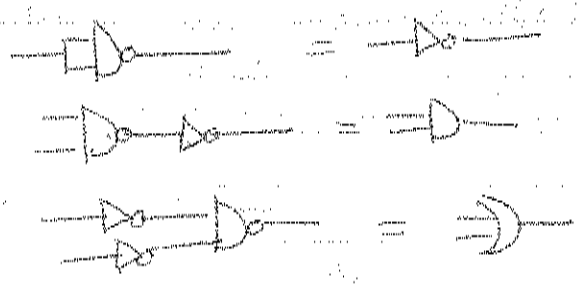
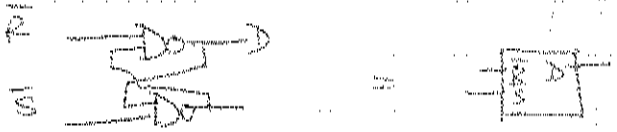
CS Hardware Review

Sunday 10
Thursday

Computer from NAND gates

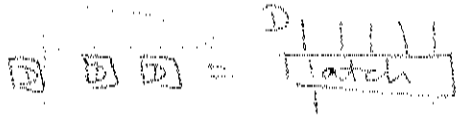
Build

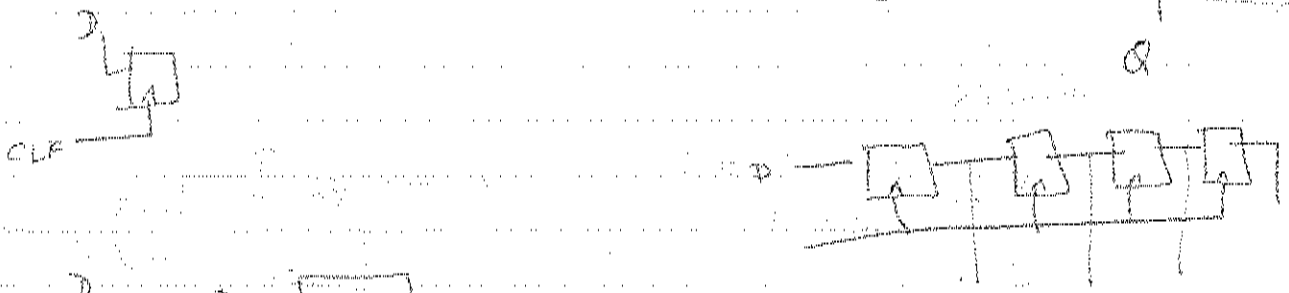


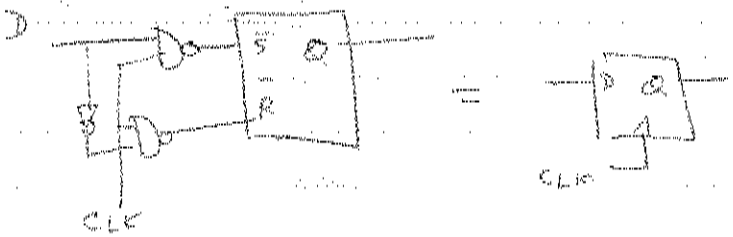


↳ make DFF

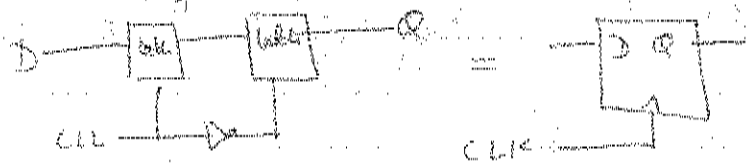
↳ make latch

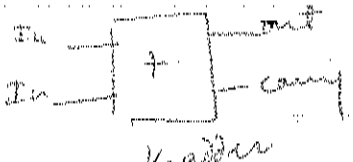




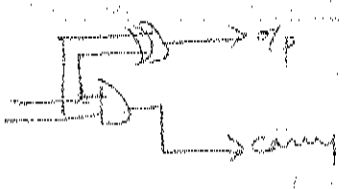


↳ make M.S. (when writing edge triggering)

