m Page No.

Outline

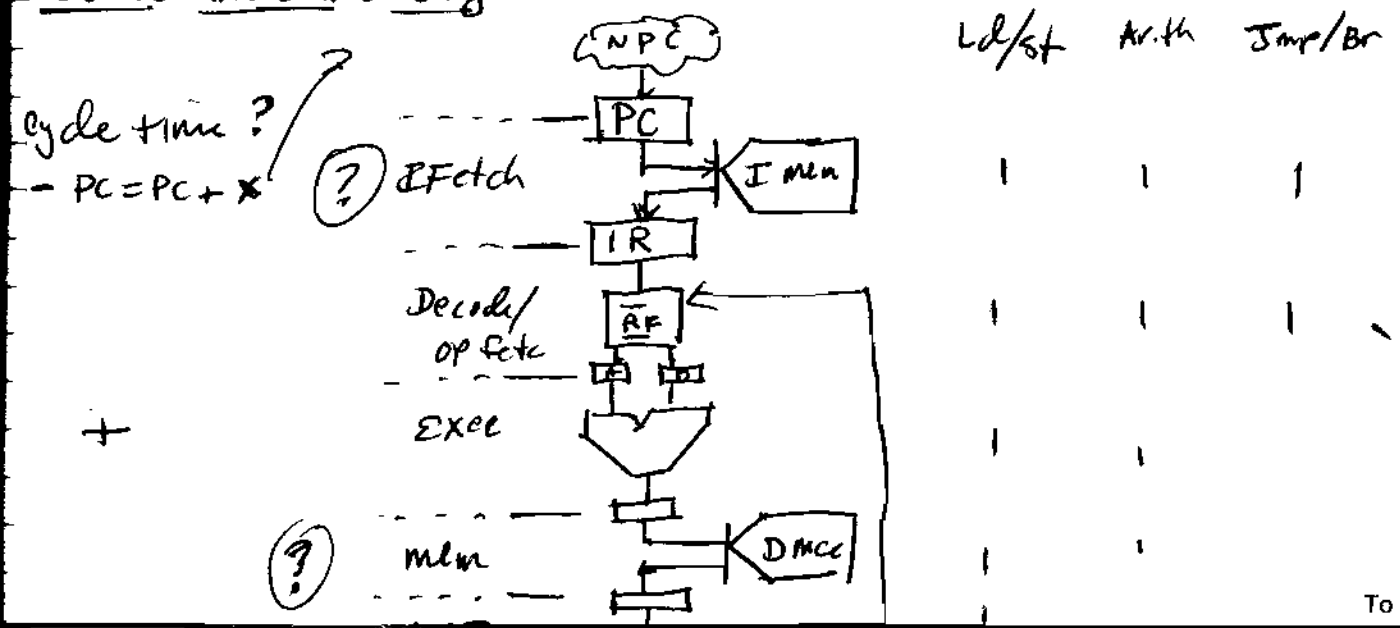
- Project 4 demo / Q&A
- CPI – Performance
- Caches

Project 4

- separation in subsystems
  - Data path
  - Sequencer
  - Memory
  - Control & decoding
- Techniques
  - hiding vs showing details
  - ~~symbolic~~ decode implementation
    - gates
    - decoders
    - muxes

Ideal

~~Basic~~ Machine Org



Cycles Per-Inst

	Ld/st	Ar.th	Jmp/Br
	1	1	1
	1	1	1
	1	1	
	1		
	1		

To Page No. 1

From Page No.

Performance

technology Arch Appl

Time for Program (s) = Cycle Time × CPI × # instructions

seconds/program = seconds/cycle × cycles/Inst × instructions/Prog

Two ways to determine CPI

Top down:
- measure ~~prog~~ time
- simulate to count # inst. (or counters)
- look up cycle time

Bottom up: - simulate to determine inst Mix

—	50 %	Arith	× 1	= 2.0
—	30 %	Ld/s	× 5	1.5
—	20 %	Jmp Br	× 2	.4
				3.9

$$\text{Ave CPI} = \sum f_i \cdot C_i$$

Memory

Big Memories are Slow.

