Combinational Logic:

X	Y	Z	O
0	0	0	
.	.	.	
.	.	.	
.	.	.	
1	1	1	

Sequential Logic:

Temporal → FSM

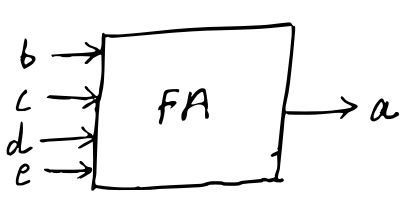
$M(S, S_0, I, O, \delta, \lambda)$

$\delta : S \times I \rightarrow S$

$\lambda : S \rightarrow O$ (Moore)

$S \times I \rightarrow O$ (Mealy)

$a = b + c + d + e$



BUT THIS IS TOO RIGID!

Each change requires complete change in hardware.

Thus, what we WANT?

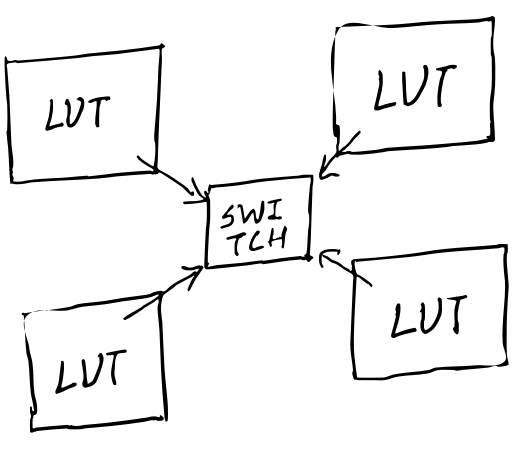
Flexibility

what we NEED?

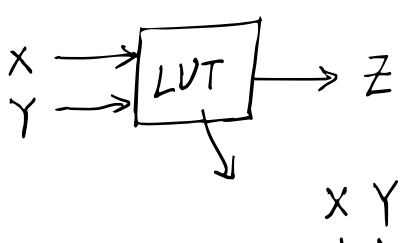
Programmable Systems

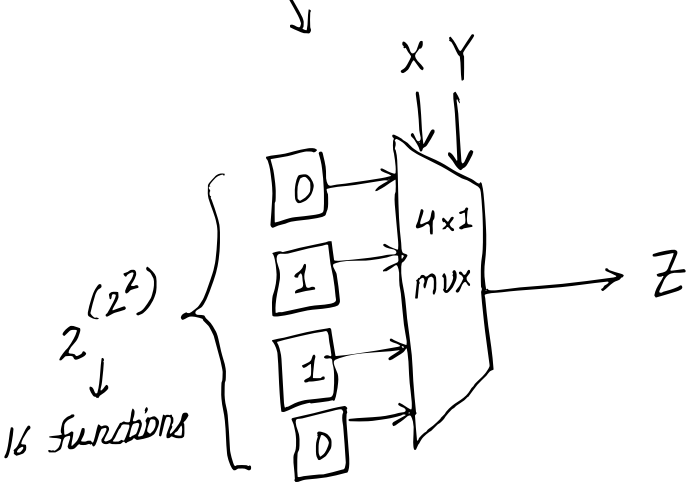
↳ Made using Generic Hardware

FPGA → Field Programmable Gate Array



LUT → Look Up Table





OR

Use flexible fabric and program it using software.

$a = b + c + d + e + f$

$t_1 = b + c$

$t_2 = t_1 + d$

$t_3 = t_2 + e$

$a = t_3 + f$

In most systems:

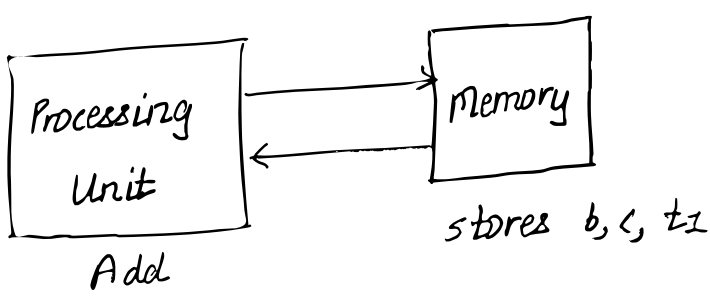
S/W → Program

Interface (Instruction set architecture) ISA

H/W ⟹ Processor (micro-processor) (Generic)

Through ISA, hardware should understand commands like

Add b, c, t_1



$a = b + c * d$

$t_1 = c * d$ MUL c, d, t_1

$a = b + t_1$ ADD b, t_1, a

But what is the minimal set of instructions do we need to specify before-hand in generic hardware to make all operations possible?

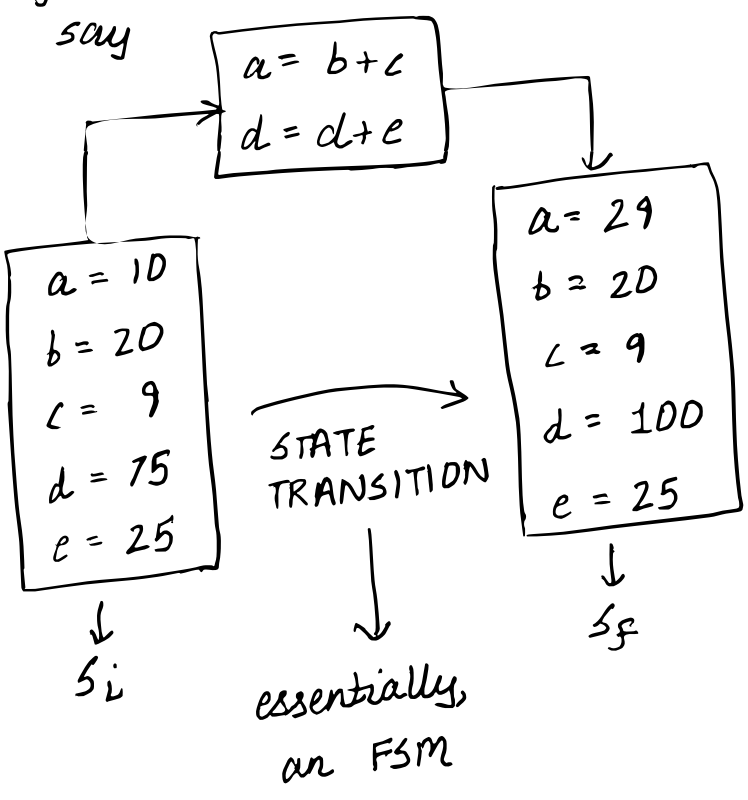
Church-Turing Thesis: 3 sets of instructions required

① Logical

② Memory communication (READ / WRITE)

③ Control Flow Change (DATA DEPENDENT / INDEPENDENT)

E.g.



Programmable Computing System

S/W

H/W

Interface
(ISA)
(Language)

Algorithm

↓

Hardware
↳ Software
(Generic System)

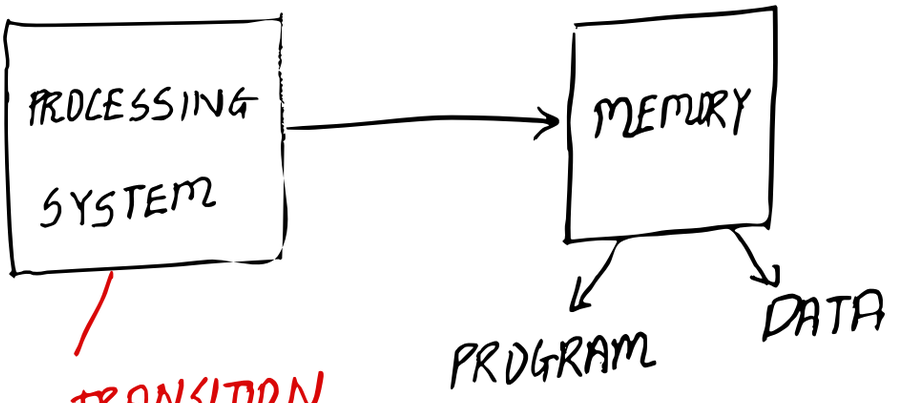
Programmer's Model
(ISA)

Specification

→ Logical/
Arithmetic

→ Memory
Communication
(Read / Write)

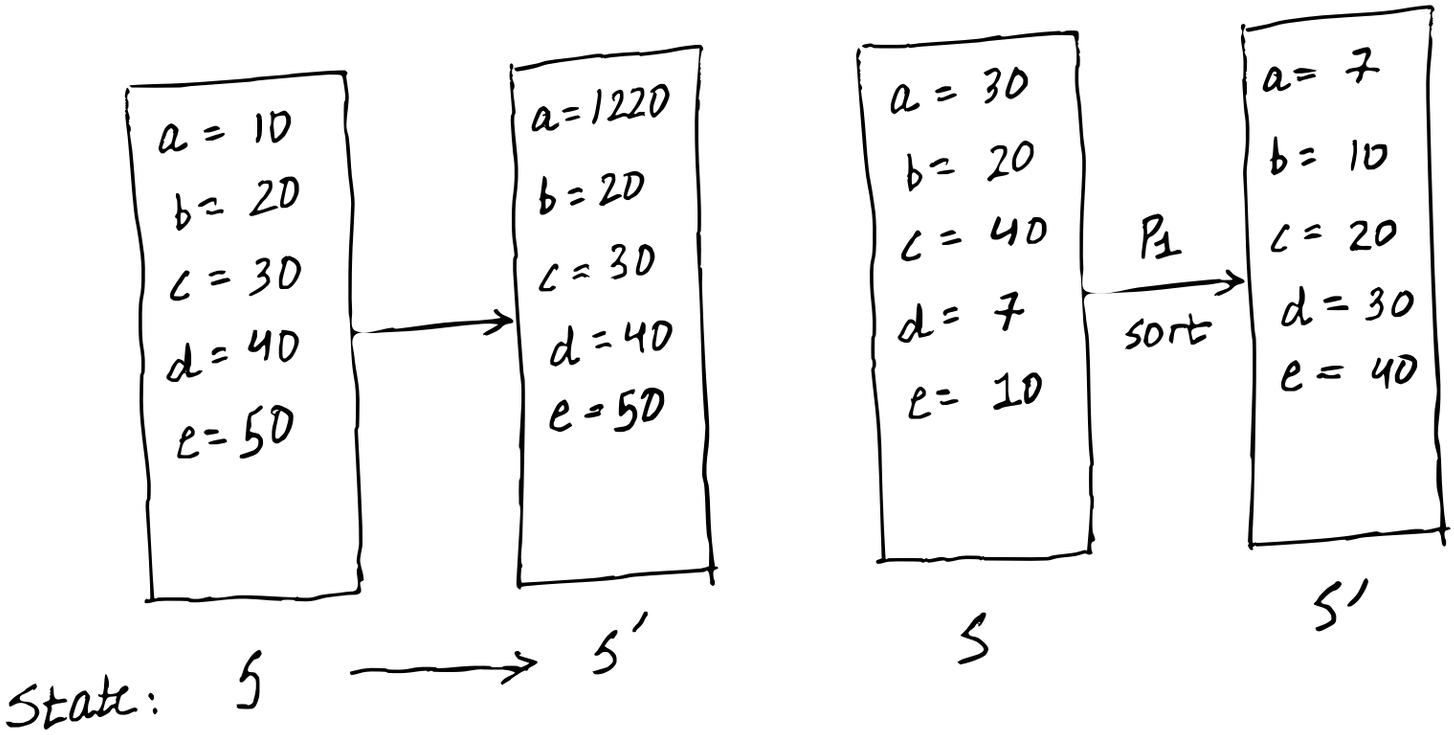
→ Change in control
flow
(conditional/
unconditional)



STATE TRANSITION
SYSTEM
(FSM)

e.g.

$P\{a = b + c * d\}$



S/W

H/W

↳ Instruction Format

- OPERATION
- OPERAND(S)
- Where to store result? (DESTINATION)

OPERATION	OPERAND1	OPERAND2	DESTINATION

If you have 8GB RAM

↓

$2^3 \times 2^{30}$ bytes

$= 2^{33}$ bytes

$\therefore$ <u>33 bits</u> required to specify memory <u>location</u> of a byte.

