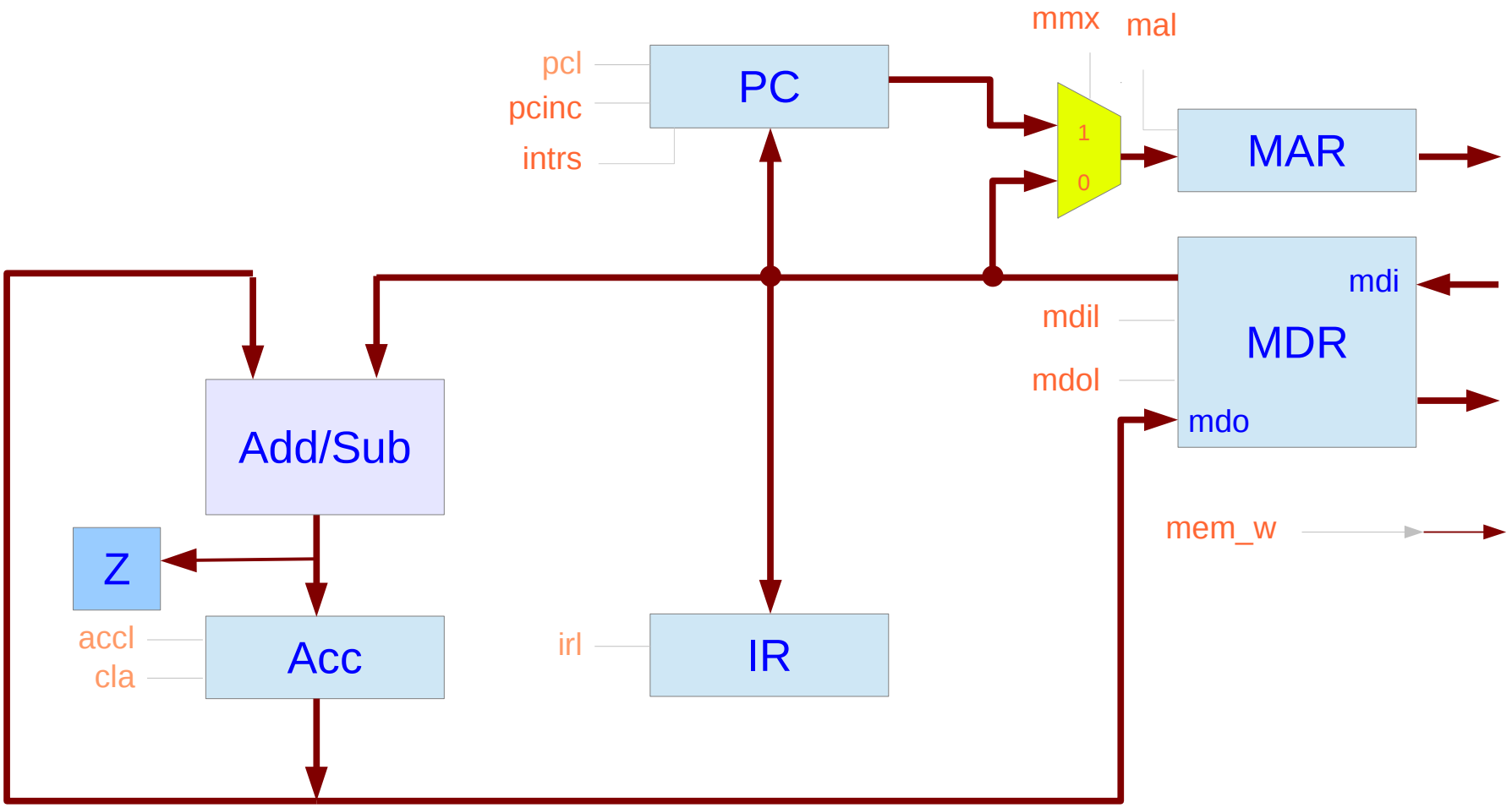
Tiny CPU – ISA (2A)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

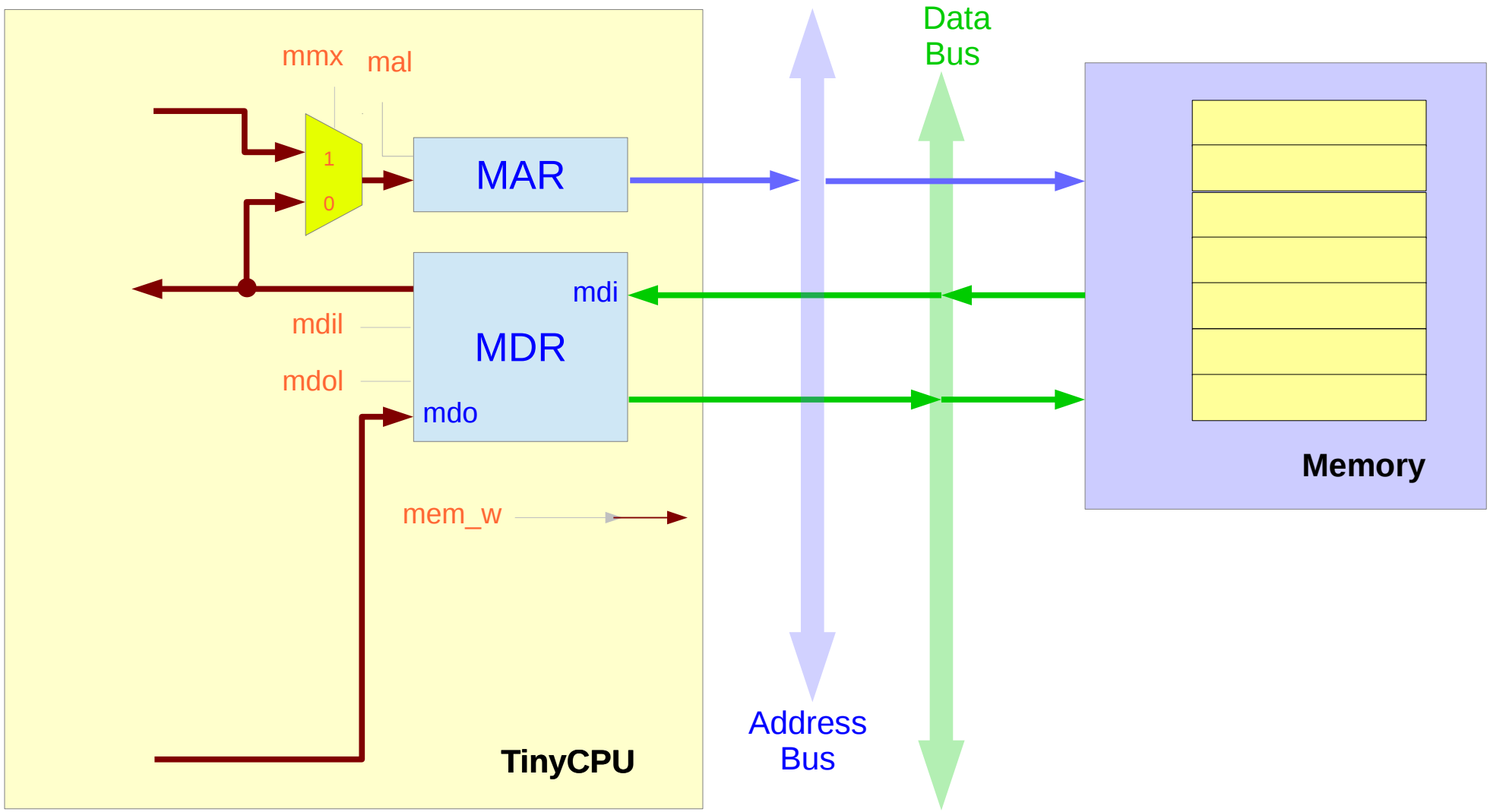
This document was produced by using LibreOffice and Octave.

Data Path



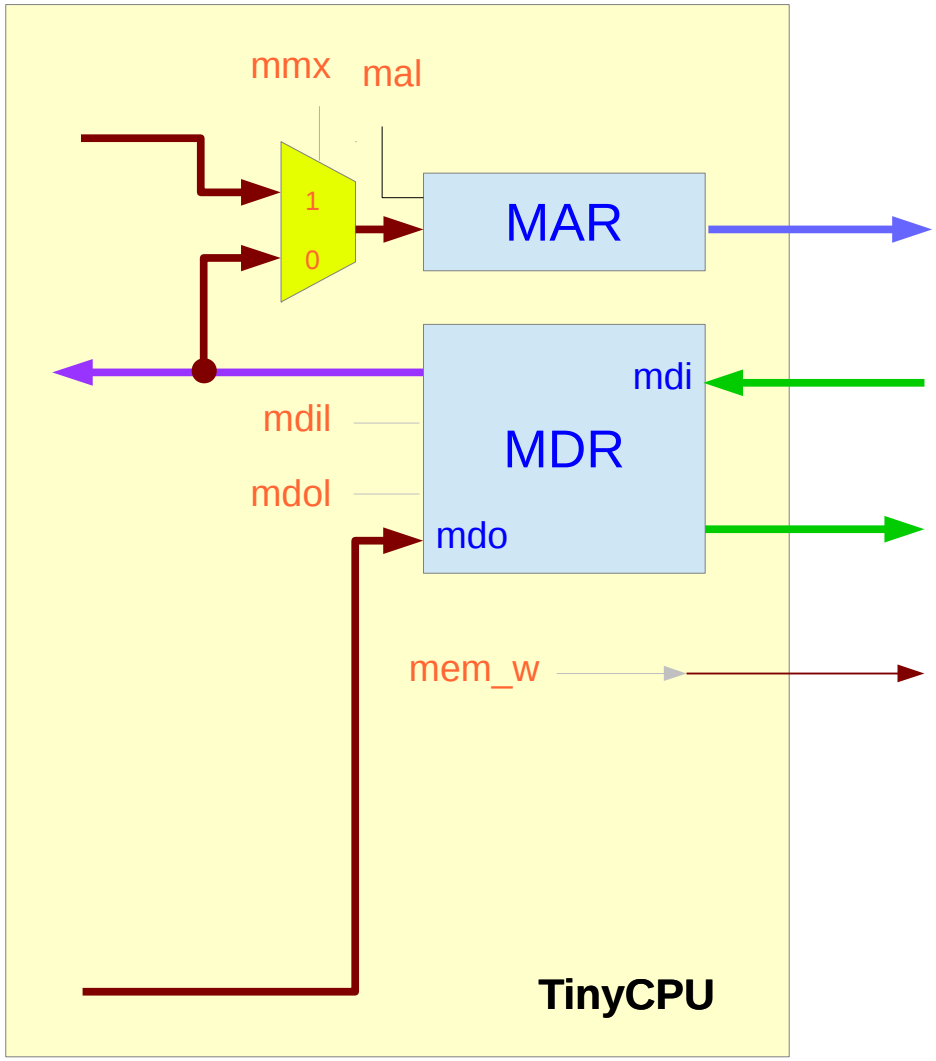
Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

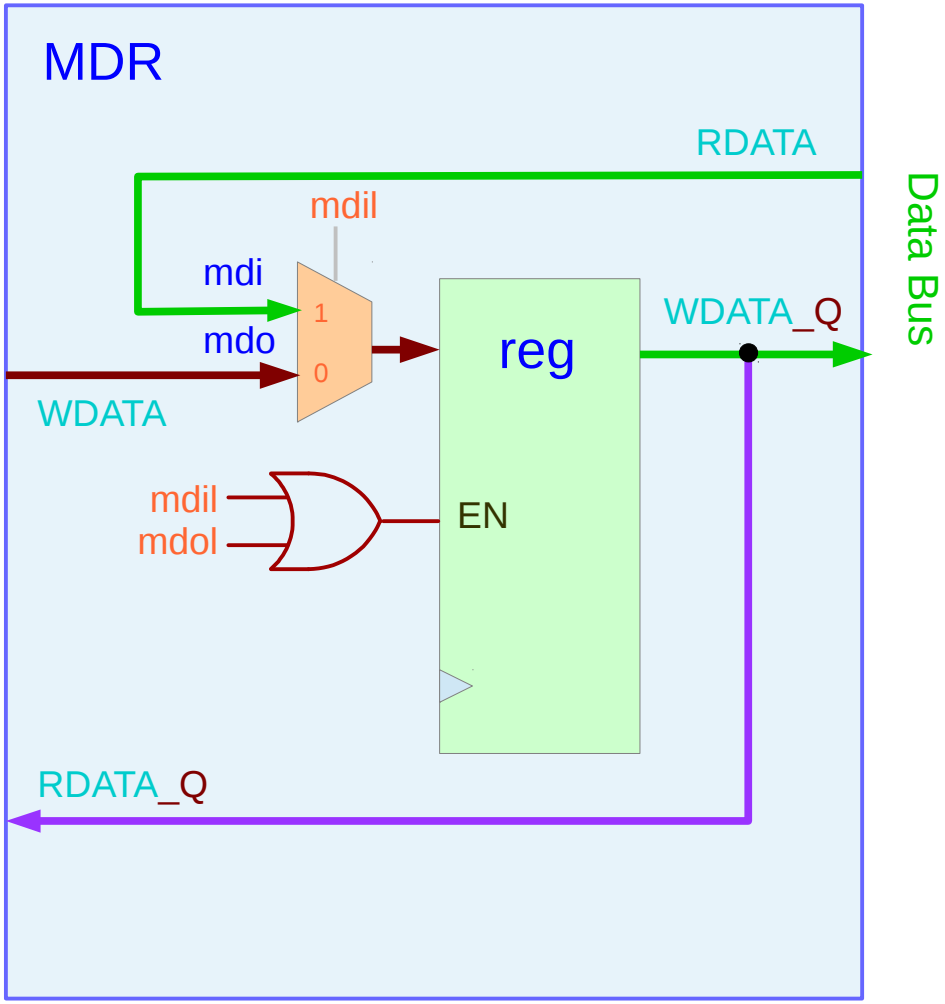
CPU and MEM



Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

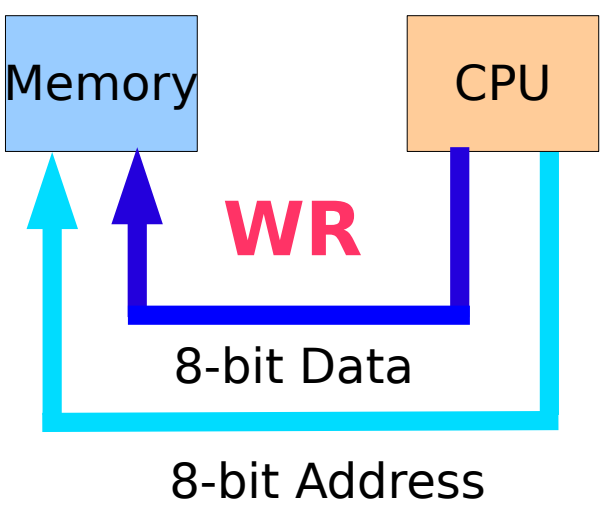
Memory Data Register

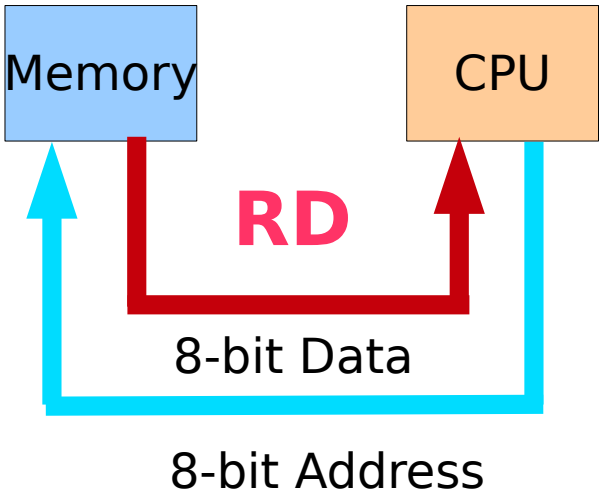




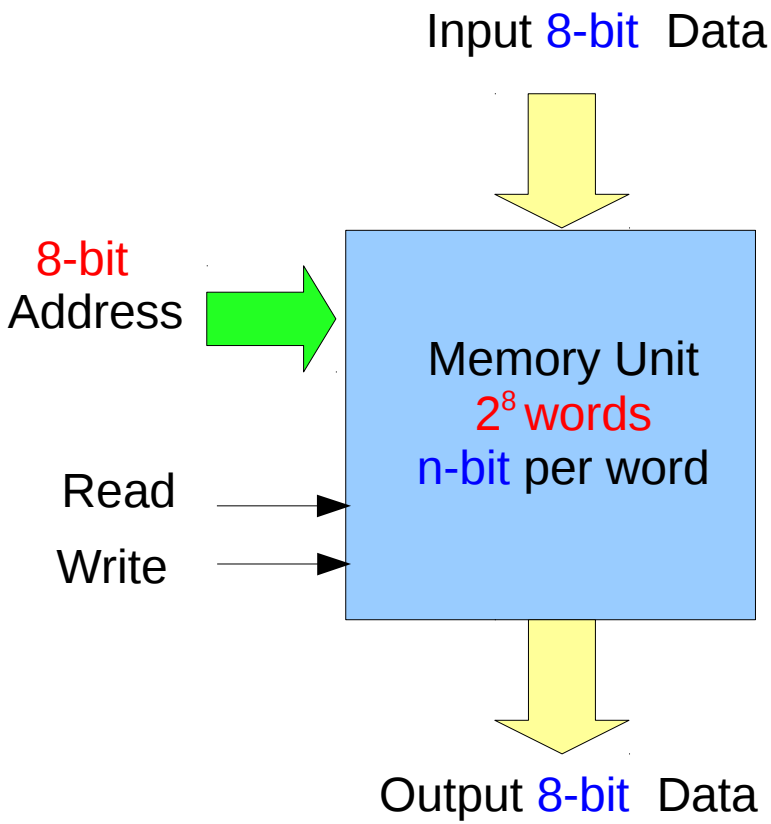
Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

