

Tiny CPU - ISA (2A)

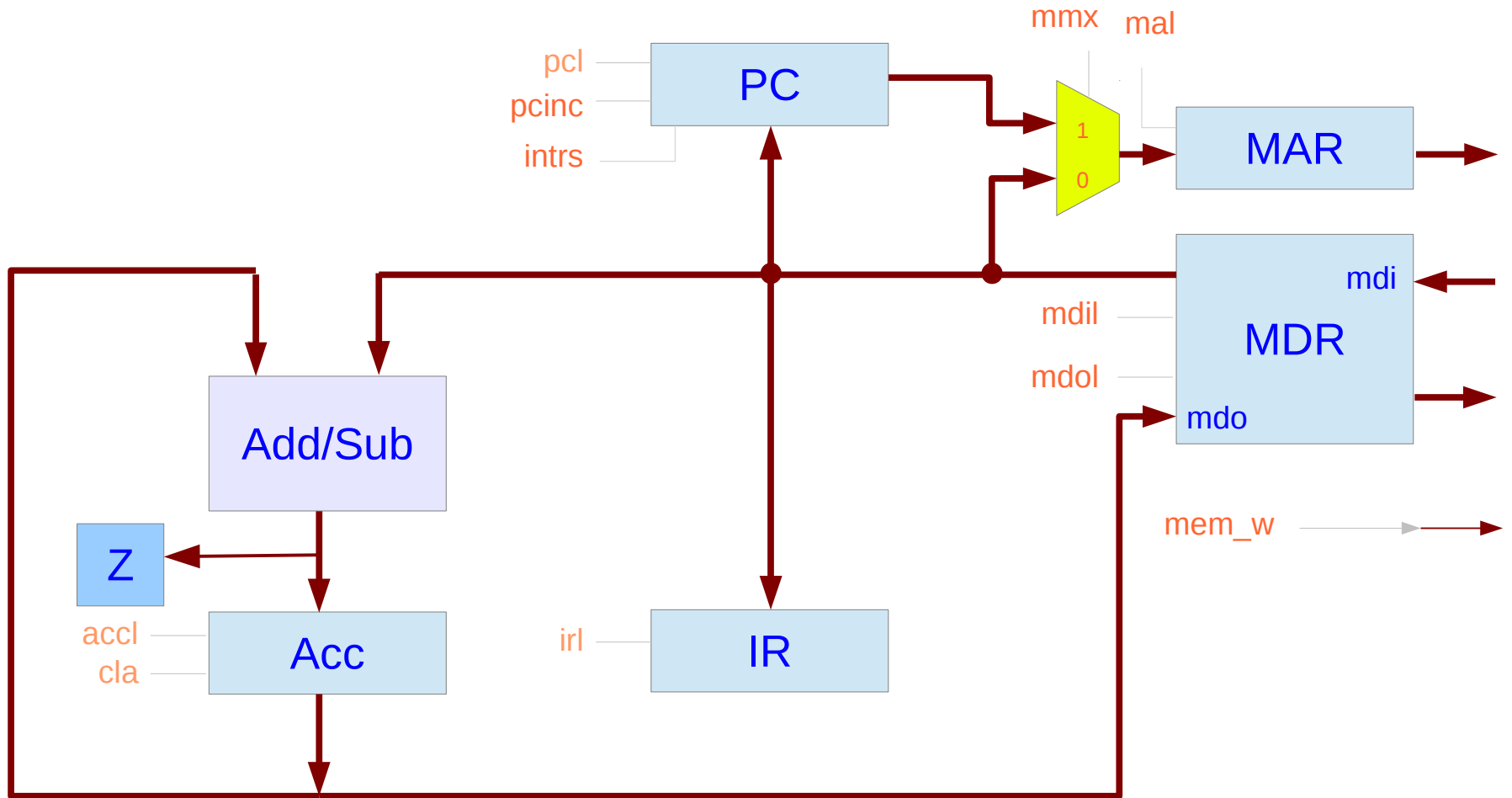
Copyright (c) 2014 - 2016 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

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

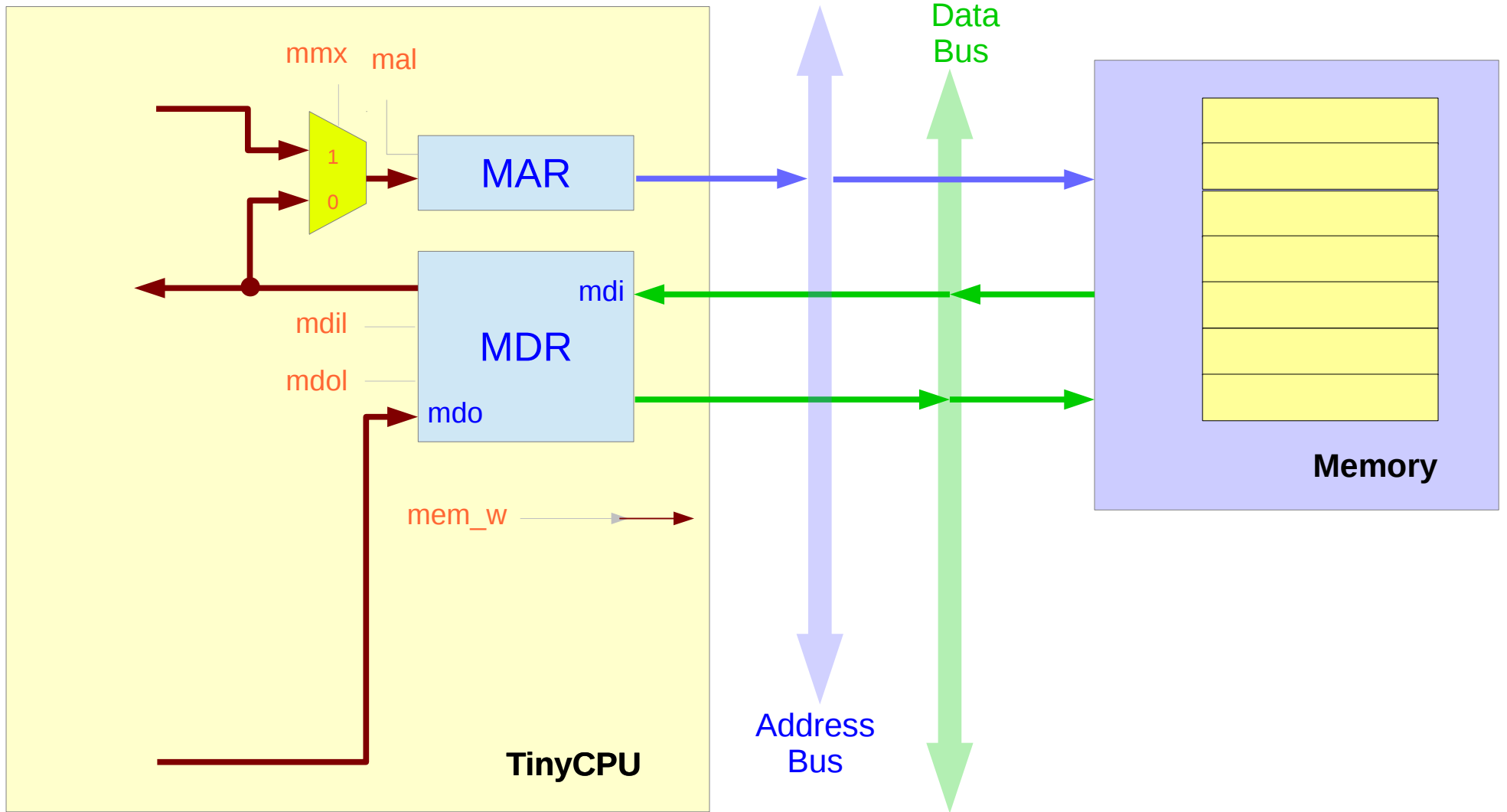
This document was produced by using LibreOffice and Octave.

Data Path



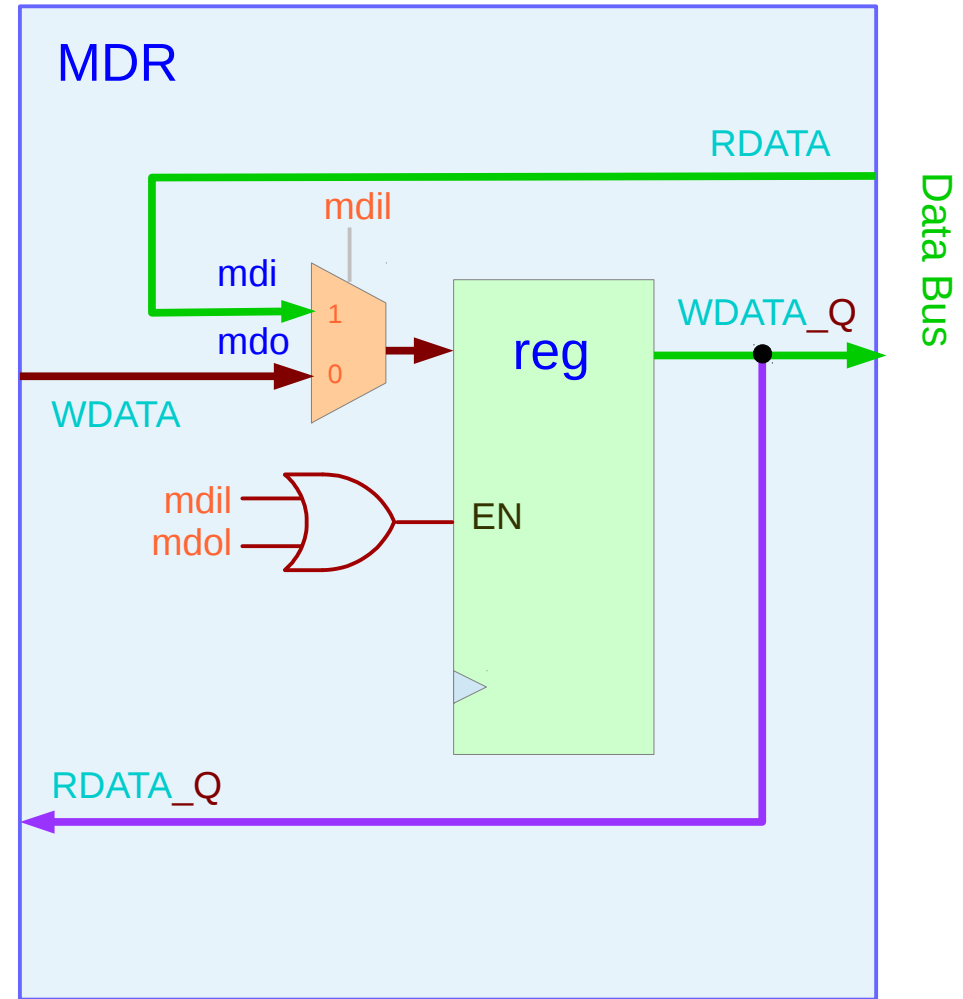
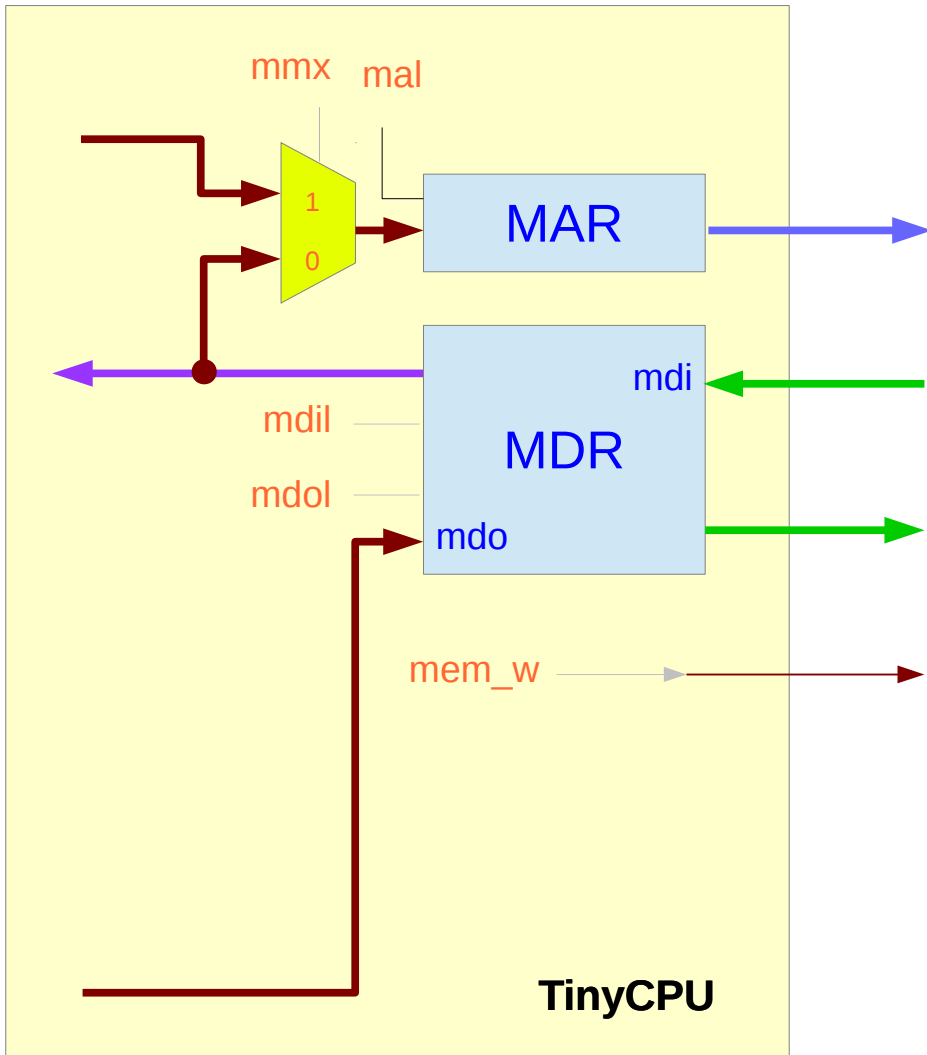
Based on <http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf>

