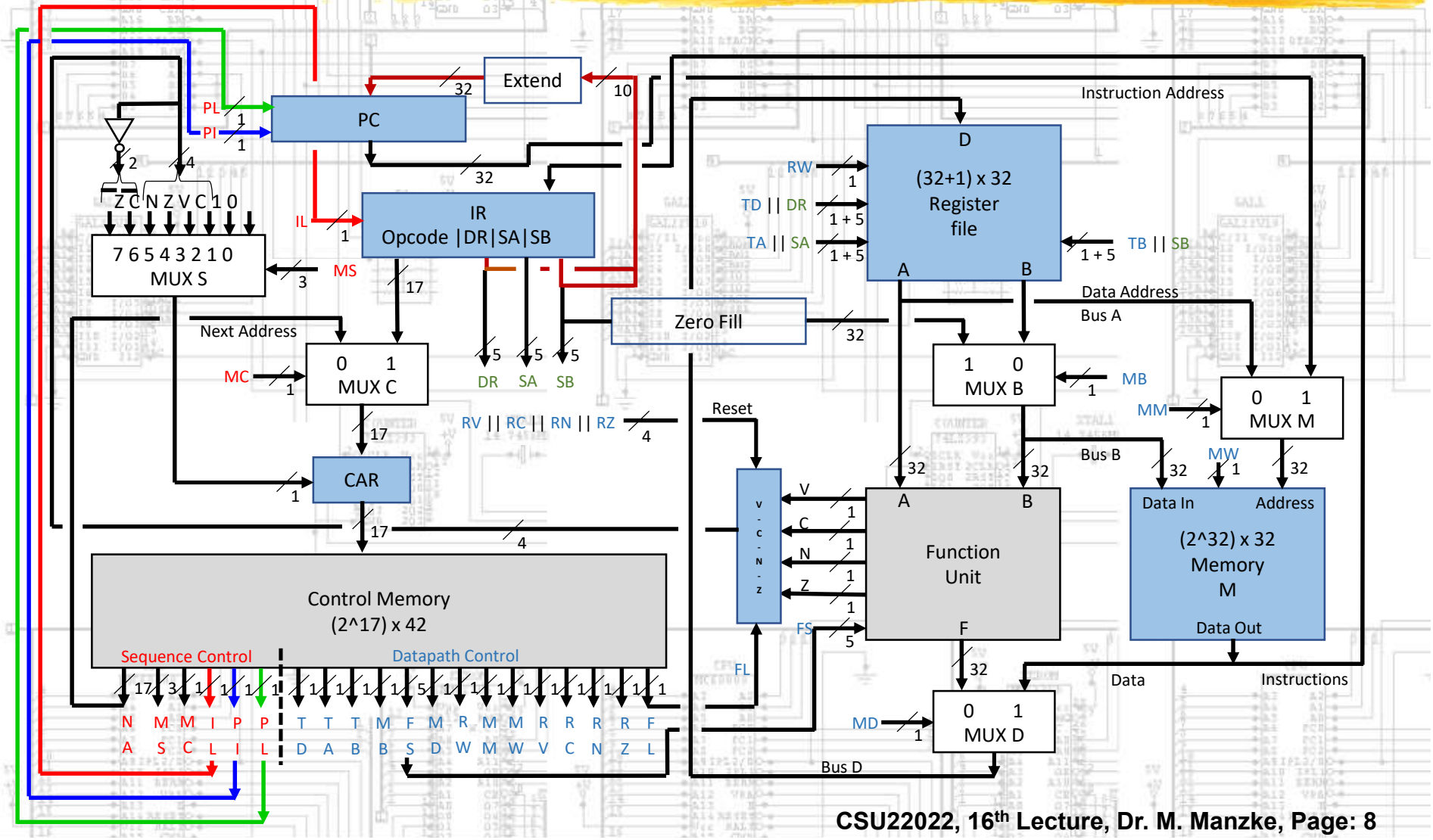
- Instructions must be held in a register during the execution of multiple micro-ops
- The IR is only loaded if an instruction is fetched from memory M
  - The IR has a load enable control signal IL
  - This signal is part of the control word

