Multiple-Cycle Design

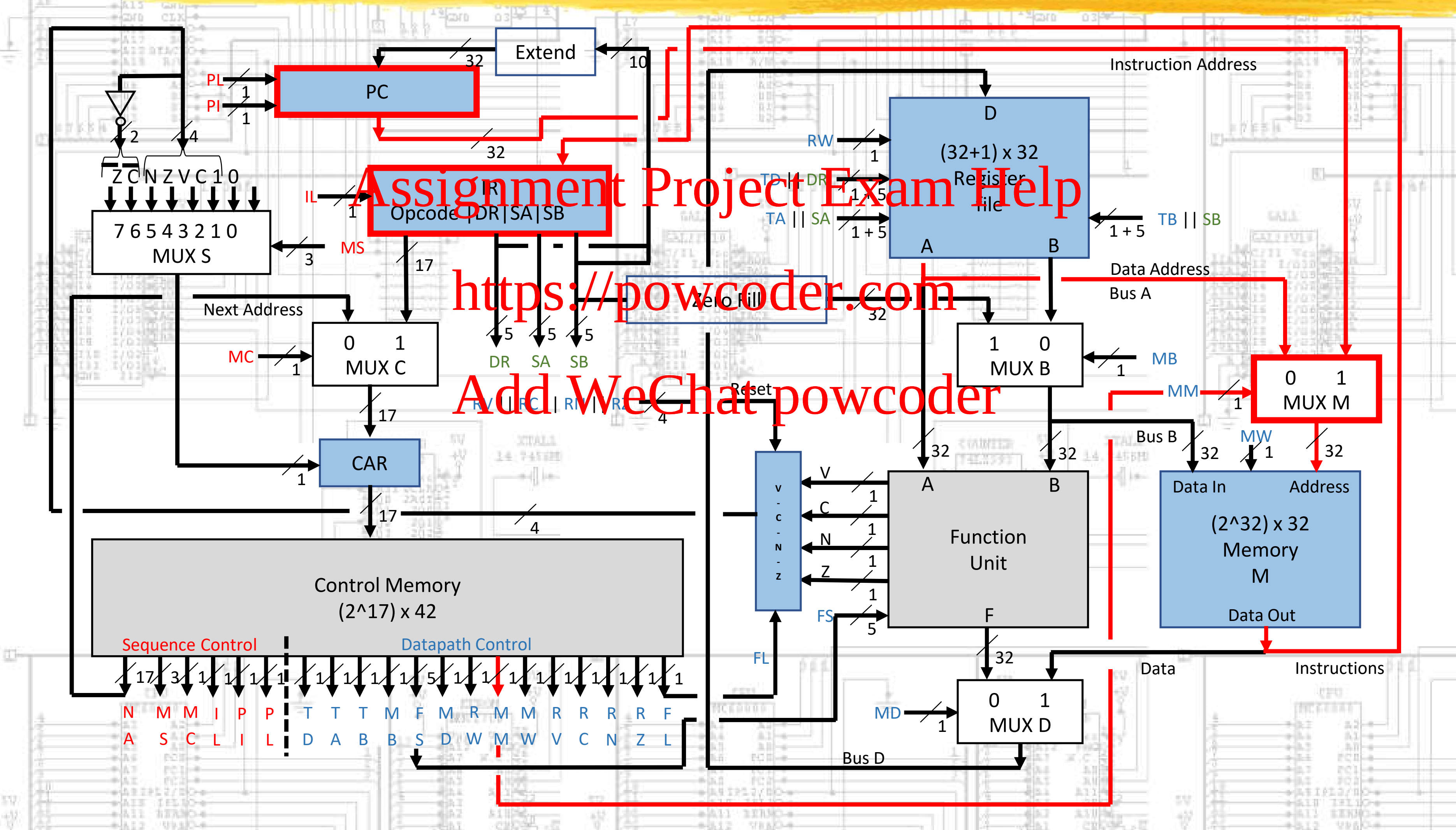
- The Multiple-Cycle Implementation demonstrates the use of a single memory for:
  - Data
  - Instruction
- This design is also used to show the implementation of more complex instructions