OPERATION	OPERAND1	OPERAND2	DESTINATION
↓	33	33	33
20 operations:	a	b	c
↓			
ceil $[\log_2^{20}] = 5$ bits required			

$\therefore$ <u>104 bits</u> required for 1 instruction.

MEMORY is the SLOWER of the 2 units.
↓
PROCESSING UNIT, MEMORY

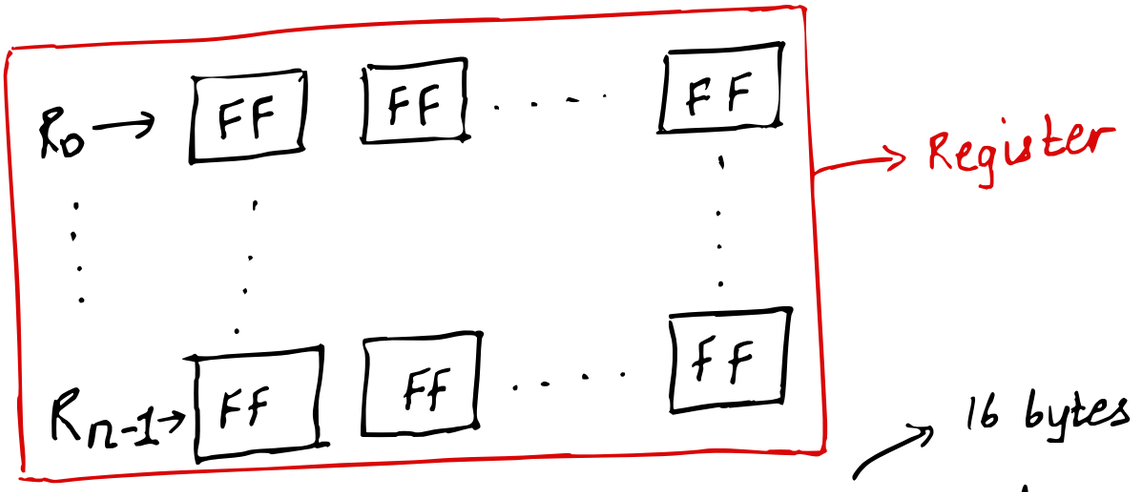
↓

still takes about 200 cycles to access 1 storage element.

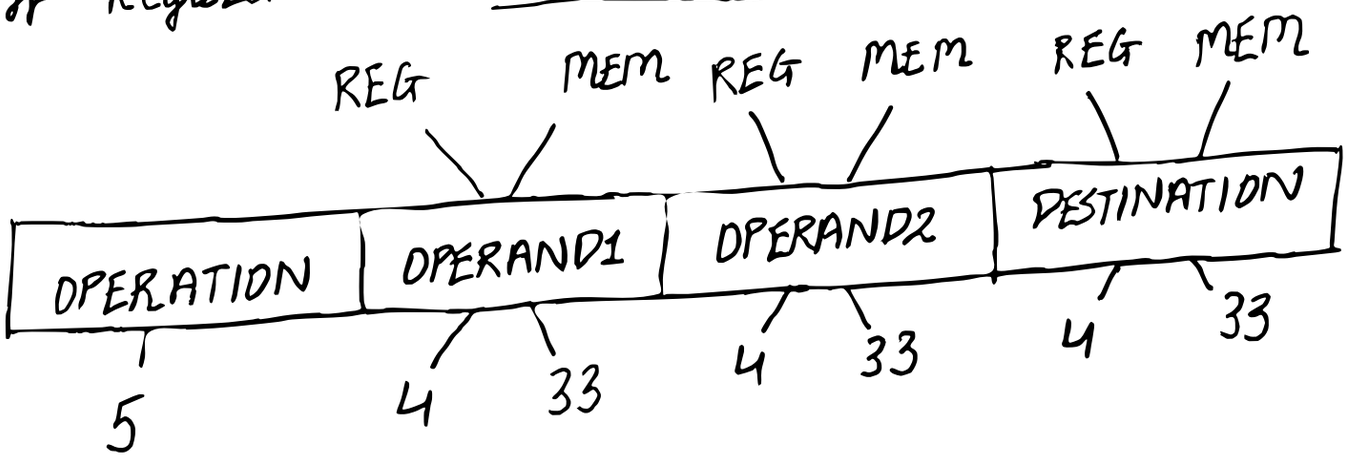
$\therefore$ We need some temporary storage in processing unit.

↳ <u>Register</u>

↳ made of Flip-Flops

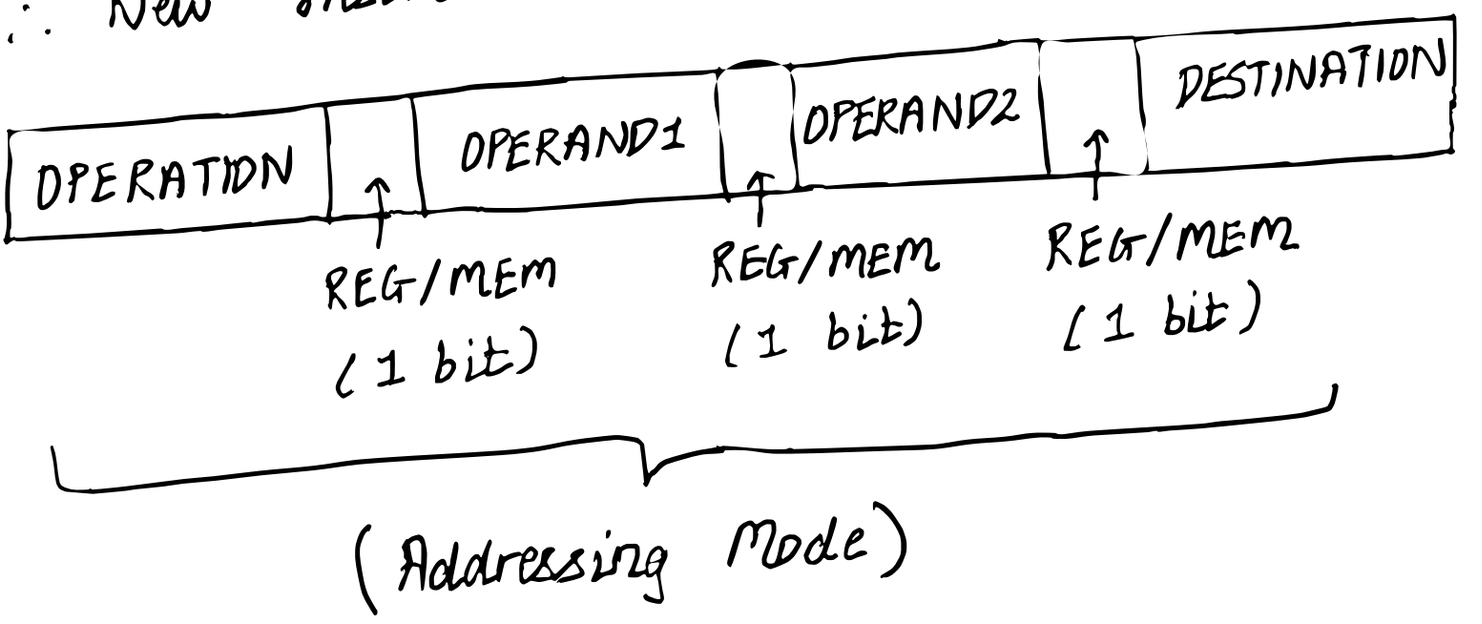


If Register has 16 storage elements → 4 bits (16 bytes)



But we need to specify whether to check in REG or MEM.

∴ New Instruction Format:



Say, instructions:

⋮

$$\left.\begin{array}{l} a = b + c \\ \vdots \\ p = a + r \\ \vdots \\ d = a + e \end{array}\right\} \text{Lifetime of } a$$

$\therefore$ Min. no. of bits :- $5 + 3 + \underbrace{3 \times 4}_{\text{REG}} = \underline{\underline{20}}$ bits

Max. no. of bits :- $5 + 3 + \underbrace{3 \times 33}_{\text{MEM}} = \underline{\underline{107}}$ bits

Say for arithmetic & logical instructions, we restrict ourselves to only registers :

Only <u>17 bits</u> required

OPERATION	OPERAND1	OPERAND2	DESTINATION
5	4	4	4

Memory Read / Write :

OPERATION	OPR 1	DEST

5 REG 4 $\longleftrightarrow$ 33 MEM

or

MEM 33 $\longleftrightarrow$ 4 REG

$\therefore$ <u>42 bits</u> required.

Register has limited storage, how do we execute iterative statements using register?

e.g.

```
for (i=0; i < 1000000; i++)
{
    A[i] := A[i] + 5;
}
```

We don't have 1000000 storage elements in register.

What do we do ?

$A \longrightarrow r_1$

$\underbrace{r_1 = r_1 + 4}_{\text{change } r_1 \text{ only}}$

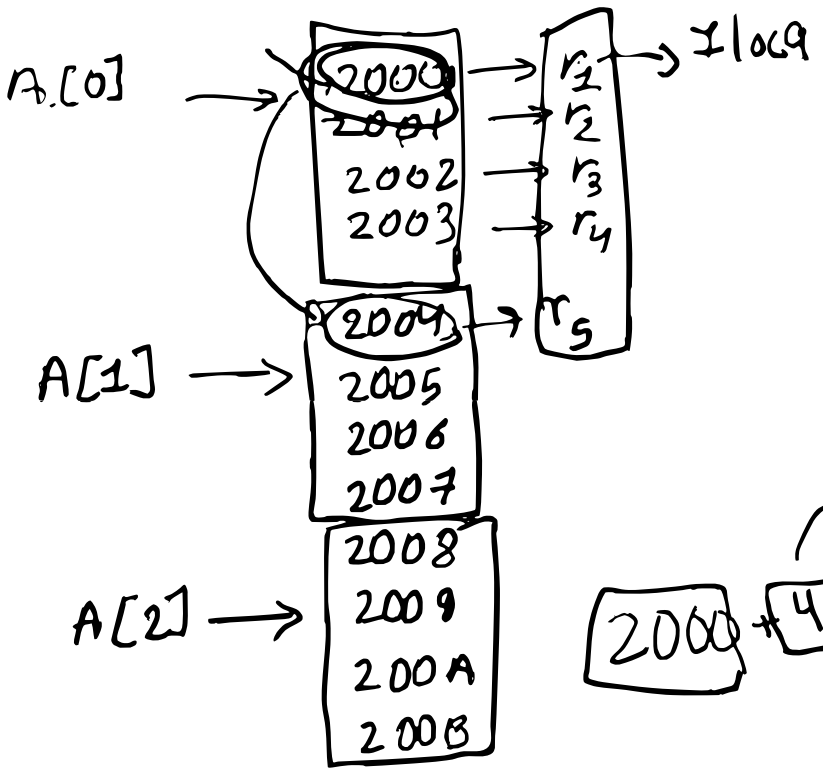
OR

$A[0] \longrightarrow r_1$

$\boxed{i} \longrightarrow r_2$

$r_2 ++$

$\underbrace{[r_1 + 4r_2] = [r_1 + 4r_2] + 5}$



$r_1 \rightarrow$ Base (B)

$r_2 \rightarrow$ Index (I)

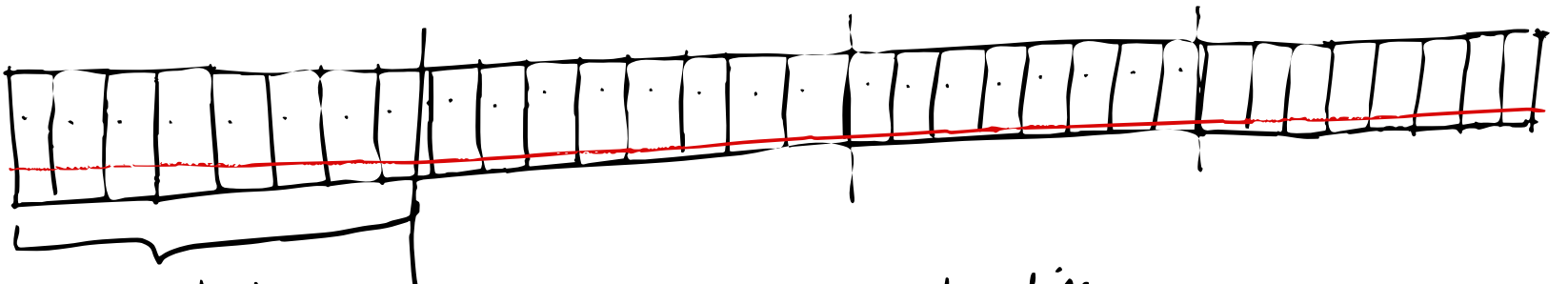
B + I notation

2000 + 4i

r ⇒ 16 bytes (storage)

→ 4 bit (address)

Integer requires ⇒ 4 bytes

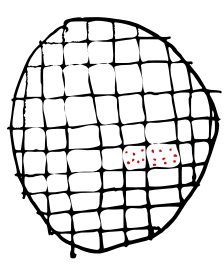


1 memory location can store 1 byte <u>only</u>

Prof. Janak Patel, UIUC

Verification

Testing (Manufacturing Test)



Silicon wafer → package

↓

each grid → 1 logic circuit

↓

Robotic arm goes from grid to grid and probes the circuit.

