Syllabus

May 3, 2022 9:01 PM

Endsem planning

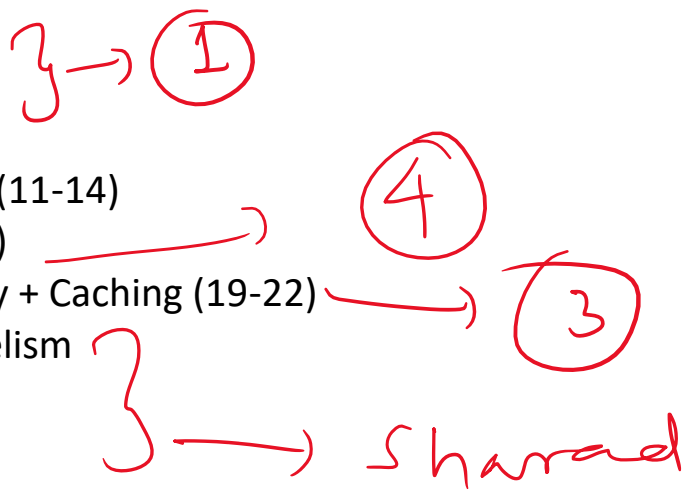
Sharad + Gatu smashers

Computer Architechture

CA Basics (1-6)
MIPS ISA (7-10)

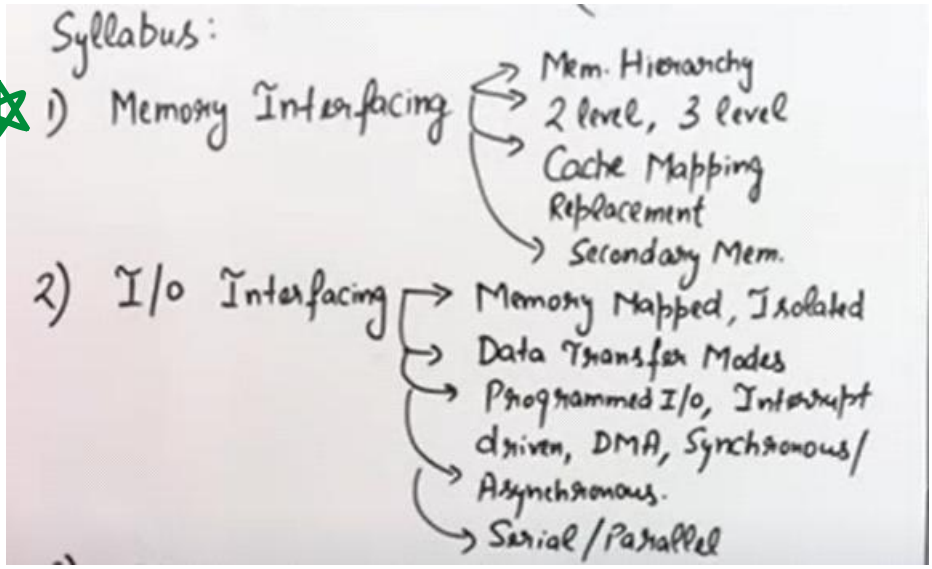
Datapath design (11-14)
Pipelining (15-19)
Memory hierachy + Caching (19-22)
Data level parallelism

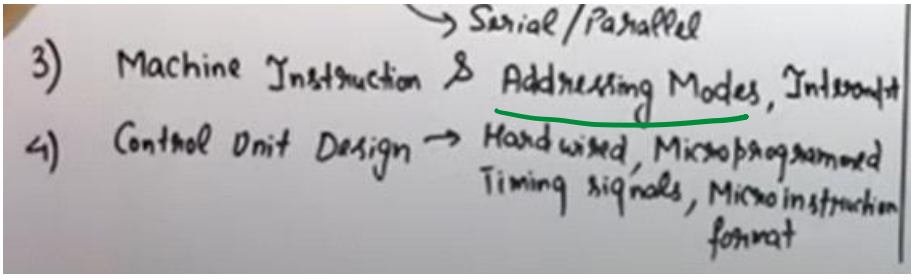
Advanced CA

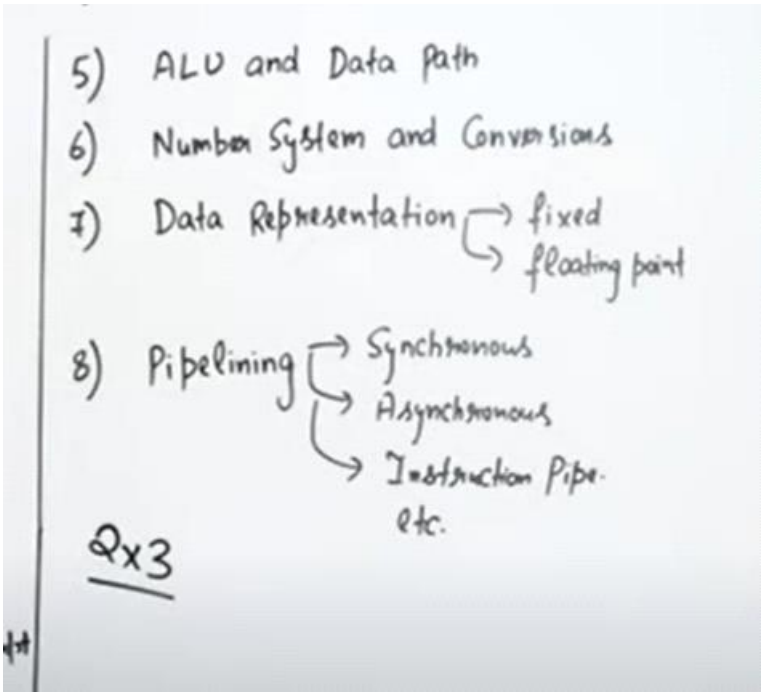


TODO

May 3, 2022 11:20 PM







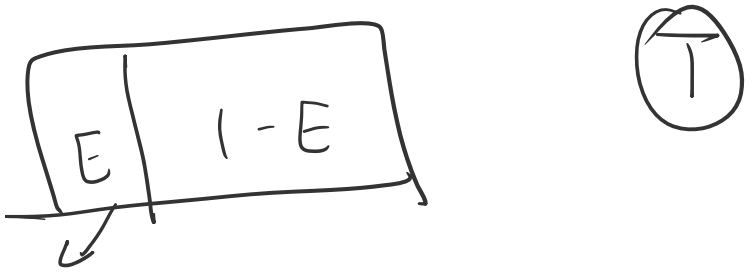
Basics

March 10, 2022 2:37 PM

- Computer architecture
- Moore Law
  x2 transistors, 2yrs
-
- Execution time
- $T = IC \times CPI \times T$

  Instruction count — cycles per instruction — clock period
- Datapath
- Speedup = $\frac{perf_A}{perf_B} = \frac{T_B}{T_A}$

  A / B
-



$$T_f \overset{S}{=} E\left(\frac{I}{S}\right) + (1-E) \times 1$$

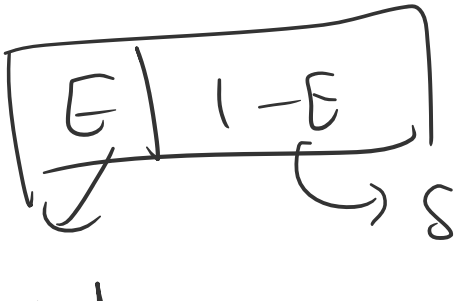
$T_f = t(\frac{E}{S} + (1-E))$

$= E/S + (1-E)$

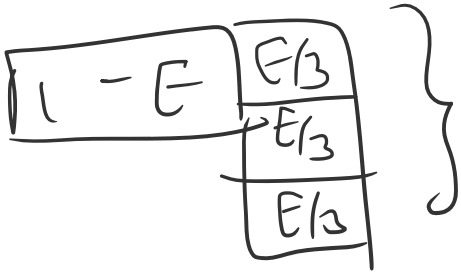
- $S \to \infty$ ∞ speedup

$T' = \frac{1}{(1-E)}$ → limited by unsped part

- Amdahl's Law (fixed problem size)