Memory M Address

41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Next Address																	MS			MC	IL	PI	PL	TD	TA	TB	MB	ES					MD	RW	MM	MW	RV	RC	RN	RZ	FL

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

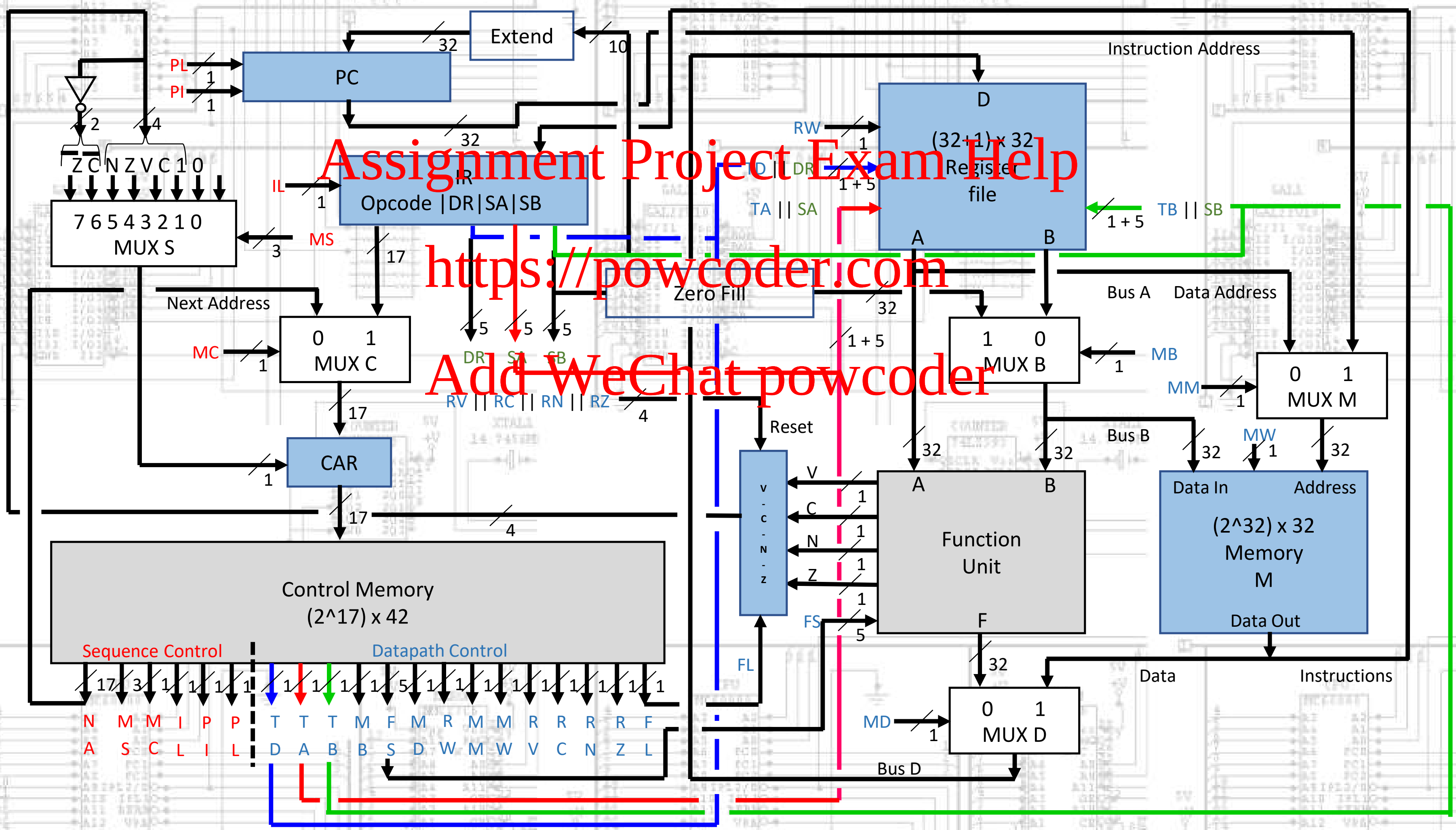
41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Next Address																	MS			MC	IL	PI	PL	TD	TA	TB	MB	FS					MD	RW	MM	MW	RV	RC	RN	RZ	FL

