SCRAM Instructions

HW1 due tonight!

Philipp Koehn

~~21 February 2018~~
13 Sept. 2019

Binary data

fgets / scanf / etc.

$\Longrightarrow$ text/strings

Binary input:

getc, fgetc

return 0-255

or EOF (-1)

fread

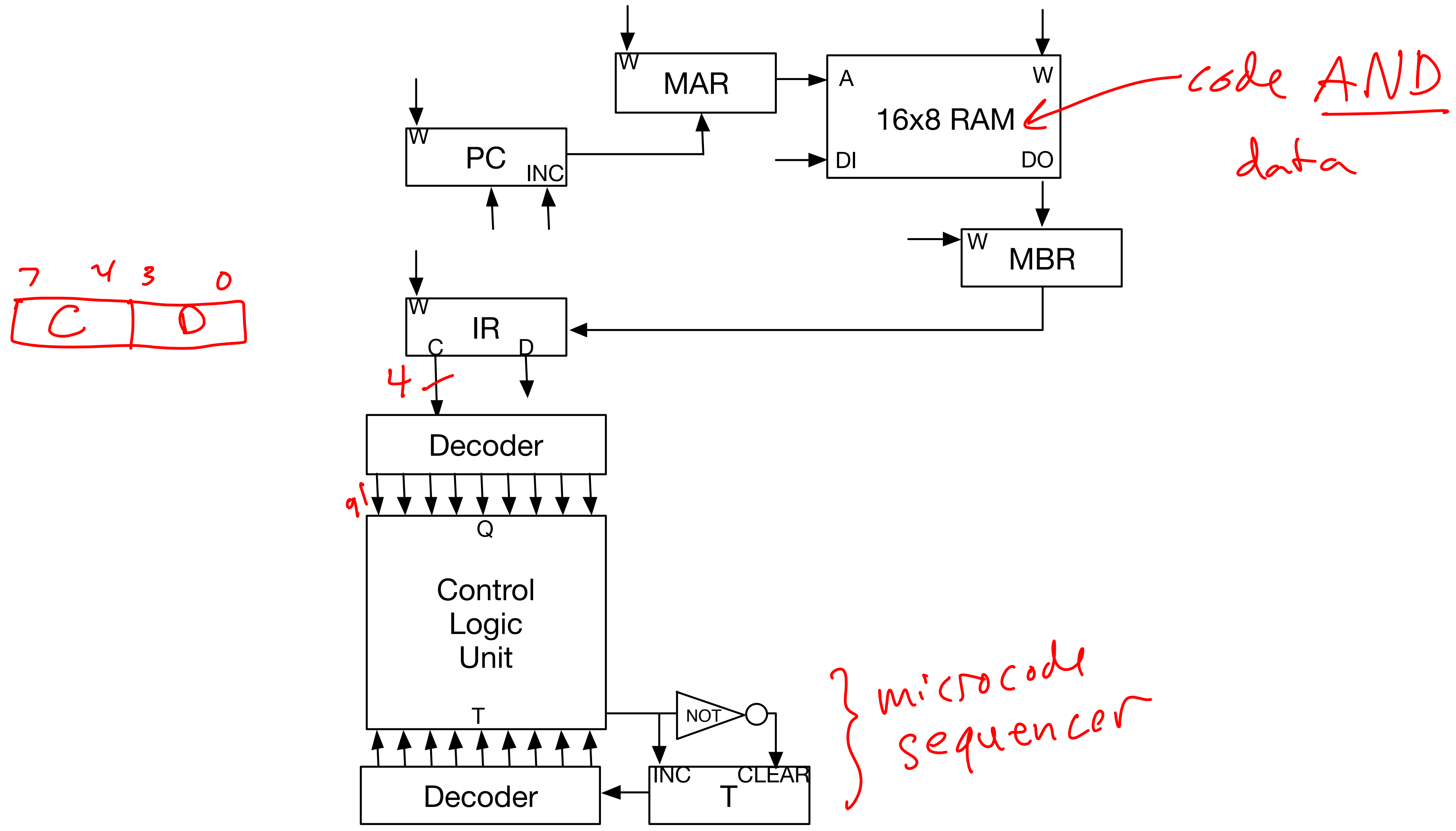
Reminder

- Fully work through a computer
  - circuit
  - assembly code

- Simple but Complete Random Access Machine (SCRAM)
  - every instruction is 8 bit
  - 4 bit for op-code: 9 different operations (of 16 possible)
  - 4 bit for address: 16 bytes of memory

- Background reading on web page
  - The Random Access Machine
  - The SCRAM

Circuit (At This Point)



Instruction Fetch

- Retrieve instruction from memory
- Increase program counter

Time	Command
t_0	MAR $\leftarrow$ PC
t_1	MBR $\leftarrow$ M, PC $\leftarrow$ PC + 1
t_2	IR $\leftarrow$ MBR

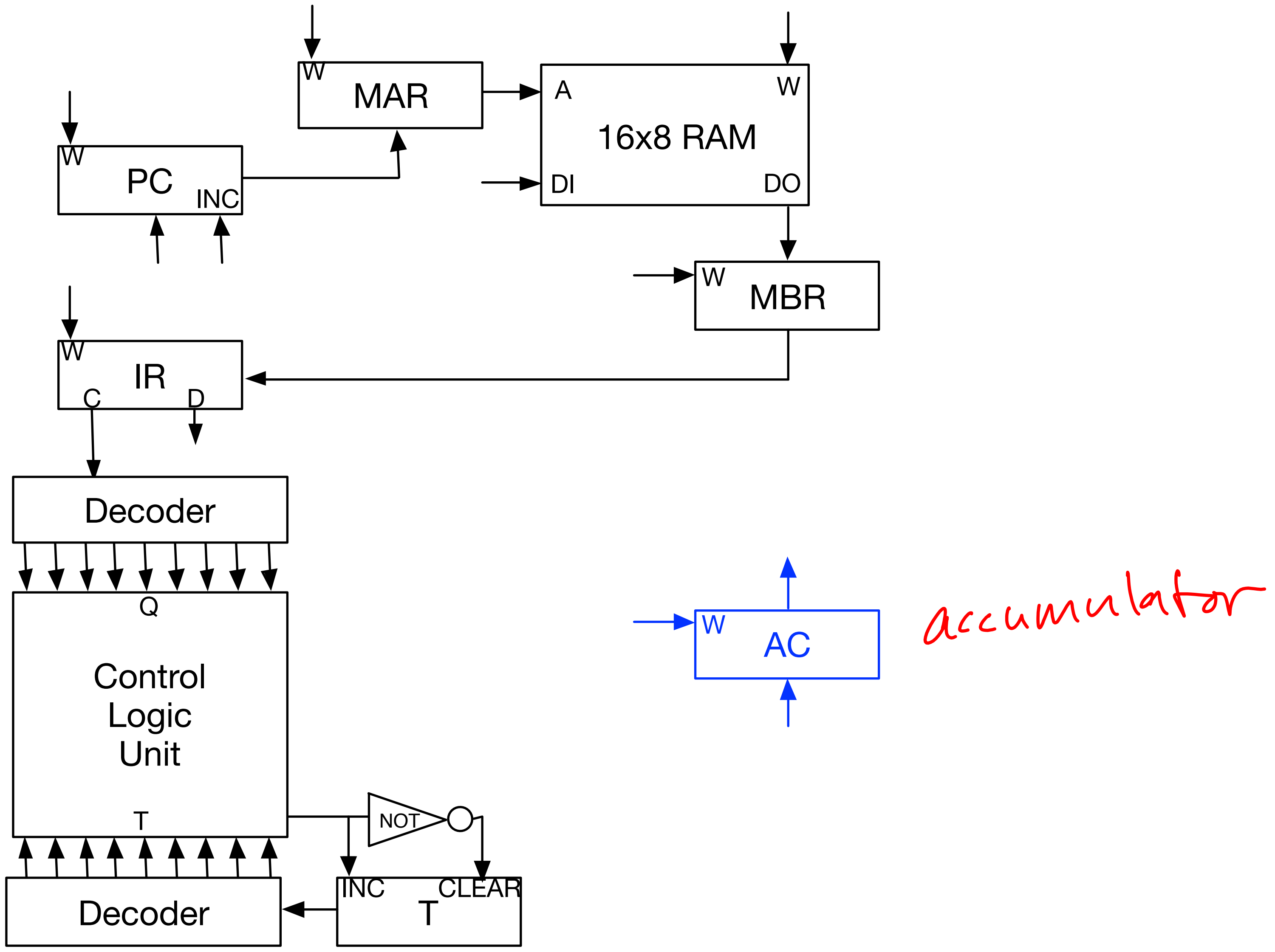
`lda`

Micro Program

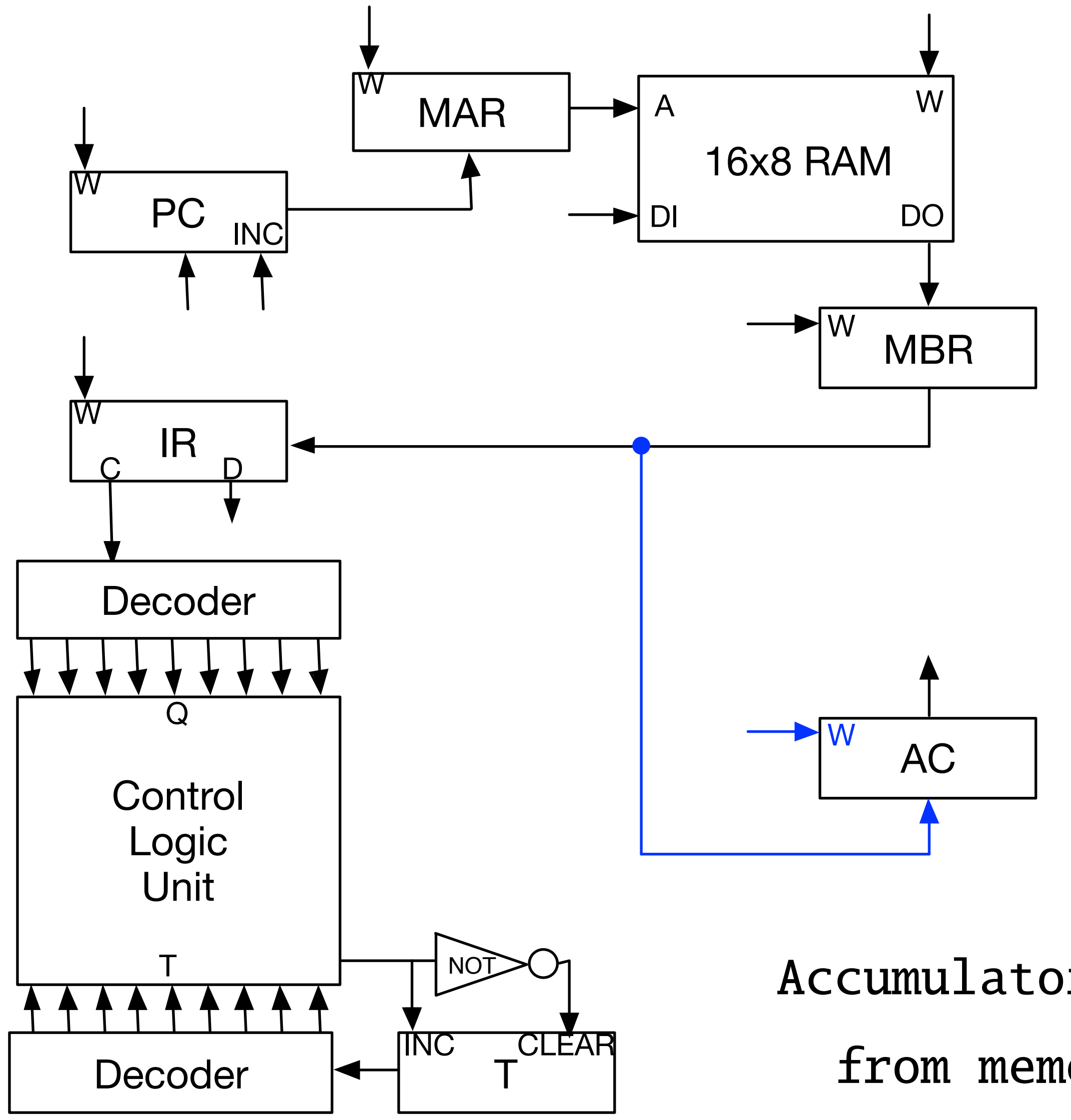
- Load into accumulator

Op Code	Time	Command
q_1	t_3	MAR $\leftarrow$ IR(D)
q_1	t_4	MBR $\leftarrow$ M
q_1	t_5	AC $\leftarrow$ MBR, end of microcode

Accumulator

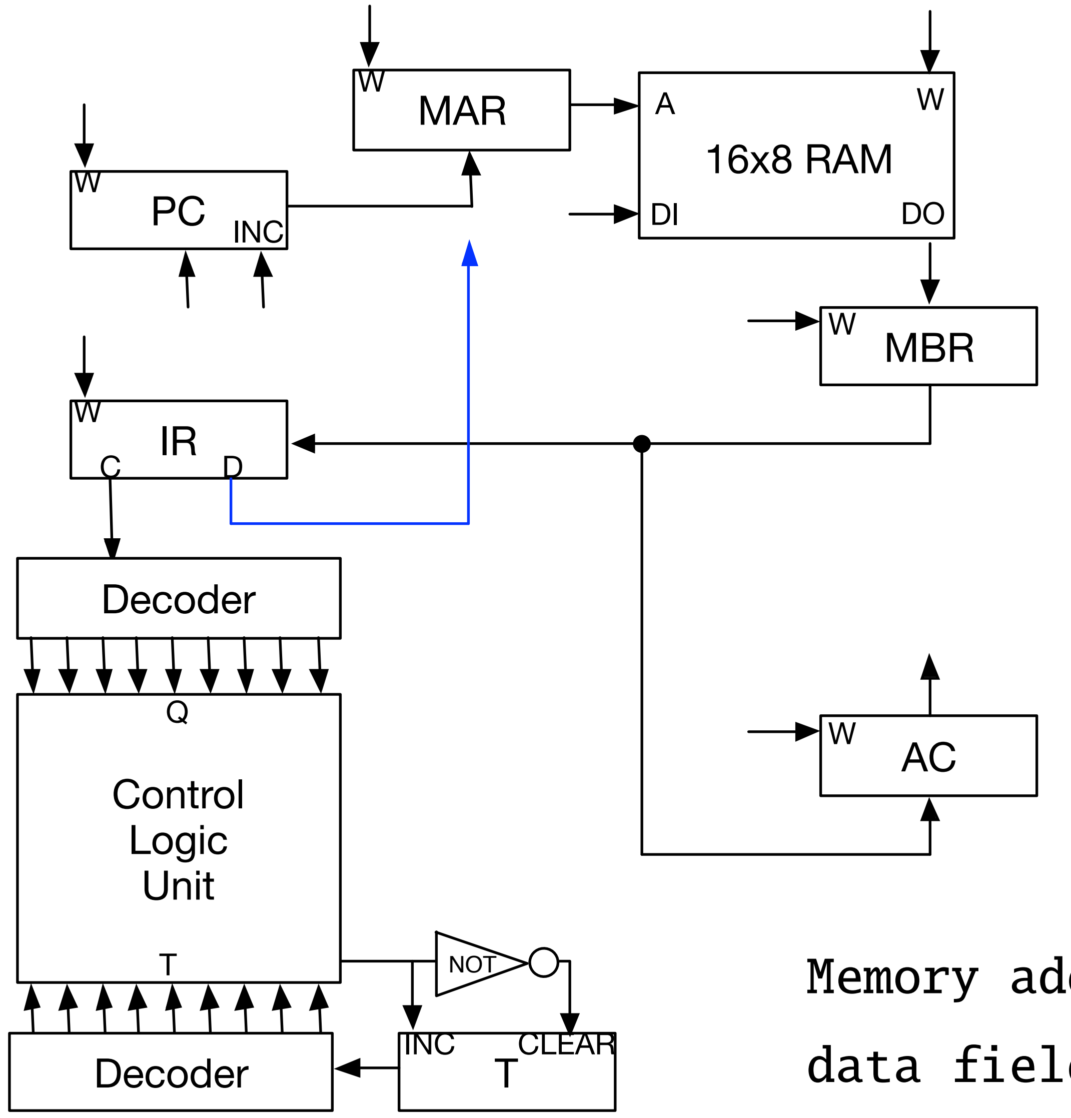


AC $\leftarrow$ MBR



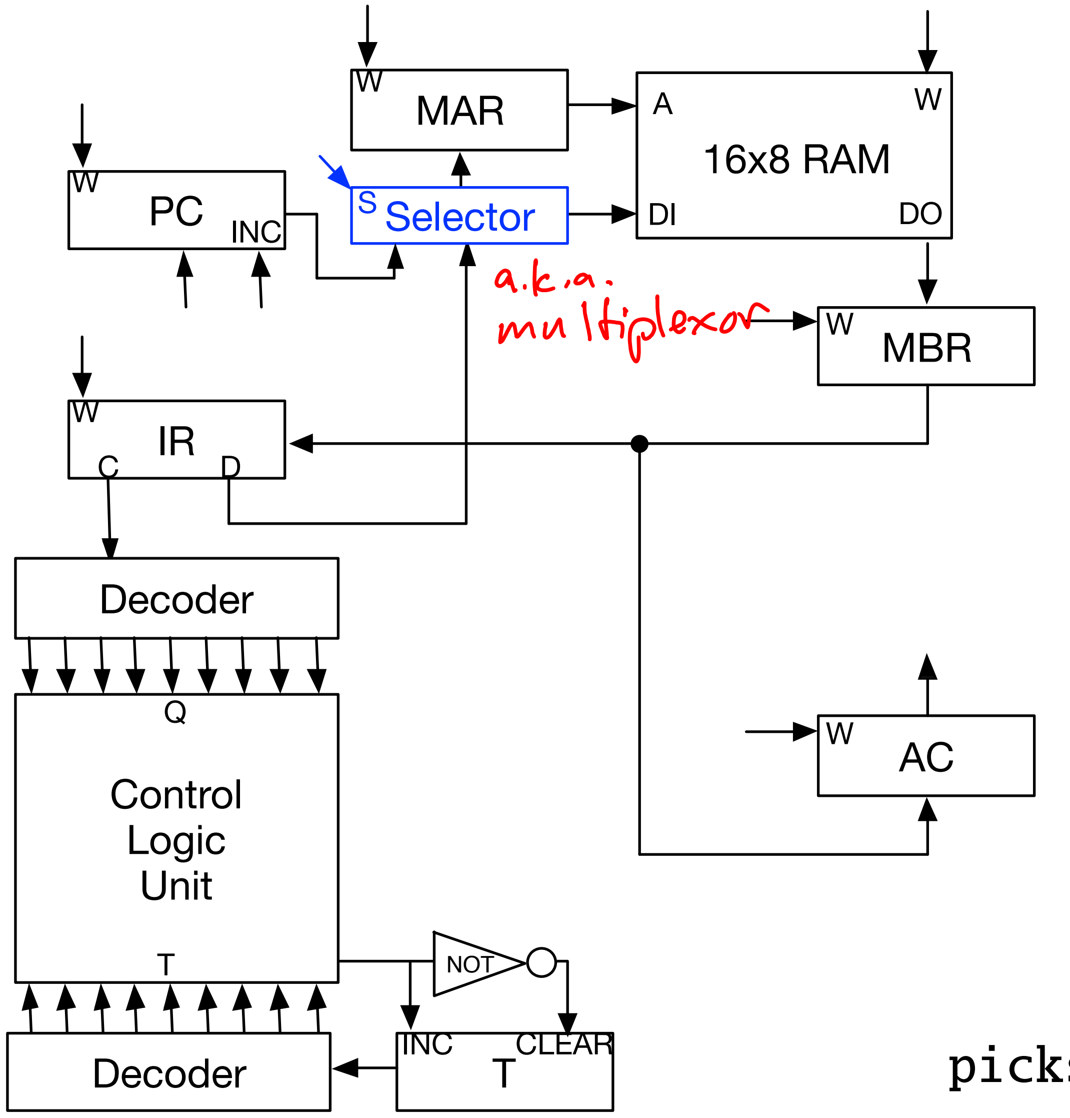
Accumulator receives value from memory buffer (MBR)