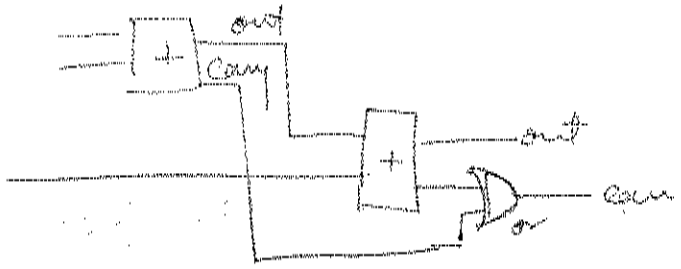




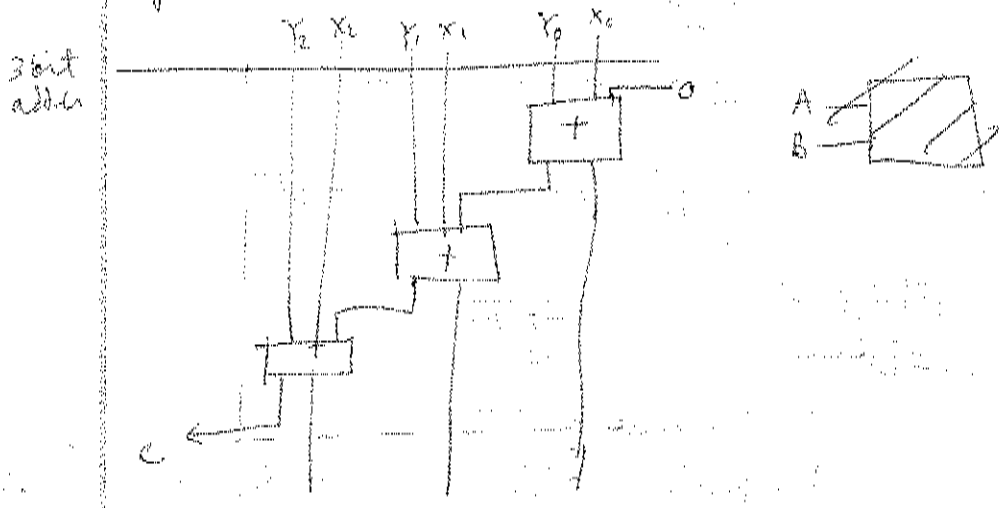
½ adder



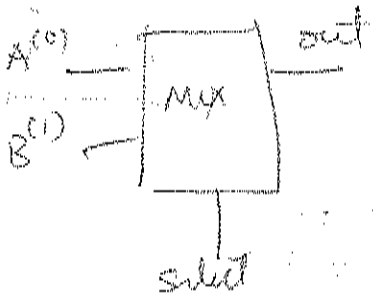
Full Adder



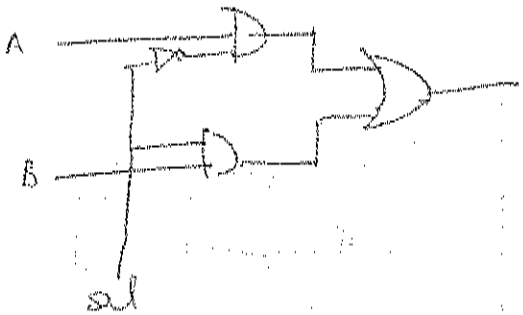
n full adders make the n bit adder



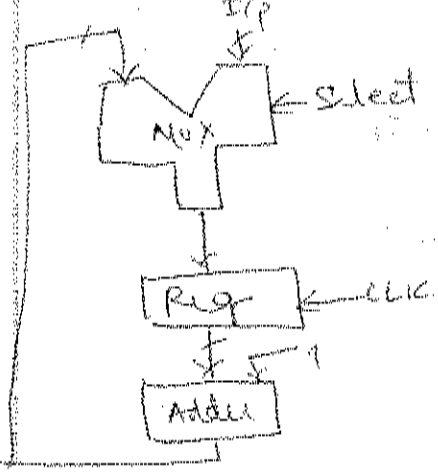
MUXES

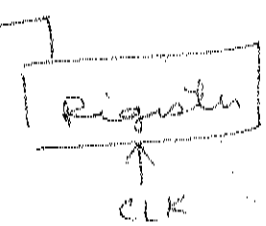


=



counter

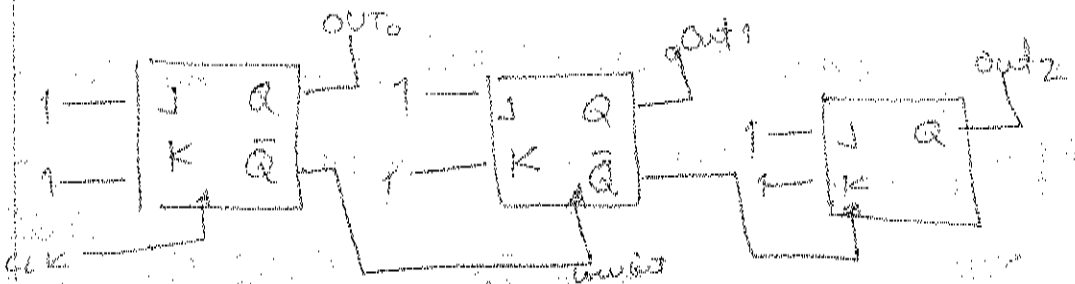




select = 0 : load

select = 1 : increment

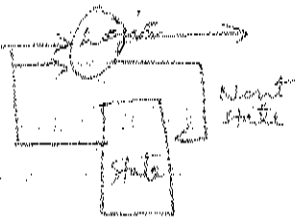
3 bit IC Ripple counter



toggles when 1→0

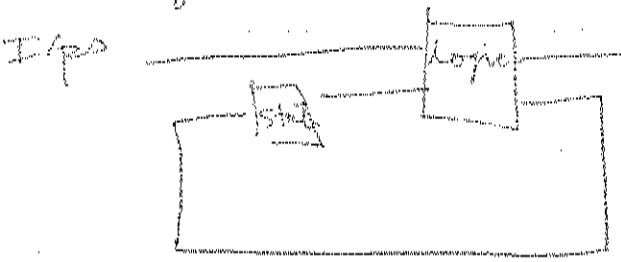
toggles on each clock

0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1



State M/cs

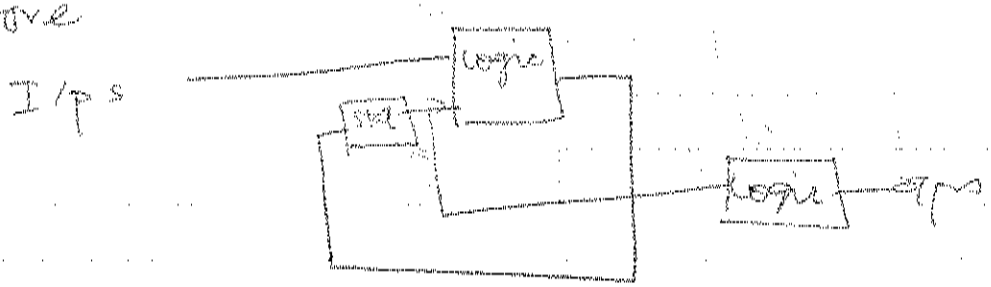
Mealy



o/p = f(i/p, state)

o/p on edges

Moore



o/p = f(STATE)