- 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

41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Next Address																	MS			MC	IL	PI	PL	TD	TA	TB	MB	FS					MD	RW	MM	MW	RV	RC	RN	RZ	FL

- 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

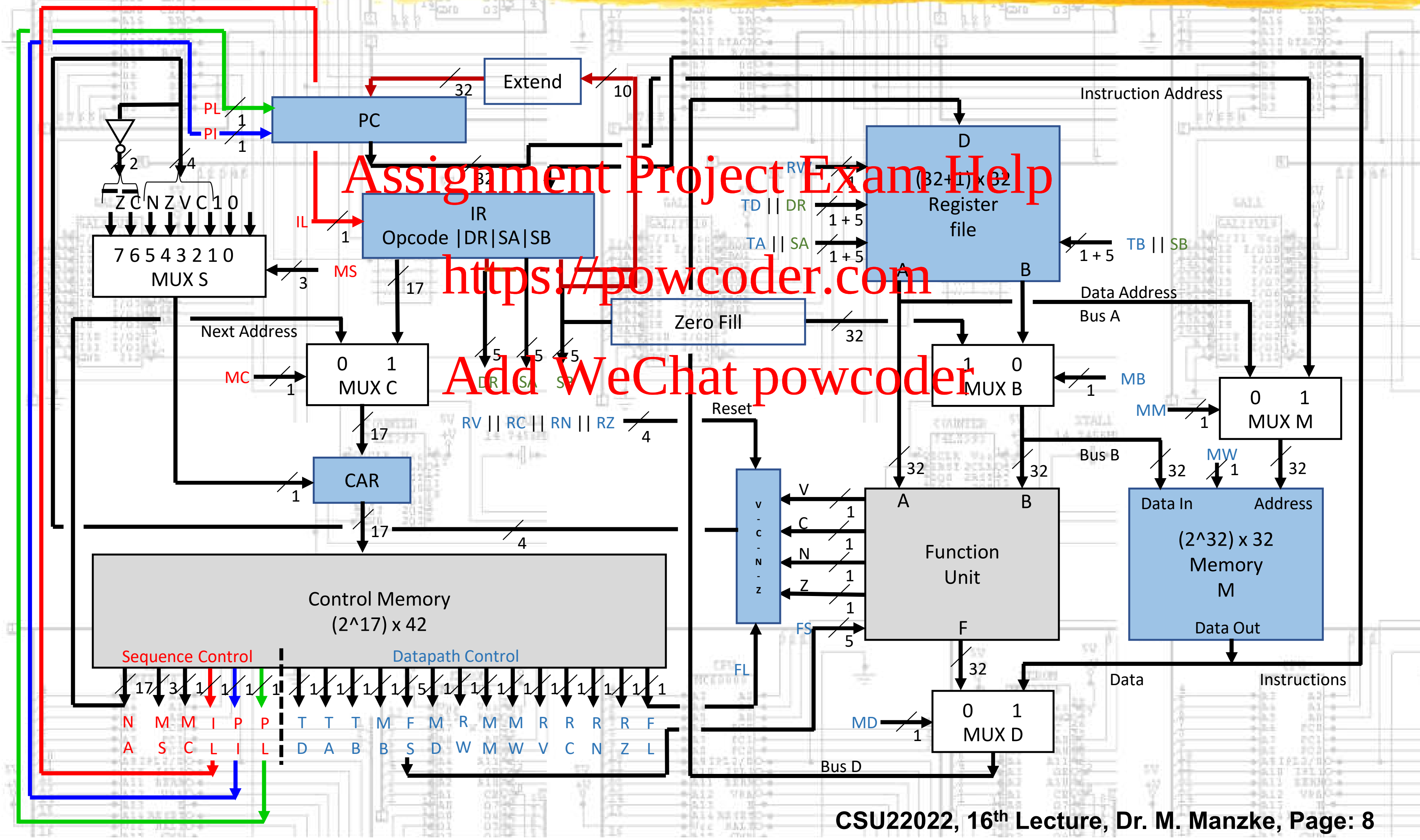
PC Program Counter Register

41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Next Address																	MS			MC	IL	PI	PL	TD	TA	TB	MB	FS					MD	RW	MM	MW	RV	RC	RN	RZ	FL