CPU and MEM



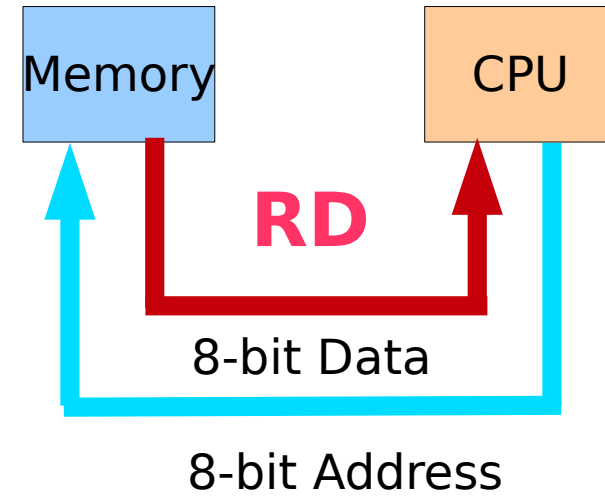
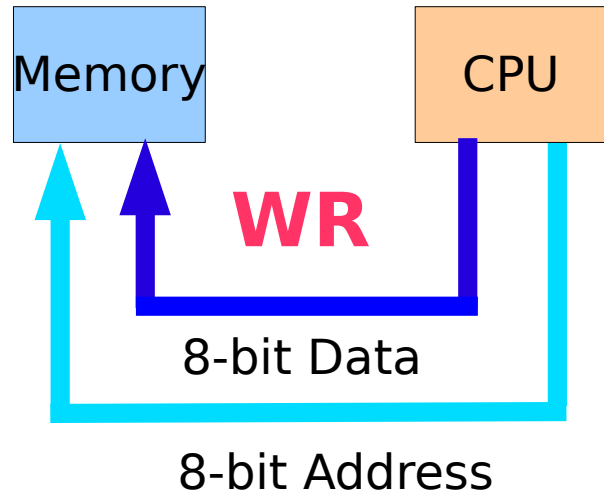
Based on <http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf>

Memory Data Register

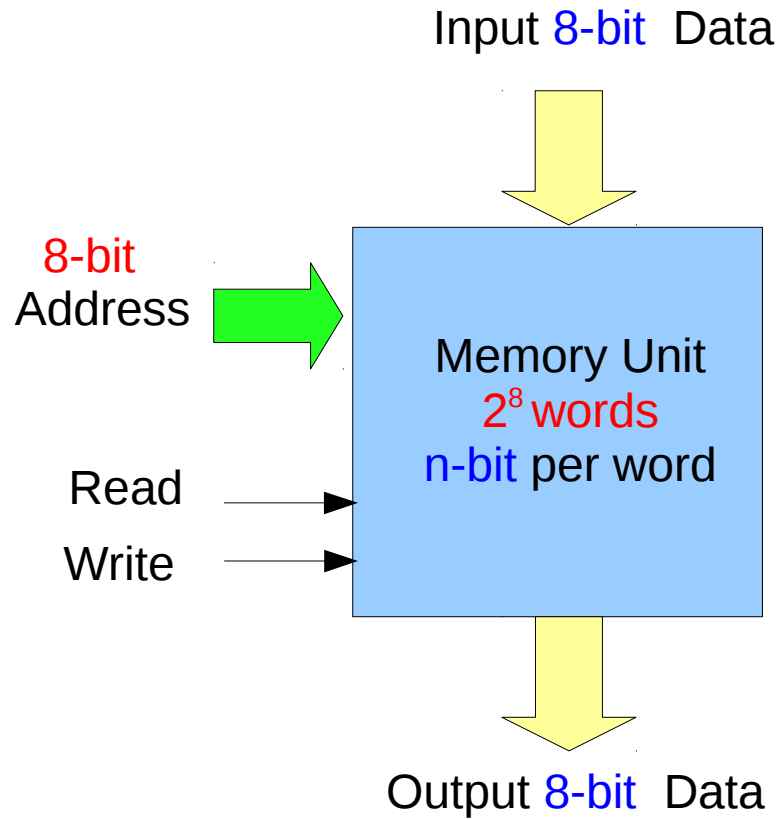


Based on <http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf>

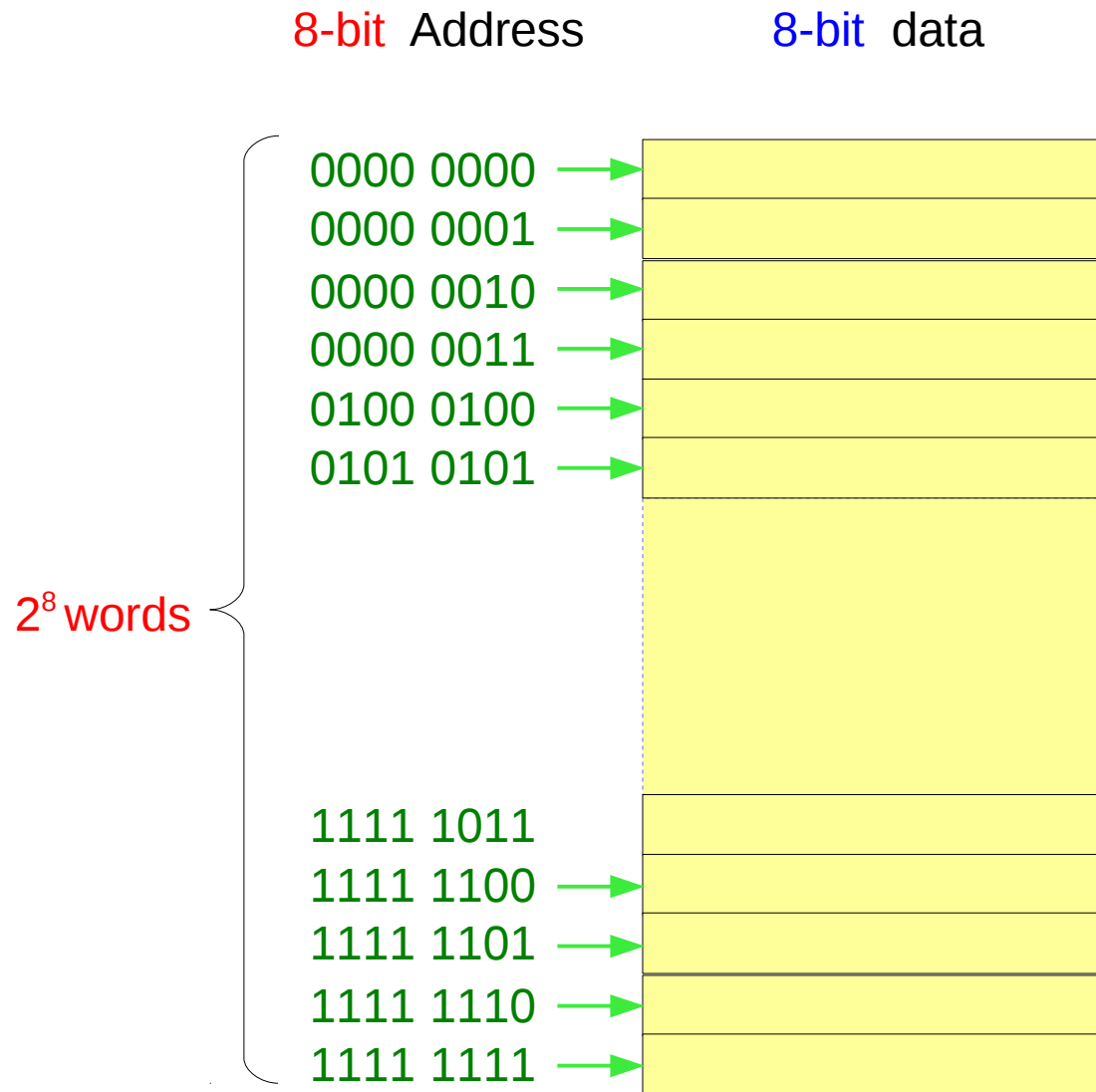
Memory Access Operations



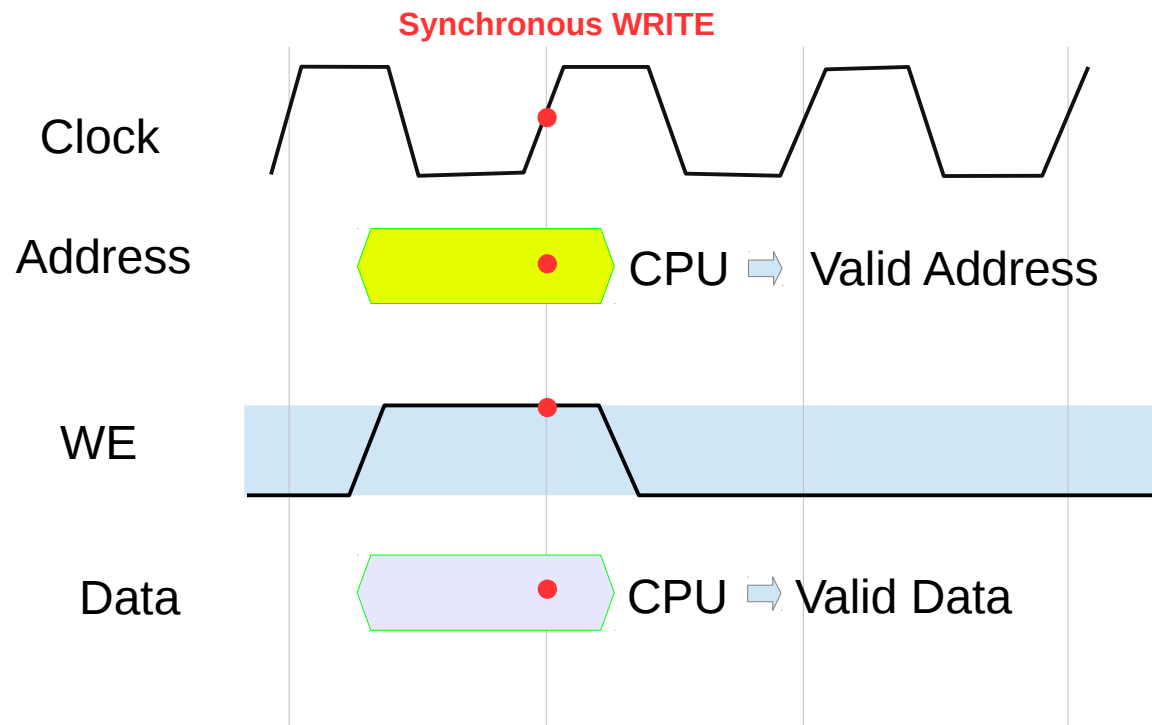
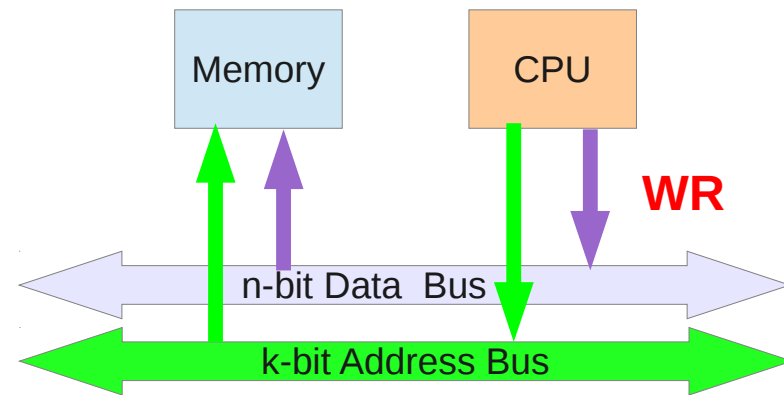
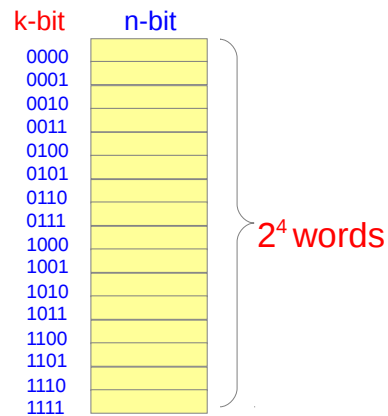
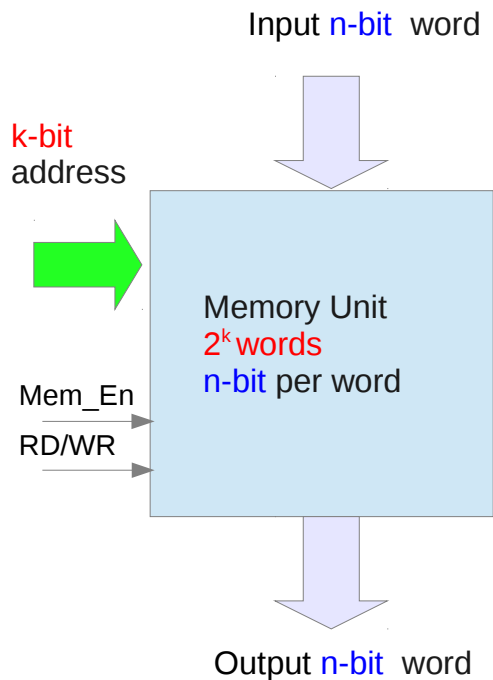
Memory Interface



