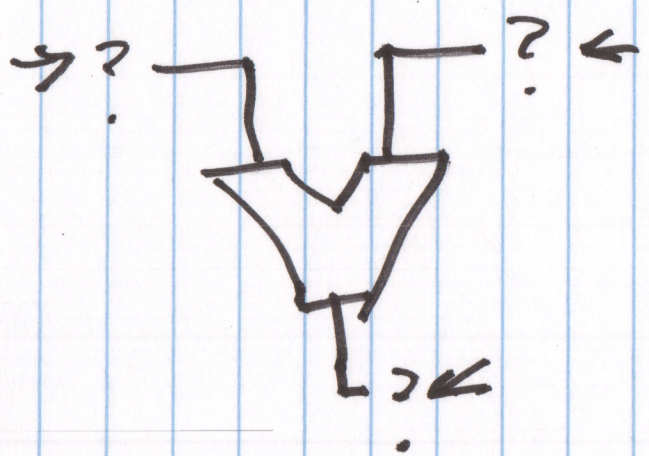
Announcements

1. Attend Recitation this week
2. Exam I moved Sept 29th

Today: Finish Ch.2
Begin Ch. 3 - start reading

Styles of ISAs

operands per instruction



3-Operand

LC-3, LC-2200, ARM v7

ADD Dest, Src1, Src2

2 operand ISA

ADD R1, R2 $\quad R1 \leftarrow R1 + R2$

x86

1 operand ISA

Implicit calculation register
Accumulator - Acc

ADD R2 $\quad Acc \leftarrow Acc + R2$

$A \leftarrow B + C$

```
LDB
ADD C
ST  A
```

0-operand ISA

Polish notation

1
2
+
3
4
+
*

(1+2)*(3+4)

Push 1
Push 2
ADD
PUSH 3
PUSH 4
ADD
MUL

*
+
1
2
+
3
4

(1+2)*(3+4)

Stack in memory

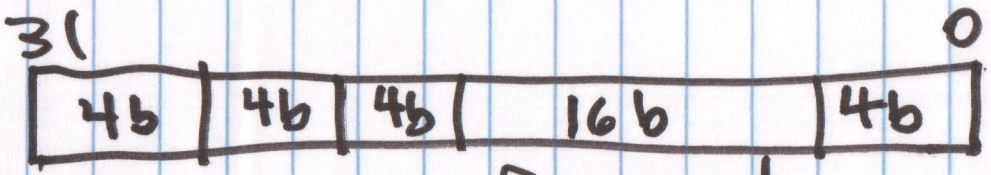
$1000

JVM

Instruction encoding

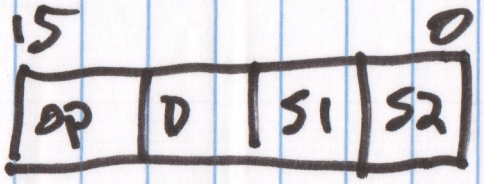
Why use Accumulator or Stack ISA in 2016?

Memory or comm B/W is expensive

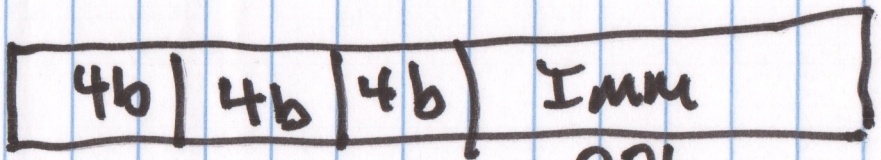


LC-2200

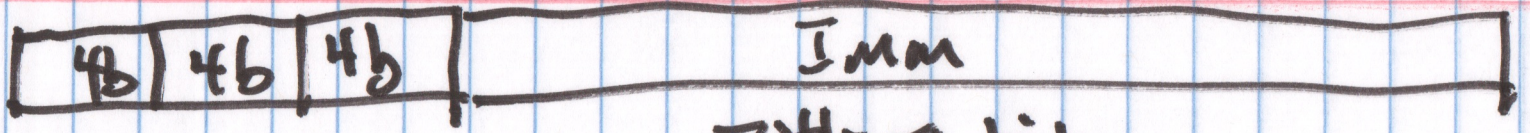
Var. length encoding



shorter - less wasteful



Good for long Immediate



~~Zillion~~ bits

32b + 12 = 44b

Variable length instructions

\+ :) Low memory use

\- xx

Added loop for var. length

1. Fetch from memory x bits
2. Decode to figure out what to do
3. Execute instruction

Fix all inst at x bits in length, this arrow goes away

Implement Processor (Ch. 3)

ISA restricts the hardware implementation

DOES NOT dictate the implementation

x86, 8088, 286, 386, 486, Pentium

Fixed width easy to decode style

Reduced Inst. Set Computer RISC

Complex " " " CISC

P6 → PentiumPro

ISA implementation is called a "Microarchitecture"

Want

Speed, Low energy consumption (battery life), low power dissipation (cooling), low cost