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

Next Address Logic

41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Next Address																	MS			MC	IL	PI	PL	TD	TA	TB	MB	FS					MD	RW	MM	MW	RV	RC	RN	RZ	FL

- 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

<table>
<tr><td>41</td><td>40</td><td>39</td><td>38</td><td>37</td><td>36</td><td>35</td><td>34</td><td>33</td><td>32</td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="17">Next Address</td><td colspan="3">MS</td><td>MC</td><td>IL</td><td>PI</td><td>PL</td><td>TD</td><td>TA</td><td>TB</td><td>MB</td><td colspan="5">FS</td><td>MD</td><td>RW</td><td>MM</td><td>MW</td><td>RV</td><td>RC</td><td>RN</td><td>RZ</td><td>FL</td></tr>
</table>

- The sources for the multiplexer can be:

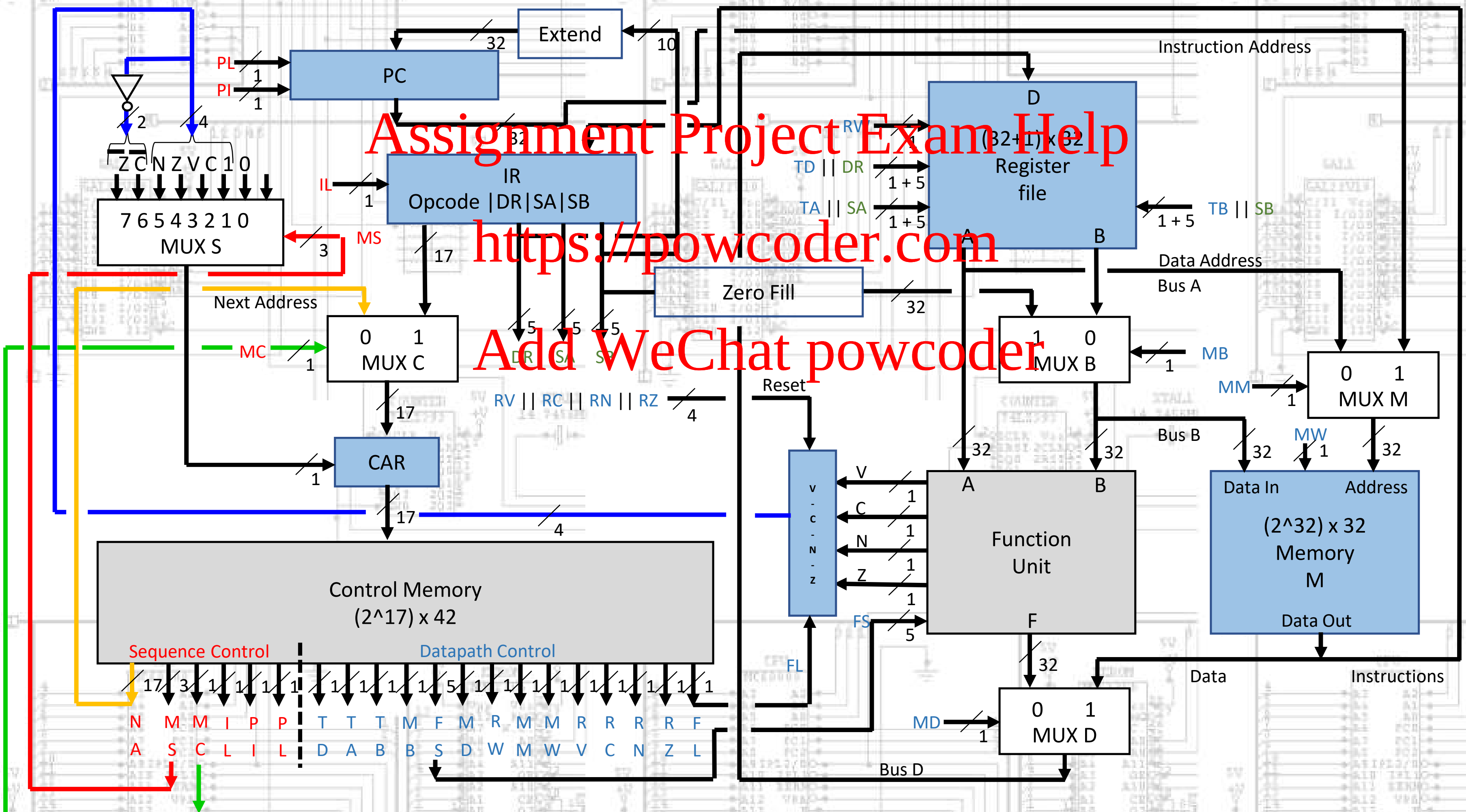
  - Contents of the 17 bit NA Next Address field
  - 17 bit from the opcode field in the IR