Enhanced

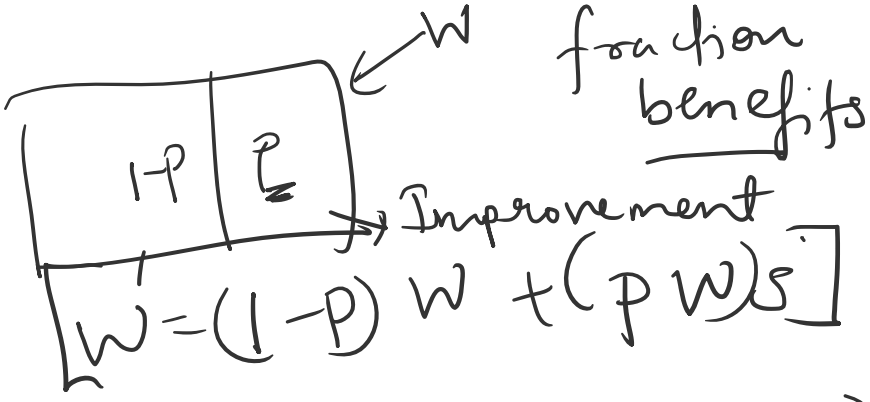
- 

Parallelization by 3

- Gustafson Law

W → workload, P ↑.

W → workload, P →



(same execution time)

Speedup = (1-P) + SP

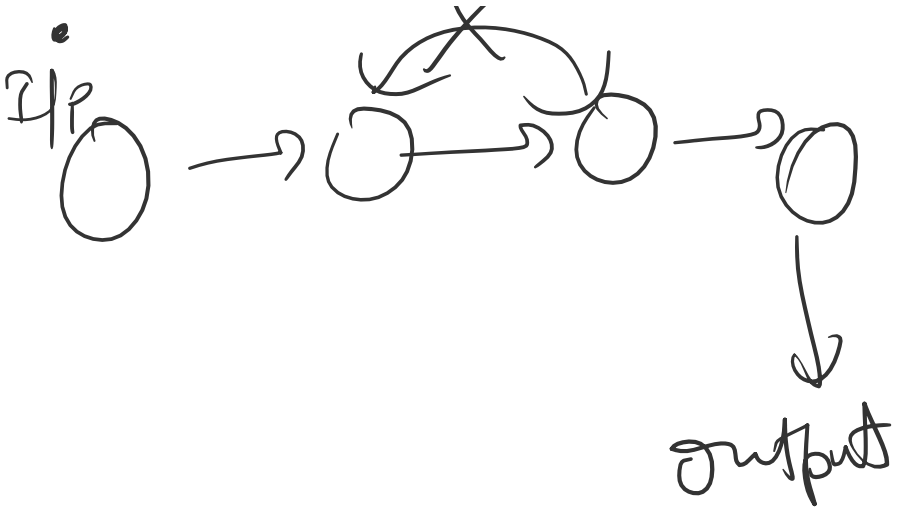
- MIPS

Microprocessor without Interlocked Pipelined Stages

From <https://en.wikipedia.org/wiki/MIPS_architecture>

Native Peak Relative

- other → FLOPS
- parallel processing
- pipelining
- Flo



- Derived concurrency

IF → Fetch

ID – Decode

EX – Execute

MEM – Mem op

WB – Write Back

5 steps

- Power

Dynamic (operations) → $P_{dyn} = AcV^2f$

Static (leakage current) → independent

- $P_{dyn} = ACV^2f$
  - independent of f
  - $\propto$ Temp
  - $P = VI_{leak}$
- $P_T = P_{Stat} + P_{dyn}$
  $= VI_{leak} + ACV^2f$

$f_{max} \propto [V - V_{th}]$

- $E = \int_0^T P_{avg}\, T = P_0 T$
- Reduction
  - Cache memory
  - Power gating → Switch off unused
  - Clock gating → less switching

switching
unused parts

- data movement

- $f = 500 \times 10^6$ Hz

$$\begin{cases} P_{dy} = 80W = A C V^2 f \\ P_{stat} = V I_{leak.} = 10kJ \end{cases}$$

$E = 540$ kJ. (T)

$$T = \frac{540}{90} = 6 \text{ sec}$$

$P'_{stat} = 0.9 P_{stat}$

$f \to f/2$

$T \to T \times 2$

$E = 49 \times$

$P = P_{dy} + P_s$

$= \frac{9P_{dy}}{10} + \frac{9P_s}{10}$

$= \frac{49}{10}$

$= 4.9$

$P'_f = 49$ kw

$P_f' = 49$ mW

- MIPS Theory

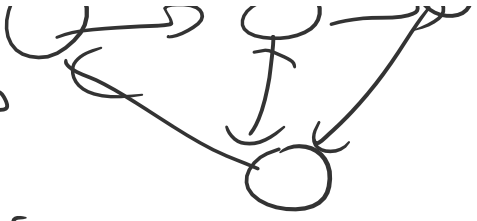
ISA → Instruction set Architecture

- Introduction to ISA
- MIPS ISA: A design example
  - Instruction format
    - R-format, I-format, and J-format
  - Addressing modes
    - Register addressing
    - Indirect addressing
    - Immediate addressing
    - Base addressing
    - PC-relative addressing
    - Pseudo-direct addressing
  - Functionality
    - ALU instructions
    - Data transfer instructions
    - Control instructions
- MIPS procedure call
- Design issues and relative advantages of RISC and CISC

CS 211 - Computer Architecture

- MIPS (Microprocessor without interlocked pipeline)
- RISC Archi (Reduced instruction set computer)
-

Interlocked pipelines



- 32 Register
- Program counter (PC) → next ins
- Accumulator (Acc)

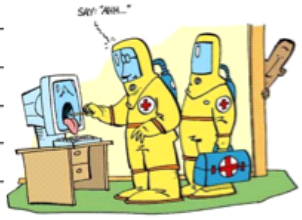
1 byte = 8 bits
1 word = 4 bytes
= 32 bits

- Each register has 32-bit (1 word)- 4 Bytes (1 byte= 8 bits)

Name	Register number	Usage
$zero	0	the constant value 0
$at	1	reserved for assembler
$v0-$v1	2-3	return values of funtions and expression eva
$a0-$a3	4-7	function arguments/parameters
$t0-$t7	8-15	temporaries, caller saved register
$s0-$s7	16-23	saved values, calle saved registers
$t8-$t9	24-25	more temporaries
$k0-$k1	26-27	Kernel use : reserved for OS
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

Temporaries are not preserved across procedure calls, while saved values are preserved across procedure calls.

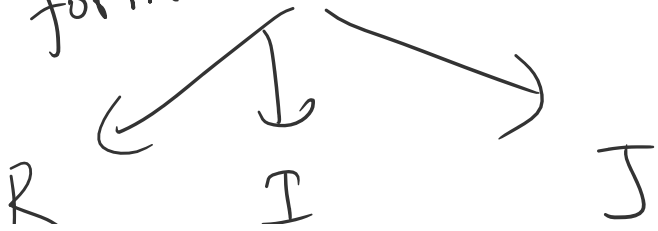
If you don't follow the convention, there will be troubles.



