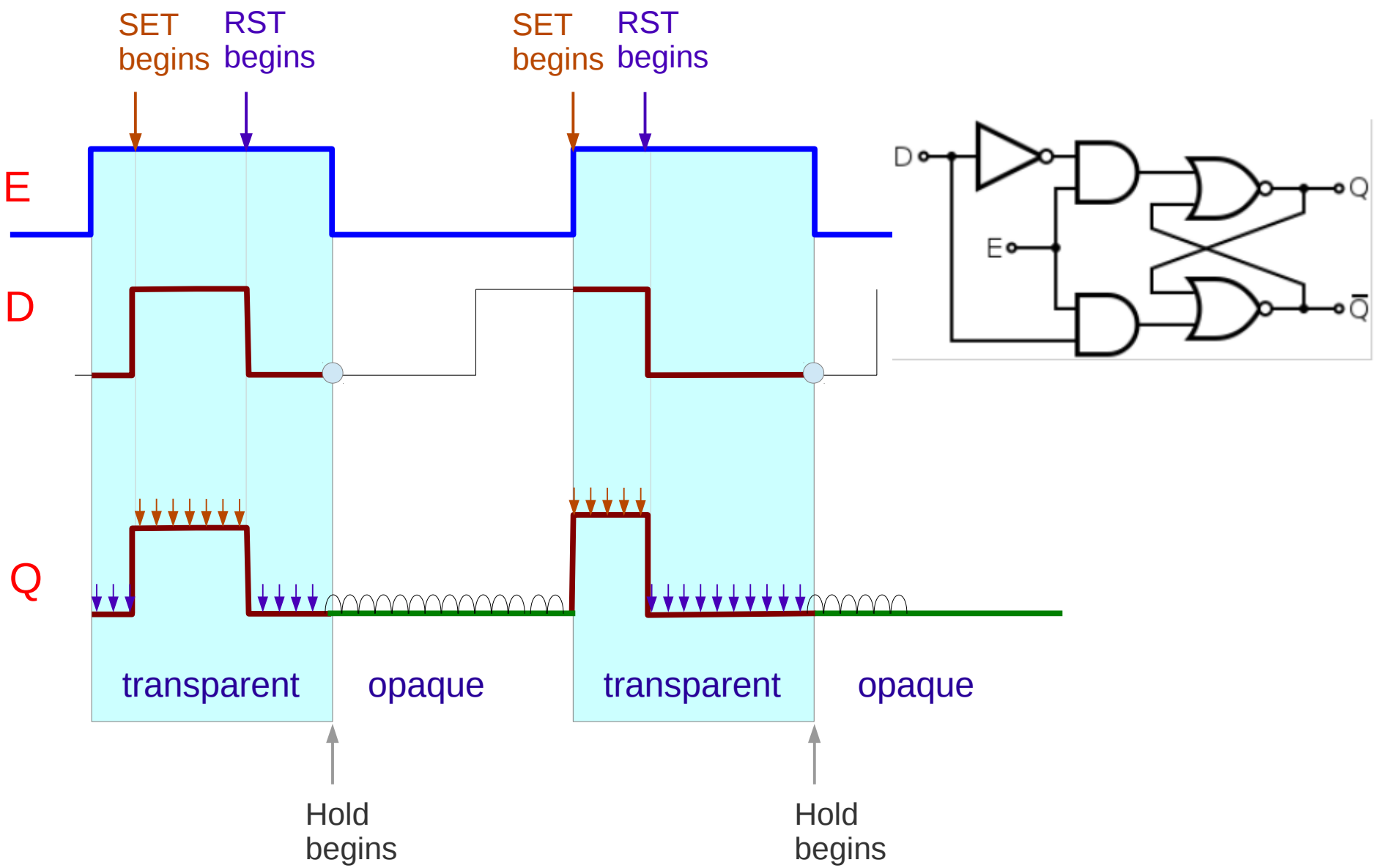
Tiny CPU Overview

Young Won Lim
5/4/16

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

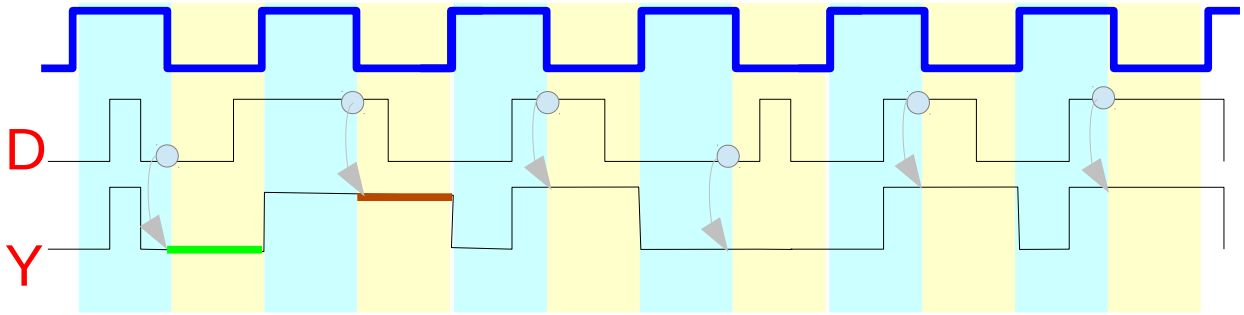
This document was produced by using OpenOffice and Octave.

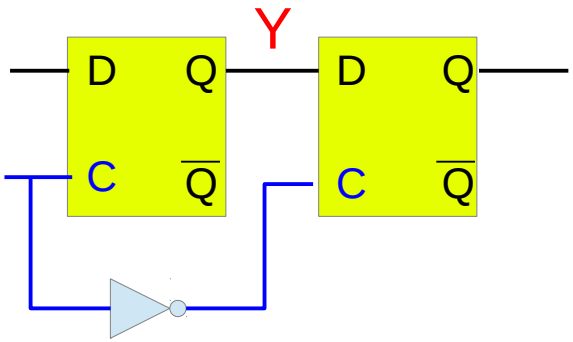
Gated D Latch



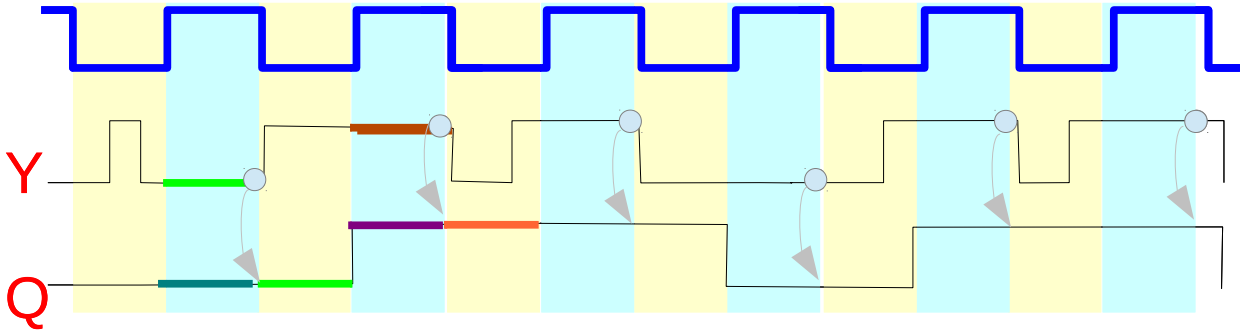
Edge Triggered D FlipFlop (2)