o/p in states

Moore is synchronous

Grey code : only 1 bit changes each step

Reflected grey code (Mirror method

000	000
001	001
010	011
011	010
111	110
101	111
100	101
110	100

Q_2	Q_1	Q_0
0	0	0
0	0	1
0	1	0
1	0	0
1	0	1
1	1	0
1	1	1

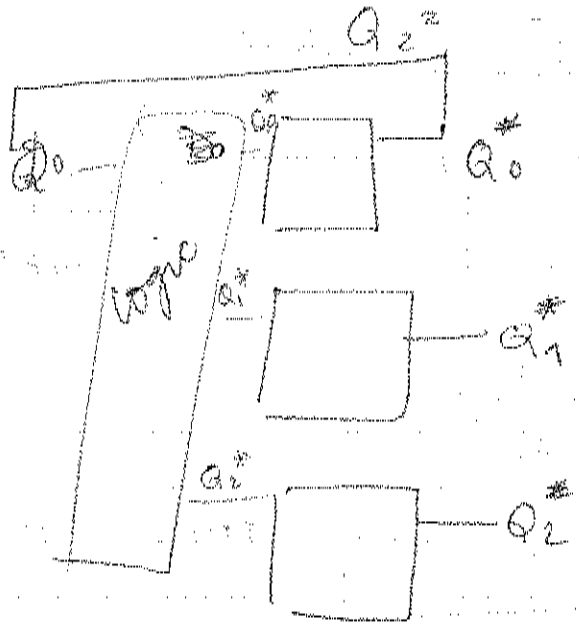
Q_2	Q_1	Q_0
0	0	0
0	1	1
1	1	0
1	1	1
1	0	1
1	0	0
0	0	0
0	1	0

o/p: $Q_2^* Q_1^* Q_0^*$

D_1D_0 / D_2	01	11	10
0	1	0	0
1	0	1	1

$Q_0^* = \bar{D}_2 \bar{D}_1 + D_2 D_1$

$Q_1^* =$



J	K	Q	Q'
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

→ VALS →

Qnow	Qnext	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

back to Grey counter:

Q_2	Q_1	Q_0	Q_0'	J	K
0	0	0	1	1	x
0	0	1	1	x	0
0	1	0	0	0	x
0	1	1	0	x	1
1	0	0	0	0	x
1	0	1	0	x	1
1	1	0	1	1	x
1	1	1	1	x	0

⇒ make kmap

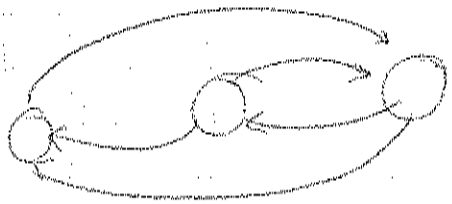
Q_1Q_0 / Q_2

Q_2 \ Q_1Q_0	00	01	11	10
0	x	0	1	x
1	x	1	0	x

$K_0 = \bar{Q}_2 Q_1 + Q_2 \bar{Q}_1$

State Assignment: counters ⇒ trivial

But



assign states to make O/p logic easy

make 0...0 initial state.

two adjacent states should differ by as little as possible

Given 5 i/p 3 F ☆ 1000 o/p

Mealy M/c

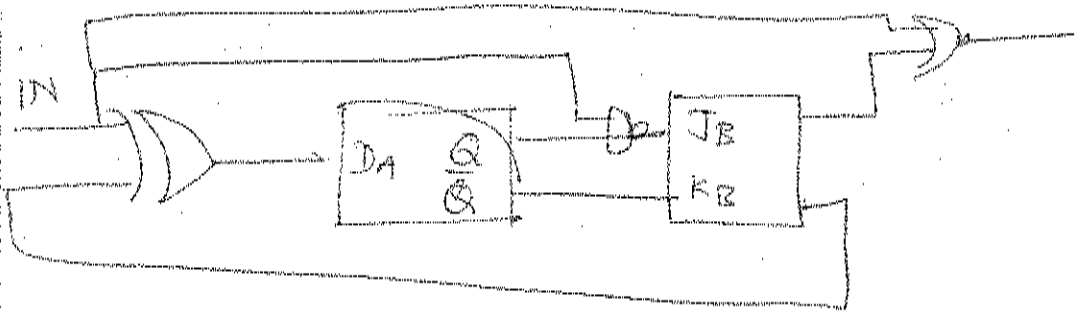
how many states? 1 to $2^5.2^3$
how many distinct o/p patterns configuration
how many bits from each state
how many bits to each state 0 to $2^5.2^3$

→ straight forward

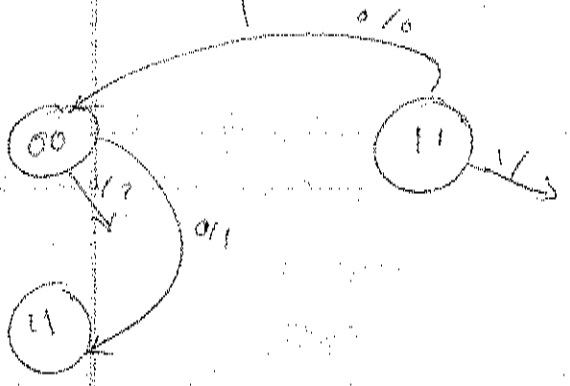
Mealy - each edge has distinct o/p pattern

more 2^3 o/p configurations

Remember to have a start state



Mealy

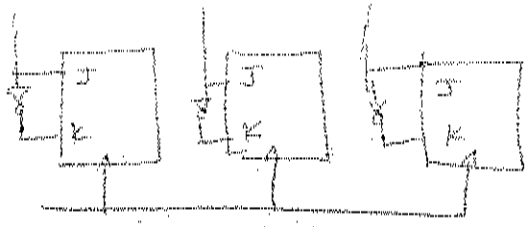


i/p	Q_A	Q_B	D_A	J_B	K_B	Out	Q_A^+
0	0	0	1	1	1	1	1
0	0	1	0	1	1	0	0
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