CS 211 - Computer Architecture

1

- word → 4 bytes
- MIPS instruction
- format

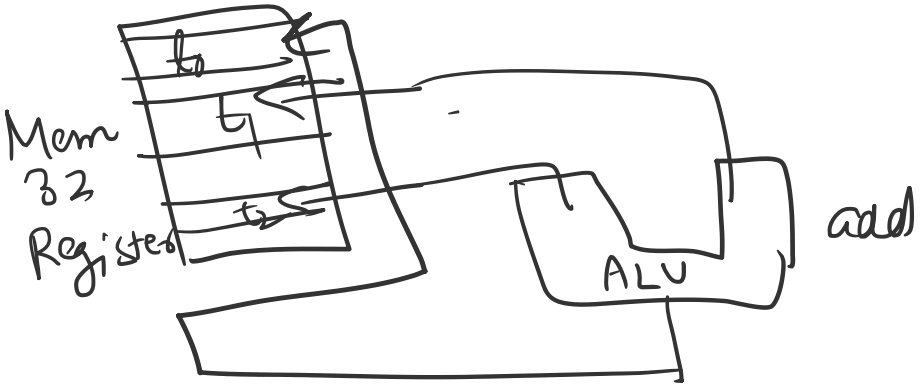


R I J

Register Imm val Jump

- R type

add $t0 $t1, $t2



- Imm

add $t0, $t1, 1

($t0 = $t1 + 1)

- lw, sw array

lw $t1, 0($s2)

lw $t1, (offset)

offset by multiple of 4

- beq → branch equal to
  beqz → branch equal to zero
  ⋮
  → I type
- J type (Jump type)
- syscall
- A → 
  $s1

lw → $s2
largest / $t0 ✓

- la , $v0, 4
  ↳ load address
  ↳ immediate

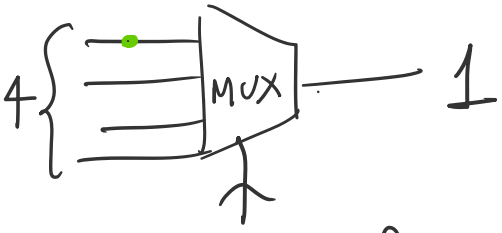
↳ load immediate

- move $t1, $t2
  $t1 → $t2
- function / sub routines / methods / procedure (same)
- fun($a0, $a1, $a2, $a3)
  ↳ arguments ($a0 - a3)
  ↓
  return $v0 → return ($v0)
-

Datapath Design

March 10, 2022 3:54 PM

- MUX

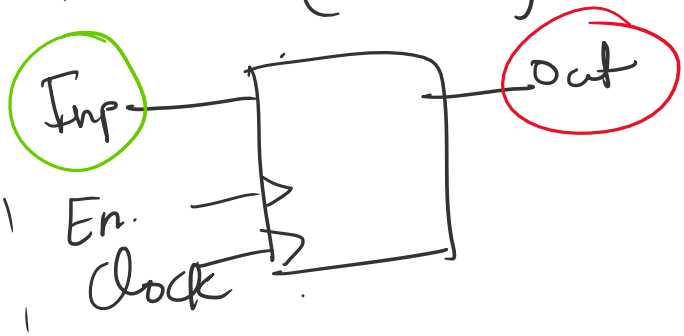


Datapath (Flow of info)

4:1

address through bus

- Decoder (Reverse of MUX)
- D-FF (1bit)



Ip	out
0	0
1	1

ak(x) → Memory

- add $f0, $f1, $f2

- Regwrite (Registry writing)
- ALU
- PC / IR (Instruction Register)

(Program counter) → Next instruction adder

→ current instruction

PC = IR+4

- Datapath (R Type – Register addressing)

Fields → Data in Register

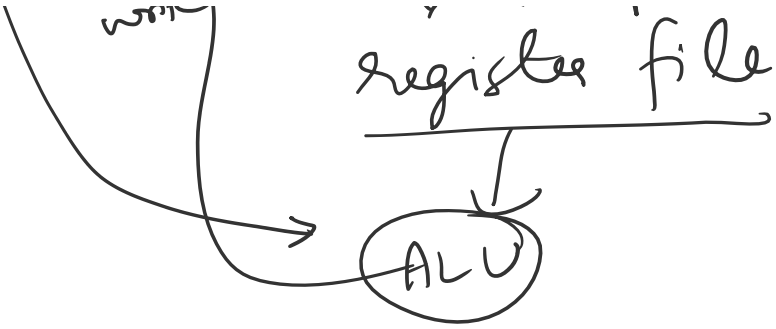
Writing / Reading op → Registers

- add $t0, $t1, $t2 ($t0 = t1 + t2)

write

read

register file



- Datapath (I type - Base addressing)
- lw uses longest path.
- Min clock period (single cycle Datapath)

Class	Fetch	Decode	ALU	Memory	Write Back	Total
R-format	2	1	2	0	1	6ns
LW	2	1	2	2	1	8ns
SW	2	1	2	2		7ns
Branch	2	1	2			5ns
Jump	2					2ns

Consider the frequency of operation as shown in table below. What will be the average time per instruction?

Instruction	Frequency
Arithmetic	50%
Loads	20%
Stores	10%
Branches	20%

- N instructions
- Total time = 8N

Actual usage time = $6 \times 0.5 + 8 \times 0.2 + 7 \times 0.1 + 5 \times 0.2$

$= 6.3\,N$ $5.6 + 0.7$

- Idle time $= 8N - 6.3\,N = 1.7\,N$

  % idle time $= \frac{1.7}{8} \times 100 = 21.25\%$

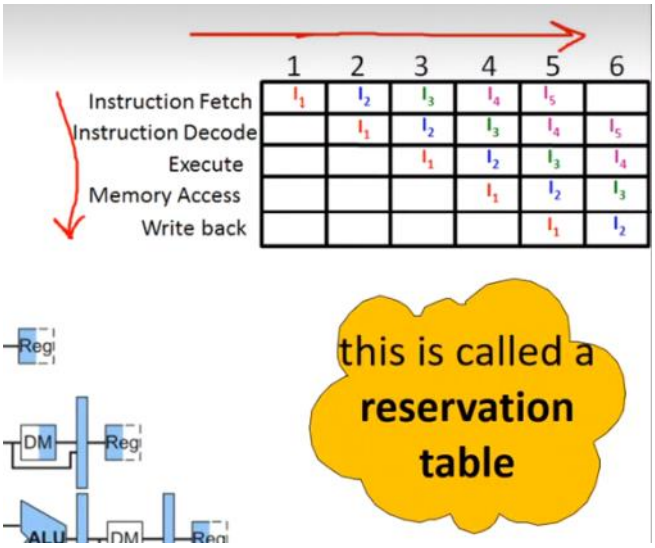
- Hardware utilisation $= 100 - 21.25$
  $= 78.75\%$

- Multi cycle Datapath
  ↳ split the steps

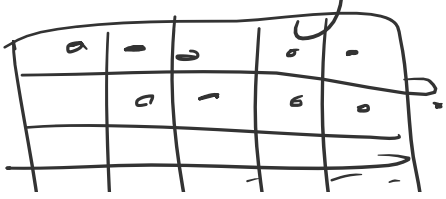
Pipelining

April 7, 2022 2:44 PM

- Out of order execution (No straight order of Exc) → Random
- Branch predictions
  - Branch statements take more time
- Superscaler processors
  - Multicycle
- VLIW (very long instruction word)

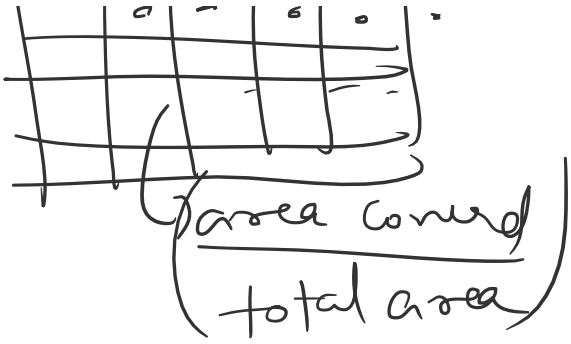


- Stage delay, speedup, efficiency
  - ↓
  - [illegible]



- Calculation

$\left(\frac{T_{wp}}{T_p}\right)$



- Hazards in Pipelining

1) Data Hazard → Read after Write (RAW)

2) Structural Hazard → Multiple instruction same resource

3) Control Hazard → Control type jump instructions, flushes immidiate instructions

- Stall → doing nothing in pipeline stage

- Control Hazard

Flushing the intermediate steps

- Flushing the intermediate steps in branching type.
- Highly performance duplete