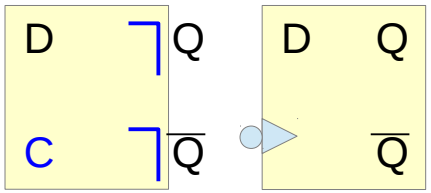
Master D Latch



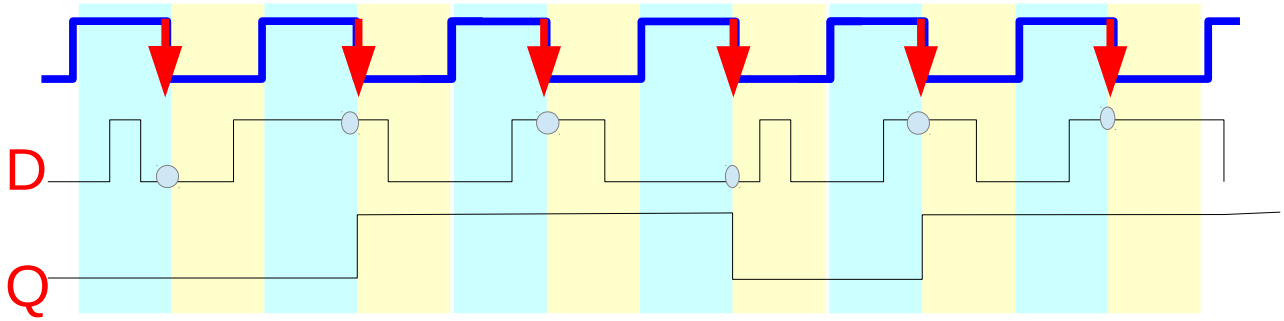


Slave D Latch



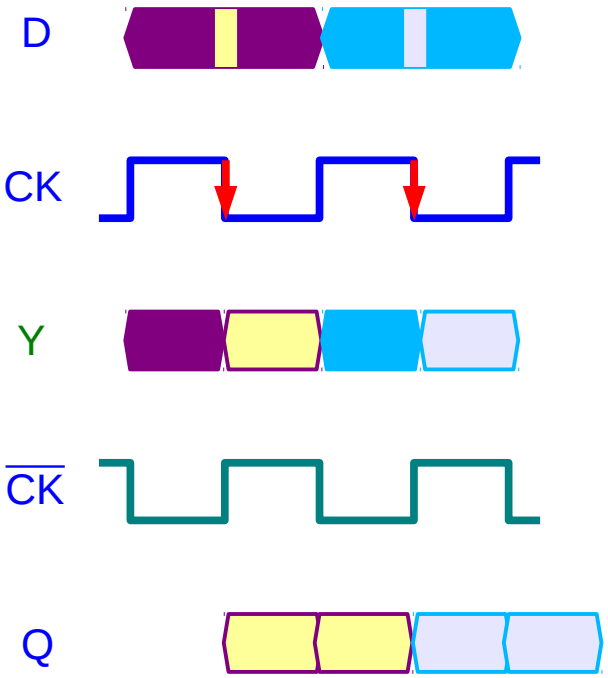


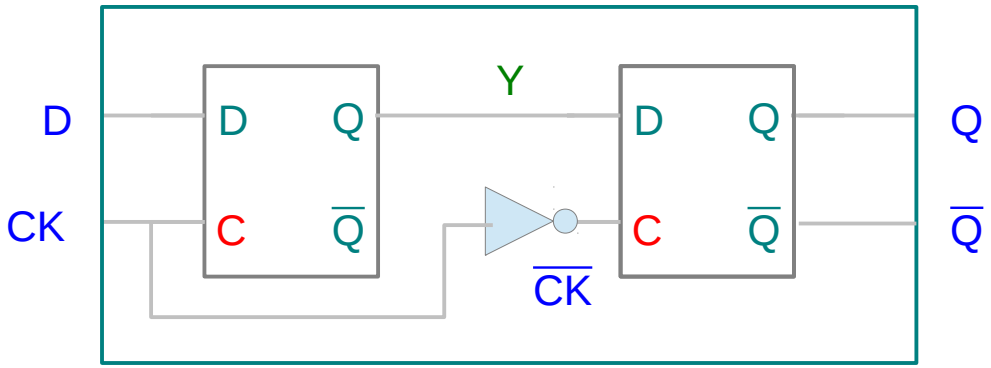
Edge Triggered D F/F



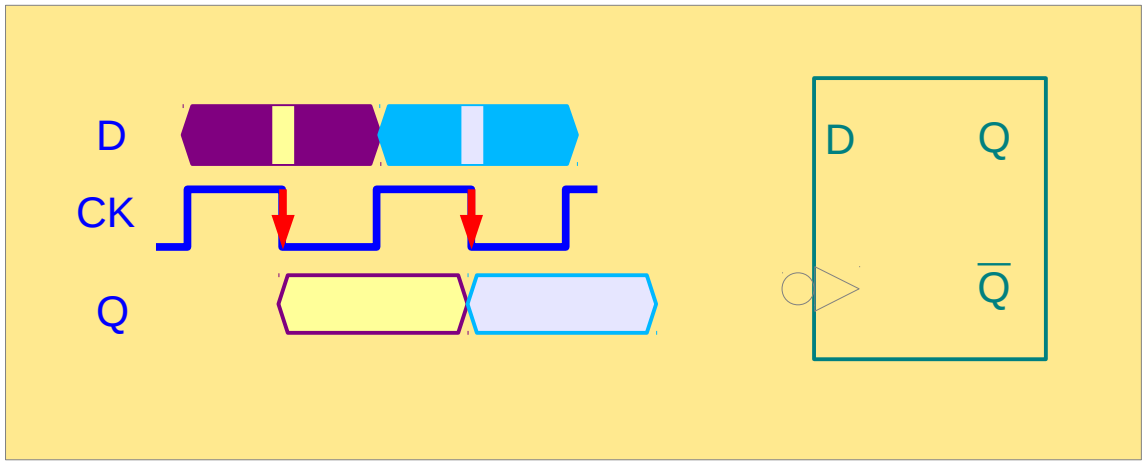
Master-Slave D FlipFlop – Falling Edge

Master D Latch



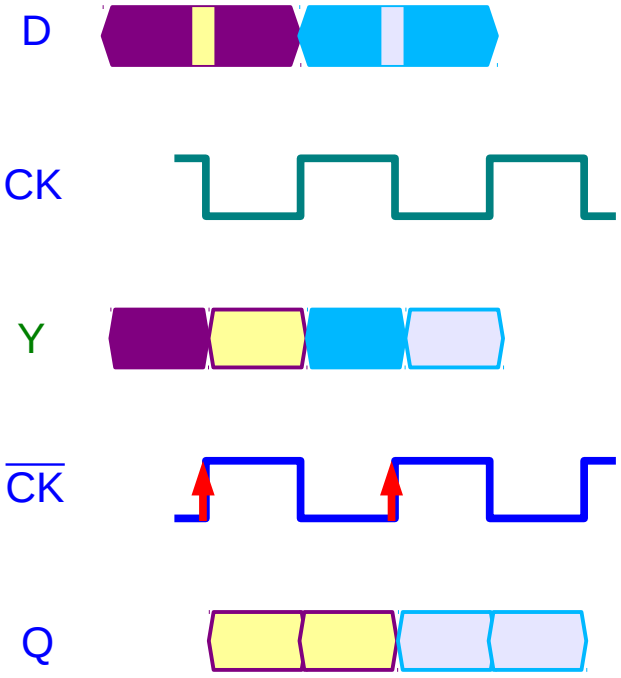


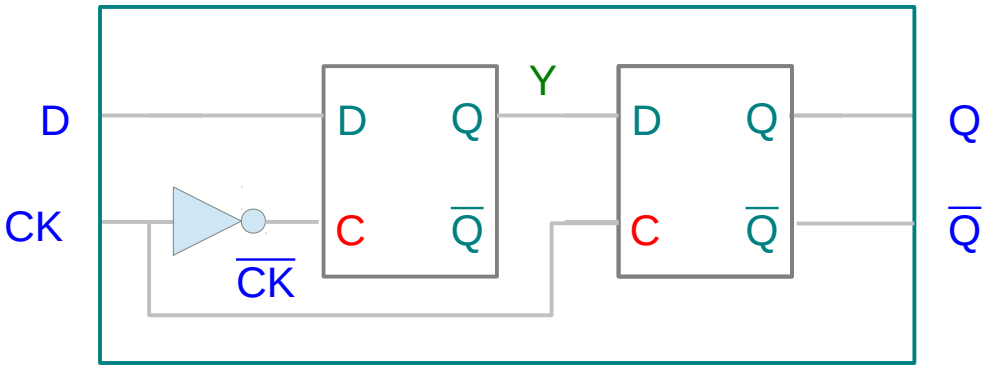
Slave D Latch



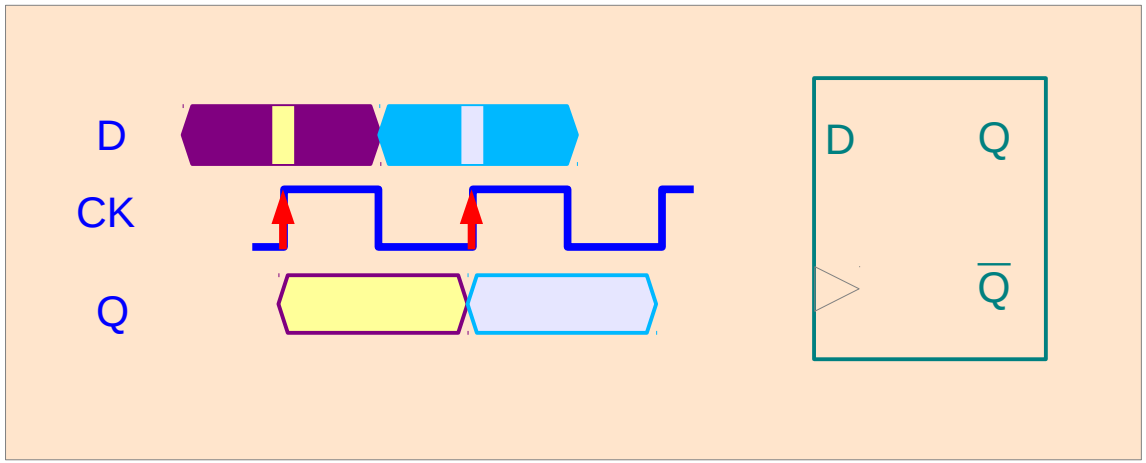
Master-Slave D FlipFlop – Rising Edge

Master D Latch





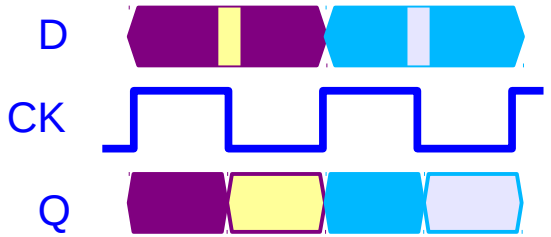
Slave D Latch

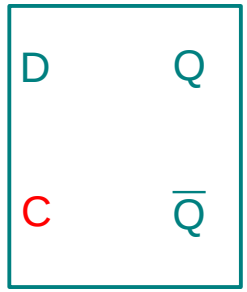


D Latch & D FlipFlop

Level Sensitive D Latch