Speed is only concern — supercomputers, Data center servers

Embedded — IoT edge - low energy, low cost

↑

vs simple

Logic Combinational logic

Time-independent, binary function

$f(x, y, z) = w$

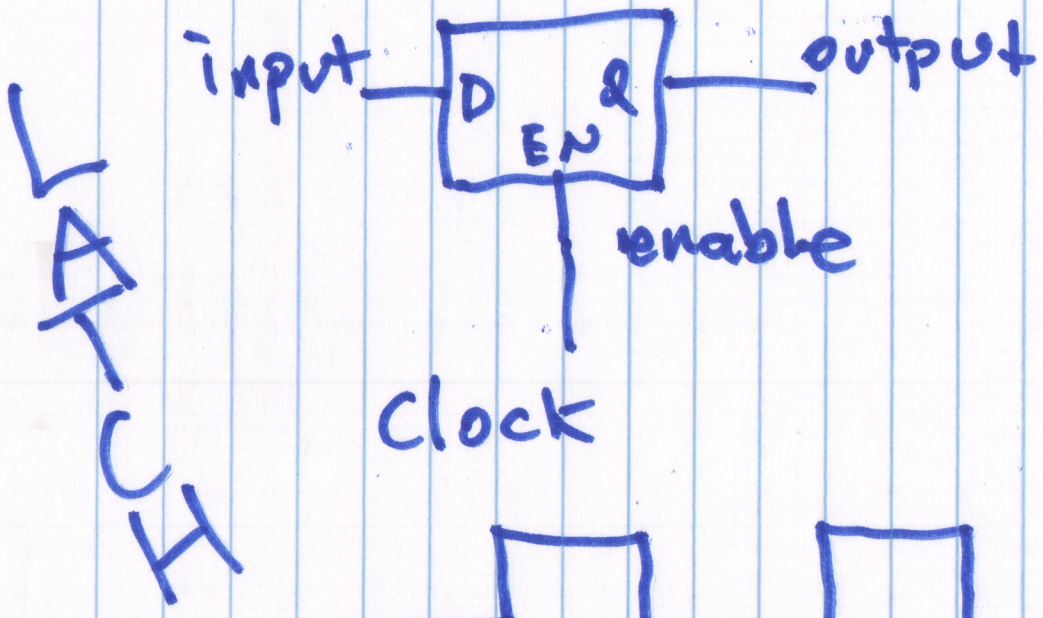
Sequential logic

$f(x, y, z, \text{past inputs}) = w$

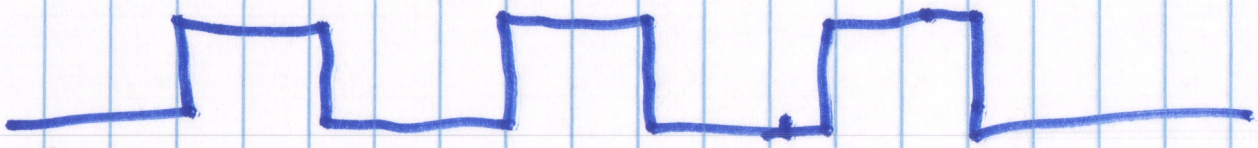
Use both kinds to build a Turing machine

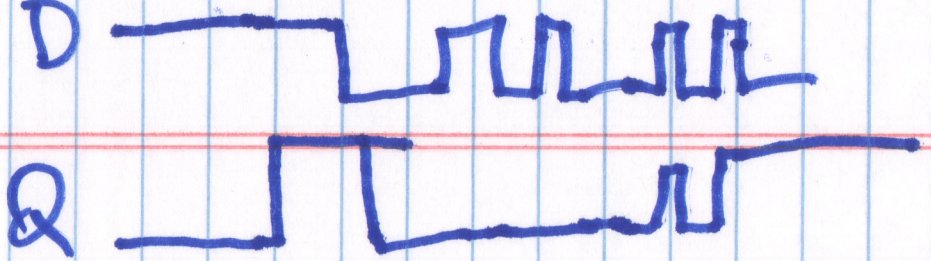
Memory elements

two kinds



D	Q	EN
0	0	1
1	1	1
x	old Q	0

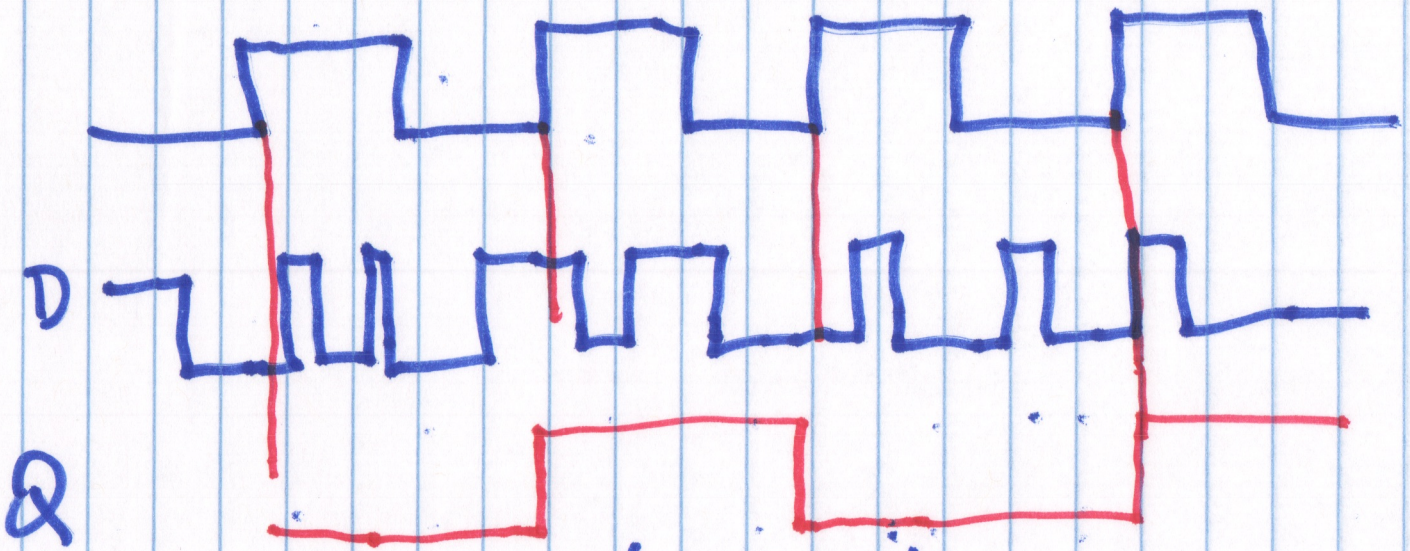




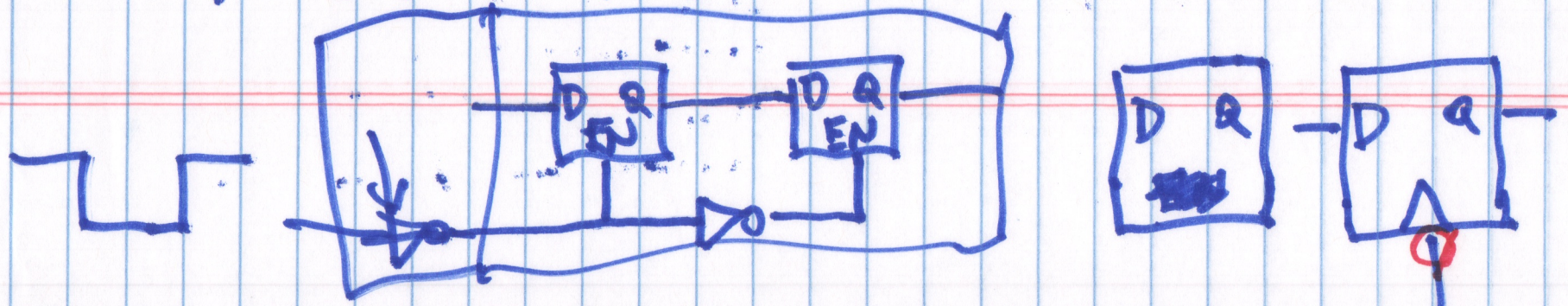
wire with delay as long as clock is high

FLIP FLOP

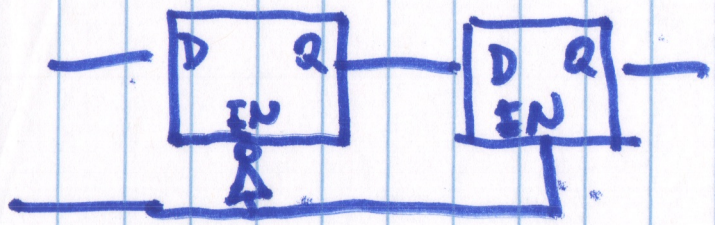
Sometimes want to sample D at an instant in time and then stop listening