Read Address ⇒ data most recently written to that address

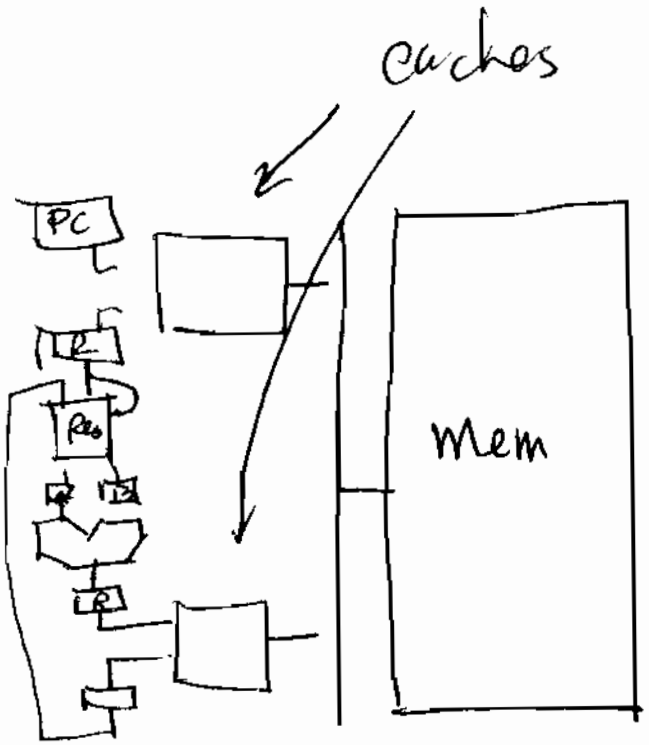
Write Address, Data ⇒ writes data to address

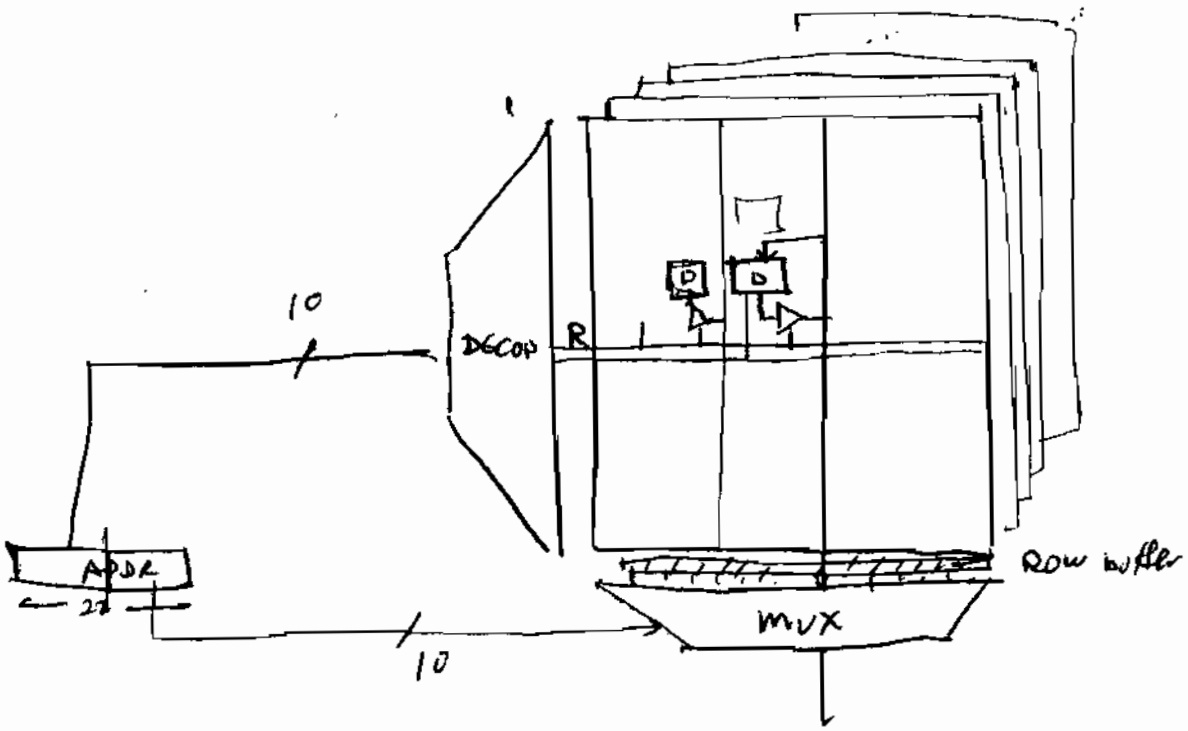
Program: series of Read(Addr), write(addr, data)

To Page No. 2

Witnessed & Understood by me.	Date	Invented by	Date
		Recorded by	

From Page No.