PC Program Counter Register

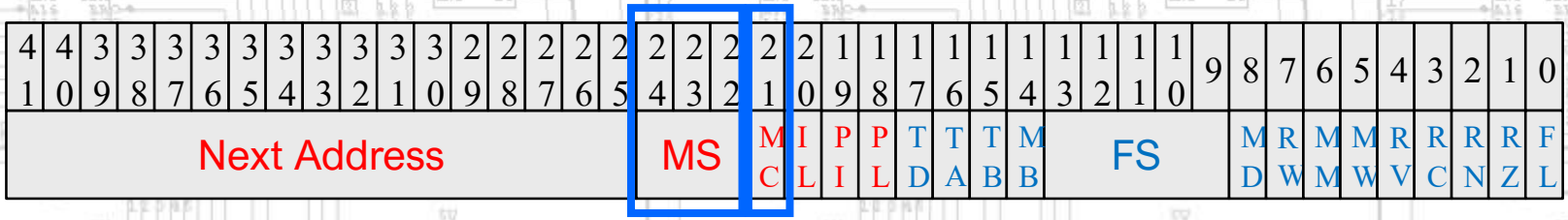


- The PC only increments if an instruction is fetched from memory M
- The control word has two bits that determine the PC modifications:
  - PI - increment enable signal
    - PC ← PC + 1
  - PL – PC load signal
    - PC ← PC + se AD

IR - IL; PC - PI - PL

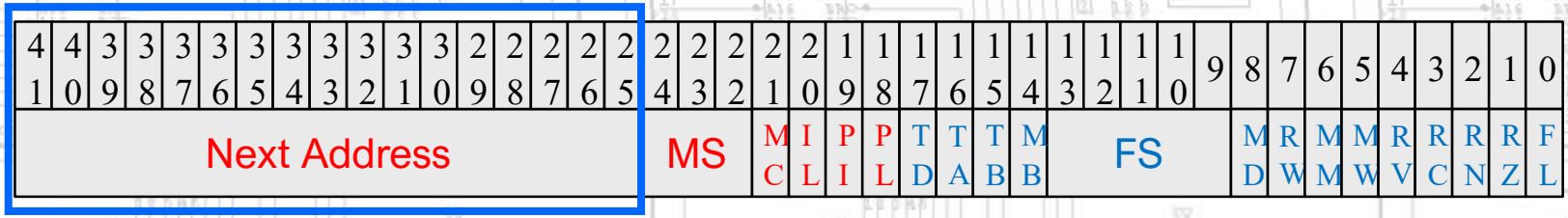

Extend
32
10
PL
PI
1
1
PC
32
2
4
Z C N Z V C 1 0
7 6 5 4 3 2 1 0
MUX S
IL
1
IR
Opcode |DR|SA|SB
3
MS
17
Next Address
MC
1
0 1
MUX C
17
CAR
1
17
4
5 5 5
DR SA SB
RV || RC || RN || RZ
4
Zero Fill
32
Instruction Address
D
(32+1) x 32
Register
file
A B
RW
1
TD || DR
1 + 5
TA || SA
1 + 5
TB || SB
1 + 5
Data Address
Bus A
1 0
MUX B
MB
1
0 1
MUX M
MM
1
MW
1
Bus B
32
32
32
32
Reset
V
C
N
Z
1
1
1
1
FS
5
FL
A B
Function
Unit
F
32
Data In
Address
(2^32) x 32
Memory
M
Data Out
Data
Instructions
MD
1
0 1
MUX D
Bus D
Control Memory
(2^17) x 42
Sequence Control
Datapath Control
17 3 1 1 1 1 1 1 1 1 1 1 1 5 1 1 1 1 1 1 1 1 1 1
N A
M S
M C
I L
P I
P L
T D
T A
T B
M B
F S
M D
R W
M W
M M
R V
R C
R N
R Z
F L