- Solution? → decider in IF stage
  create stall for next instruction
  ↳ saving intermediate steps

- Data Hazards (Read after write)

★ Read after write (RAW)

$R_1 \leftarrow R_2 + R_3$

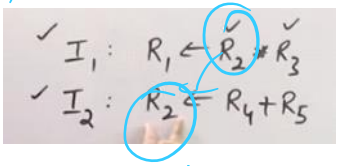
$R_5 \leftarrow R_1 + R_4$

	Domain	Range
I_1	R_2 R_3	R_1
I_2	R_1 R_4	R_5

$I_1(\text{Range}) \cap I_2(\text{Domain}) \neq \phi$

- Write after Read (WAR)

Domain	Range



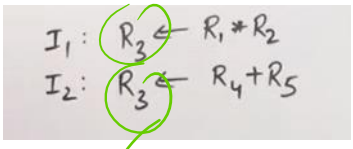
Domain | Range

I_1 $R_2 R_3$ - R_1

I_2 $R_4 R_5$ R_2

I_1(Domain) $\cap$ I_2(Range) $\neq \phi$

- write after write (WAW)



(2 diff values at same time)

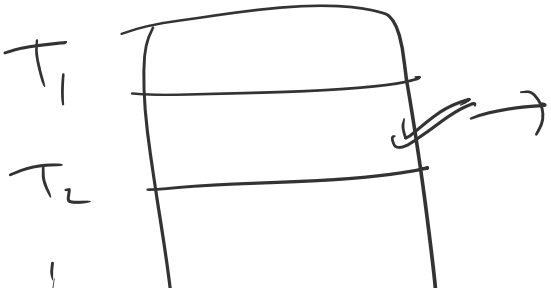
	Domain	Range
I_1	R_1, R_2	R_3
I_2	R_4, R_5	R_3

Range (I_1) $\cap$ Range (I_2) $\neq 0$

- for RAW, WAW, WAR
  (True) (output) (Anti)

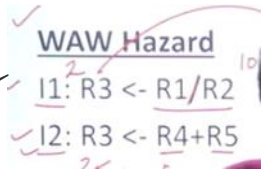
↓

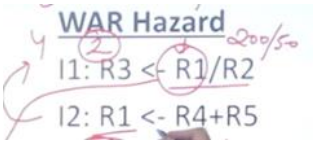
Register Renaming



→ Take last value

WAR Hazard

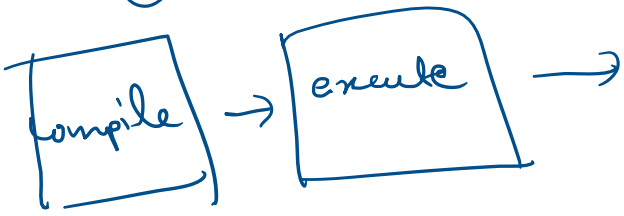




Storing temp values but not overriding registers

- Remove stalls → Data forwarding

- In-order / Out order execution



Compiler generated order

→ compiler generated order not followed.

→ dynamic changing of order

- Instruction reordering
- For WAW, WAR →

for

Register Renaming / Reallocation

location changes but final structure same

- Architecture, OS, Compilers
- Loop unrolling

- Example

```
for (i=0; i < 16; i++) {
c[i] = a[i]+ b[i];
}
```

Unroll once

```
for (i=0; i < 8; i++) {
c[2 * i] = a[2 * i] + b[2 * i];
c[2 * i+1] = a[2 * i+1] + b[2 * i+1] ;
}
```

more wasted clock cycles

Runs faster

More register pressure

How data hazards can be eliminated?

- Summary

How data hazards can be eliminated?

- Summary
  - Detect and wait (stalling unnecessarily)
  - Data forwarding
  - Reordering (out of order)
  - Renaming (to remove WAR and WAW hazard)
  - Loop unrolling and reordering

Branch Prediction

Branch prediction deals with branch penalty by continuing executing the instructions following the branch speculatively.

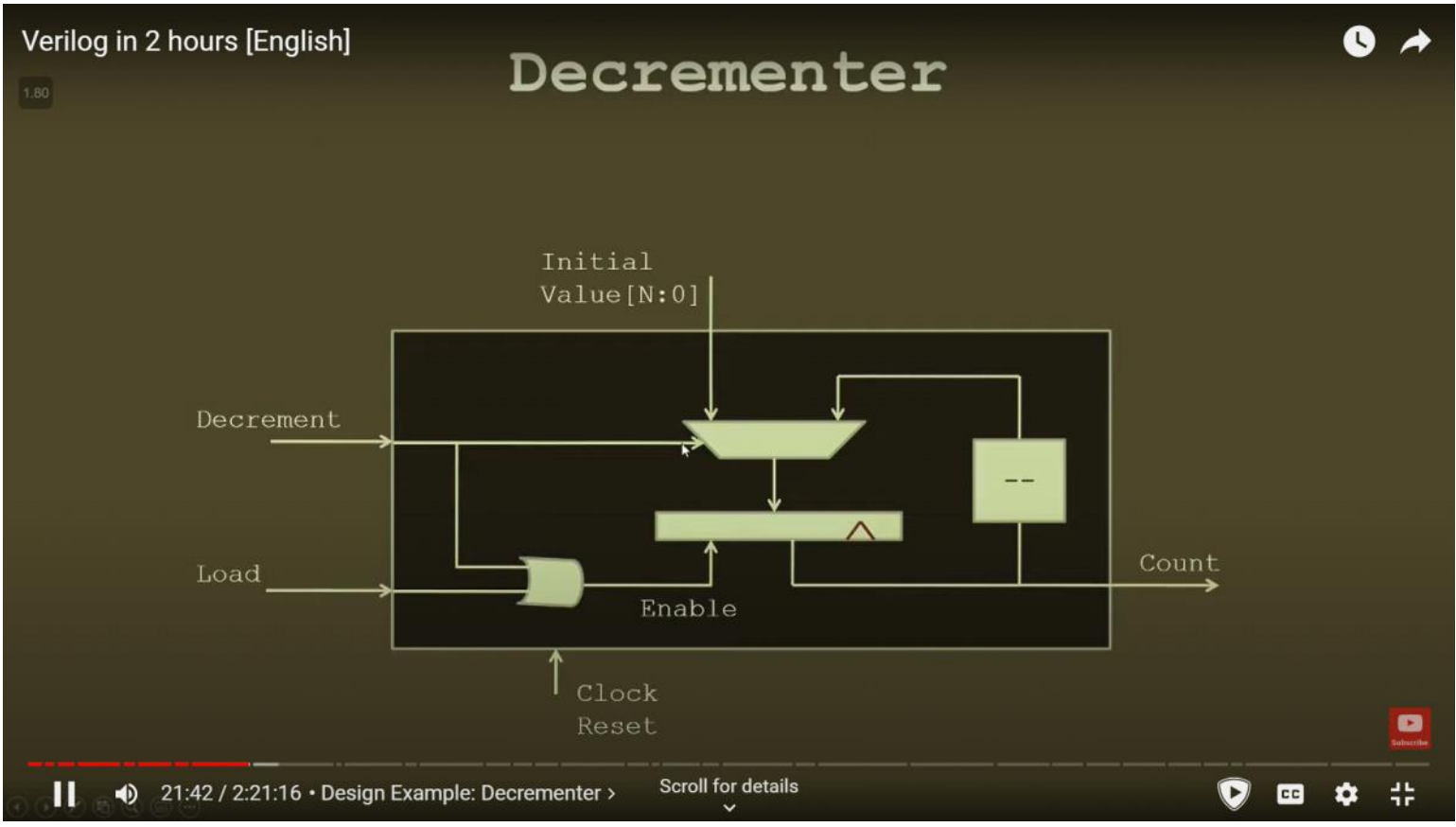
- If the branch does not occur then there was no pipeline stall.
- If, however, the branch does occur, the pipeline must be flushed of the speculative instructions.
- On average, this reduces pipeline stalls by 50%.