Read
- select row
- Drive data lines
- select col

Write
- Read to RB
- update bit
- drive data
- select row to write

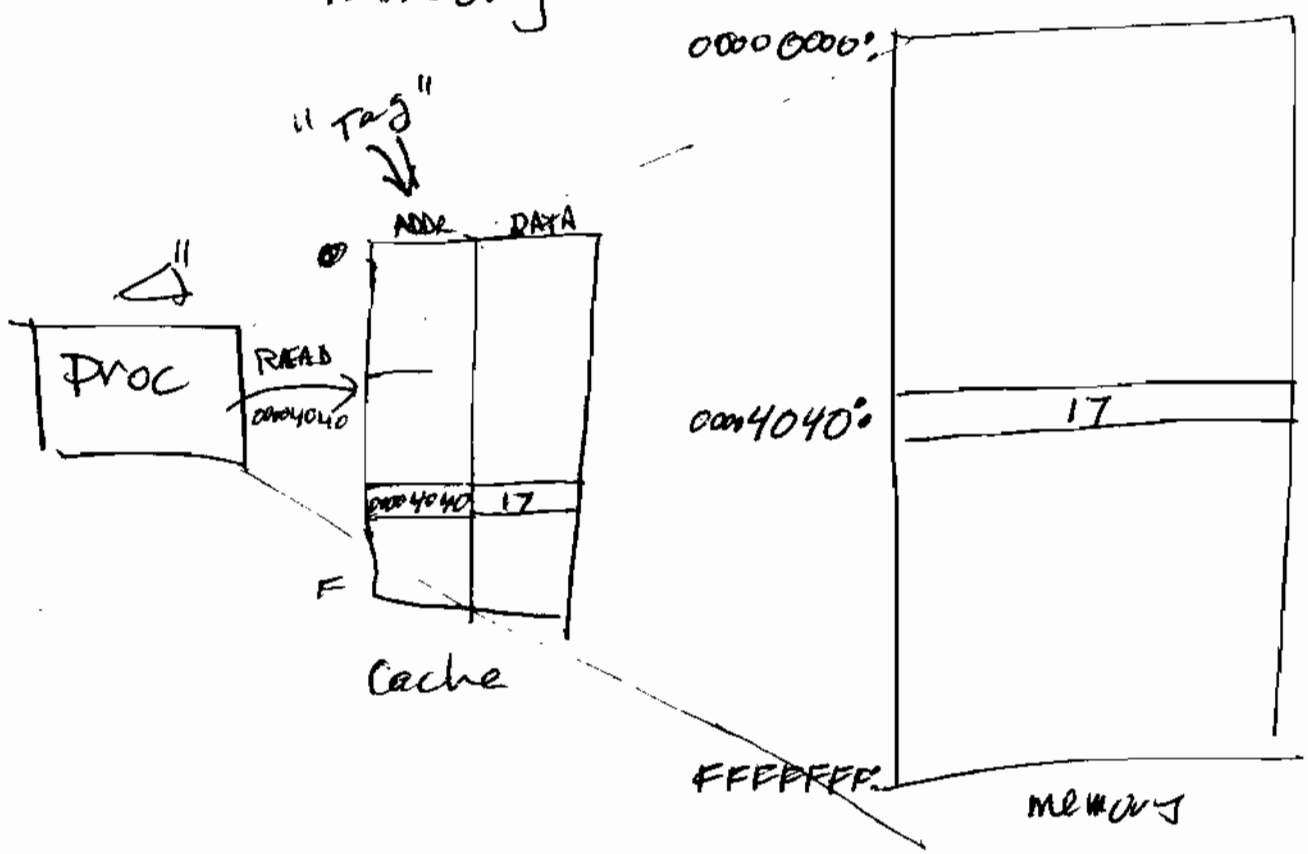
- static RAM: 6 Ts, Faster, Stable
- dynamic RAM: 1T, Slow, dissipates, refresh

To Page No. 3

Witnessed & Understood by me,	Date	Invented by	Date
		Recorded by	

From Page No.

Caches: small fast memories that hold the "most important" words of memory



holds address/value pairs

stores value @ address

Hit: Proc makes access to address/value "block" that is present in cache
- read - return value
- write - update value (mem?)

Miss: Not present in cache

		ideal		mem		miss cache		
Arit	50%	4	2.0	5	2.0	4	2.0	
L/S	30%	5	1.5	104	31.2	5% 10	3.0	
B/J	20%	2	.4	2	.4	2	2.0	
			3.9	100	100	1% 1	1	To Page No.