Memory Access Operations

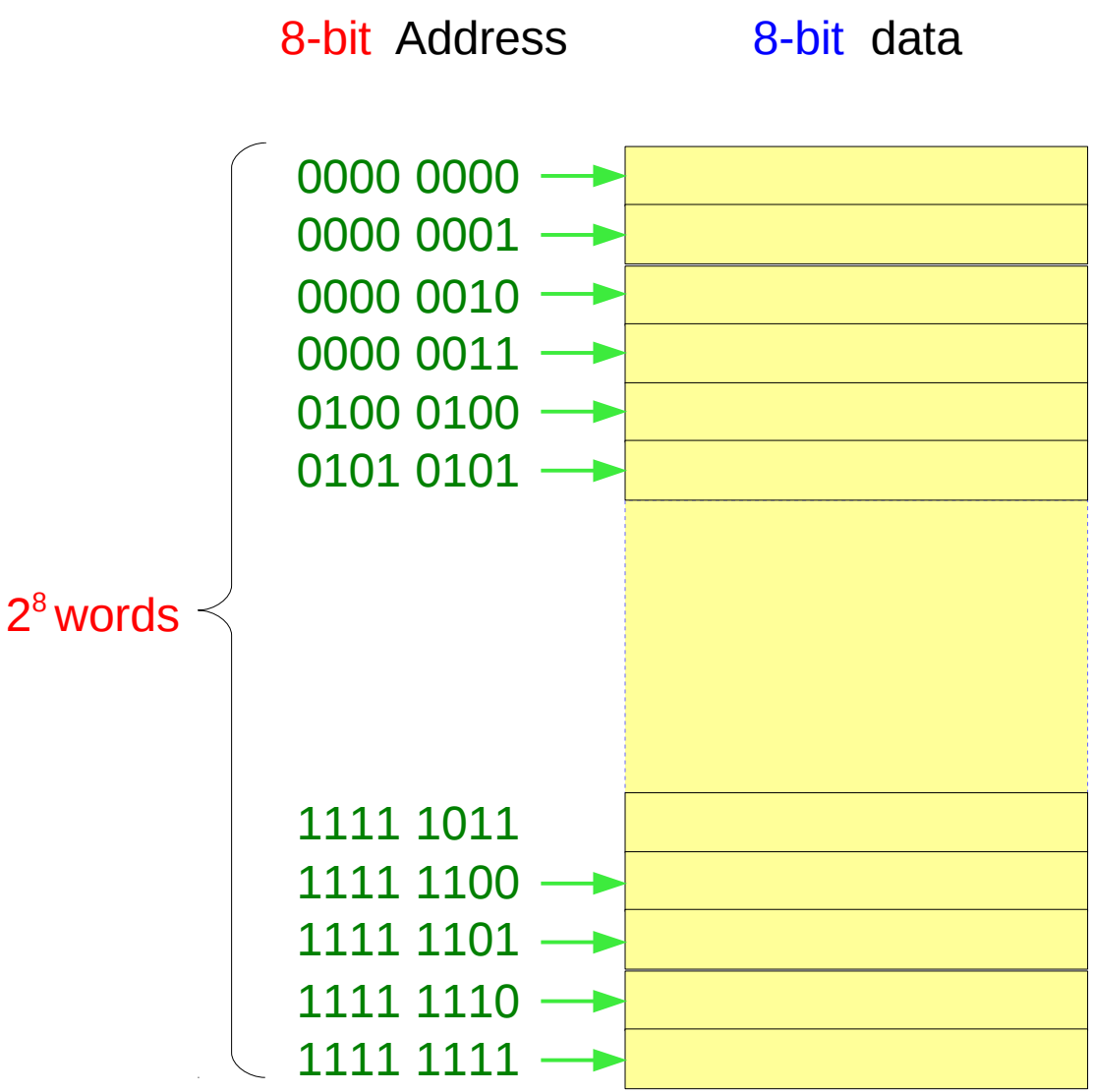




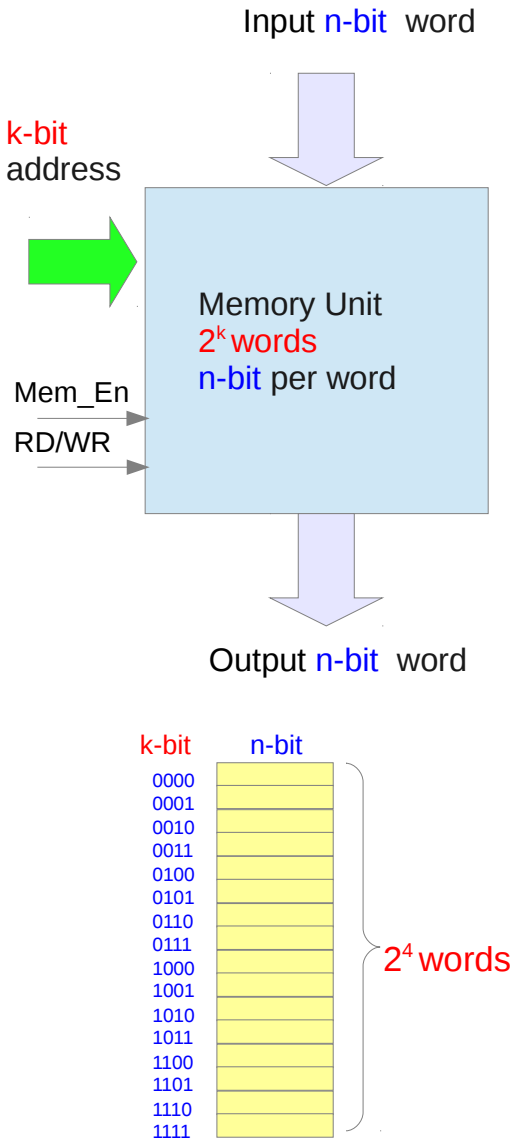
Memory Interface

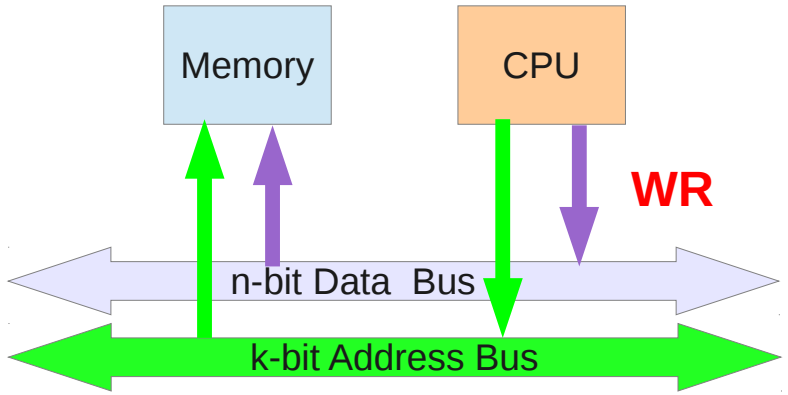


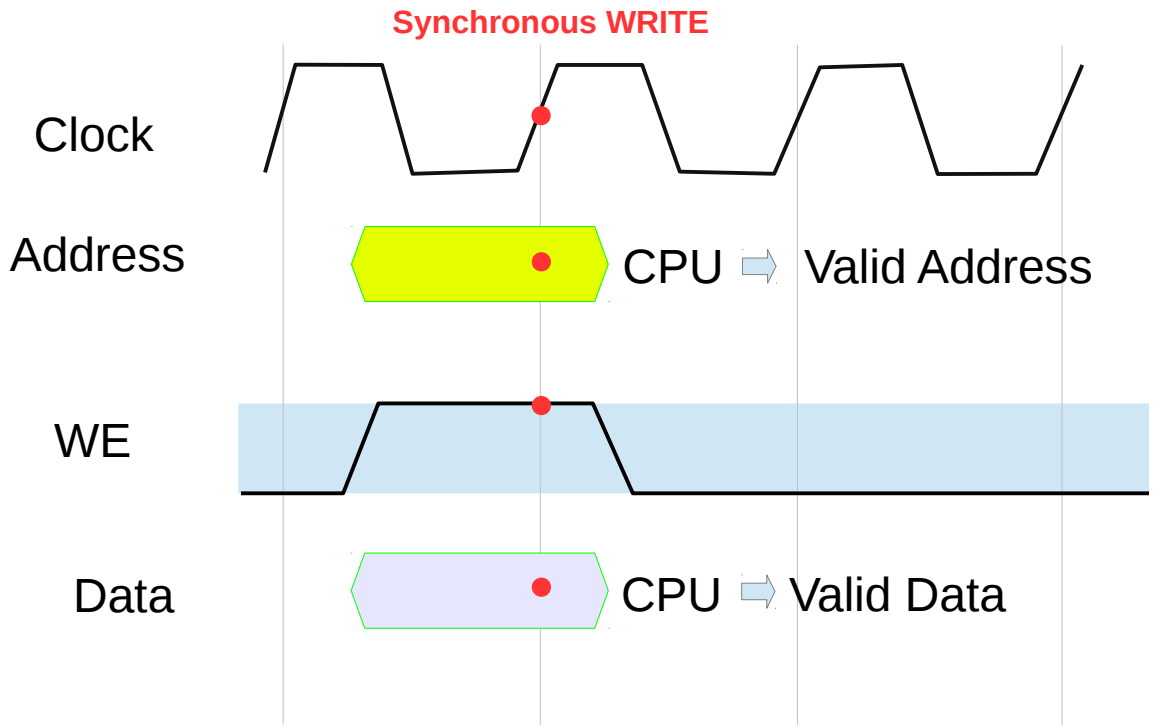
Memory Map


8-bit Address
8-bit data
0000 0000
0000 0001
0000 0010
0000 0011
0100 0100
0101 0101
2^8 words
1111 1011
1111 1100
1111 1101
1111 1110
1111 1111