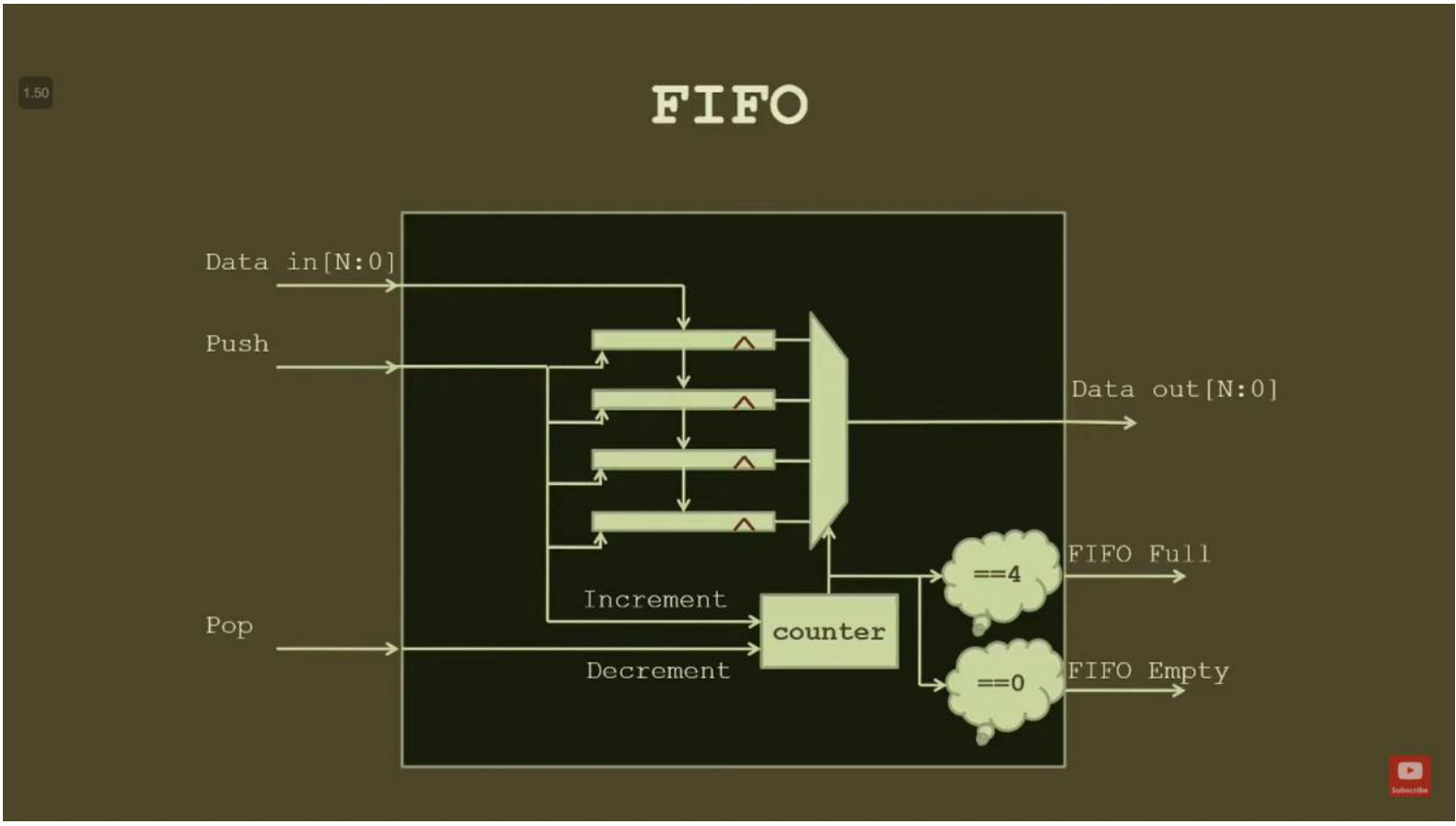
Static branch prediction techniques:- The actions for the branch are fixed for each branch during the entire execution. (when behaviour is highly predictable)

Dynamic branch prediction techniques:- The prediction decision may change depending on the execution history. (when behaviour is not predictable)