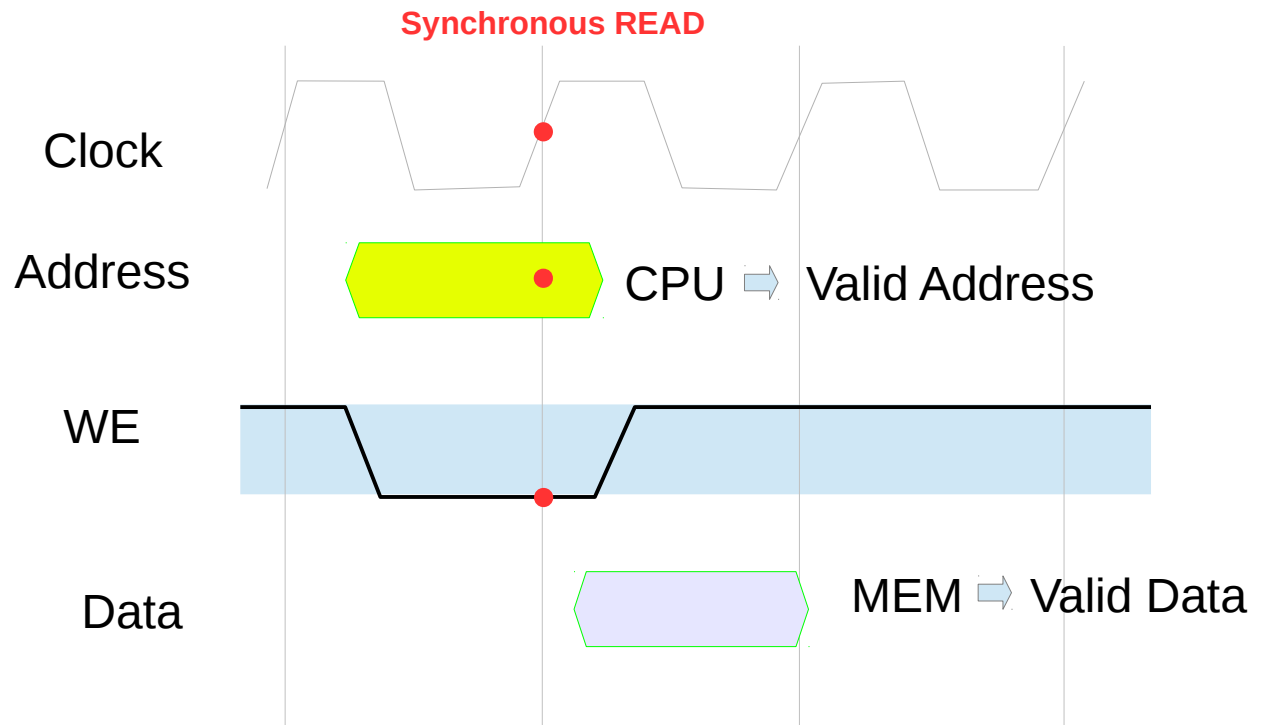
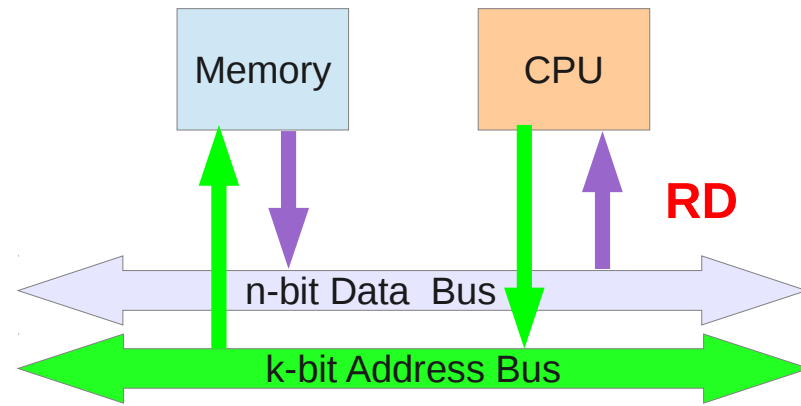
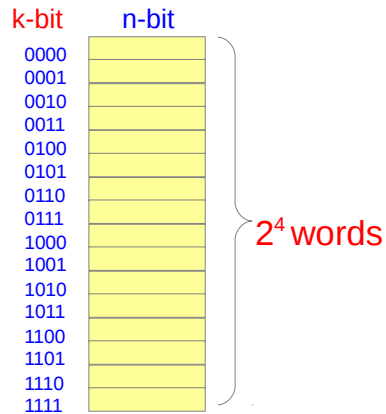
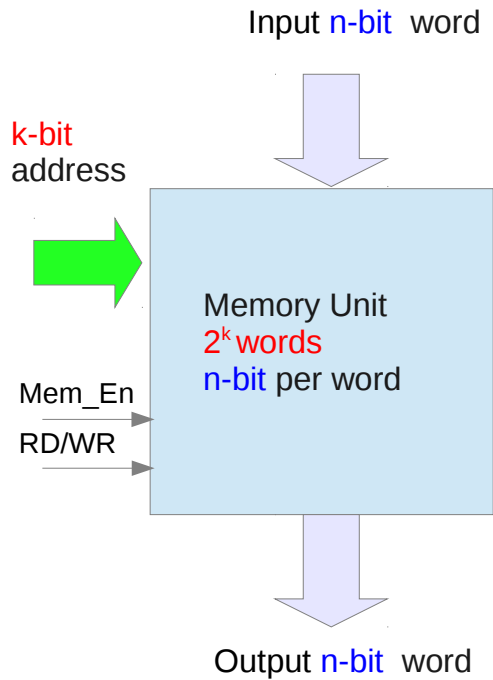
Memory Map



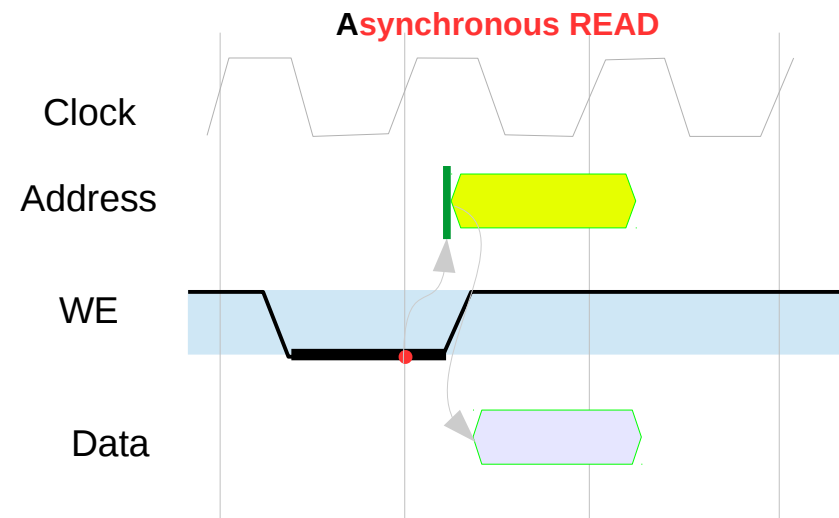
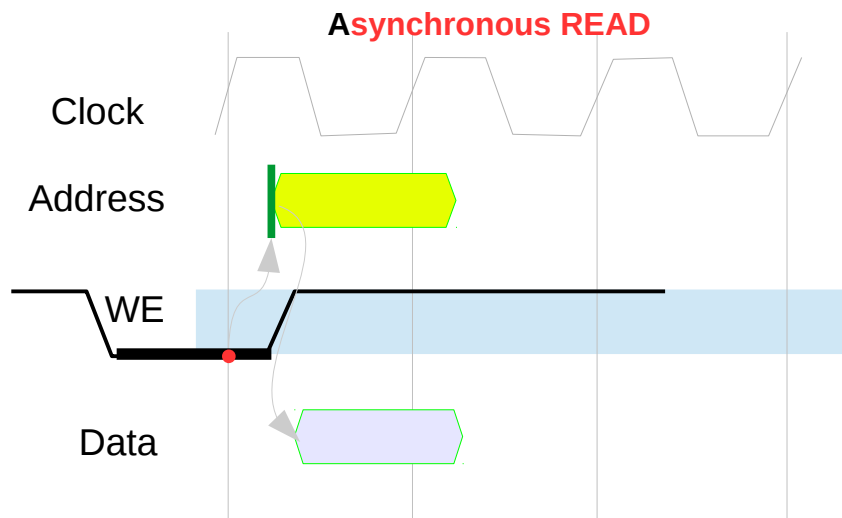
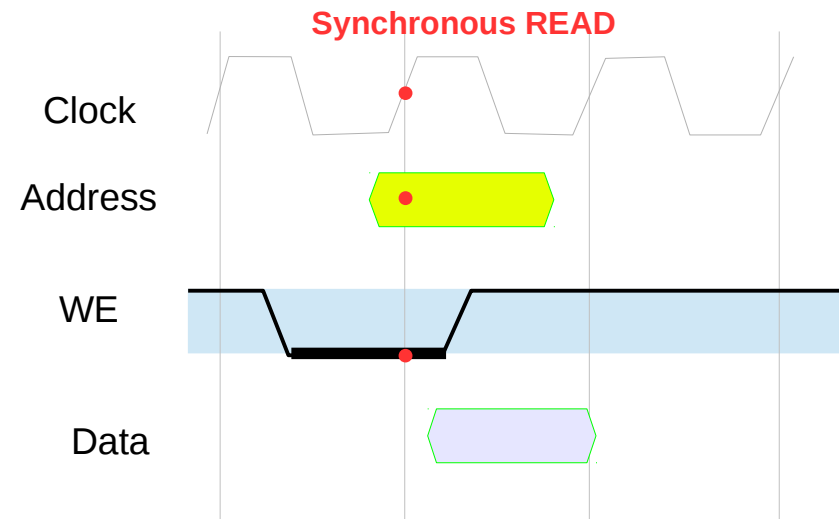
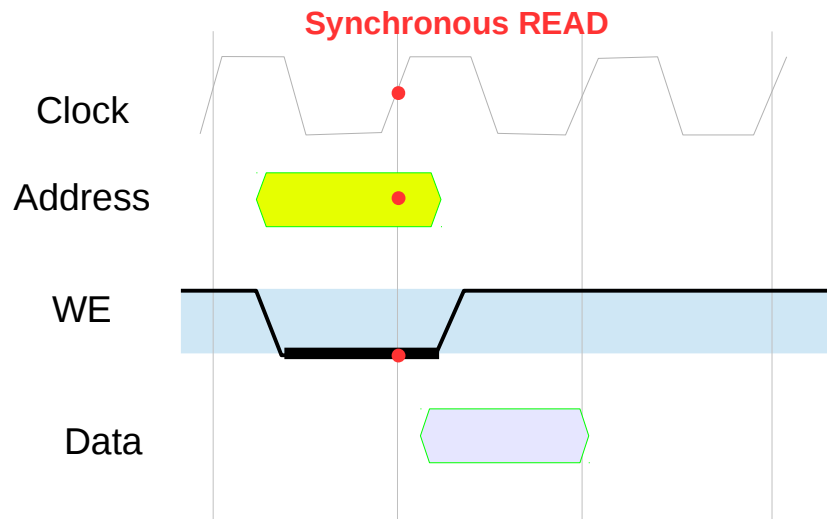
Memory Write Cycle



Memory Read Cycle



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
```

Synchronous
Write

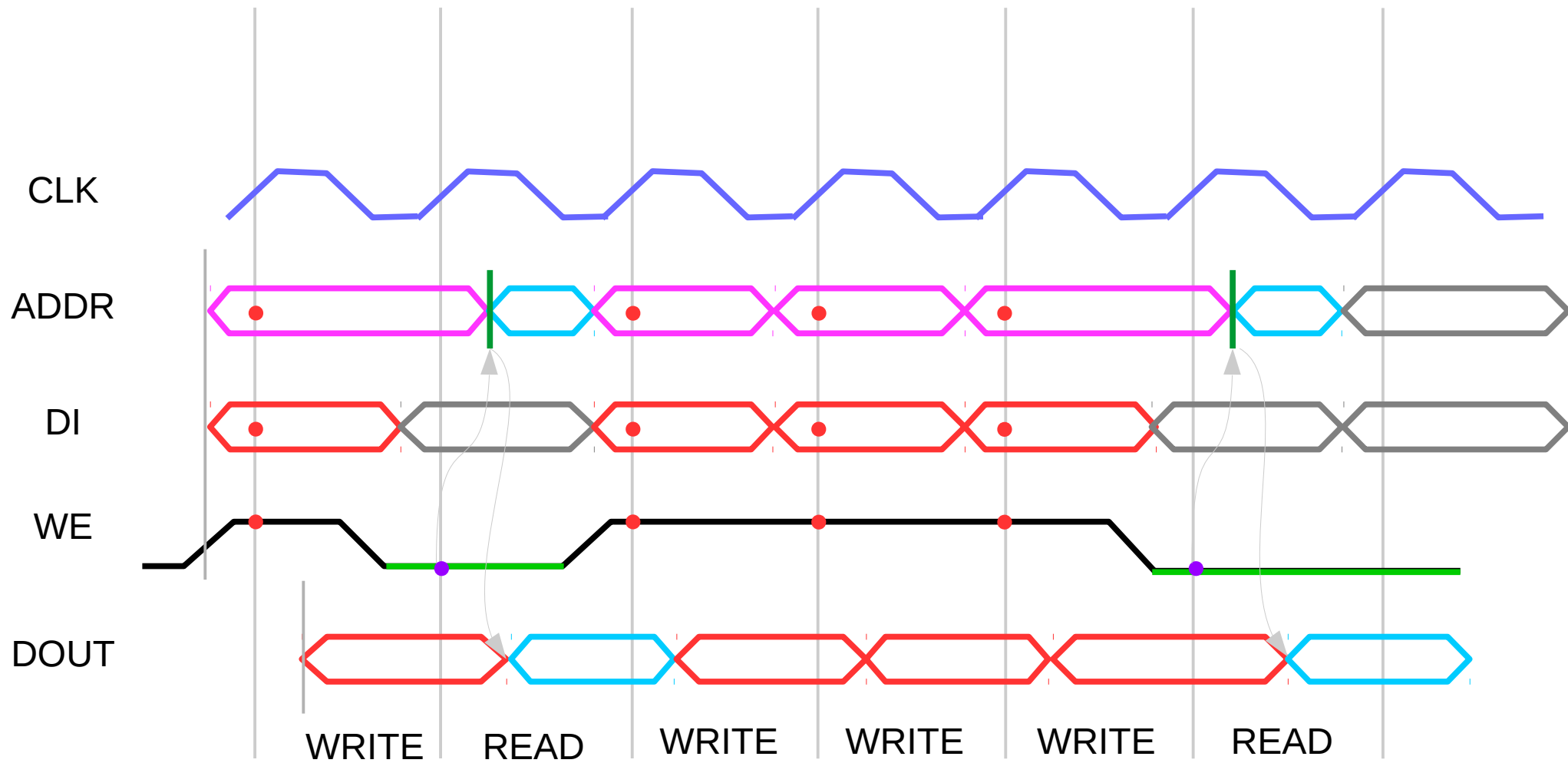
```
assign q = mem[a];
```

Asynchronous
Read