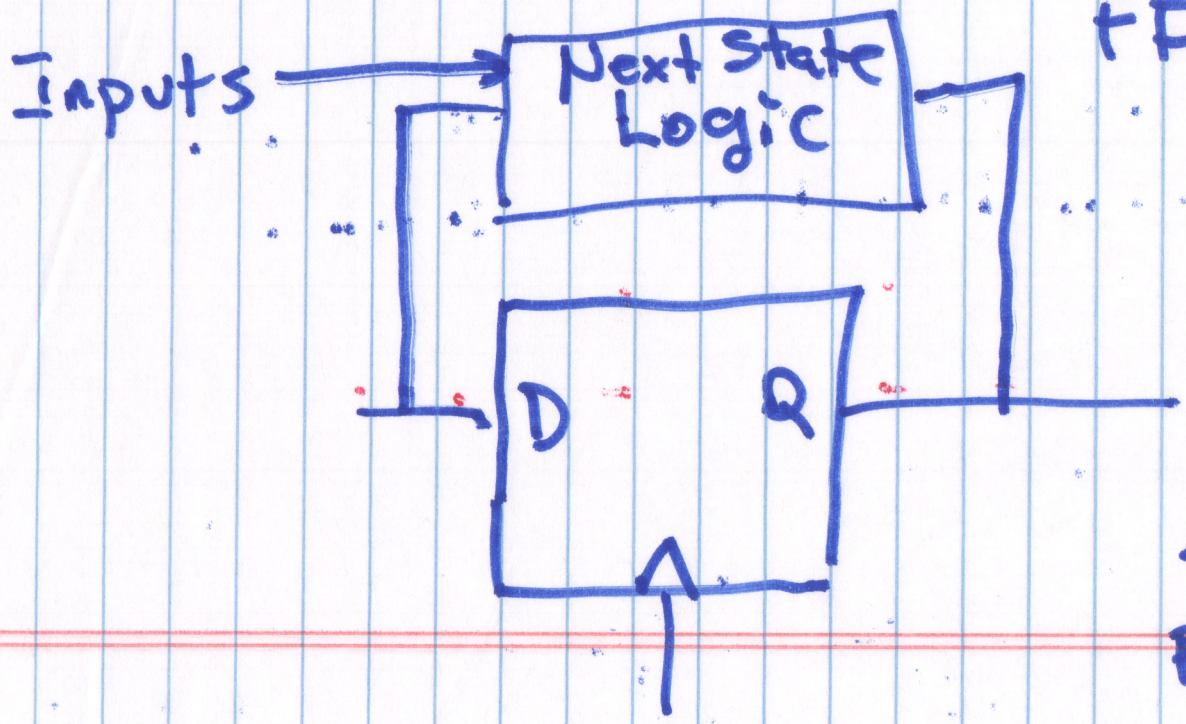
Clock edge triggered



Alternative Implementation



Positive edge triggered FF



State Machine

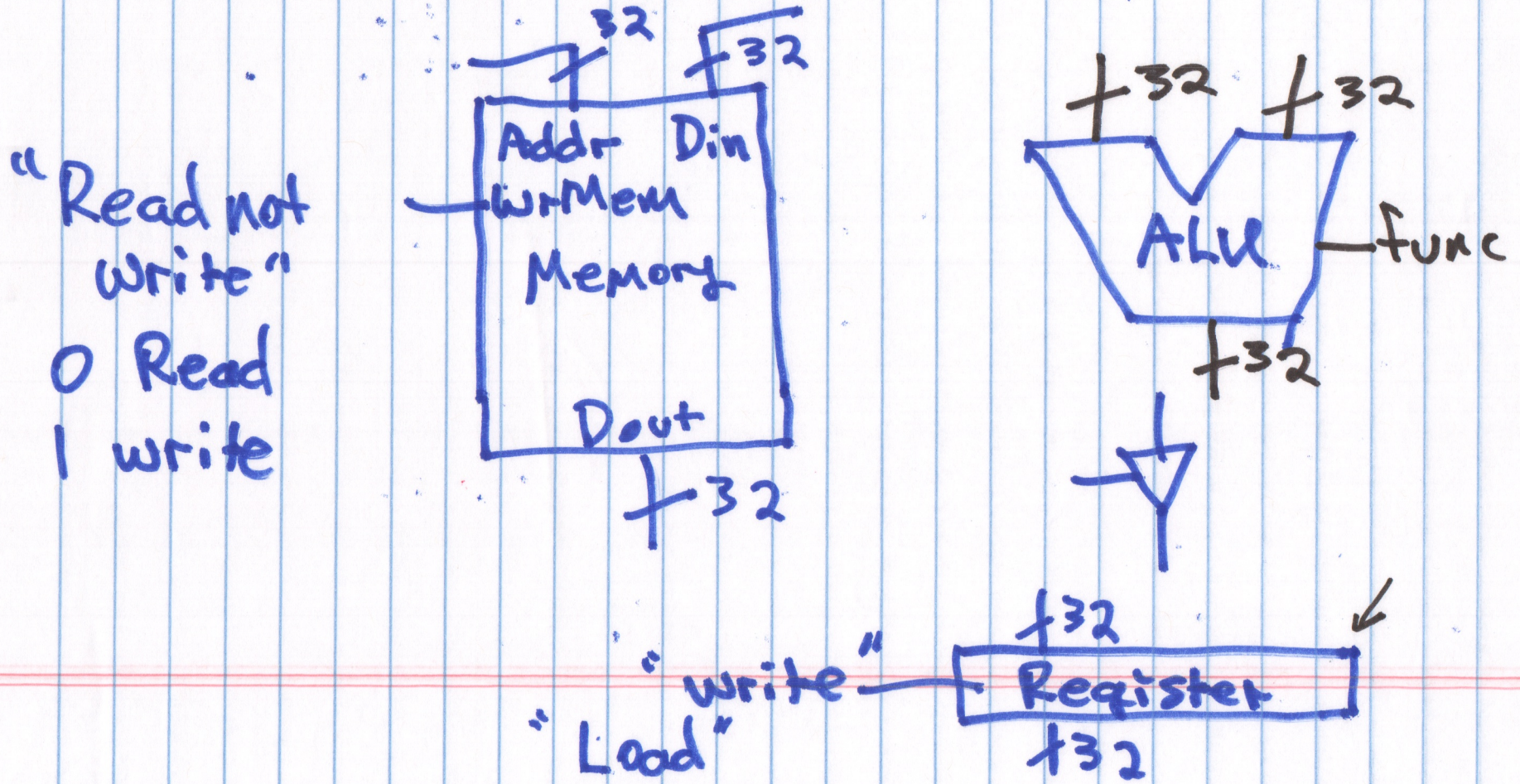
Use FF

Only one state advanced per clock

Why ever use a latch?