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

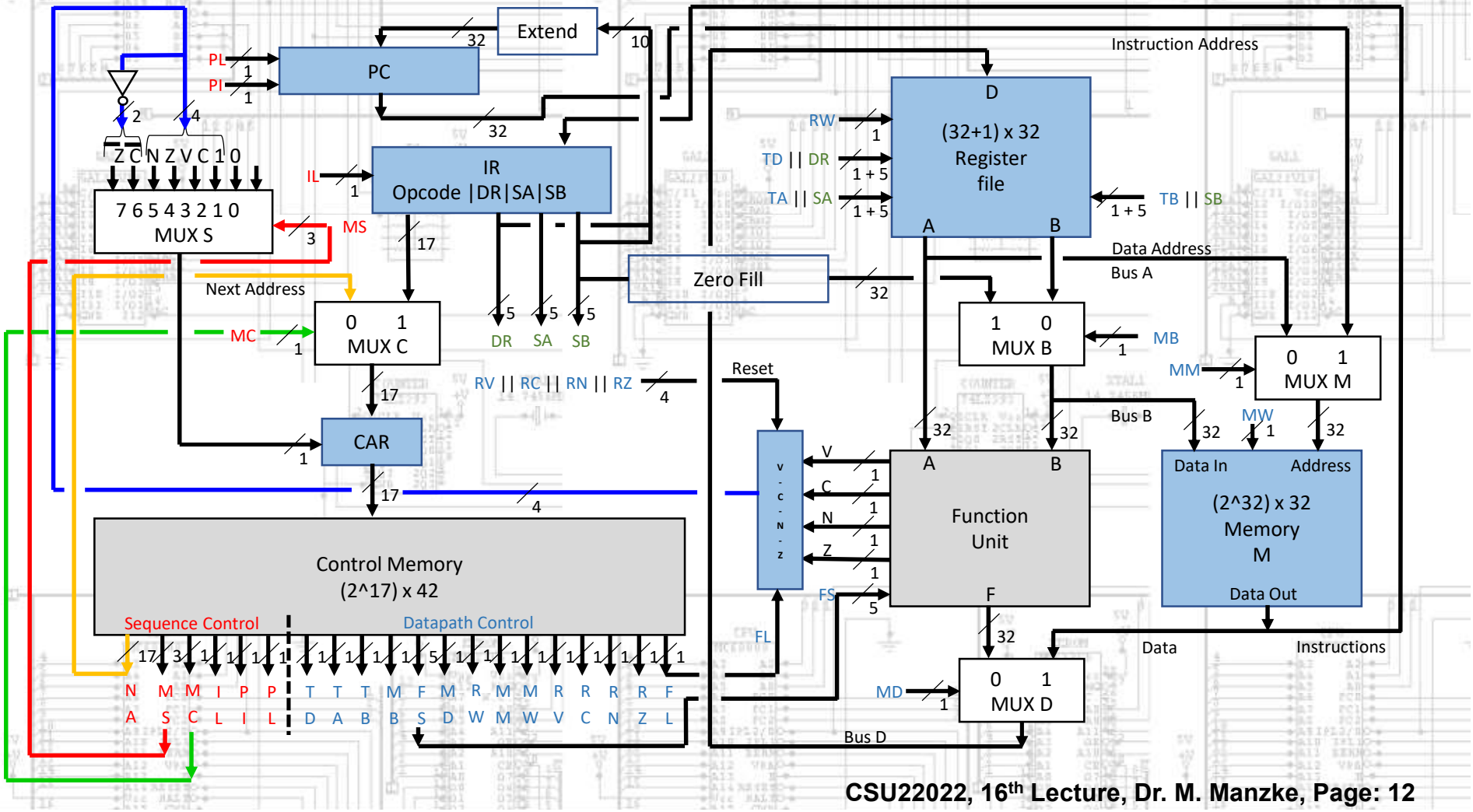


- The sources for the multiplexer can be:
  - Contents of the 17 bit NA Next Address field
  - 17 bit from the opcode field in the IR
- An opcode loaded into the CAR points to:
  - Microprogram in Control Memory
  - This program implements the instruction through the execution of a sequence of micro-operations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