MAR ← IR(D)



Memory address comes from data field of instruction

MAR $\leftarrow$ IR(D)



Selector

picks between inputs

let's do this again

but focus on flags

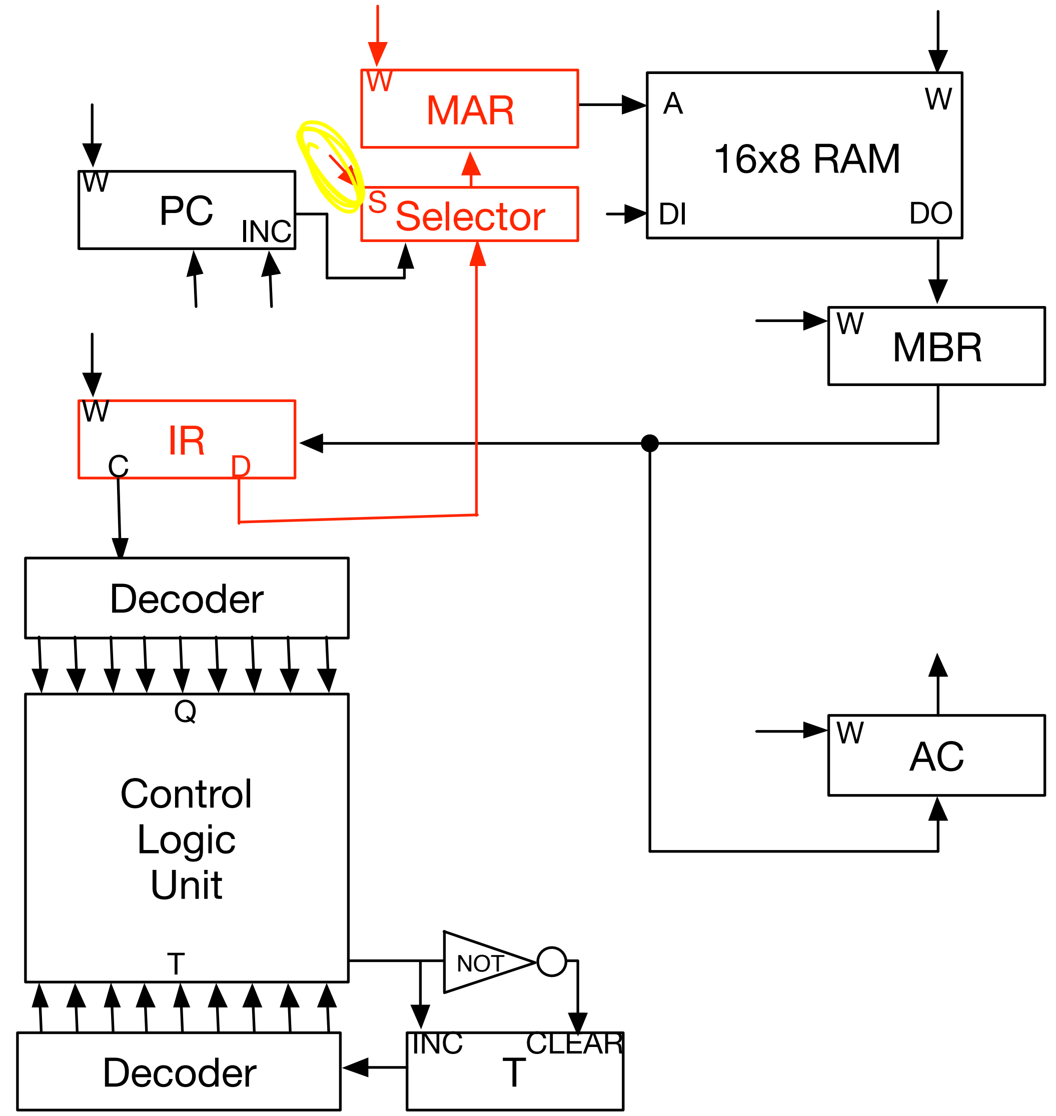
Micro Program

- Load into accumulator

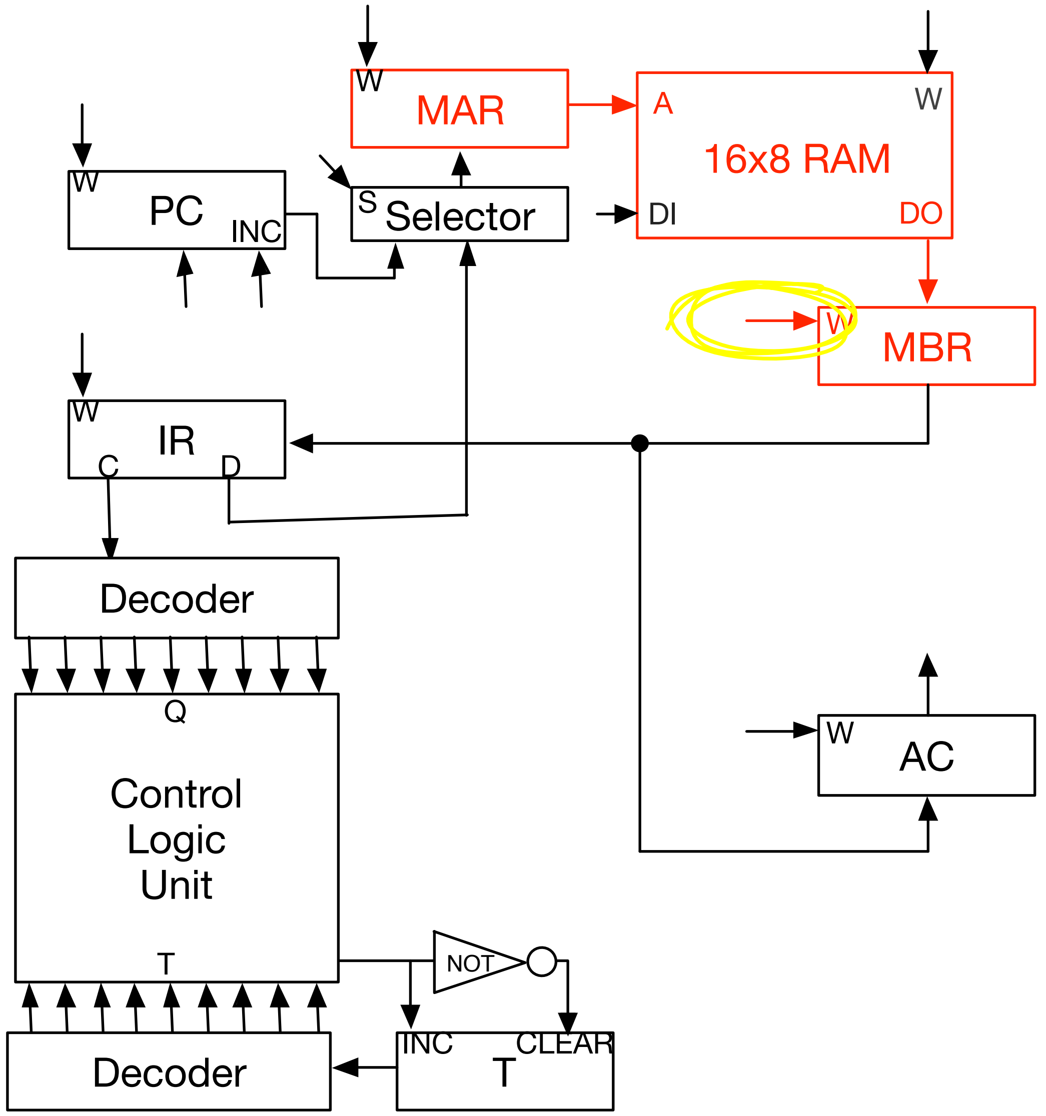
Op Code	Time	Command
q_1	t_3	MAR $\leftarrow$ IR(D)
q_1	t_4	MBR $\leftarrow$ M
q_1	t_5	AC $\leftarrow$ MBR

$\Big\}$ control signals

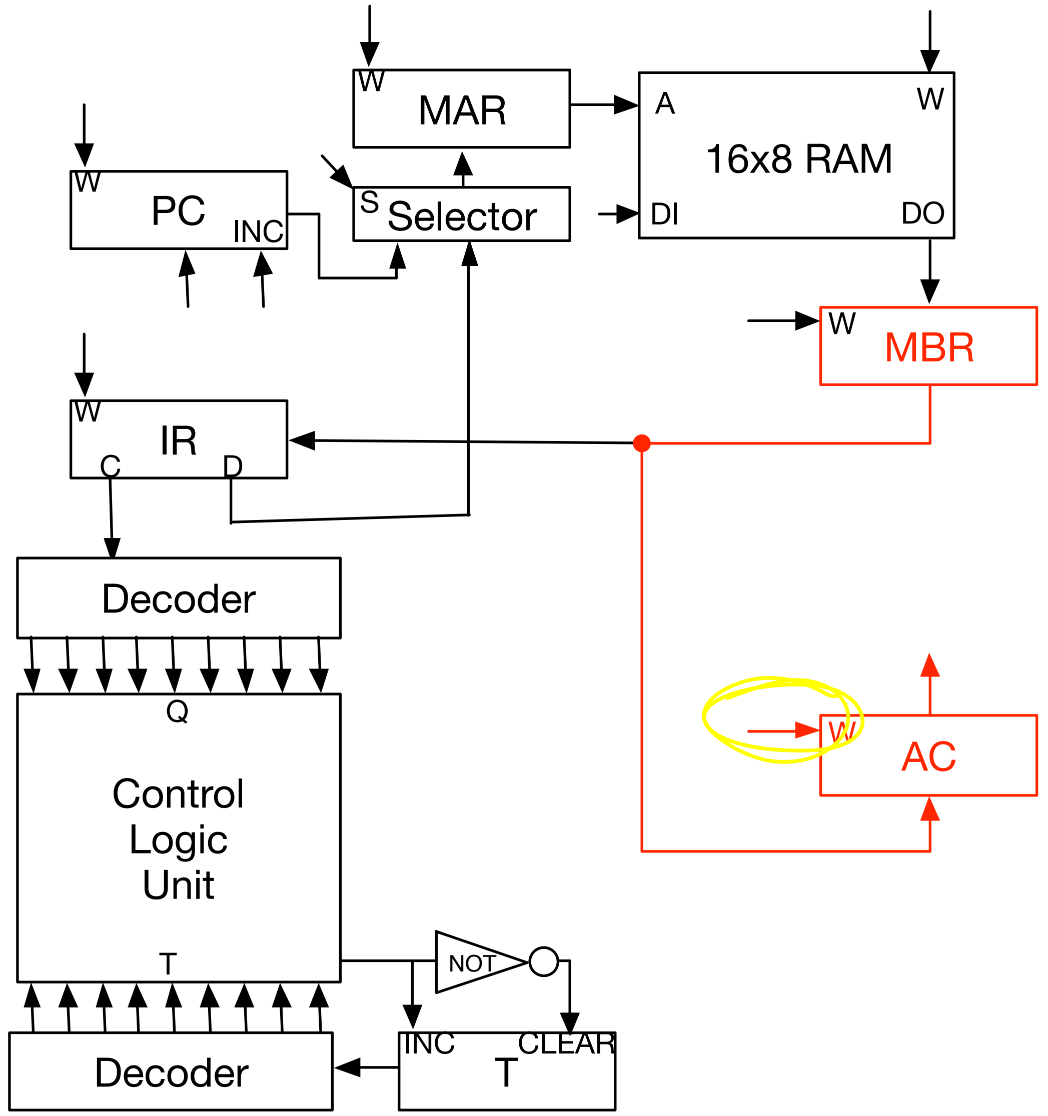
q_1 t_3: MAR $\leftarrow$ IR(D)



q_1 t_4: **MBR** $\leftarrow$ **M**



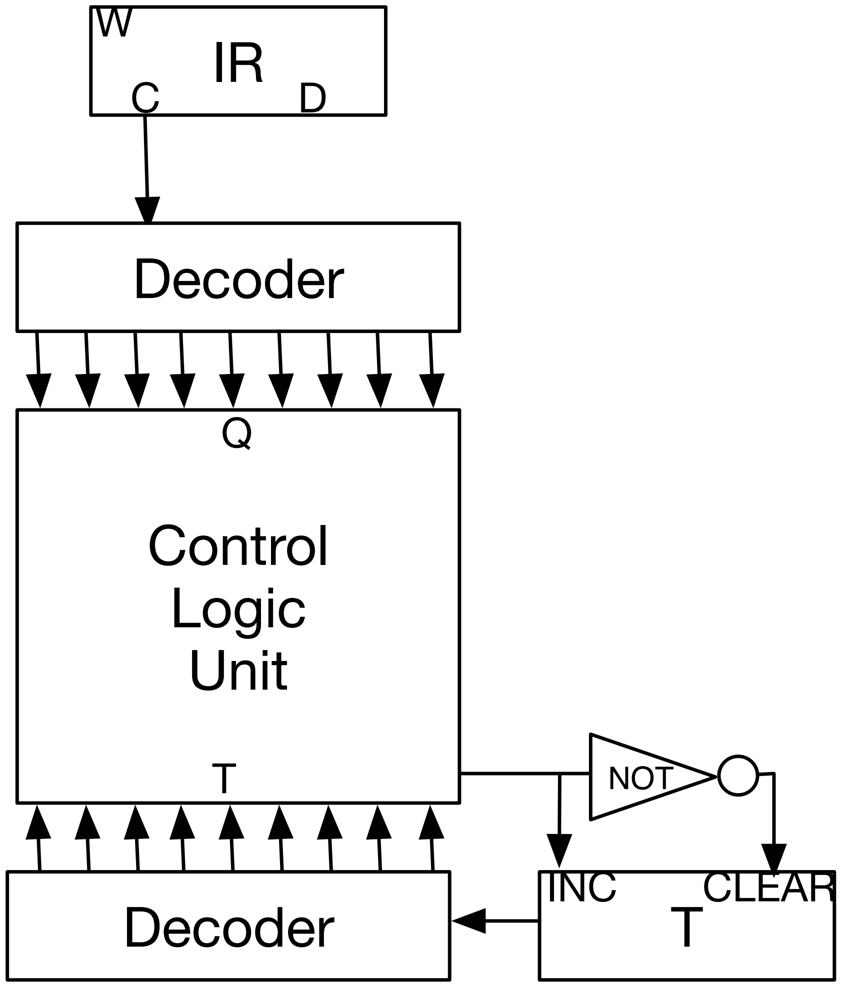
q_1 t_5: **AC** ← **MBR**


W
MAR
A
W
16x8 RAM
W
PC
INC
S
Selector
DI
DO
W
MBR
W
IR
C
D
W
AC
Decoder
Q
Control
Logic
Unit
T
NOT
INC
CLEAR
T
Decoder

control logic unit

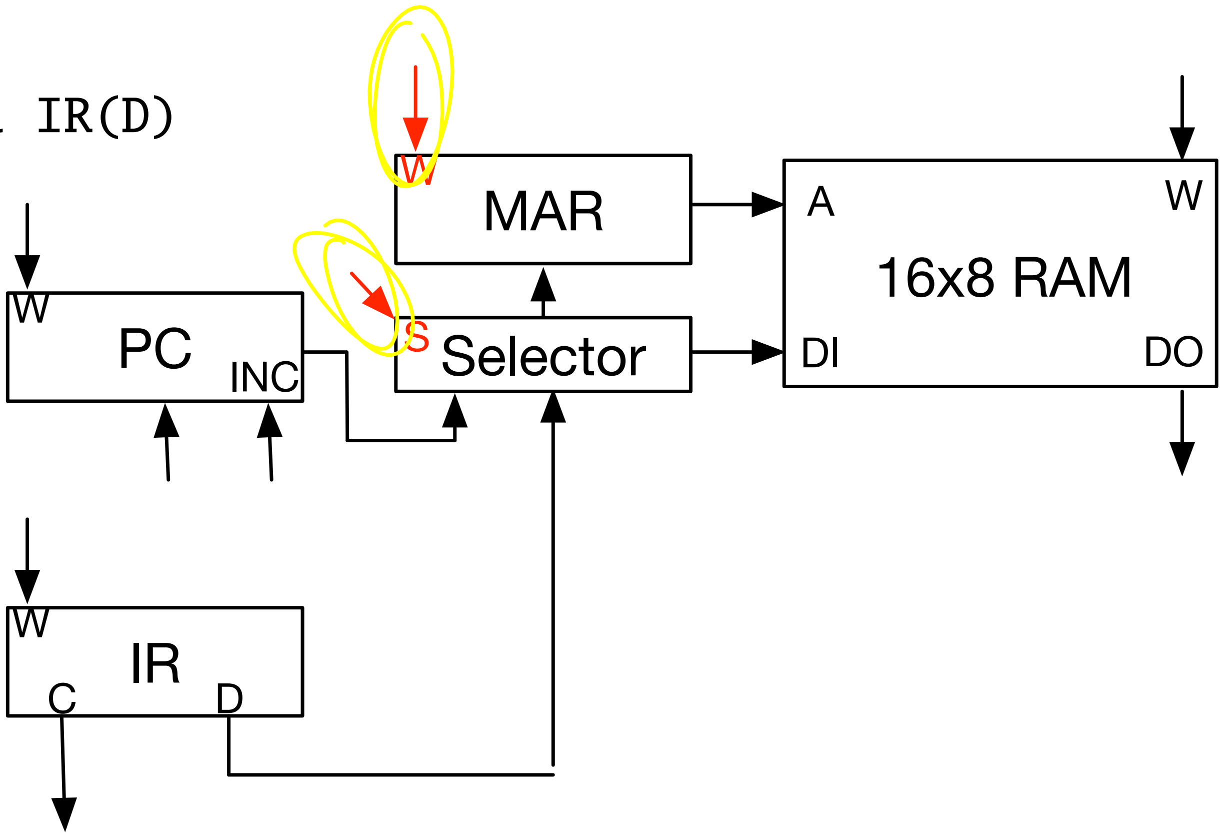
Objective

- Given
  - Instruction op code Q
  - Time step in micro program T
- Output
  - signals to register transfer
  - signals to selectors