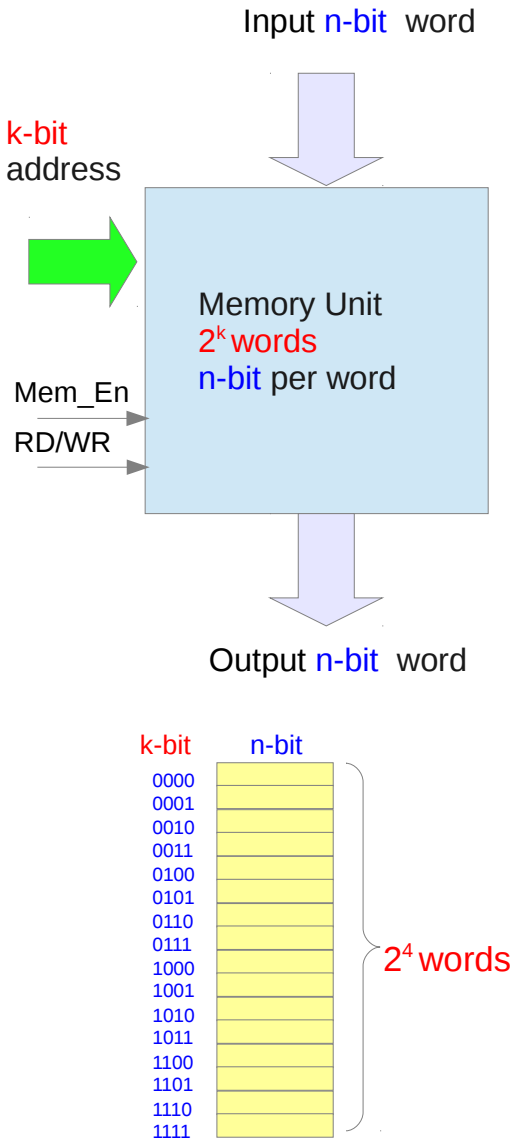
Memory Write Cycle

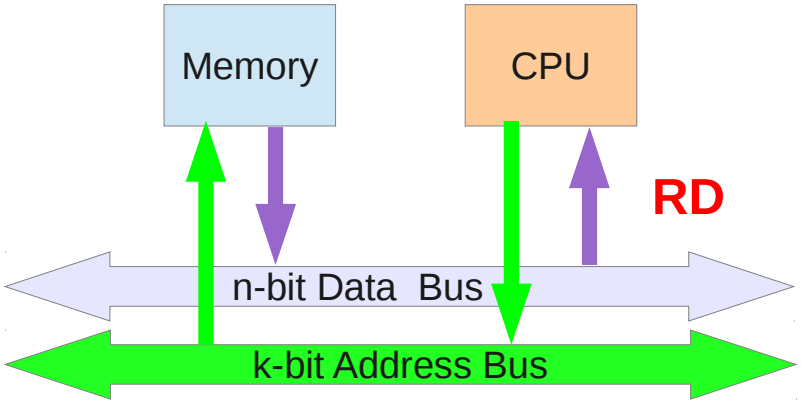


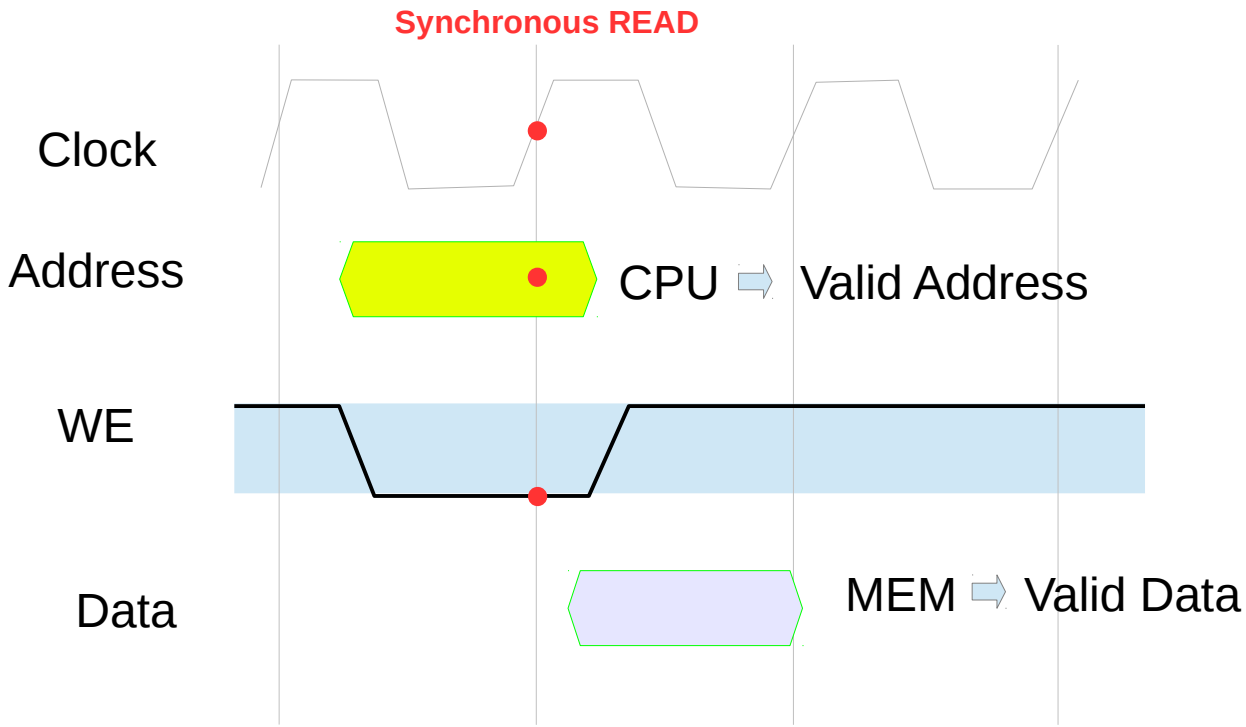




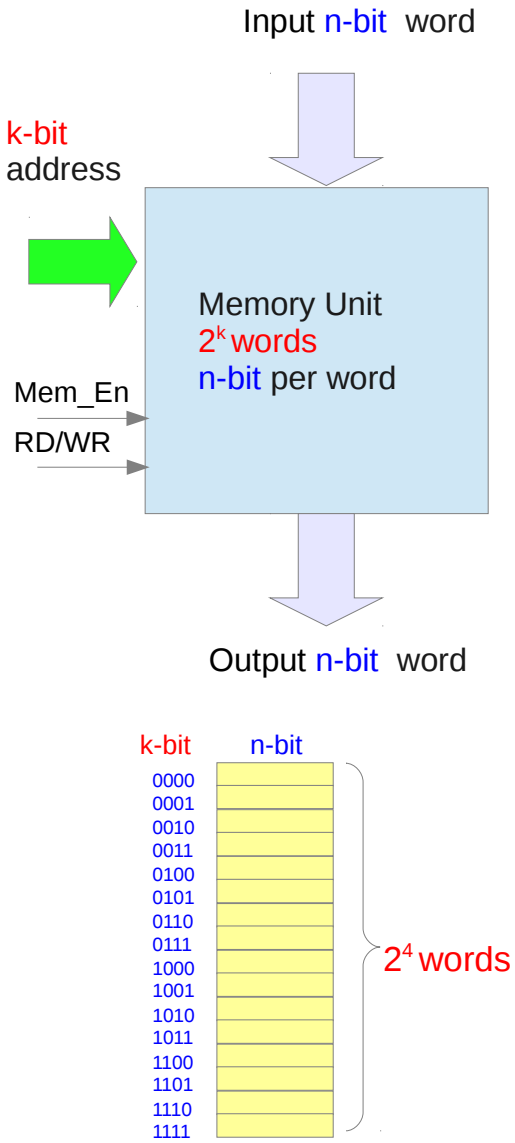
Memory Read Cycle – Synchronous

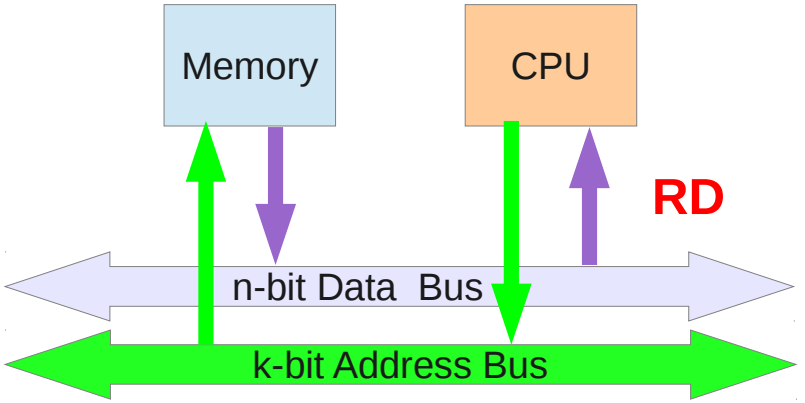


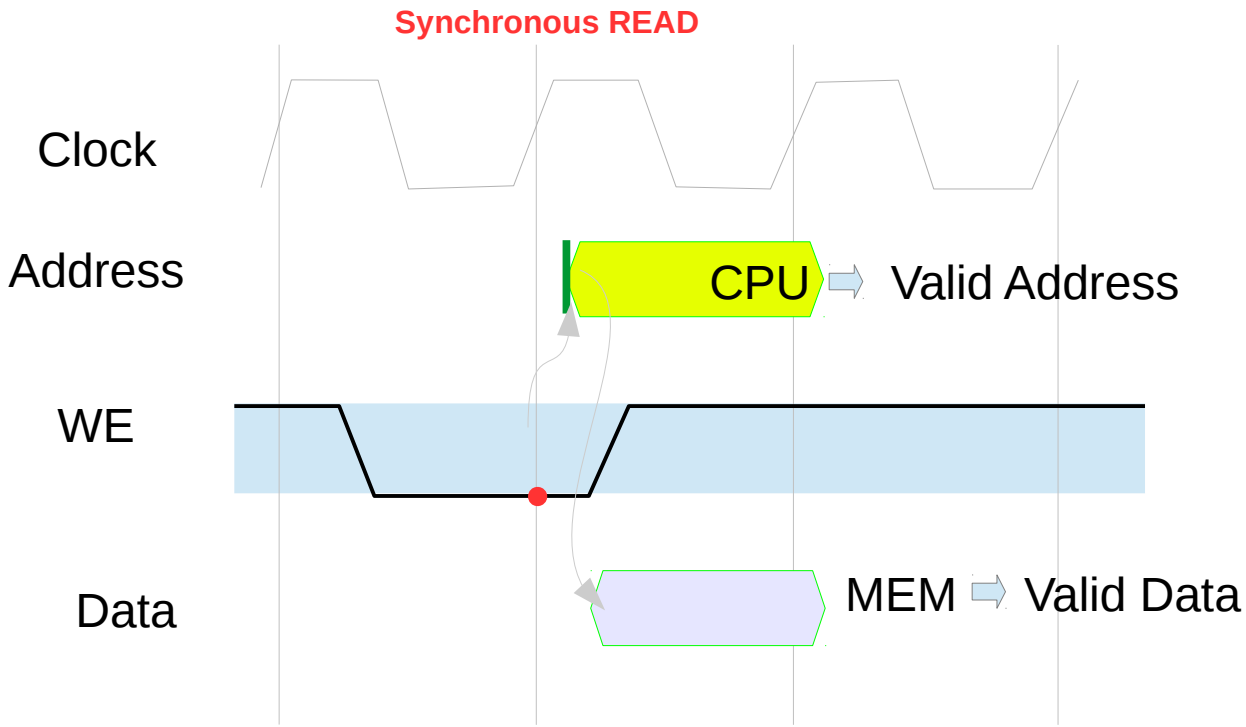




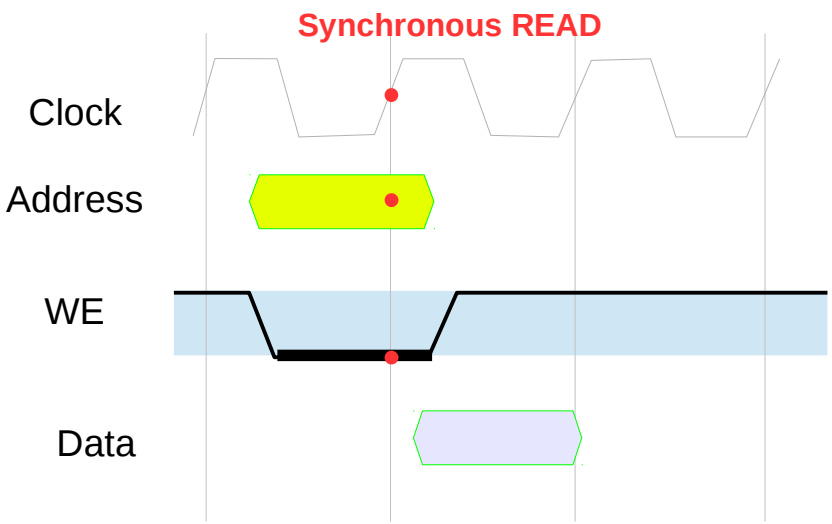
Memory Read Cycle – Asynchronous

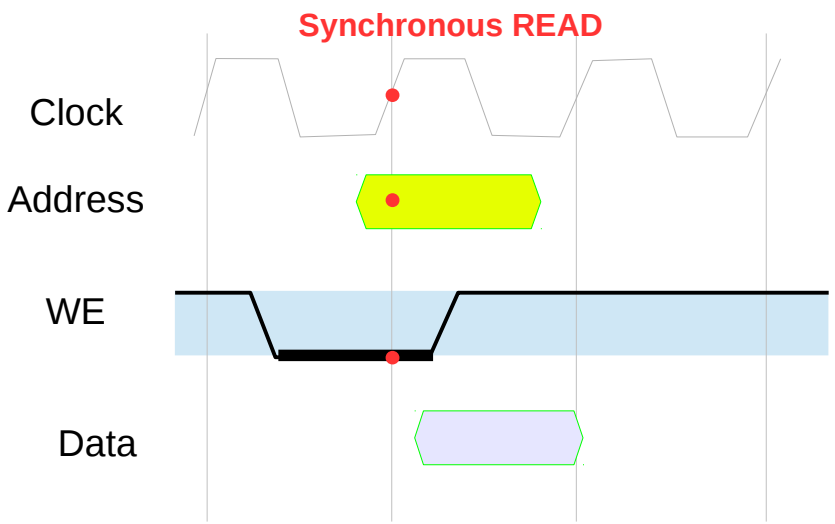


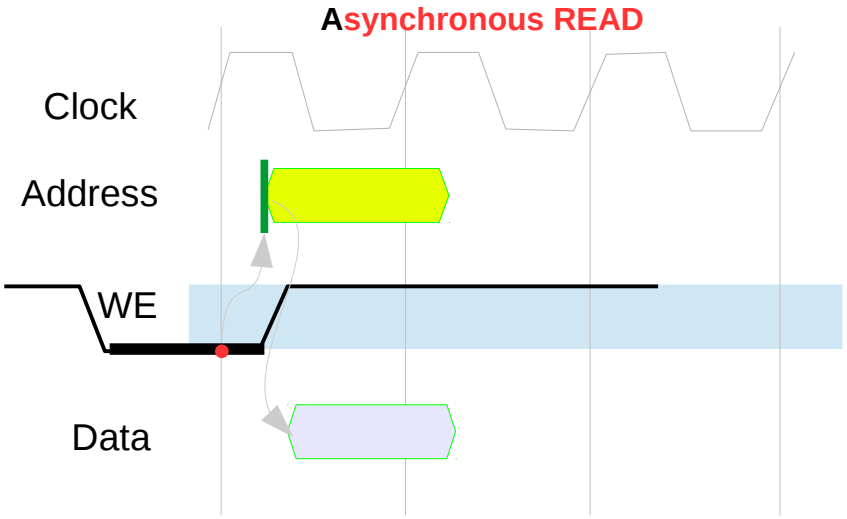


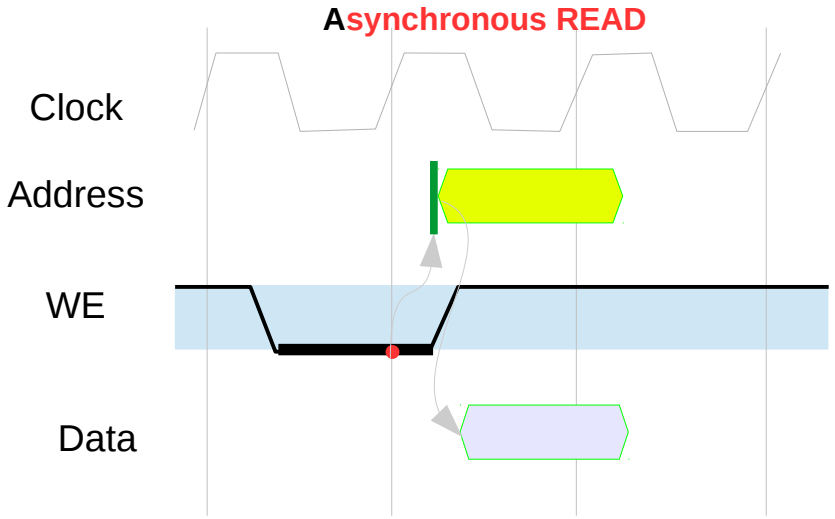


Async & Sync Read Cycle









Virtex LUT RAM

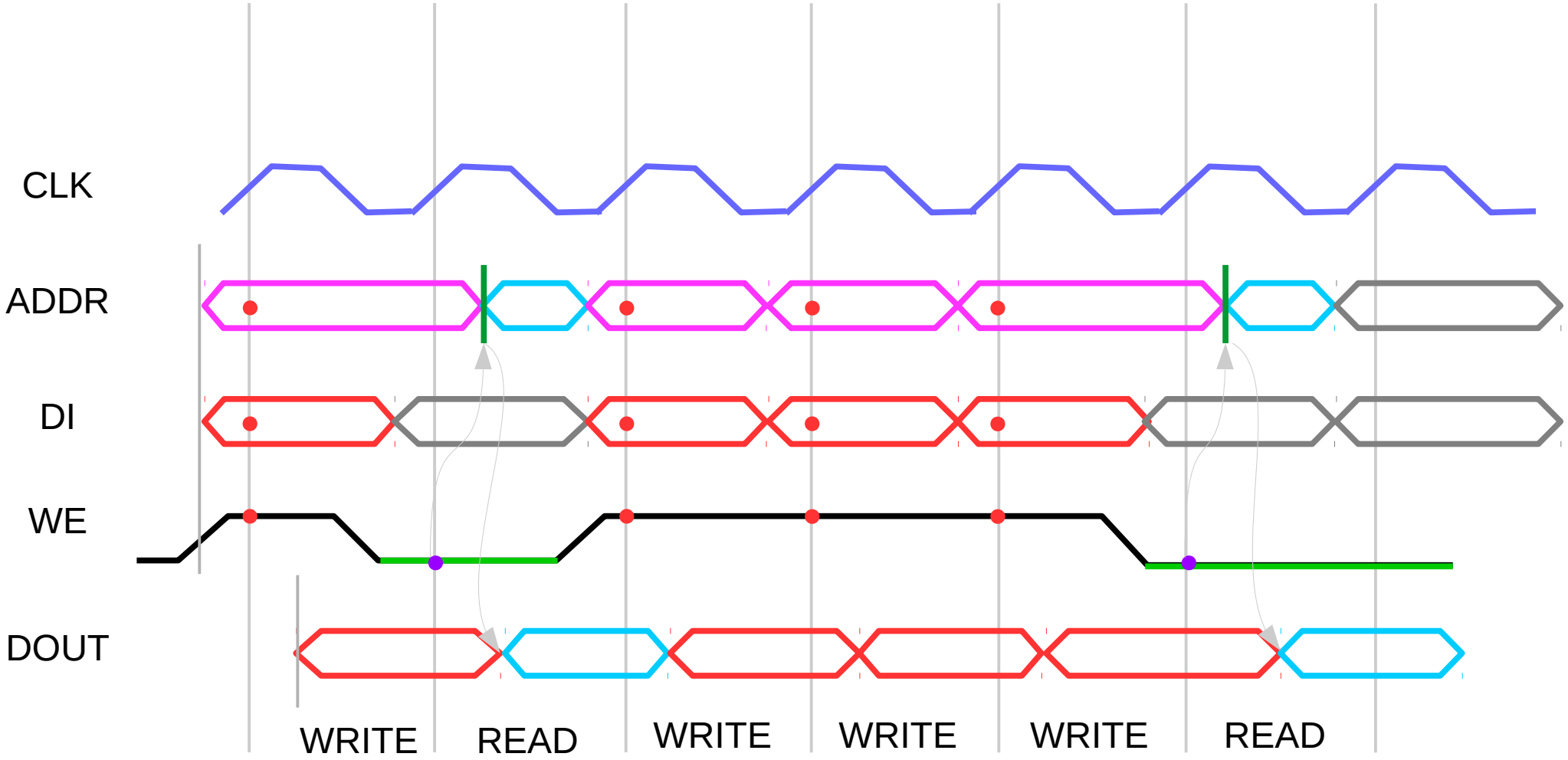
```
module ram16x1(q, a, d, we, clk);
output      q;
input       d;
input [3:0] a;
input       clk, we;

reg         mem [15:0];

always @(posedge clk) begin
   if(we)
      mem[a] <= d;
end

assign q = mem[a];

endmodule
```