Fault Modelling:

Physical Design

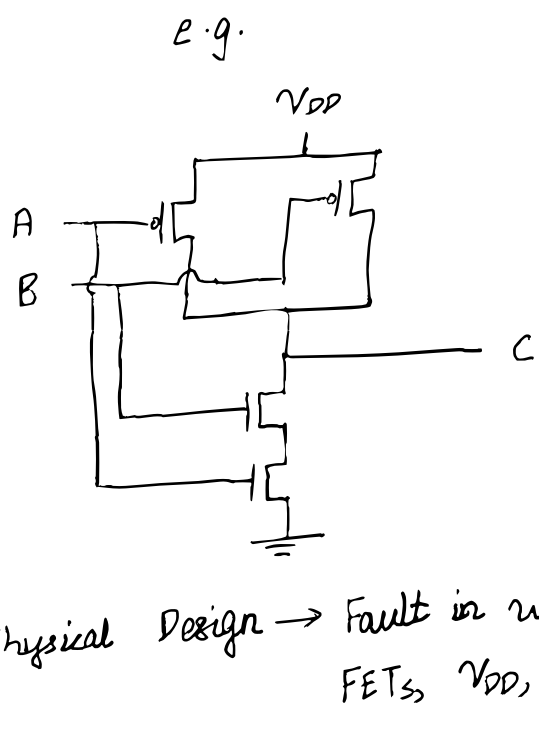
↓

Circuit

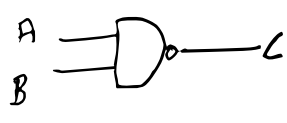
↓

Logic

e.g.



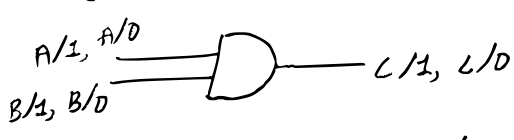
Physical Design → Fault in wires, FETs, V_{DD}, GND



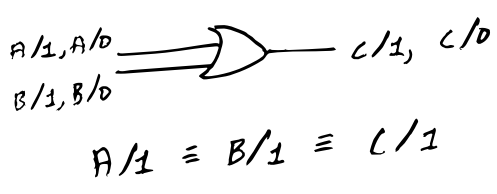
Circuit → Testing

↳ set a logic stuck at 1 or 0

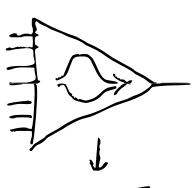
e.g.



A/0 ≡ B/0 ≡ C/0



A/1 ≡ B/1 ≡ C/1



↓

If Tree circuit, can be tested by stuck logics

If Fan-out, cannot.

ATPG – Automatic Test Pattern Generator

History:

1962 – Prof. Seshu Sunderan, Univ. of Illinois

1965 – John Paul Roth, IBM – D-Algorithm

1981 – Prabhu Goel, IBM – PODEM

1988 – Prof. Schulz, Univ. of Munich,
Trischler and Sarfert, Siemens

1998 – Harnzuoglu & J. Hinter, Univ. of Illinois
↳ ATOM → fastest algorithm present today

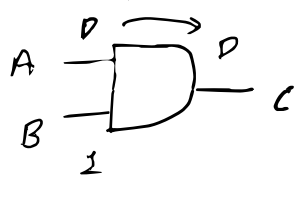
D-Algorithm:

error changes a line from 1 to D

D $1 \rightarrow 0$

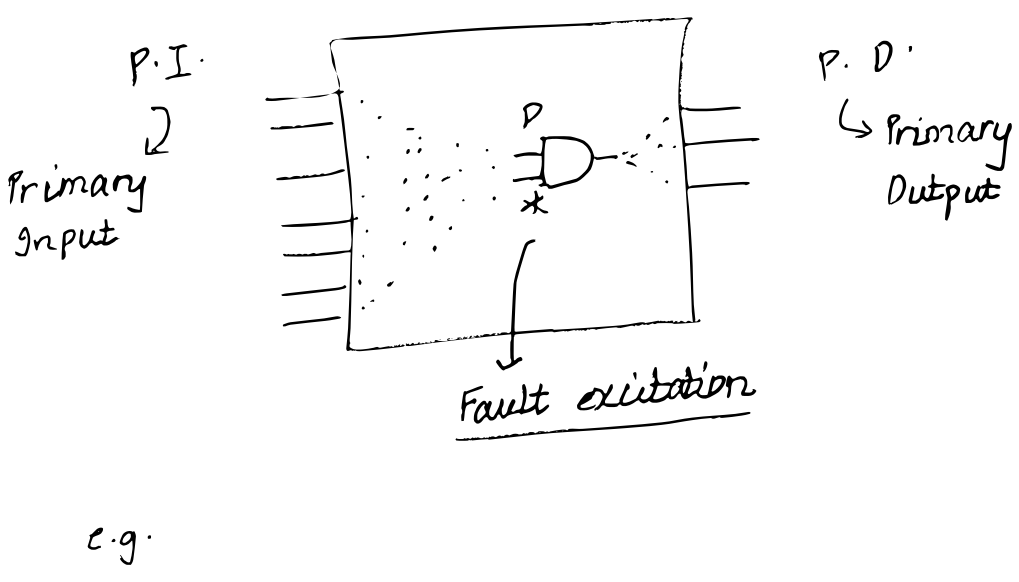
$\bar{D}$ $0 \rightarrow 1$

propagation

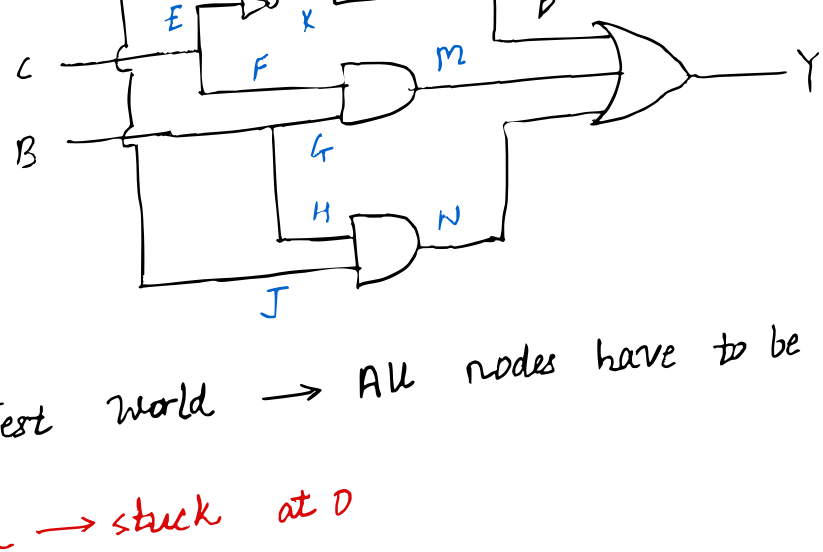


PODEM:

Branch-and-Bound Algorithm



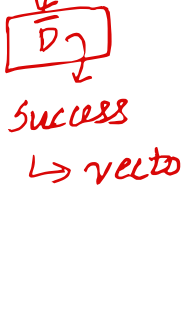
e.g.



Test World → All nodes have to be tested.

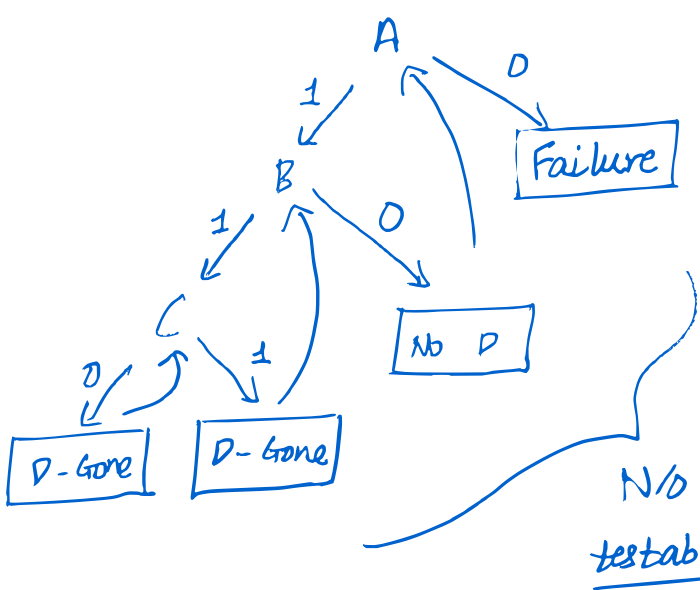
L → stuck at 0

A → 0



Now, take N stuck at 0

↓



N/0 is not testable

↓

Wire N is redundant

even if we remove it, function does not change

Test vector T_1

↓

Fault Simulate T_1

Test vector T_2

↓

Fault Simulate T_2

T_1
T_2
⋮
T_n

↑ Reverse Fault Sim.

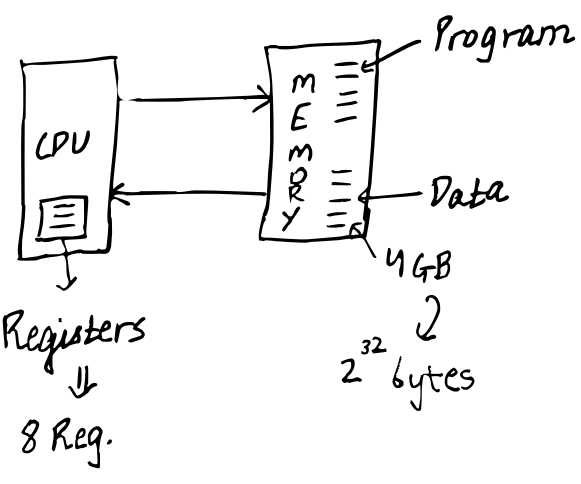
Programmable System Design:

Computing System

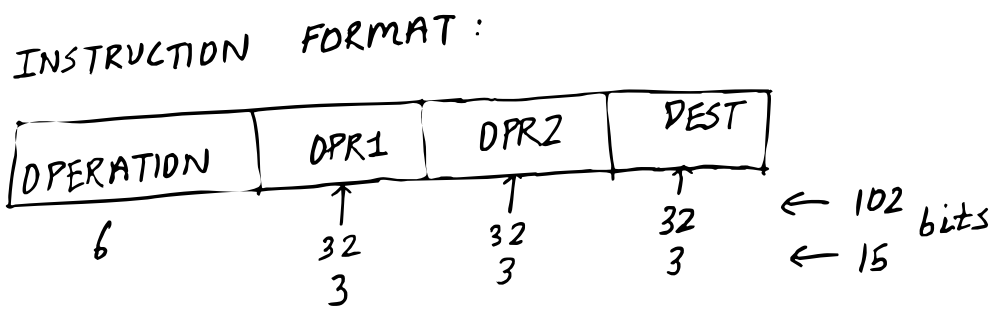
→ S/W ← ISA

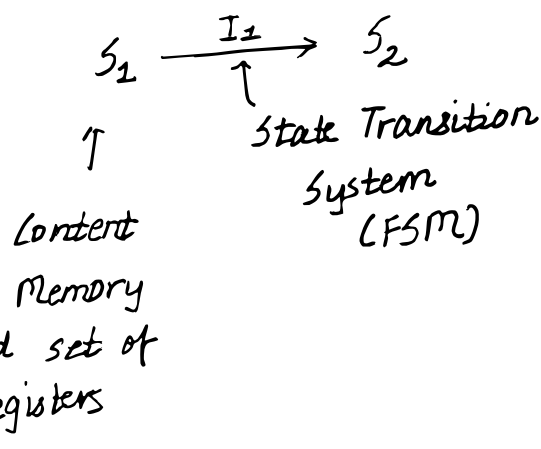
H/W → Logical
→ Memory communication
→ Change control flow