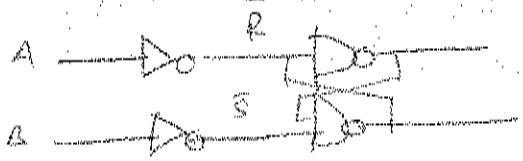
start

Ccf identification

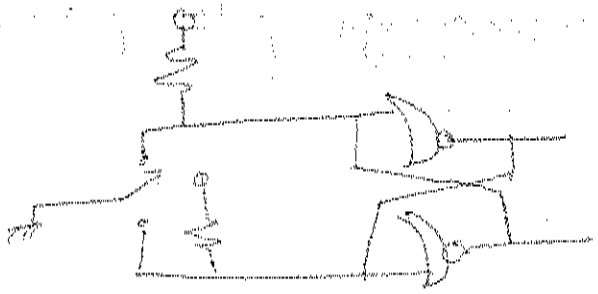
① XNOR

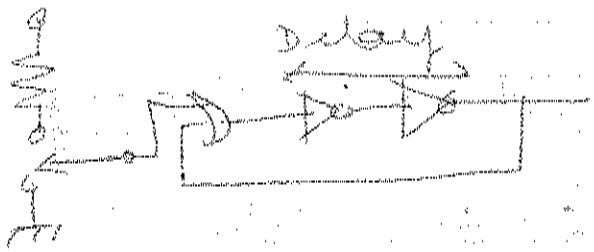
② 

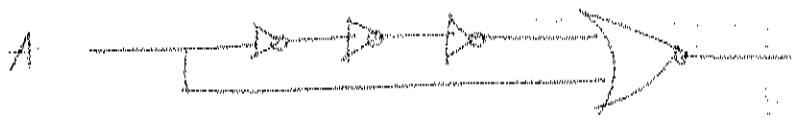
3Bit latch

③ 

= RS latch

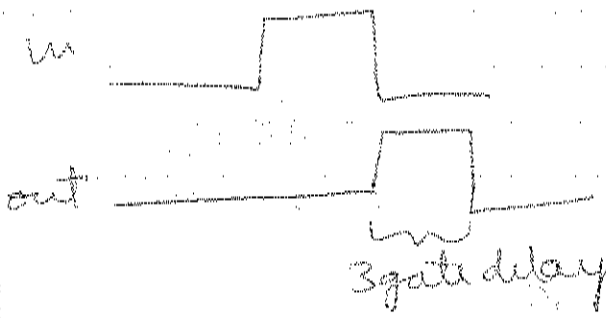
④ 
Debounced Switch

⑤ 

⑥ 

0 in ⇒ 0 out

1 in ⇒ 0 out, then goes to 1 ε 3 gate delays later

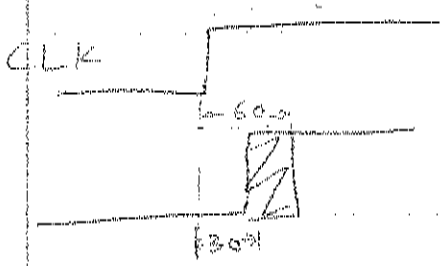


*16

Timing Analyzes

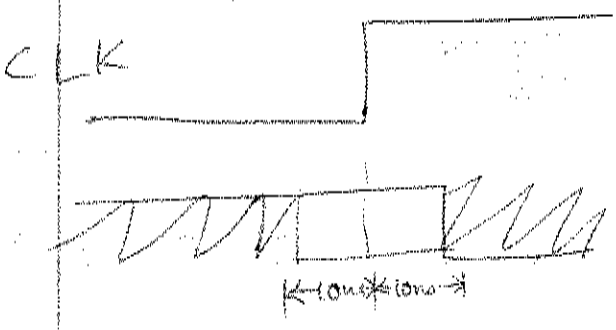
Flip Flops [max delay 60 ns
min delay 30 ns
setuptime 10 ns
holdtime 10 ns