Cheaper than a FF (by a factor of ~2)

Use for memories

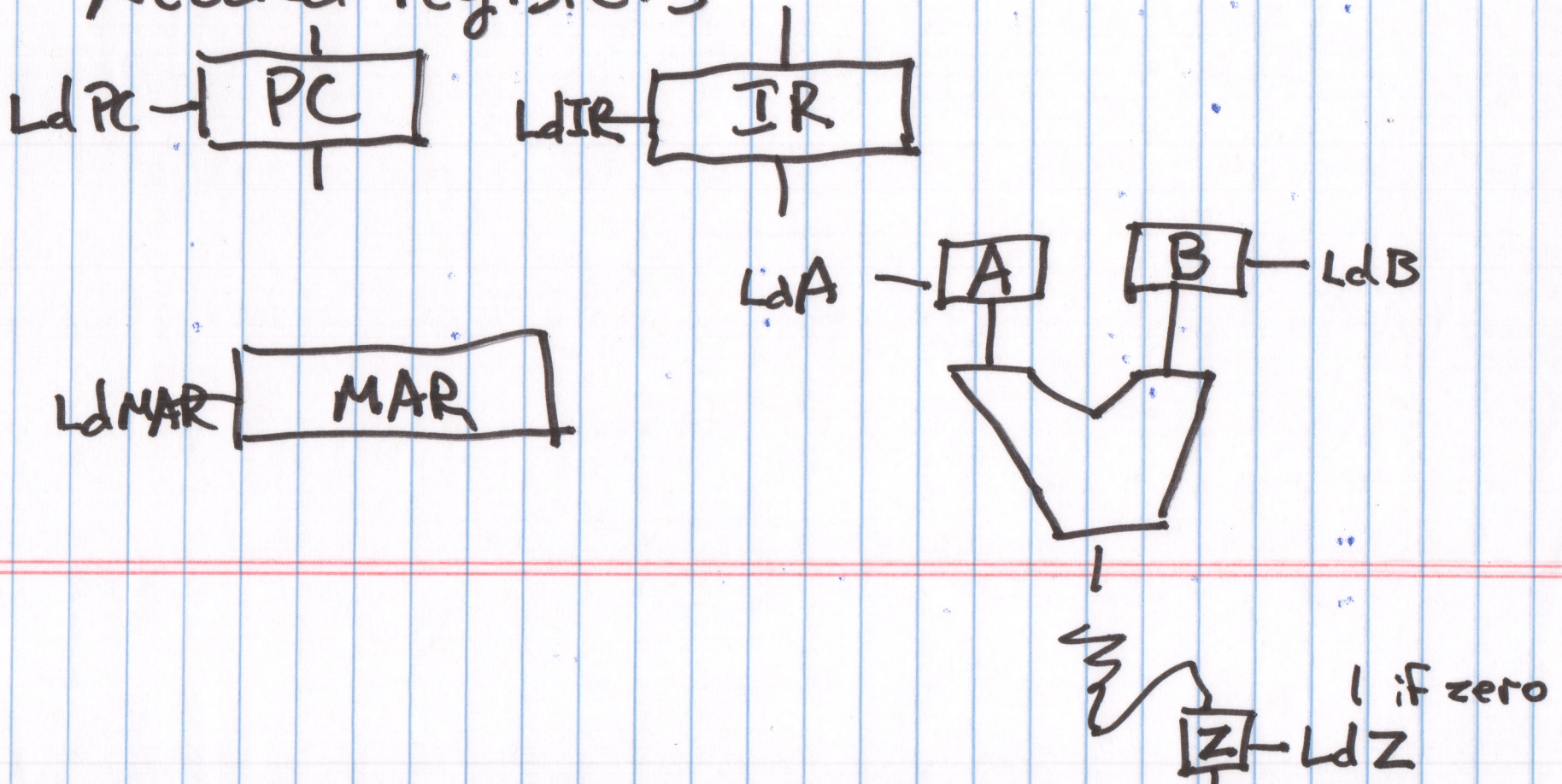


Announcements:

Exam moved to Sept. 29th

HW #2 posted, Due at Recitation

Needed registers



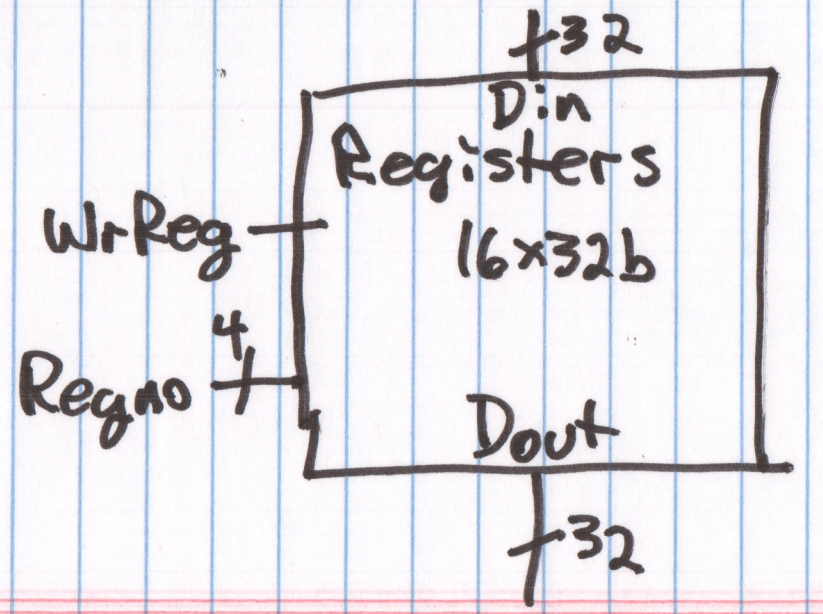
Datapath

Processes data

Oh yeah, and

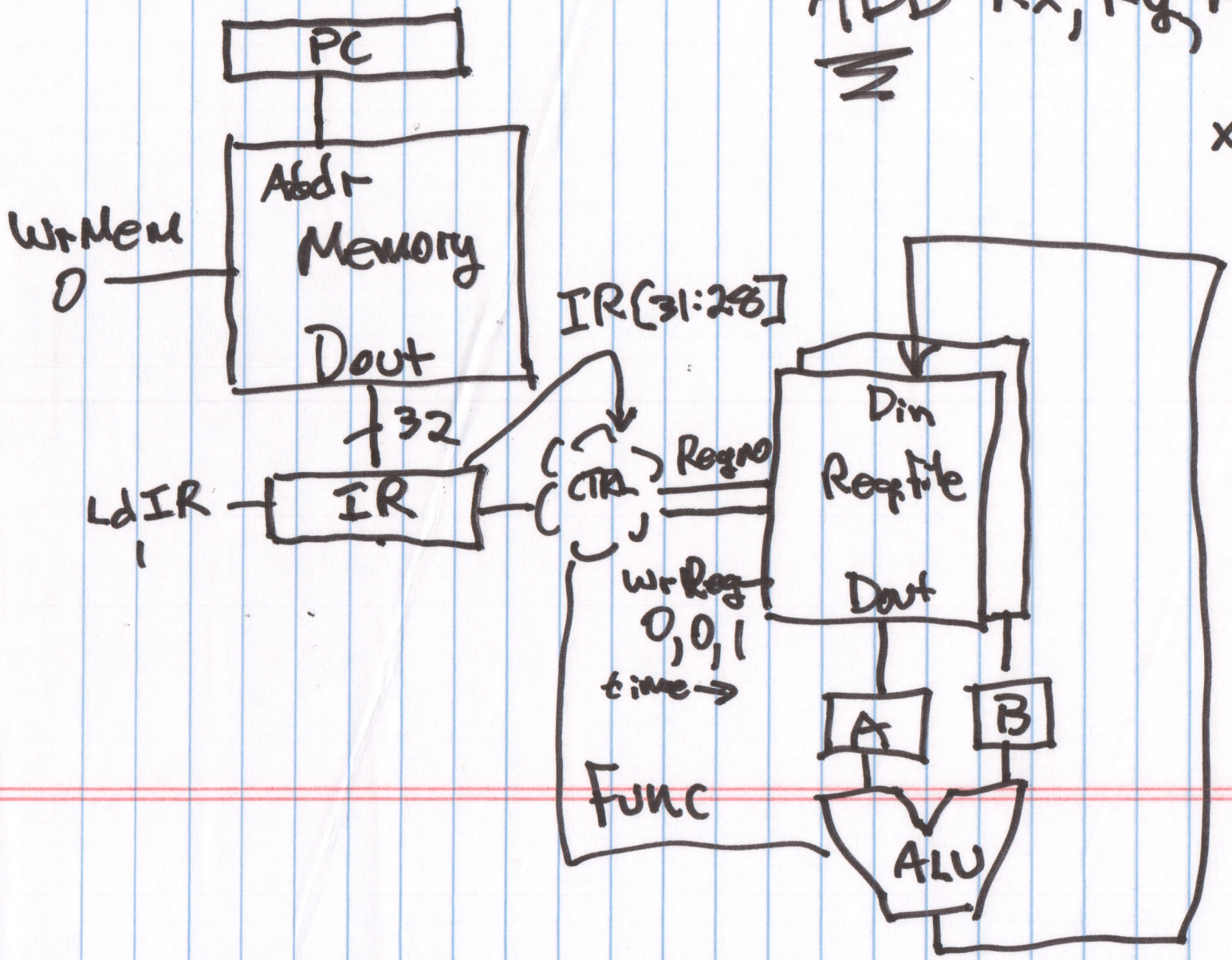
Control path

State machine



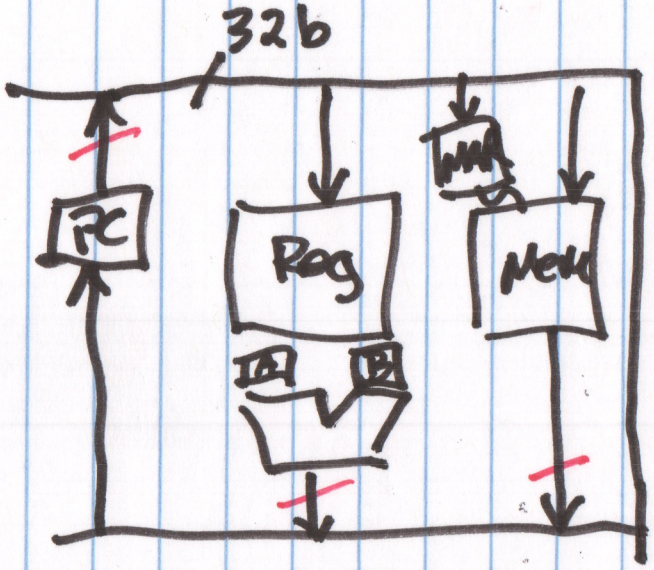
The story of an ADD instruction

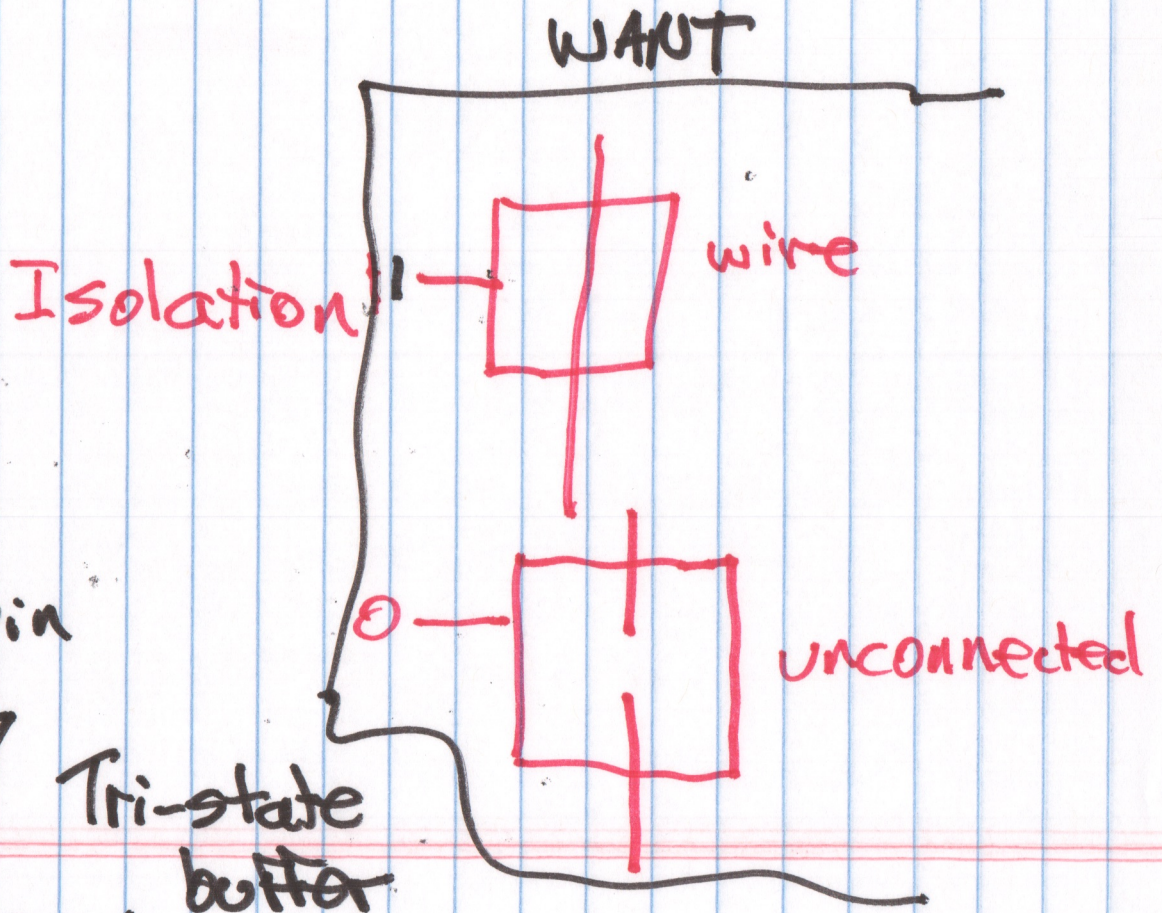
ADD R_x, R_y, R_z

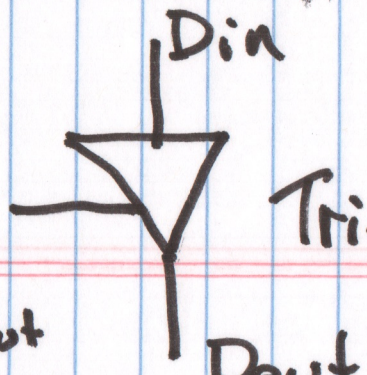