- 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

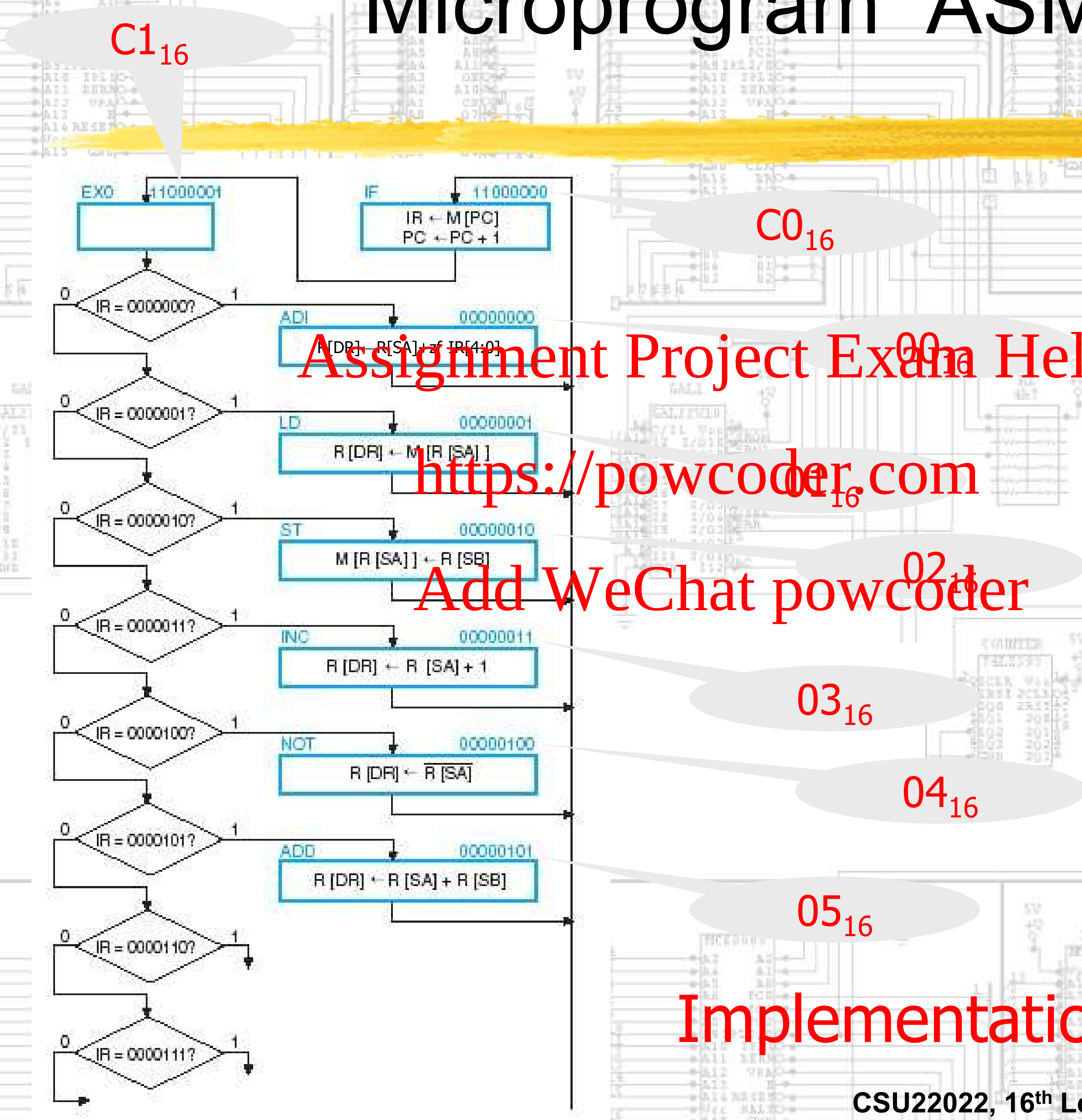
MS			MC		IL		PI		PL		
Action	**Symbolic Notation**	**Code**	**Select**	**Symbolic Notation**	**Action**	**Symbolic Notation**	**Action**	**Symbolic Notation**	**Action**	**Symbolic Notation**	**Code**
Increment *CAR*	CNT	000	NA	NXA	No load	NLI	No load	NLP	No load	NLP	0
Load *CAR*	NXT	001	Opcode	OPC	Load 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
PC
PL
PI
IR
Opcode |DR|SA|SB
IL
Z C N Z V C 1 0
7 6 5 4 3 2 1 0
MUX S
MS
Next Address
MC
0 1
MUX C
CAR
Control Memory
(2^17) x 42
Sequence Control
Datapath Control
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
RV || RC || RN || RZ
Zero Fill
Reset
V-C-N-Z
V
C
N
Z
FS
FL
D
(32+1) x 32
Register
file
A
B
TD || DR
TA || SA
TB || SB
1 + 5
1
0
MUX B
MB
MM
0 1
MUX M
MW
Instruction Address
Data Address
Bus A
Bus B
Function
Unit
F
Data In
Address
(2^32) x 32
Memory
M
Data Out
Data
Instructions
MD
0 1
MUX D
Bus D
DR
SA
SB
32
17
10
5
4
3
2
1