Synchronous Write

Asynchronous Read

http://www-inst.eecs.berkeley.edu/~cs150

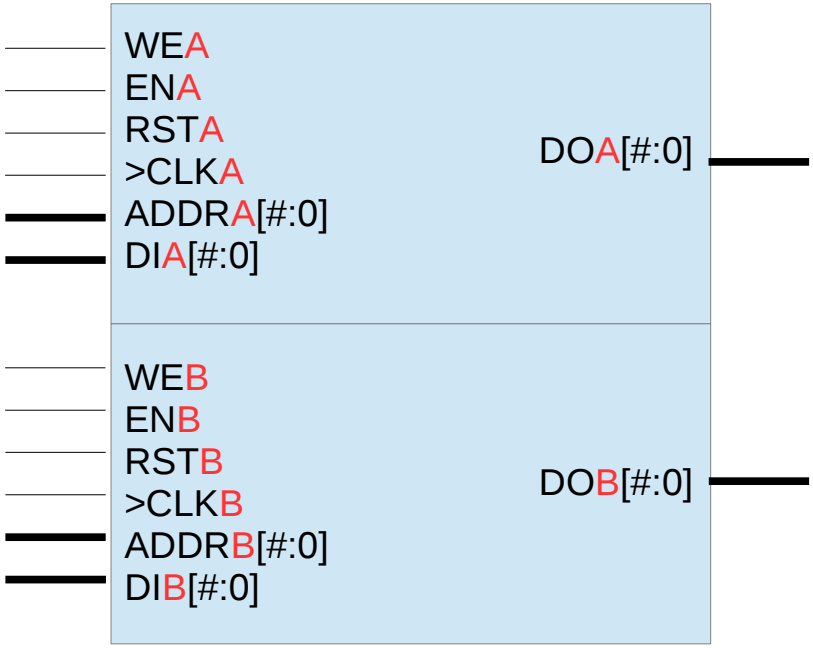
LUT RAM Timing



Xilinx Document

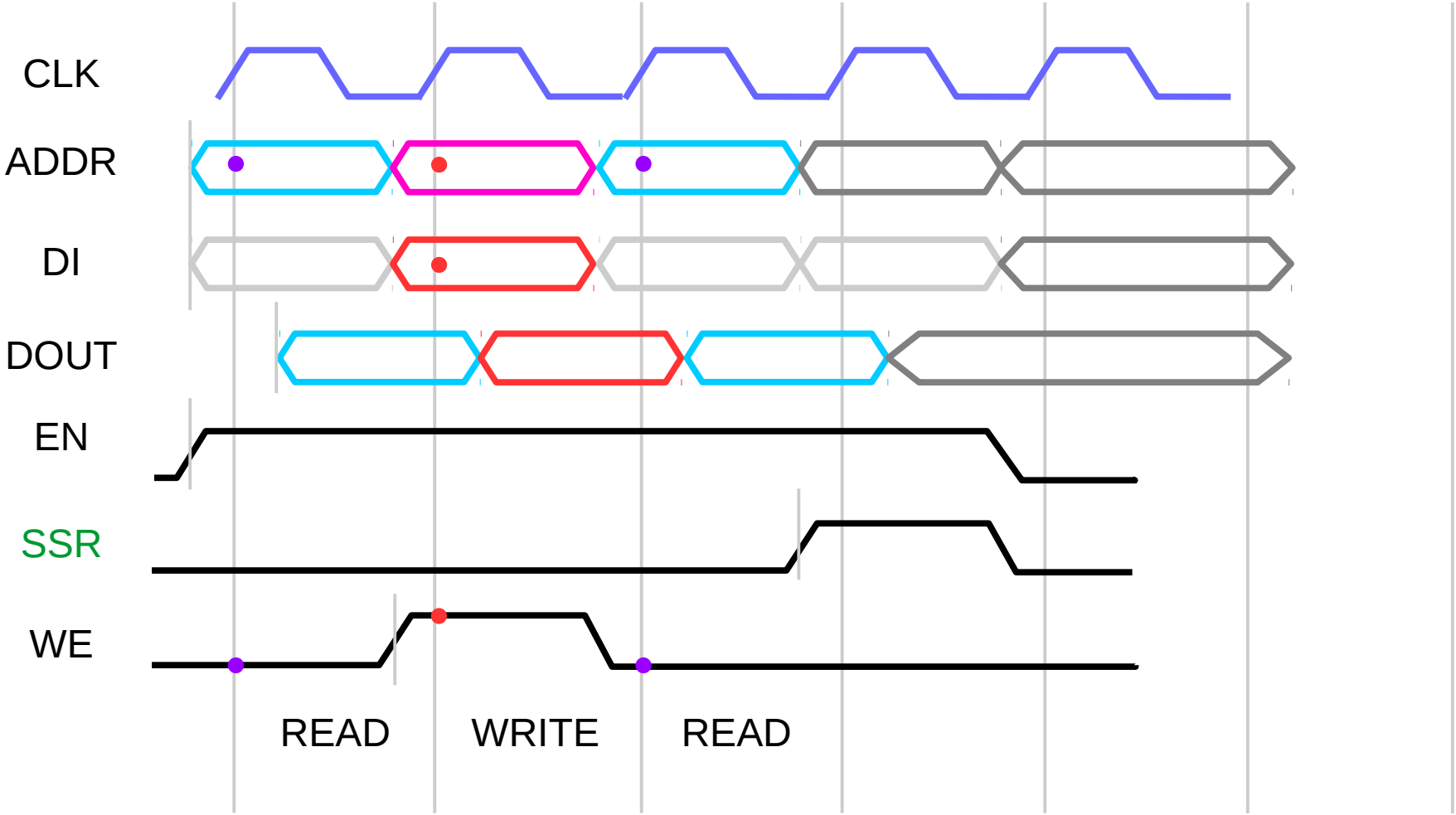
Virtex Block RAM

Every block SelectRAM
- synchronous write
- syncrhronous read
- dual-ported
- independent control signals
- independent data widths
- independent CLKA & CLKB
- 4096-bit RAM