Interconnect blocks

Directly connect everything - spaghetti



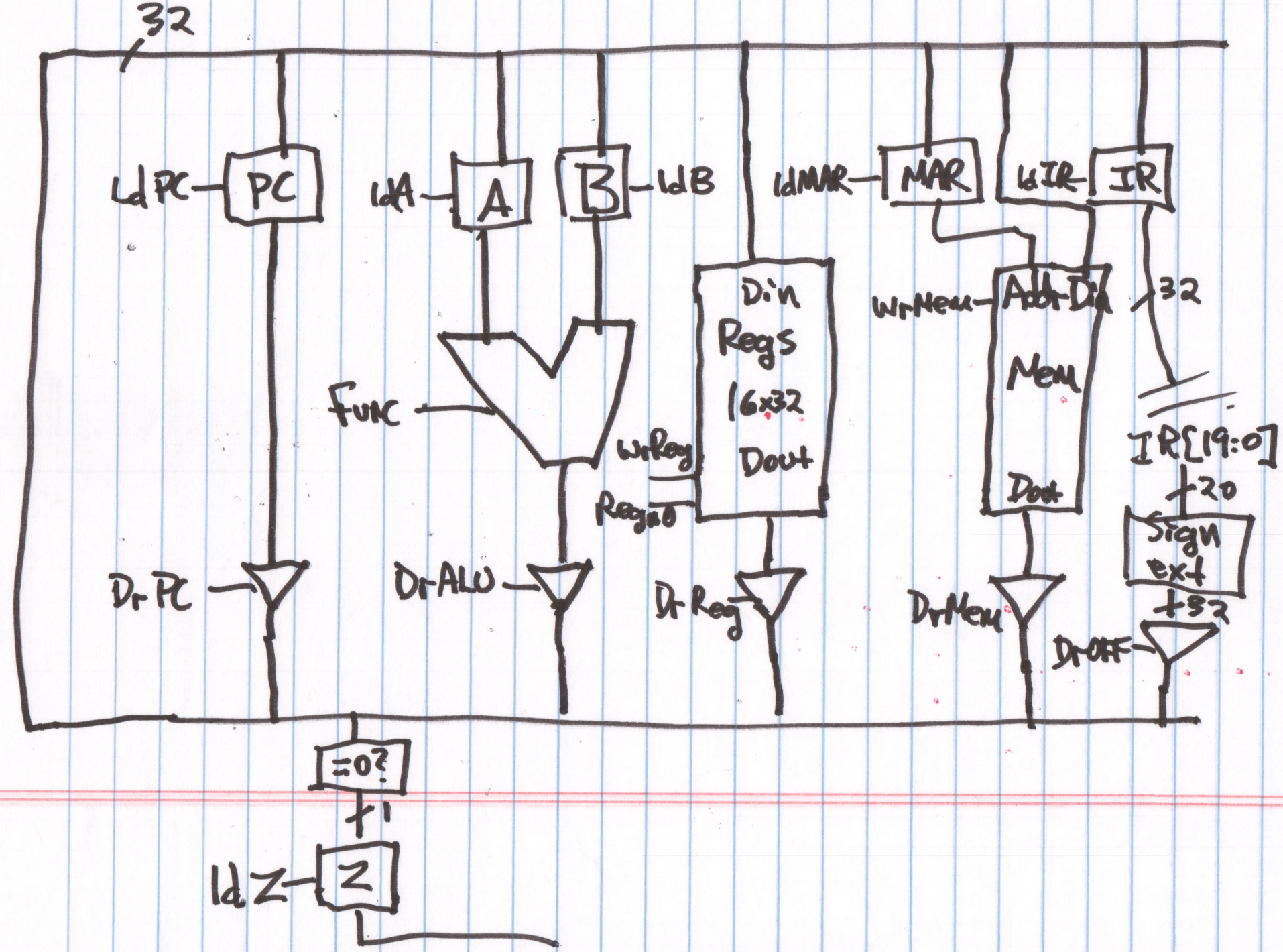


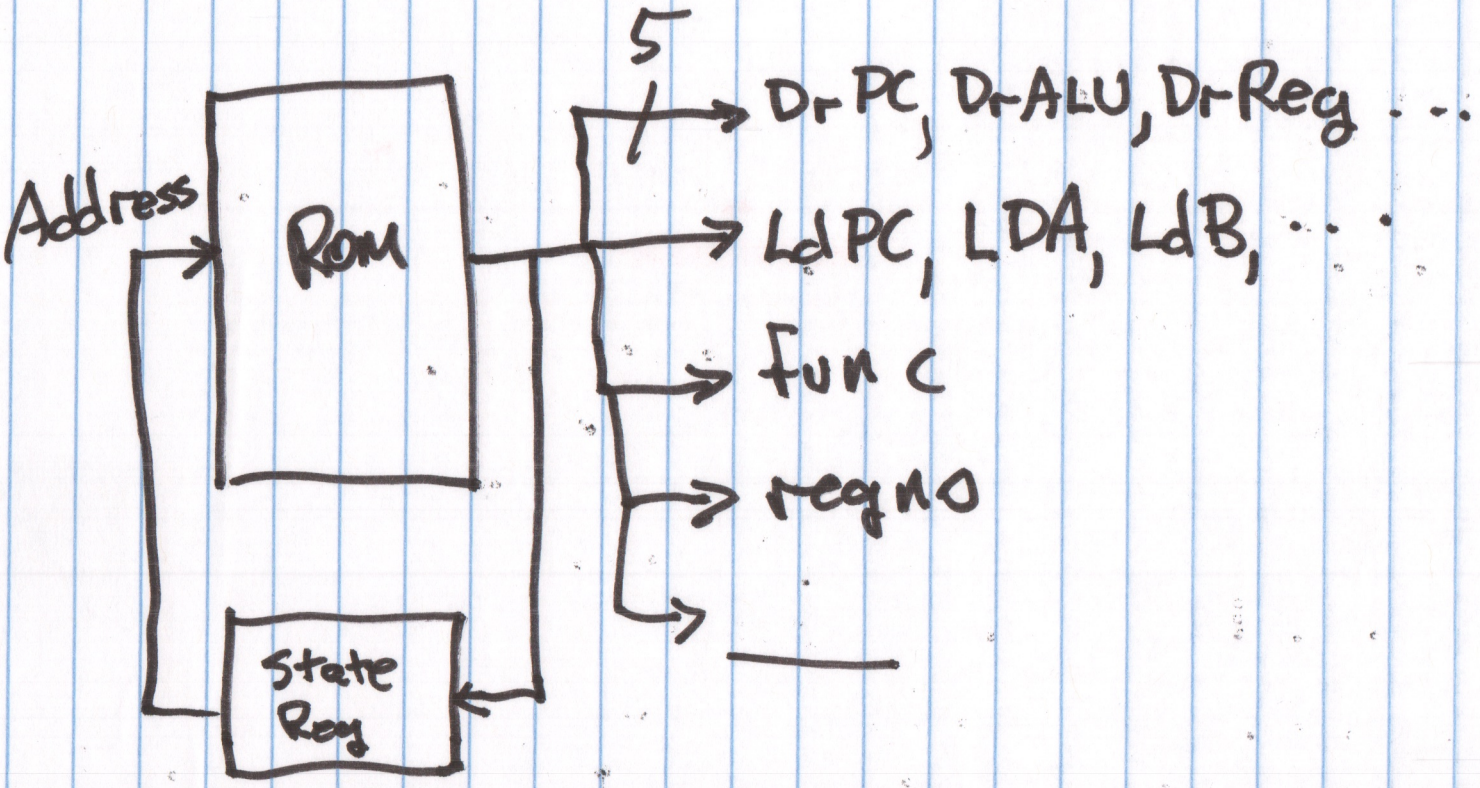


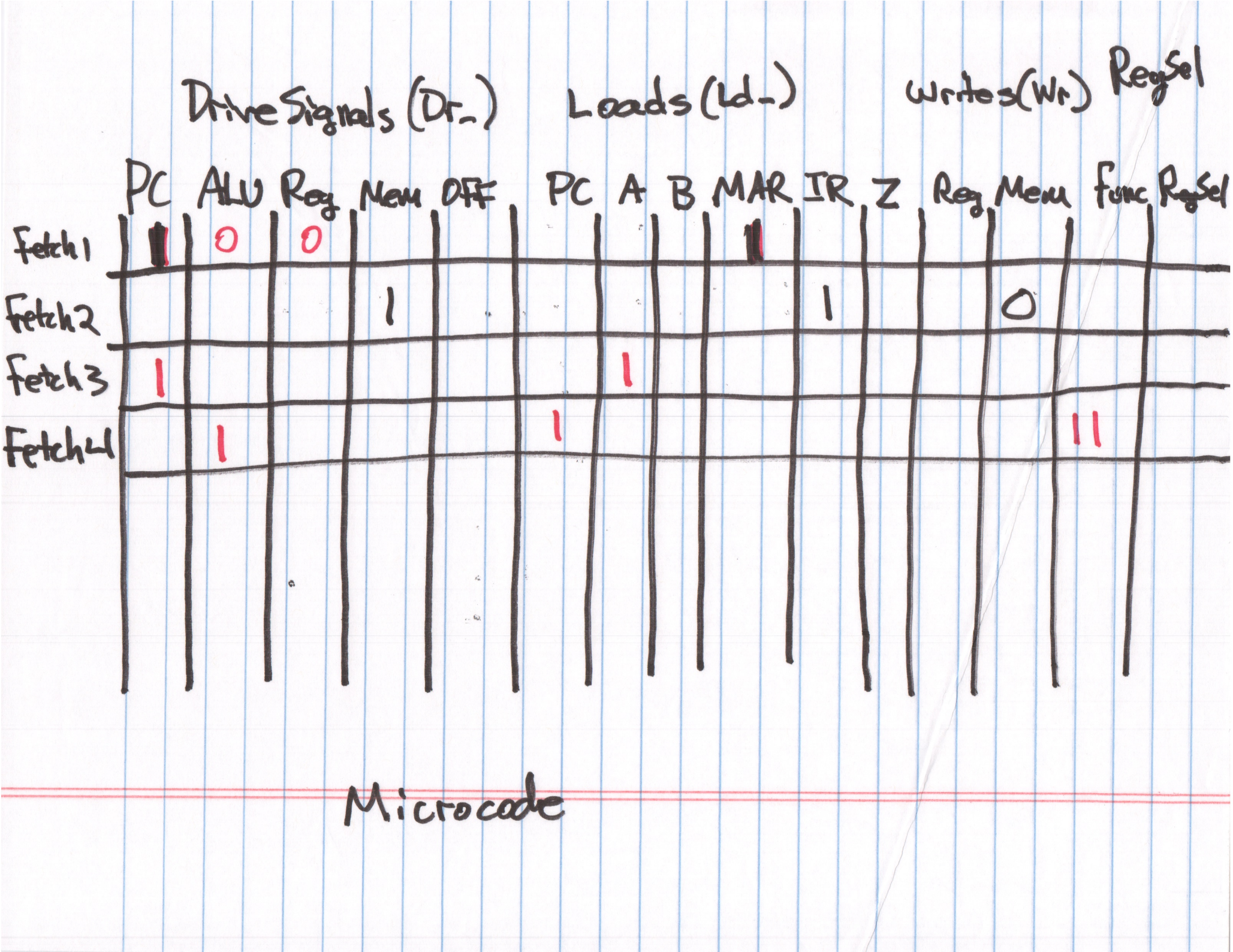
Tri-state ~~buffer~~

1 Din = Dout

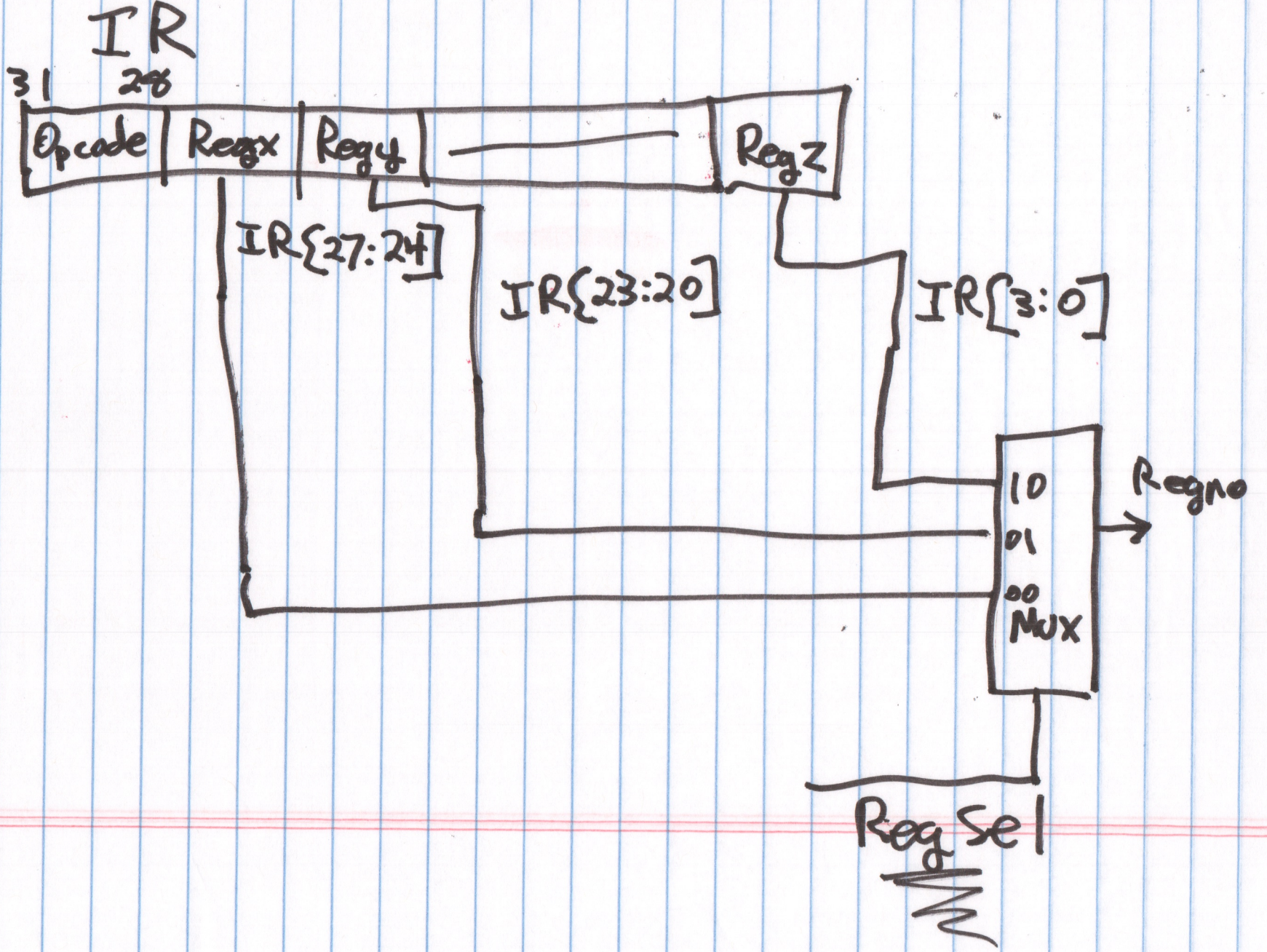
0 No connection (hi Z)


32
LdPC
PC
LdA
A
B
LdB
LdMAR
MAR
LdIR
IR
D:n
Regs
16x32
Dout
WrReg
RegA0
Func
WrMem
Addr
Din
Mem