```
endmodule
```

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

LUT RAM Timing

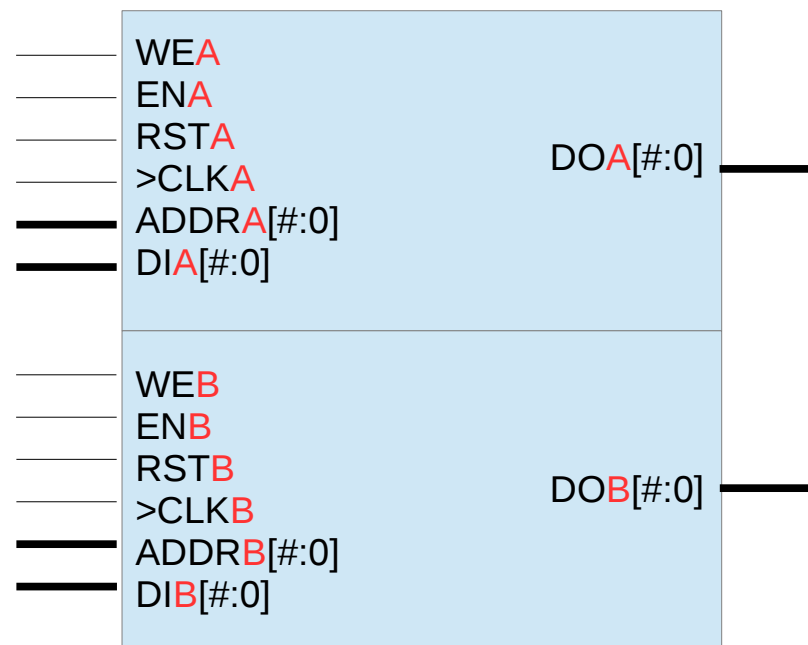


Xilinx Document

Virtex LUT RAM

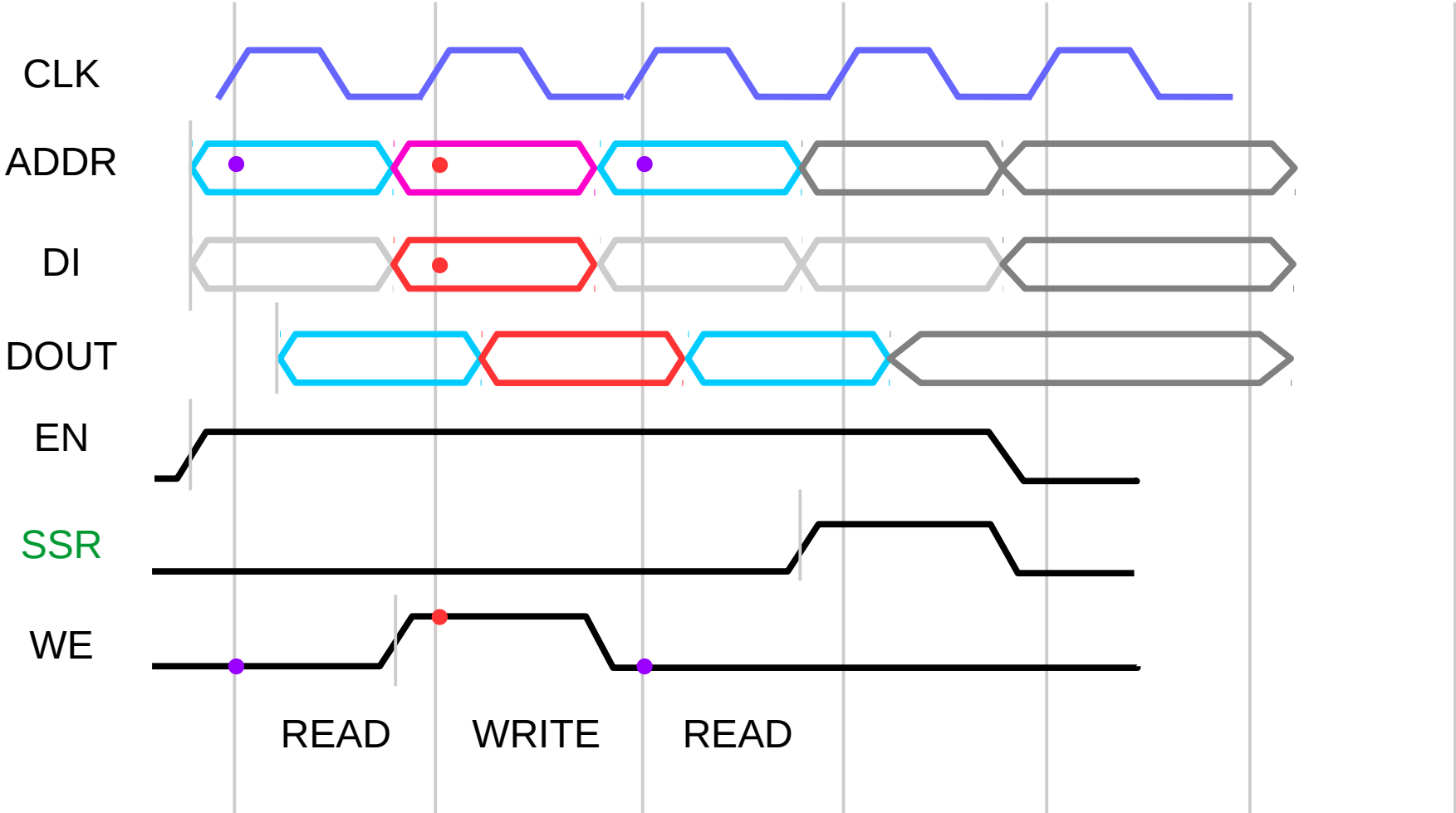
Every block SelectRAM

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



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

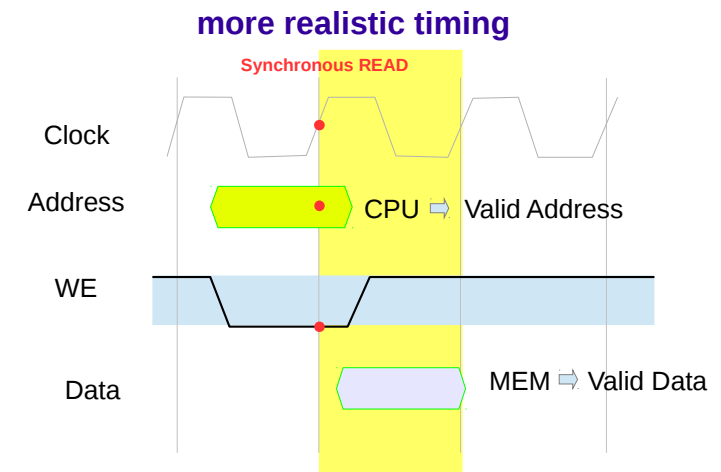
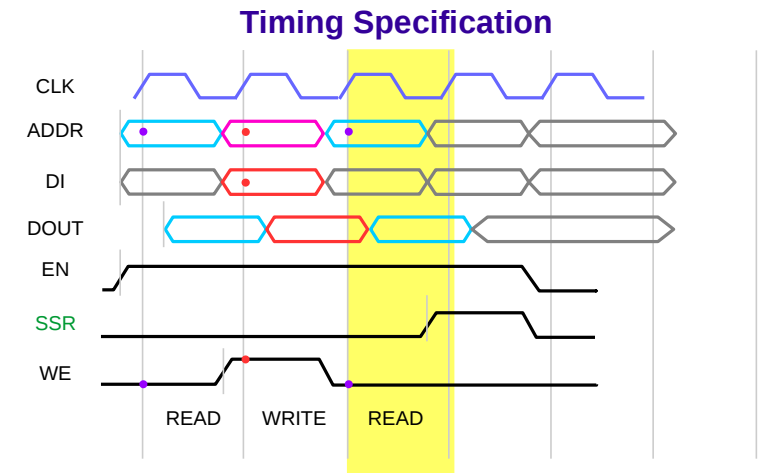
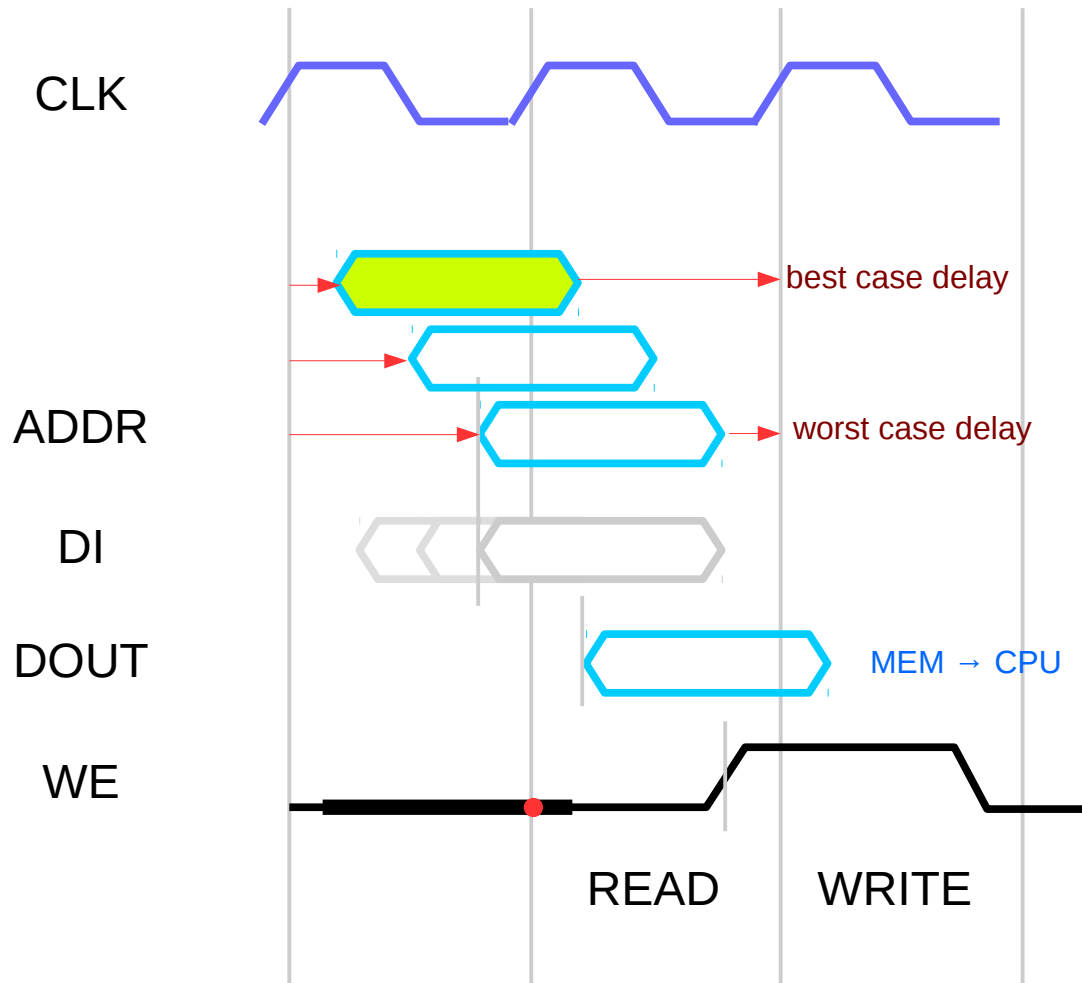
Virtex Block RAM Timing



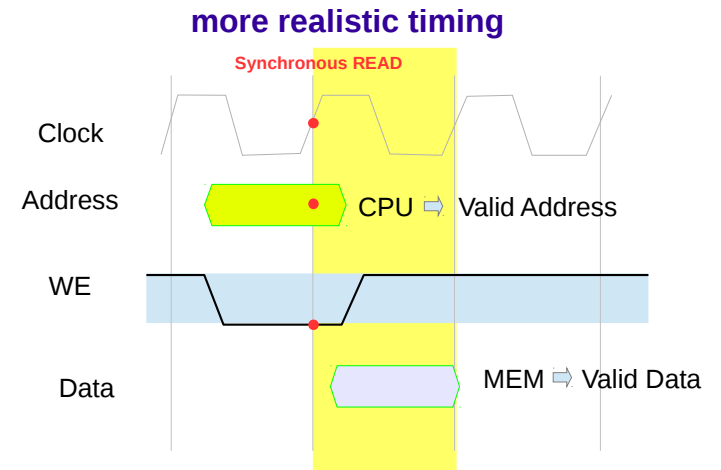
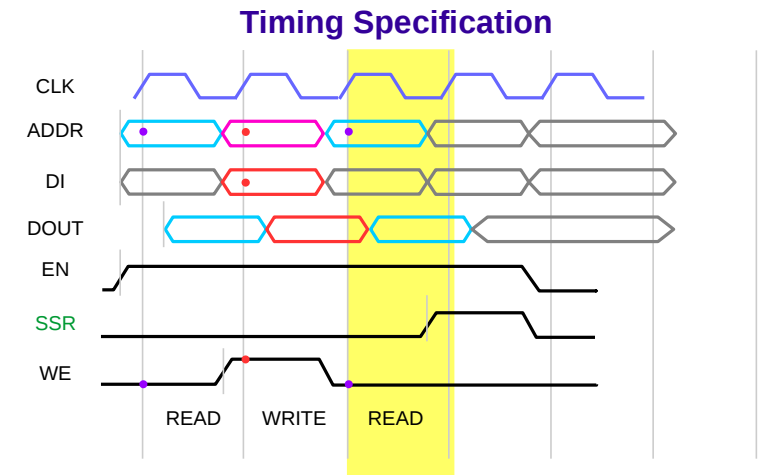
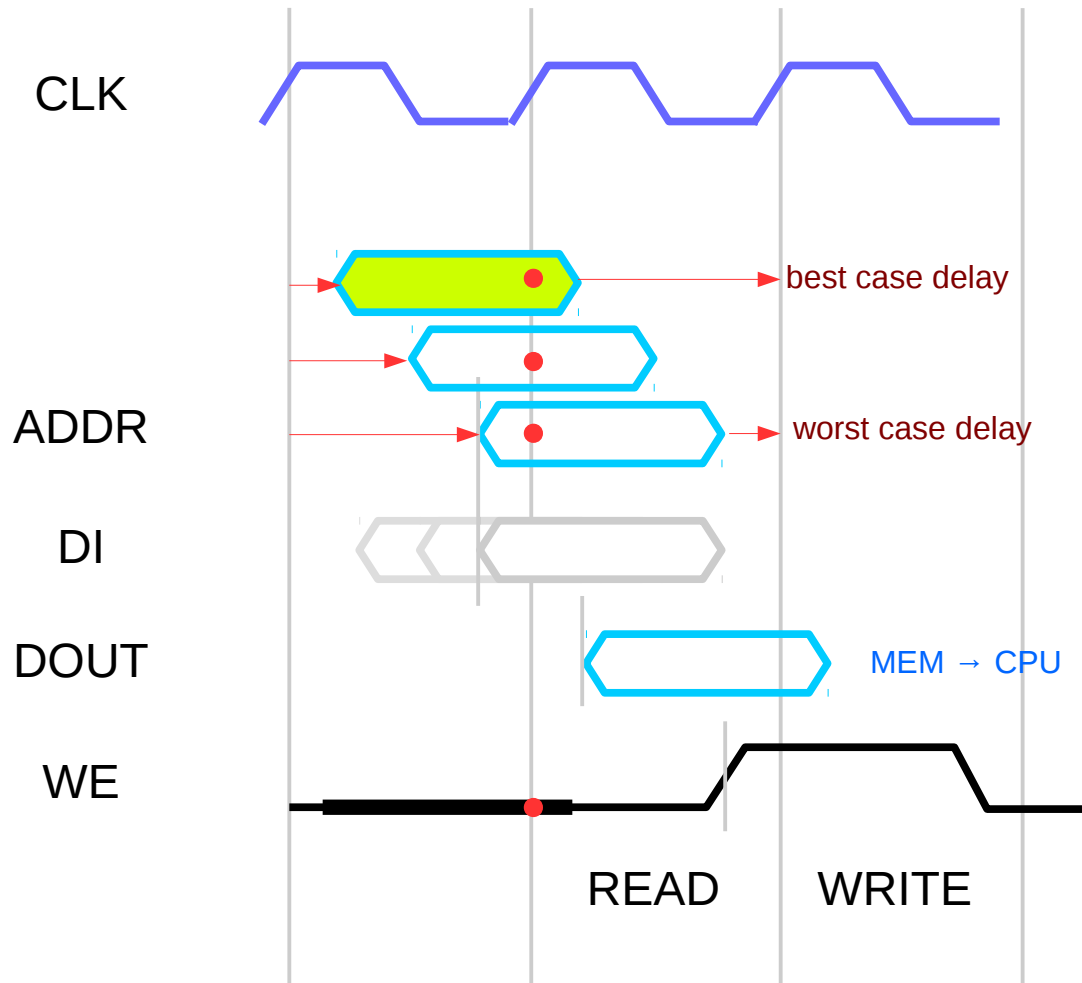
Xilinx Document

SSR (Synchronous Set Rest) value = 1010

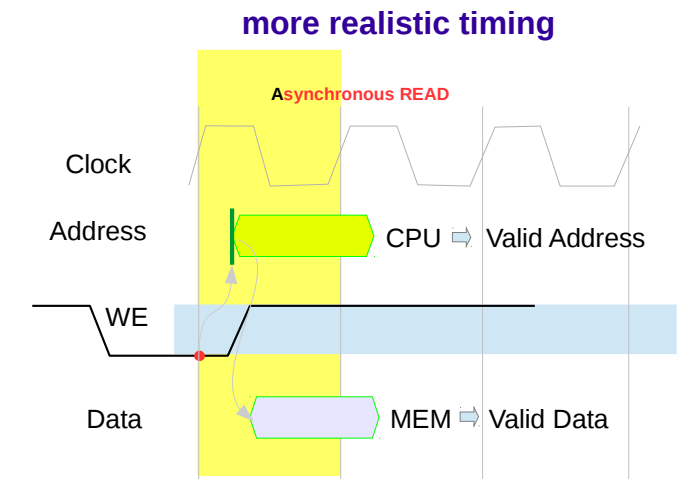
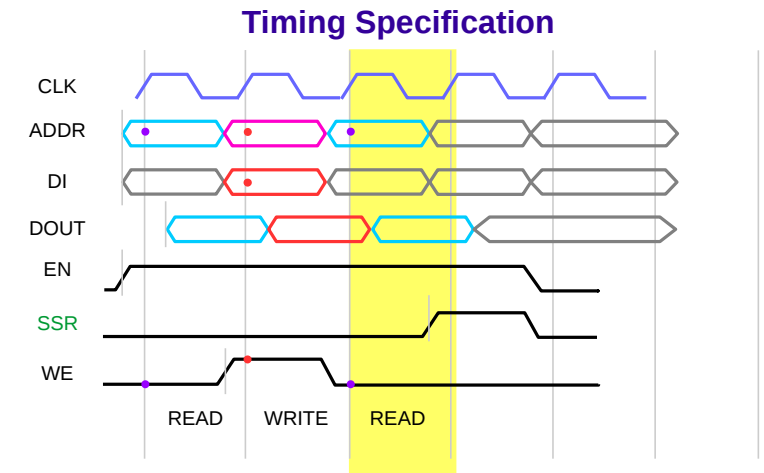