http://www-inst.eecs.berkeley.edu/~cs150

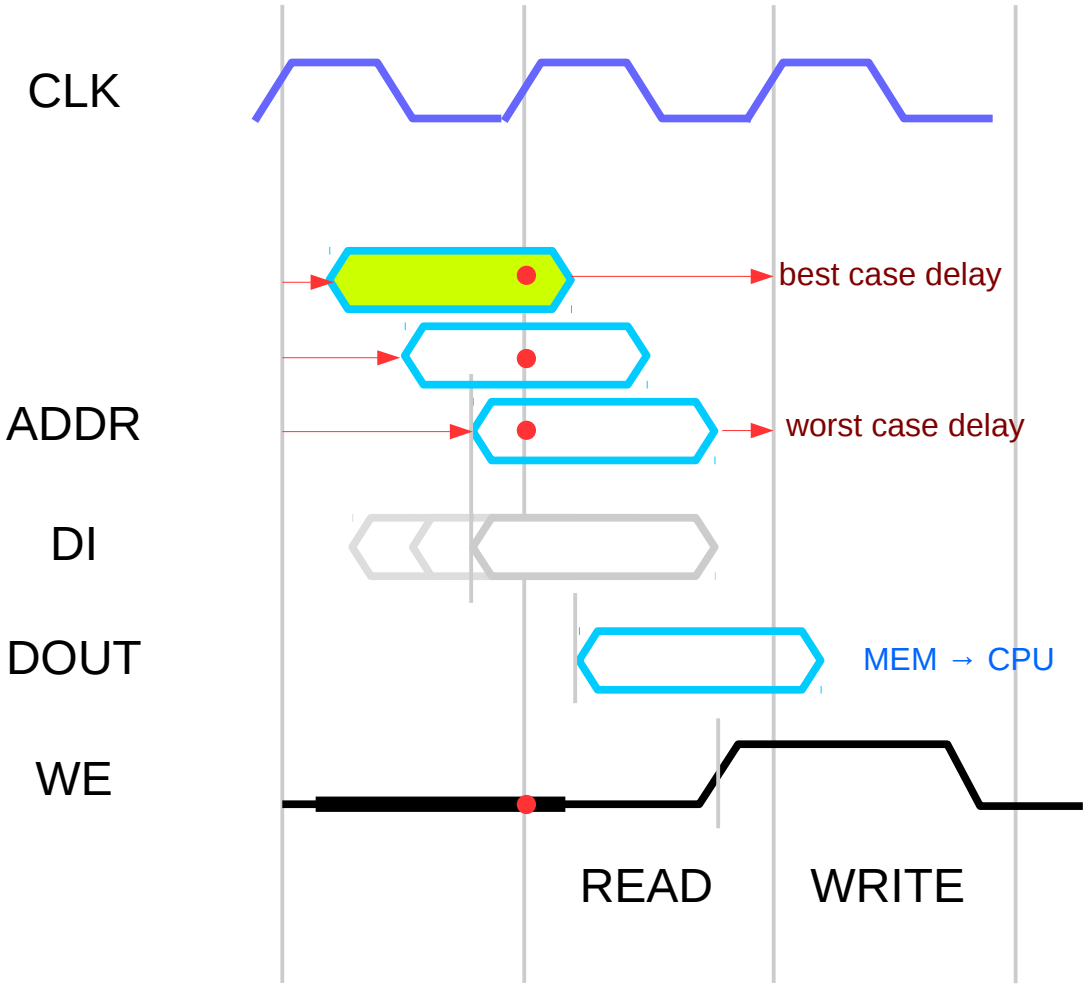
Virtex Block RAM Timing



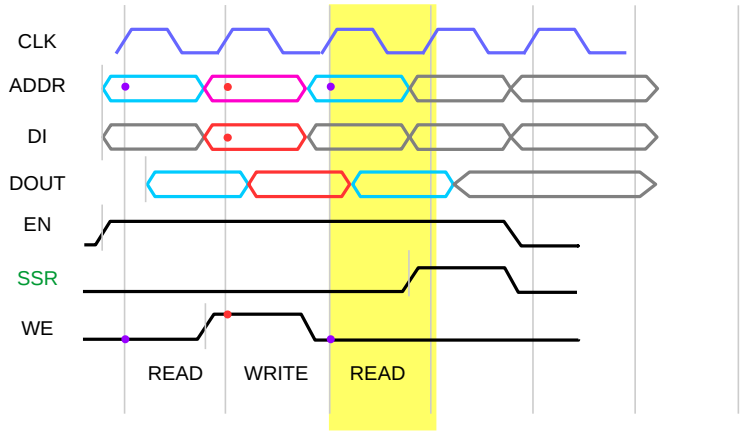
Xilinx Document

SSR (Synchronous Set Rest) value = 1010

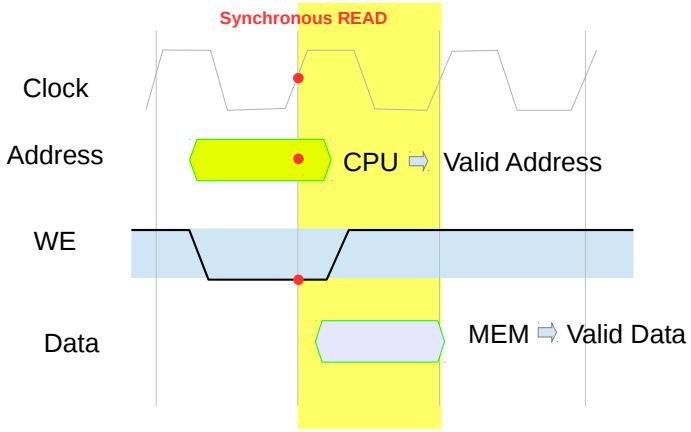
Sync **RD** Cycle - worst & best case



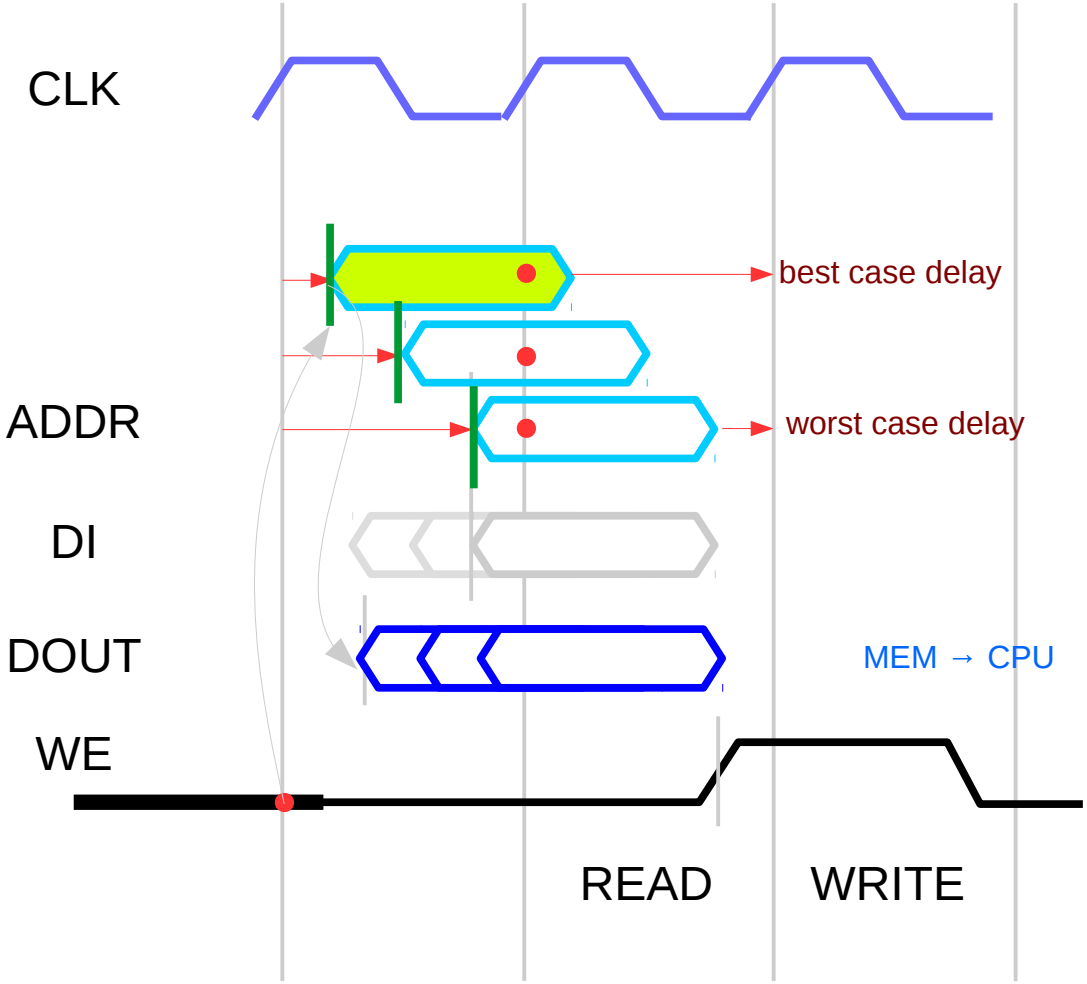
Timing Specification



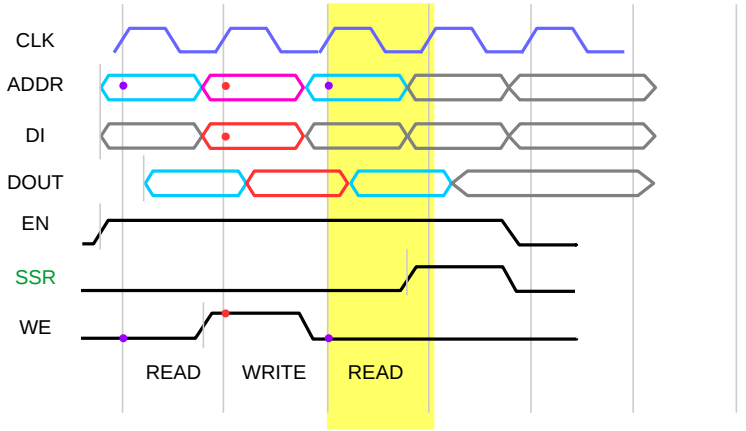
more realistic timing



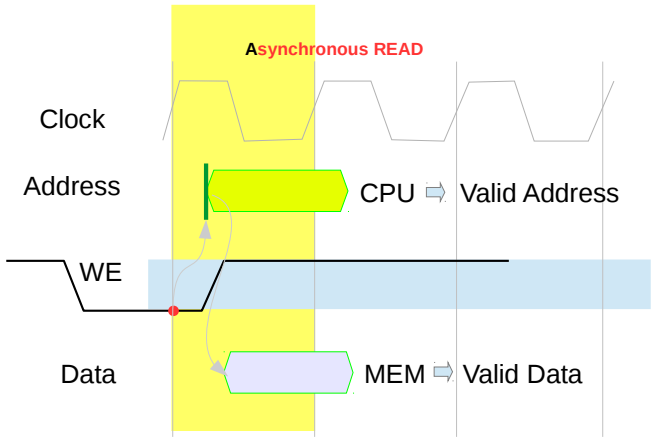
Async **RD** Cycle - worst & best case



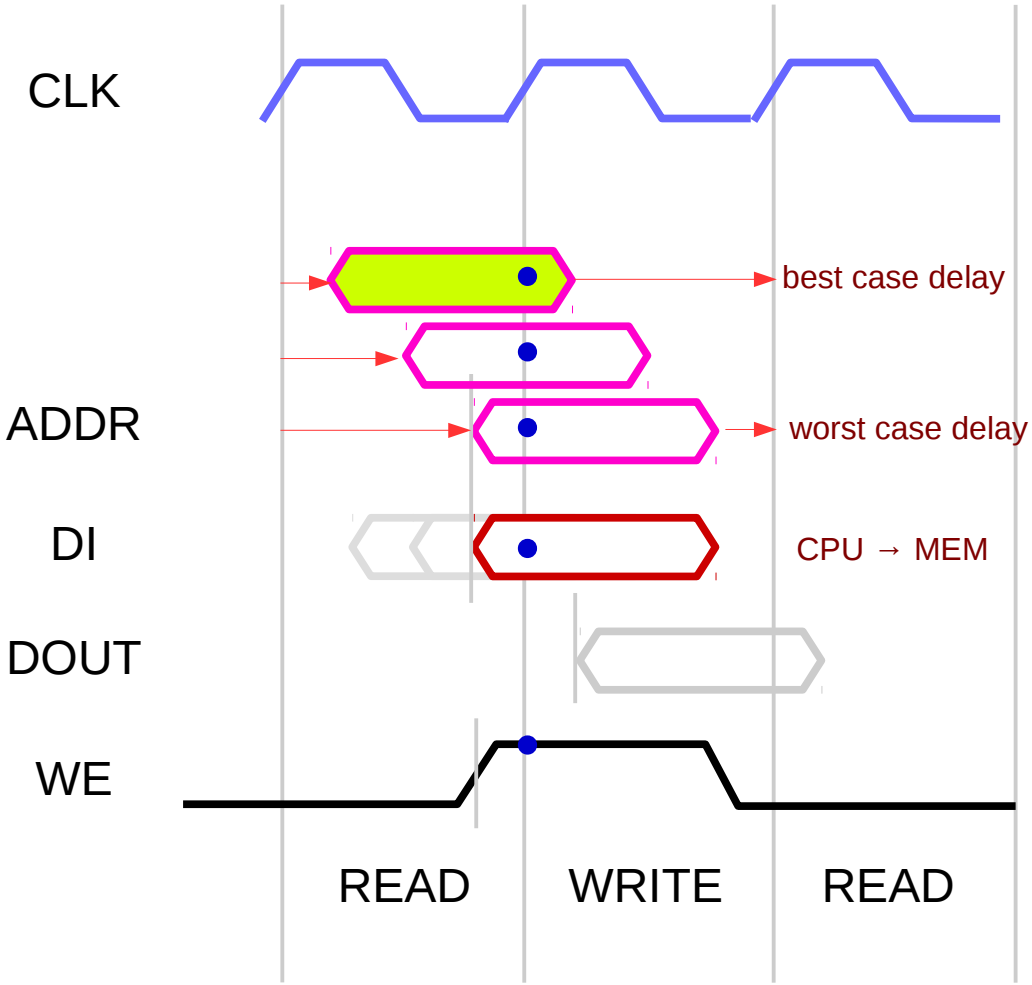
Timing Specification

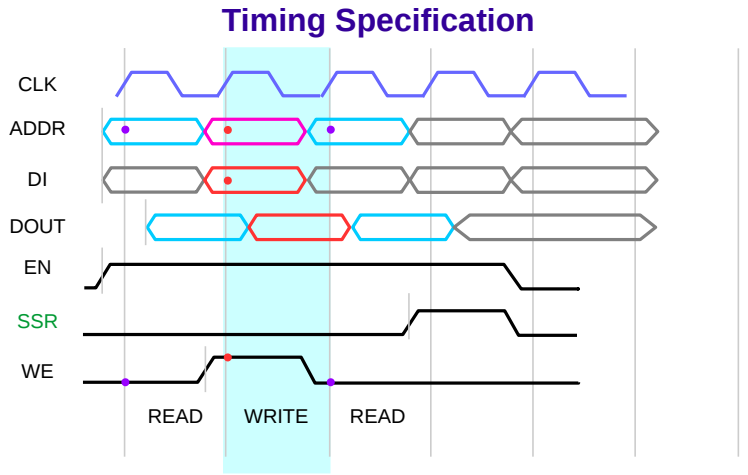


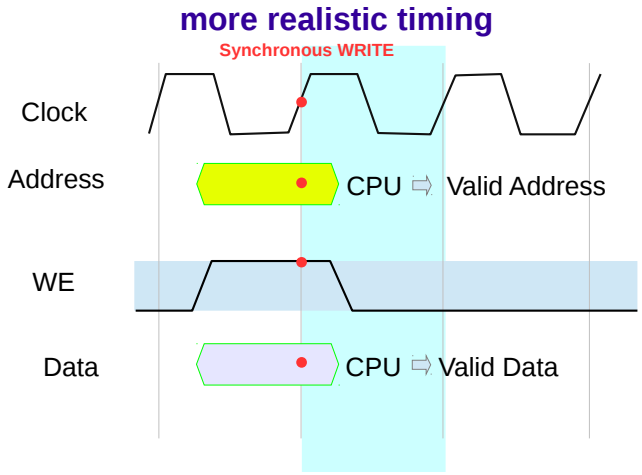
more realistic timing



WR Cycle – worst & best case







Waveform Viewer Timing (1)

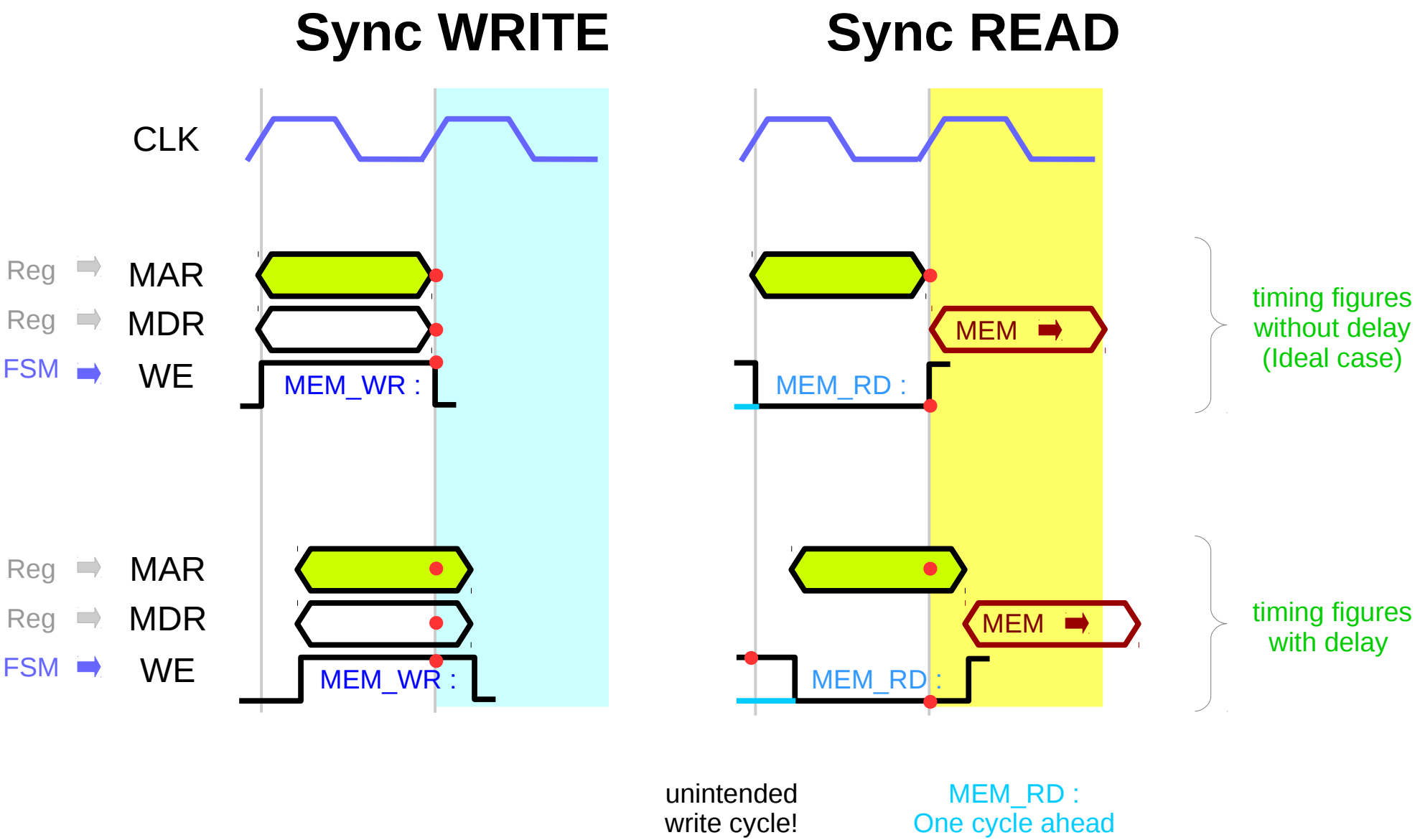
* timing figures without delay (Ideal case)