→ from i/p CLK changing to o/p changing

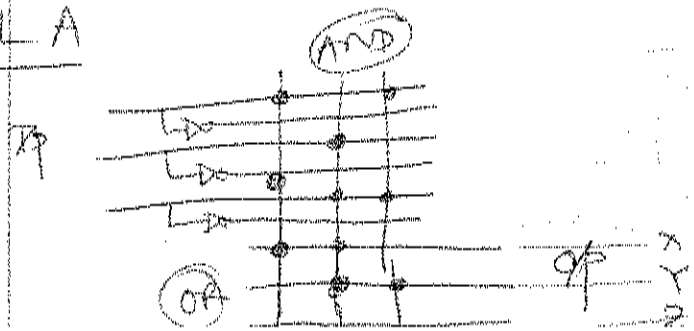


→ time i/p s must be unchanging before CL

→ time unchanging after CLK



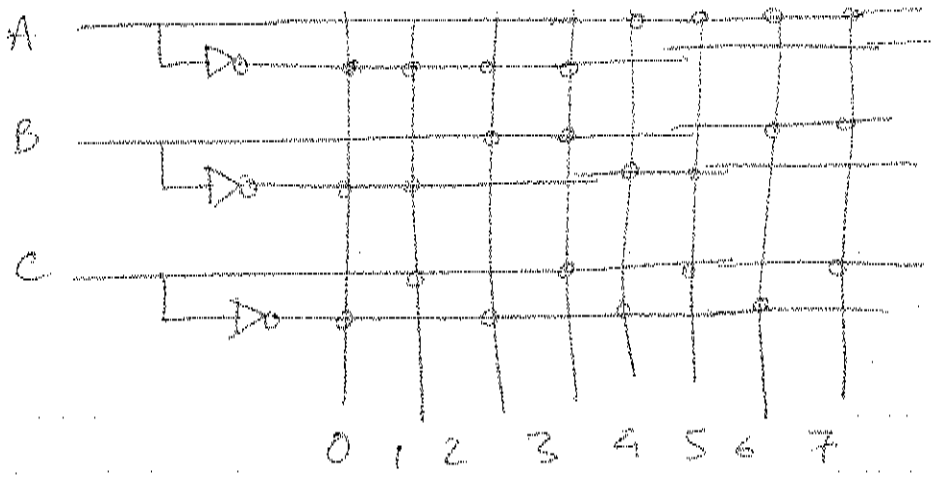
PLA

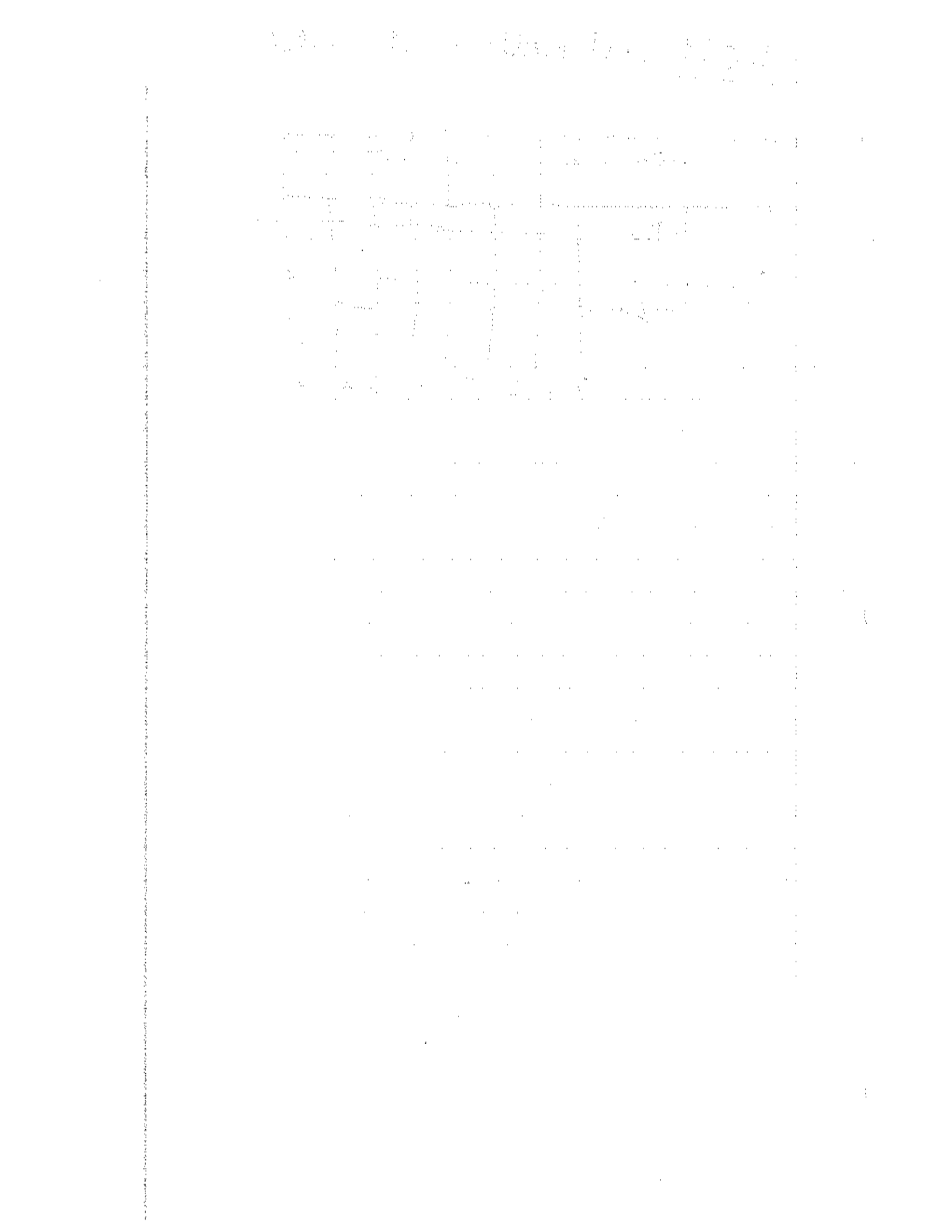


$X = A\bar{B} + BC$

$Y = AC + BC$

ROM all pattern decoded

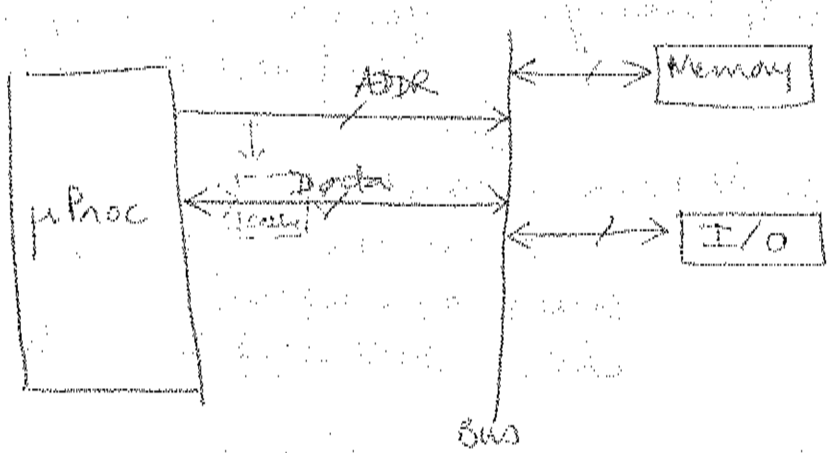




Hardware Prelim

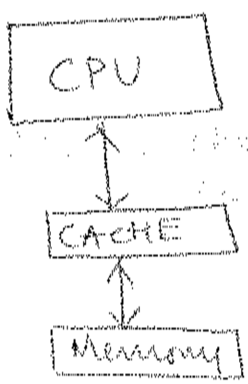
6 Sept '90

Caches
Virtual Memory
I/O
Instruction Set Design
Processor Implementation



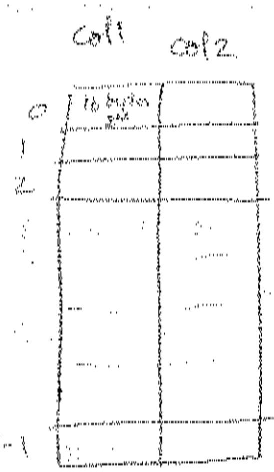
DRAM ~ 100nsec

Temporal locality
Spatial locality



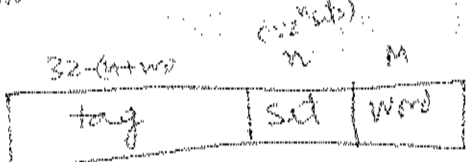
two way set associative →

direct mapped is the same as 1 way set

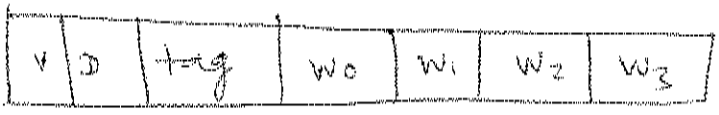


1024 bytes
⇒ 32 lines

16 bytes = 4 32 bit words!