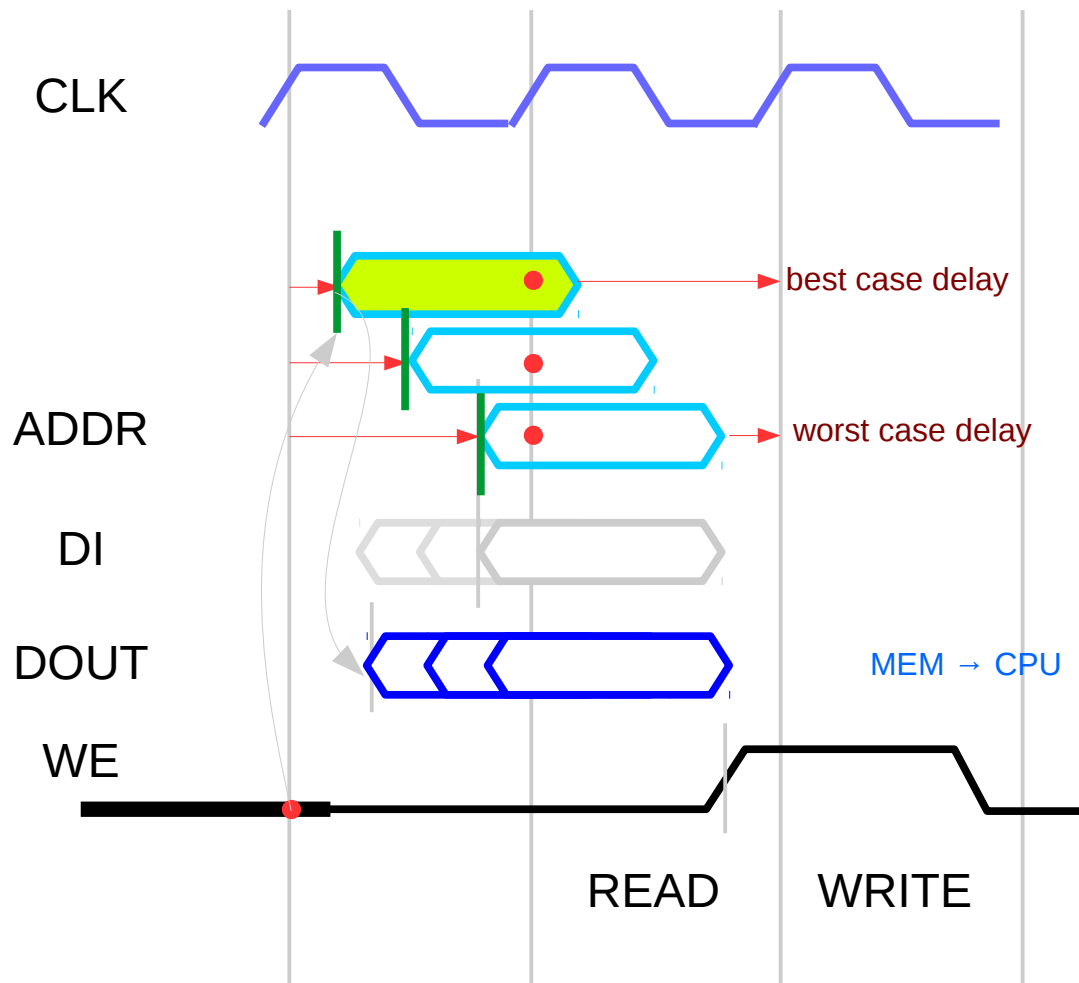
Sync **RD** Cycle - worst & best case



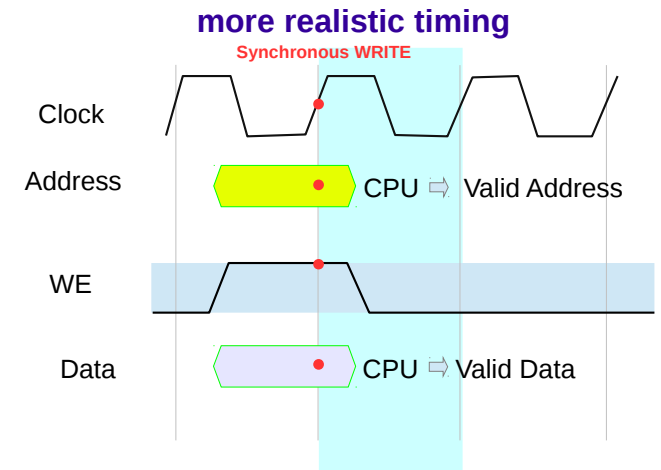
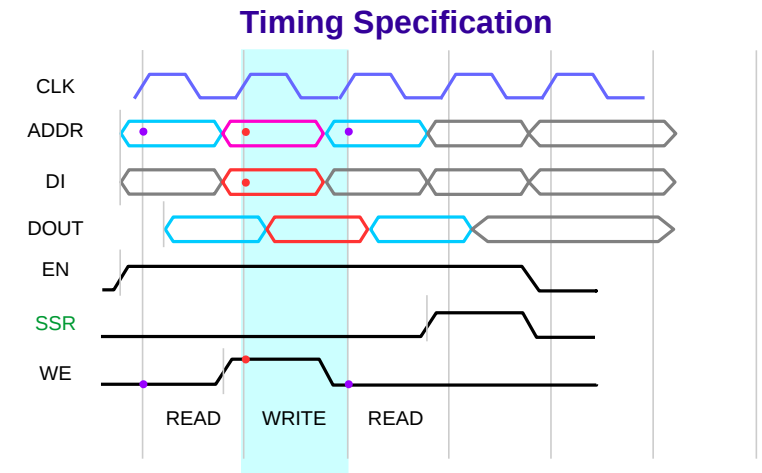
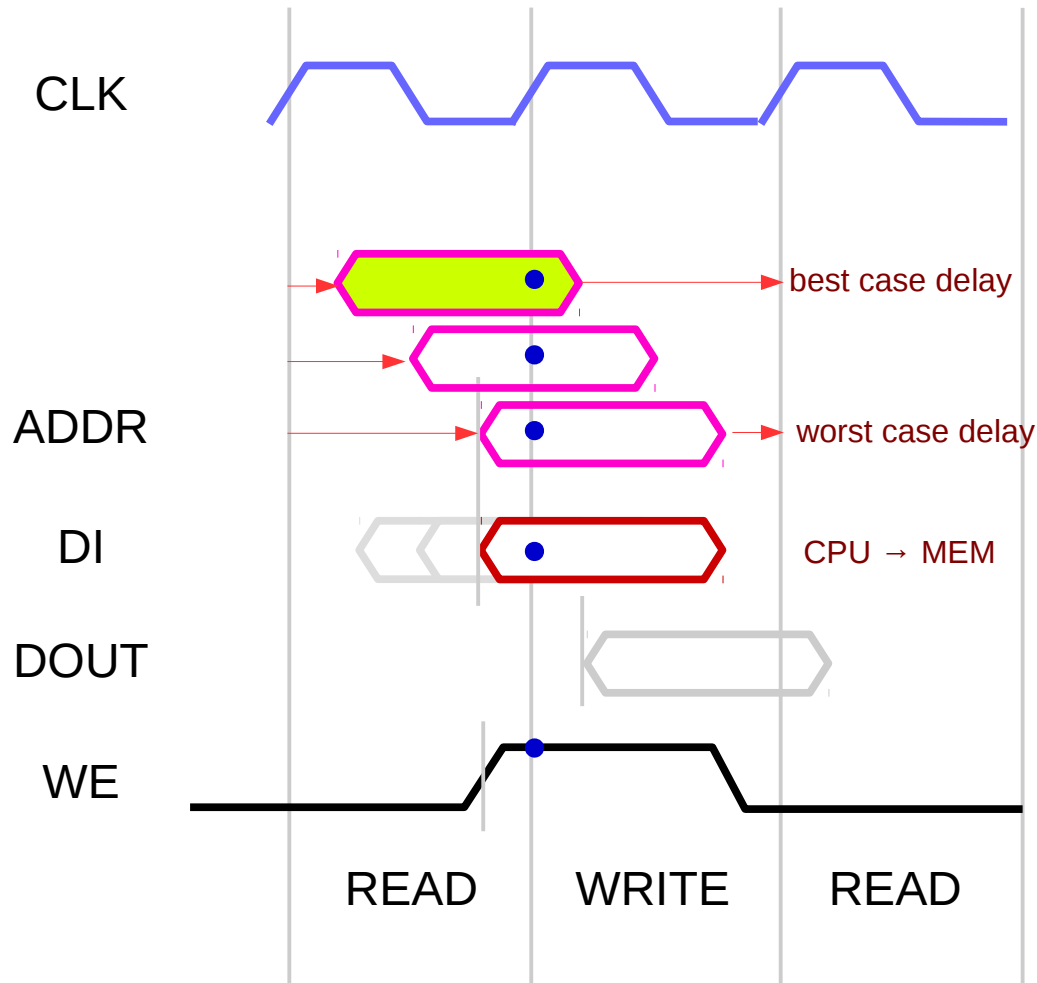
Sync **RD** Cycle - worst & best case



Async RD Cycle - worst & best case



WR Cycle - worst & best case



Waveform Viewer Timing (1)

* timing figures without delay (Ideal case)

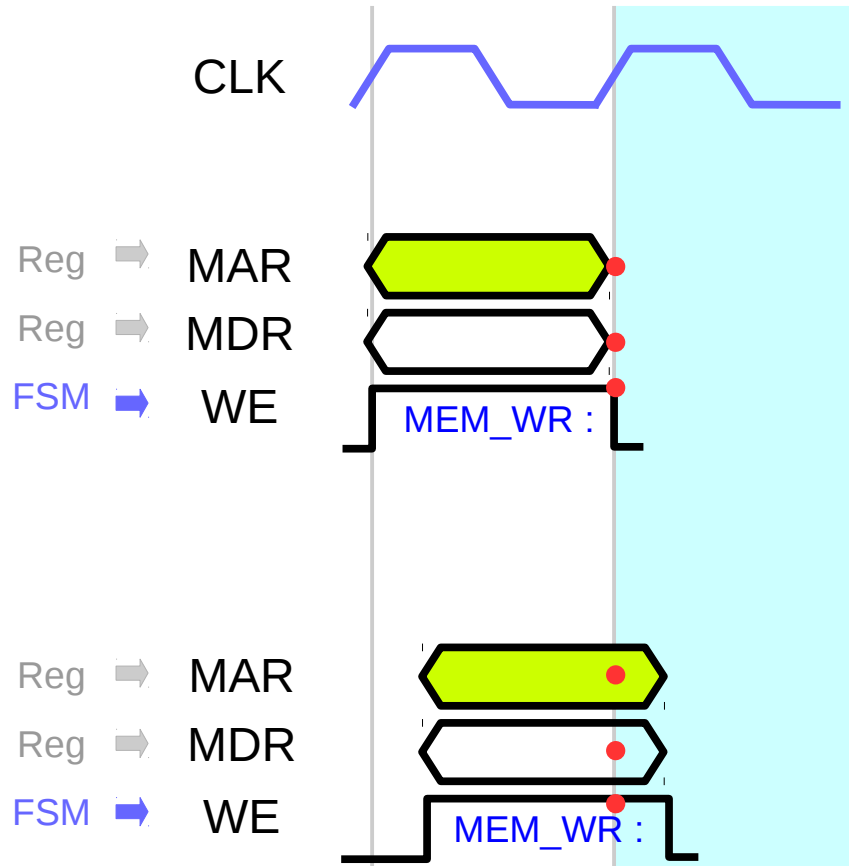
RTL functional simulation

* timing figures with delay

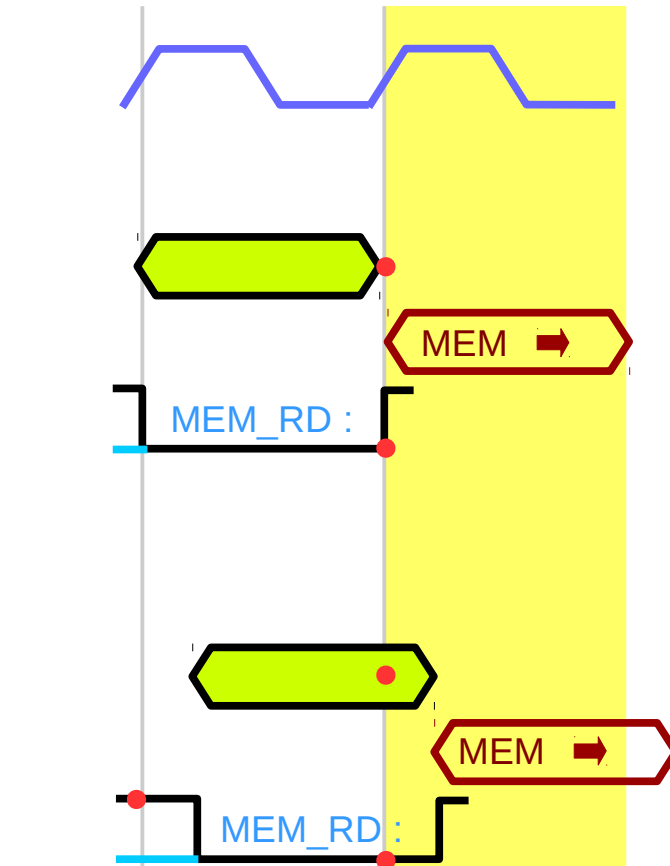
Gate level simulation
with SDF annotation

Waveform Viewer Timing (2)

Sync WRITE



Sync READ



timing figures without delay (Ideal case)

timing figures with delay

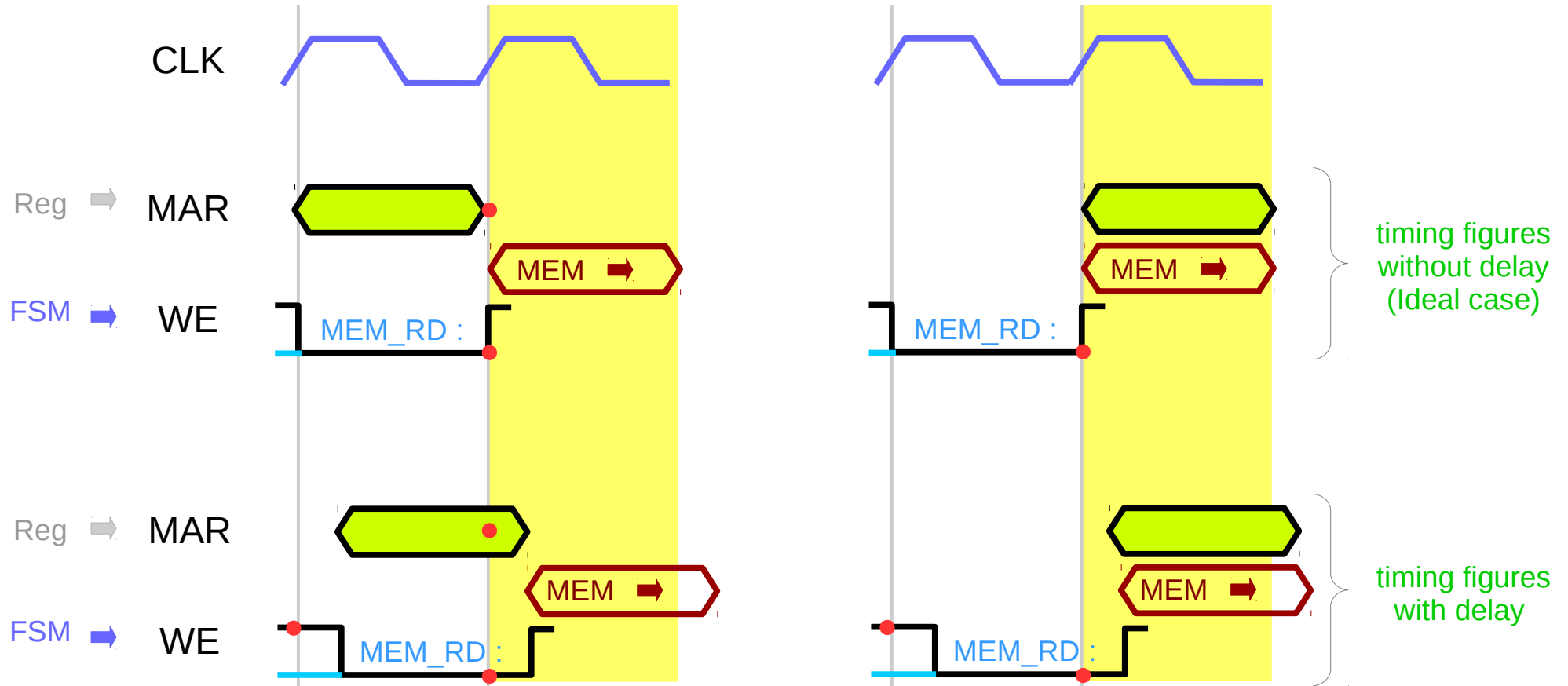
unintended write cycle!