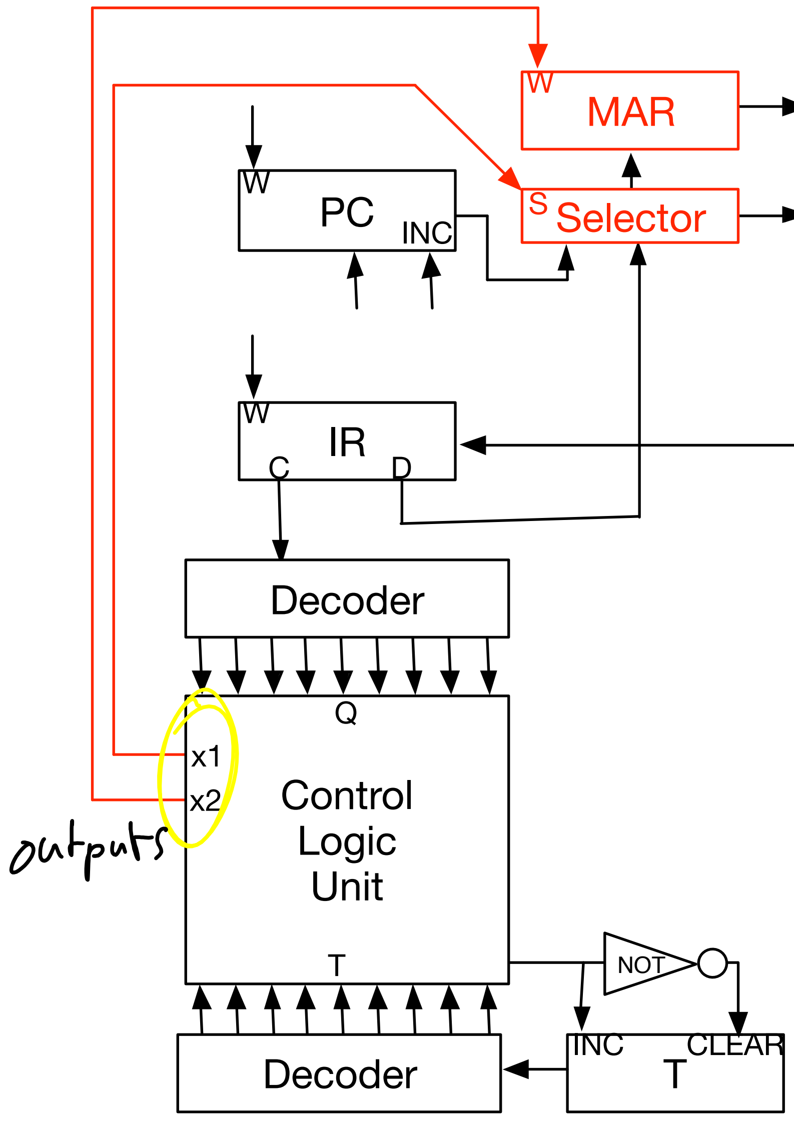


Example

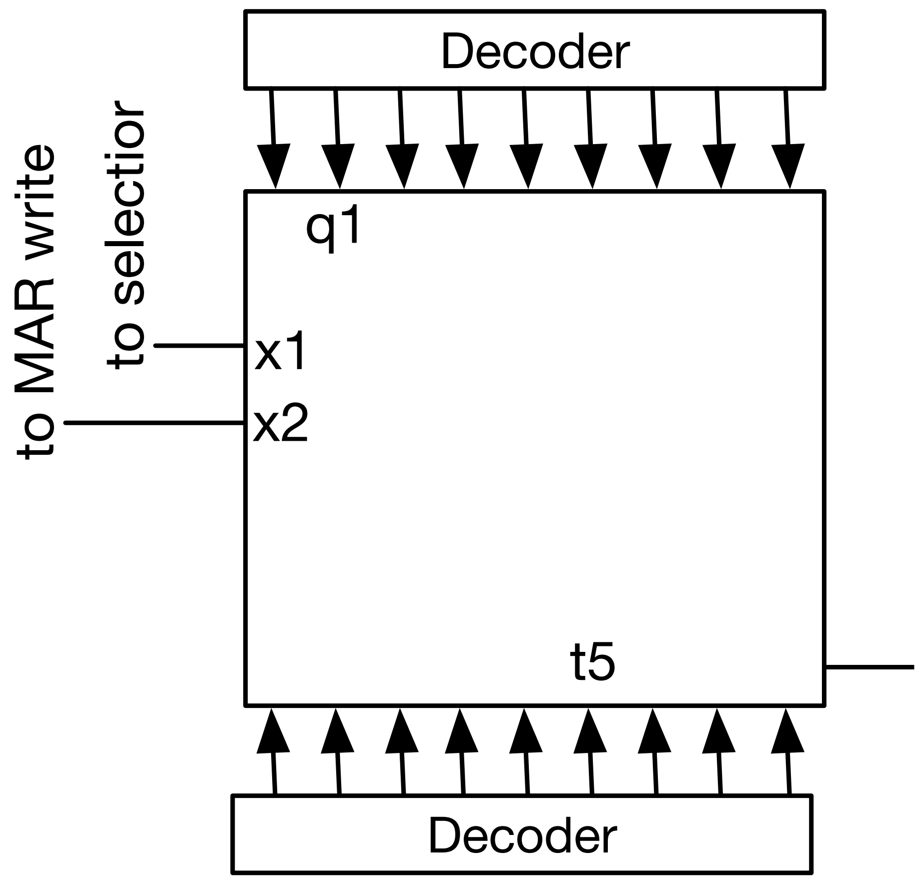
- Step in micro program: q_1 t_3 MAR $\leftarrow$ IR(D)
- Needs to signal
  - MAR write flag set
  - MAR's selector to input IR(D)



Add Wires to the Circuit

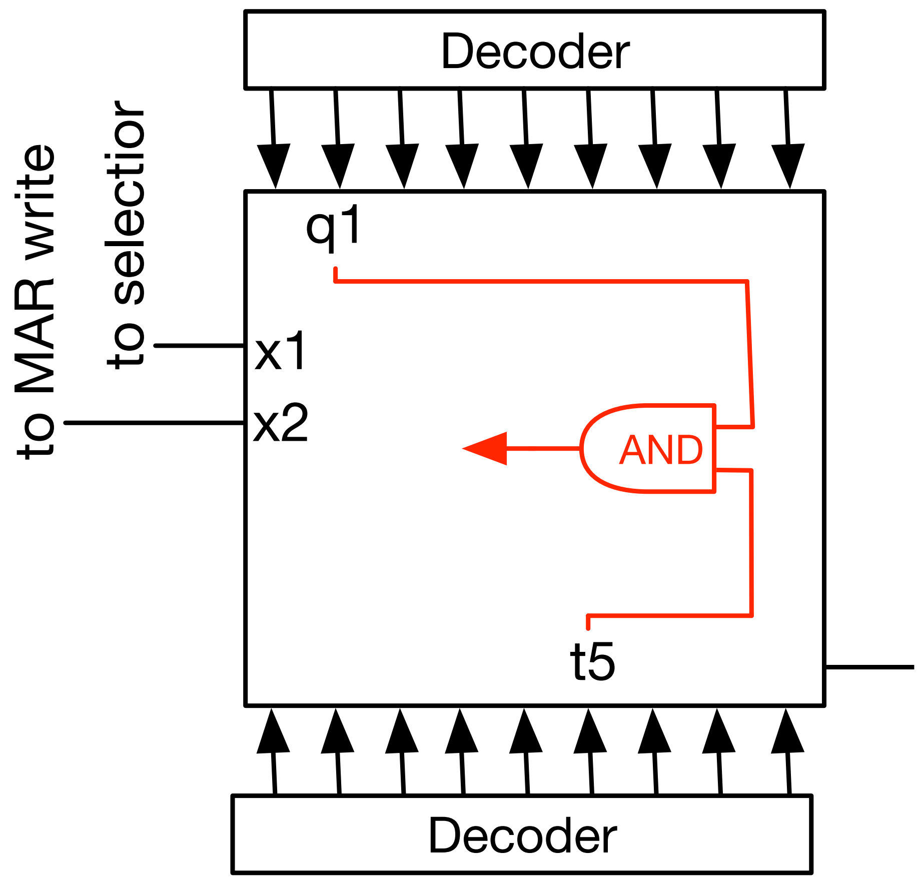


Inside the Control Logic Unit



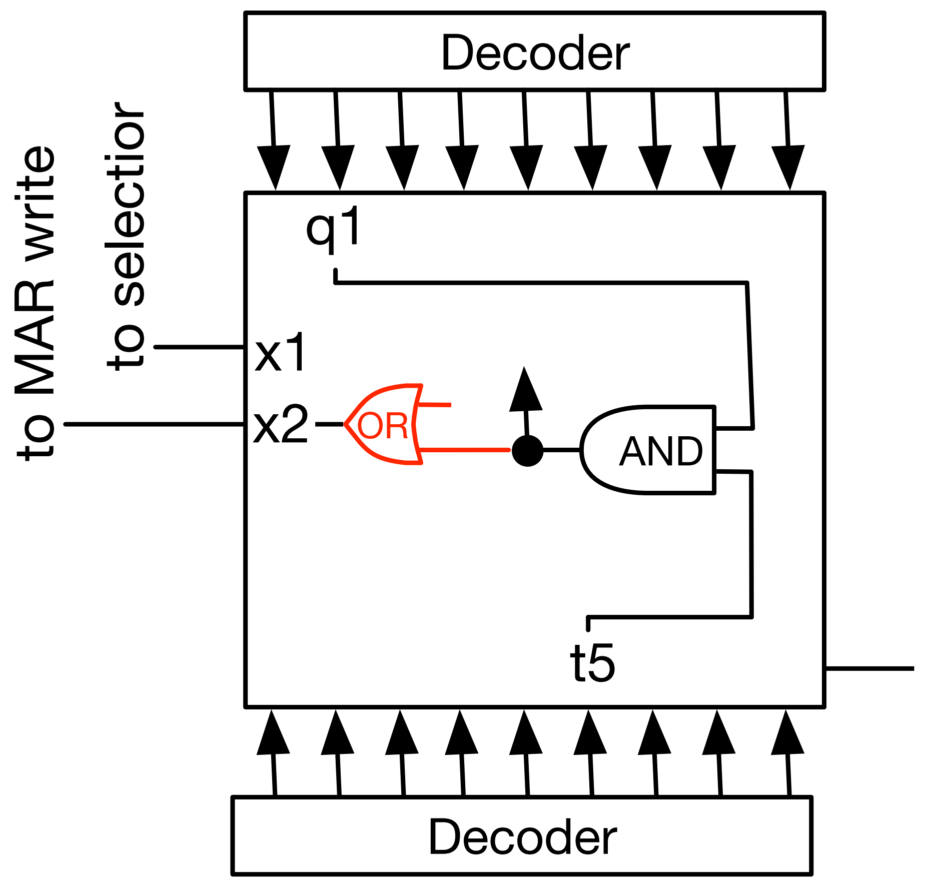
Micro instruction: q_1 AND t_5: MAR $\leftarrow$ IR(D)

Inside the Control Logic Unit



Micro instruction: q_1 AND t_5: MAR $\leftarrow$ IR(D)

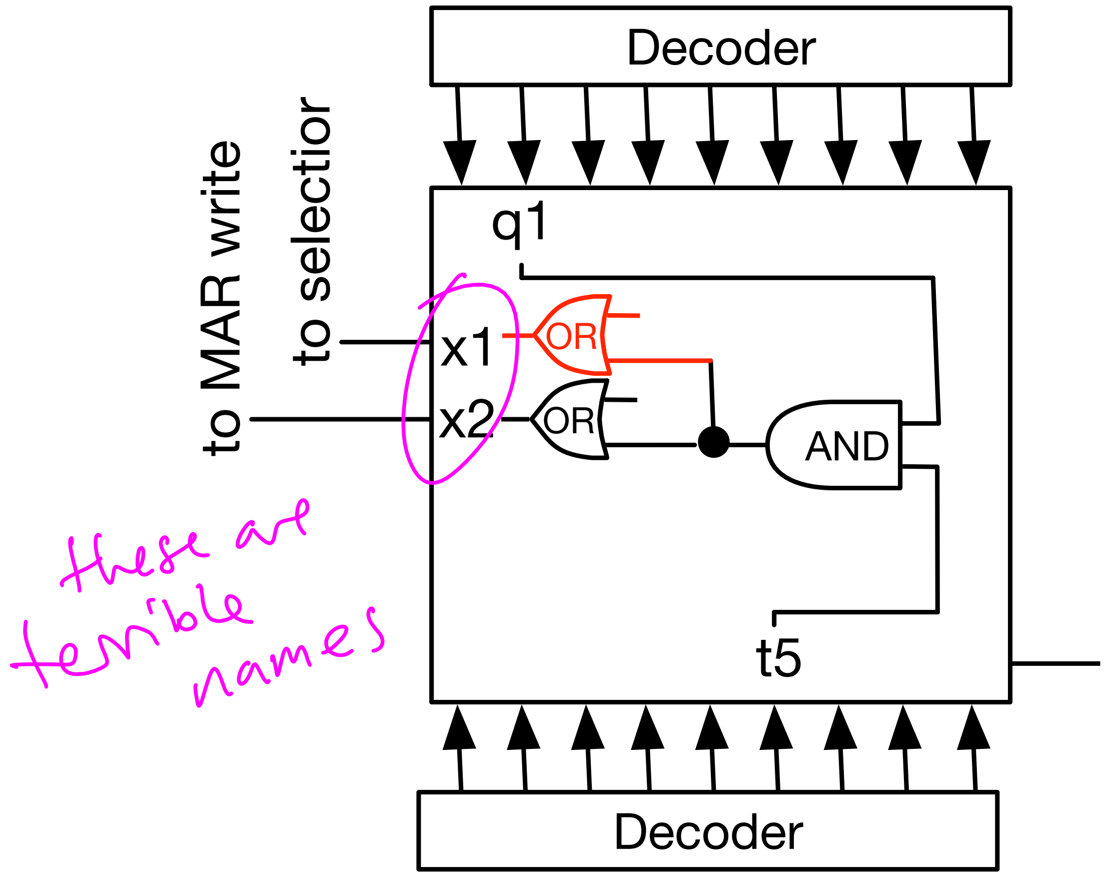
Inside the Control Logic Unit



Micro instruction: q_1 AND t_5: MAR ← IR(D)

Set signal to MAR write flag

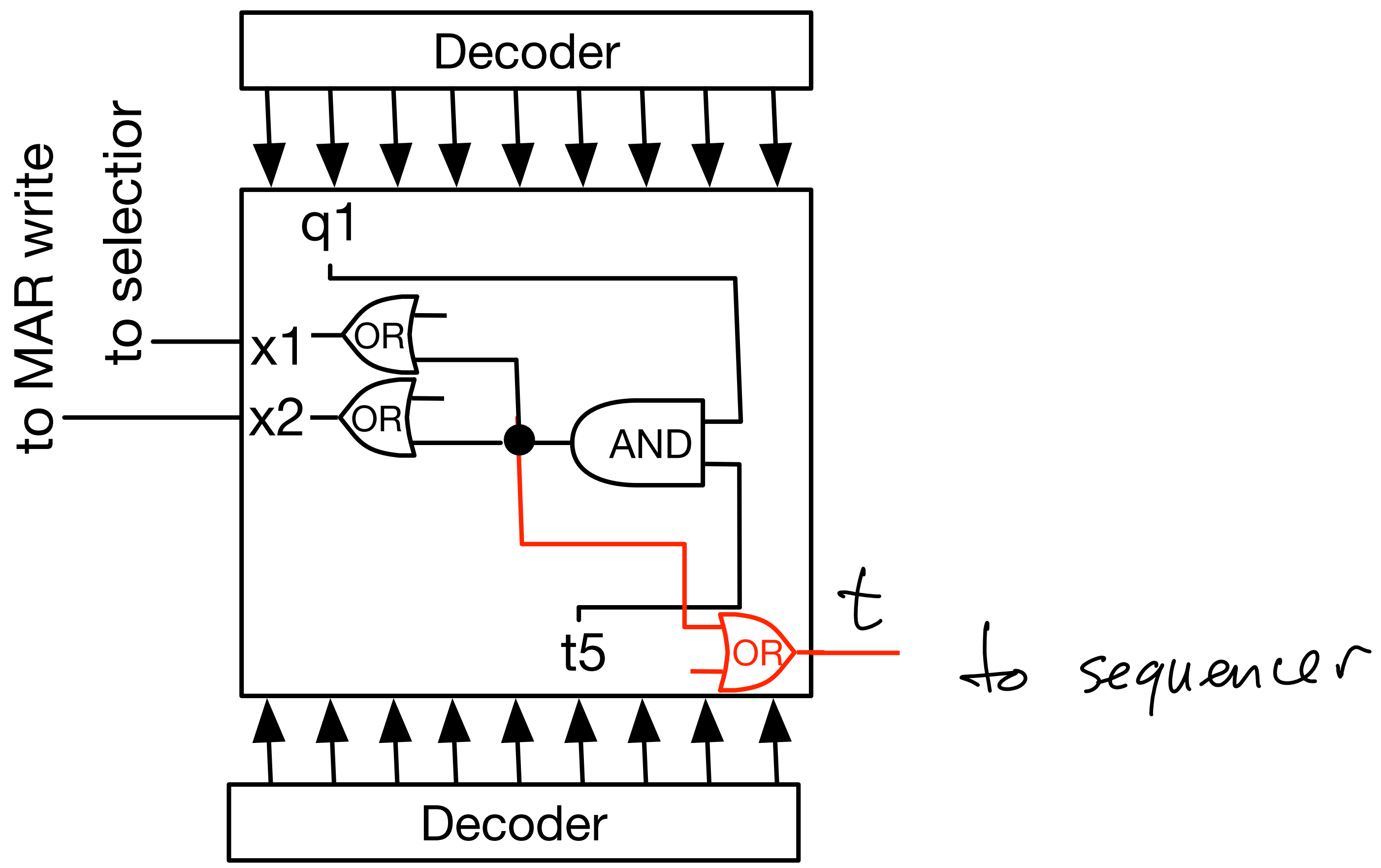
Inside the Control Logic Unit



Micro instruction: q_1 AND t_5: MAR $\leftarrow$ IR(D)