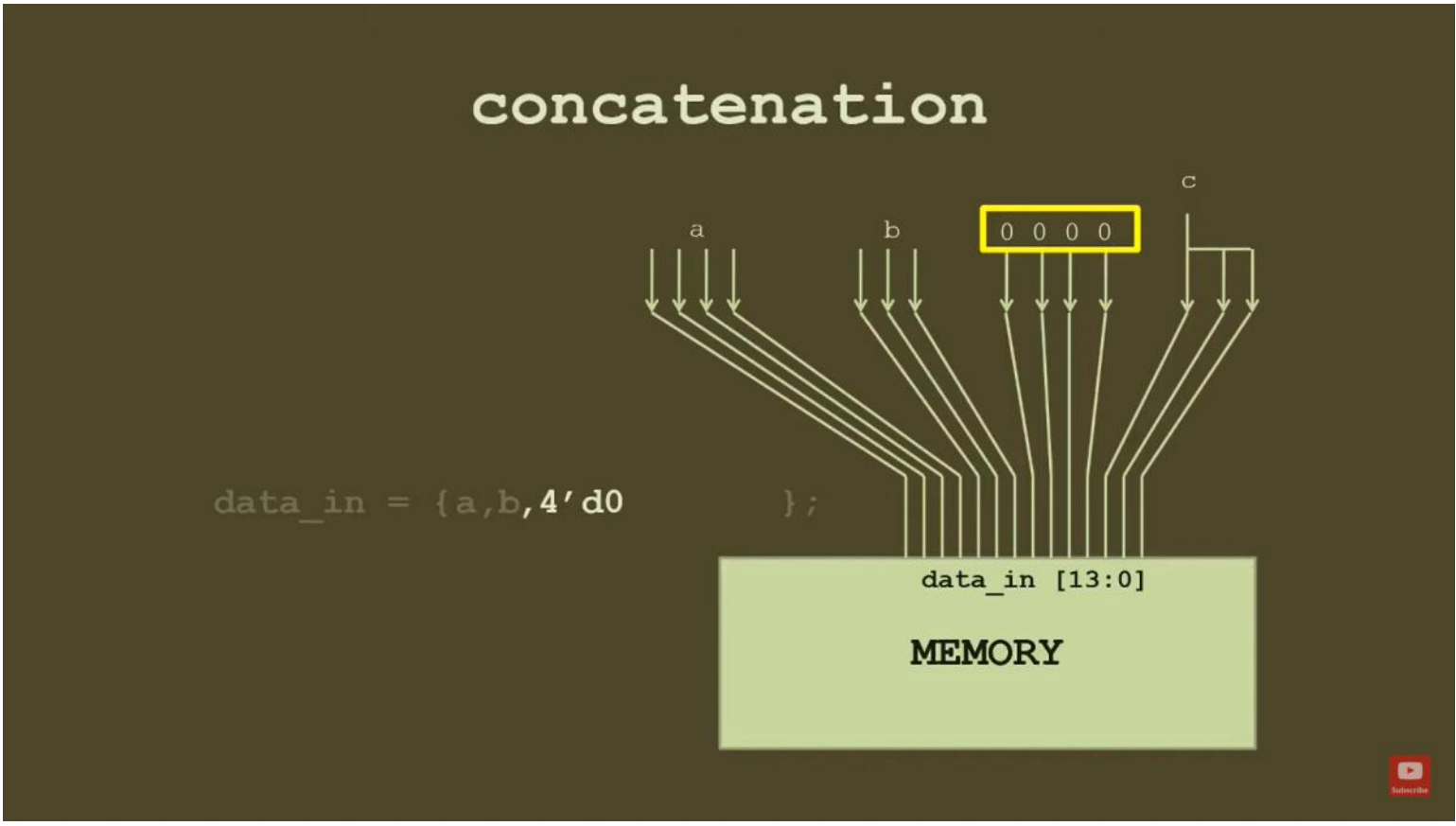
Verilog

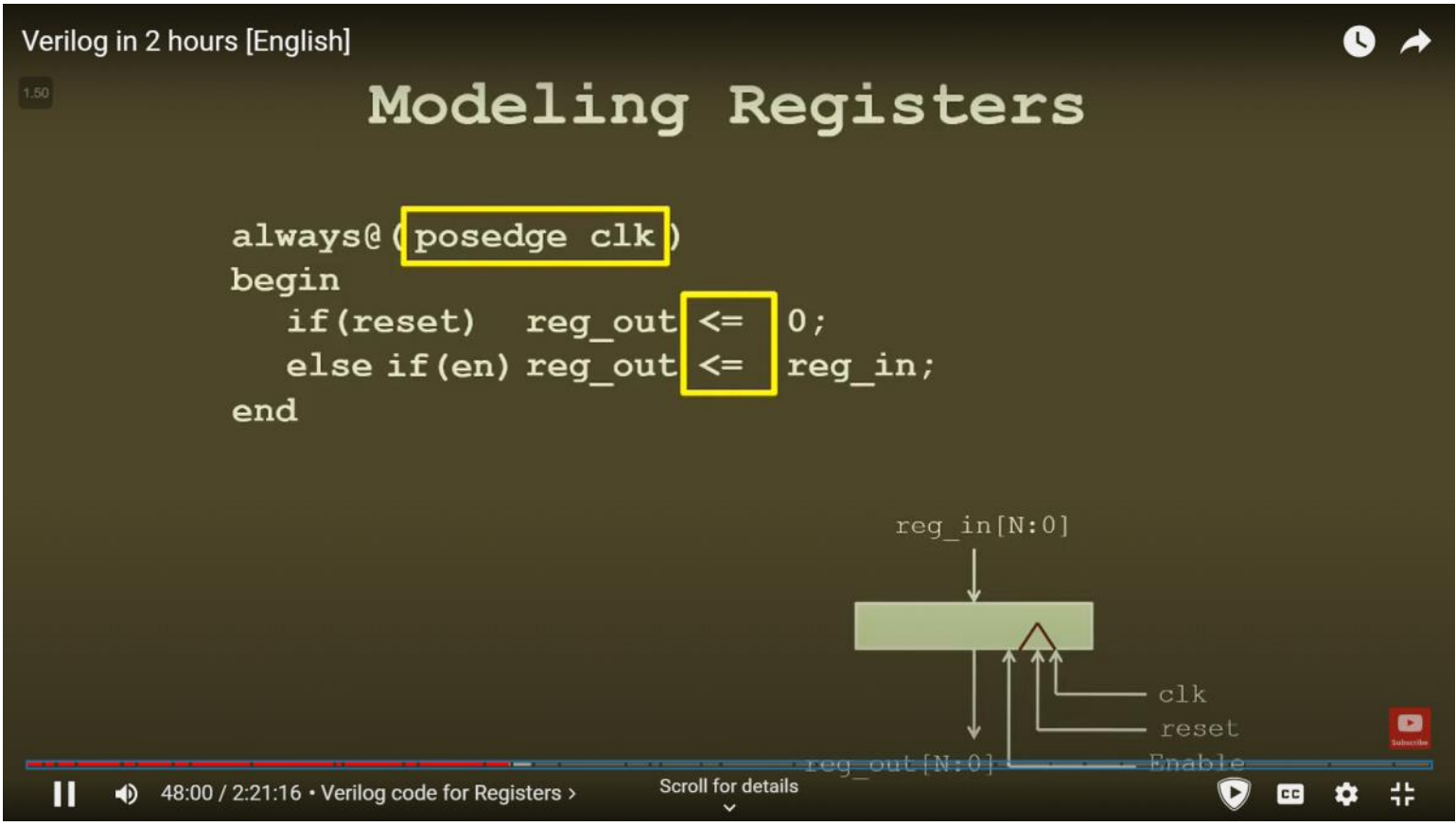
April 22, 2022 1:20 AM




FIFO
Data in[N:0]
Push
Pop
Increment
Decrement
counter
==4
==0
Data out[N:0]
FIFO Full
FIFO Empty

Review FIFO

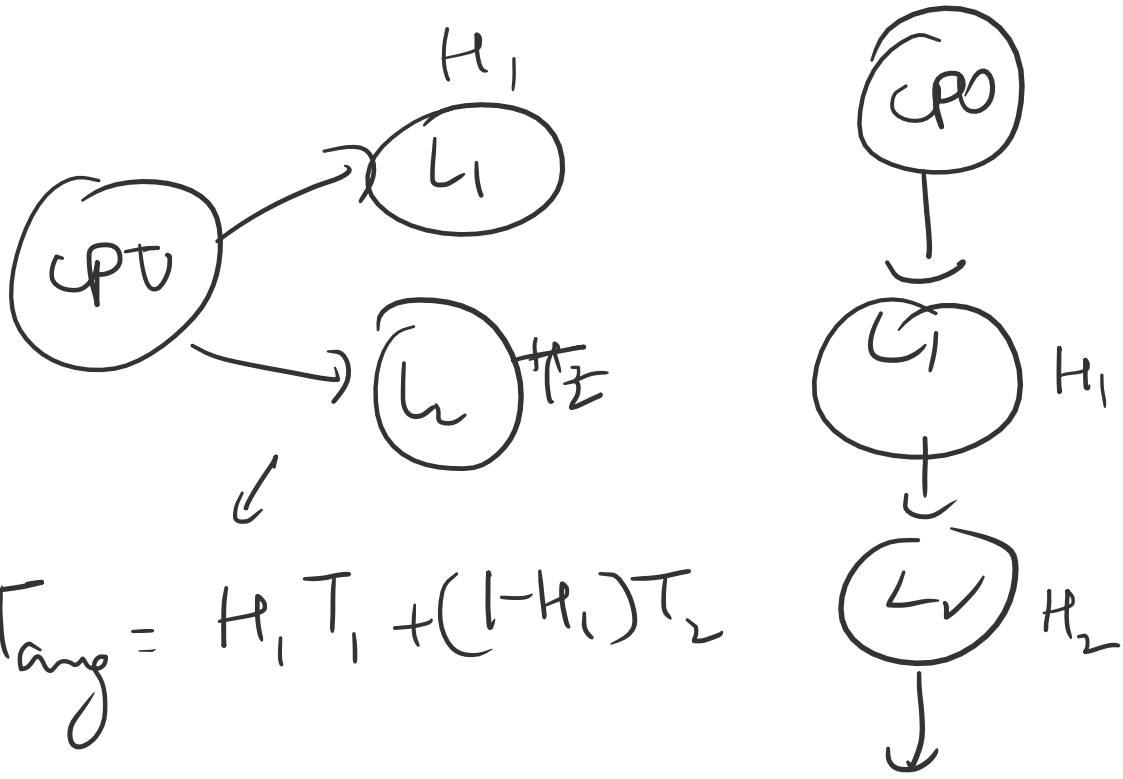

concatenation
a
b
0 0 0 0
c
data_in = {a,b,4'd0 };
data_in [13:0]
MEMORY


Verilog in 2 hours [English]
1.50
Modeling Registers
always@(posedge clk)
begin
if(reset) reg_out <= 0;
else if(en) reg_out <= reg_in;
end
reg_in[N:0]
clk
reset
Enable
reg_out[N:0]
48:00 / 2:21:16 • Verilog code for Registers ›
Scroll for details