RTL functional simulation

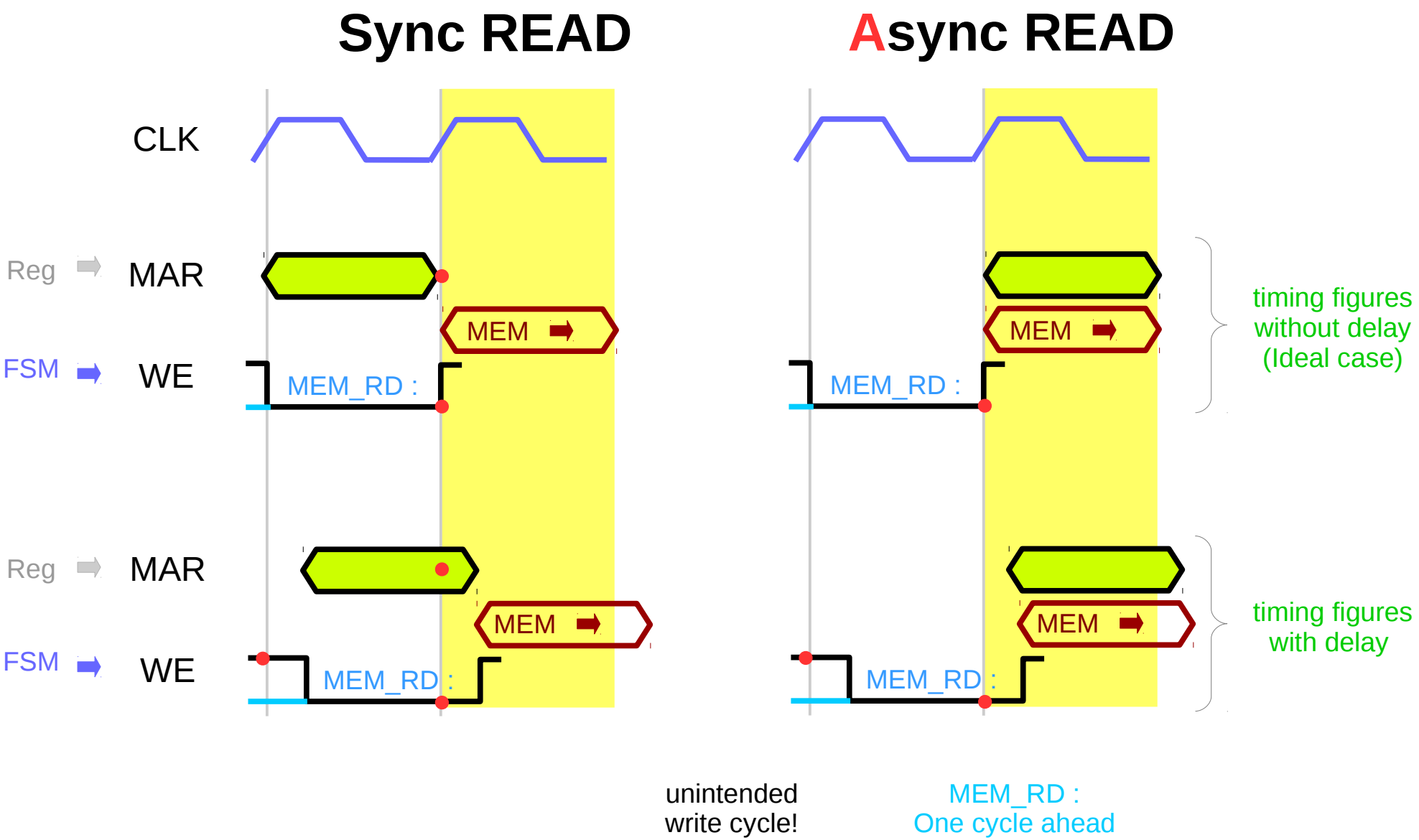
* timing figures with delay

Gate level simulation
with SDF annotation

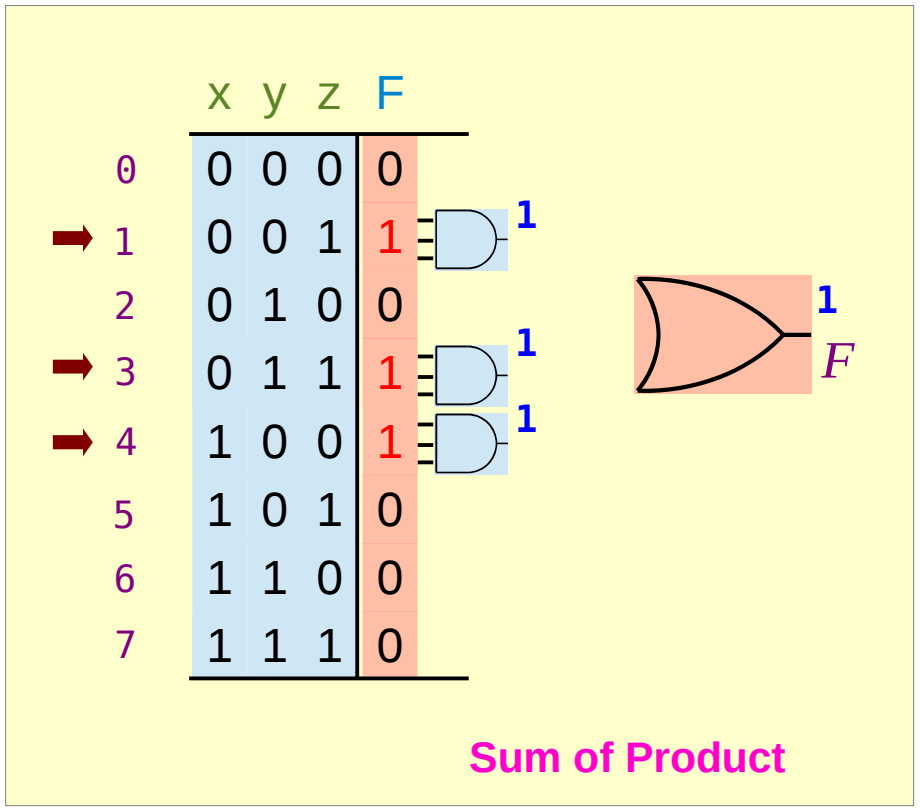
Waveform Viewer Timing (2)

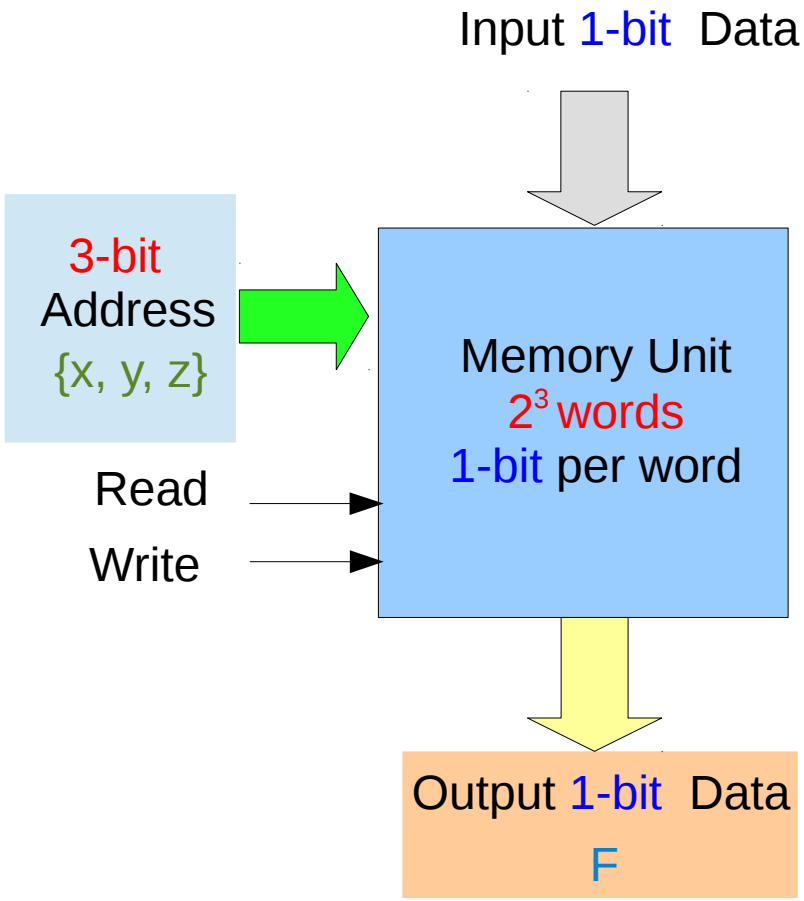


Waveform Viewer Timing (3)



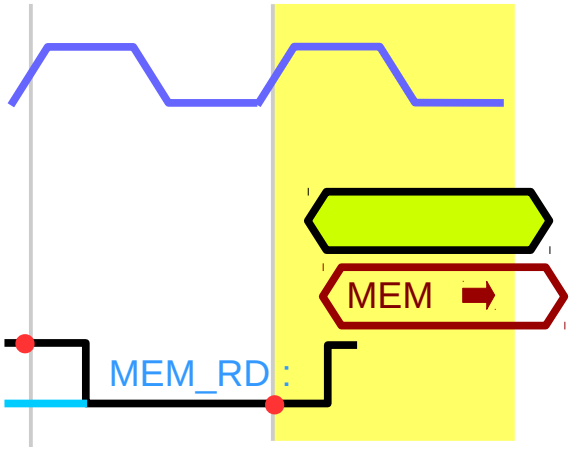
LUT as a combination logic block

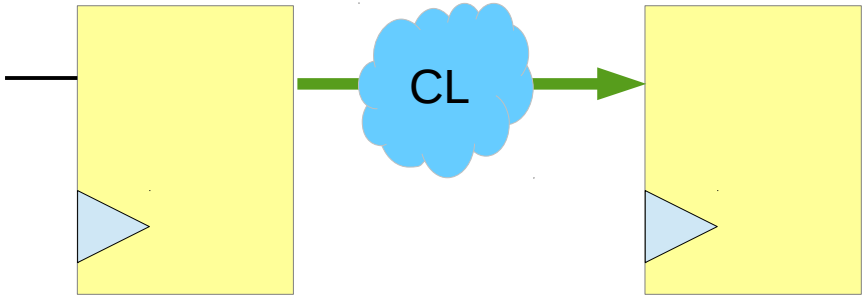




LUT : Async Read

Async READ



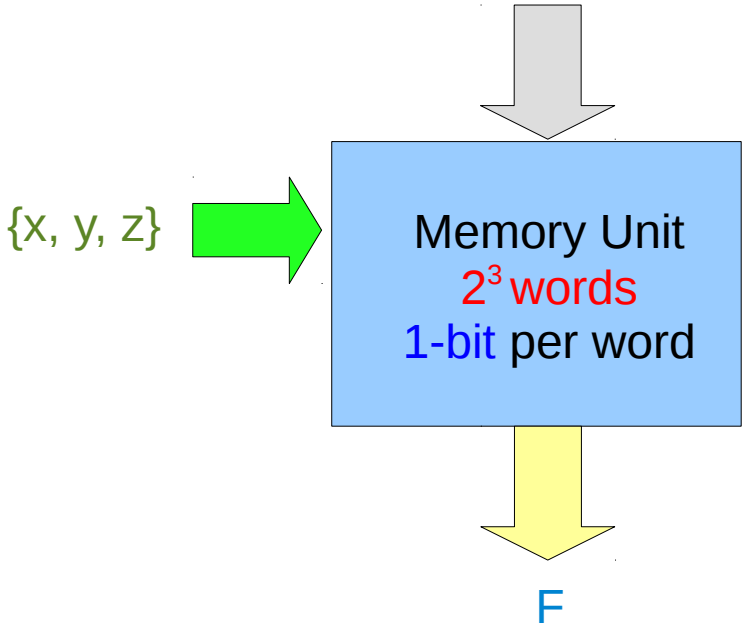


Address : Inputs
Data : Outputs

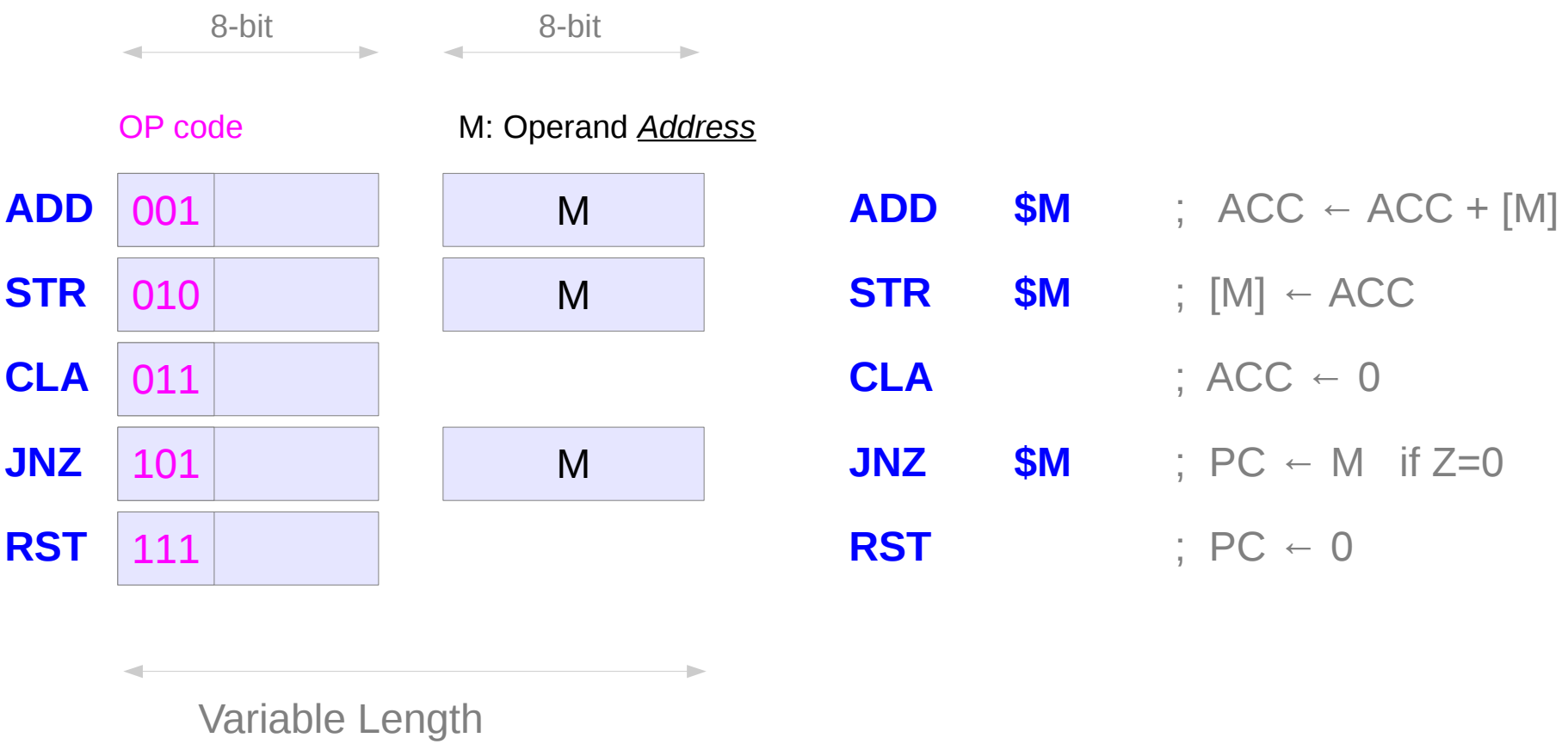
The outputs available in the same cycle where the inputs are applied