Cam Address.
don't worry if virtual/physical



{R1} {R2} ...

1K ⇒ 1K of data (not incl. the tag & dirty bits)

V → valid bit (eg switching processes, virtual memory) (not valid)

D → dirty bit

write through → goes to memory (memory is updated)
(don't need the extra bit)
faster on a cache miss

write back (= copy back)
cache doesn't write to memory
slower on misses!
adv. don't need to write each time

How do you decide which to throw away?
(1) Random (use counter)
(2) R ← replace dep on R

Random – for HW caches almost as good (if many # of bits)
LRU – don't want to do each access

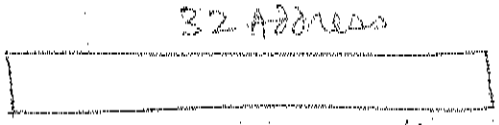
no D bit in write through cache
D bit direct mapped ≠ write back ⇒ yes
↳ true of other kind no.

write through → maybe yes / no
direct mapped ⇒ no

p. 25

valid bit always always
dirty bit never sometimes
replacement bit sometimes never

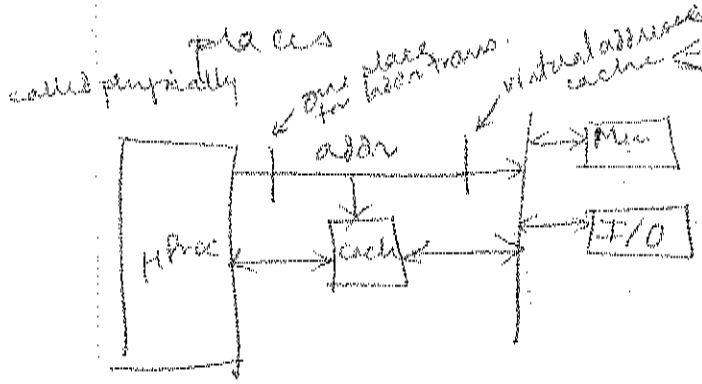
Virtual Memory: (= addr translation)



bad
(1) slows the translation
(2) need memory to do addr translation

good
(1) more Memory
(2) multiple processes can start at zero
(3) memory protection

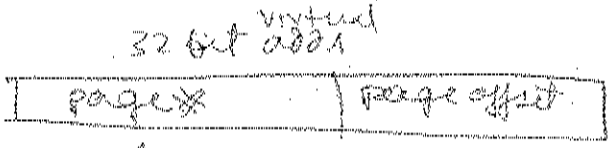
virtual Addr. translation can occur in one of two places
(virtually addressed)



problem: don't want to have shared virtual addresses

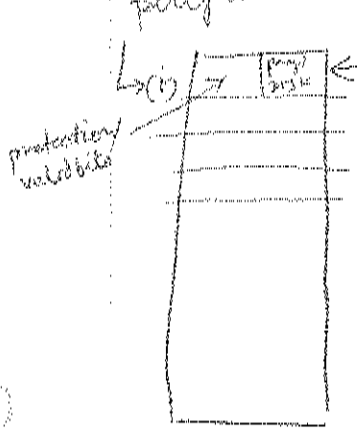
Pro: don't need to do the Addr translation for cache

can't do virtual mem w/o addr translation!



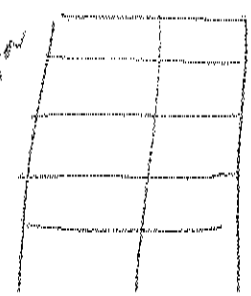
BSD 4.3
4Kbyte page size

translate from virtual to physical



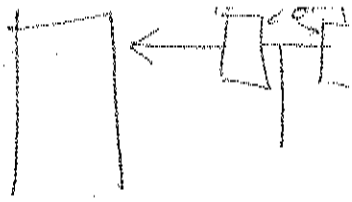
page table
(could get large)

prob: go to memory anytime to get access
: TLB table lookaside buffer



Recently a page table entry

← read only cache changes only when page table



page fault:

many processes each has own page table

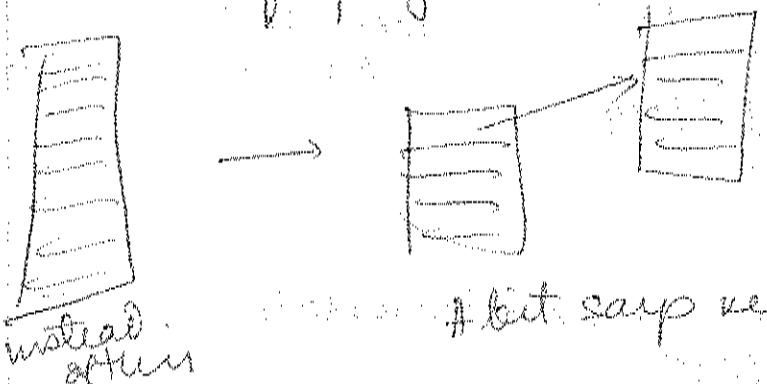
now no program ever needs entire page table

VAX: page the page table itself
- lives in virtual memory

Now you need two memory accesses

page table hangs out in system space
⇒ protection

Tree of pages



bit says keep going or stop!

segment: variable length pages

I/O: now memory mapped

eg. any address in ffff8000 - ffff8010 will go to Disk Drive

↑ could be virtual addresses

how do you tell the device is ready

(i) poll: check each & place anywhere and then status bit

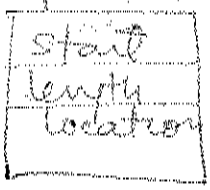
(ii) interrupts

3 ways to actually comm^n

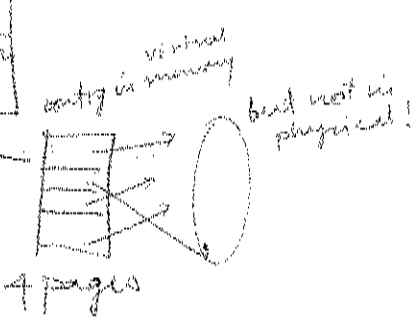
(i) Programmed I/O CPU actually goes out - for slow devices