MEM_RD: One cycle ahead

Waveform Viewer Timing (3)

Sync READ

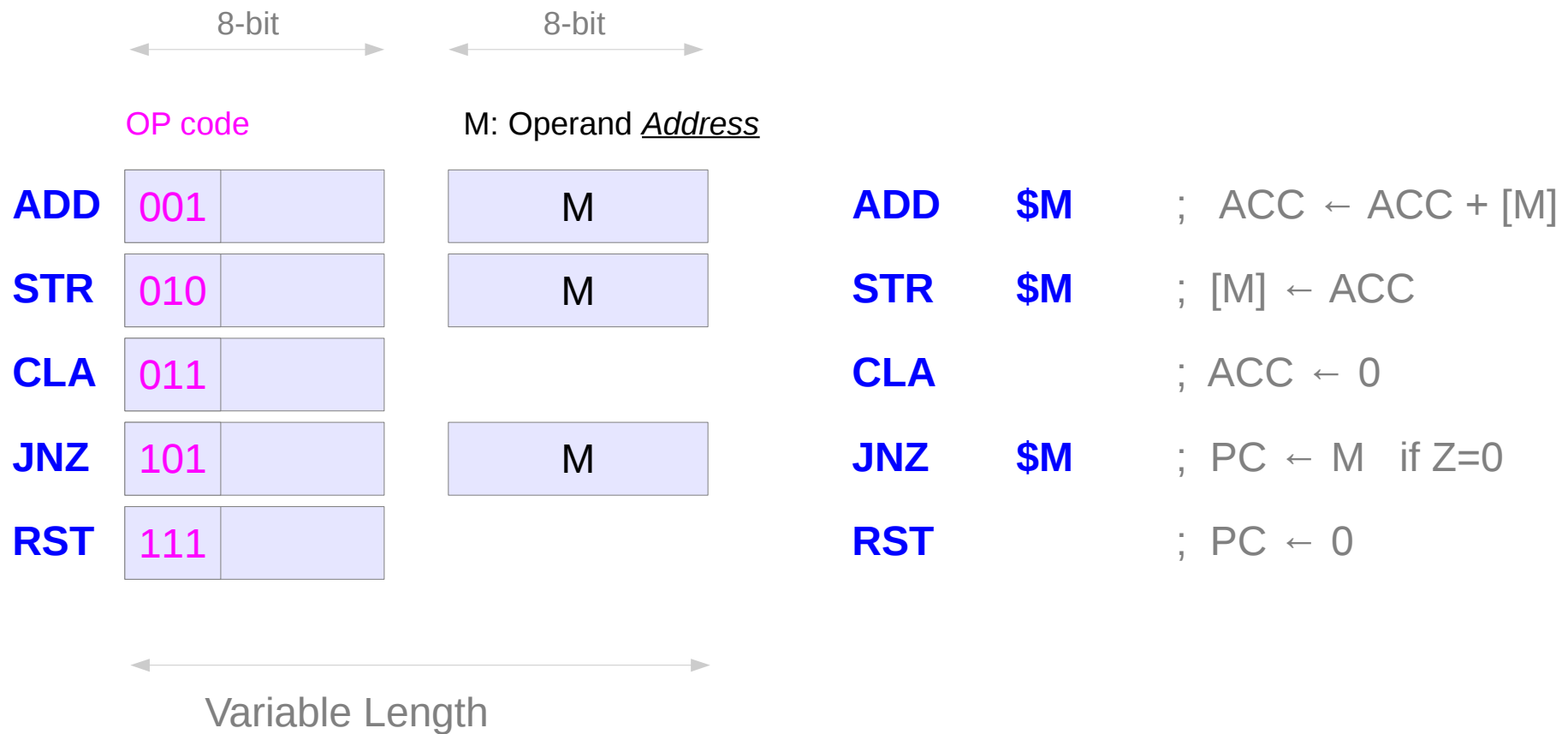
Async READ



unintended write cycle!

MEM_RD :
One cycle ahead

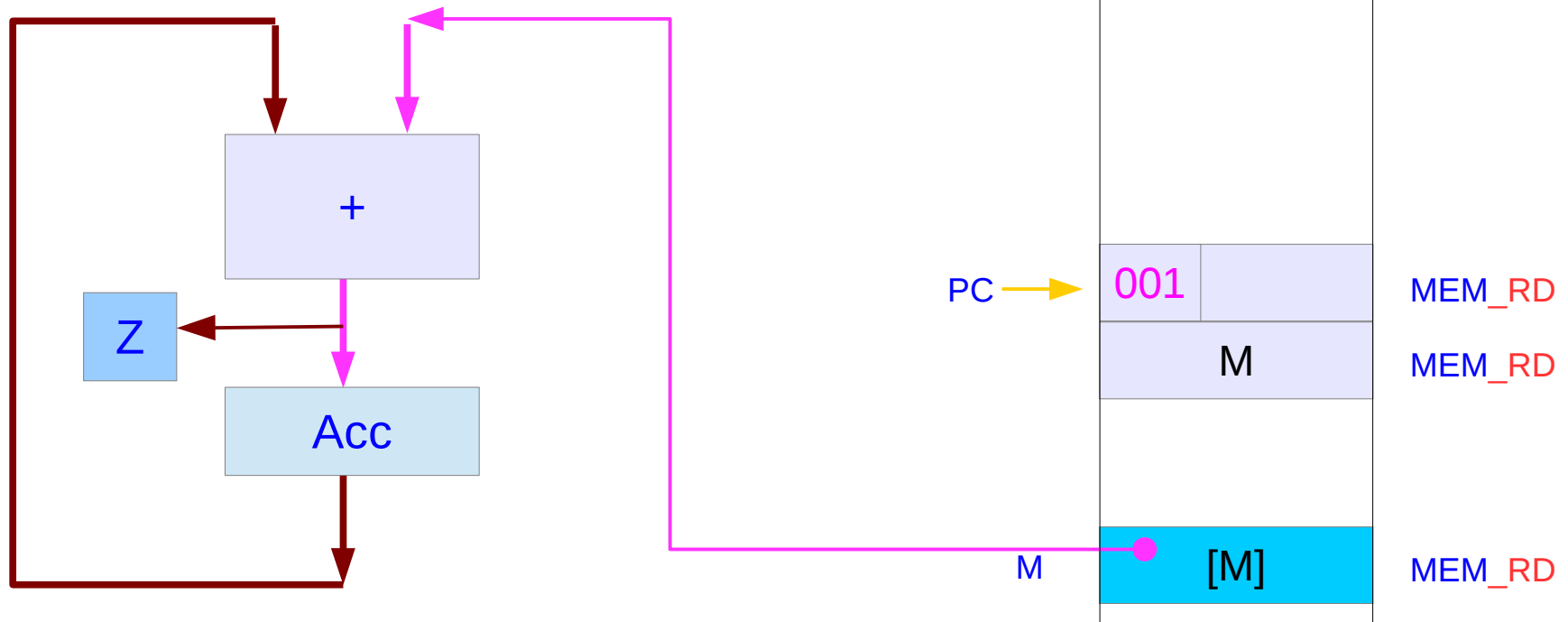
Instruction Set Architecture



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

ADD Operation

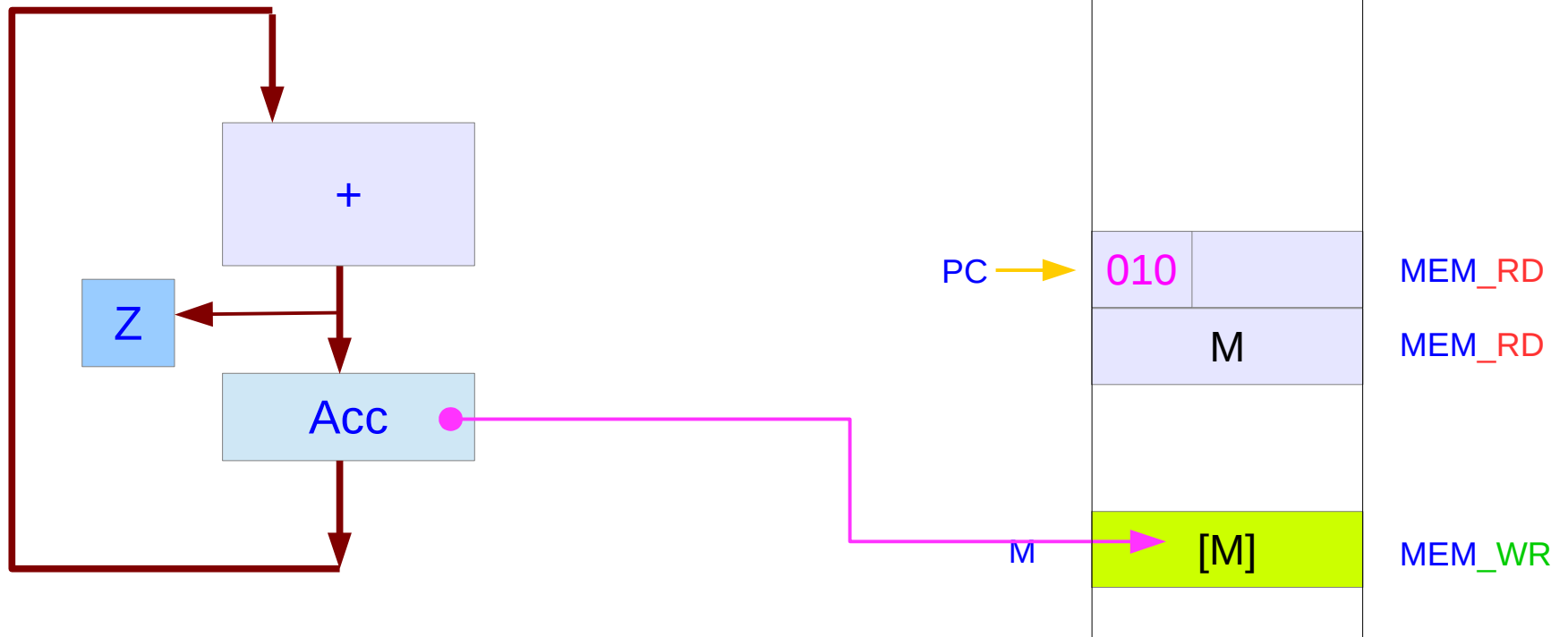
ADD **\$M** ; $ACC \leftarrow ACC + [M]$



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

STR Operation

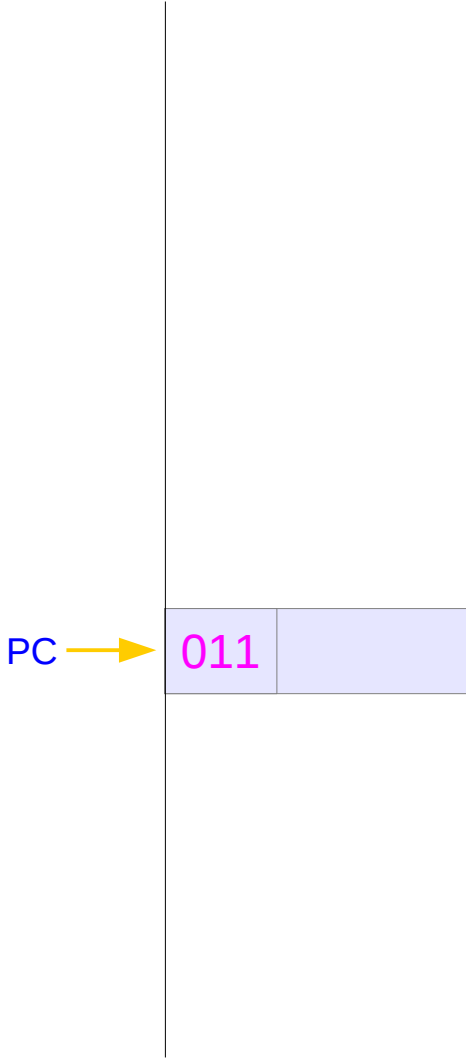
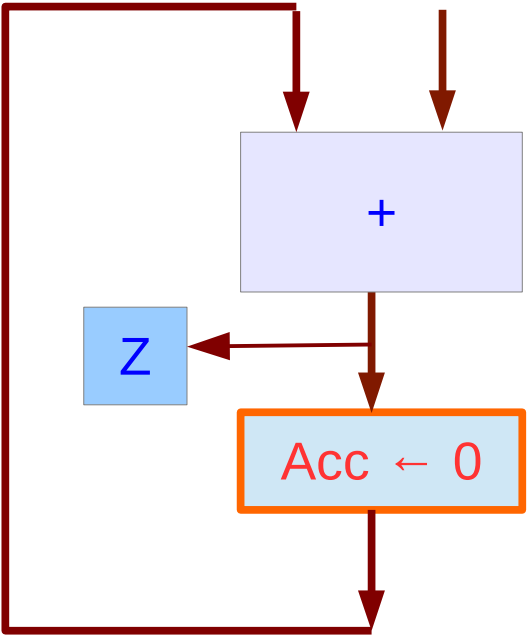
STR **\$M** ; [M] ← ACC



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

CLA Operation

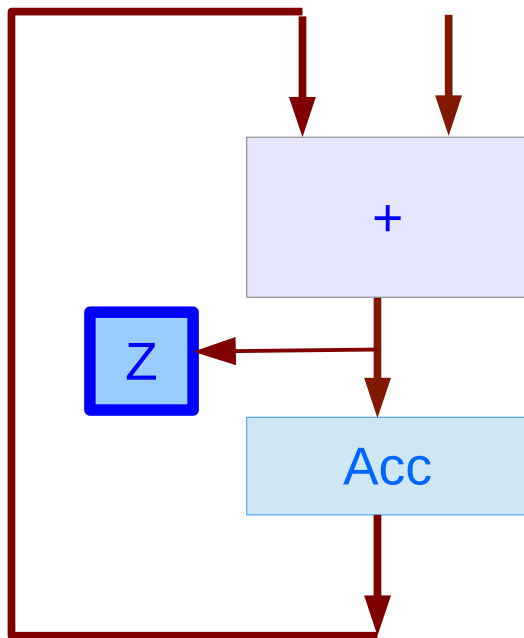
```
CLA ; ACC ← 0
```



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