Memory organization

May 4, 2022 9:37 PM

- Independent / Hierarchical

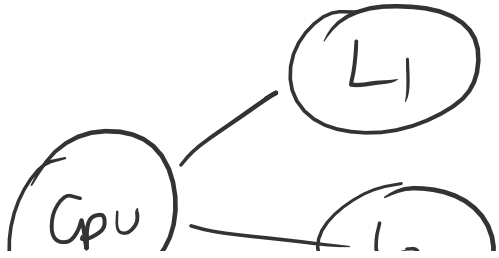


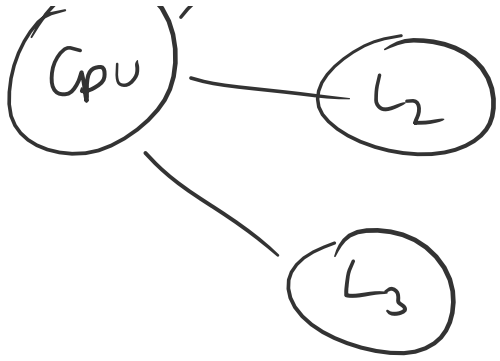
$$\rightarrow T_{avg} = H_1 T_1 + (1-H_1) T_2$$

$$T_{avg} = H_1 T_1 + (1-H_1)\, H_2 (T_1 + T_2)$$

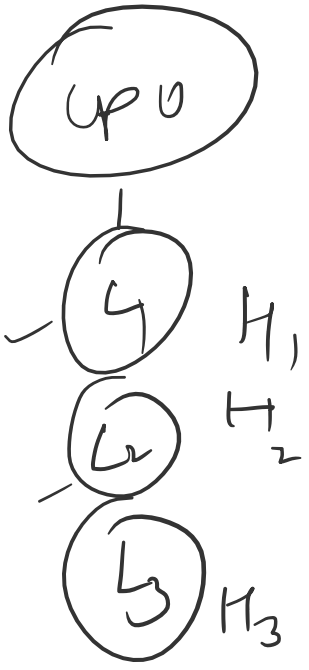
---

3 Level





$$T_{avg} = H_1 T_1 + (1-H_1) H_2 T_2 + (1-H_1)(1-H_2) T_3$$



$$T_{avg} = H_1 T_1 + (1-H_1) H_2 (T_1 + T_2) + (1-H_1)(1-H_2)(T_1 + T_2 + T_3)$$

$$J$$

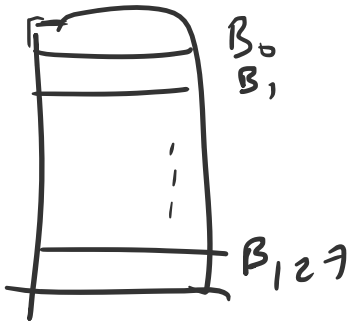
Hit ratio =

Cache Mapping

a) Direct Mapping –

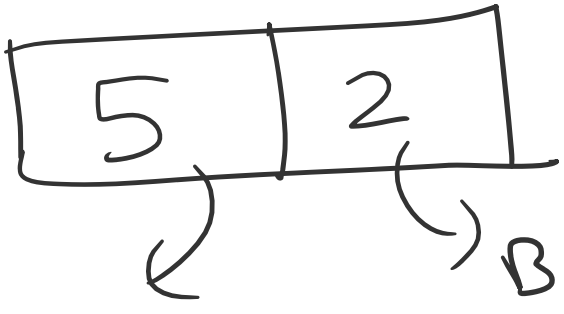
Cache → Lines,

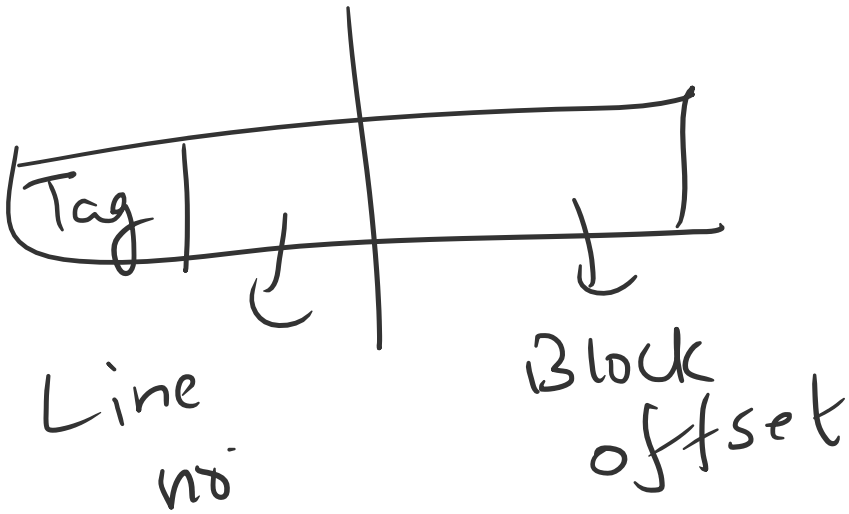
Main Memory → Blocks



i mod n

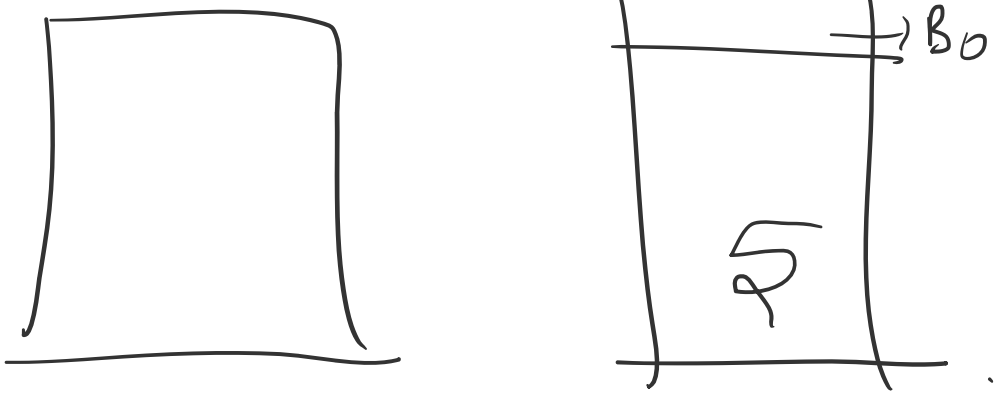
0 - 127 (128 words)