Set appropriate value to MAR selector

Inside the Control Logic Unit



Micro instruction: q_1 AND t_5: MAR $\leftarrow$ IR(D)

Increase micro program time step

Inside the Control Logic Unit

Control logic is a large matrix

$$\begin{array}{c|ccccccccc}
 & t_0 & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 \\
q_0 & * & * & * & * & * & * & * & * & * \\
q_1 & * & * & * & * & * & * & * & * & * \\
q_2 & * & * & * & * & * & * & * & * & * \\
q_3 & * & * & * & * & * & * & * & * & * \\
q_4 & * & * & * & * & * & * & * & * & * \\
q_5 & * & * & * & * & * & * & * & * & * \\
q_6 & * & * & * & * & * & * & * & * & * \\
q_7 & * & * & * & * & * & * & * & * & * \\
q_8 & * & * & * & * & * & * & * & * & *
\end{array}$$

Inside the Control Logic Unit

Control logic is a large matrix

$$
\begin{array}{c|ccccccccc}
 & t_0 & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 \\
q_0 & * & * & * & * & * & * & * & * & * \\
q_1 & * & * & * & * & * & * & * & * & * \\
q_2 & * & * & * & * & * & * & * & * & * \\
q_3 & * & * & * & * & * & * & * & * & * \\
q_4 & * & * & * & * & * & * & * & * & * \\
q_5 & * & * & * & * & * & * & * & * & * \\
q_6 & * & * & * & * & * & * & * & * & * \\
q_7 & * & * & * & * & * & * & * & * & * \\
q_8 & * & * & * & * & * & * & * & * & *
\end{array}
$$

Inside the Control Logic Unit

Control logic is a large matrix

Could use ROM!

$$\begin{array}{cccccccccc}
 & t_0 & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 \\
q_0 & * & * & * & * & * & * & * & * & * \\
q_1 & * & * & * & * & * & x_1x_2t & * & * & * \\
q_2 & * & * & * & * & * & * & * & * & * \\
q_3 & * & * & * & * & * & * & * & * & * \\
q_4 & * & * & * & * & * & * & * & * & * \\
q_5 & * & * & * & * & * & * & * & * & * \\
q_6 & * & * & * & * & * & * & * & * & * \\
q_7 & * & * & * & * & * & * & * & * & * \\
q_8 & * & * & * & * & * & * & * & * & *
\end{array}$$

`ldi`

LDI: Load Indirectly

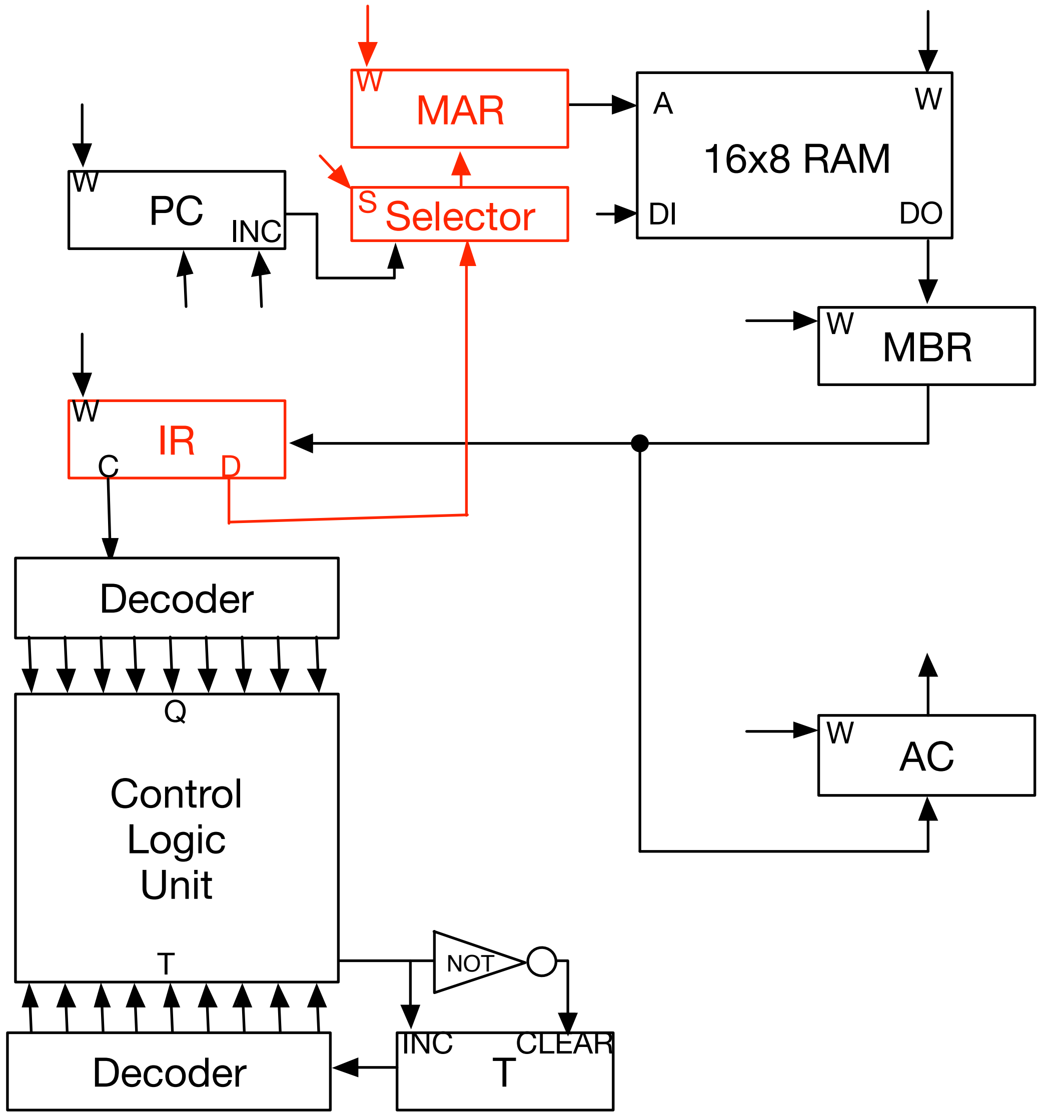
- Specified memory address contains address for value
- Basically a pointer operation
- Steps
  - load value of specified memory address
  - use that value as a memory address (second lookup)
  - store value from second lookup into accumulator

Micro Program for LDI

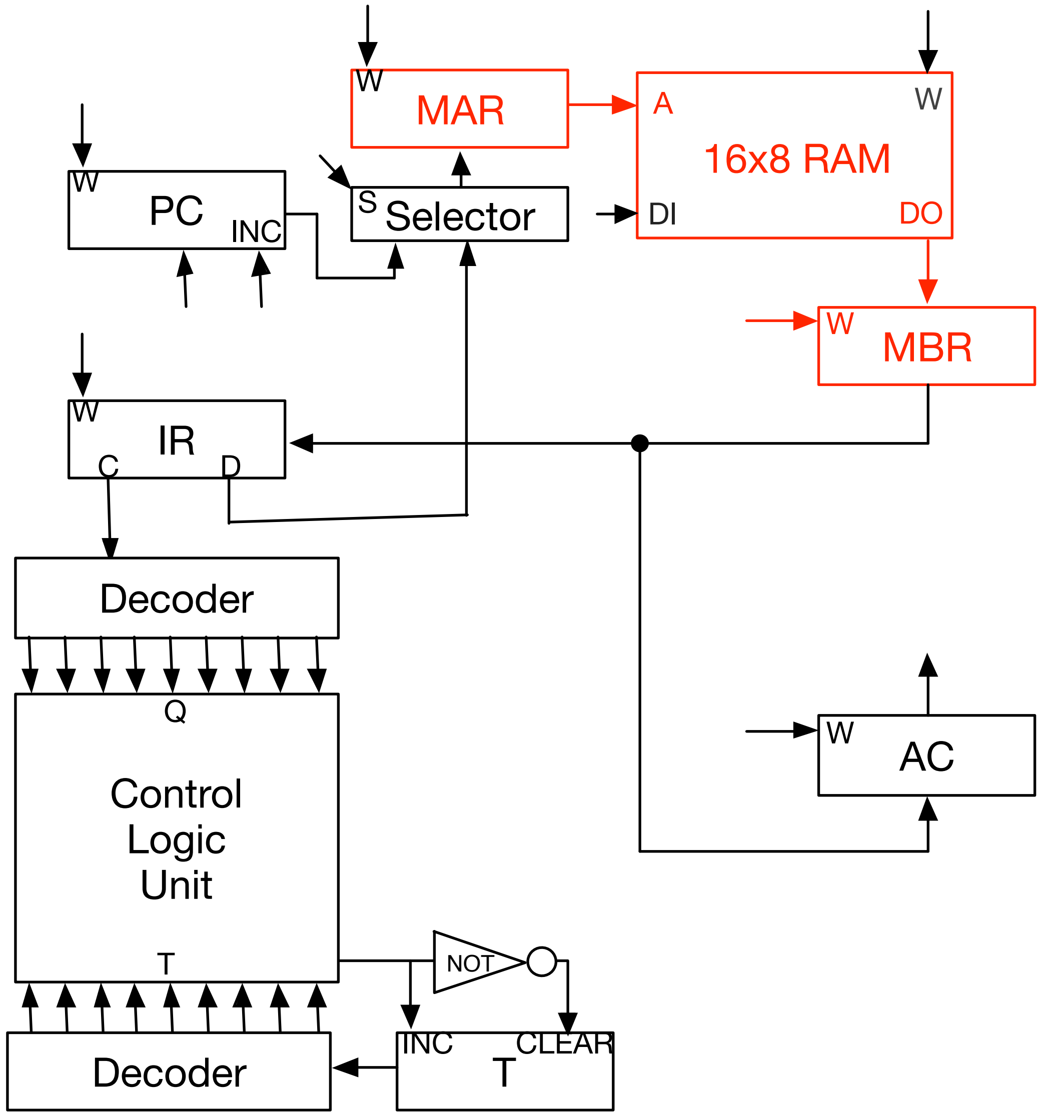
- Load indirectly into accumulator

Op Code	Time	Command
q_2	t_3	MAR $\leftarrow$ IR(D) *
q_2	t_4	MBR $\leftarrow$ M
q_2	t_5	MAR $\leftarrow$ MBR * (low 4 bits)
q_2	t_6	MBR $\leftarrow$ M
q_2	t_7	AC $\leftarrow$ MBR

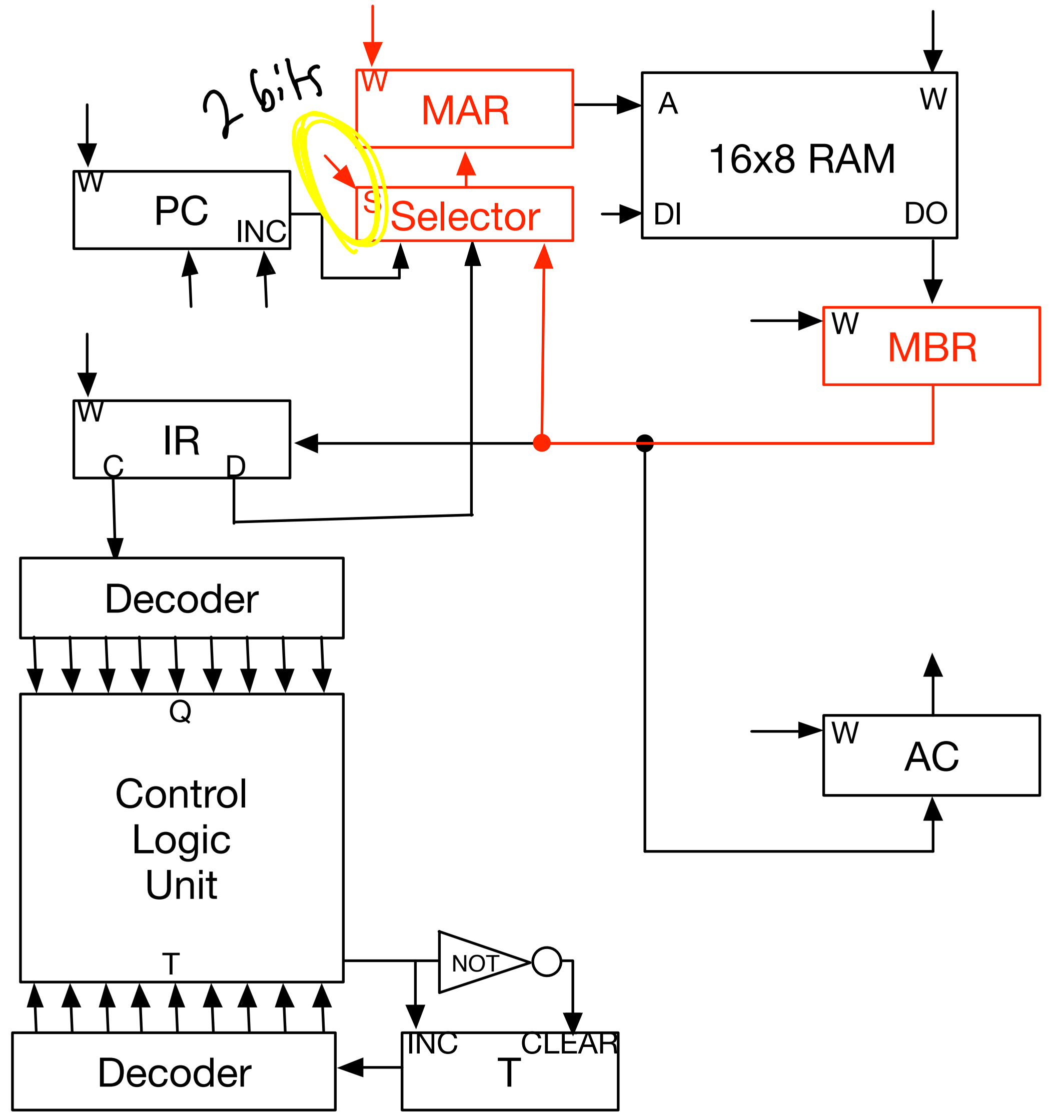
q_2 t_3: MAR $\leftarrow$ IR(D)