Church Turing thesis

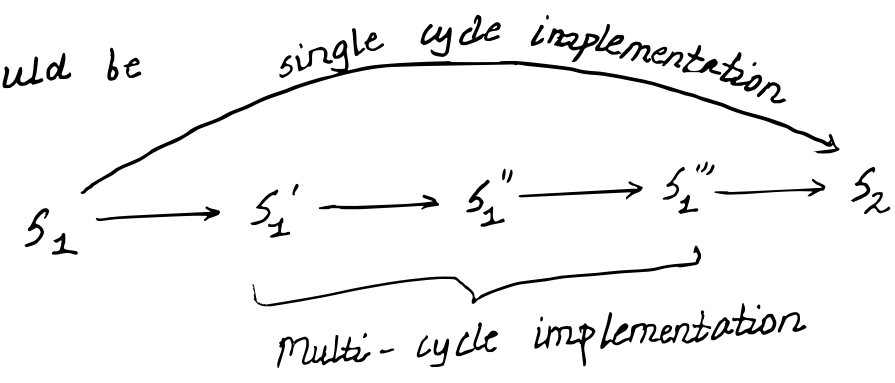


INSTRUCTION FORMAT:





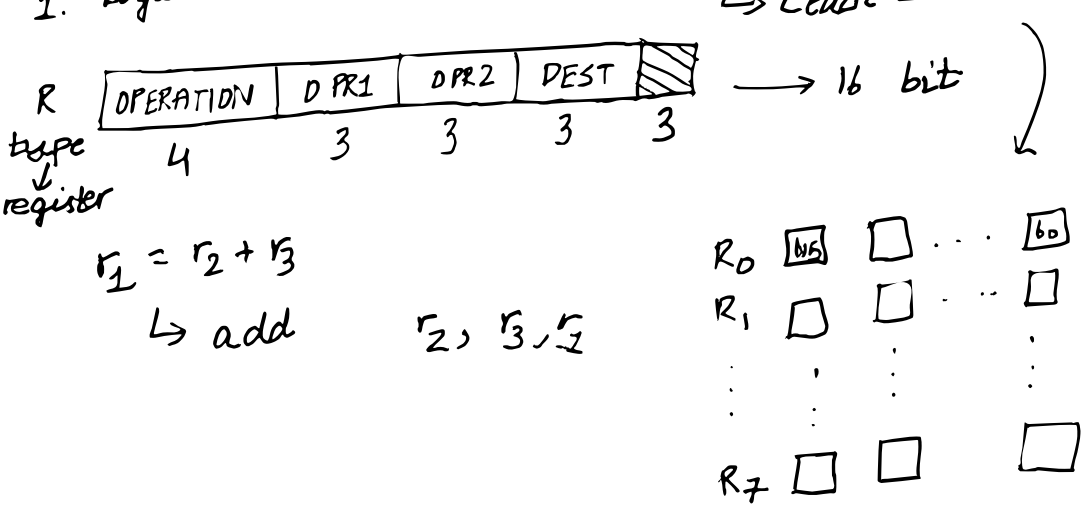
↑
Content of Memory and set of Registers



Instruction:

1. Logical / Arithmetic

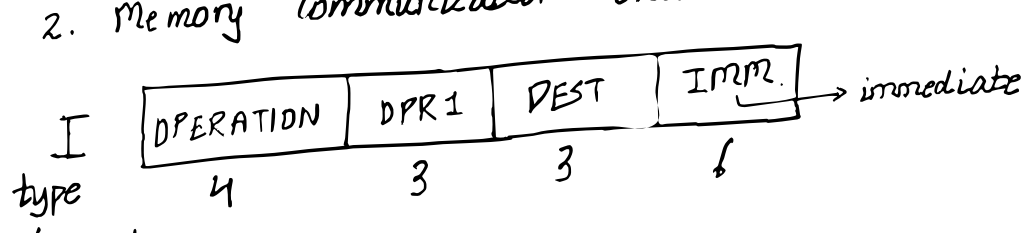
8 Registers
↳ (each 16 bit)



$r_1 = r_2 + r_3$

↳ add r_2, r_3, r_1

2. Memory Communication (Read / Write)



Read $r_2, r_1, 10$

$r_2 \leftarrow m(r_1 + 10)$

Base + Displacement notation

int A[100]

r_1 1000

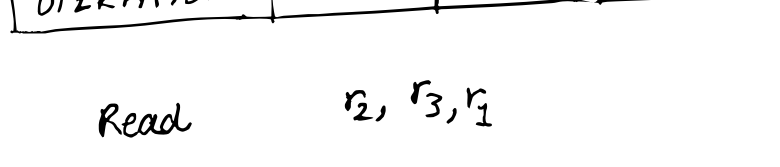
$(r_1)40$:
$r_1 + 10 \times 4$
$= r_1 + 40$
$= 1040$

Write $r_2, (r_1)10$

$m(r_1 + 10) \leftarrow r_2$

For Logical / Arithmetic:

Immediate:

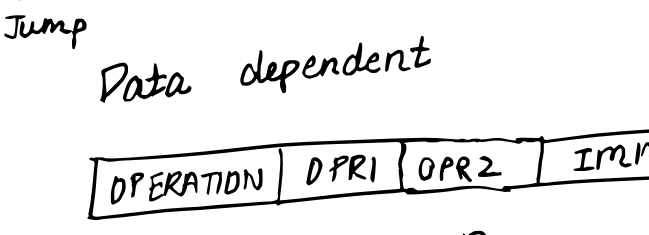


Read r_2, r_3, r_1

$r_1 \leftarrow m(r_2 + r_3)$

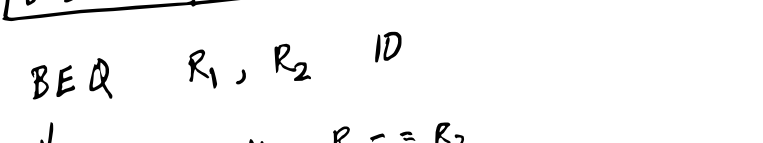
↓
Base + Index notation.

Change in control flow:
data independent (unconditional)



JUMP 40

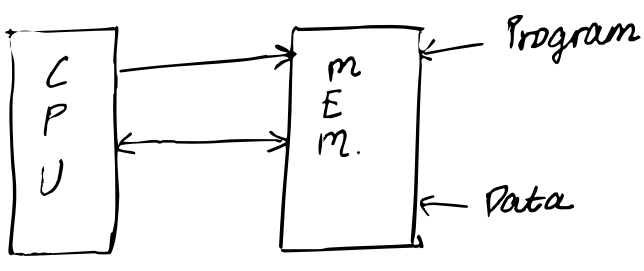
Data dependent



BEQ R_1, R_2 10

↓
Branch if equal

if $R_1 == R_2$
move 10 places.



Program → sequential composition of instructions $= \{ I_1, I_2, \ldots, I_n \}$ ← Instruction Pointer

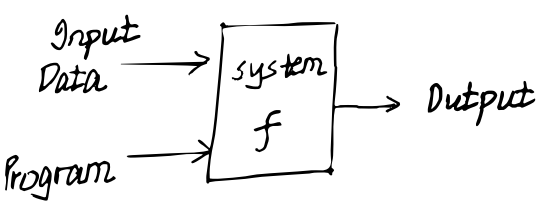
↓
Program Counter (misnomer)

I_1: $r_1 = r_2 + r_3$

↑
location - 1000
↳ Instruction Pointer

1. Bring instruction
2. Understand instruction
3. Read operands
4. Perform operation
5. Store result in destination
6. Update Instruction Pointer (IP)

Computing System:

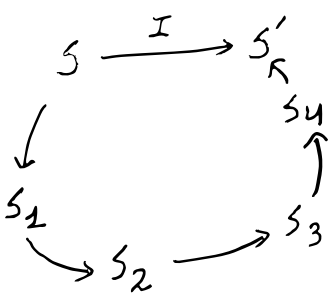


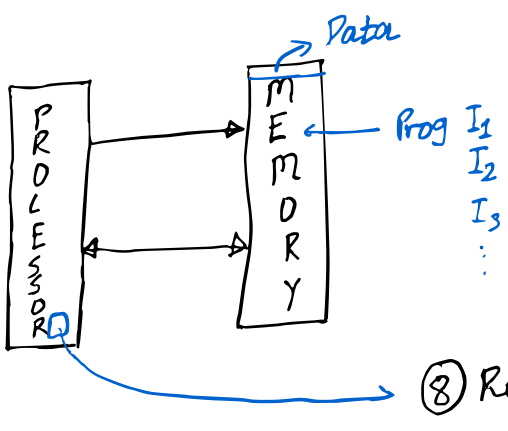
Output = f(Input, Program)