(ii) DMA disks, n/w, high speed stuff

CPU builds table

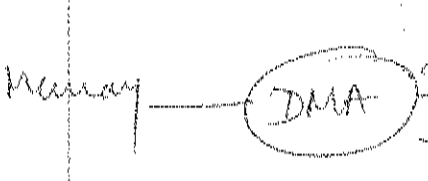


adv of virtual mem address for table



(iii) data channel ~~bunch of processors that are I/O processors~~

bunch of I/O devices hang off a single DMA processor



(i) do one fully (ii) round robin (bit of each)

ISA

Wish to design an instruction set

32

Word/Address

How many addresses in each m/c instr

0
1
2
3

→ ADD ____, ____, ____ 2 sources & 1 destination

mathew ⇒ 3 instr

ADD ____, ____, blah
⇒ 2

ADD ____ ____
implicitly using accumulator ⇒ 1

0 ⇒ stack based

Addressing Modes : out these days
∴ it's a subset of instr
(gained by shift & add)

Now ⇒ Load Store

Register file Size ⇒ 32 is a popular ✳

Floating Pt vs Int registers
Addrs vs Data registers

Before : wanted to encode instr densely (∵ mem expensive)
now not a real issue
so don't need separate Addr & Data reg

Most instr:

- MOV
- ADD
- SUB
- MUL
- DIV/REM
- AND
- OR
- XOR

→ [ADD | SUB | MUL | DIV]

branch / jump
brc → branch on cond
scc → set reg on cond

VAX — # of operands (could specify)

[op1] [R1] [R2] [R3]

1 byte opcode ← some specify indirect

VERY LITTLE space, but slow decode

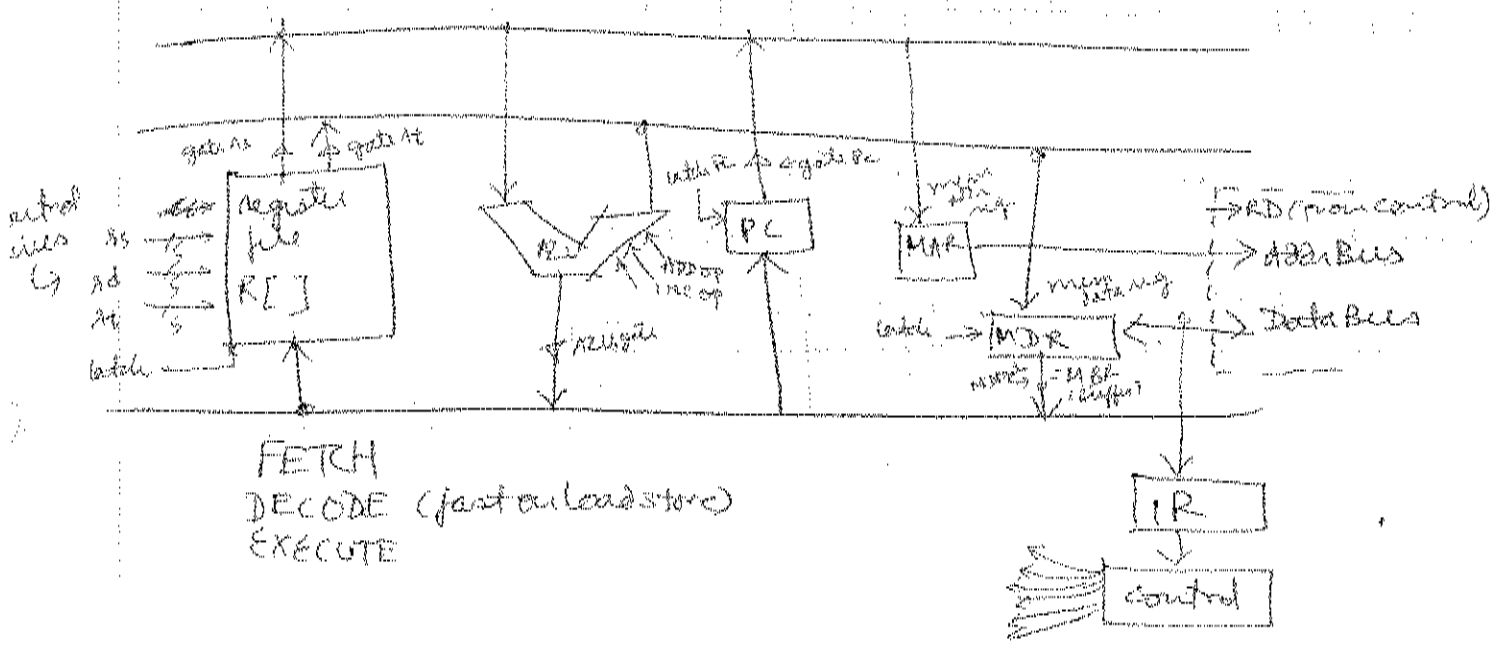
MIPS — destination

[ADD | rd | rs | rt | FL | unused]

∴ every instr is 32 bits wide!
VERY Fast

← one of 4 instr formats

IMPLEMENTATION



31 26 0
ADD R_S R_T R_D ...