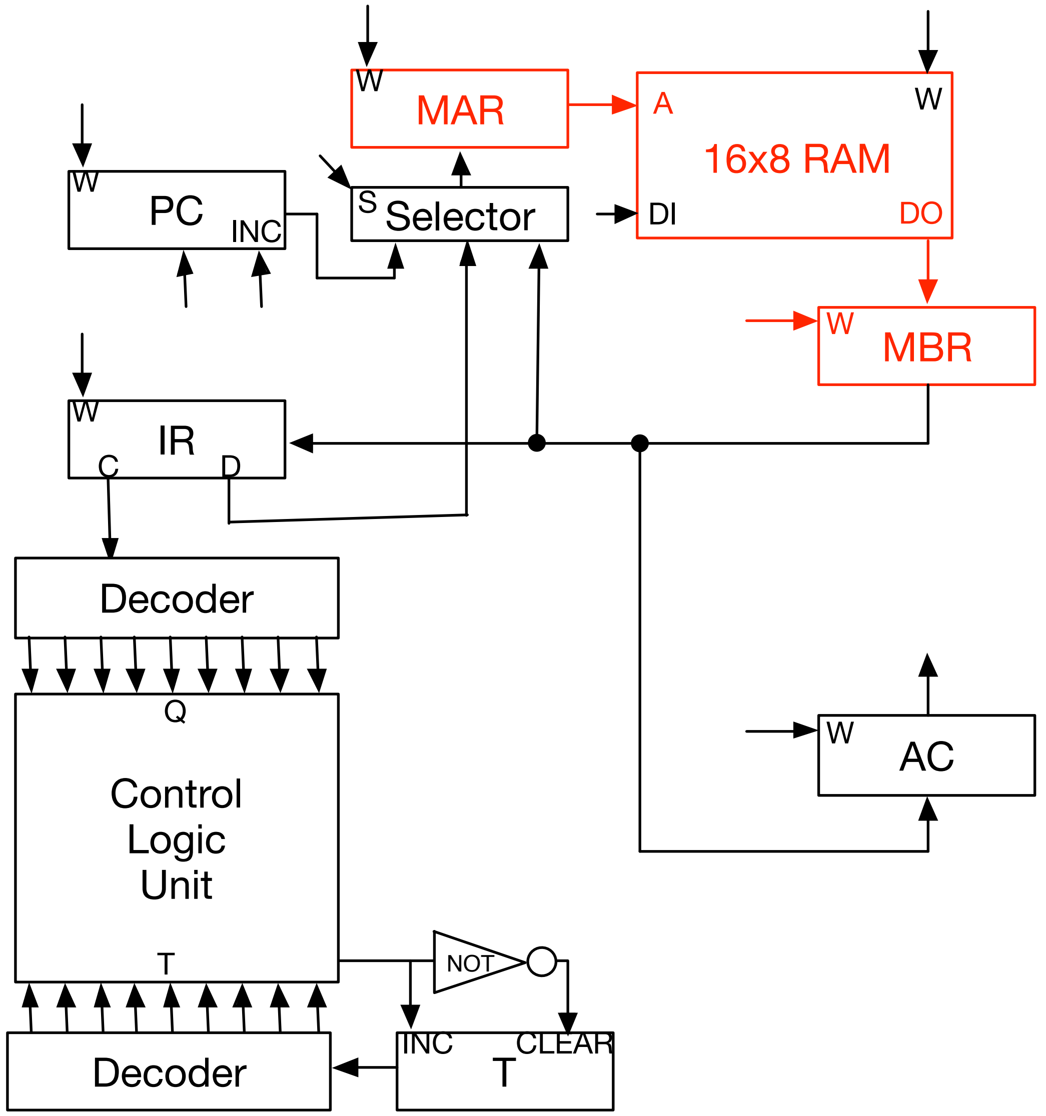
q_2 t_4: **MBR** $\leftarrow$ **M**



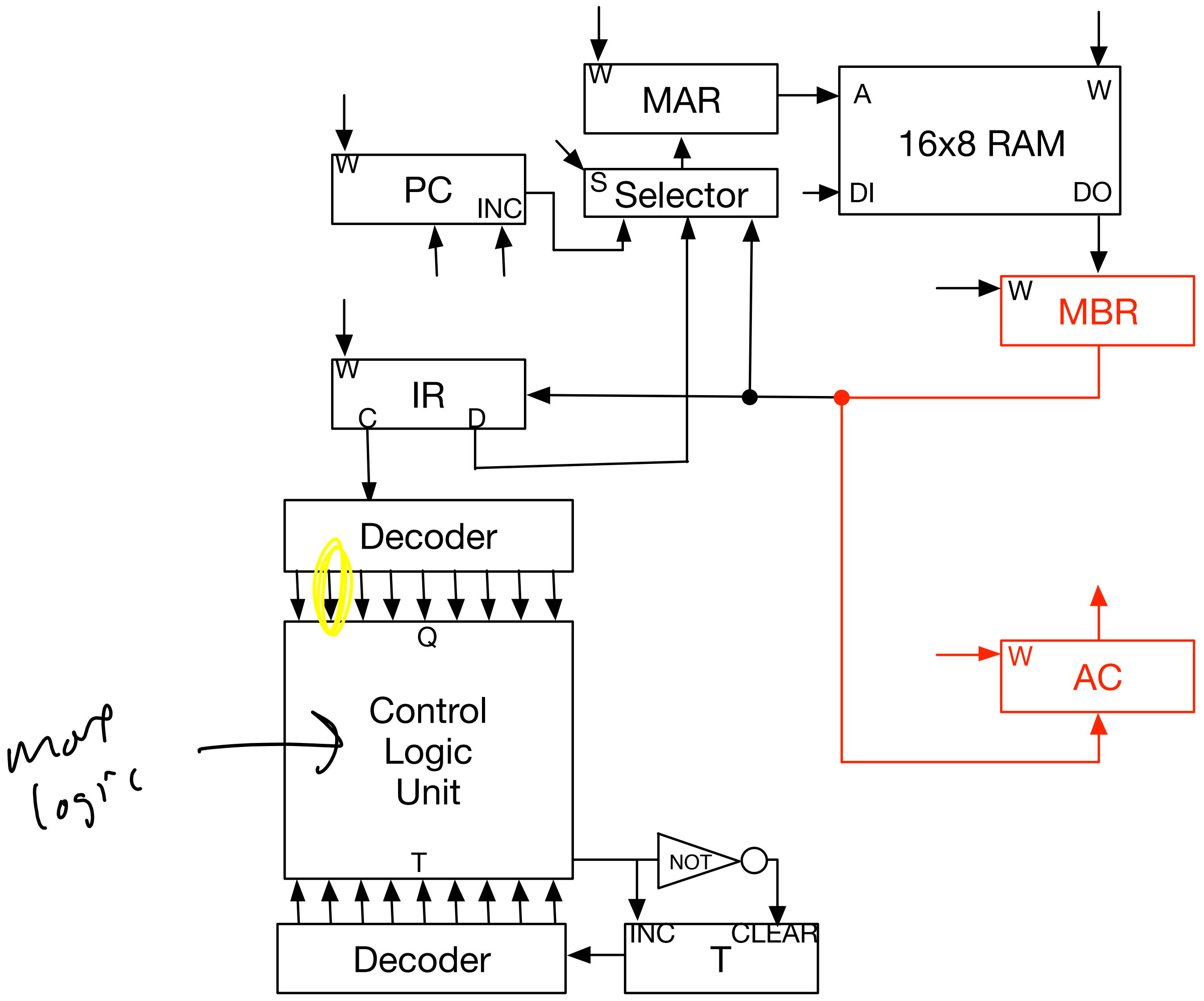
q_2 t_5: **MAR** $\leftarrow$ **MBR**



q_2 t_6: MBR $\leftarrow$ M



q_2 t_7: **AC** $\leftarrow$ **MBR**



`sta`

STA: Store Value from Accumulator

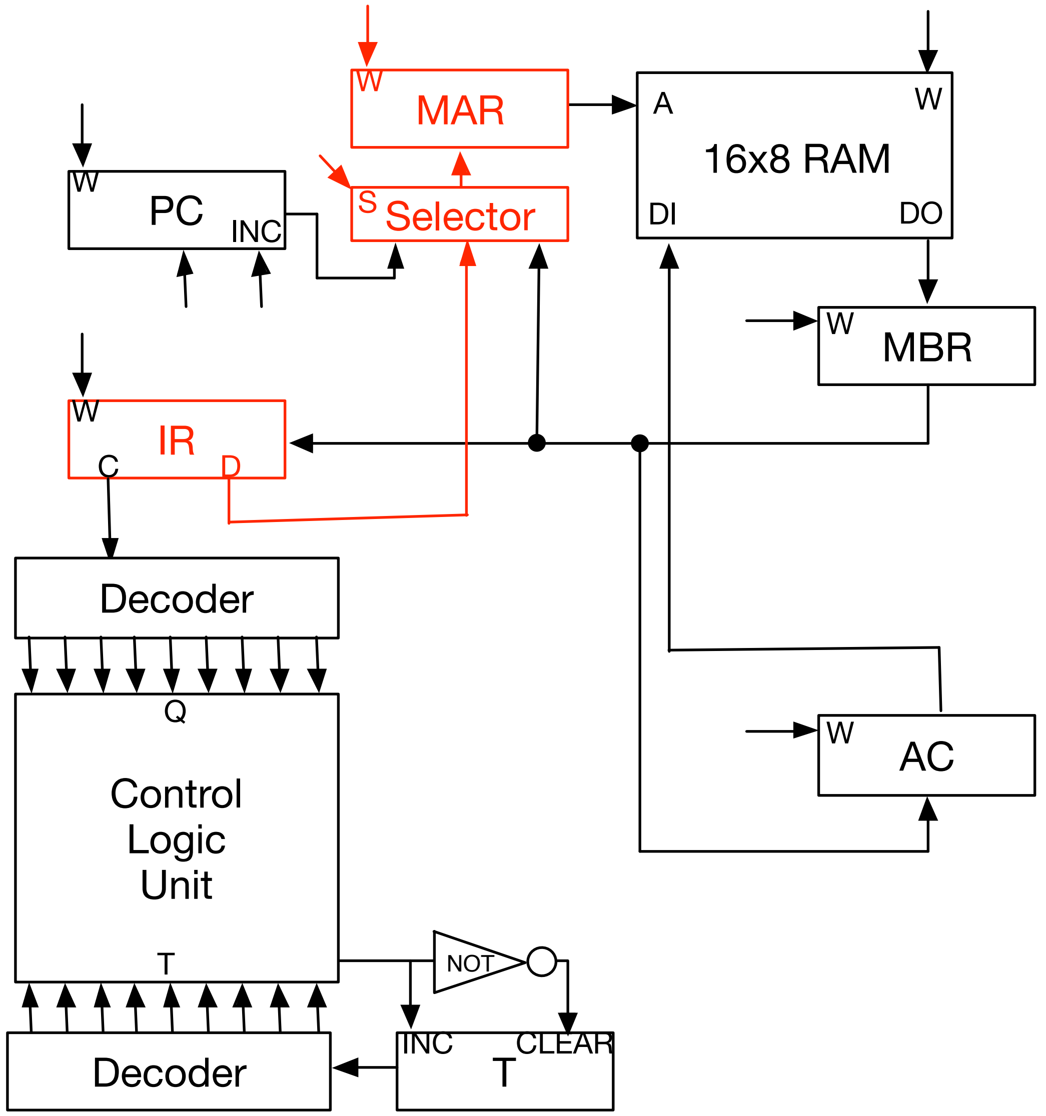
- We now need to write to memory
- Address to be written with comes from instruction
- Value needs to be transferred from accumulator

Micro Program for STA

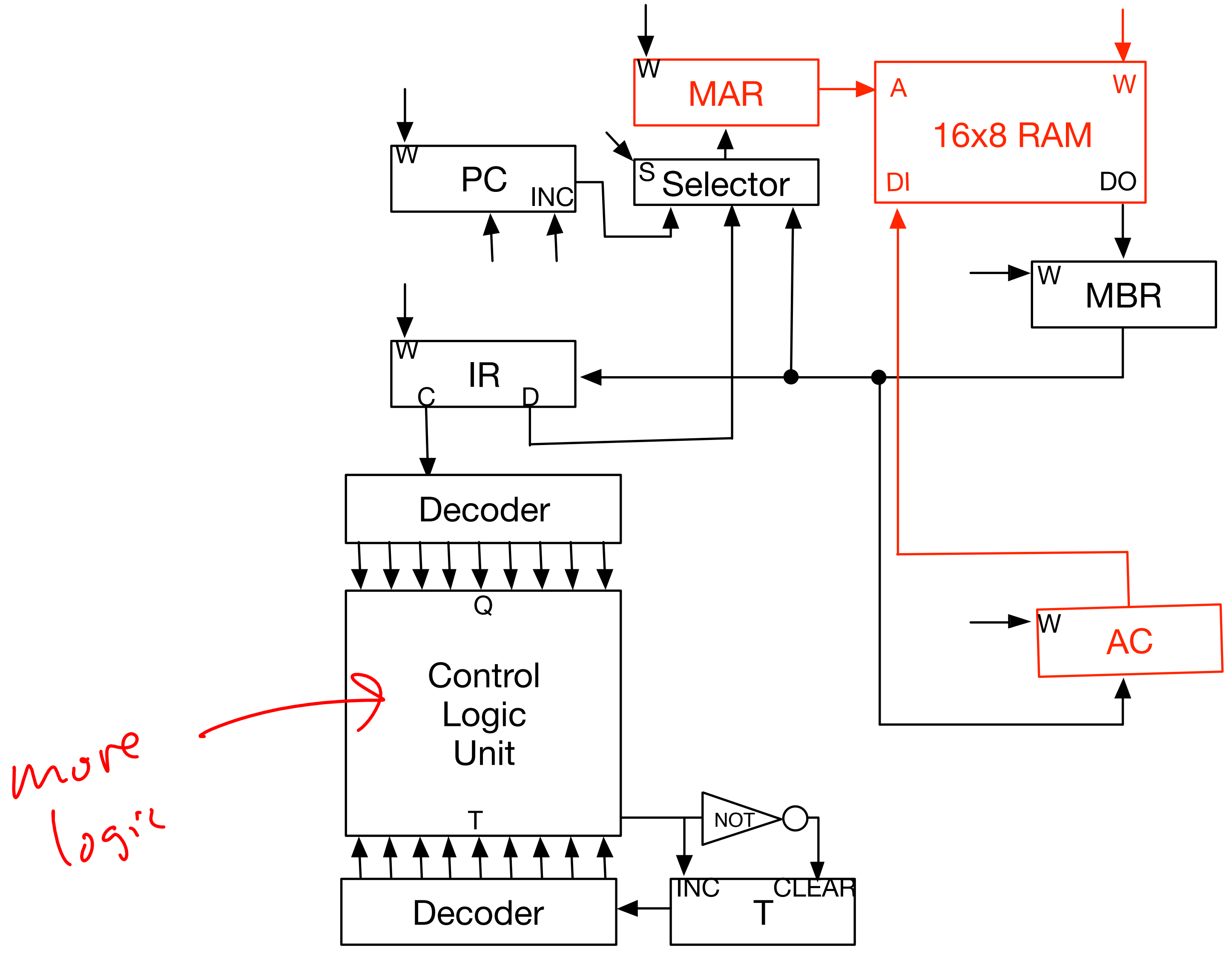
- Store value from accumulator

Op Code	Time	Command
q_3	t_3	MAR $\leftarrow$ IR(D)
q_3	t_4	M $\leftarrow$ AC *

q_3 t_3: MAR $\leftarrow$ IR(D)



$\mathtt{q_3\ t_4:}\quad \mathbf{M} \leftarrow \mathbf{AC}$



`sti`

STI: Store Value Indirectly

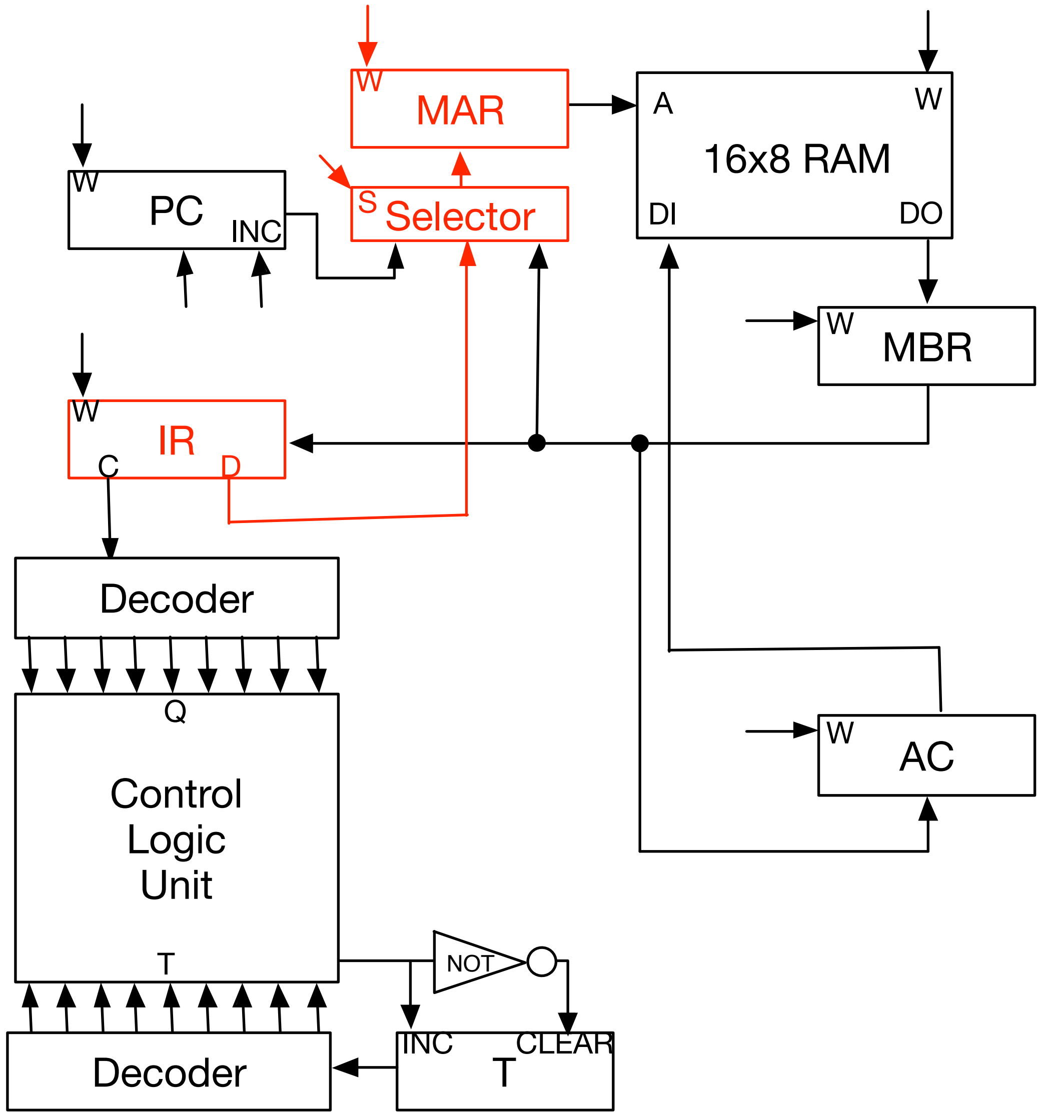
- Specified memory address contains address for value
- Steps
  - load value of specified memory address
  - use that value as a memory address (second lookup)
  - store value from accumulator to that address