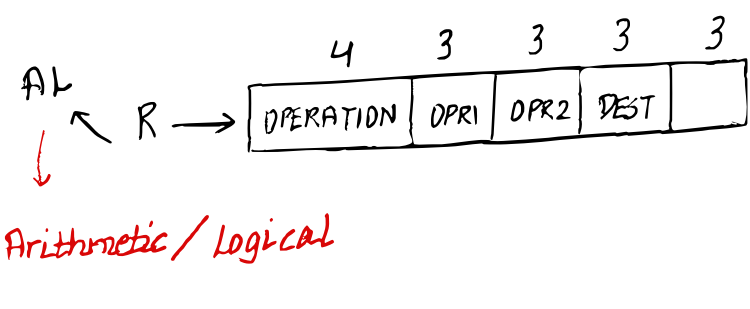
Instructions

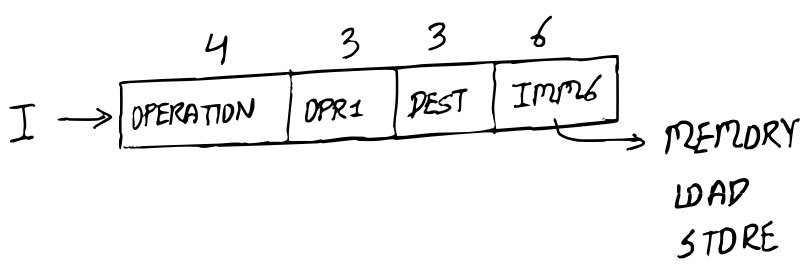
$P = \{I_1, I_2, \ldots, I_n\}$

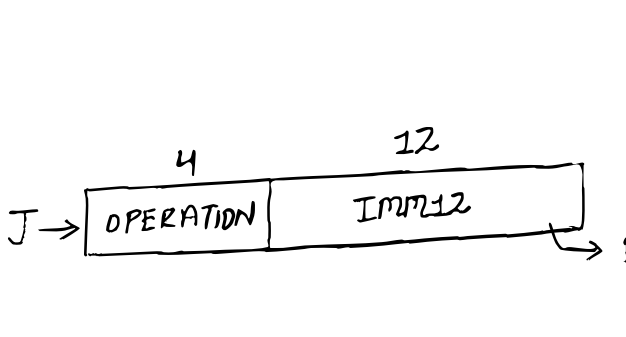
$S \xrightarrow{P} S'$



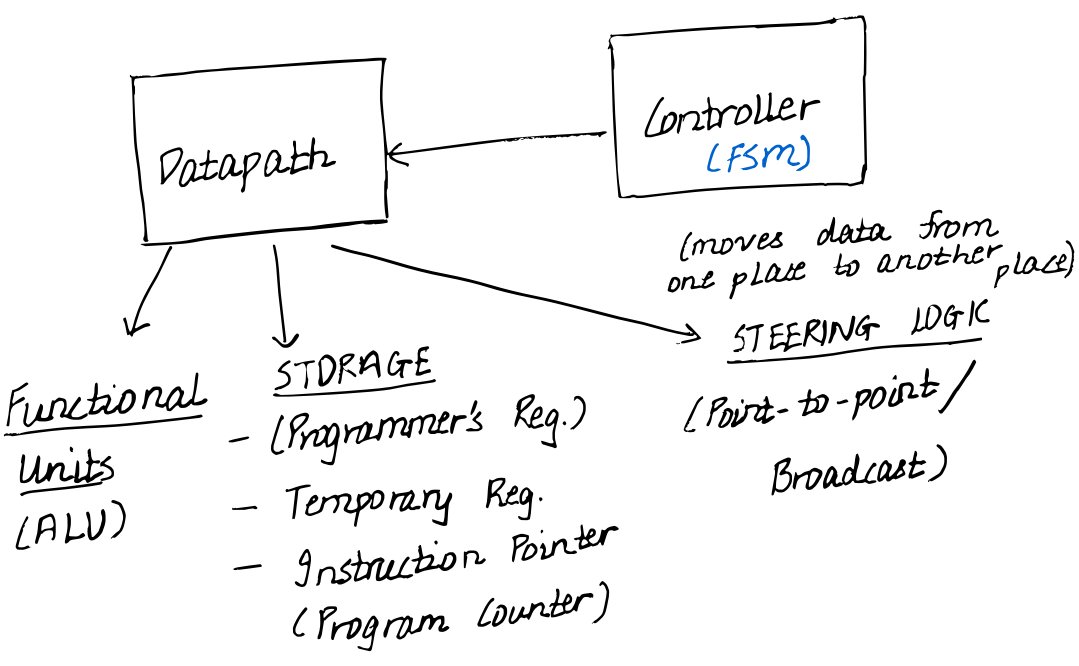


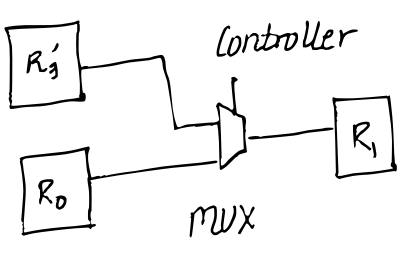






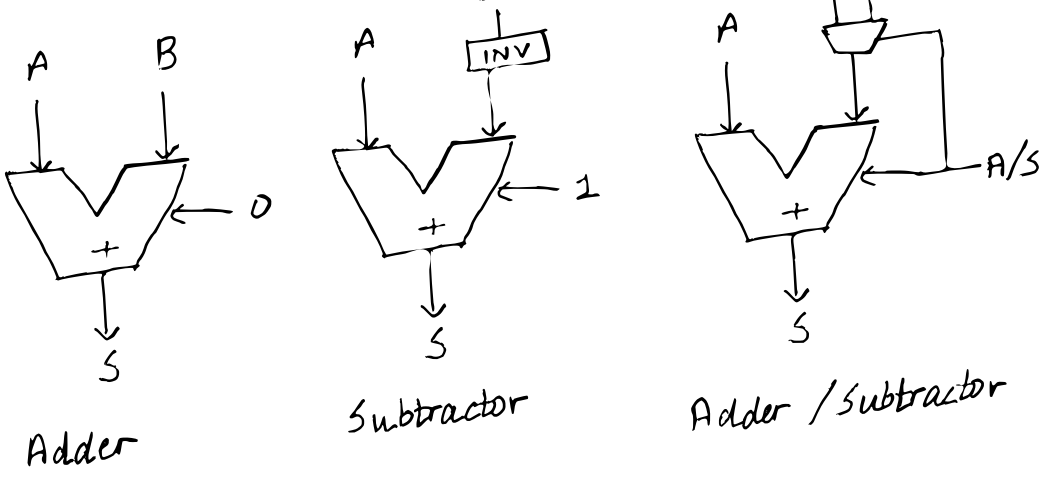
CPU (PROCESSOR):



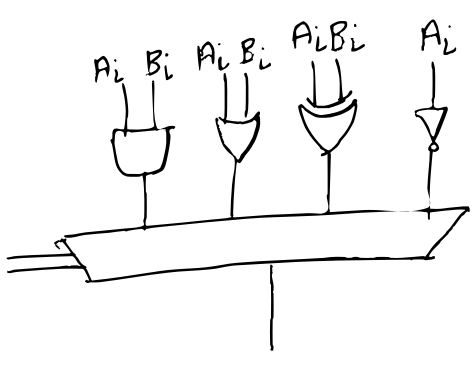


ALU

Arithmetic: (Adder / Subtractor)



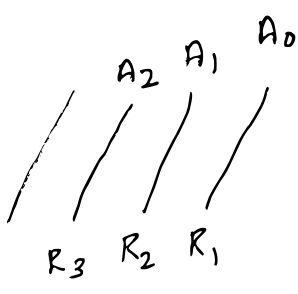
Logical: (4-bit input)



(Array of AND, OR, XOR, NOT)

each 4 input, so essentially four gates of each

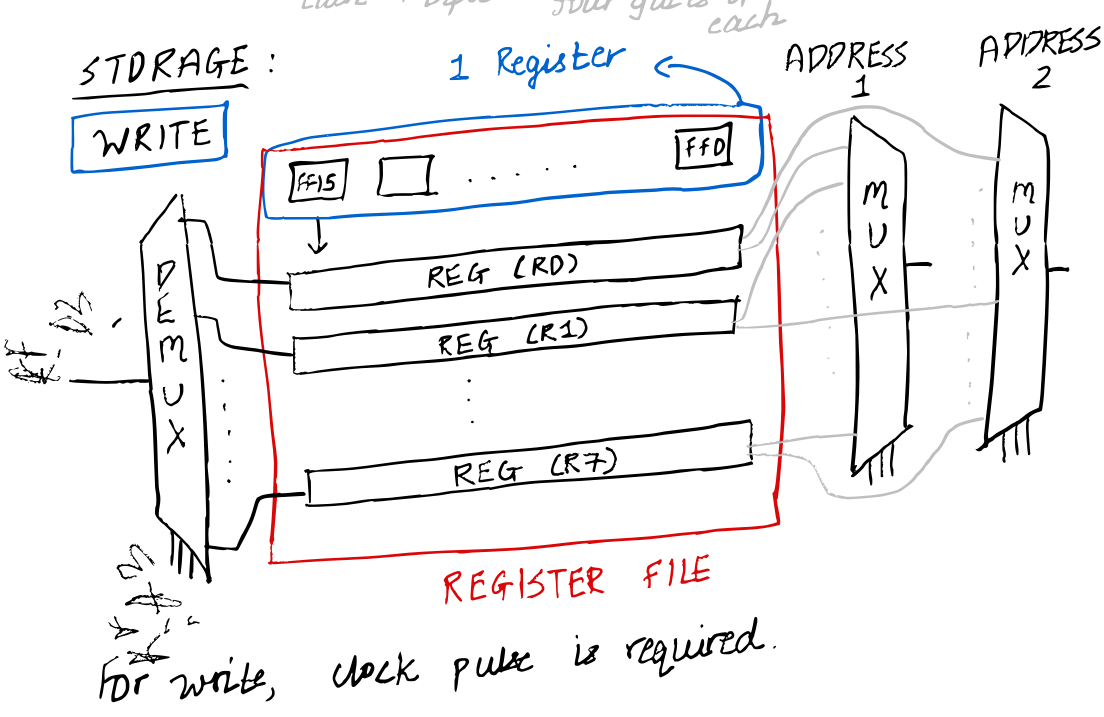
SHIFT



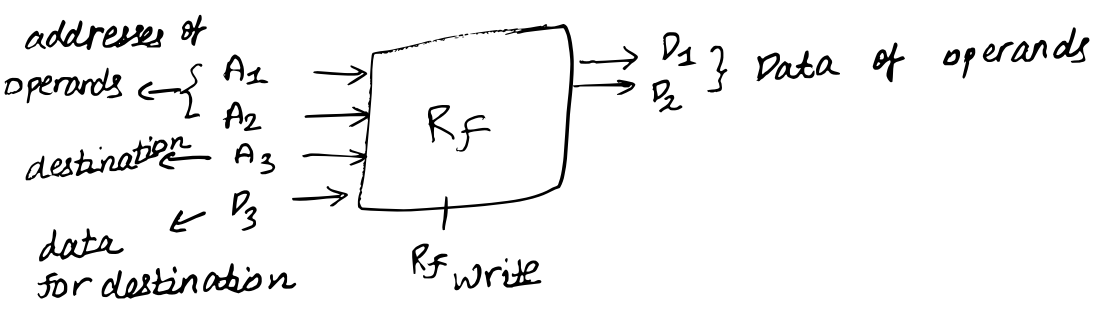
STORAGE:

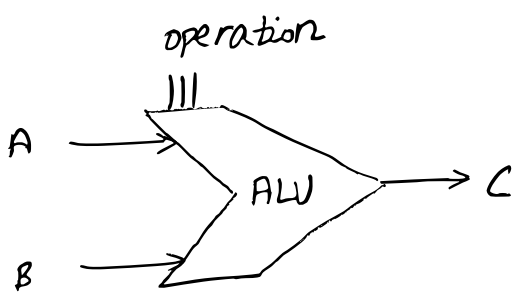
WRITE

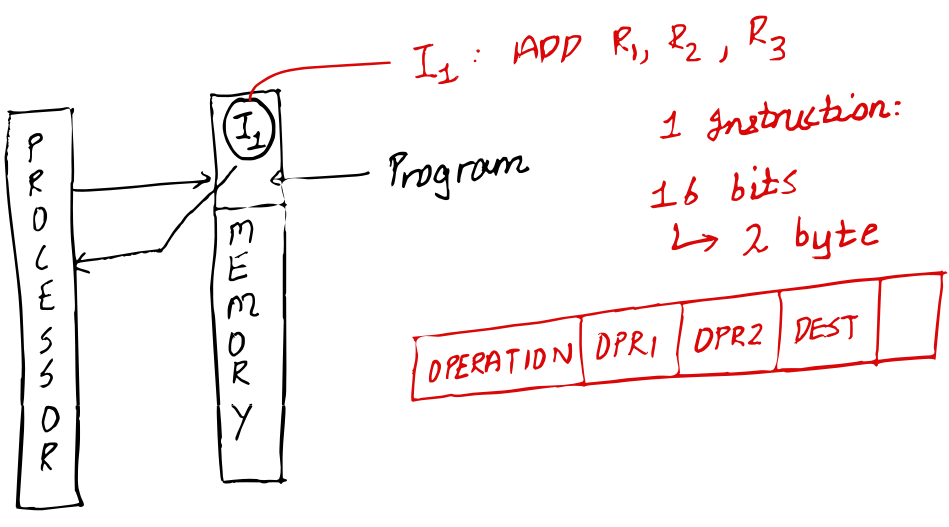
READ



For write, clock pulse is required.