Spring '00 — Katz & Culler
↓ I/O project

for MIPS

RTL: ① MAR ← PC
PC ← PC+1
assert RD
] fetch (in early μPs had two phase clks so could do both in 1 clock cycle)

② IR ← databus

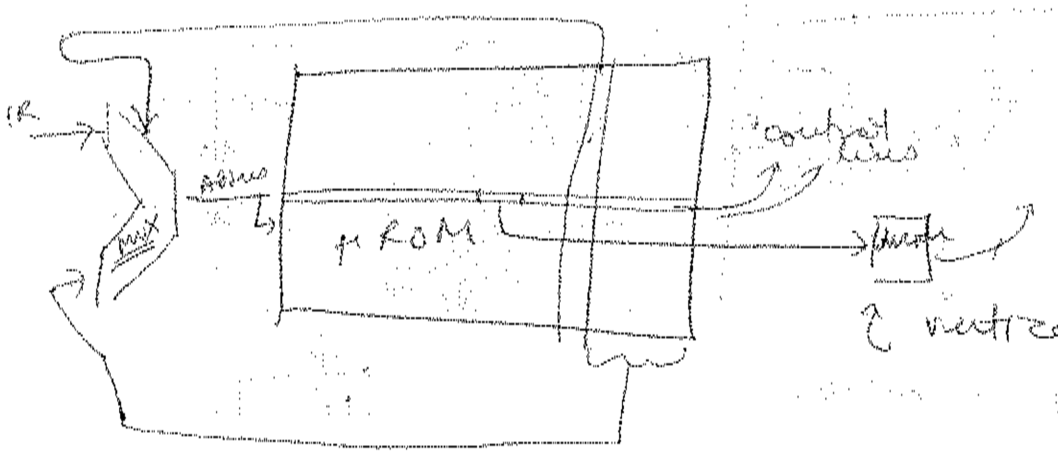
can't merge {

decode [③ branch (IR$_{31..26}$) ← basically a procedure call

execute [④ ⋮
R[rd] ← R[rs] + R[rt]
branch (1)

USE LOTS OF REGISTERS TO LATCH EVERYTHING!

RTL	gate PC	gate RS	gate RT	latch RD	latch MAR	op PC	op ADD	RD	latch PC	latch IR	next ADDR from IR	next ADDR from DR	next μinst
① MAR ← PC, PC ← PC+1, assert RD	1				1	1		1	1				000010
② ~~branch~~ IR ← databus										1			000011
③ branch (IR$_{31..26}$) ⋮												1	~
④ R[rd] ← R[rs] + R[rt], branch 1		1	1	1			1						000001

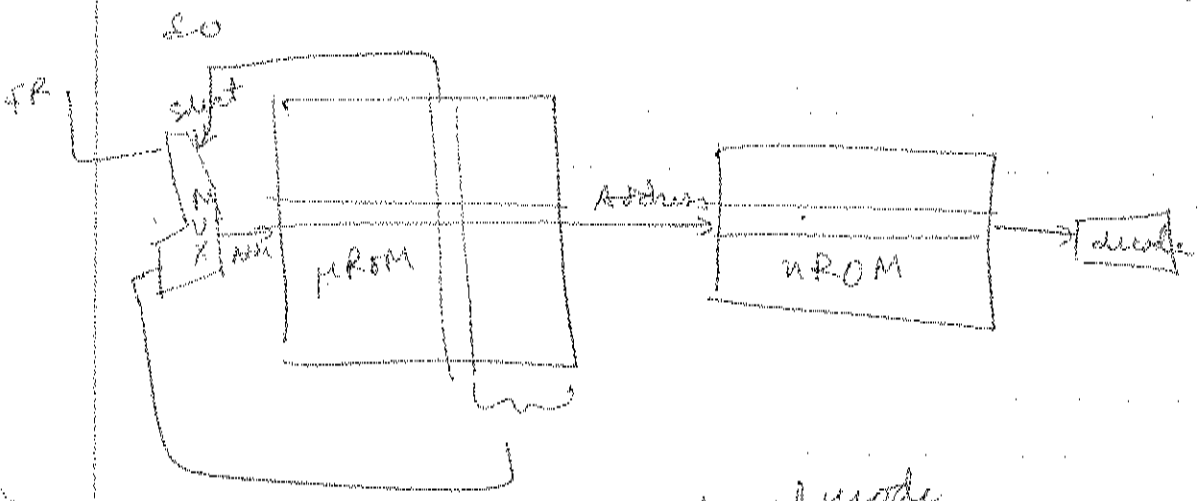


? vertical μ coding

could wasteful, lots of zeroes

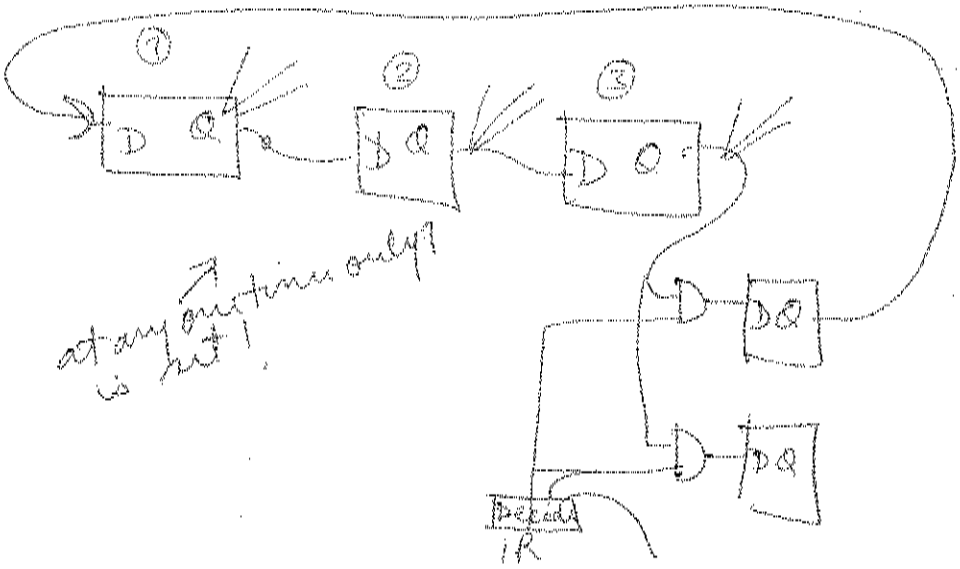
Makes sense to collapse ALU ops down → decoder entry

↳ can narrow further
have a separate line of opcode for encoded AND, +, OR etc
even through ~~anything~~ everything same



2 word inside

hardwired control



for same above each