Witnessed & Understood by me, Date

Invented by 1 3510 8.0 Date

Recorded by

Cache Operation

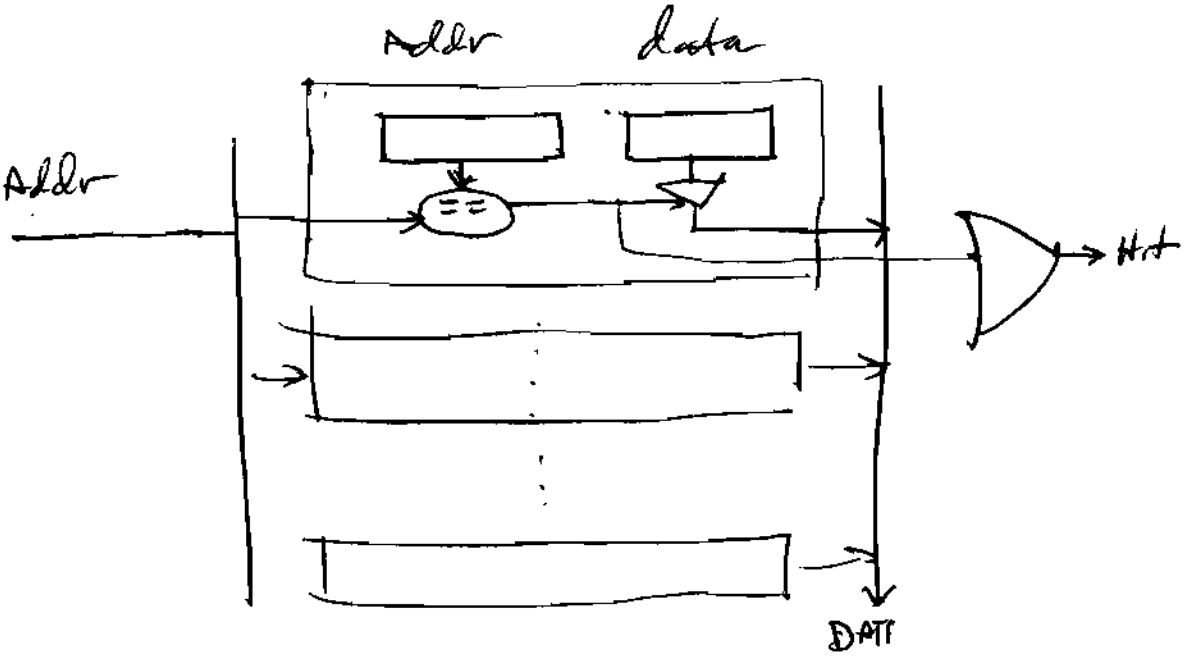
Read (addr)

~~lookup block~~

```
<present, block> = lookup(addr)
if(present) return ~~block.data~~ cache[block].data
else
    data = real mem[addr]
    block = place(addr)                    * what if
    cache[block] = <addr, data>
```

```
Write (addr, data)
    Mem[addr] = block                      — write thru
<present, block> = lookup(addr)
if (present)                               — "no allocate
    cache[block].data = data
~~else~~
    ~~Mem[addr] = block~~                  "~~write-thru~~
```

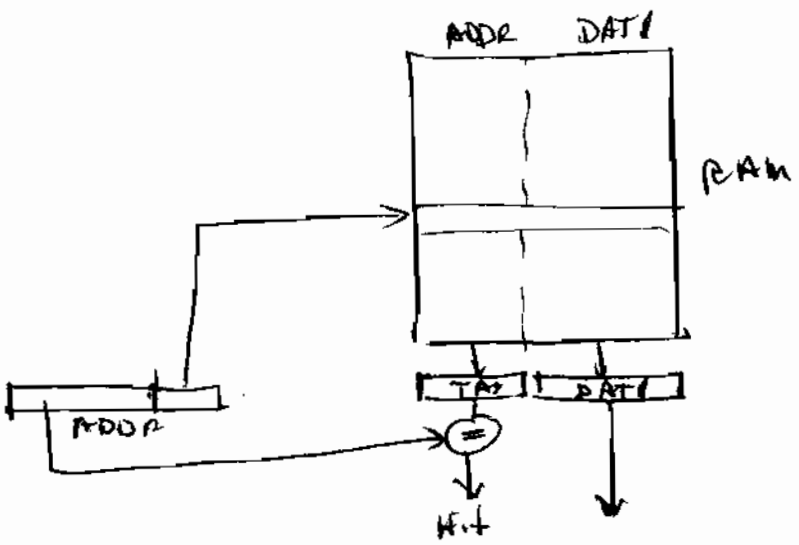
Fully Associative Cache



To Page No. 5

From Page No.

Direct Map Cache



Cache Design Parameters

- # blocks
- block size
- ~~Asso~~ associativity
- write policy
- replacement policy

Hit: 1 cycle

Miss: ~~1~~ + miss penalty (say 100 cycles)

Miss Rate: $\frac{\#\text{ misses}}{\text{total mem refs}} = \frac{\text{Misses}}{\text{Hits + Misses}}$

Wait on miss

CPI = ideal CPI + Ave Wait

" + Miss Rate · Miss Penalty

ex 2% @ 100 cycles ⇒ 2 CPI

3.4 ⇒ 5.9

Table on Page 4

To Page No. 6

Witnessed & Understood by me.	Date	Invented by	Date
		Recorded by	