→ Bring (Fetch)
→ Understand
→ Read
→ Execute
→ Write
→ Update IP

↓ compressed to ↓

→ Fetch + update IP
→ Understand and Read
→ Execute
→ Write

IP → Instruction Pointer
IR → Instruction Register
D → Data
A → Address

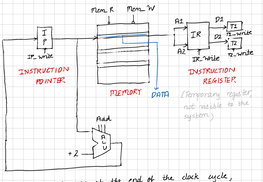
RF → Register File
T → Temporary register
R → Read
W → Write

Instruction Flow:
1) Bring (Fetch) instruction from memory
2) Understand instruction
3) Read operands
4) Execute instruction
5) Write result
6) Update IP

$S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5 \rightarrow S_1$

Optimize it!
1) Fetch instruction and update IP
2) Understand instruction and read operands
3) Execute instruction
4) Write result

$S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4$



All writes happen at the end of the clock cycle, but read can happen any time

ADD

ADD R_1 , R_2 , R_3

1) Fetch instruction & update IP
2) Understand and read operands
3) Compute
4) Write result

S1/
Fetch & Update

IP ⟶ Mem_A		Mem_R
Mem_D ⟶ IR		IR_W
IP ⟶ ALU_A		ADD
+2 ⟶ ALU_B		IP_W
ALU_C ⟶ IP		

S2/
Understand and read

IR_{4-11} ⟶ RF_A1		
IR_{8-11} ⟶ RF_A2		
RF_D1 ⟶ T1		T1_W
RF_D2 ⟶ T2		T2_W

S3/
Compute / Execute

T1 ⟶ ALU_A		
T2 ⟶ ALU_B		Add
ALU_C ⟶ T3		T3_W

S4/
Write Result

T3 ⟶ RF_D3		
IR_{7-5} ⟶ RF_A3		RF_W

For SUB, everything else is same except ALU set to SUB instead of ADD using IR_{6-10}

LOAD

LOAD R_2 , (R2) 10

$R_1 = M[R_2 + 10]$

1) Fetch instruction and update IP
2) Understand and read operands
3) Execute
4) Write result
5) Compute Address
6) Read Memory
7) Write result

S1/
Fetch and Update

IP ⟶ Mem_A		Mem_R
Mem_D ⟶ IR		IR_W
IP ⟶ ALU_A		
+2 ⟶ ALU_B		Add
ALU_C ⟶ IP		IP_W

S2/
Understand and read

IR_{8-11} ⟶ RF_A1		
RF_D1 ⟶ T1		T1_W

S3/
Compute Address

IR_{0-5} ⟶ SEE_in		
SEE_out ⟶ ALU_B		
T1 ⟶ ALU_A		Add
ALU_C ⟶ T3		T3_W

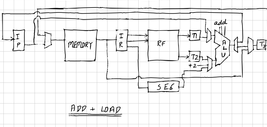
SEE ?
↓
ALU_B 16 bit but IR_{0-5} 6 bit extend to 16 bit by replicating MSB

S4/
Read Memory

T3 ⟶ Mem_A		Mem_R
Mem_D ⟶ T3		T3_W

S5/
Write Result

T3 ⟶ RF_D3		RF_W
IR_{2-9} ⟶ RF_A3		

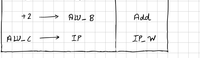


ADD + LOAD

STORE

STORE R_2 , (R_2) 10

$MEM[R_2 + 10] = R_1$



S1/
Fetch and update

IP ⟶ Mem_A		Mem_R
Mem_D ⟶ IR		IR_W
IP ⟶ ALU_A		
+2 ⟶ ALU_B		Add
ALU_C ⟶ IP		IP_W

S2/
Understand and Read

IR_{8-11} ⟶ RF_A1		
IR_{8-8} ⟶ RF_A2		
RF_D1 ⟶ T1		T1_W
RF_D2 ⟶ T2		T2_W

S3/
Compute Address

IR_{0-5} ⟶ SEE_in		
SEE_out ⟶ ALU_B		
T1 ⟶ ALU_A		Add
ALU_C ⟶ T3		T3_W

S4/
Write Result (Write to Address)

T3 ⟶ Mem_A		
T2 ⟶ Mem_D		Mem_W

26/10

BEQ R_1, R_2, ID

If $R_1 == R_2$ THEN

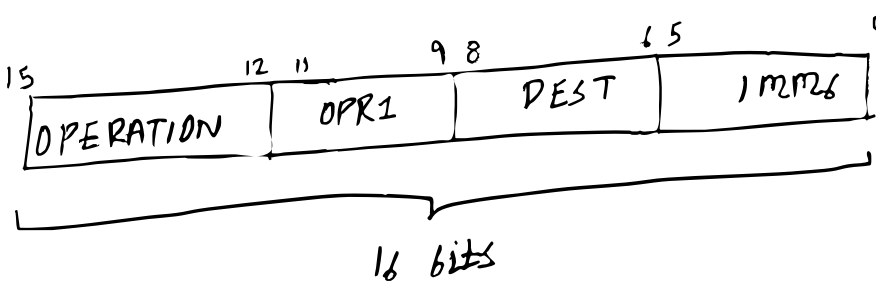
$IP = IP + 2 + ID \times 2$

↳ next instruction

→ IF $R_1 == R_2$ then

$IP = IP + 2 + Imm6 \times 2$

Why 2?
↳ 1 instruction = 2 byte



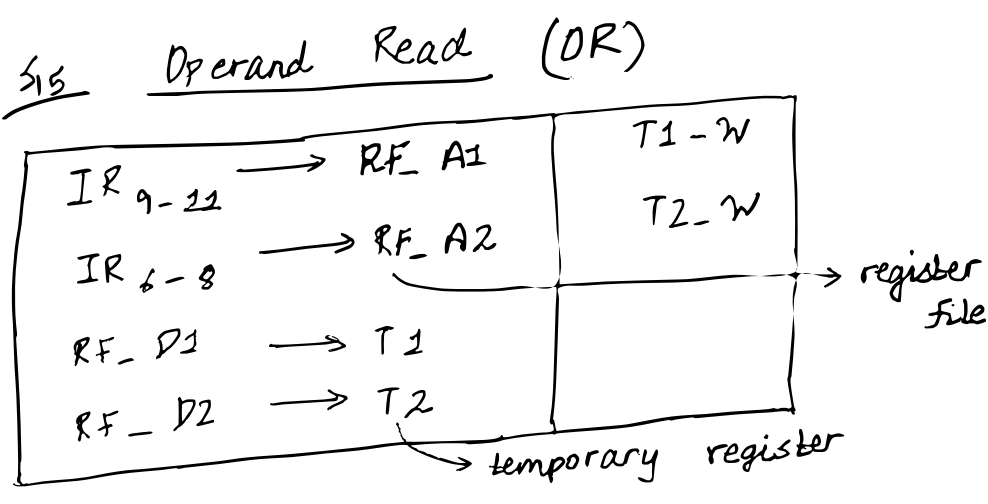
Imm6 → no. of instructions

① Fetch Instruction and update IP.
② Understand and Read Operands (R_1 & R_2)
③ Compute (Compare R_1 & R_2) If $(R_1 - R_2) == 0$ $Z = 1$
④ If $(Z == 1)$ THEN $IP = IP + Imm6 \times 2$

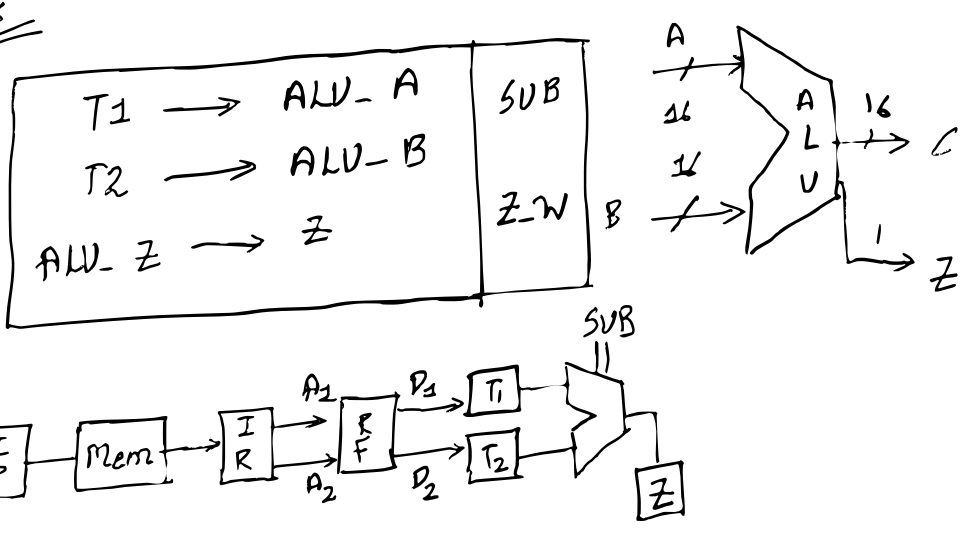
S14 Fetch

IP → Mem. Add	Mem-R
Mem. D → IR (→ inst. reg.)	Add
IP → ALU_A	IP_W
+2 → ALU_B	IR_W
ALU_C → IP	

S15 Operand Read (OR)



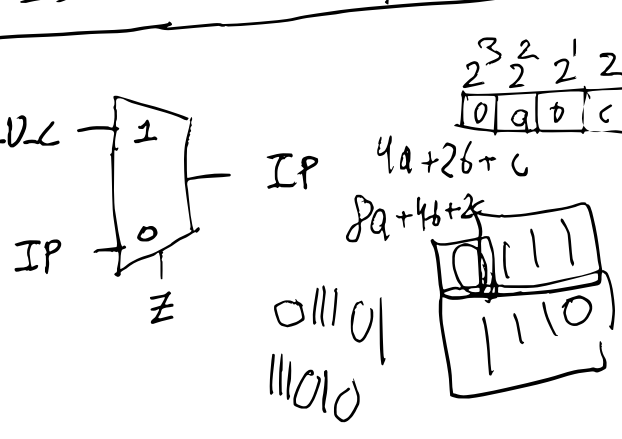
S16 Execute



S17 Update IP

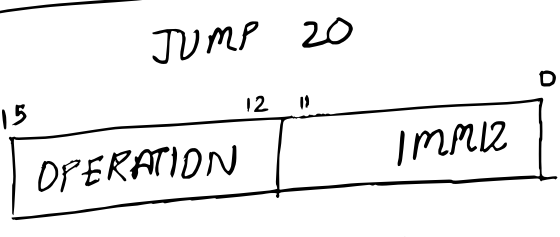
IP → ALU_A	ADD
Imm6 → SE6 → Shift (×2) → ALU_B	
IF (Z == 1) ALU_C → IP	IP_W
else IP → IP	

Extend to 16-bit



Jump Rel. loc :

JUMP 20



$IP = IP + 2 + IMM12 \times 2$ → Why 2? (Check above)

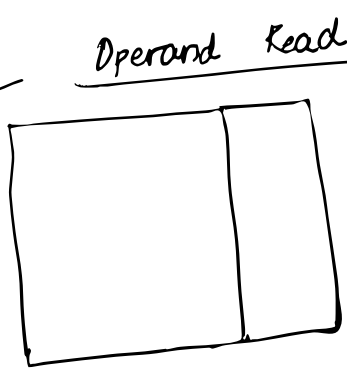
① Fetch & Update IP
② Understand & Read Operand
③ Compute IP & Update [Execute]

$IP = IP + 2 + 2 \times IMM12$

S18 Fetch

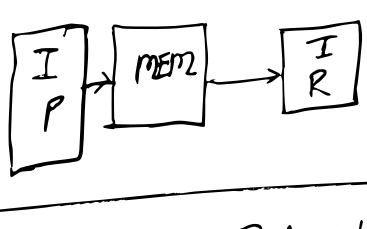
IP → Mem_Add	Mem_R
Mem_D → IR	Add
IP → ALU_A	IP_W
+2 → ALU_B	IR_W
ALU_C → IP	

S19 Operand Read (OR) → No operands



S20 Execute

IP → ALU_A	Add
IR_{0-11} → SE12 → Shift (×2) → ALU_B	IP_W
ALU_C → IP	



Summary: Datapath

Arithmetic & Logical
$S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4$

LOAD (memory READ)
$S_5 \rightarrow S_6 \rightarrow S_7 \rightarrow S_8 \rightarrow S_9$

STORE (memory WRITE)
$S_{10} \rightarrow S_{11} \rightarrow S_{12} \rightarrow S_{13}$

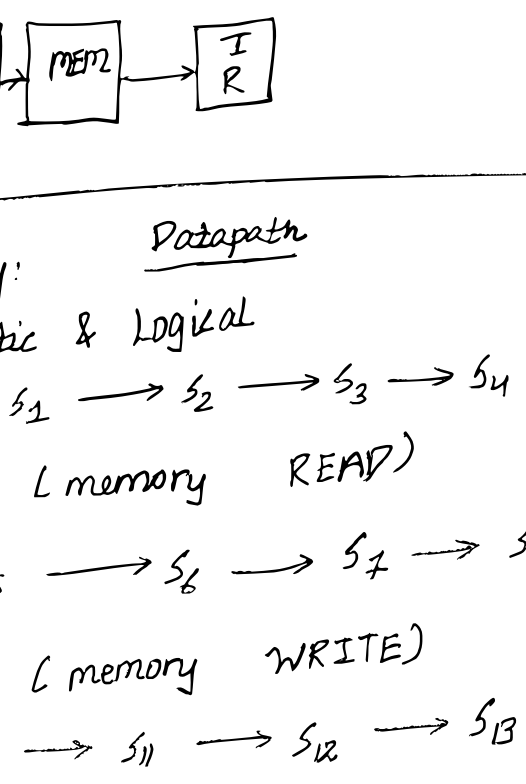
BRANCH
$S_{14} \rightarrow S_{15} \rightarrow S_{16} \rightarrow S_{17}$

JUMP
$S_{18} \rightarrow S_A \rightarrow S_{20}$

20 states

Now, merge all equivalent states

$S_1 \equiv S_5 \equiv S_{10} \equiv S_{14} \equiv S_{18}$ are equivalent provided next states are equivalent.
↳ other than (S_{19}, S_6), remaining states are same
↳ But even if they are equivalent, functionality will not change.