Combinational Logic Block :
A set of Boolean Functions

LUT Data Write



Instruction Set Architecture

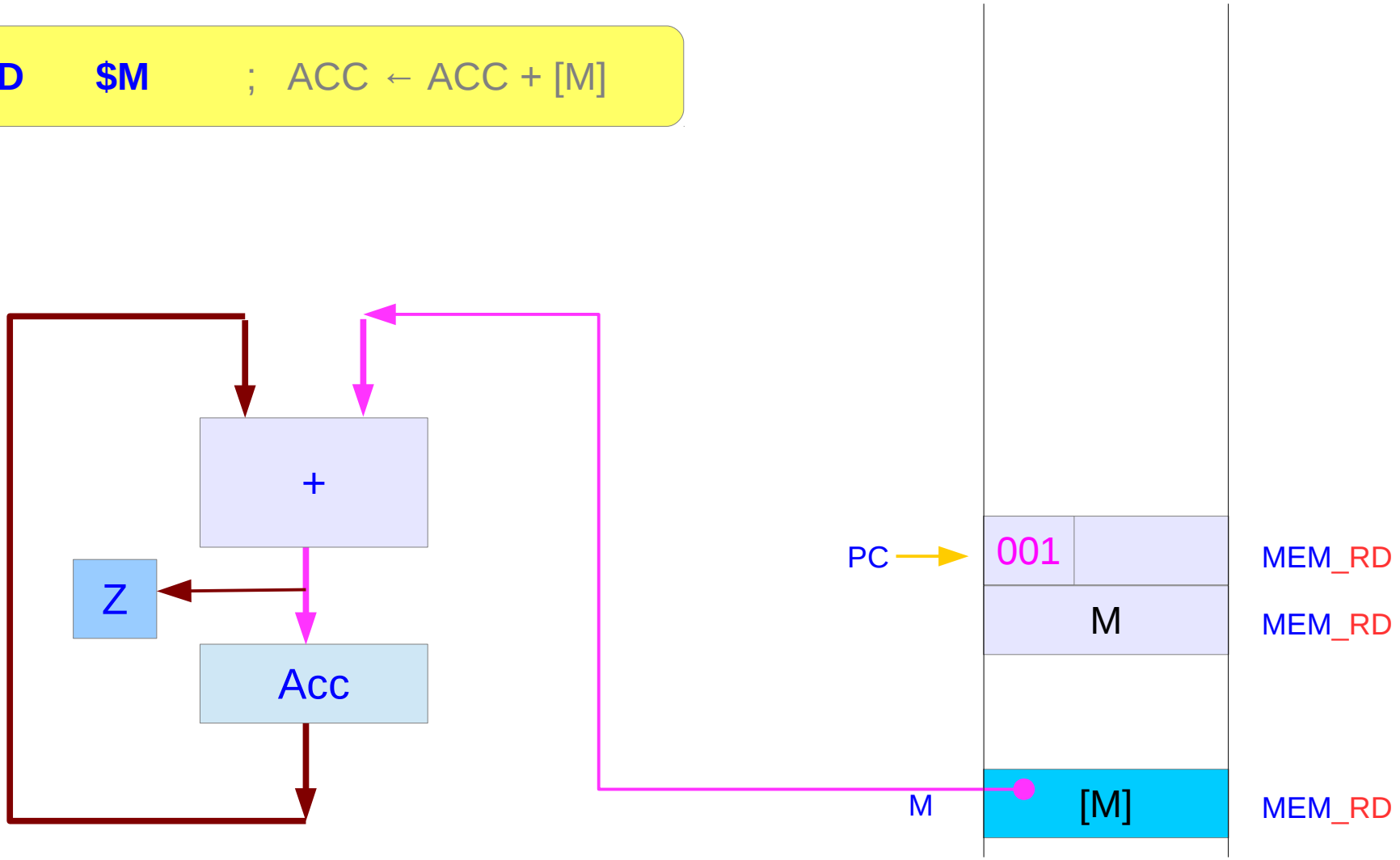


ADD	**$M**	;	ACC ← ACC + [M]
STR	**$M**	;	[M] ← ACC
CLA		;	ACC ← 0
JNZ	**$M**	;	PC ← M if Z=0
RST		;	PC ← 0

Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

ADD Operation

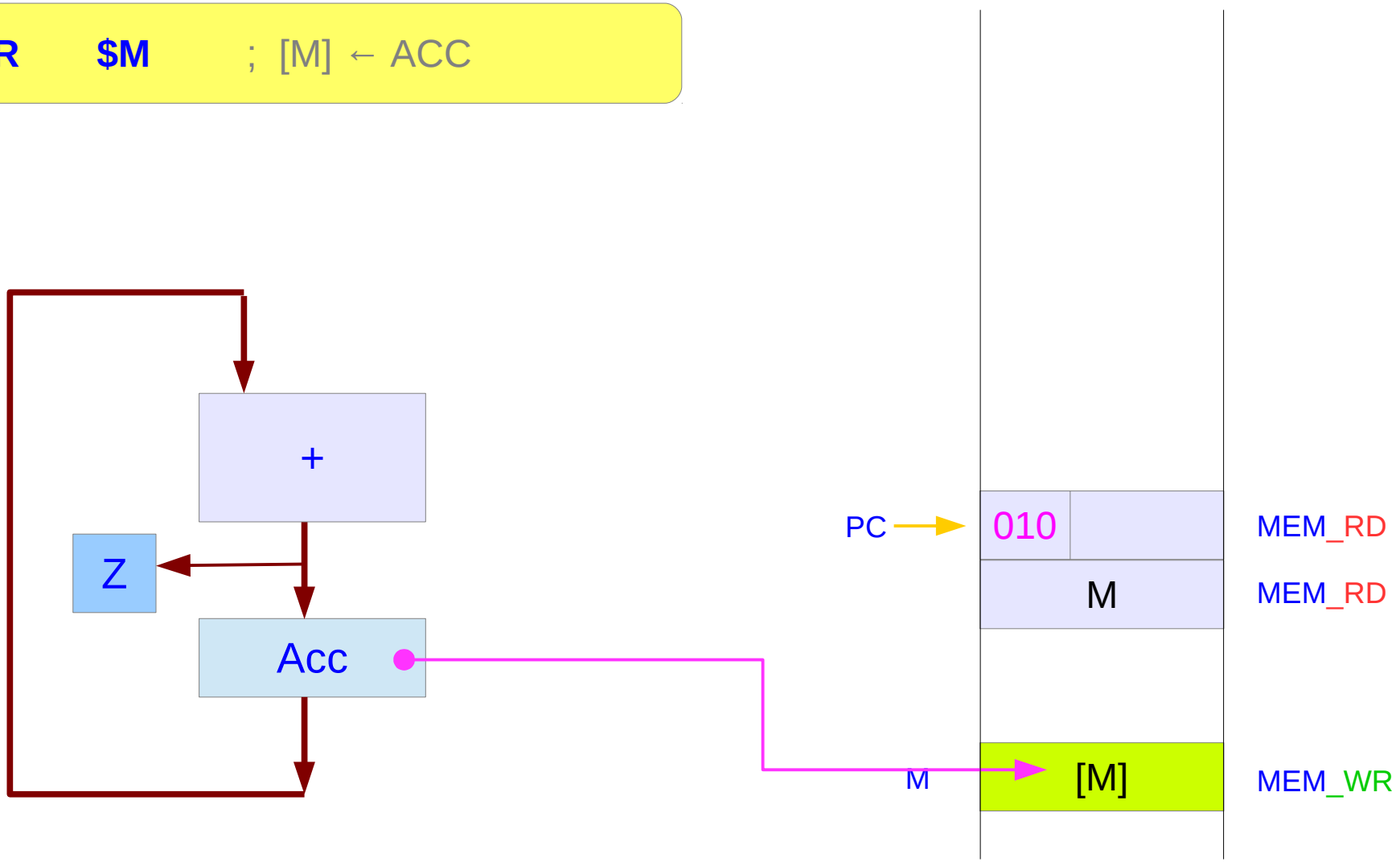
ADD **$M** ; ACC ← ACC + [M]



Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

STR Operation

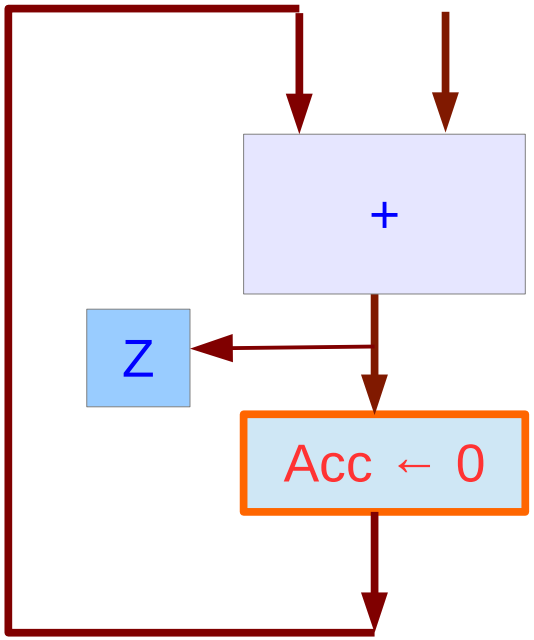
STR **$M** ; [M] ← ACC

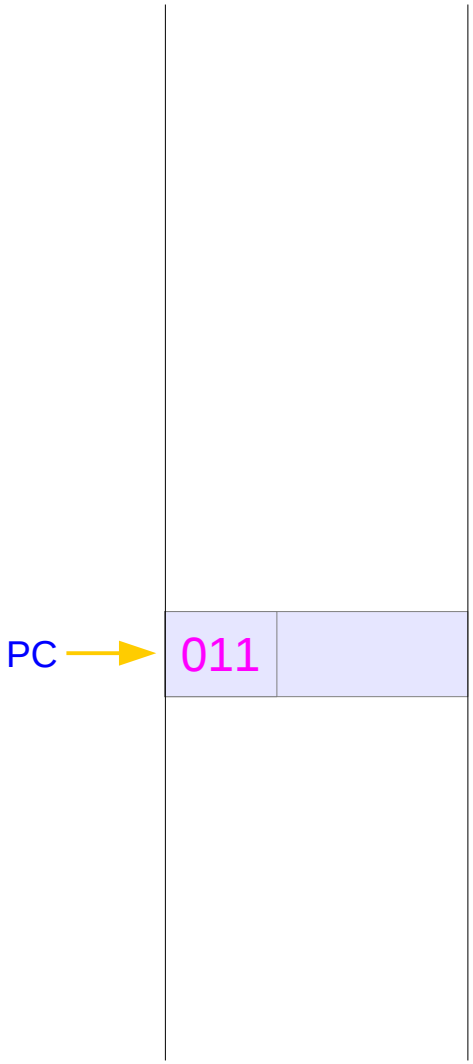


Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

CLA Operation

CLA ; ACC ← 0

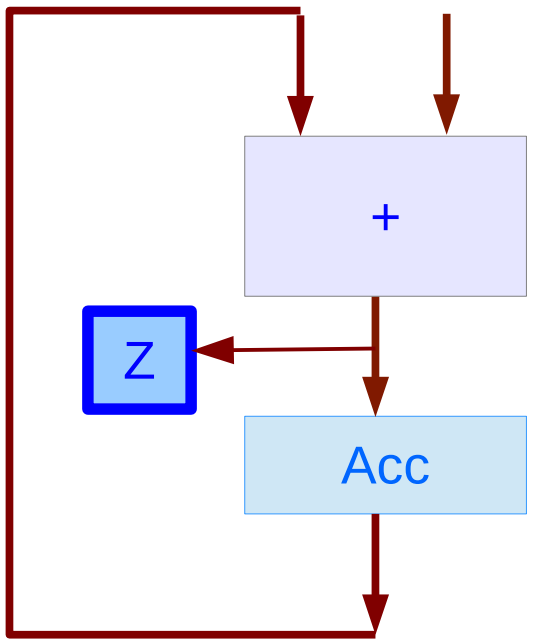




Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

JNZ Operation

JNZ **$M** ; PC ← M if Z=0



If ACC is **not** zero

flag **Z**=0

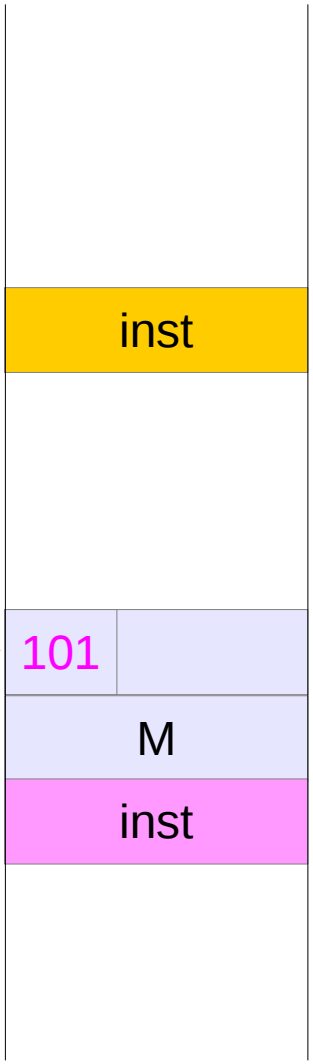
New PC → M

If ACC is **zero**

flag **Z**=1

New PC →

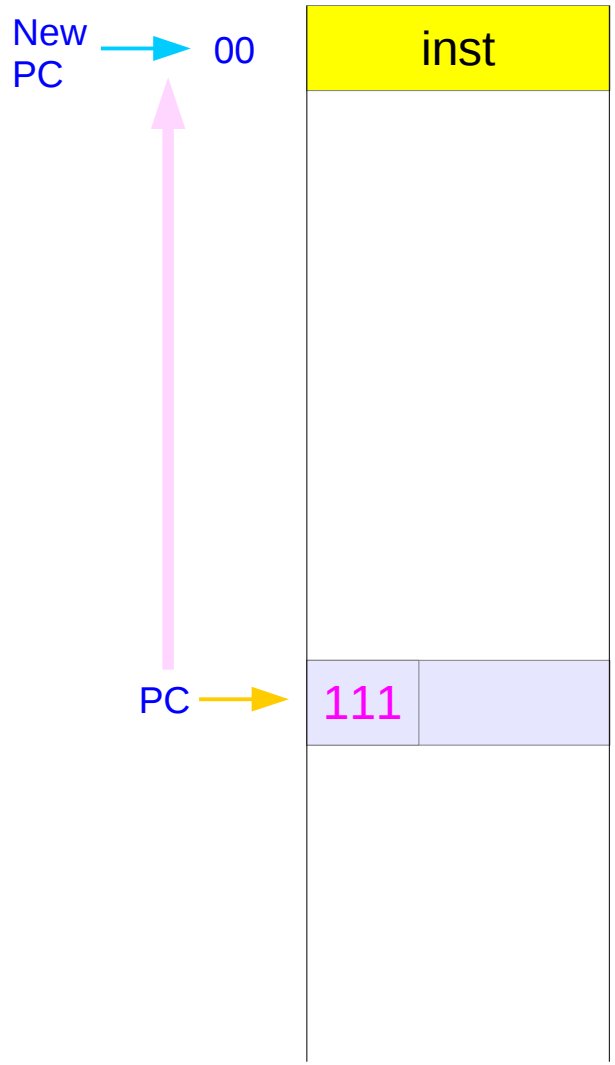
PC →



Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

RST Operation

RST ; PC ← 0



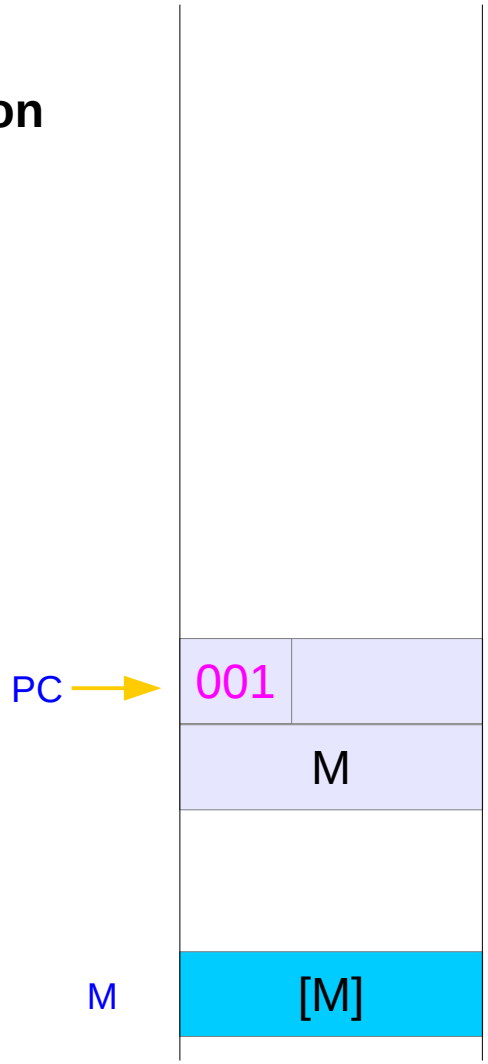
Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

MEM Access in ADD Instruction

2 RD – Instruction

1 RD – Operand

ADD $M ; ACC ← ACC + [M]