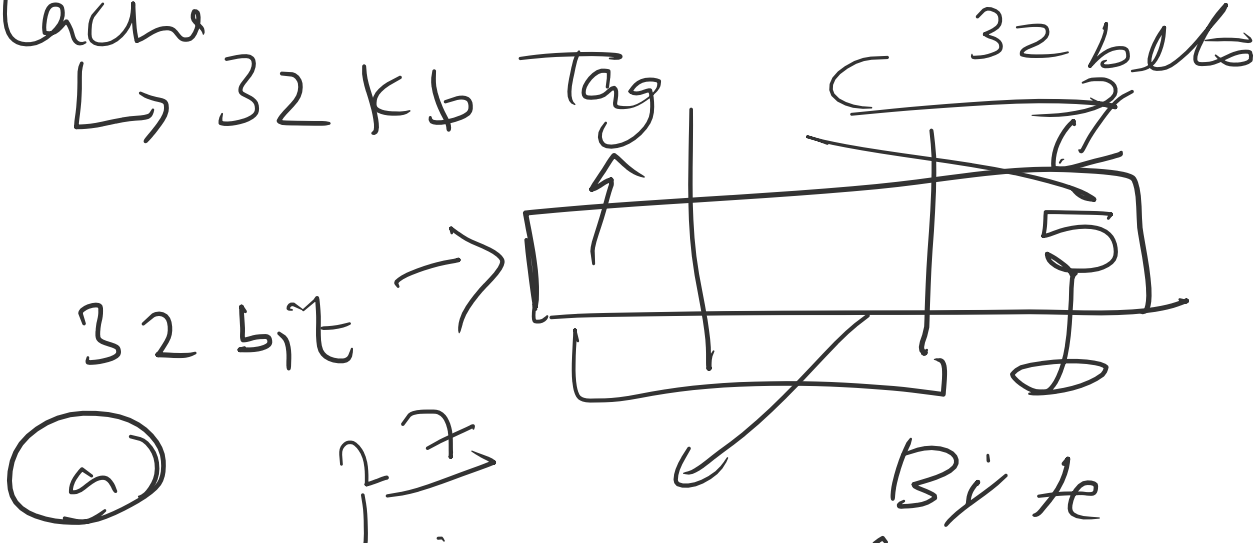


- Cache Hit / Miss

Block size 32 kB



Cache
↳ 32 kb



(a) Line Byte
zzz
numbers Size

$\log\left(\frac{\text{Size Main Memory}}{\text{Size cache Memory}}\right)$

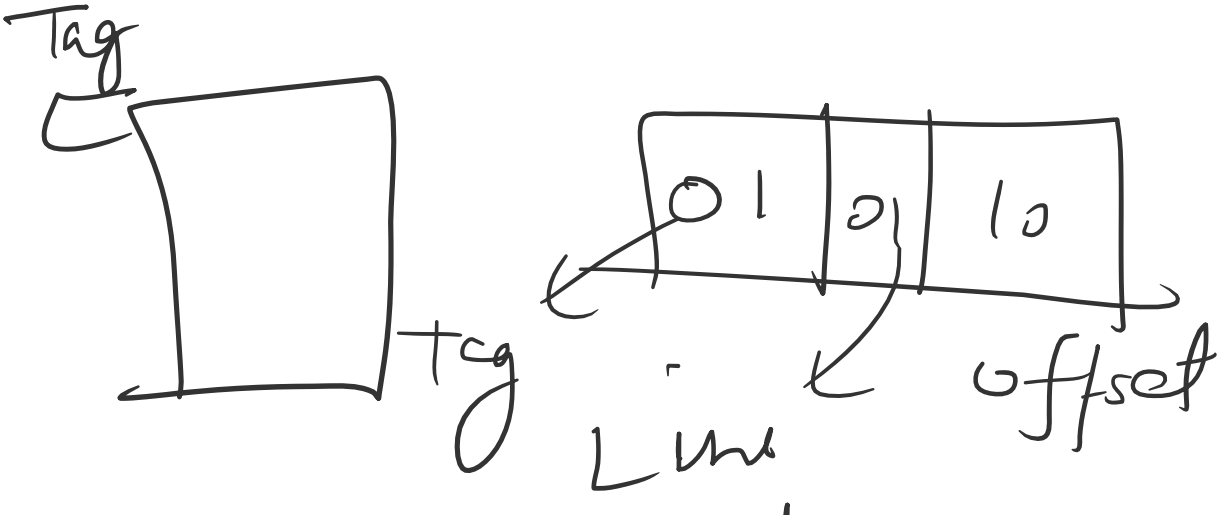
=

b) Fully associative

- More hits

No fixed position

- Ad/ pisad - Direct Mapping

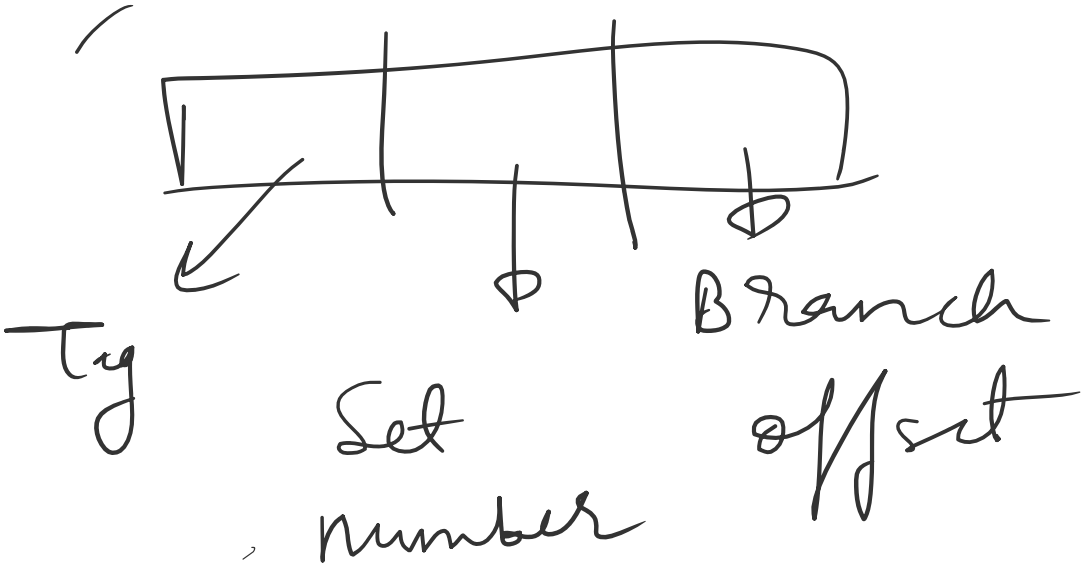


U Line number

- Easy access
- Disadvantage
  ↓
  conflict Miss

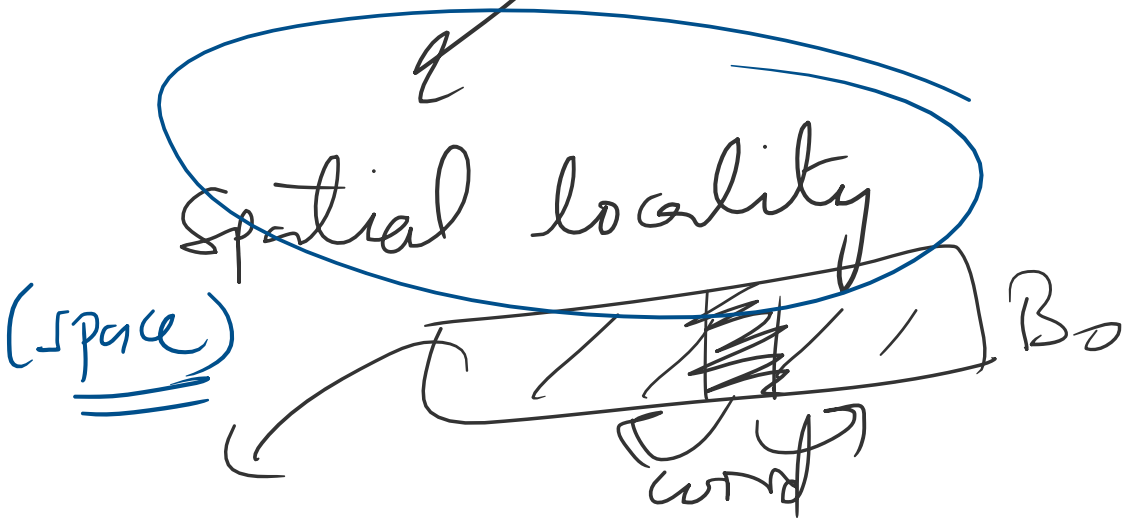
C) Set associative

- (Direct + Fully associative)



- Locality of Reference

- Locality of Reference



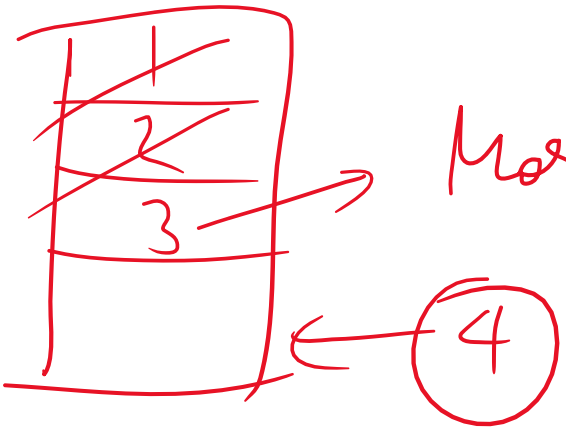
Cache

↳ More hits

- Higher block size ↓

More spatial locality

- Temporal locality (time)



More probably to come next

LRU → Least recently used

Cache replacing policies

- FIFO
- LRU
- MRU
- Random

(set associative Mapping)

- Main Memory - RAM
- HDD
- LRU

- FIFO