LOAD R_1, (R2) ID

$R_1 = M[R2 + ID]$

OPERATION	OPR1	DEST	IMM6
15 — 12	11 — 9	8 — 6	5 — 0
1000	010 (say)	001 (say)	101 D

Say Data at $R_2 = 500$

$\therefore R_1 = M[500 + ID]$

$= M[510]$ ← Data at memory location 510

R_2 stores value → can be address / can be data

STORE R_2, (R2) ID

$M[R_2 + ID] = R_1$

Data at Memory location $R_2 + 10$

ARITHMETIC & LOGIC :

$S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4$

LOAD :

$S_5 \rightarrow S_6 \rightarrow S_7 \rightarrow S_8 \rightarrow S_9$

STORE :

$S_{10} \rightarrow S_{11} \rightarrow S_{12} \rightarrow S_{13}$

BRANCH :

$S_{14} \rightarrow S_{15} \rightarrow S_{16} \rightarrow S_{17}$

JUMP :

$S_{18} \rightarrow S_{19} \rightarrow S_{20}$

$S_2 \equiv S_5 \equiv S_{10} \equiv S_{14} \equiv S_{18}$

↓

IF → Mem_A
Mem_D → IR
IF → ALU_A
+2 → ALU_B
ALU_C → IF

extra useless stuff being done

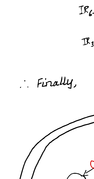
$S_2 \equiv S_{11} \equiv S_{15} \equiv S_6 \equiv S_9$

↓

IR_{9-11} → RF_A1
IR_{6-8} → RF_A2
RF_D1 → T1
RF_D2 → T2

OPERATION ← DEC 1 ← IR_{12-15} → All states

With the help of decoder 1 we can disconnect state transition from S_2 onwards. Thus making them equivalent



$S_7 \equiv S_{12}$ → Potential candidates to be equivalent

T_3 → ALU_A
IR_{0-5} → SEL → ALU_B
ALU_C → T3

ENCODING 4 :
1000 LOAD
1001 STORE

IR_{12} → DEC 2 → Address

S_4

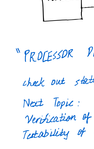
T_3 → RF_D3
IF_{3-5} → RF_A3

S_9

T_3 → RF_D3
IR_{1-8} → RF_A3

If I say IR_{15} i.e MSB = 0 always represents arithmetic and logical operations, we can use :

∴ Finally,



Coming to each component,

IF, T3 → MEM_A

→ MEM_A ?
IF → MEM_A
T3 → MEM_A

ALU :

→ ALU_A ?
T_1 → ALU_A
IF → ALU_A

→ ALU_B ?
+2 → ALU_B
T2 → ALU_B
IR_{0-5} → SEL → ALU_B
IR_{9-11} → SEL → Shift → ALU_B

But what instruction should ALU run?

ADD ?
SUB ?
IR_{12-15} ?
→ 4x1 MUX

∴ δ defined by state diagram.

↳ state transition function

e.g. when S1 ⇒ next_state ⇐ S2

when S2 ⇒ next_state ⇐ DEC1

What about λ ? → (output function)

↳ defined by control signals

↓

When S1 :
When S2 :
When S3 :

p
q ⇐ +
r ⇐ -
s ⇐
t ⇐
u ⇐

IR_W ⇐
RF_W ⇐
Mem_R ⇐
Mem_W ⇐
T1_W ⇐
T2_W ⇐
T3_W ⇐

Need to specify all for each state (because λ is a combinational circuit) → ref. VHDL

"PROCESSOR DOES NOT HAVE MEMORY"

check out status register / flag register design

Next Topic :
Verification of state machine
Testability of state machine
→ for synchronous sequential circuits

State Machine:

$M = (S, S_0, I, O, \delta, \lambda)$

VERIFICATION: Check Implementation = Specification

$D_1 \overset{?}{\equiv} D_2$

EQUIVALENCE CHECKING

$M_1 \overset{?}{\equiv} M_2$

$M_1 = (S, S_0, I, O, \delta, \lambda)$ say $|S| = n_1$

$M_2 = (S', S_0', I, O, \delta', \lambda')$ say $|S'| = n_2$

Have to be same for equivalence

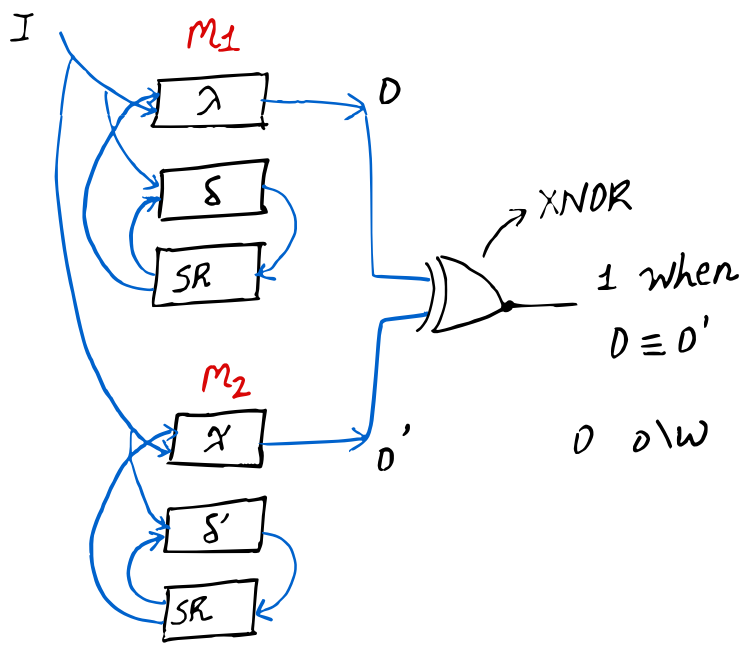
$M_1 \overset{?}{\equiv} M_2$

→ Optimized $SM \rightarrow M_1'$

→ Optimized $SM \rightarrow M_2'$

If M_1' & M_2' are isomorphic, then they are equivalent.

BUT, the converse is not true.



$M = (S'', S_0'', I, O, \delta'', \lambda'')$

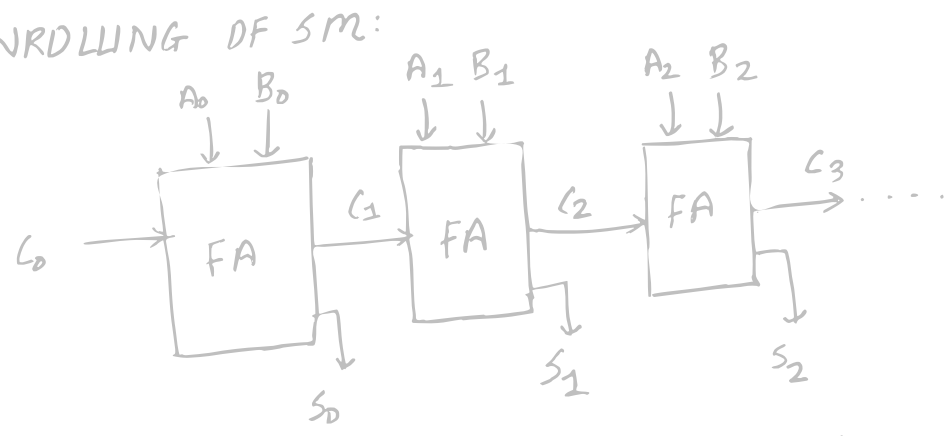
$|S''| = |S| \times |S'| = n_1 \times n_2$

PRODUCT MACHINE

$S'' = S \times S'$

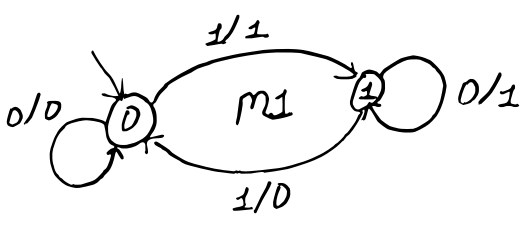
$S_0'' = (S_0, S_0')$

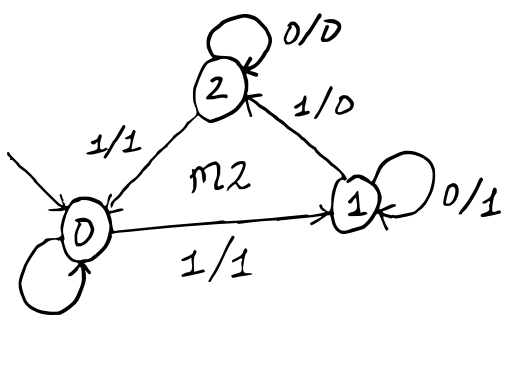
UNROLLING OF SM:



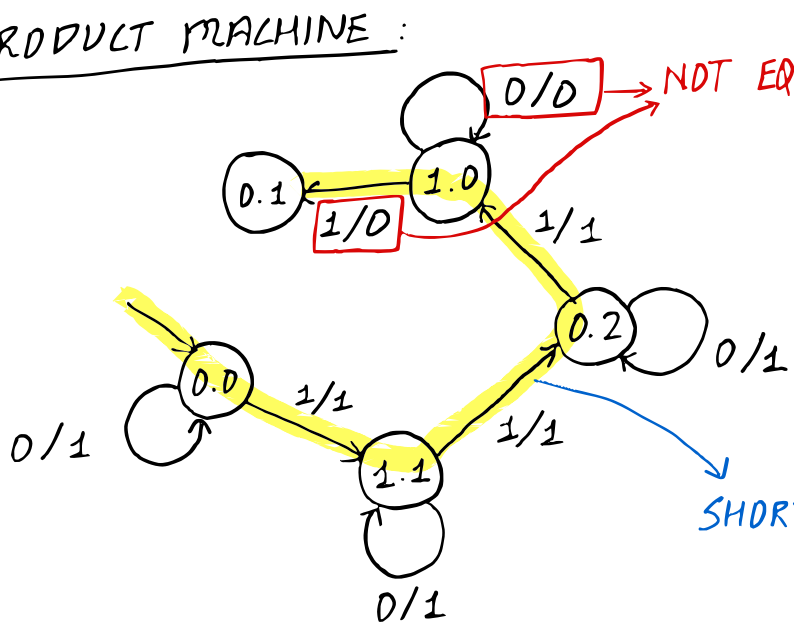
If for an infinite length sequence, S_i's come equal, then they are equivalent.