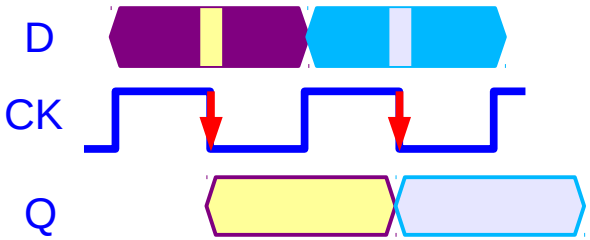
CK=1 transparent
CK=0 opaque

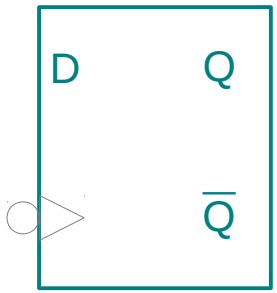




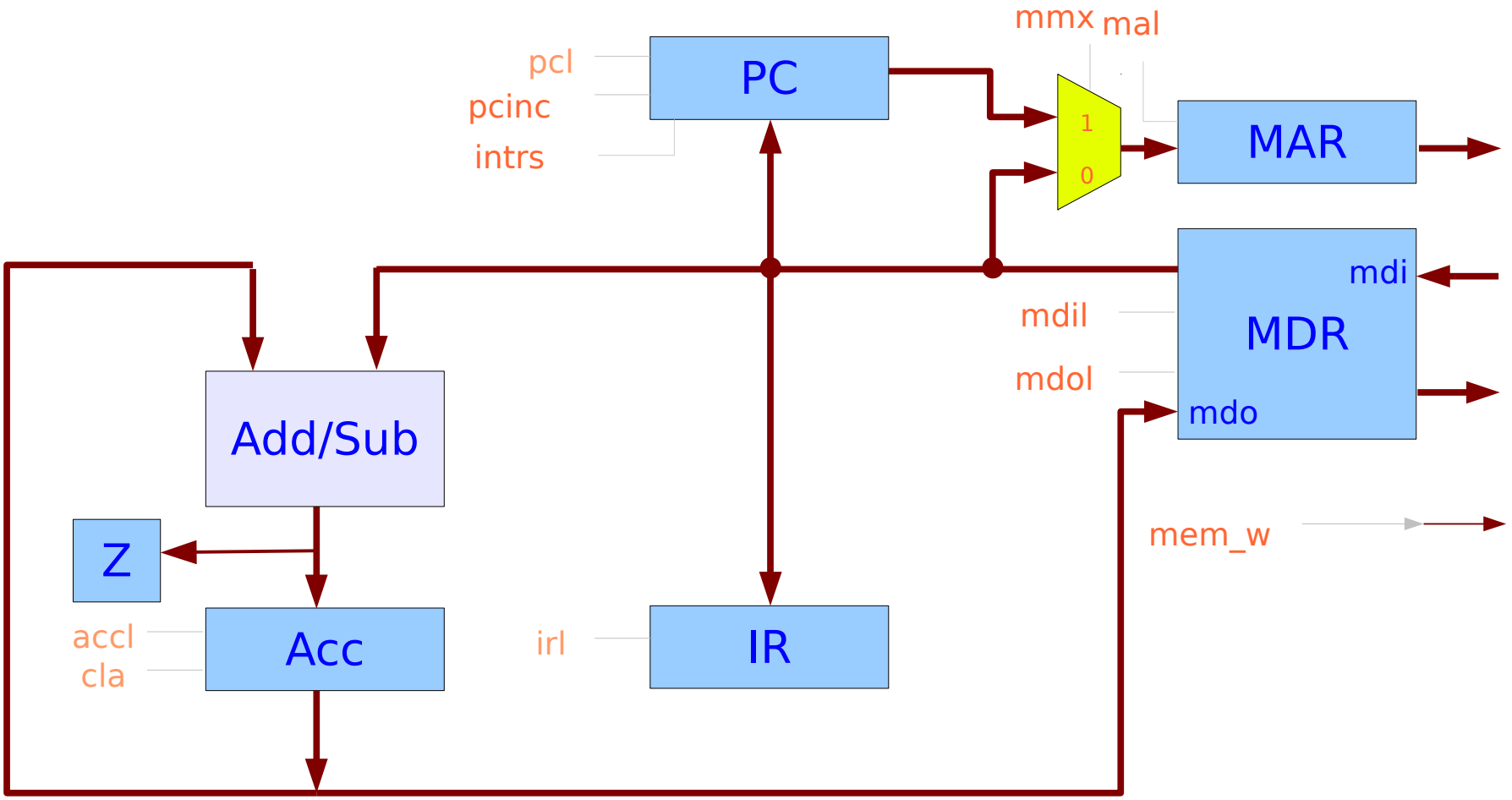
Edge Sensitive D FlipFlop

CK=1→0 transparent
else opaque



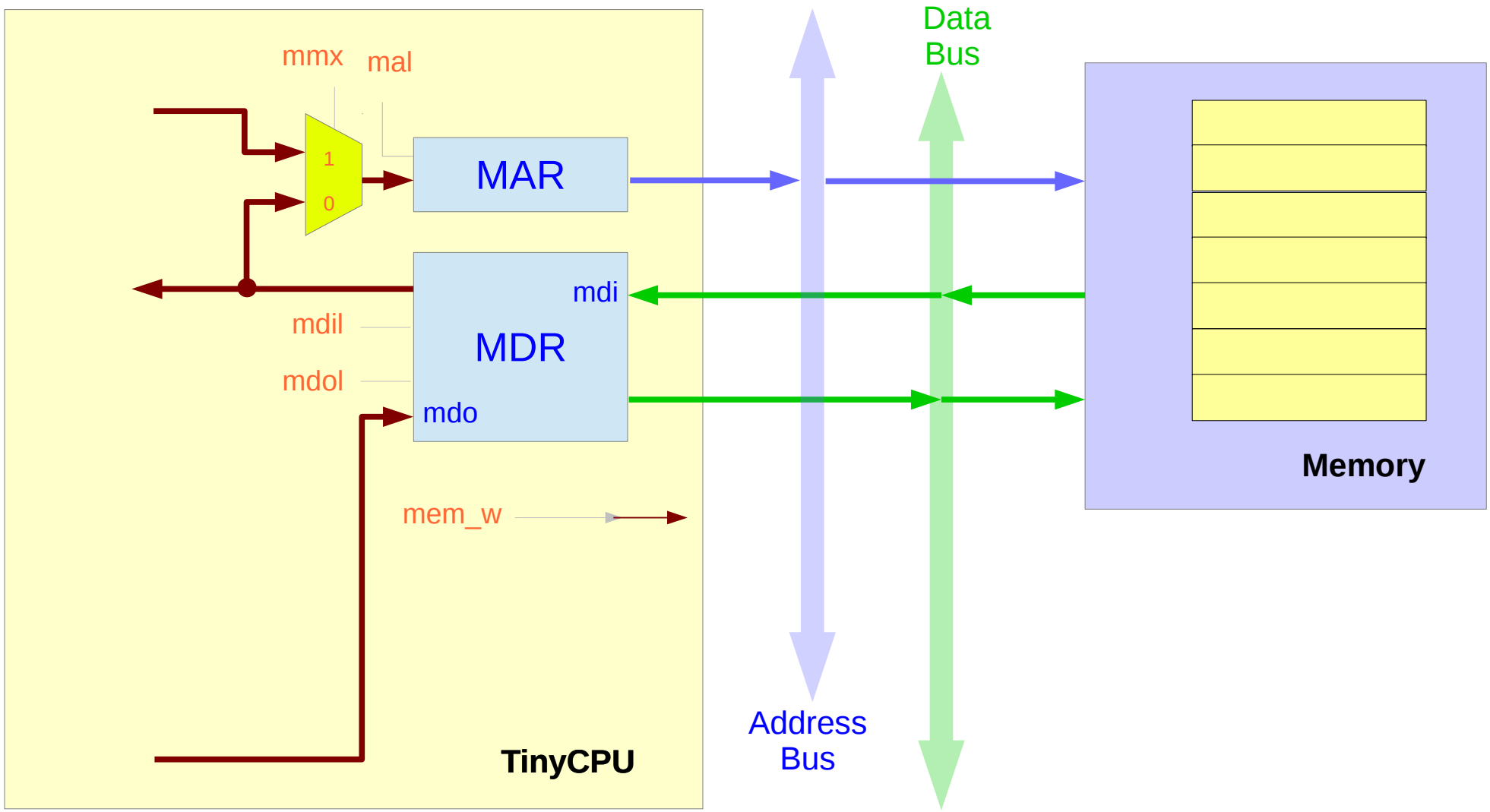


Data Path



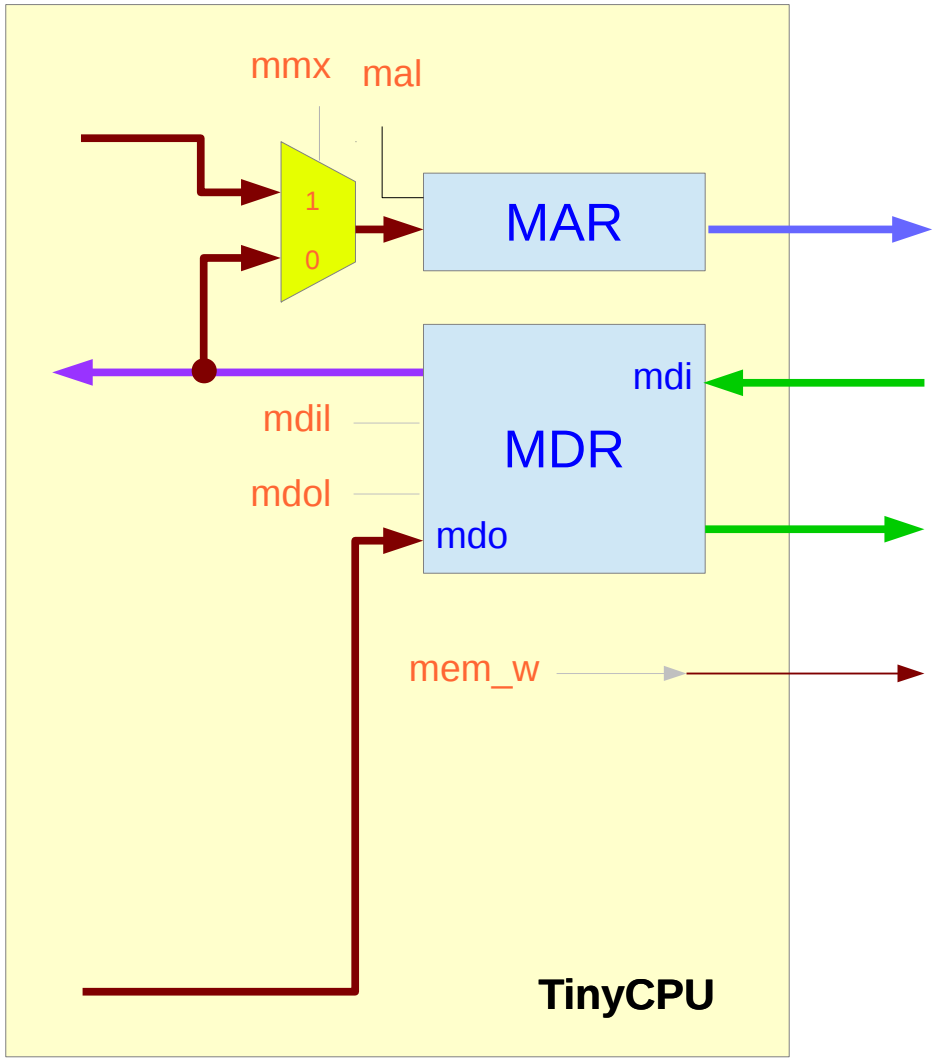
Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

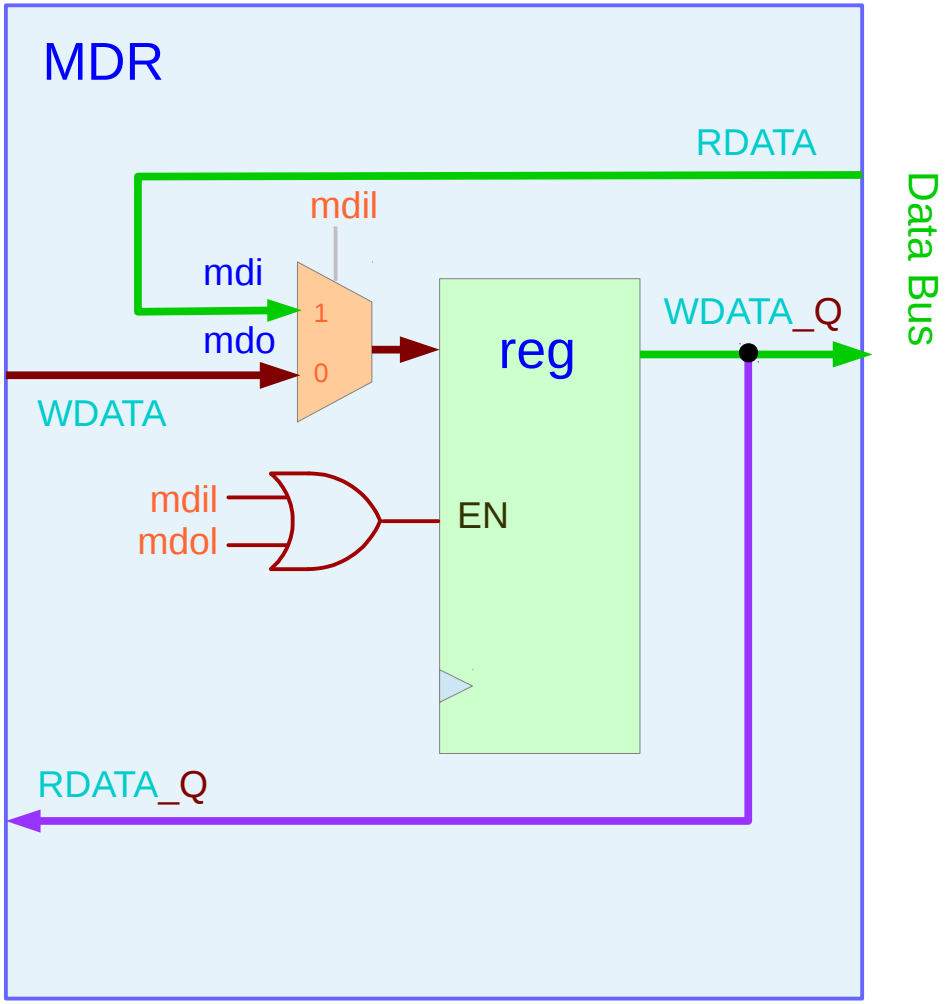
CPU and MEM



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

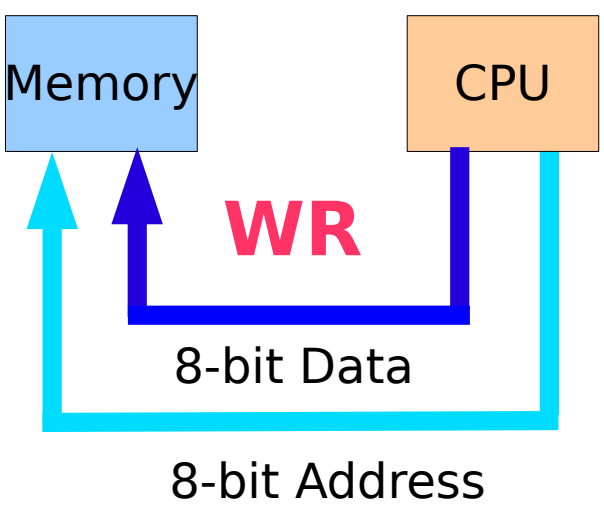
Memory Data Register

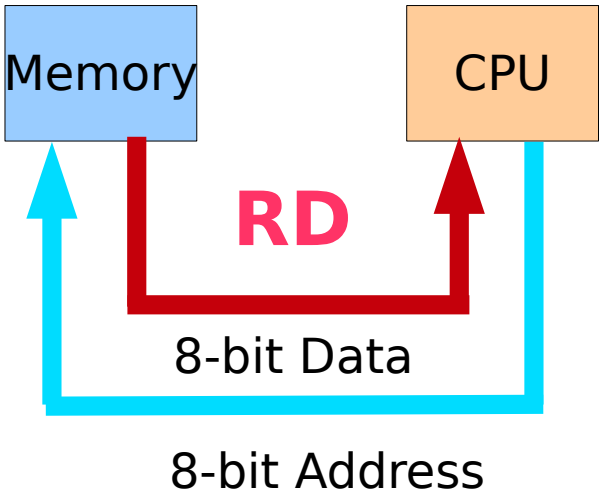




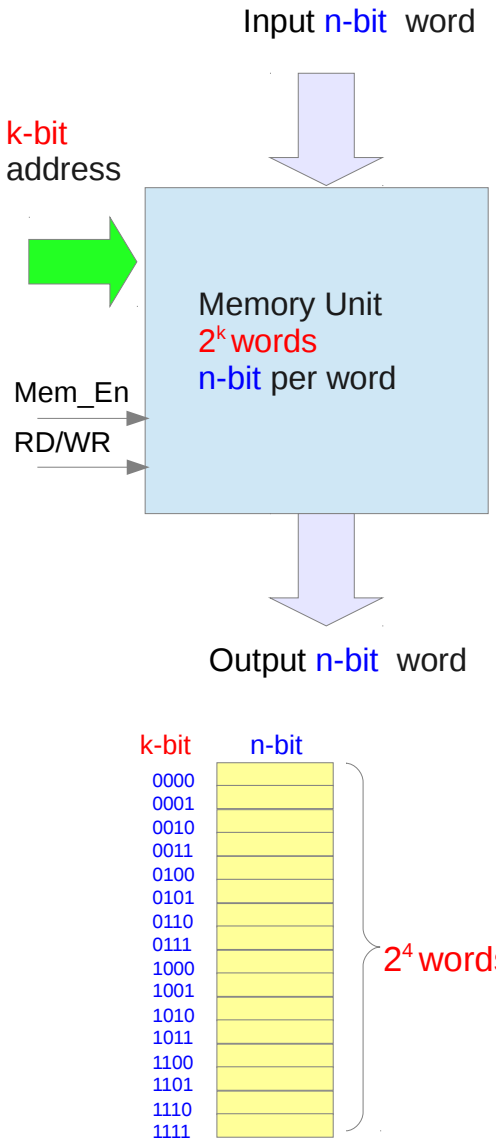
Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

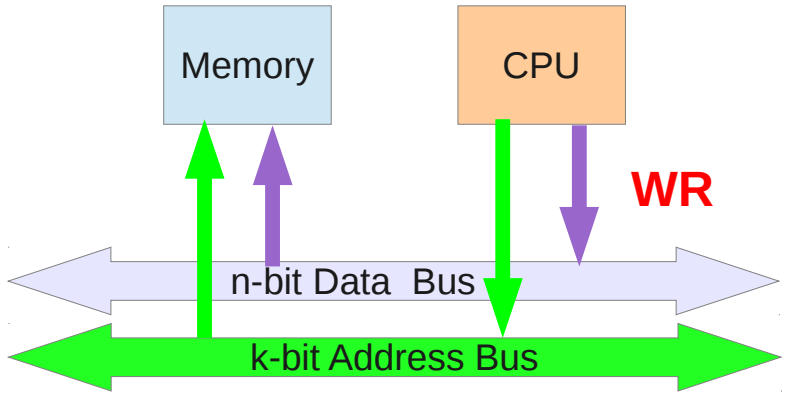
Memory Access Operations

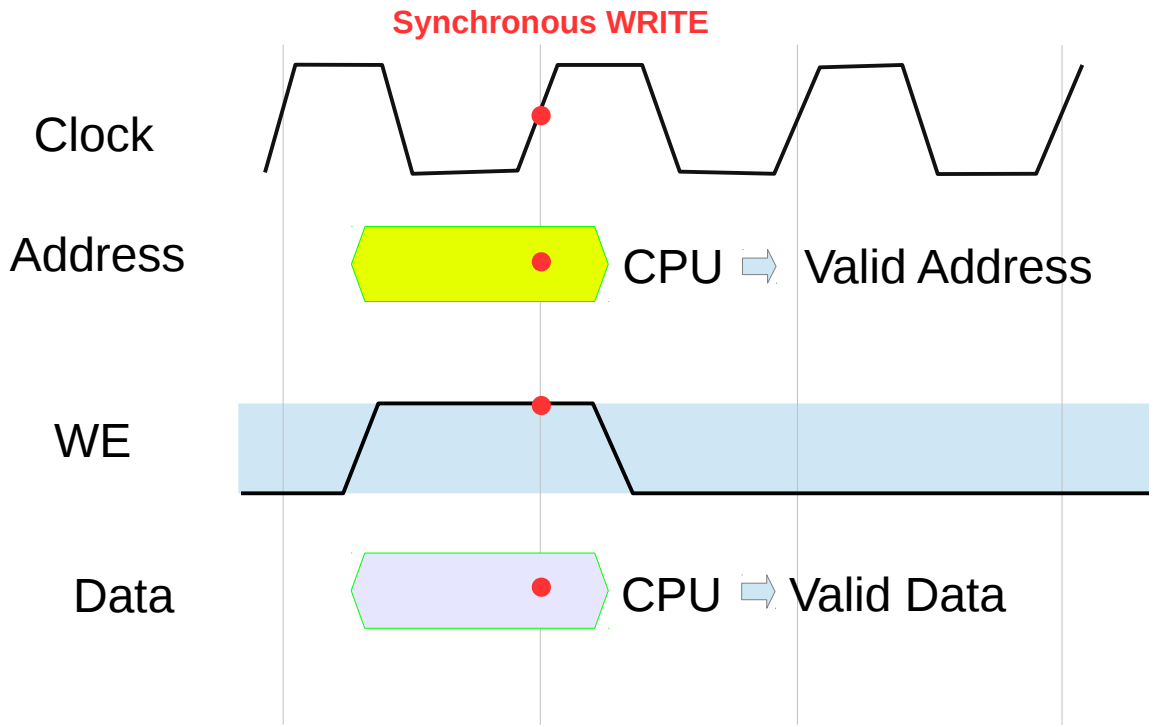




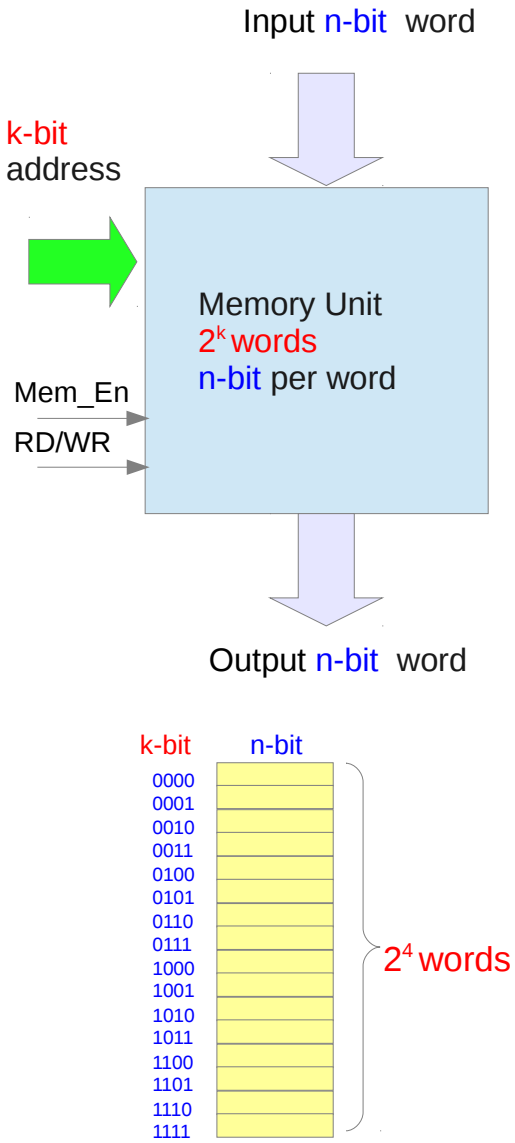
Memory Write Cycle – Synchronous

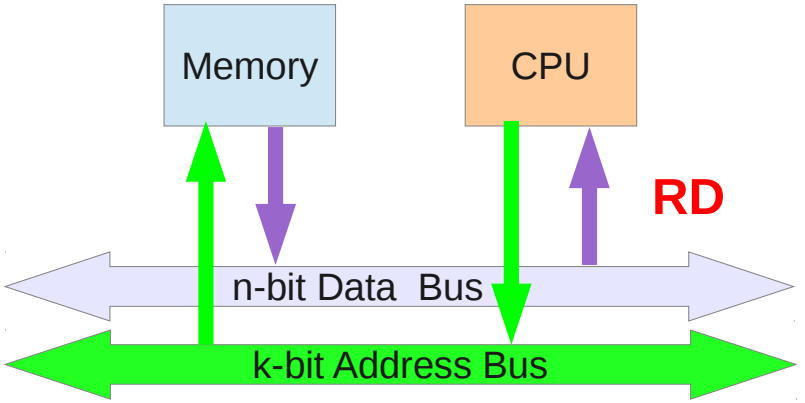


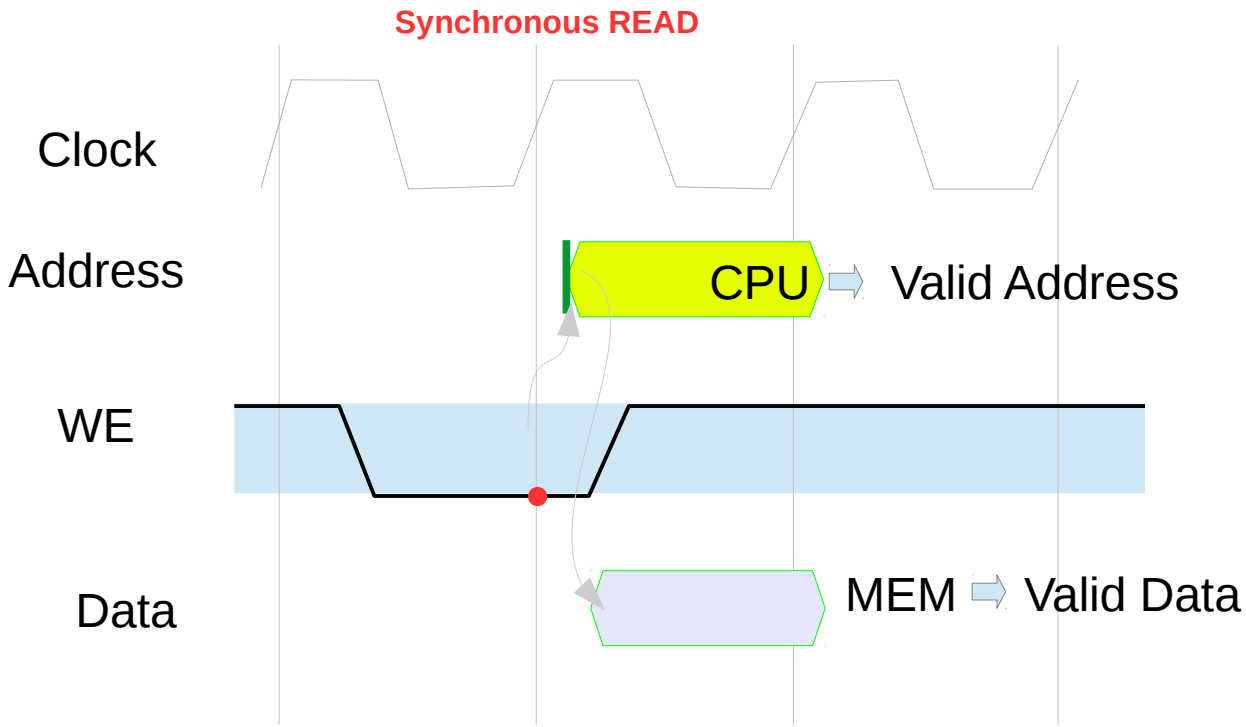




Memory Read Cycle – Asynchronous







Waveform Viewer Timing (1)

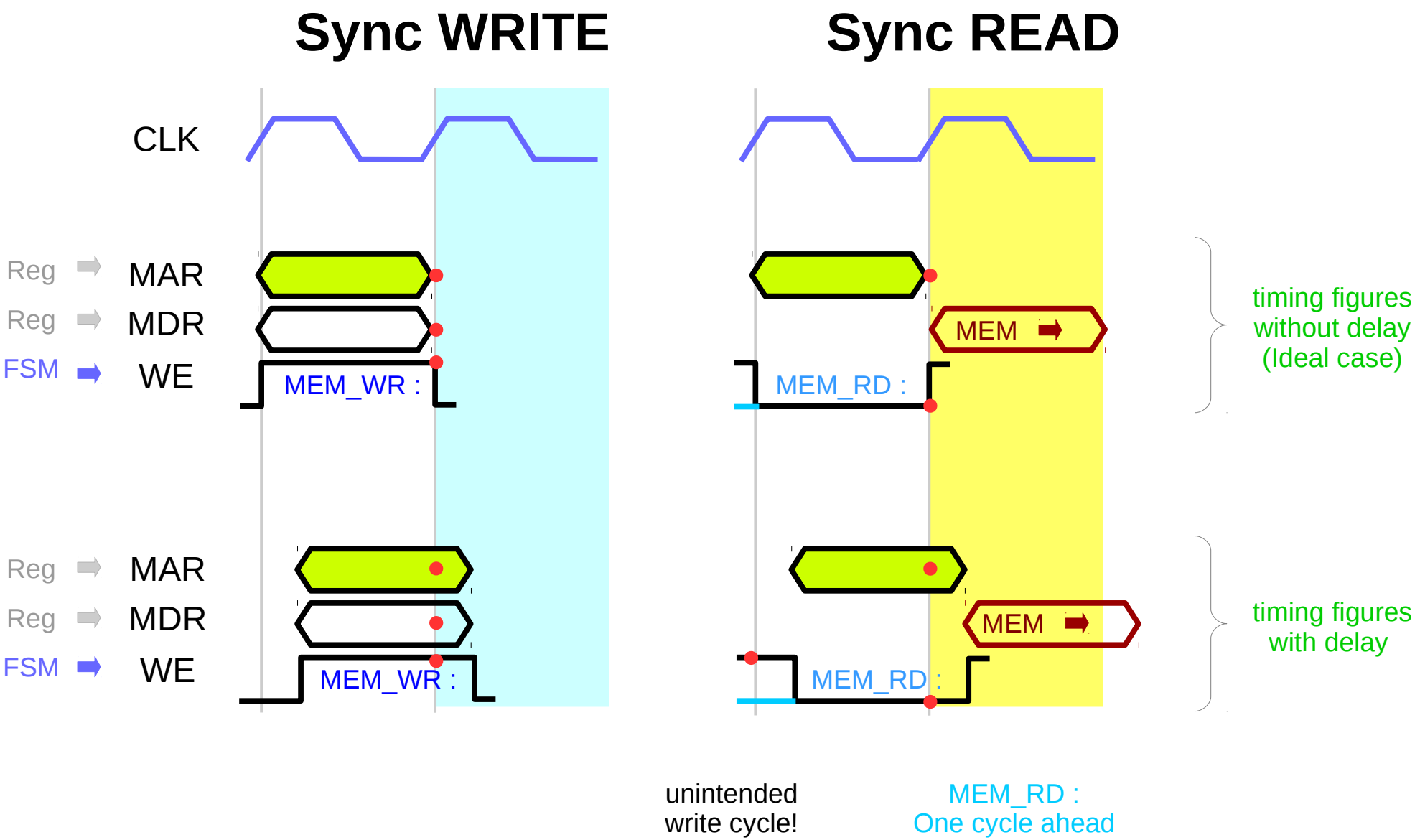
* timing figures without delay (Ideal case)

RTL functional simulation

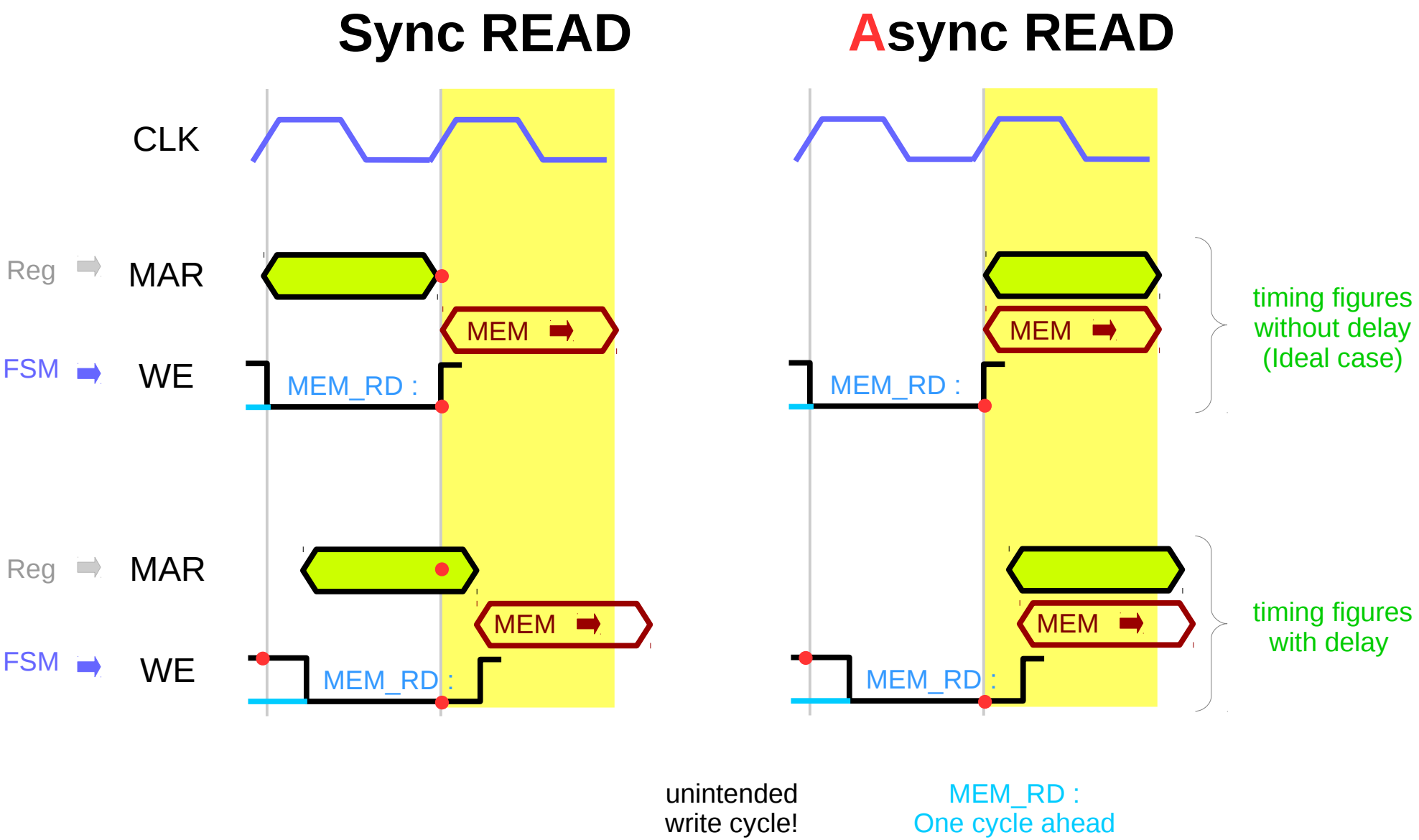
* timing figures with delay

Gate level simulation
with SDF annotation

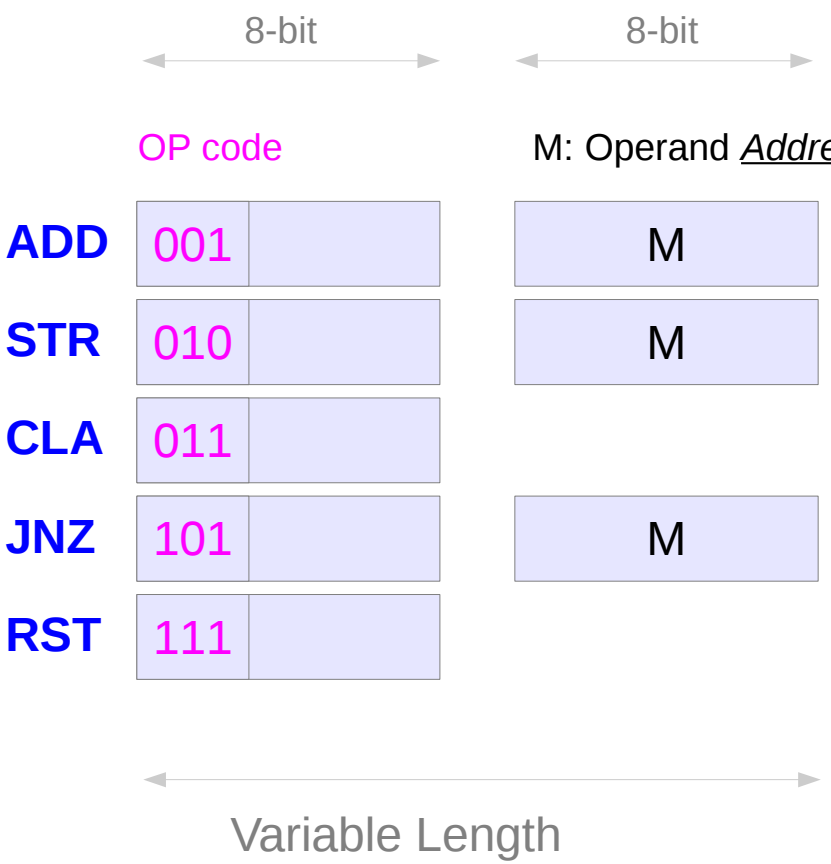
Waveform Viewer Timing (2)



Waveform Viewer Timing (3)



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

Instruction Cycles

	common cycles
c1	MAR ← PC; **MEM_RD**;
c2	**MDR ← MEM;** PC ← PC+1;
c3	IR ← MDR

	ADD specific cycles
c4	MAR ← PC; **MEM_RD**
c5	**MDR ← MEM;** PC ← PC+1
c6	MAR ← **MDR**; **MEM_RD**
c7	**MDR ← MEM;**
c8	ACC ← ACC + MDR;

	CLA specific cycles
c4	ACC ← “00”

	RST specific cycles
c4	PC ← “00”

Opcode_RD

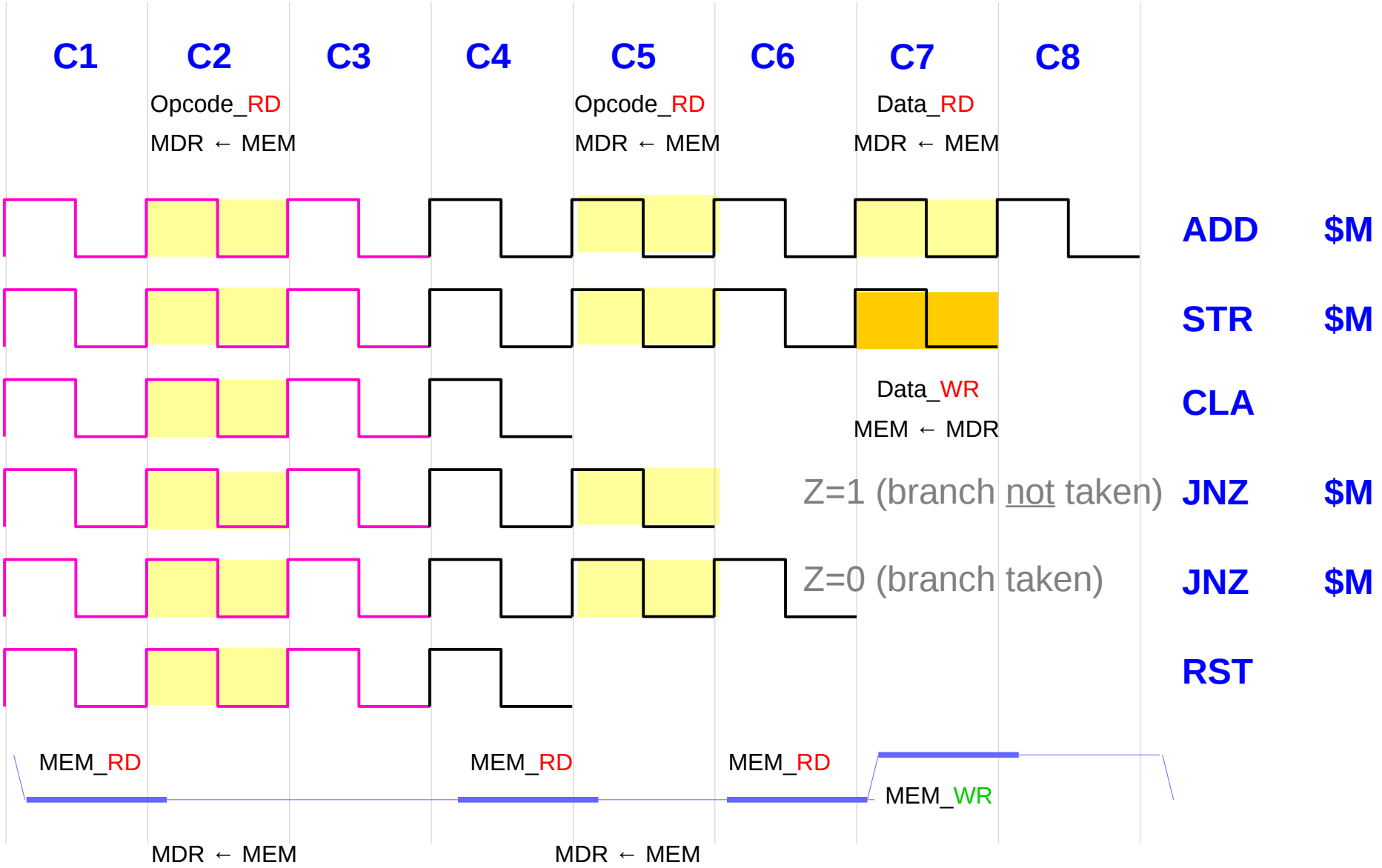
Operand_RD

	STR specific cycles
c4	MAR ← PC; **MEM_RD**
c5	**MDR ← MEM;** PC ← PC + 1
c6	**MAR ← MDR; MDR ← ACC**
c7	**MEM_WR**;

	JNZ specific cycles
c4	MAR ← PC; **MEM_RD**;
c5	**MDR ← MEM;** PC ← PC + 1
c6	If (Z=='0') PC ← MDR;

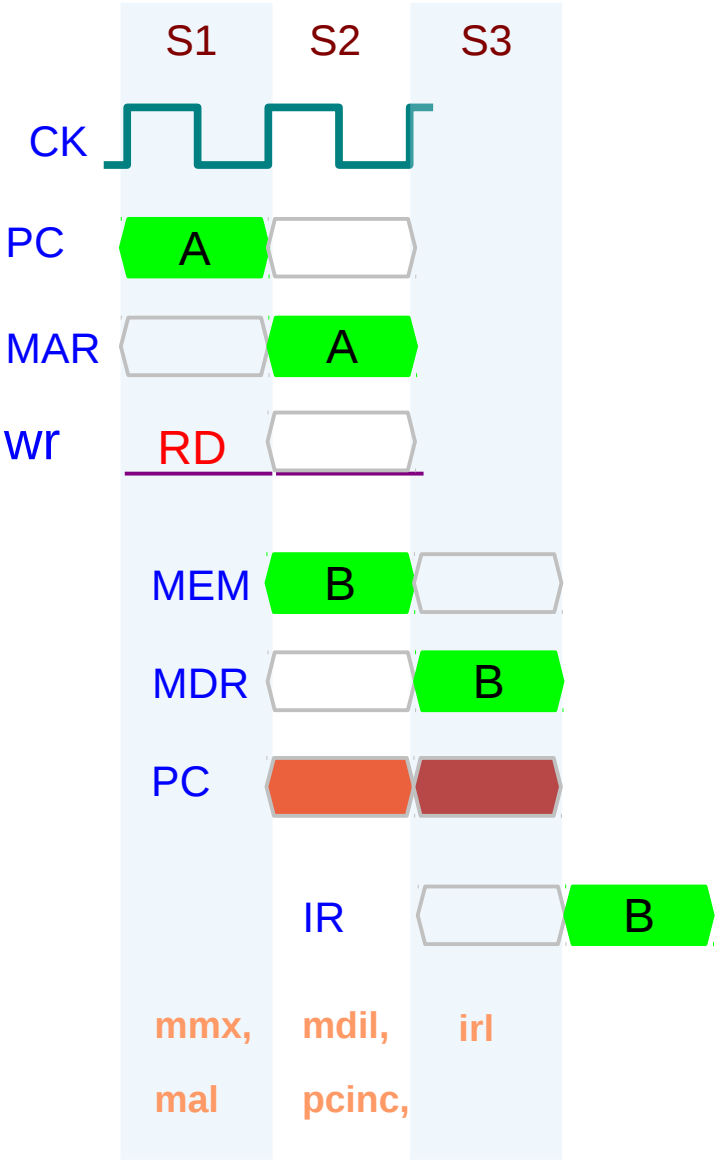
Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

Clock Cycle Counts



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

IF (S1,S2,S3)



S1 MAR ← PC; MEM_RD;

S2 MDR ← MEM; PC ← PC+1;

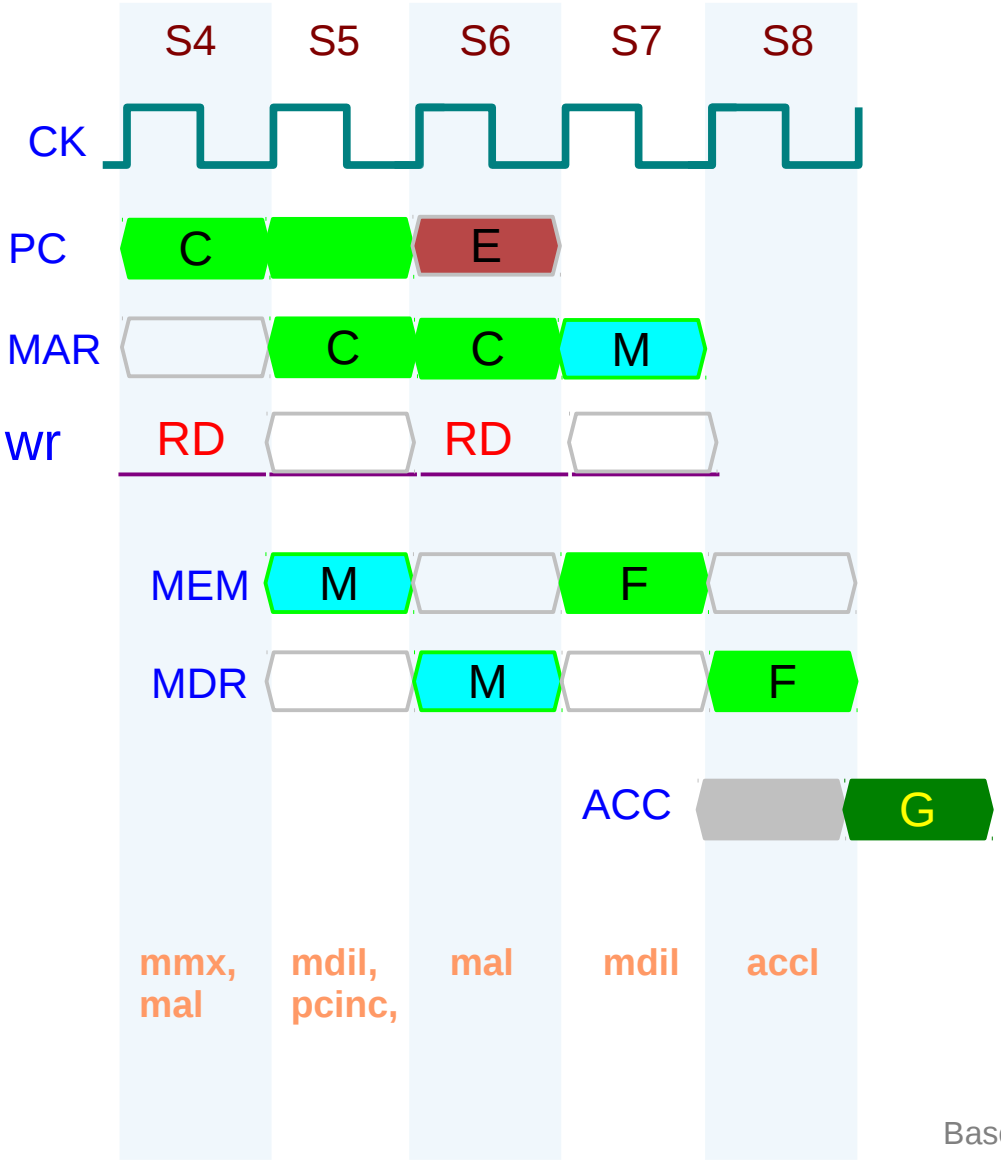
S3 IR ← MDR

IR[7:5]

ADD	001
STR	010
CLA	011
JNZ	101
RST	111

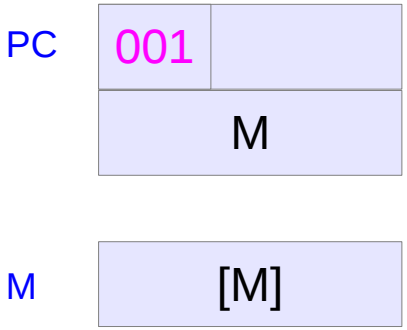
Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

ADD (S4,S5,S6,S7,S8)



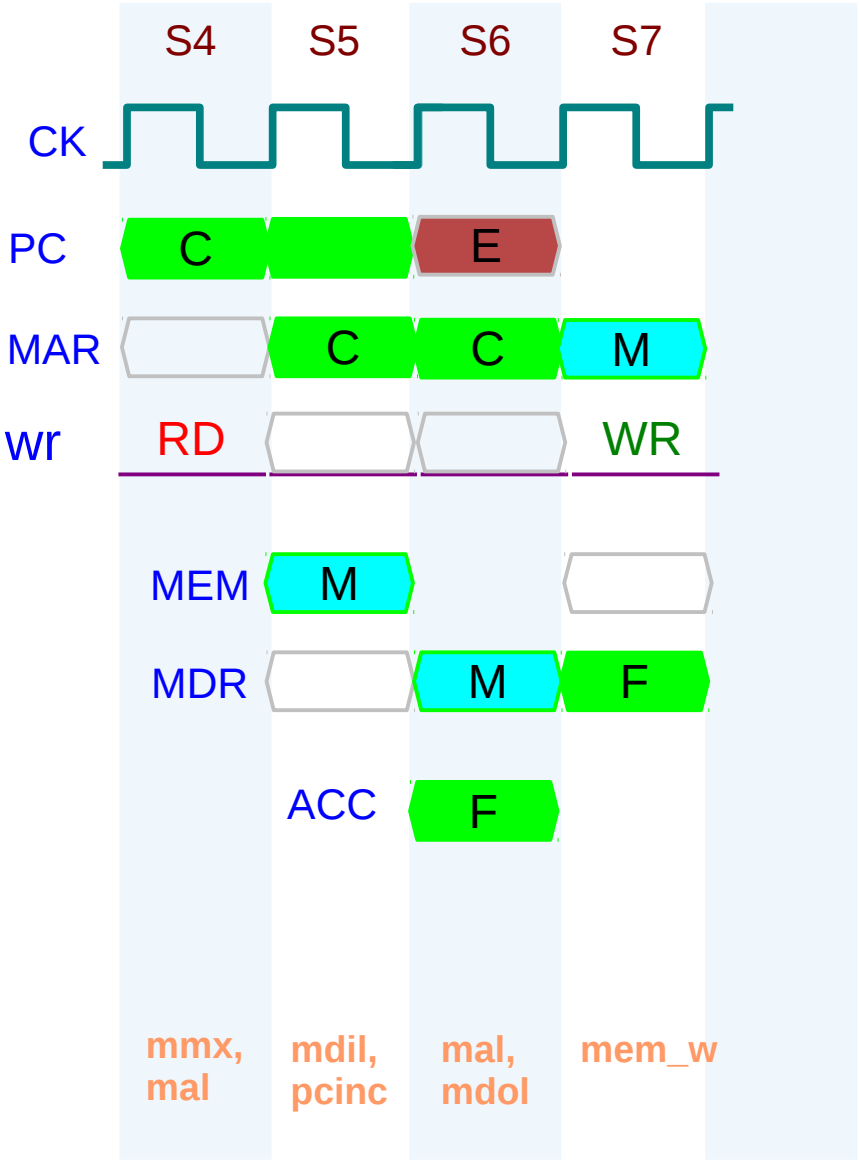
S4	MAR ← PC; MEM_RD
S5	MDR ← MEM; PC ← PC+1
S6	MAR ← MDR; MEM_RD
S7	MDR ← MEM;
S8	ACC ← ACC + MDR;

ADD **$M** ; ACC ← ACC + [M]



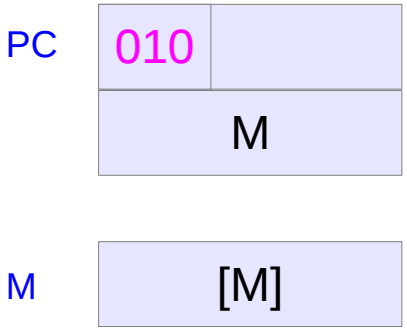
Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

STR (S4,S5,S6,S7)



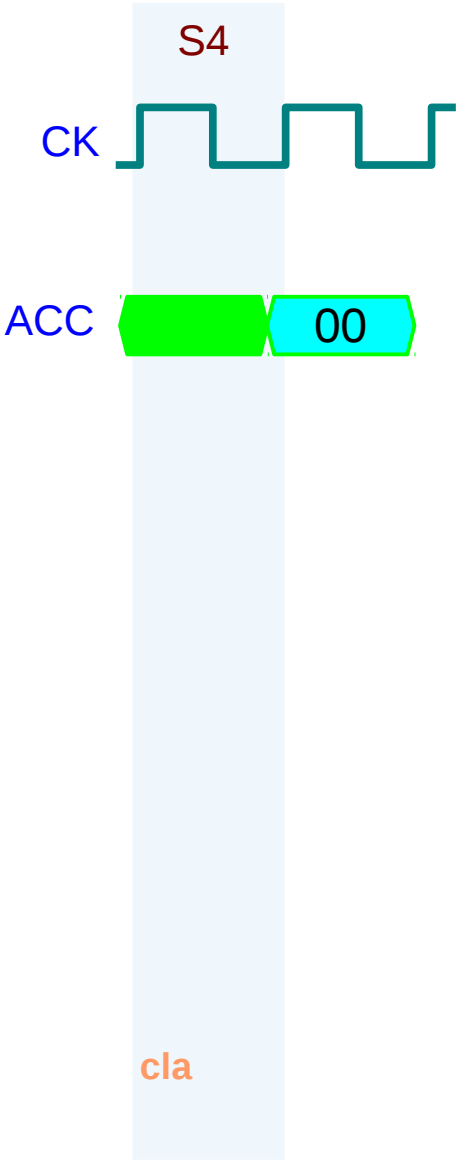
S4	MAR ← PC; MEM_RD
S5	MDR ← MEM; PC ← PC + 1
S6	MAR ← MDR; MDR ← ACC
S7	MEM_WR;

STR $M ; [M] ← ACC



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

CLA (S4)



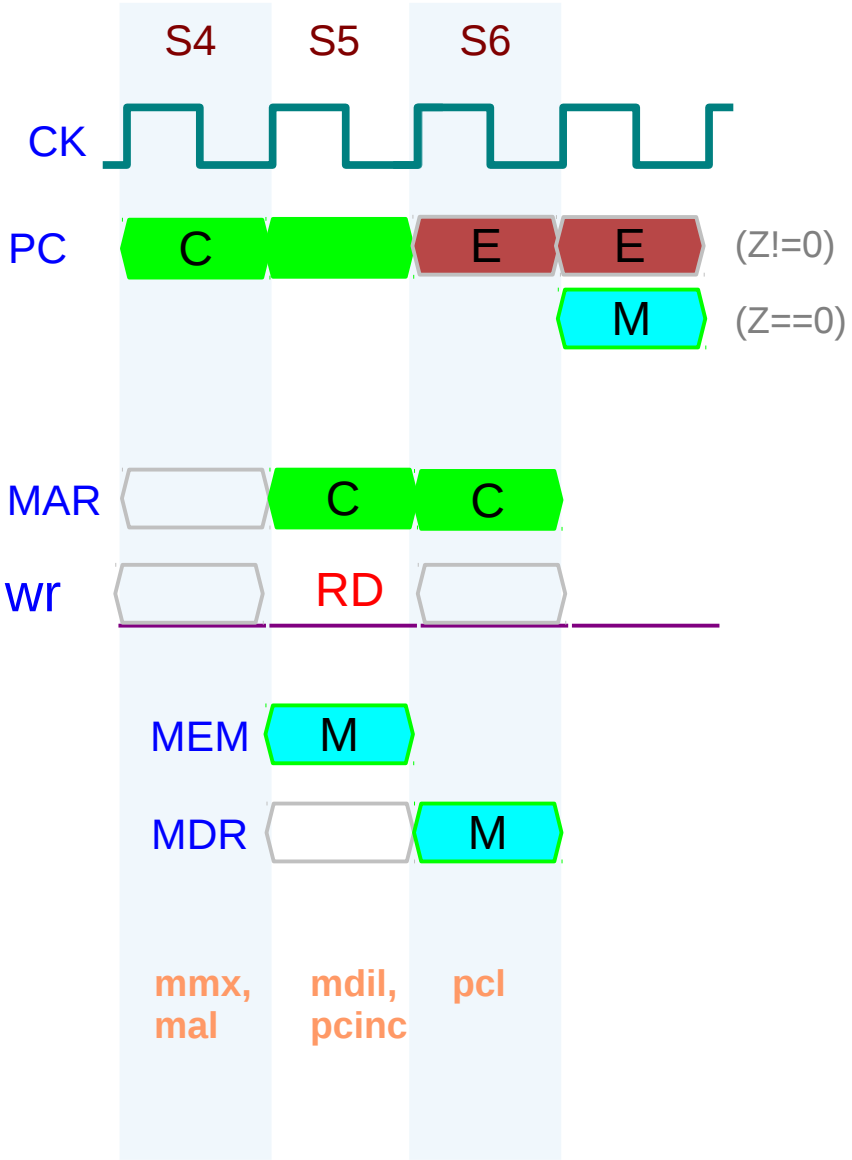
S4 ACC ← “00”

CLA ; ACC ← 0

PC 011

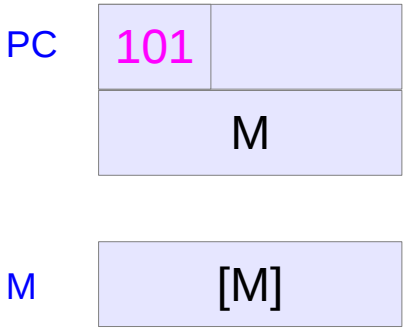
Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

JNZ (S4,S5,S6)



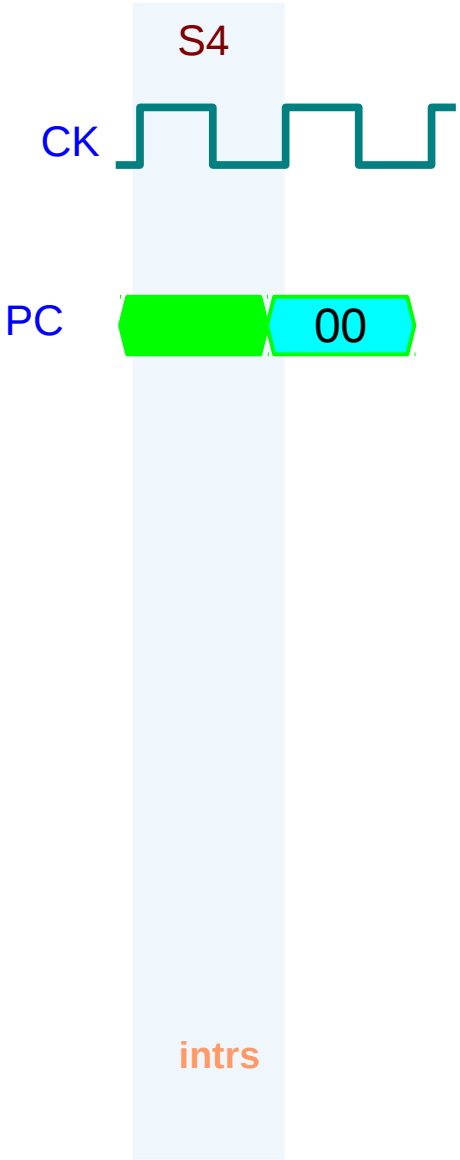
S4 MAR ← PC; MEM_RD;
S5 MDR ← MEM; PC ← PC + 1
S6 If (Z=='0') PC ← MDR;

JNZ **$M** ; PC ← [M] if Z=0



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

RST (S4)



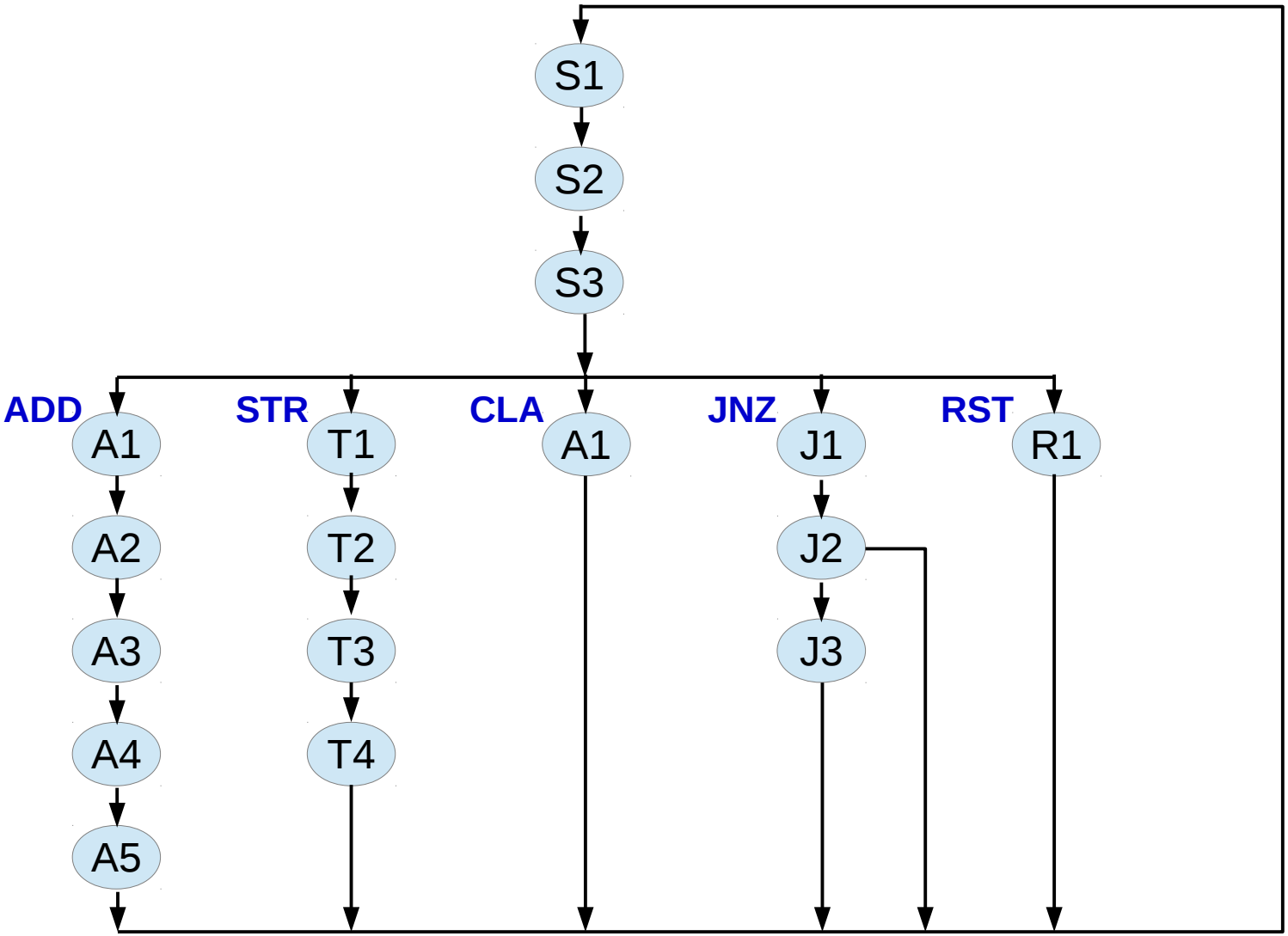
S4 PC ← “00”;

RST ; PC ← 0

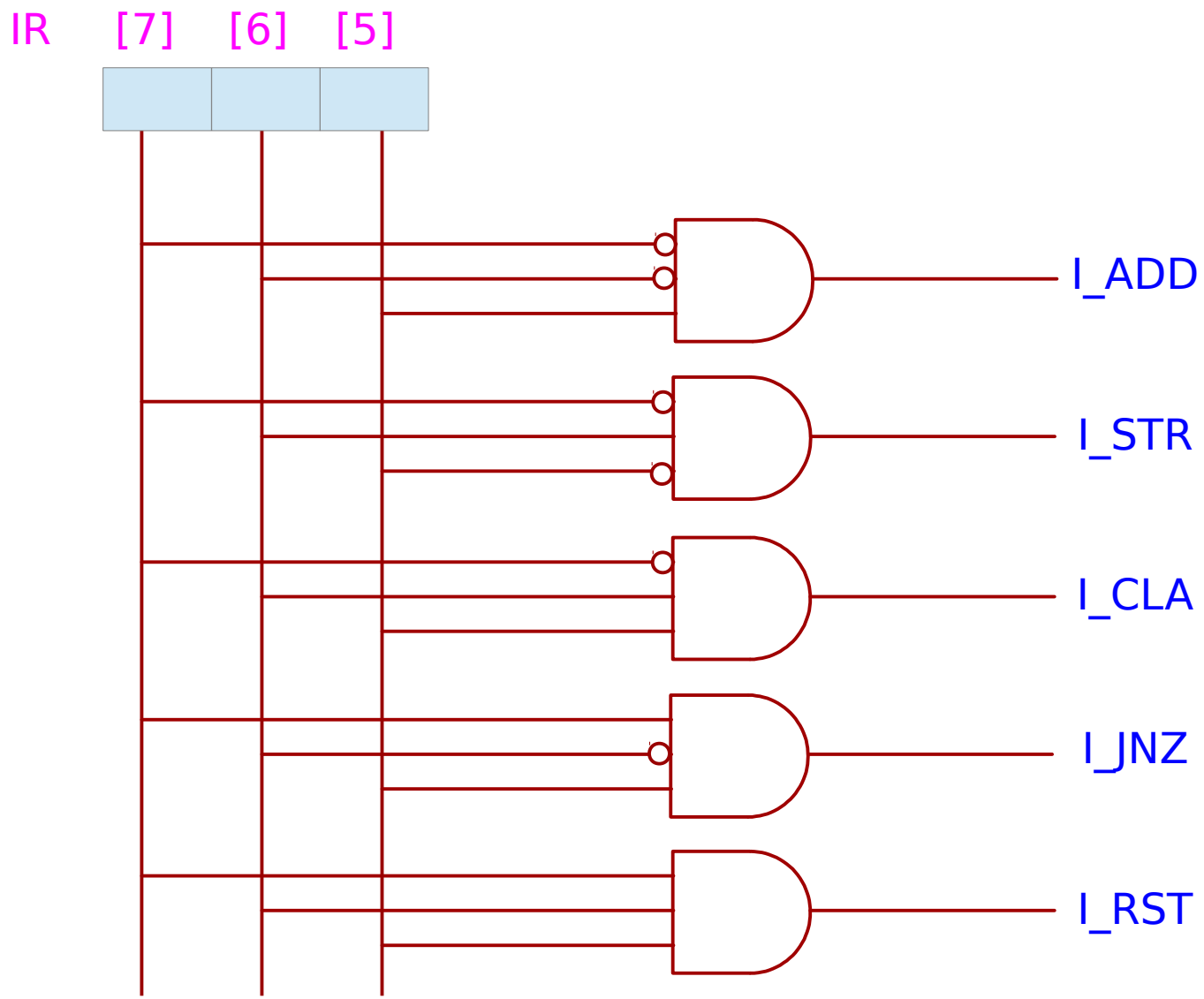
PC 111

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

FSM I



Instruction Decoder



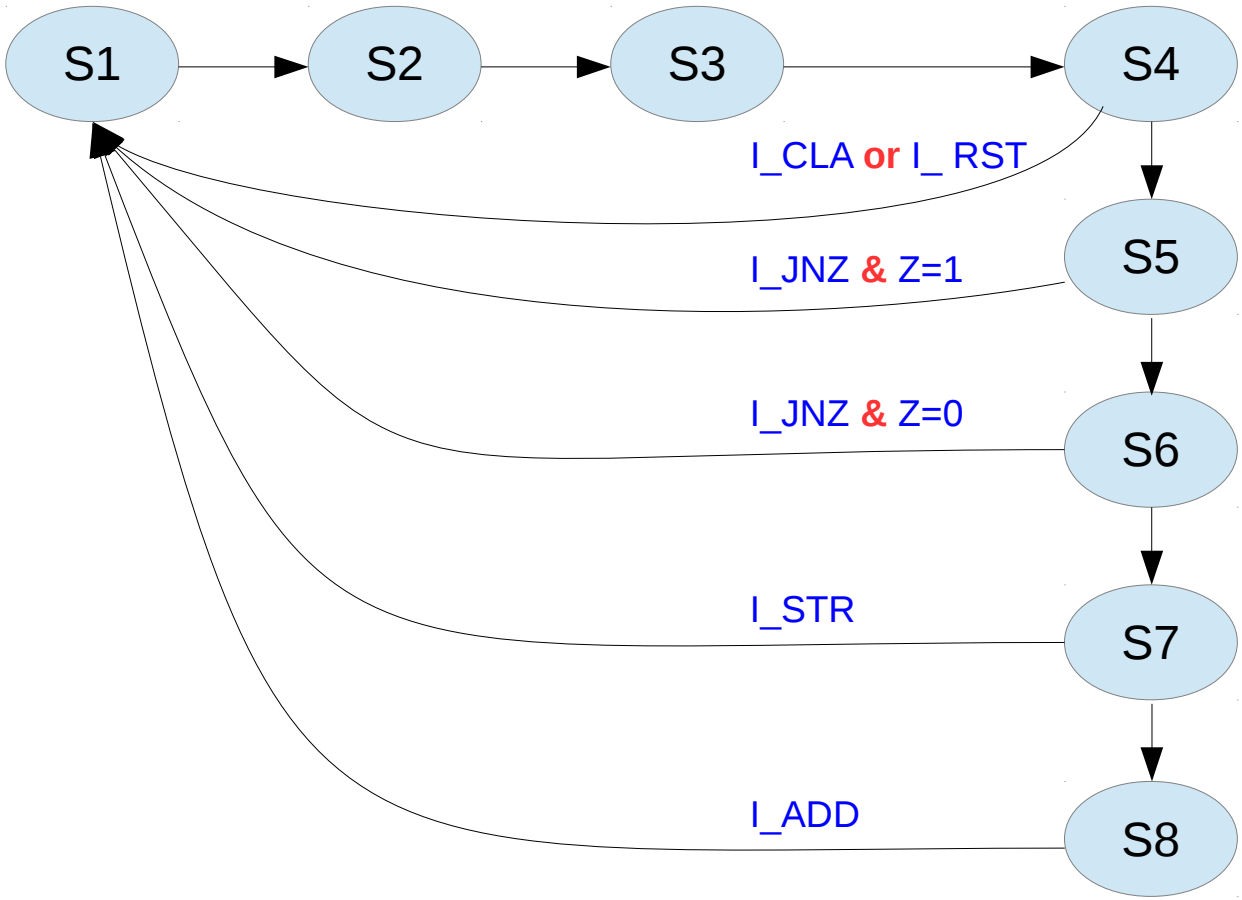
8-bit

OP code

ADD	001	
STR	010	
CLA	011	
JNZ	101	
RST	111	

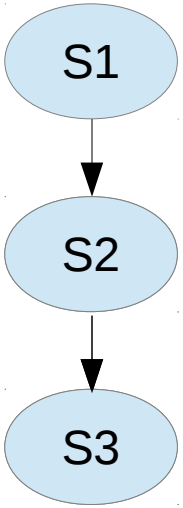
Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

FSM II



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

Control Signal – IF



mmx, mal

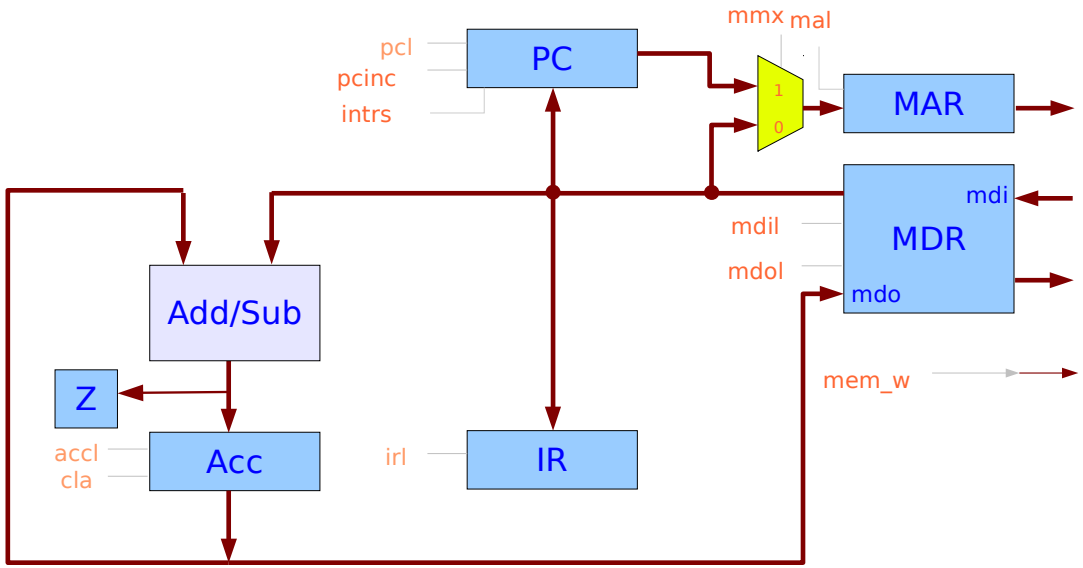
mdil, pcinc,

irl

S1 MAR ← PC; MEM_RD;