Micro Program for STI

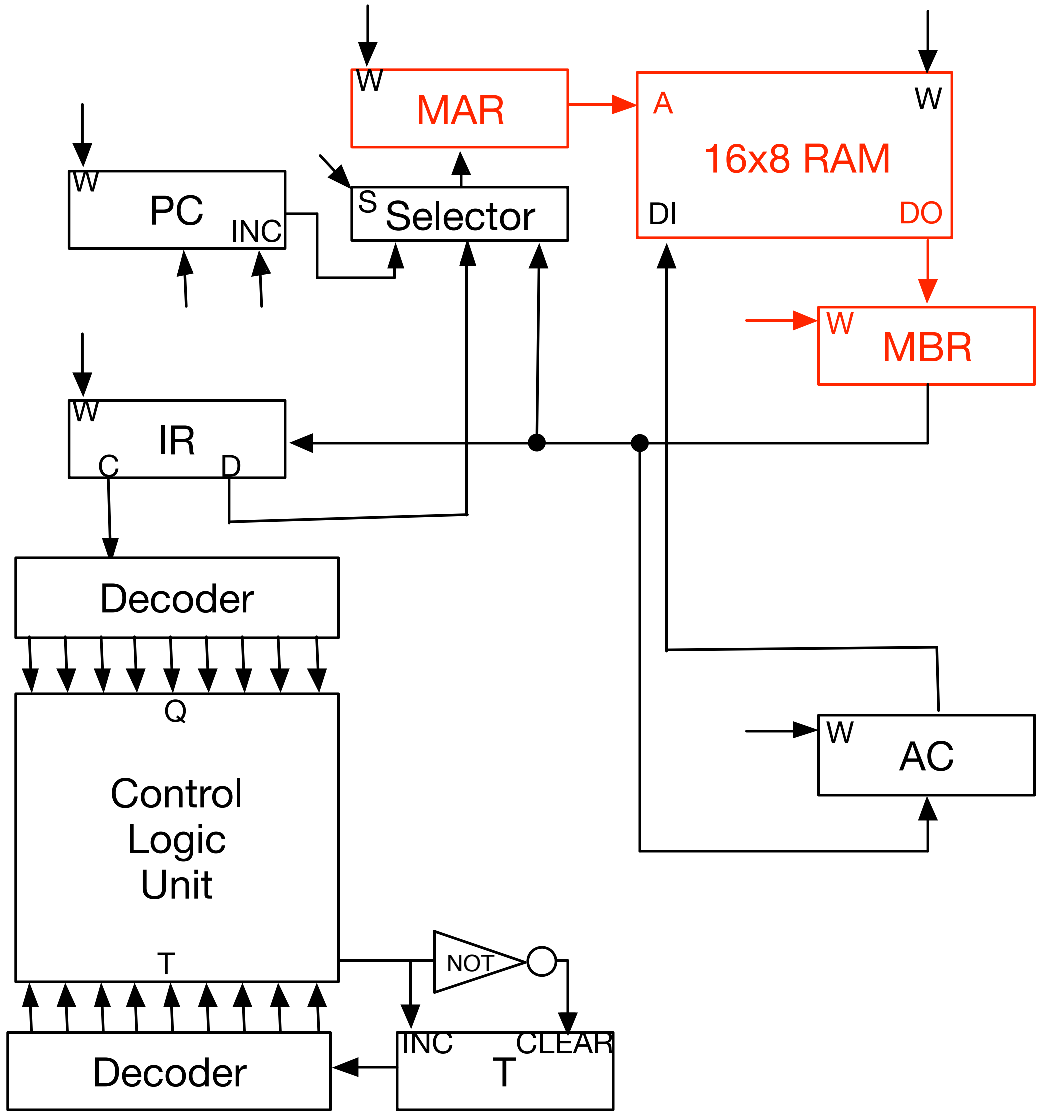
- Store indirectly into accumulator

Op Code	Time	Command
q_4	t_3	MAR $\leftarrow$ IR(D)
q_4	t_4	MBR $\leftarrow$ M
q_4	t_3	MAR $\leftarrow$ MBR *
q_4	t_4	M $\leftarrow$ AC

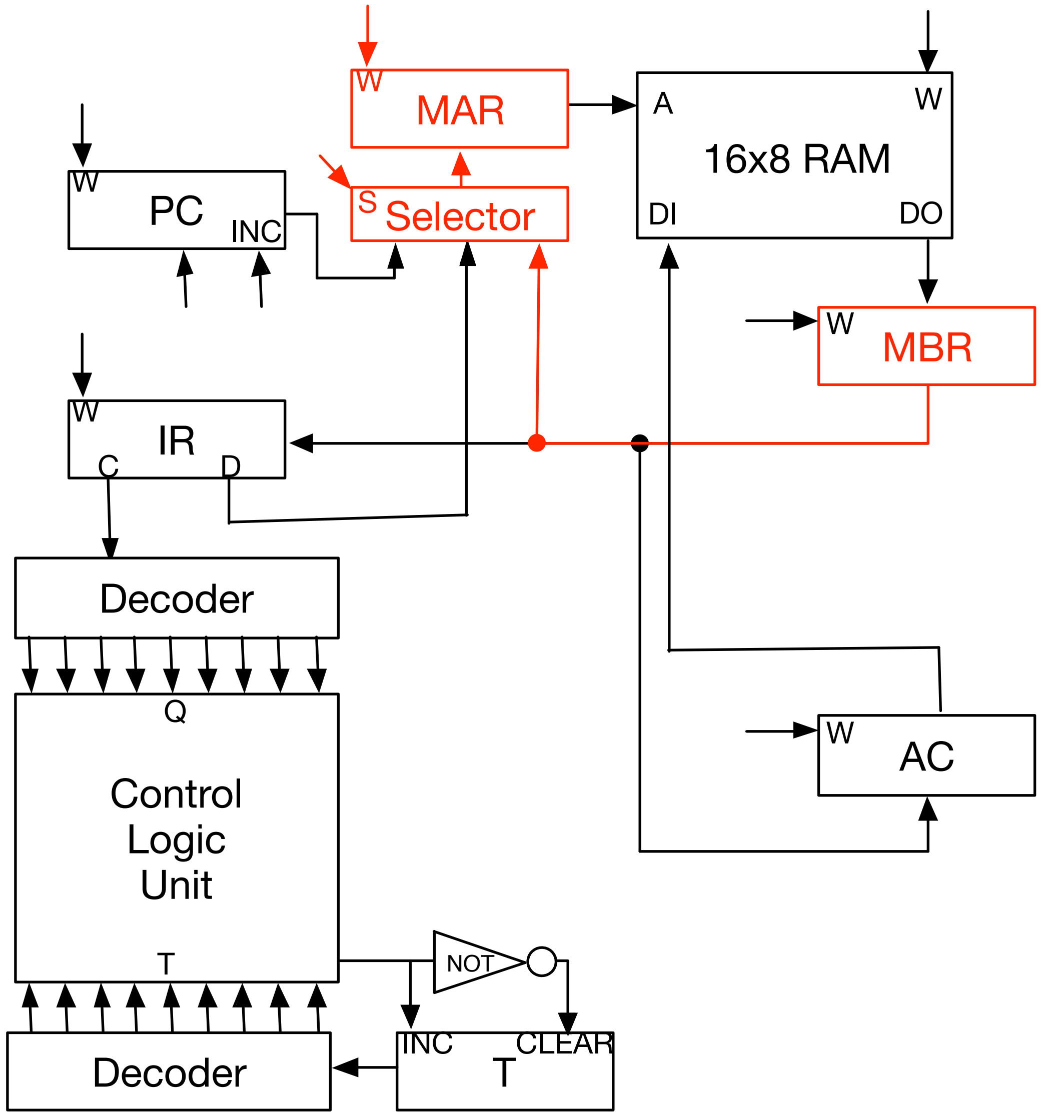
q_4 t_3: **MAR** $\leftarrow$ **IR(D)**



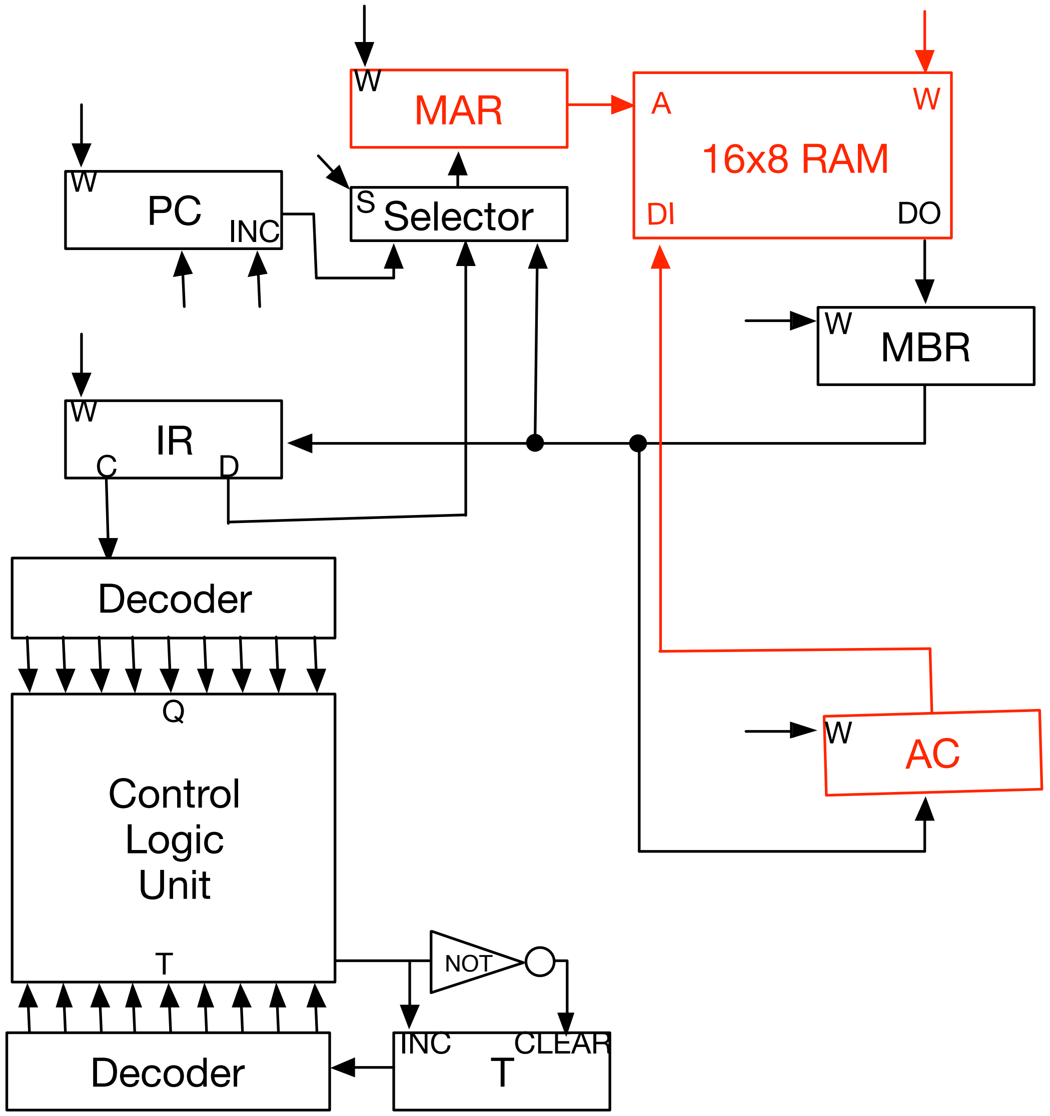
q_4 t_4: **MBR** $\leftarrow$ **M**



q_4 t_5: MAR $\leftarrow$ MBR

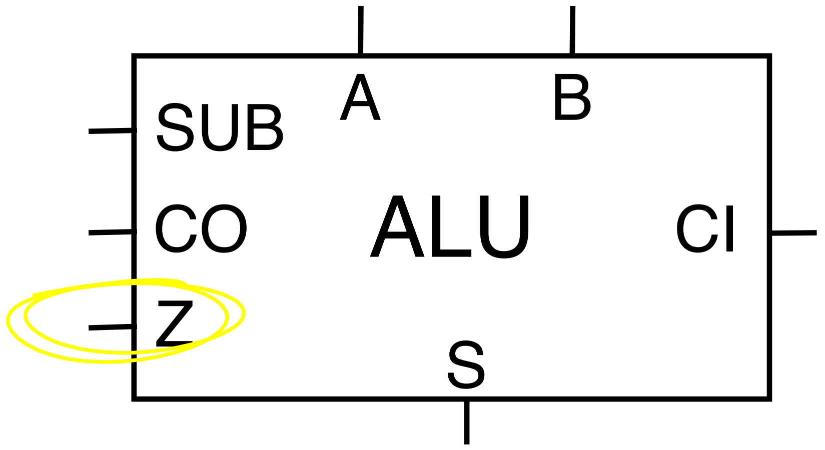


$q_4\ t_6$: $\mathbf{M} \leftarrow \mathbf{AC}$



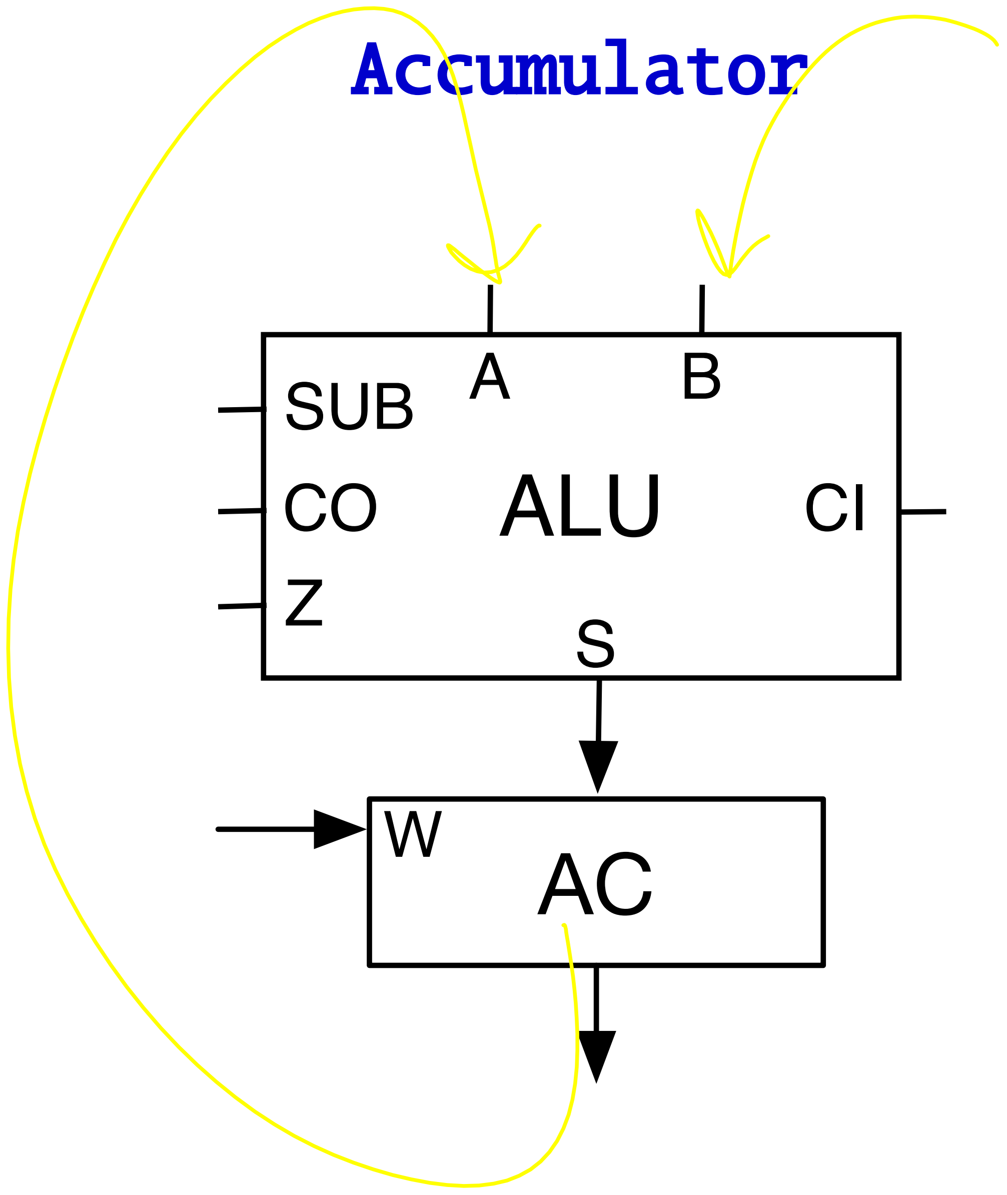
arithmetic logic unit

Arithmetic Logic Unit



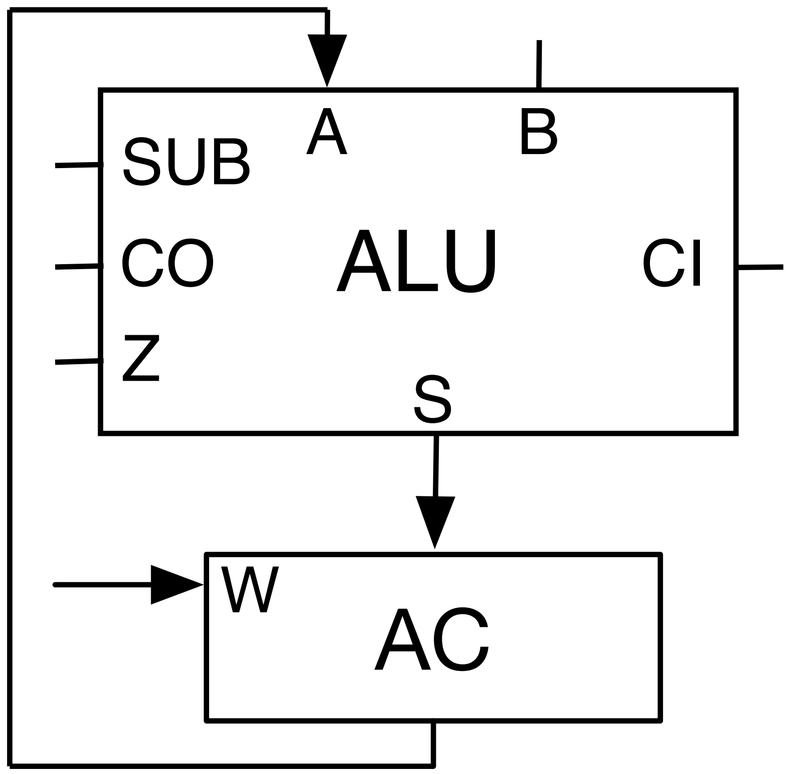
- Adds two numbers: S=A+B
- With subtraction flag: S=A-B
- Overflow handling with carry in (CI) and carry out (CO)
- Zero flag: set if result of operation is 0