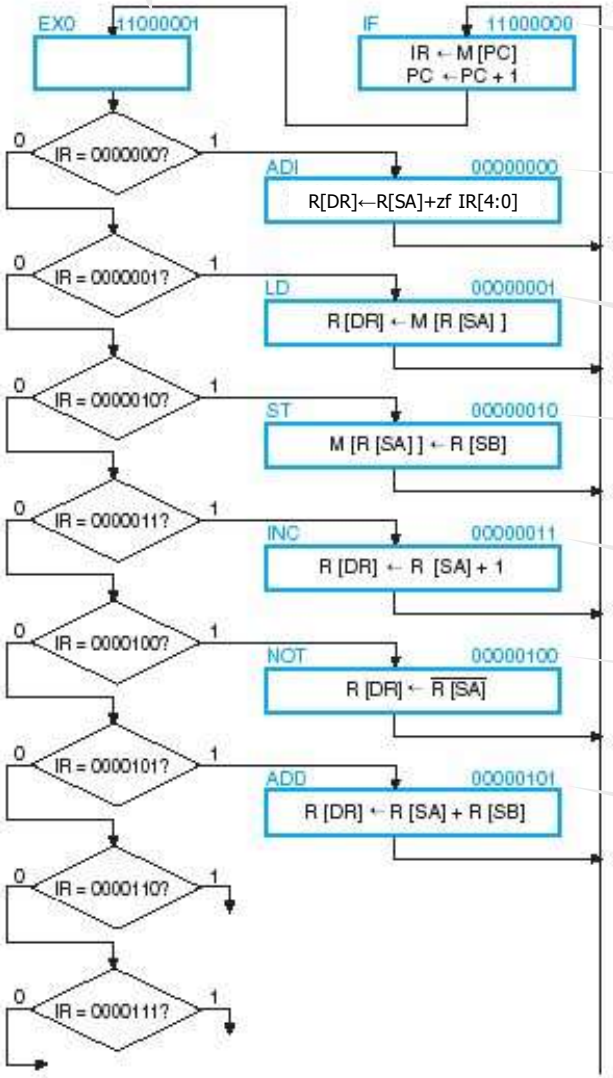
Sequencer Control Fields

MS		MC		IL		PI		PL		
Action	Symbolic Notation	Code	Select	Symbolic Notation	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									

NA - MS - MC


Extend
32
10
PL
PI
1
1
PC
32
Instruction Address
D
(32+1) x 32
Register
file
A
B
RW
TD || DR
TA || SA
1
1 + 5
1 + 5
TB || SB
Z C N Z V C 1 0
2
4
7 6 5 4 3 2 1 0
MUX S
IL
1
IR
Opcode |DR|SA|SB
MS
3
17
Zero Fill
32
Data Address
Bus A
Next Address
MC
1
0 1
MUX C
5
5
5
DR
SA
SB
1 0
MUX B
MB
1
MM
1
0 1
MUX M
RV || RC || RN || RZ
4
Reset
17
CAR
1
17
4
Bus B
32
32
32
MW
1
32
V
C
N
Z
1
1
1
1
FS
5
A
B
Function
Unit
F
Data In
Address
(2^32) x 32
Memory
M
Data Out
Control Memory
(2^17) x 42
Sequence Control
Datapath Control
17 3 1 1 1 1 1 1 1 1 1 1 1 5 1 1 1 1 1 1 1 1 1 1
N A
M S
M C
I L
P I
P L
T D
T A
T B
M B
F S
M D
R W
M W
M M
R W
R C
R N
R Z
F L
FL
32
MD
1
0 1
MUX D
Bus D
Data
Instructions

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|2422|21|20|19|18|17|16|15|14|13  9|8|7|6|5|4|3|2|1|0|
-- | Next Address  | MS | M| I| P| P| T| T| T| M|  FS |M|R|M|M|R|R|R|R|F|
-- | Next Address  | MS | C| L| I| L| D| A| B| B|  FS |D|W|M|W|V|C|N|Z|L|

-- ADI              R[DR]←R[SA]+zf 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);
```