Dout
32
IR[19:0]
+20
Sign
ext
+32
DrPC
DrALU
DrReg
DrMem
DrOff
=0?
+1
LdZ
Z

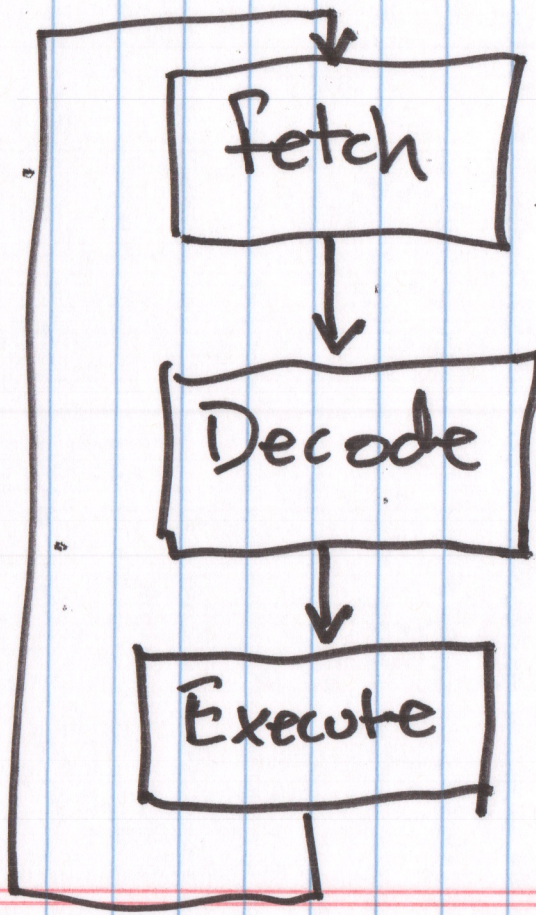

5
Dr PC, Dr ALU, Dr Reg . . .
Ld PC, LDA, LdB, . . .
Func
regNo
Address
ROM
State Reg


Drive Signals (Dr_)
Loads (Ld_)
Writes (Wr_)
RegSel
PC
ALU
Reg
Mem
OFF
PC
A
B
MAR
IR
Z
Reg
Mem
Func
RegSel
Fetch1
Fetch2
Fetch3
Fetch4
0
0
1
1
1
1
1
1
0
1
1

Microcode


IR
31
28
Opcode
RegX
RegY
RegZ
IR[27:24]
IR[23:20]
IR[3:0]
10
01
00
MUX
Regno
RegSel

State Machine



Fetch
Macro state
(set of states)

Decode
Macro state

Execute
Macro State

Fetch macro state

Fetch 1 PC → MAR

DrPC = 1, LdMAR = 1

Fetch 2 Mem[MAR] → IR

DrPC = 0 LdMAR = 0
wrMEM = 0
DrMem = 1
LdIR = 1

Fetch 3 PC → A

DrPC = 1
LdA = 1

Fetch 4 A+1 → PC

Func = 11
DrALU
Ld PC

Func	
00	Add
01	Nand
10	A - B
11	A+1