Accumulator



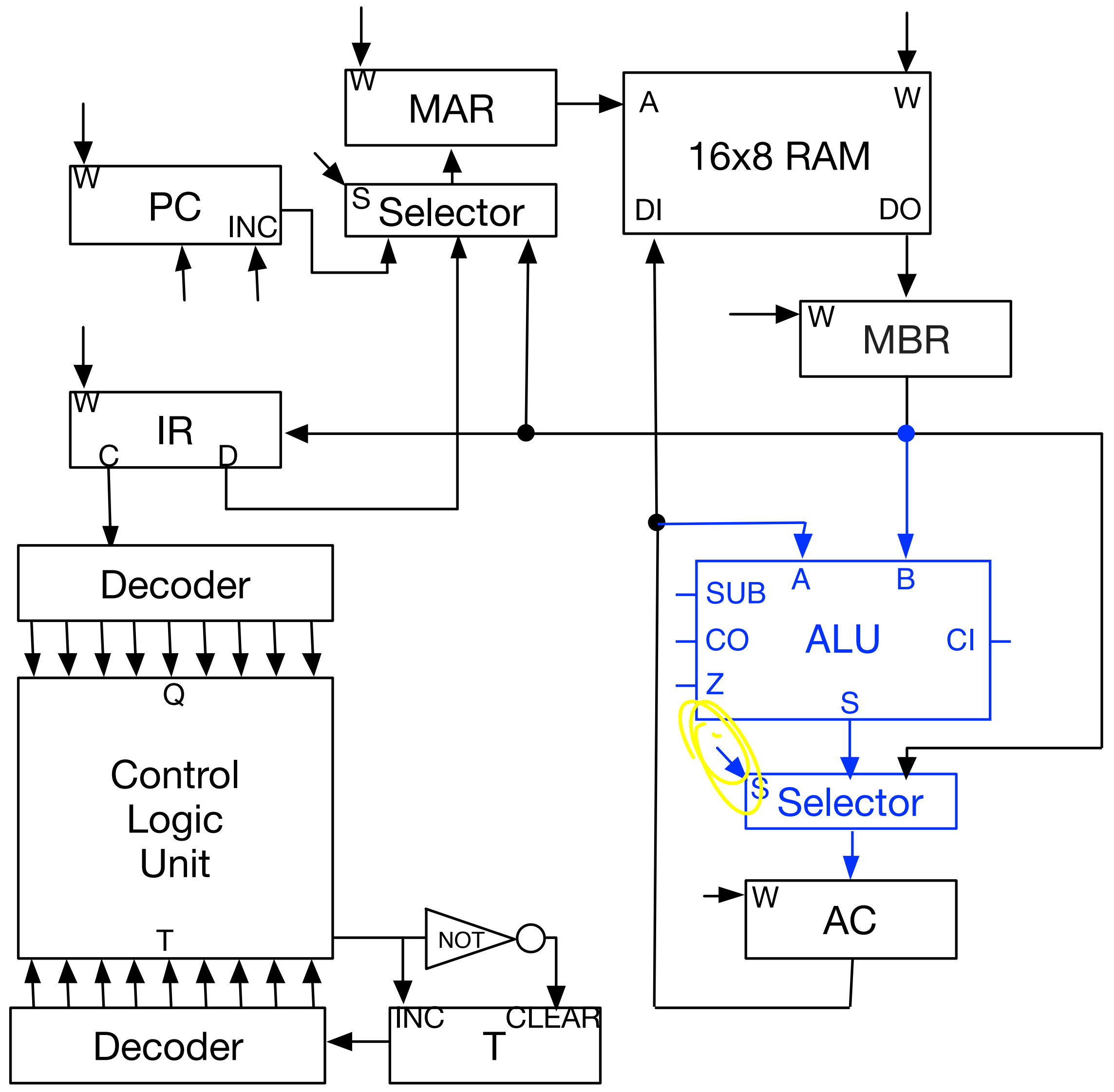
- Store result of ALU operation in accumulator (AC)

AC = AC ± B



- Accumulator feeds back into ALU
- Operations are AC = AC + B or AC = AC - B

ALU in Circuit



`add`

ADD: Add to Accumulator

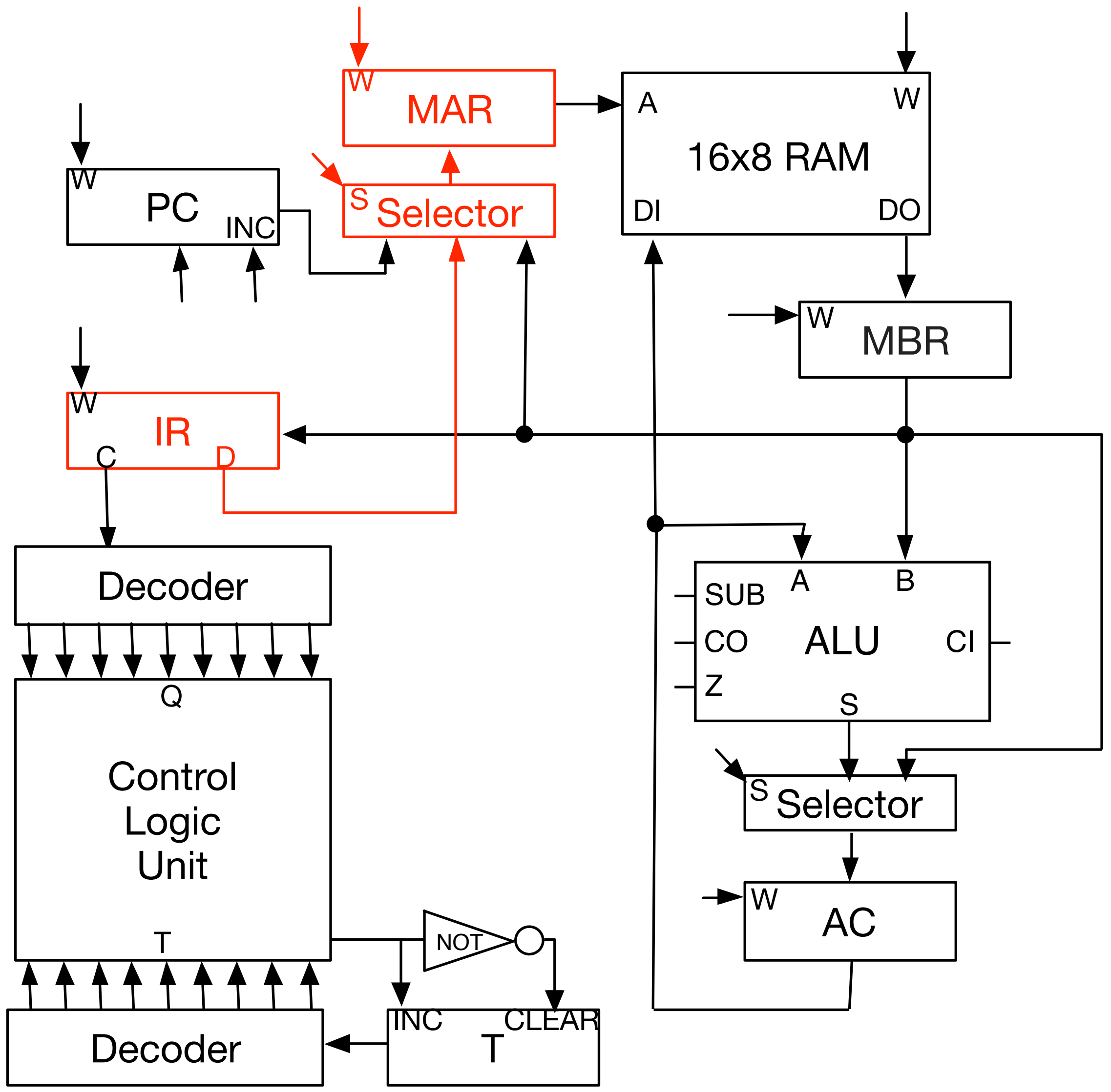
- Add value from memory address to accumulator
- Steps
  - load value of specified memory address
  - use that value as a memory address (second lookup)
  - store value from second lookup into accumulator

Micro Program for ADD

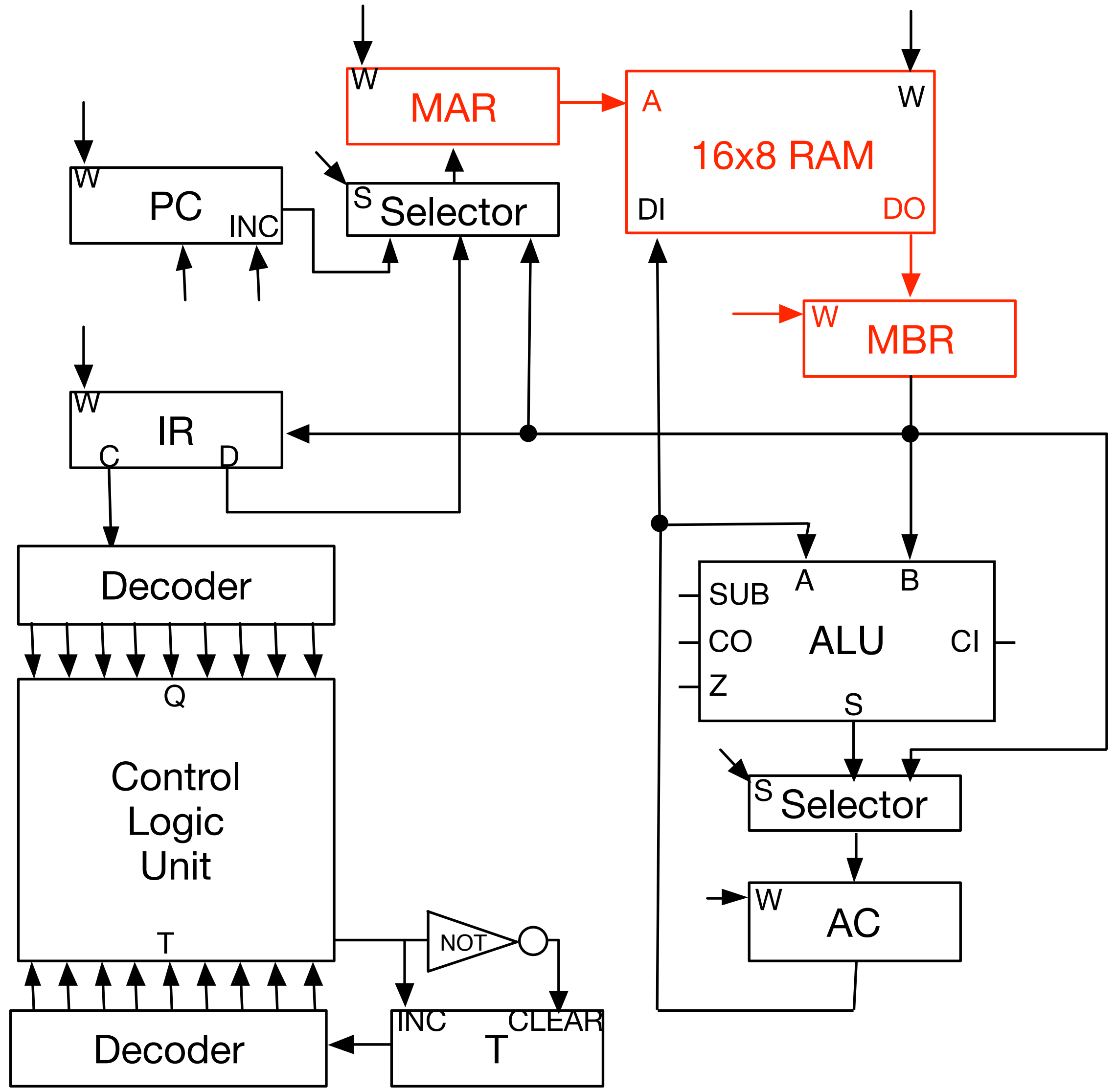
- Load indirectly into accumulator

Op Code	Time	Command
q_5	t_3	MAR $\leftarrow$ IR(D)
q_5	t_4	MBR $\leftarrow$ M
q_5	t_5	AC $\leftarrow$ AC + MBR *

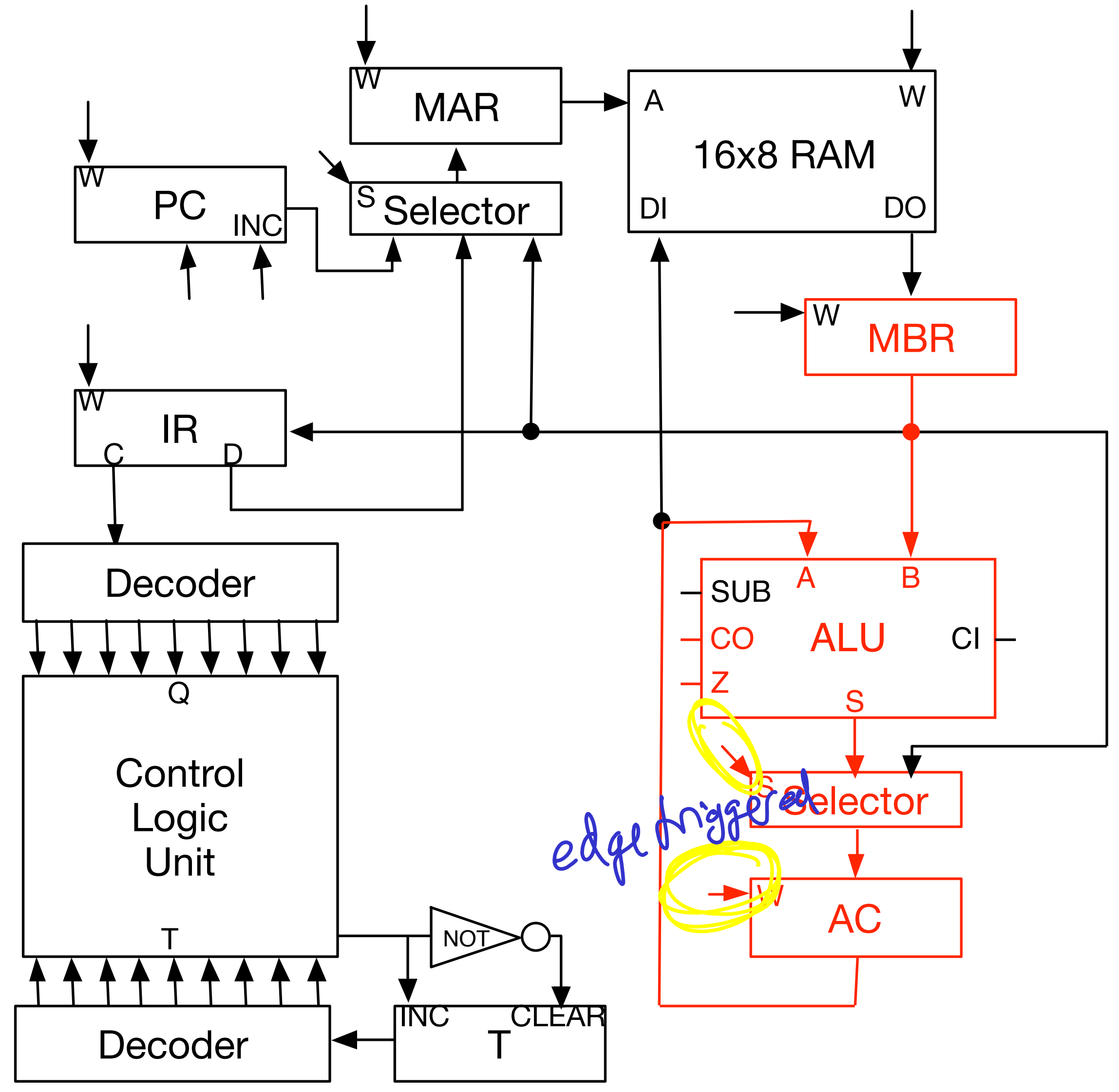
q_5 t_3: MAR $\leftarrow$ IR(D)



q_5 t_4: MBR $\leftarrow$ M



q_5 t_5: AC ← AC + MBR



sub

SUB: Subtract from Accumulator

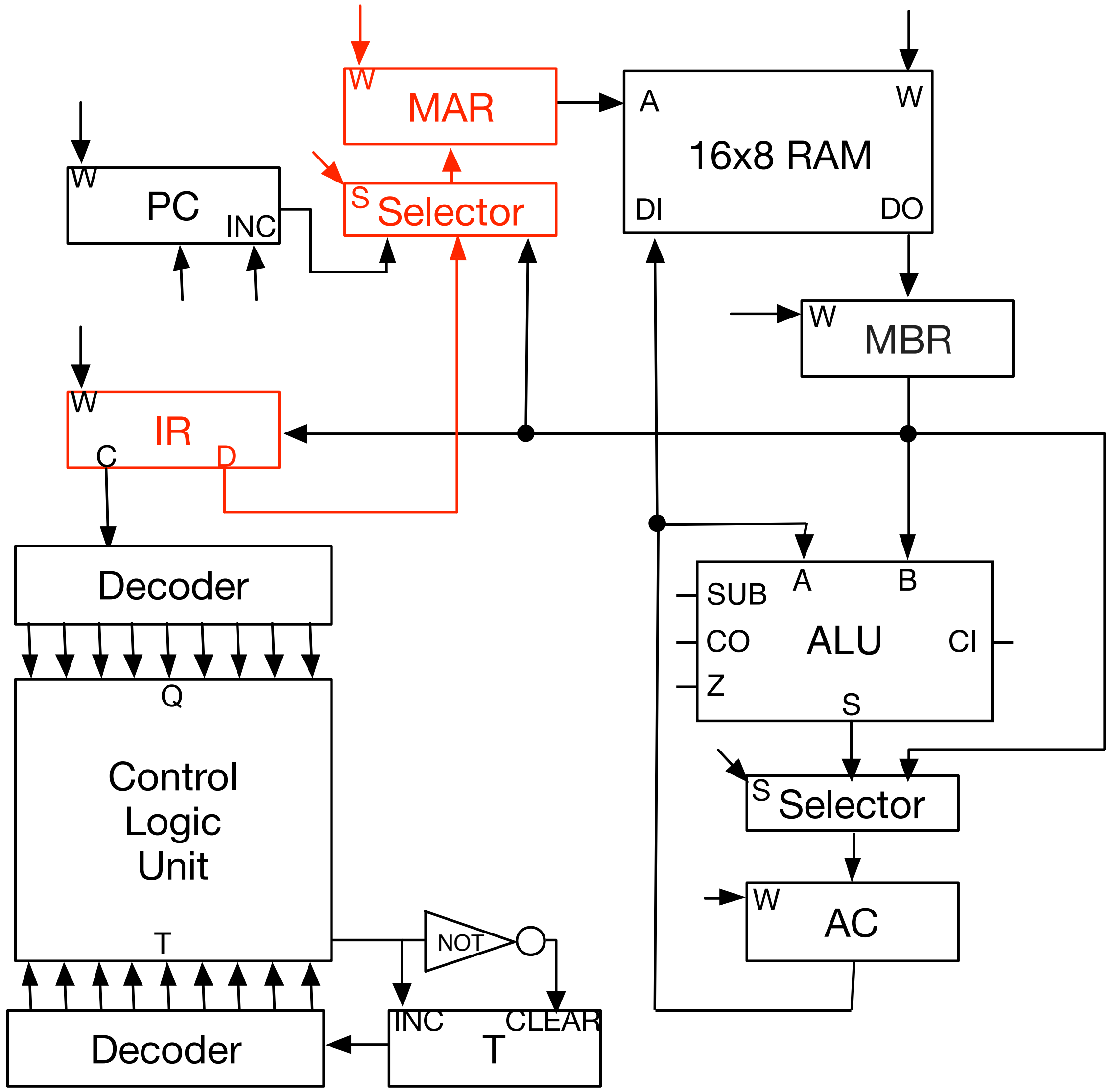
- Subtract from accumulator the value from memory
- Same as ADD, just set subtraction flag of ALU