JNZ Operation

JNZ \$M ; PC ← M if Z=0



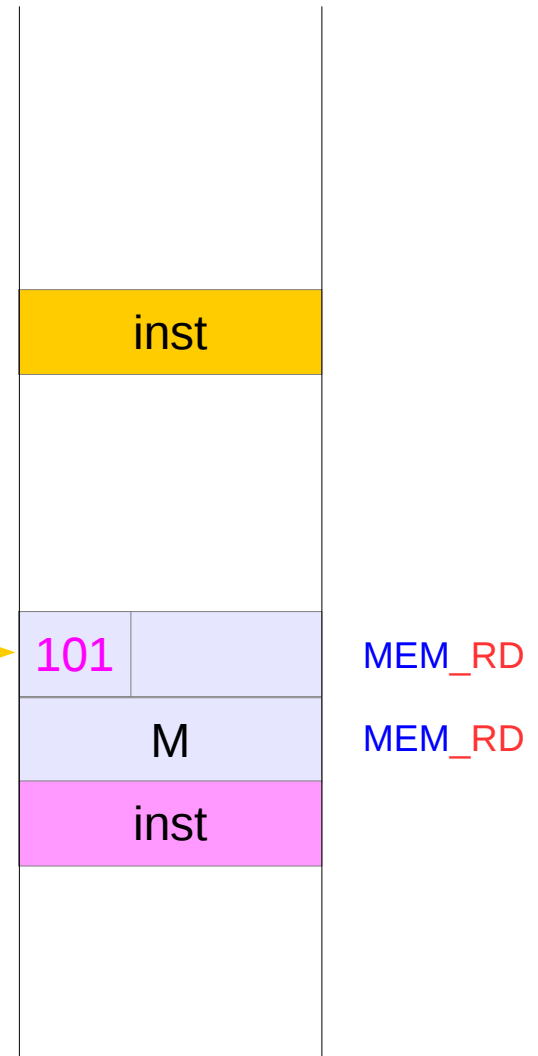
If ACC is **not** zero
flag Z=0

If ACC is **zero**
flag Z=1

New PC → M

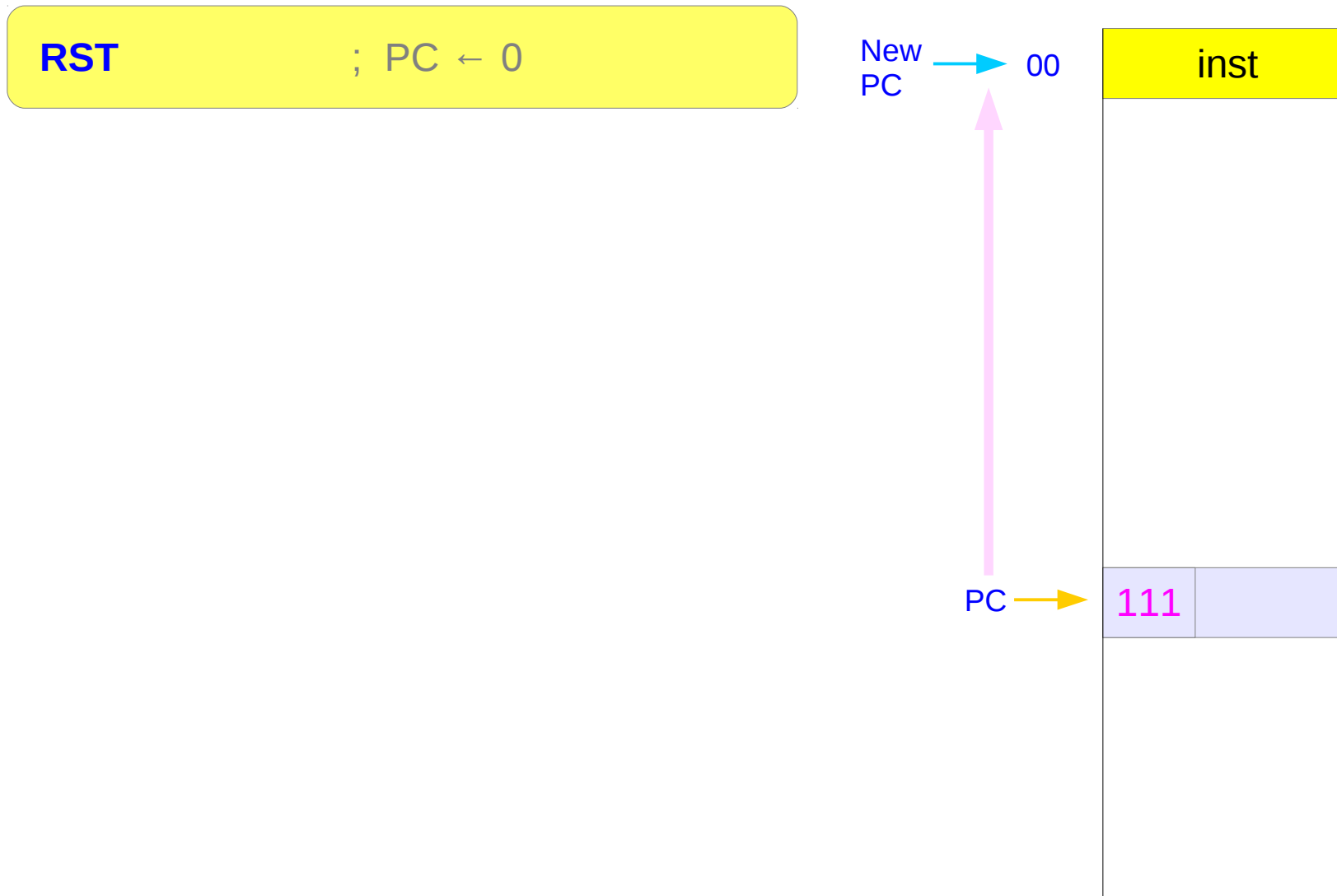
PC →

New PC →



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

RST Operation

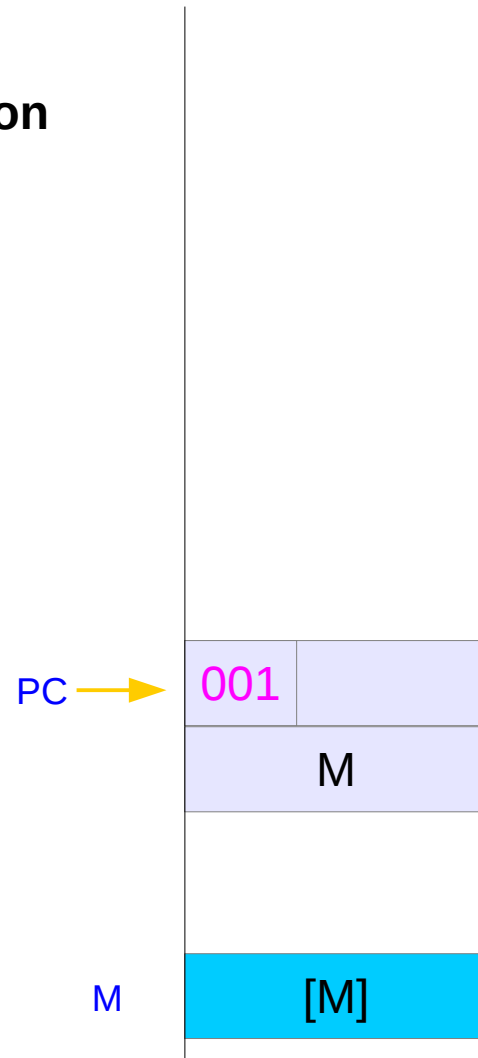


Based on <http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf>

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] →

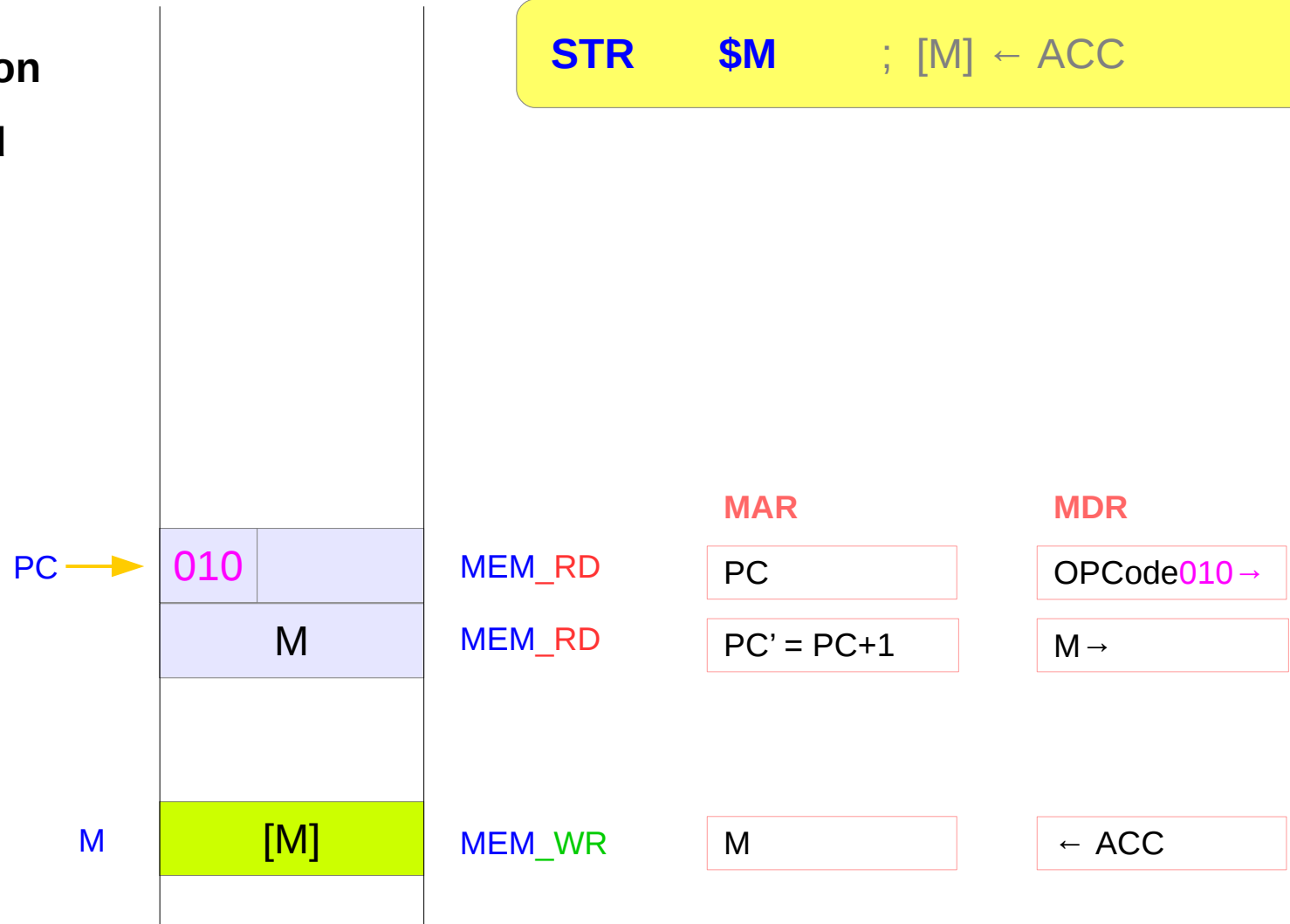
Based on <http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf>

MEM Access STR Instruction

2 RD – Instruction

1 WR – Operand

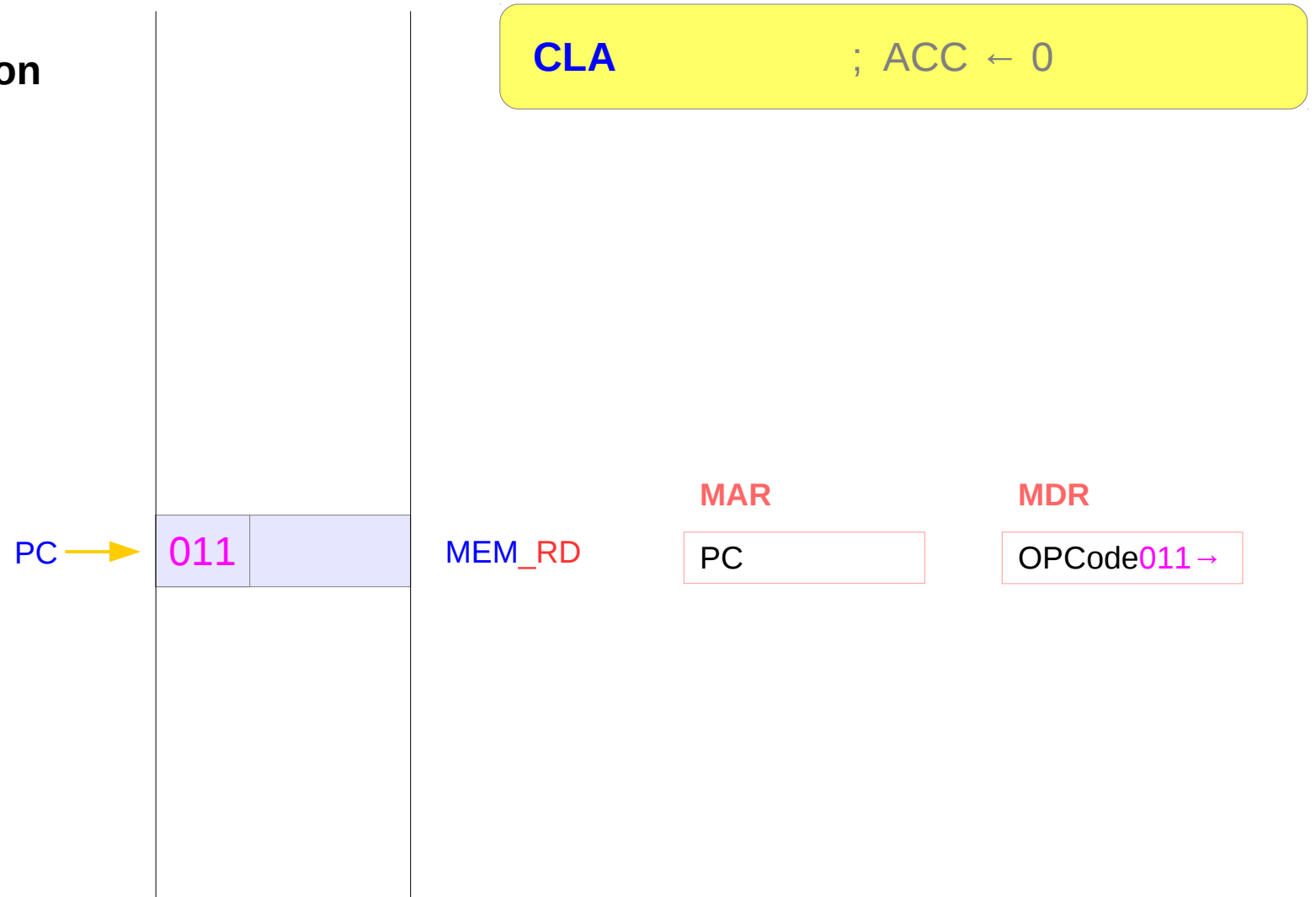
STR \$M ; [M] ← ACC



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

MEM Access CLA Instruction