Microprogram ASM



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]
  "000000000???????? ???  ?  ?  ?  ?  ?  ?  ?   ? ????? ? ? ? ? ? ? ? ? ?",-- 00
-- LD               R[DR]←M[R[SA]]
  "000000000???????? ???  ?  ?  ?  ?  ?  ?  ?   ? ????? ? ? ? ? ? ? ? ? ?",-- 01
-- ST               M[R[SA]]←R[SB]
  "000000000???????? ???  ?  ?  ?  ?  ?  ?  ?   ? ????? ? ? ? ? ? ? ? ? ?",-- 02
-- INC              R[DR]←R[SA]+1
  "000000000???????? ???  ?  ?  ?  ?  ?  ?  ?   ? ????? ? ? ? ? ? ? ? ? ?",-- 03
-- NOT              R[DR]←NOT[R[SA]]
  "000000000???????? ???  ?  ?  ?  ?  ?  ?  ?   ? ????? ? ? ? ? ? ? ? ? ?",-- 04
-- ADD              R[DR]←R[SA]+R[SB]
  "000000000???????? ???  ?  ?  ?  ?  ?  ?  ?   ? ????? ? ? ? ? ? ? ? ? ?",-- 05

                                  :

-- IF               IR←M[PC], PC←PC+1
  "000000000???????? ???  ?  ?  ?  ?  ?  ?  ?   ? ????? ? ? ? ? ? ? ? ? ?",-- C0
-- EX0              CAR←IR[31:15]
  "000000000???????? ???  ?  ?  ?  ?  ?  ?  ?   ? ????? ? ? ? ? ? ? ? ? ?",-- C1
                                  :
                                  );
      variable addr : integer;
      variable control_out : std_logic_vector(41 downto 0);
```