	MAR	MDR
MEM_RD	PC	OPCode001→
MEM_RD	PC' = PC+1	M→
MEM_RD	M	[M]→

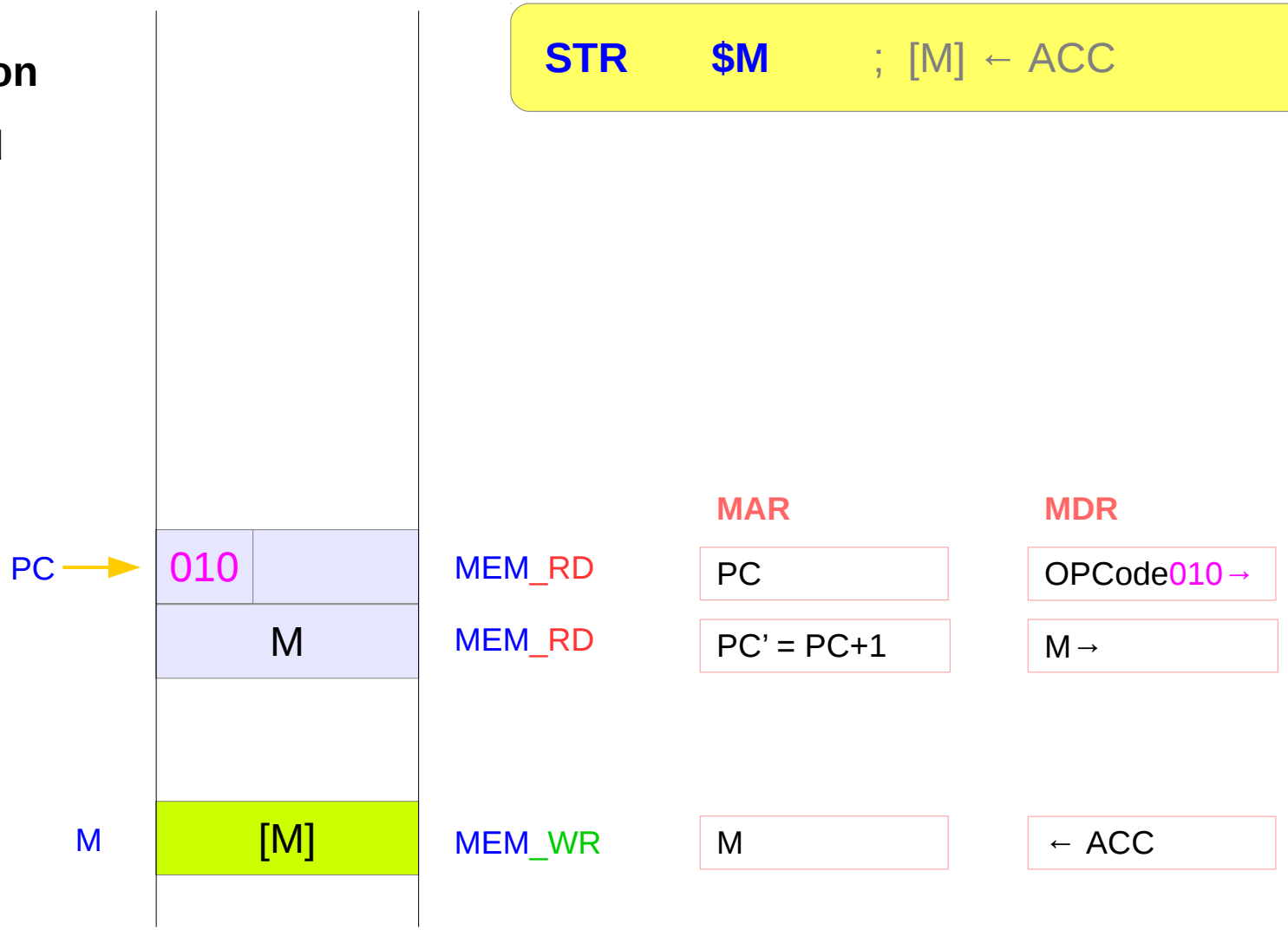
Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

MEM Access STR Instruction

2 RD – Instruction

1 WR – Operand

STR $M ; [M] ← ACC

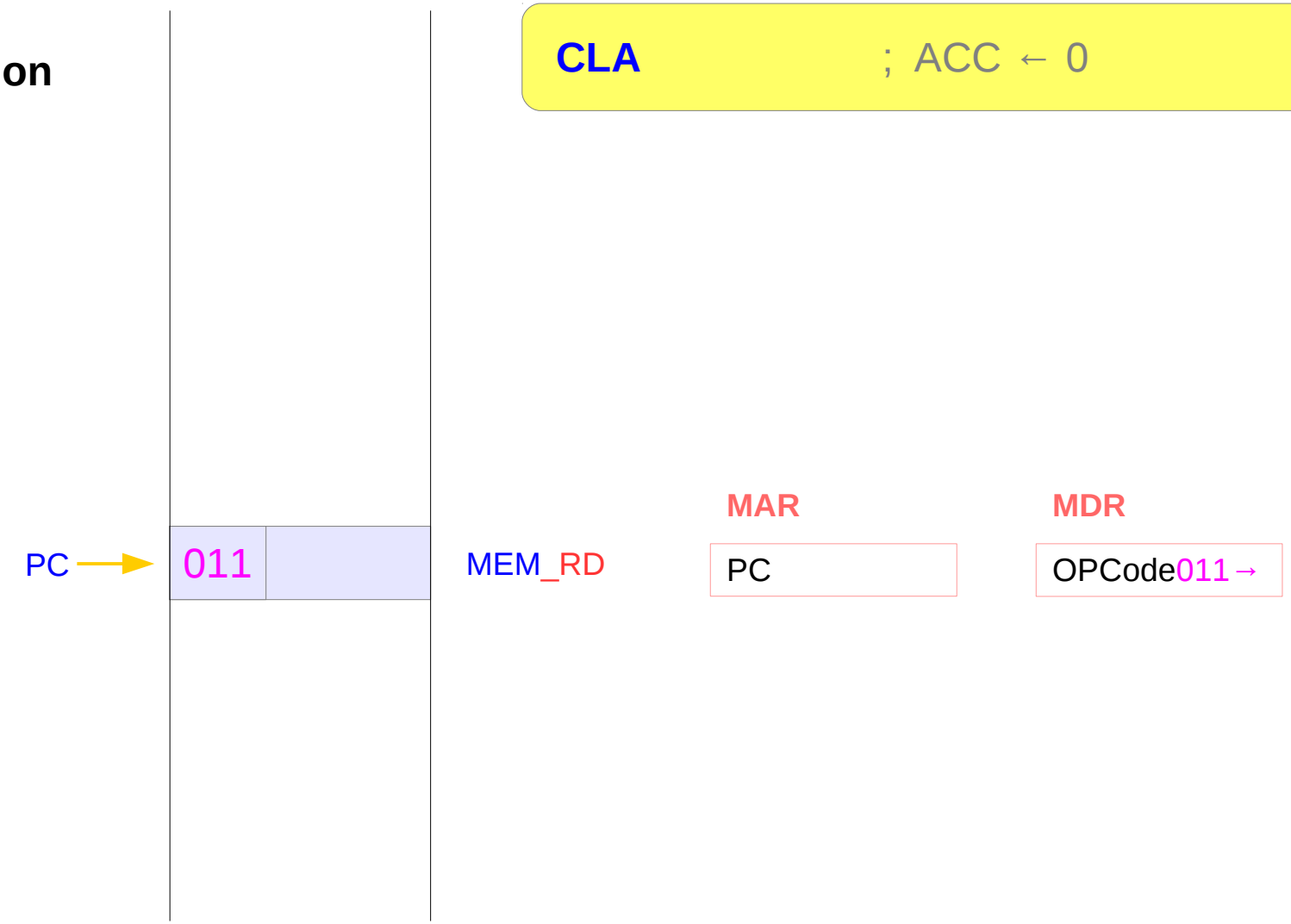


Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

MEM Access CLA Instruction

1 RD – Instruction

CLA ; ACC ← 0

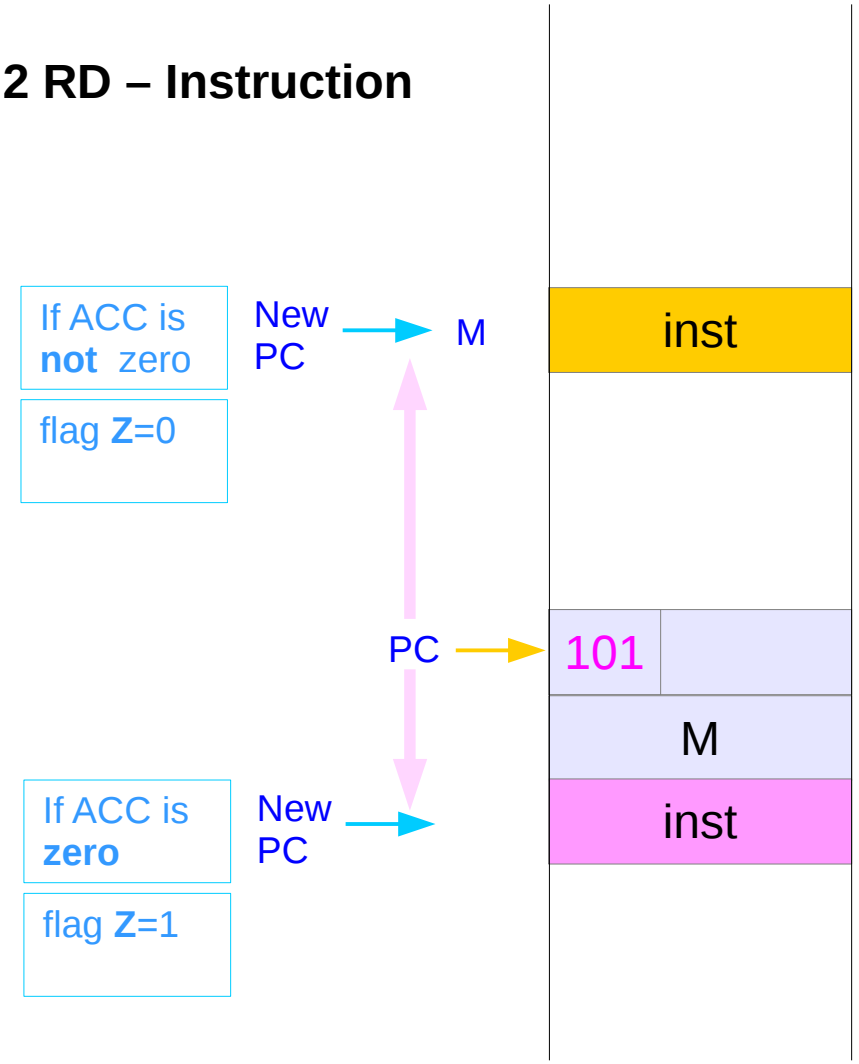


Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

MEM Access JNZ Instruction

2 RD – Instruction

JNZ **$M** ; PC ← M if Z=0



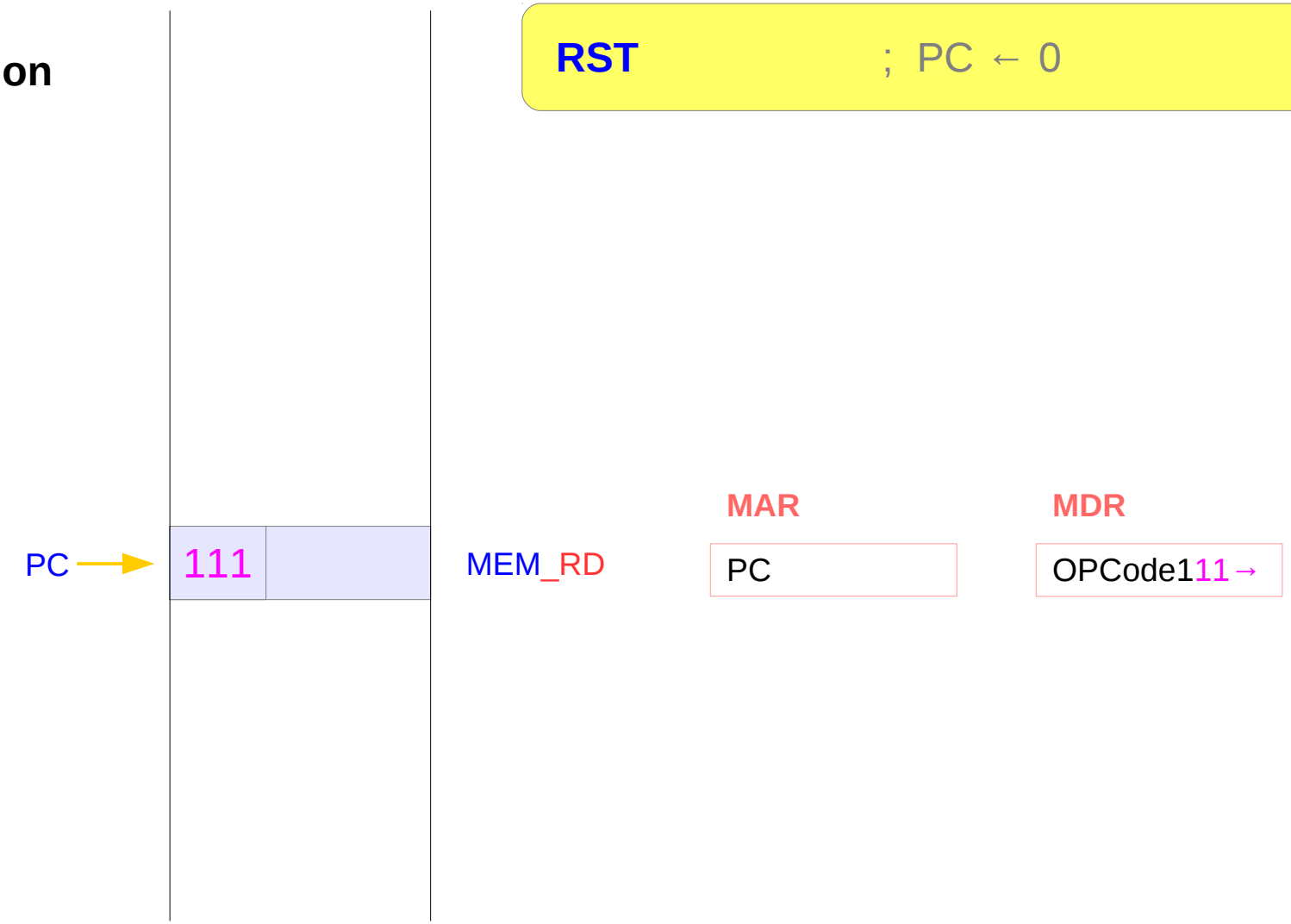
	MAR	MDR
MEM_RD	PC	OPCode101→
MEM_RD	PC' = PC+1	M →

Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

MEM Access RST Instruction

1 RD – Instruction

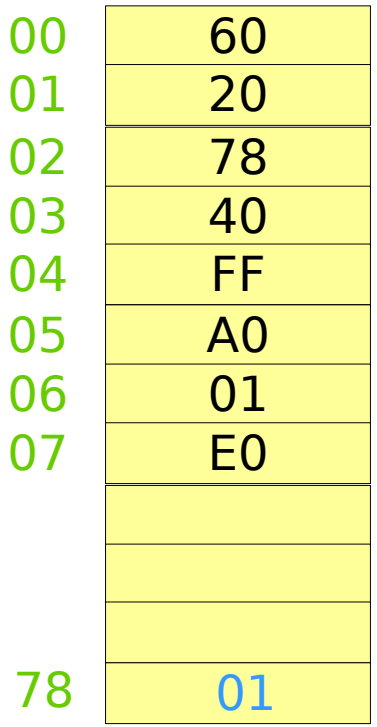
RST ; PC ← 0

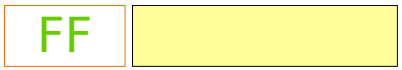


Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

```
Addr. Machine Codes
00   60                 CLA          ;Clear Acc and then adding the operand is
01   20 78              ADD $ONE     ;equivalent to move the operand to Acc
03   40 FF              STR $FF
05   A0 01              JNZ $01
07   E0                 RST
78   01         ONE:    DAT    0000 0001    ;Constant 1
```

00 → 01 → … → FE → FF → 00





References

[1] http://en.wikipedia.org/
[2] https://en.wikiversity.org/wiki/The_necessities_in_SOC_Design
[3] https://en.wikiversity.org/wiki/The_necessities_in_Digital_Design
[4] https://en.wikiversity.org/wiki/The_necessities_in_Computer_Design
[5] https://en.wikiversity.org/wiki/The_necessities_in_Computer_Architecture
[6] https://en.wikiversity.org/wiki/The_necessities_in_Computer_Organization
[7] https://en.wikiversity.org/wiki/Understanding_Embedded_Software
[8] Digital Systems, Hill, Peterson, 1987
[9] http://en.wikipedia.org/
[10] http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf