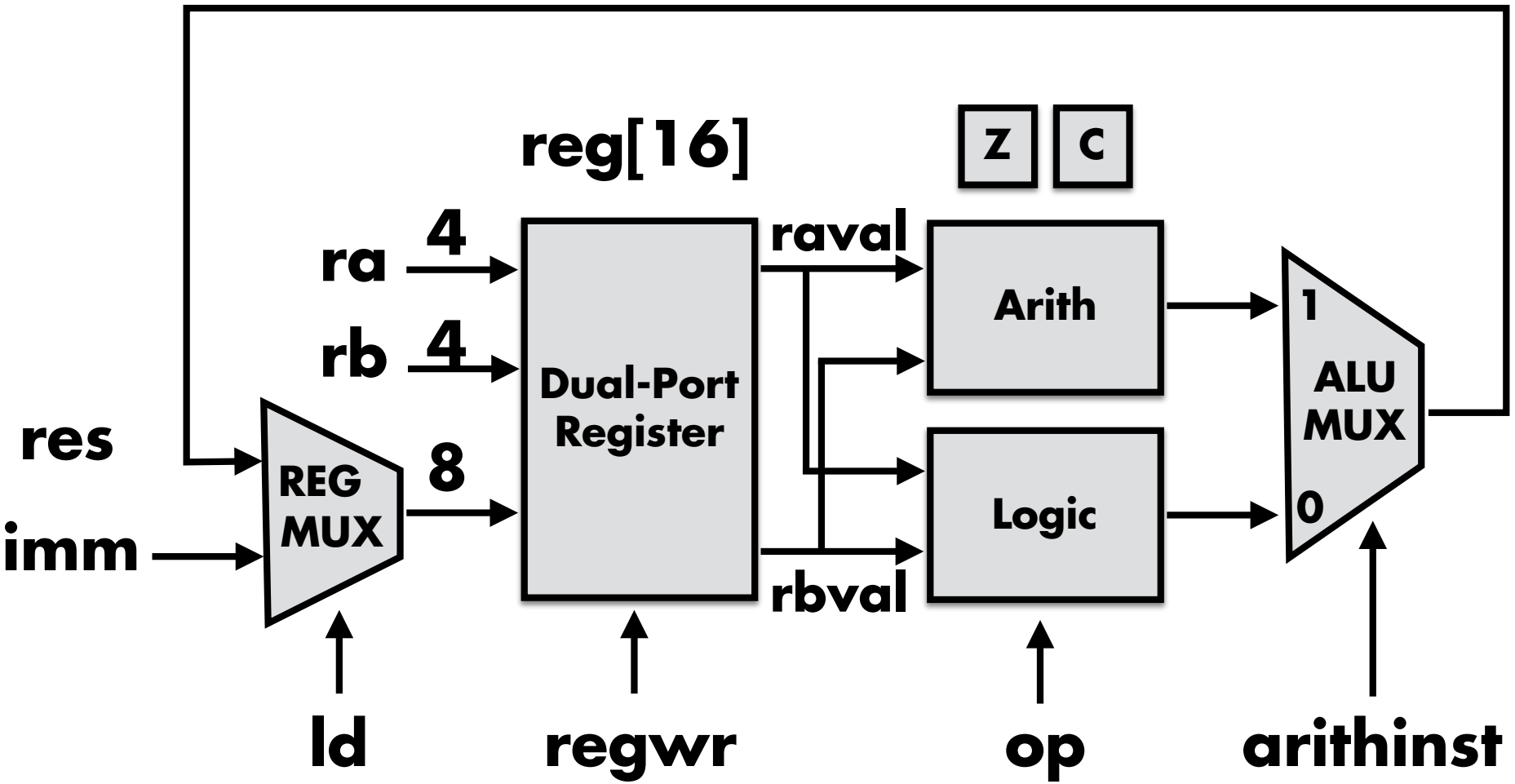
Pico40 Processor

Variation on a Theme by Ken Chapman

Pat Hanrahan

CS448H: Agile Hardware Design
Winter 2017

Pico40

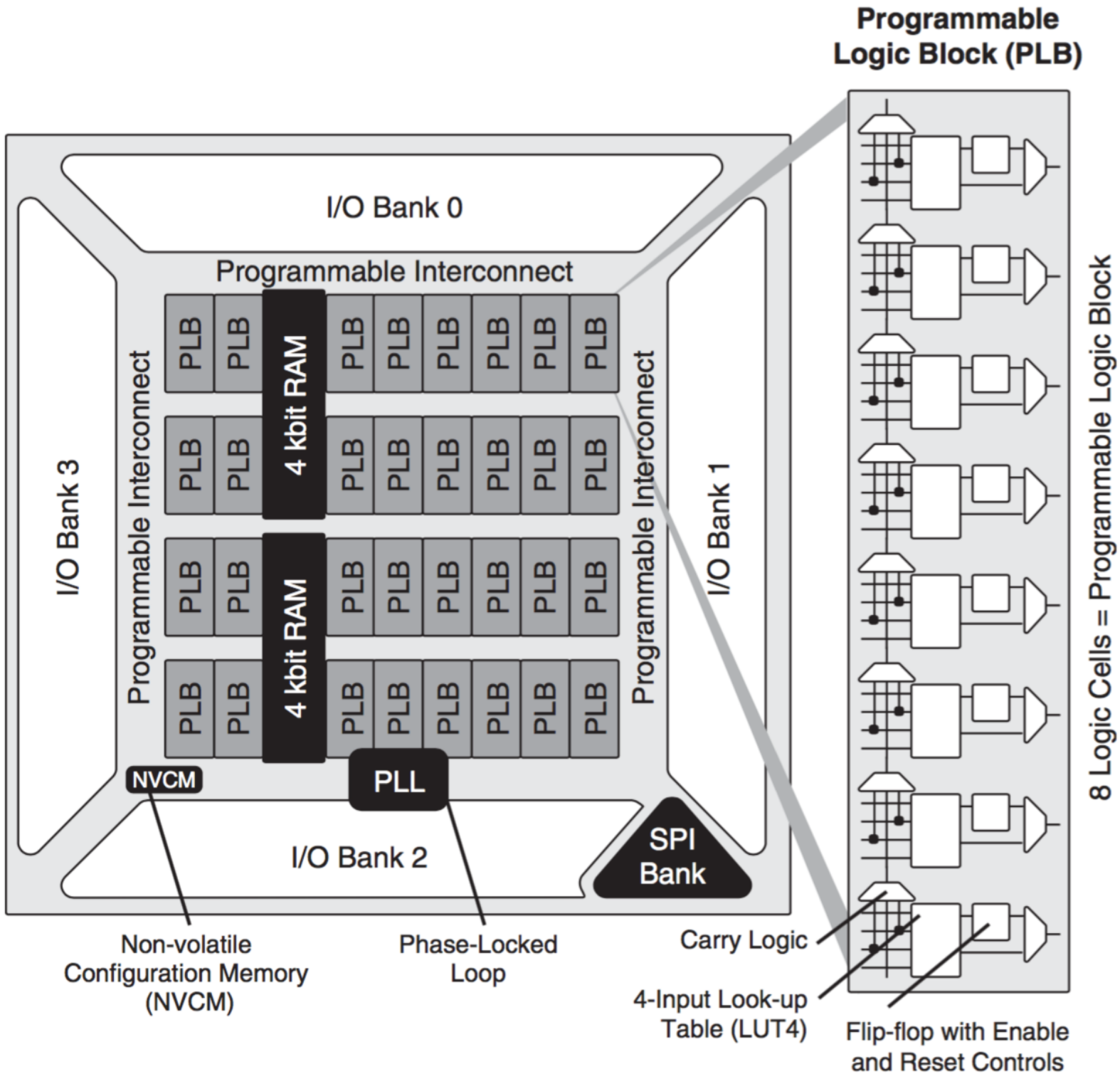

reg[16]
Z
C
ra
4
rb
4
raval
Arith
1
ALU
MUX
Dual-Port
Register
res
REG
MUX
8
Logic
0
imm
rbval
ld
regwr
op
arithinst

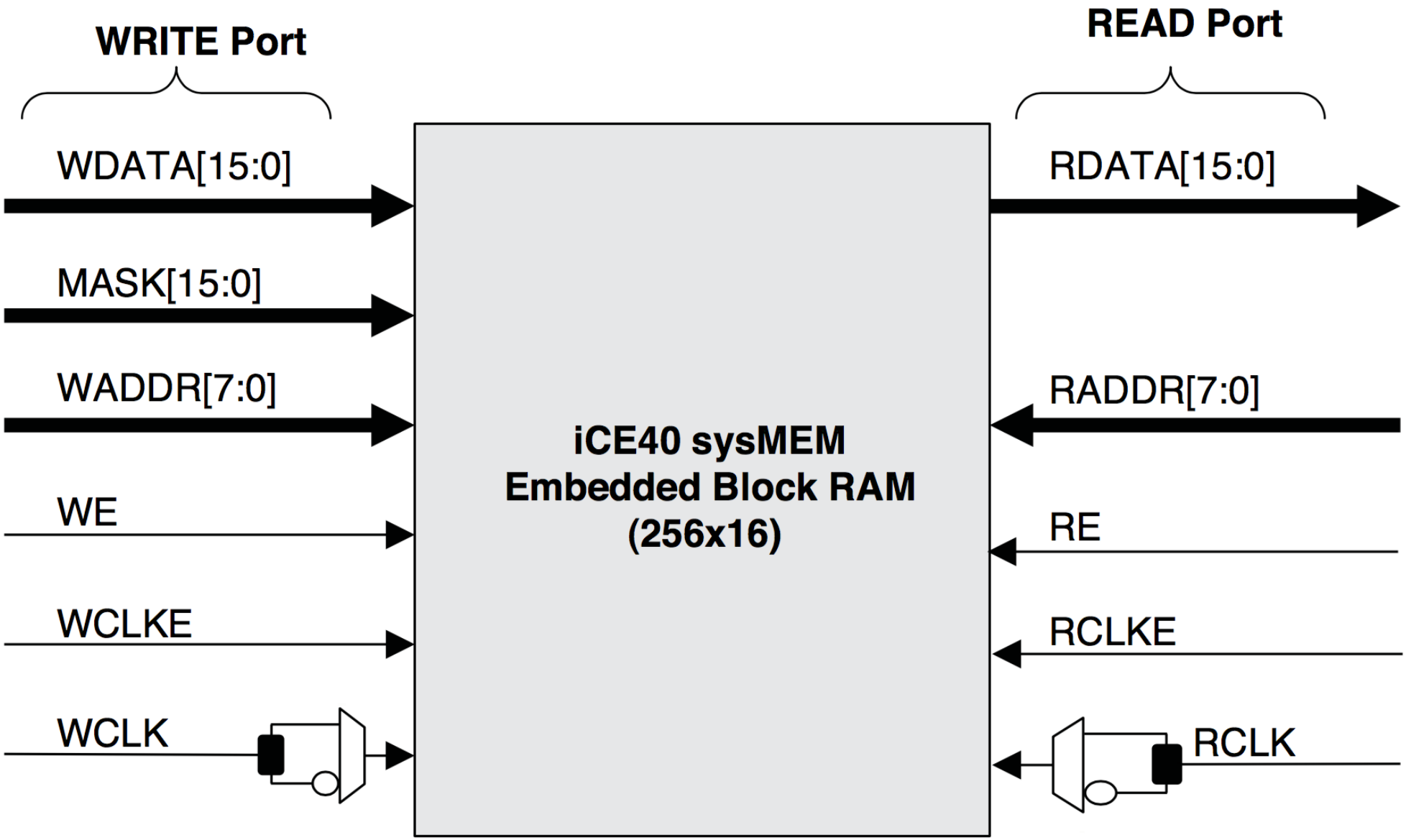
```
0000aaaabbbb0000 "mov"
0001aaaabbbb0000 "and_"
0010aaaabbbb0000 "or_"
0011aaaabbbb0000 "xor"

0100aaaabbbb0000 "add"
0101aaaabbbb0000 "sub"
0110aaaabbbb0000 "adc"
0111aaaabbbb0000 "sbc"

1000aaaaiiiiiiii "ldlo"
1001aaaaiiiiiiii "ldhi"
1010aaaaiiiiiiii "ld"
1011aaaaiiiiiiii "st"

1100cccciiiiiiii "jmpc"
1101cccciiiiiiii "callc"
1110cccc00000000 "retc"
```

```
from tiny import *

def mov ( a, b ):
    a = checku ( a , 4 )
    b = checku ( b , 4 )
    emit( 0x0 | (a<<8) | (b<<4) )

def and_ ( a, b ):
    a = checku ( a , 4 )
    b = checku ( b , 4 )
    emit( 0x1000 | (a<<8) | (b<<4) )

def or_ ( a, b ):
    a = checku ( a , 4 )
    b = checku ( b , 4 )
    emit( 0x2000 | (a<<8) | (b<<4) )

def xor ( a, b ):
    a = checku ( a , 4 )
    b = checku ( b , 4 )
    emit( 0x3000 | (a<<8) | (b<<4) )
```

```
from pico import *

def prog():
    ldlo(r0,0)
    ldlo(r1,1)
    loop = label()
    add(r0,r1)
    st(r0, 0)
    jmp(loop)

mem = assemble(prog)

save(mem, 'a.mem') // Show a.mem
```

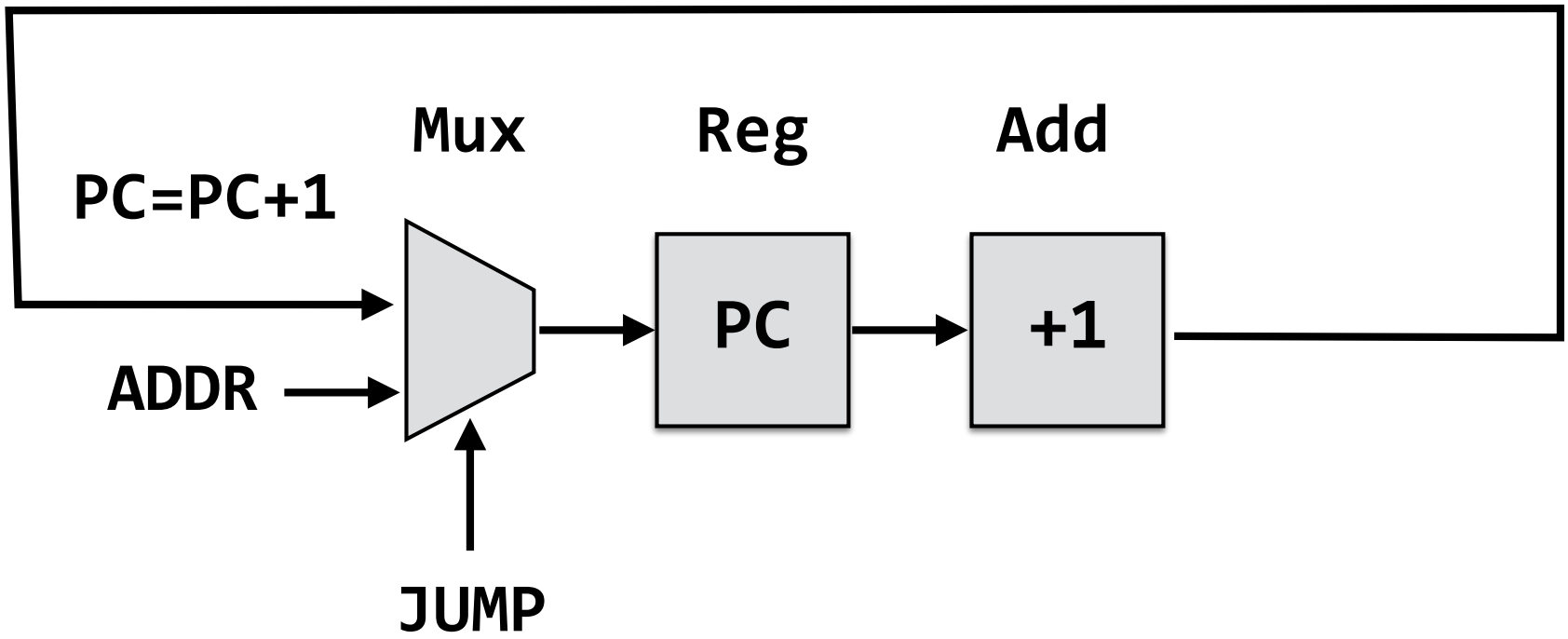
RAMB


Programmable
Logic Block (PLB)
I/O Bank 0
Programmable Interconnect
I/O Bank 3
Programmable Interconnect
Programmable Interconnect
I/O Bank 1
PLB
4 kbit RAM
4 kbit RAM
NVCM
PLL
I/O Bank 2
SPI
Bank
8 Logic Cells = Programmable Logic Block
Non-volatile
Configuration Memory
(NVCM)
Phase-Locked
Loop
Carry Logic
4-Input Look-up
Table (LUT4)
Flip-flop with Enable
and Reset Controls


WRITE Port
READ Port
WDATA[15:0]
MASK[15:0]
WADDR[7:0]
WE
WCLKE
WCLK
iCE40 sysMEM
Embedded Block RAM
(256x16)
RDATA[15:0]
RADDR[7:0]
RE
RCLKE
RCLK

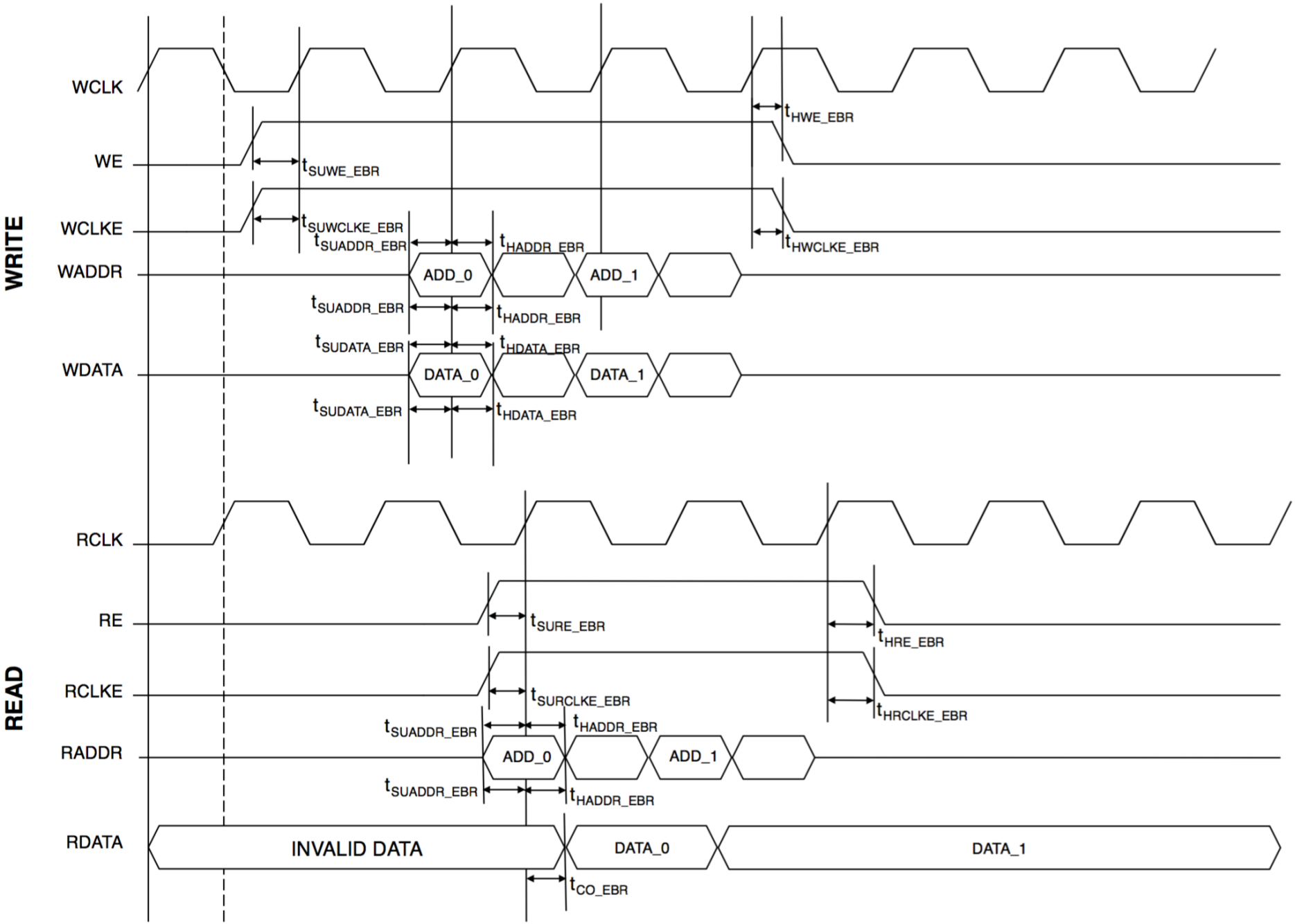
```
ADDRN, INSTN, N = 8, 16, 8

mem = readmem('a.mem', 1<<ADDRN)

# initialize program memory
romb = ROMB(mem) )
inst = romb.RDATA

# instruction decode
addr = inst[0:ADDRN]
imm = inst[0:N]
rb = inst[4:8]
ra = inst[8:12]
cc = inst[8:12]
op = inst[12:14]
insttype = inst[14:16]
```

Sequencer


Mux
Reg
Add
PC=PC+1
PC
+1
ADDR
JUMP

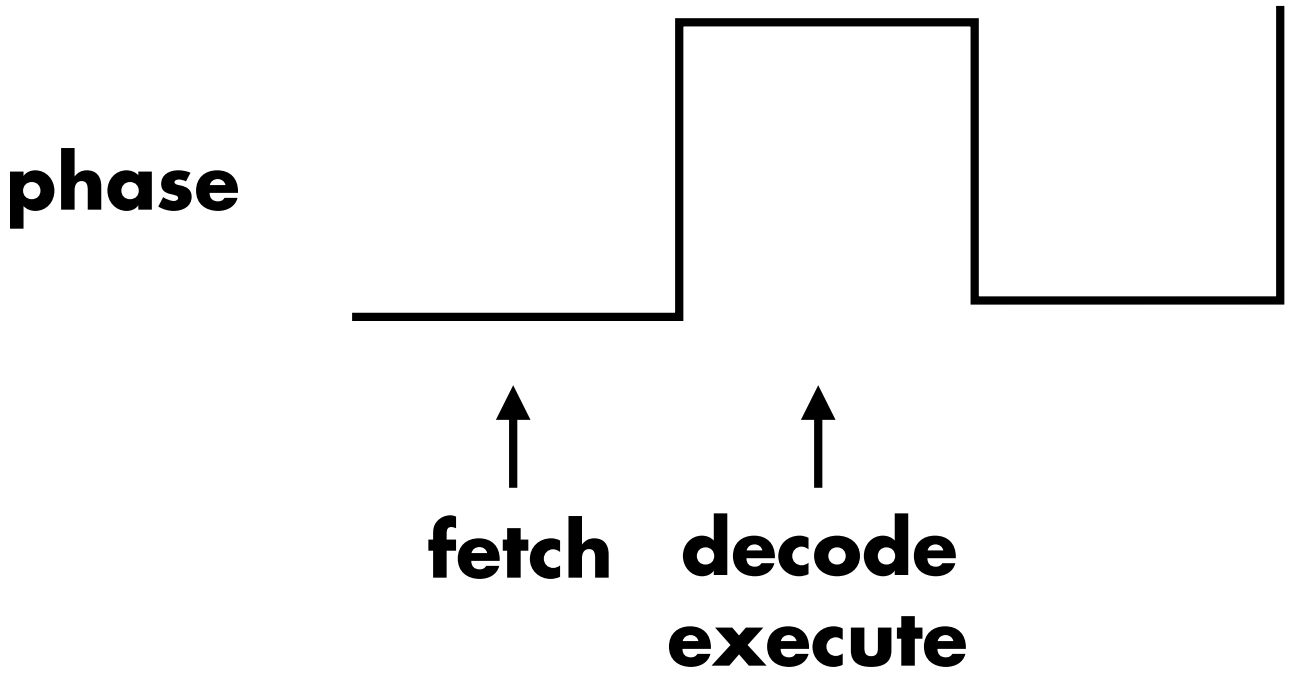
```
def Sequencer(n):
  pc = Register(n, ce=True)
  add = Add(n)
  add( pc, constarray(1,n) )
  mux = Mux(2, n)
  wire(add.O, mux.I0)
  pc(mux)

  return AnonymousCircuit("addr", mux.I1
                          "jump", mux.S,
                          "we",   pc.CE,
                          "O",    pc.O)
```

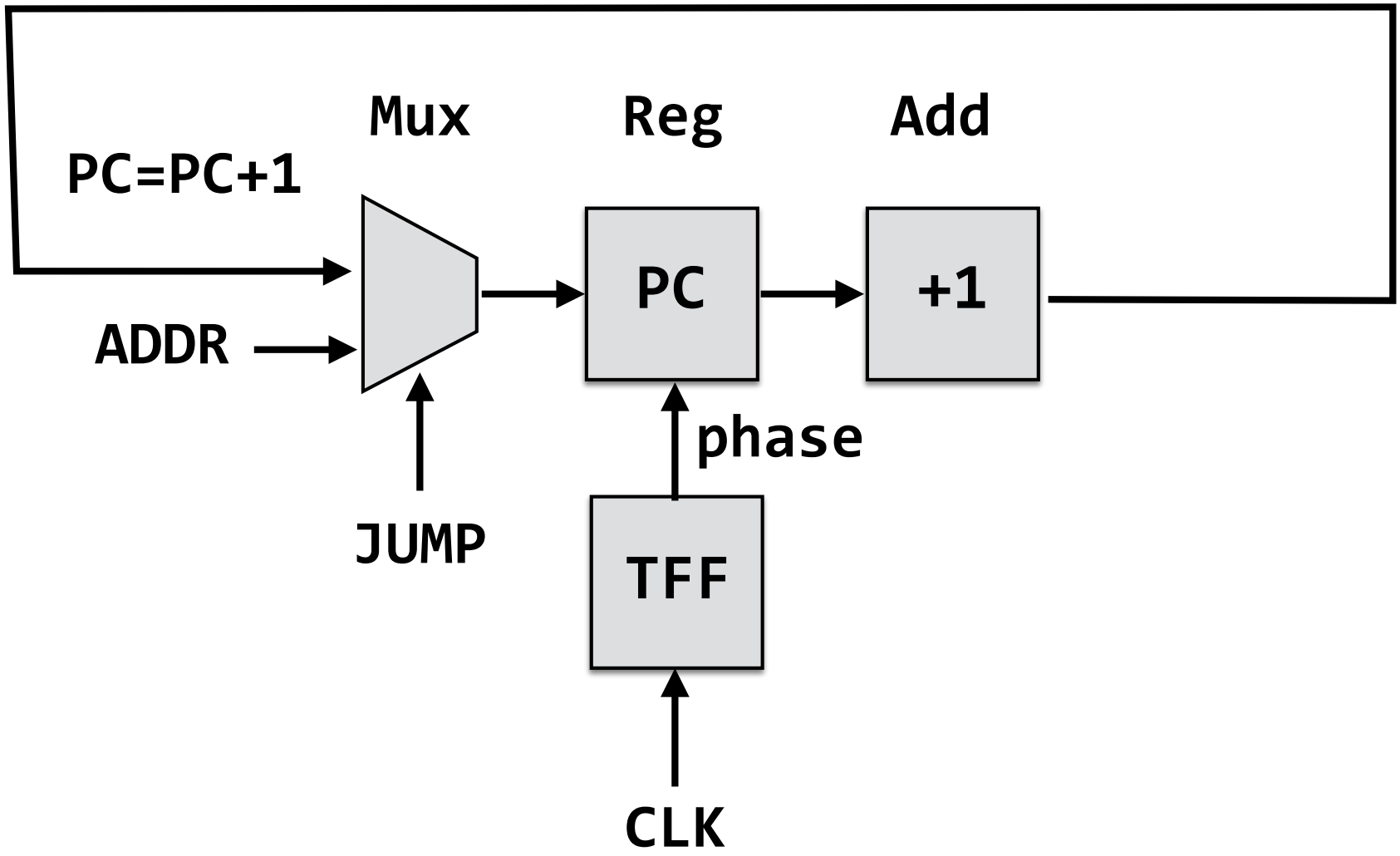
Figure 16-3. EBR Module Timing Diagram[1]



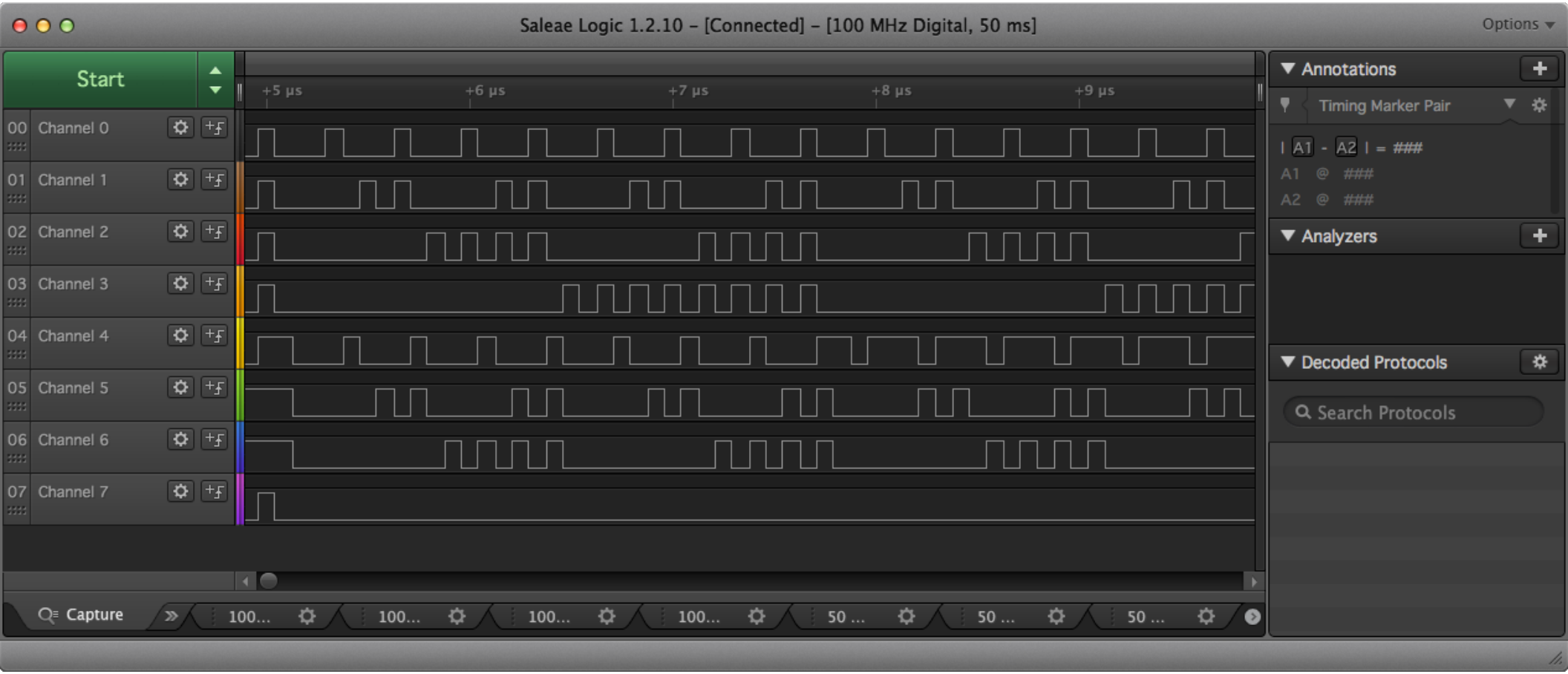
RADDR=pc

inst=RAMB[RADDR]

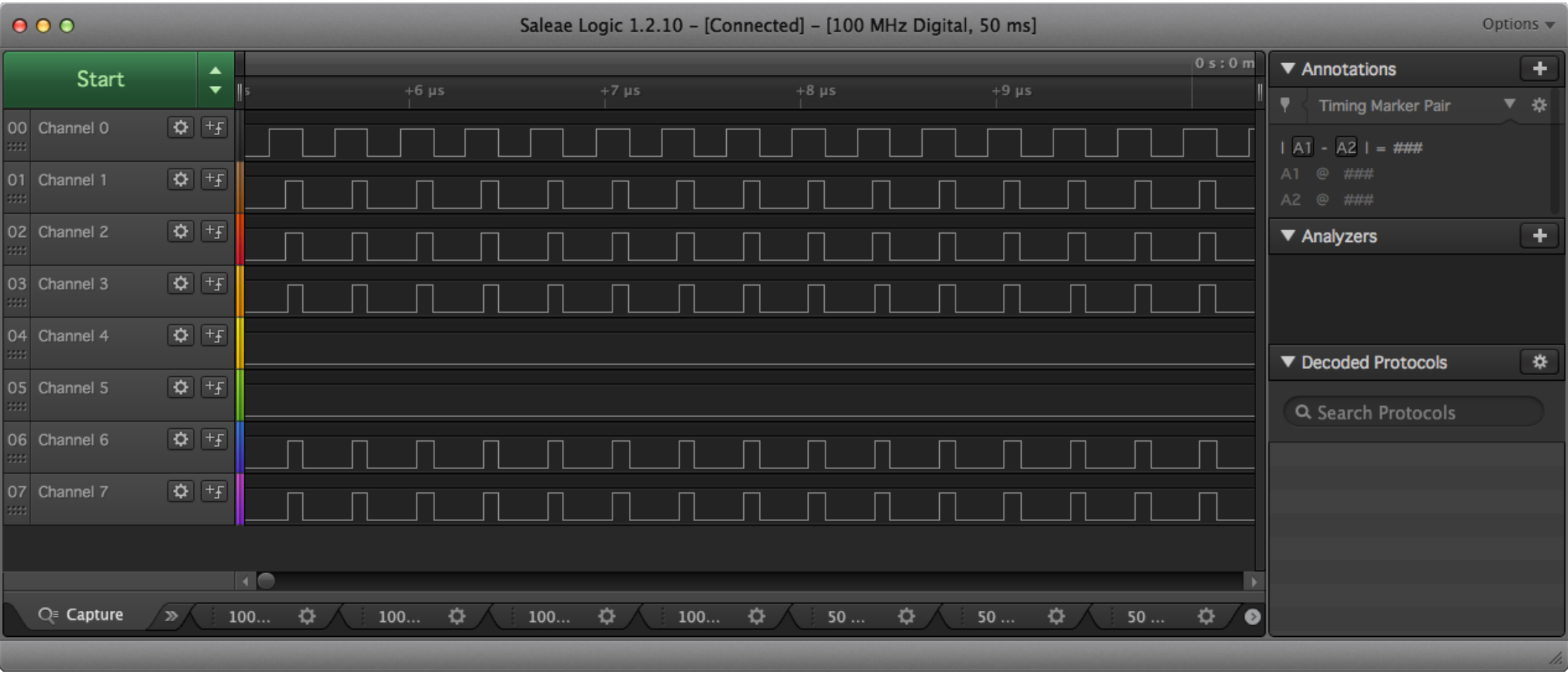



Mux
Reg
Add
PC=PC+1
ADDR
PC
+1
phase
JUMP
TFF
CLK

alu.py


Saleae Logic 1.2.10 – [Connected] – [100 MHz Digital, 50 ms]
Options
Start
+5 µs
+6 µs
+7 µs
+8 µs
+9 µs
00 Channel 0
01 Channel 1
02 Channel 2
03 Channel 3
04 Channel 4
05 Channel 5
06 Channel 6
07 Channel 7
Annotations
Timing Marker Pair
| A1 - A2 | = ###
A1 @ ###
A2 @ ###
Analyzers
Decoded Protocols
Search Protocols
Capture
100...
100...
100...
100...
50 ...
50 ...
50 ...

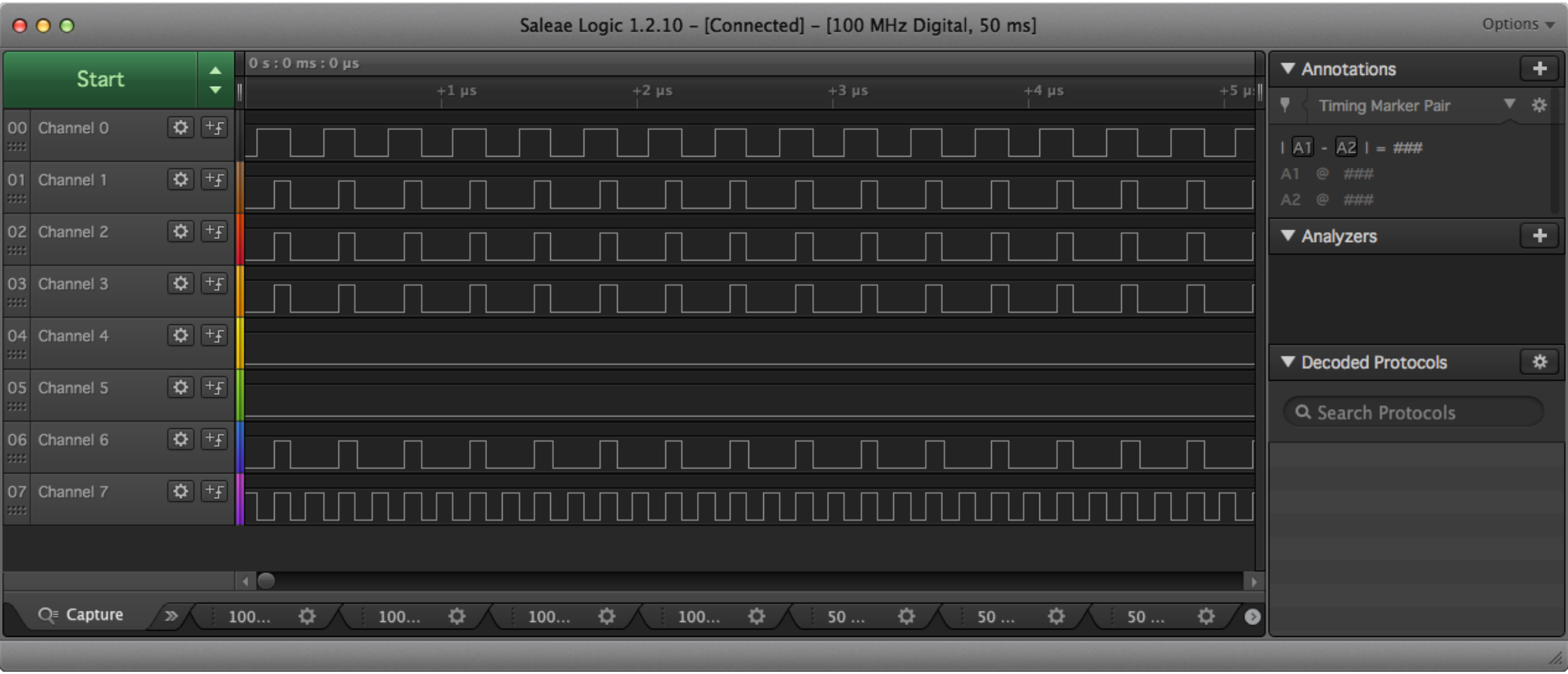
jmp.py


Saleae Logic 1.2.10 – [Connected] – [100 MHz Digital, 50 ms]
Options
Start
+6 µs
+7 µs
+8 µs
+9 µs
00 Channel 0
01 Channel 1
02 Channel 2
03 Channel 3
04 Channel 4
05 Channel 5
06 Channel 6
07 Channel 7
Annotations
Timing Marker Pair
Analyzers
Decoded Protocols
Search Protocols
Capture

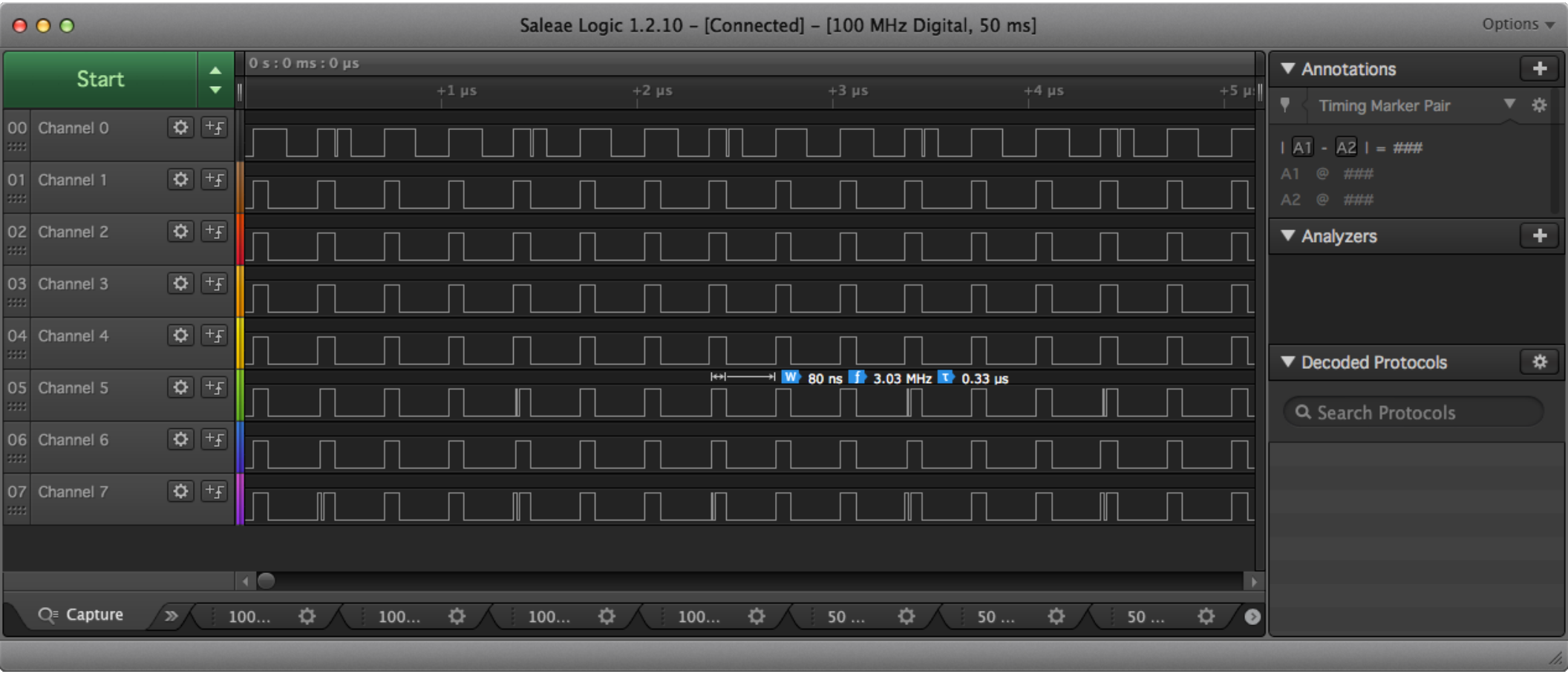
Dual-Ported Registers

`ram.py`

ldlo.py (show pc and inst)



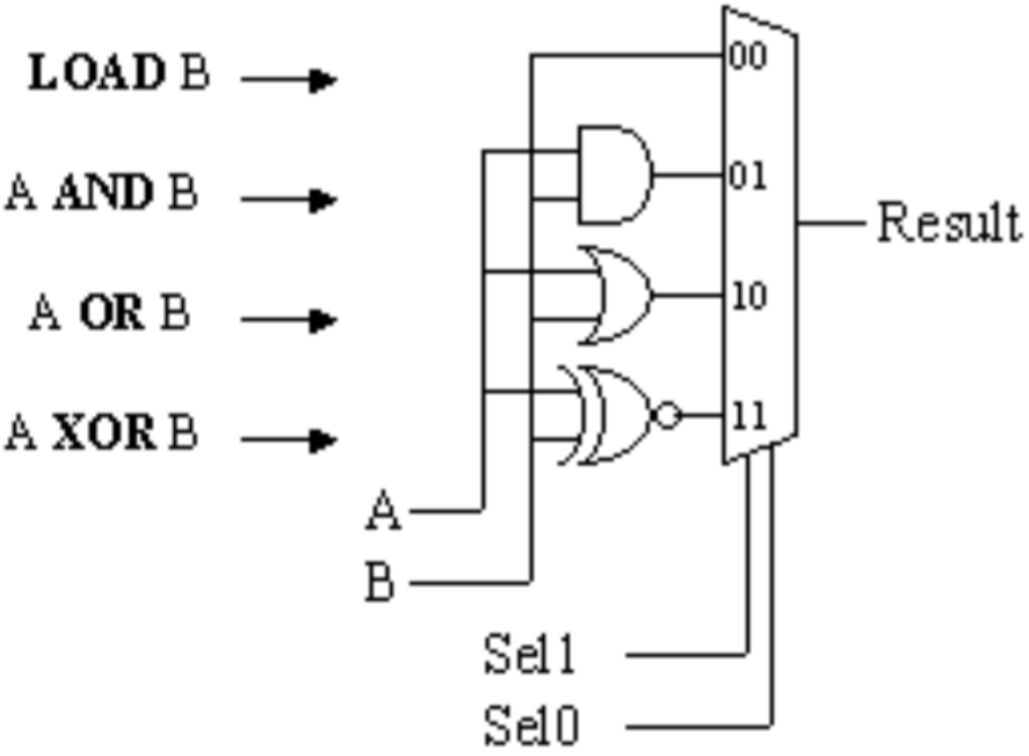
ldlo.py (show pc and res)



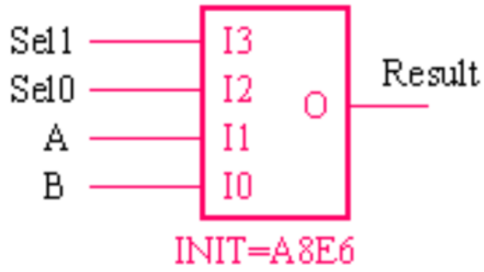
Note glitches!

Logic Unit

Logic Unit



Logical Group					
I3	I2	I1	I0	O	INIT=A8E6
Sel1	Sel0	A	B	Result	
0	0	0	0	0	6
0	0	0	1	1	
0	0	1	0	1	
0	0	1	1	0	
0	1	0	0	0	E
0	1	0	1	1	
0	1	1	0	1	
0	1	1	1	1	
1	0	0	0	0	8
1	0	0	1	0	
1	0	1	0	0	
1	0	1	1	1	
1	1	0	0	0	A
1	1	0	1	1	
1	1	1	0	0	
1	1	1	1	1	



```
def logicfunc(A, B, S0, S1):
    if S1 == 0:
        if S0 == 0:
            return B
        else:
            return A&B
    else:
        if S0 == 0:
            return A|B
        else:
            return A^B

def Logic(n):
    def logic(y):
        return LUT4(logicfunc)
    return braid(col(logic, n),
                 joinargs=['I0', 'I1'],
                 forkargs=['I2', 'I3'])
```

logic.py
mov.py
and.py
or.py
xor.py

Arith Unit

arith.py
add.py
sub.py
count.py

Condition Codes

blink.py

PLBs/LUTs

Decode
Sequencer
Registers
Conditions
Logic
Arith
ALU
MUXes
Output