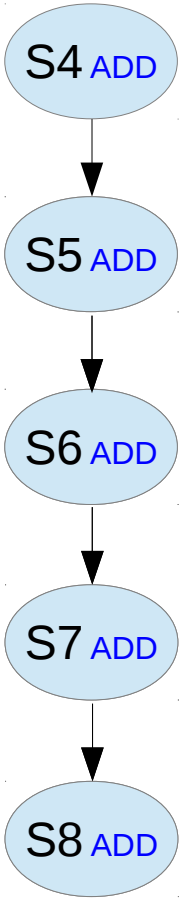
S2 MDR ← MEM; PC ← PC+1;

S3 IR ← MDR



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

Control Signal – ADD



mmx, mal

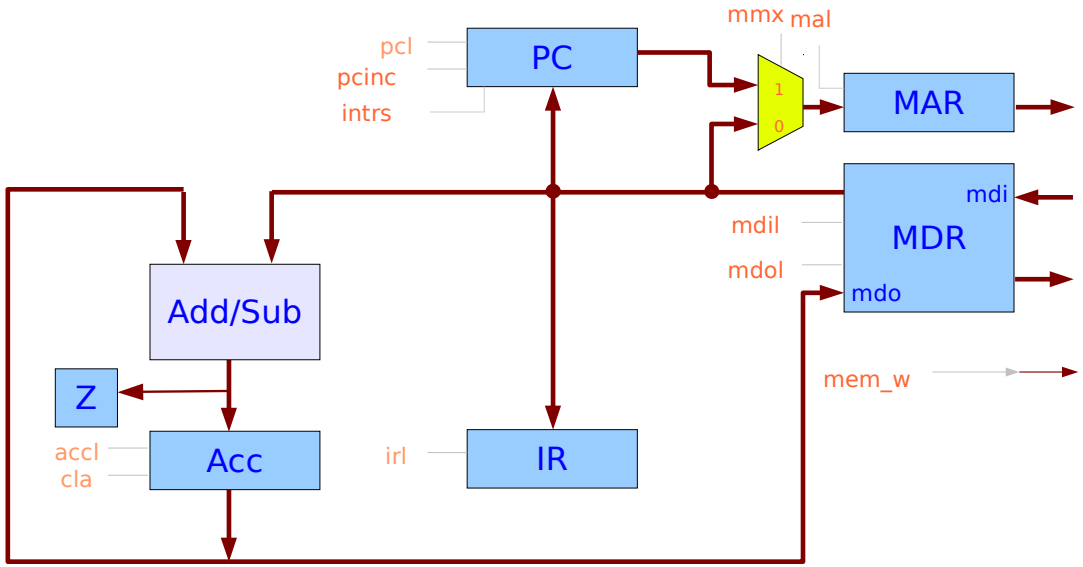
mdil, pcinc,

mal

mdil

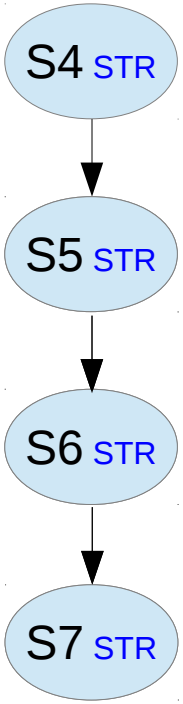
accl

S4	MAR ← PC; MEM_RD
S5	MDR ← MEM; PC ← PC+1
S6	MAR ← MDR; MEM_RD
S7	MDR ← MEM;
S8	ACC ← ACC + MDR;



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

Control Signal – STR



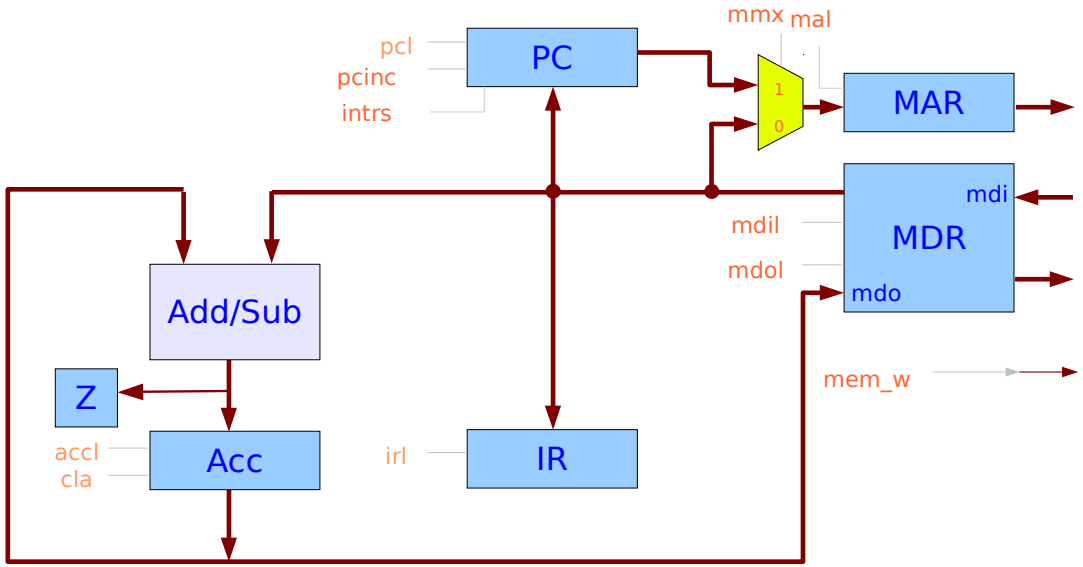
mmx, mal

mdil, pcinc

mal, mdol

mem_w

S4	MAR ← PC; MEM_RD
S5	MDR ← MEM; PC ← PC + 1
S6	MAR ← MDR; MDR ← ACC
S7	MEM_WR;



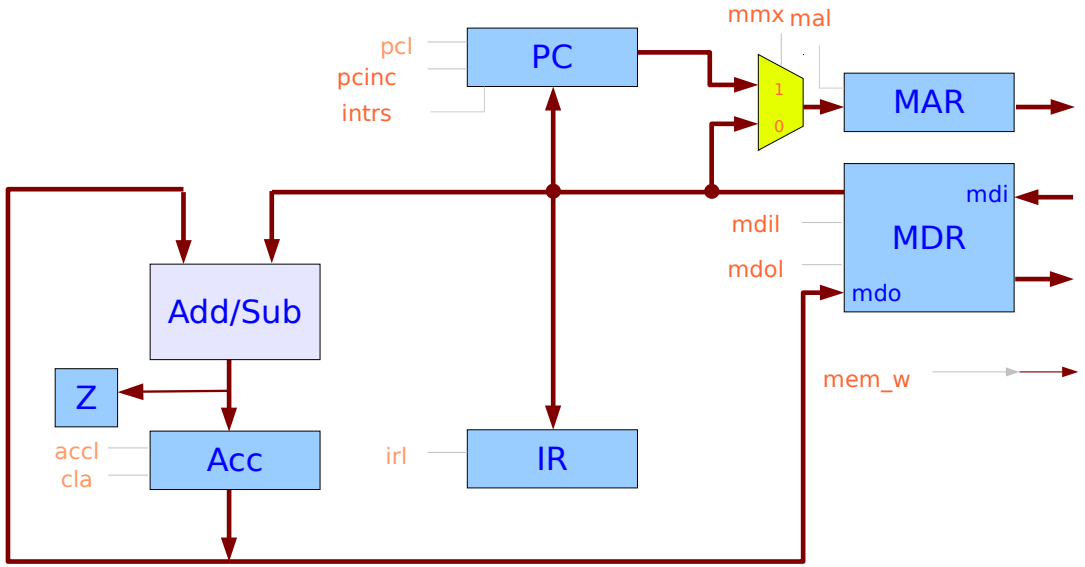
Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

Control Signal – CLA

S4 CLA

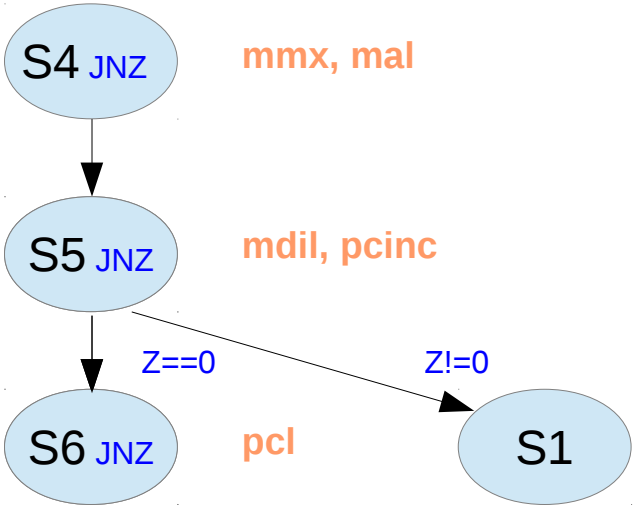
cla

S4 ACC ← “00”

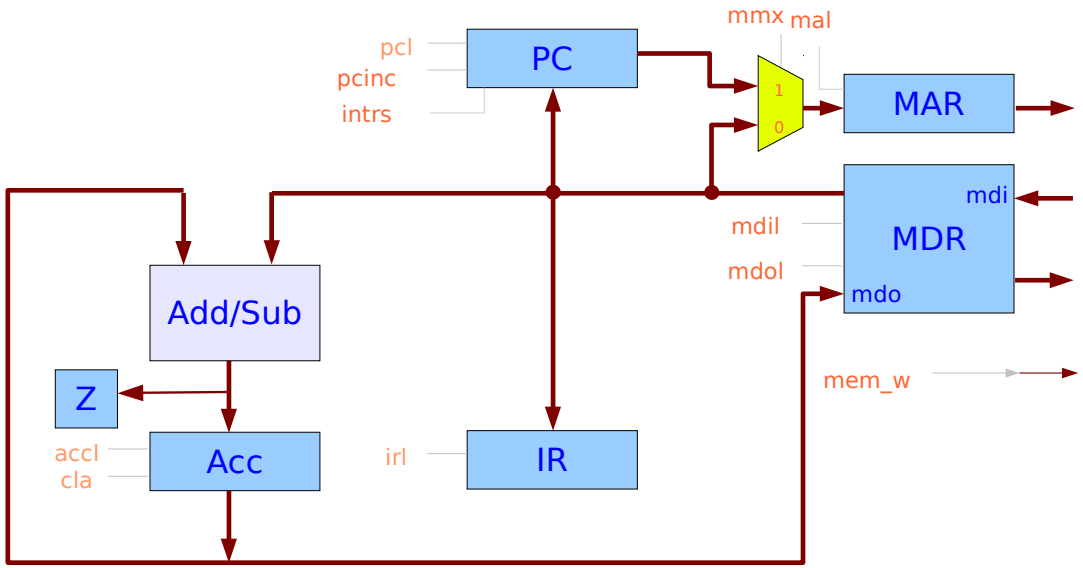


Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

Control Signal – JNZ



S4 MAR ← PC; MEM_RD;
S5 MDR ← MEM; PC ← PC + 1
S6 If (Z=='0') PC ← MDR;

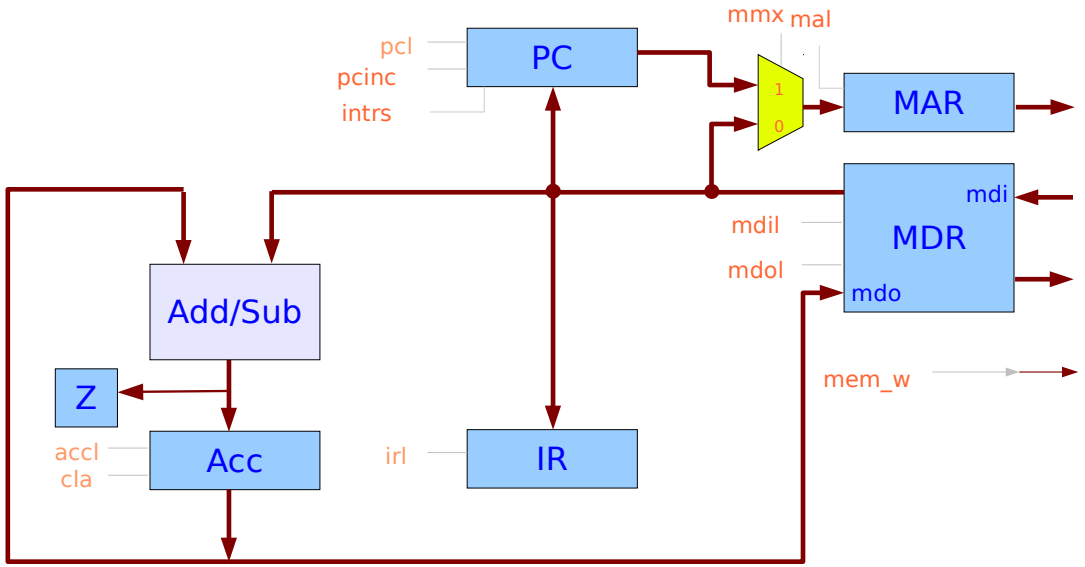


Based on http://www.ele.uri.edu/Courses/ele306/f01/Tinydoc.pdf

Control Signal – RST

intrs

S4 PC ← “00”



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
06   E0                  RST
78   01         ONE:     DAT      0000 0001    ;Constant 1
```

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