Micro Program for SUB

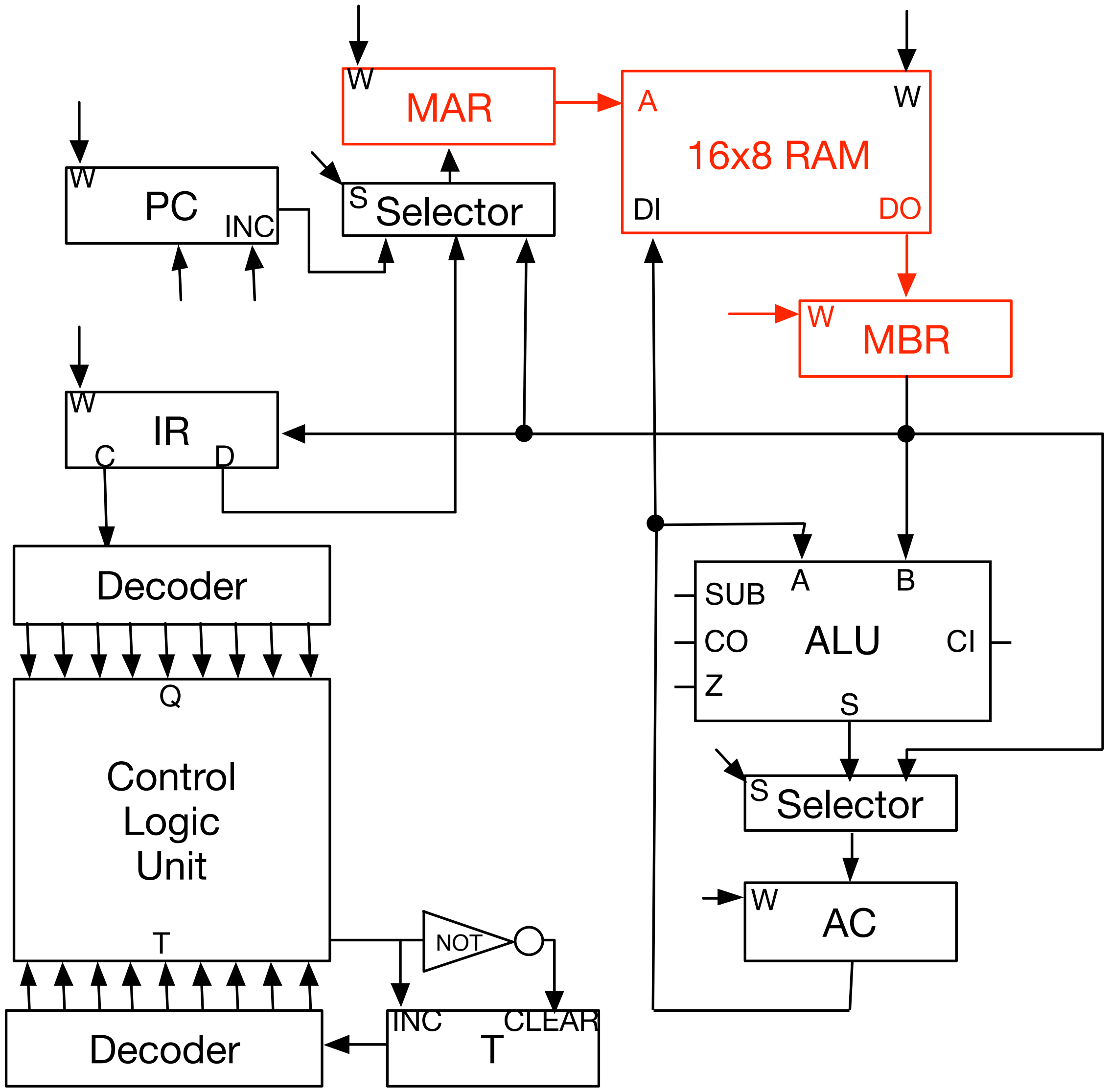
- Load indirectly into accumulator

Op Code	Time	Command
q_5	t_3	MAR $\leftarrow$ IR(D)
q_5	t_4	MBR $\leftarrow$ M
q_5	t_5	AC $\leftarrow$ AC - MBR

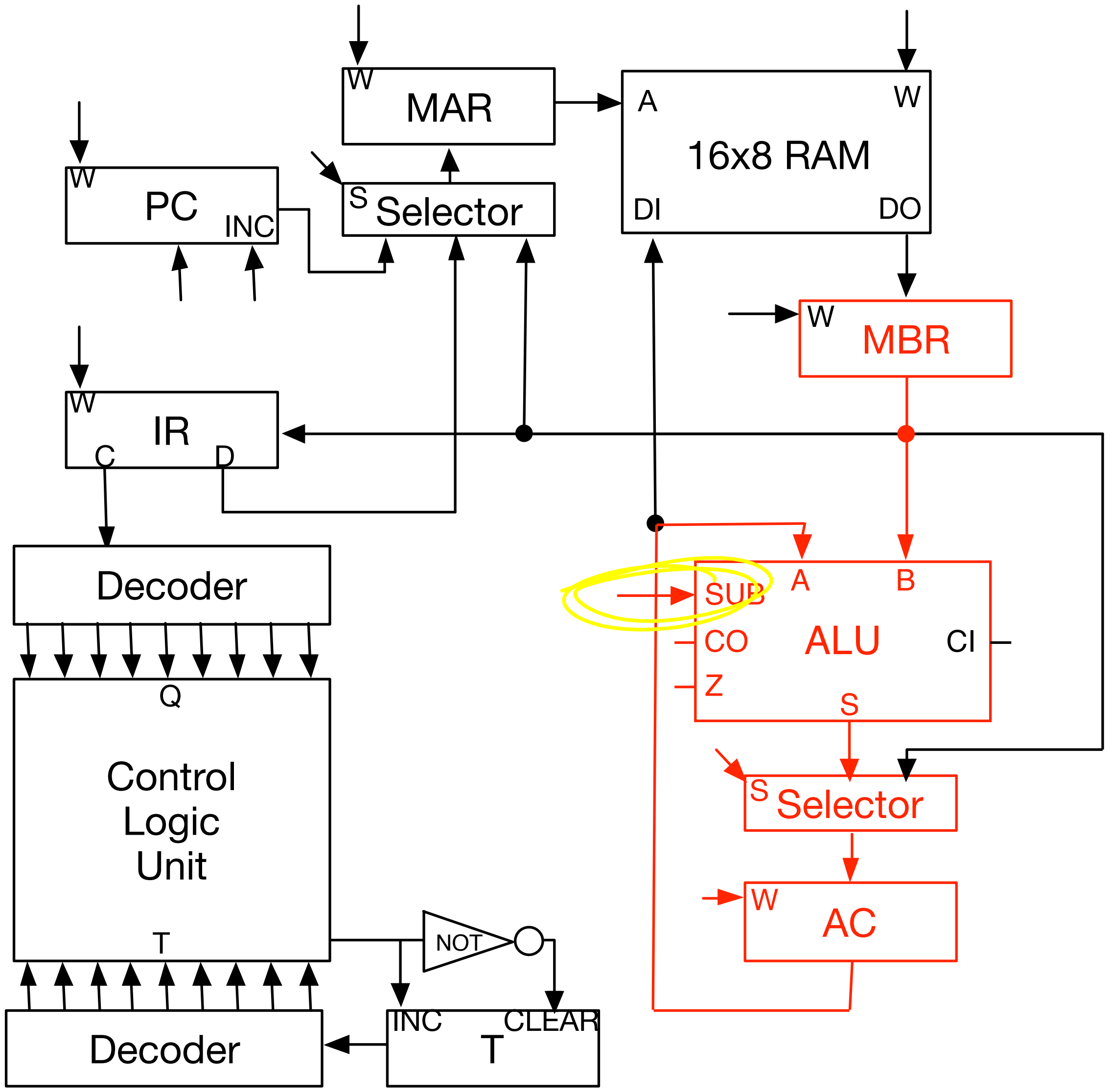
q_5 t_3: **MAR** $\leftarrow$ **IR(D)**



q_5 t_4: MBR $\leftarrow$ M



q_5 t_5: AC $\leftarrow$ AC ~~+~~ − MBR



`jmp`

Program Counter (PC)

- Position of the next instruction is stored in program counter
- This gets updated during instruction fetch

Time	**Command**
t_0	MAR $\leftarrow$ PC
t_1	MBR $\leftarrow$ M , PC $\leftarrow$ PC + 1
t_2	IR $\leftarrow$ MBR
~~t_3~~	~~PC $\leftarrow$ PC + 1~~

JMP: Jump

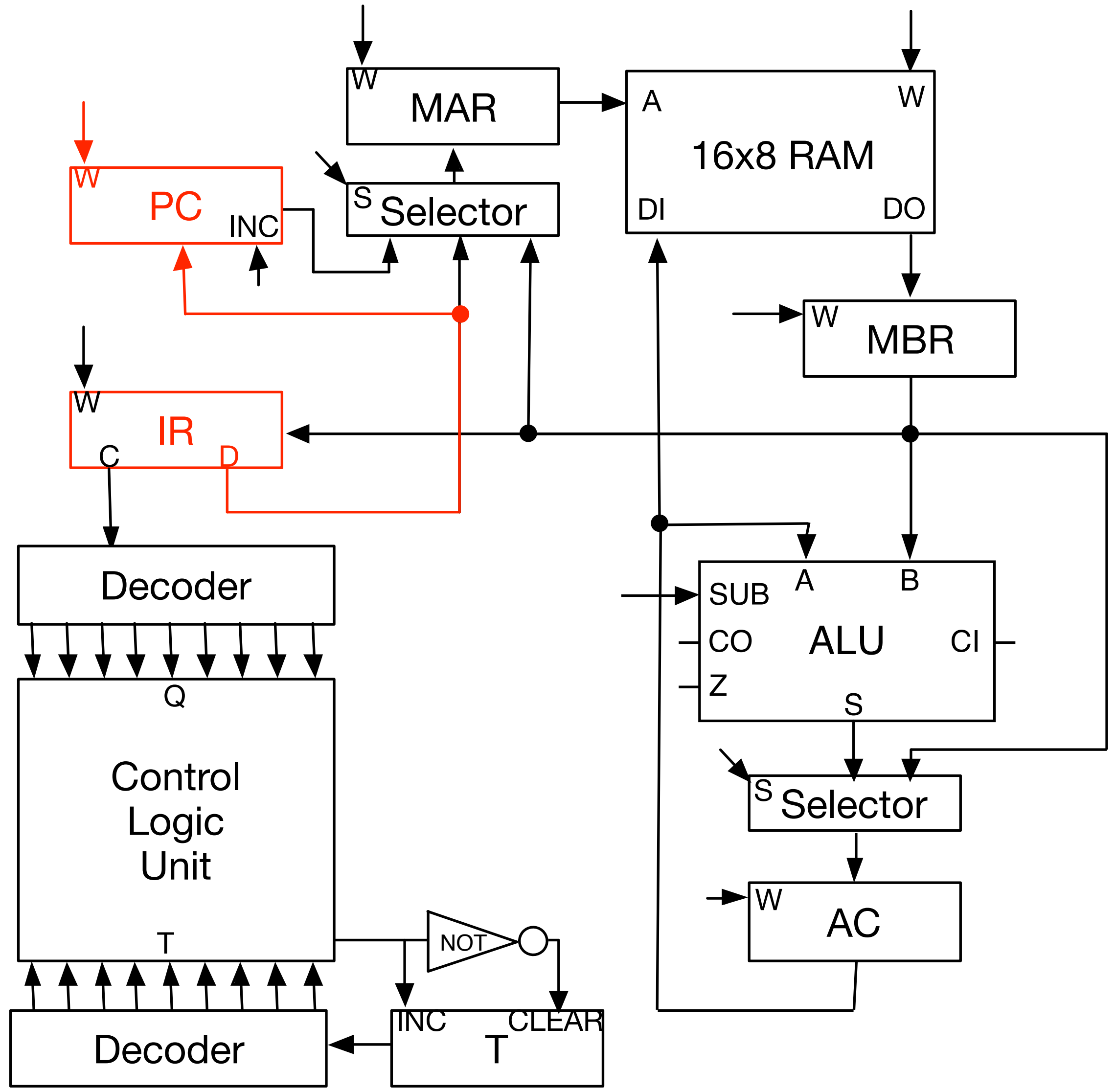
- Assign value to position of the next instruction
- Sequencing of micro program
  - instruction fetch (includes program counter inc)
  - command-specific micro instructions
- No problem that program counter gets modified twice

Micro Program for JMP

- Change program counter to specified address

Op Code	**Time**	**Command**
q_7	t_3	PC $\leftarrow$ IR(D)

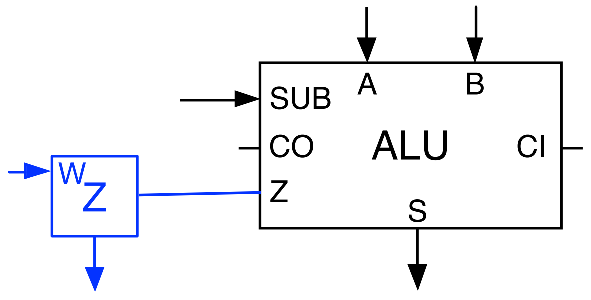
q_7 t_3: PC $\leftarrow$ IR(D)



jpz

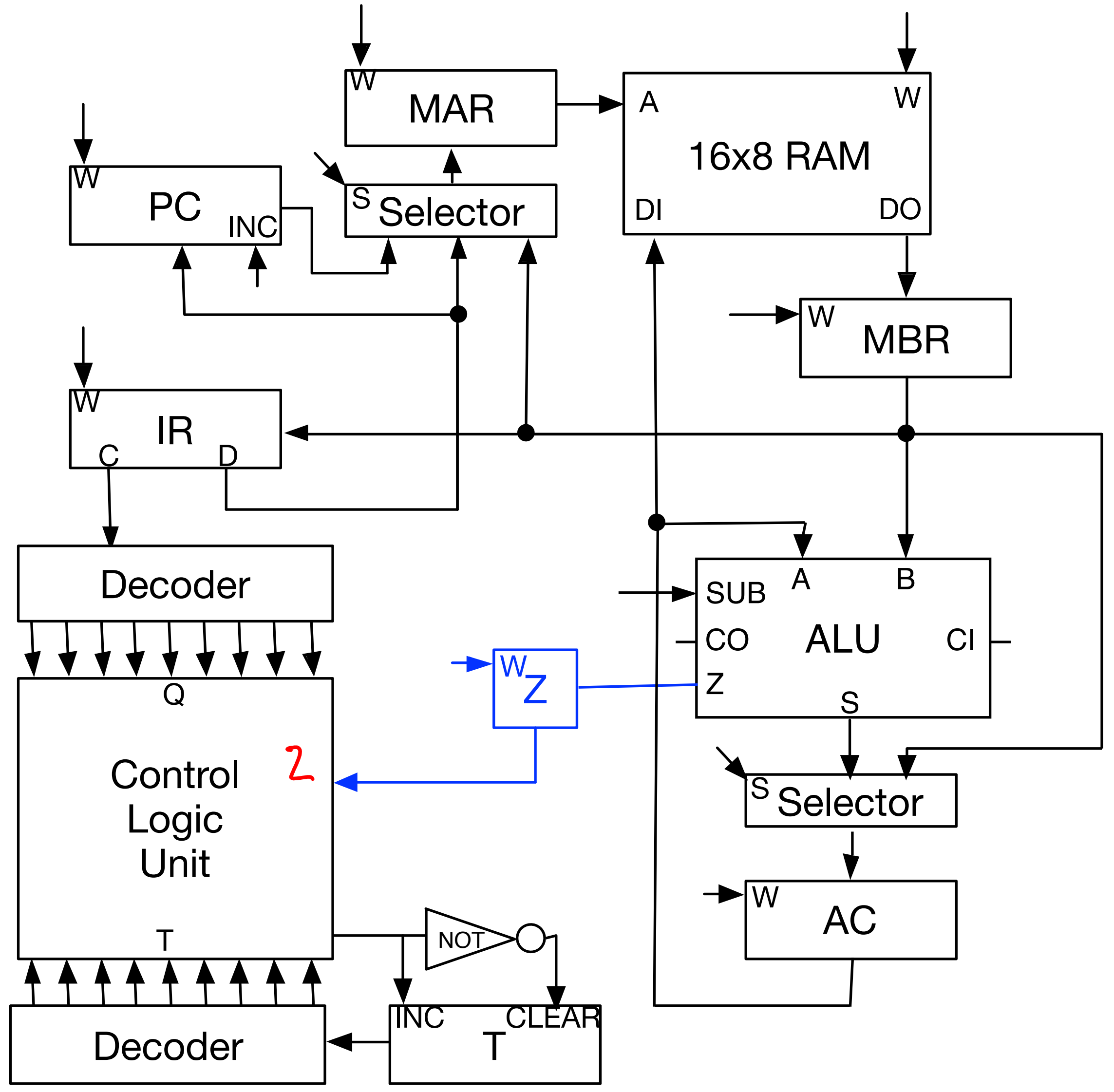
Zero Flag

- Zero flag
  - set when result of a ALU operation is 0
  - stored in flag



condition code

Z Flag in Circuit



Micro Program for JPZ

- Z flag is a condition for executing a micro program (same as JMP)

Zero	Op Code	Time	Command
$\boxed{1}$	q_7	t_3	PC $\leftarrow$ IR(D)

conditional
execution

- If not set, no micro program is executed