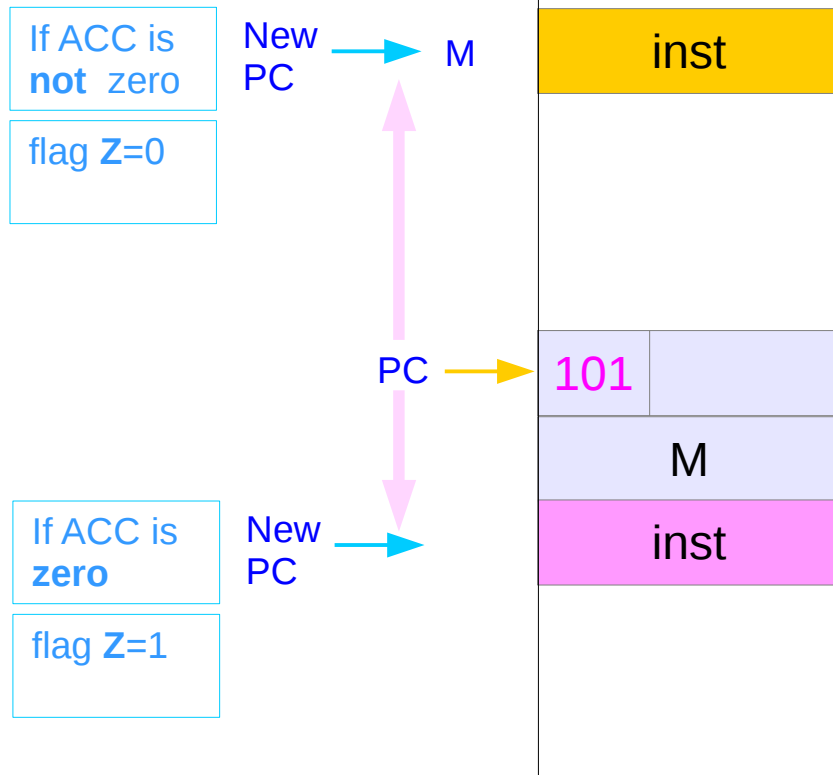
1 RD – Instruction



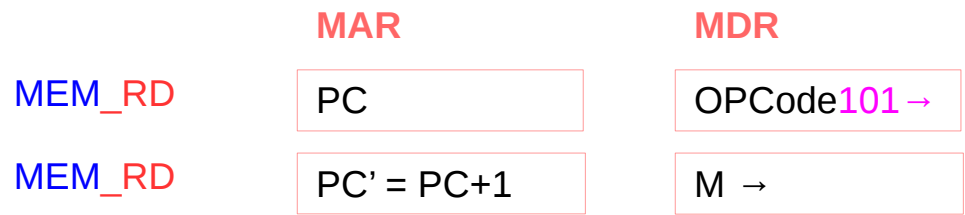
Based on <http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf>

MEM Access JNZ Instruction

2 RD – Instruction



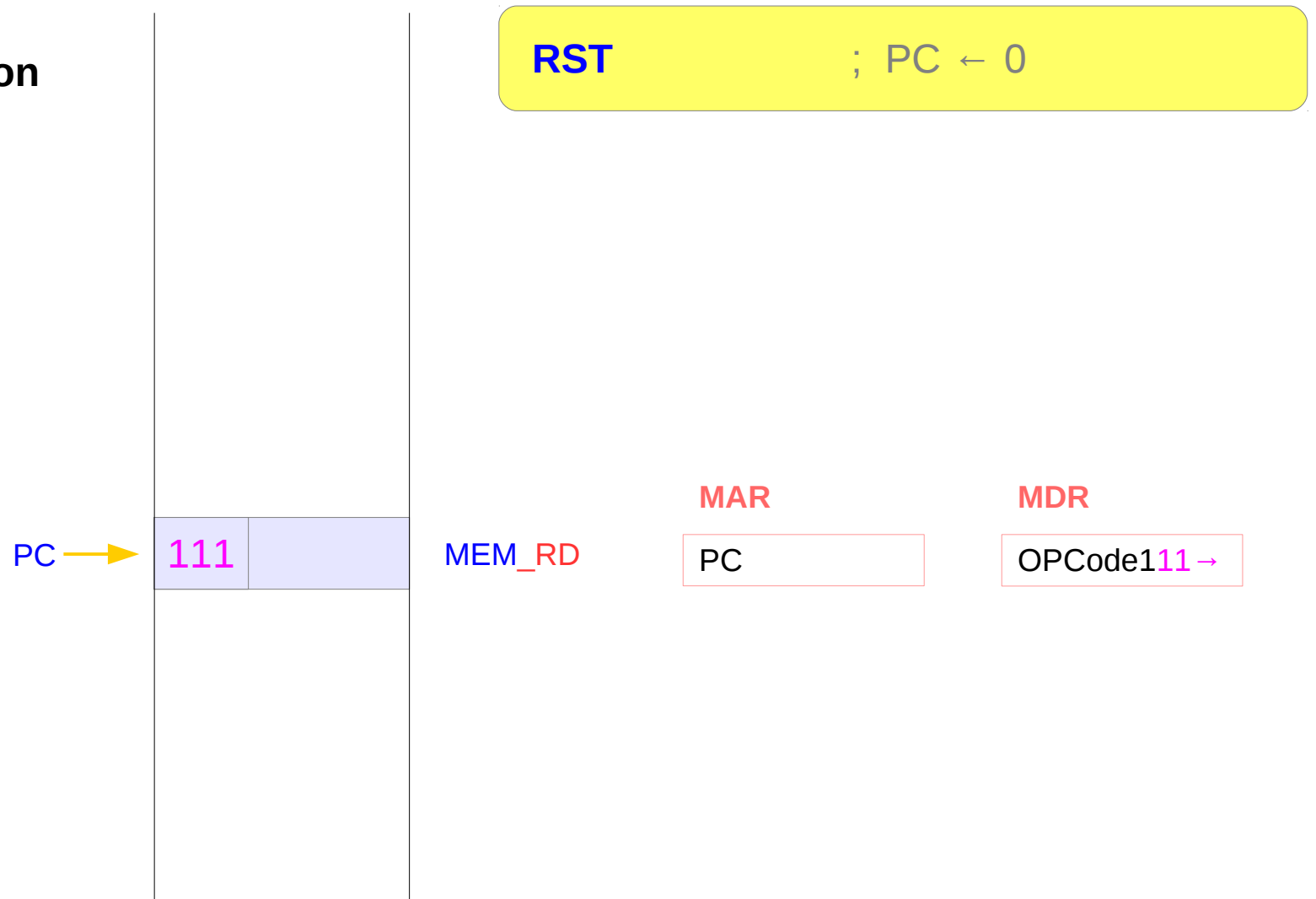
```
JNZ $M ; PC ← M if Z=0
```



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

MEM Access RST Instruction

1 RD – Instruction



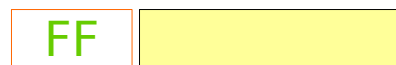
Based on <http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf>

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
 78 01 ONE: DAT 0000 0001 ;Constant 1

00 → 01 → ... → FE → FF → 00

00	60
01	20
02	78
03	40
04	FF
05	A0
06	01
07	E0
78	01



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>