NOT COMPLETELY CLEAR





PRODUCT MACHINE:



THUS, $M1 \not\equiv M2$

OR

THERE ARE NO EXTRA REACHABLE STATES i.e. the loop becomes complete with all outputs 1)

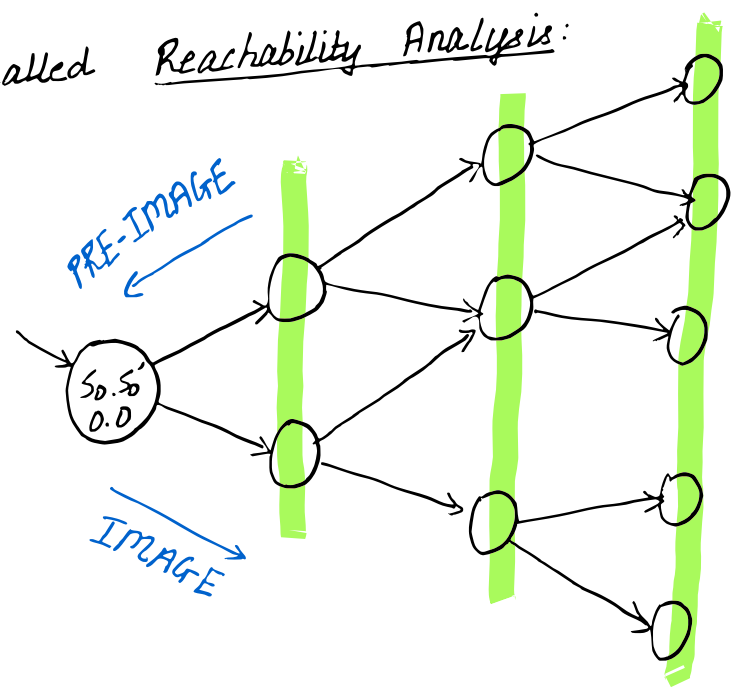
$(M_1 \equiv M_2)$

TERMINATION CONDITION:

→ No new reachable state is discovered (EQUIVALENT)

→ Output is zero (NOT EQUIVALENT)

Also called Reachability Analysis:



$M^{PM} = (S'', S_0'', I, O', \delta'', \lambda'')$

Set of states?

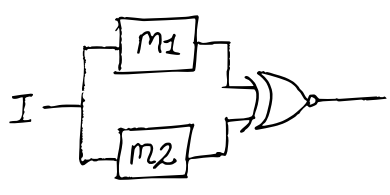
Functions?

Midsem Solutions:

Did not note down.

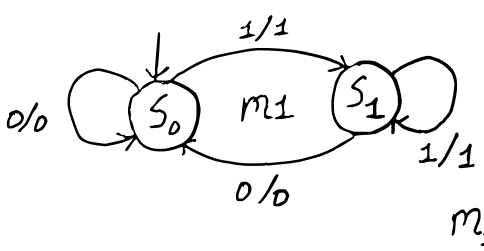
Verification :

Product Machine

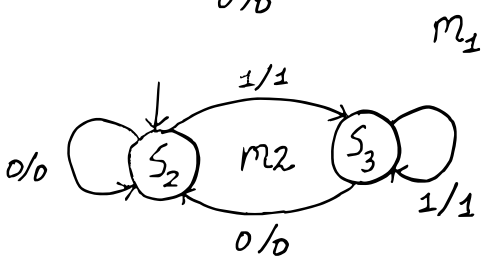


1, when $\lambda_1 \equiv \lambda_2$

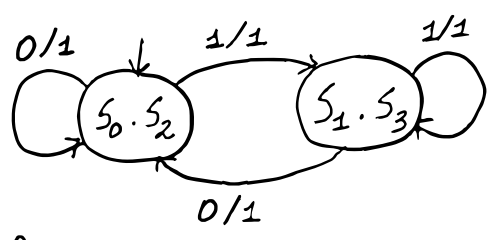
0, o\w



$M_1 \overset{?}{\equiv} M_2$



Product Machine:

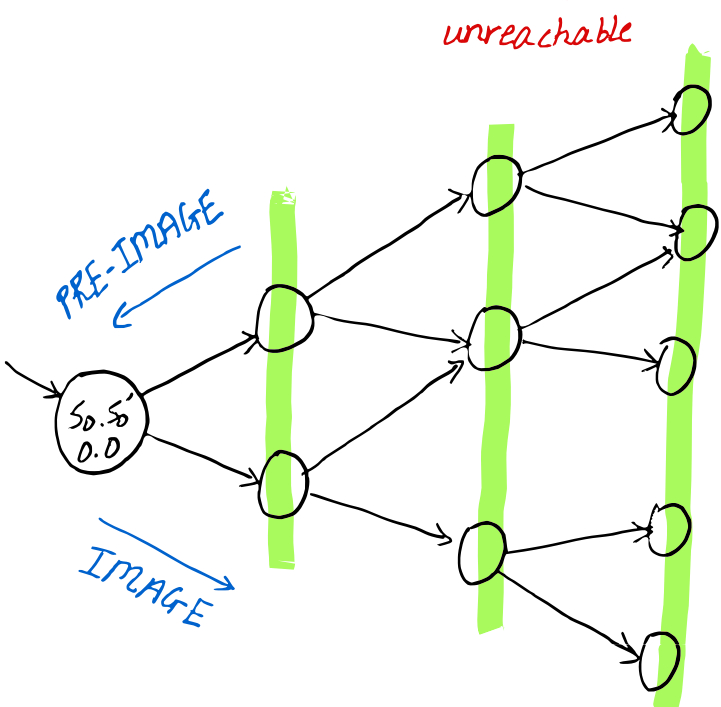


$\lambda_{pm} = 1 \rightarrow$ always

hence $M_1 \equiv M_2$!

Here $(S_0 \cdot S_2)$ and $(S_1 \cdot S_3)$ are reachable states

But $(S_1 \cdot S_2)$ and $(S_0 \cdot S_3)$ are unreachable



$L(S) \rightarrow$ set of reachable states at any given time.

$\delta(S_i) \rightarrow S_j$

Transition Relation $\rightarrow R(x, s, t)$

- x: input
- s: value of current state variables
- t: value of next state variables

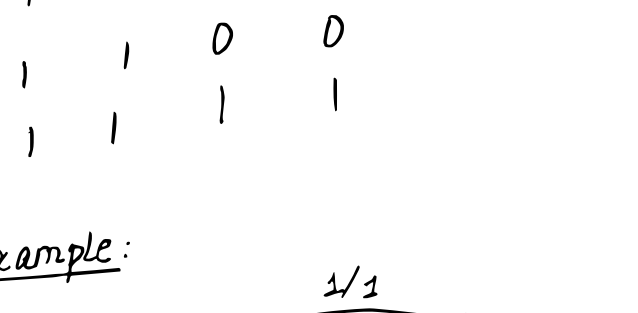
x	s_i	t_i	R
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Transition Relation $\rightarrow R(x, s, t)$

$\rightarrow C(s)\, R(x, s, t)$

at a given state at a given time

example:



x : current state variables

s : next state variables

t : next state variables

transition relation $\leftarrow R(x, s, t) \equiv (t \equiv \delta(x, s)) \equiv (t \odot \delta(x, s))$ $\rightarrow$ transition function

$\rightarrow$ XNOR (equivalence checking)

$\delta(x, s)$:

s \ x	0	1
0	0	1
1	1	0

$\therefore \delta(x, s) = x\bar{s} + \bar{x}s = x \oplus s$

$\therefore t \equiv x \oplus s$

$\therefore R(x, s, t) = t \odot (x \oplus s)$

$= t(\bar{x}s + x\bar{s}) + \bar{t}(\bar{x}\bar{s} + xs)$

$\therefore \boxed{R(x, s, t) = s\bar{x}t + \bar{s}xt + \bar{s}\bar{x}\bar{t} + sx\bar{t}}$

$\rightarrow$ TRANSITION RELATION

s	
S_0	0 $\rightarrow \bar{s}$
S_1	1 $\rightarrow s$

Reachability Analysis:

$C(s)\ R(x, s, t)$

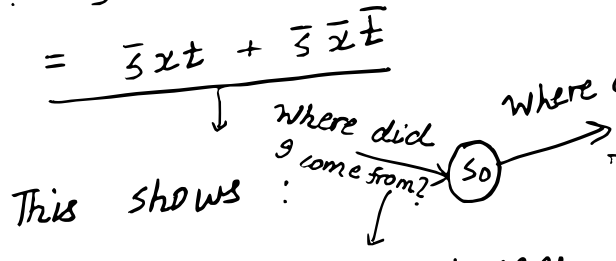
For S_0:

$C(s) = \bar{s}$

$\therefore \bar{s}(s\bar{x}t + \bar{s}xt + \bar{s}\bar{x}\bar{t} + sx\bar{t})$

$= \bar{s}xt + \bar{s}\bar{x}\bar{t}$

This shows :



For pre-images, eliminate x, t

For images, eliminate x, s

How to eliminate ?

Recall Shannon's Decomposition :

$f(x_1, \ldots, x_n) = x_i f_{x_i} + \bar{x}_i f_{\bar{x}_i}$

f_{x_i} : independent of x_i

$f_{\bar{x}_i}$: independent of x_i

existential quantification of x

$\therefore \exists_x f(x_1, \ldots, x_n) = f_{x_i} + f_{\bar{x}_i}$

(there exists x :)

For S_0,

$F(x, s, t) = \bar{s}xt + \bar{s}\bar{x}\bar{t}$

$\exists_x F(x, s, t) = F_x + F_{\bar{x}}$

$F(s, t) = st + \bar{s}\bar{t}$

$\exists_s F(s, t) = F_s + F_{\bar{s}}$

$= 0 + t + \bar{t} = t + \bar{t}$

$F(t) = 1$ (always true)

$\rightarrow$ Image $\rightarrow$ Both are reachable (S_0, S_1)

$\exists_t F(s, t) = F_t + F_{\bar{t}}$

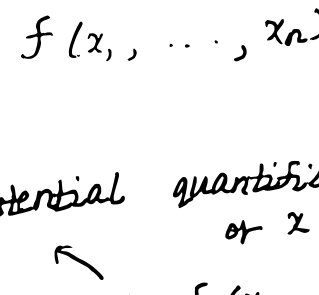
$= \bar{s} + s$

$\therefore F(s) = \bar{s}$?

$\rightarrow$ It is starting from S_0

$\rightarrow$ hasn't reached S_1 yet

i.e. $C(s) = S_0 = \bar{s}$



Initially, $C(s) = S_0 = \bar{s}$

next time, $C(s) = S_0 \cup S_1 = \bar{s} + s = 1$

For all subsequent time, $C(s) = S_0 \cup S_1 = 1$

as all states have been reached.

SEQUENTIAL CIRCUIT VERIFICATION:

→ Equivalence Checking

$M_1 \overset{?}{\equiv} M_2$

↓

Product Machine (PM)

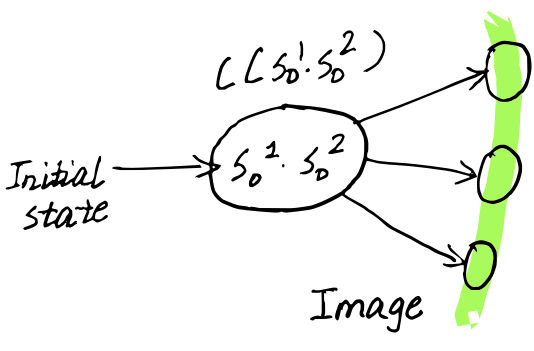
$|S_1| \quad |S_2|$

↓

$|S_1| \times |S_2|$

$M = M_1 \times M_2$

$= (S^1 \times S^2, S_0^1 \times S_0^2, I, O, \delta, \lambda)$



Transition Relationship:

$t^1 \equiv \delta^1(s^1, x)$

$t^2 \equiv \delta^2(s^2, x)$

$\} \; \Pi(t_i \equiv \delta_i(s_i, x))$

$= (t_1 \equiv \delta^1(s^1, x))(t_2 \equiv \delta^2(s^2, x))$

t^1: next state variable; s^1: current state variable; x: set of inputs

e.g.

↳ $M_1 \rightarrow$ 2 states $\rightarrow S_0$

$M_2 \rightarrow$ 4 states $\rightarrow S_2 S_1$

} state variables

two transition functions

∴ Transition relationship:

$(t_0 \equiv \delta_0^1(S_0, x))(t_1 \equiv \delta_1^2(S_2, S_1, x))(t_2 \equiv \delta_2^2(S_2, S_1, x))$

$\equiv \rightarrow \odot$

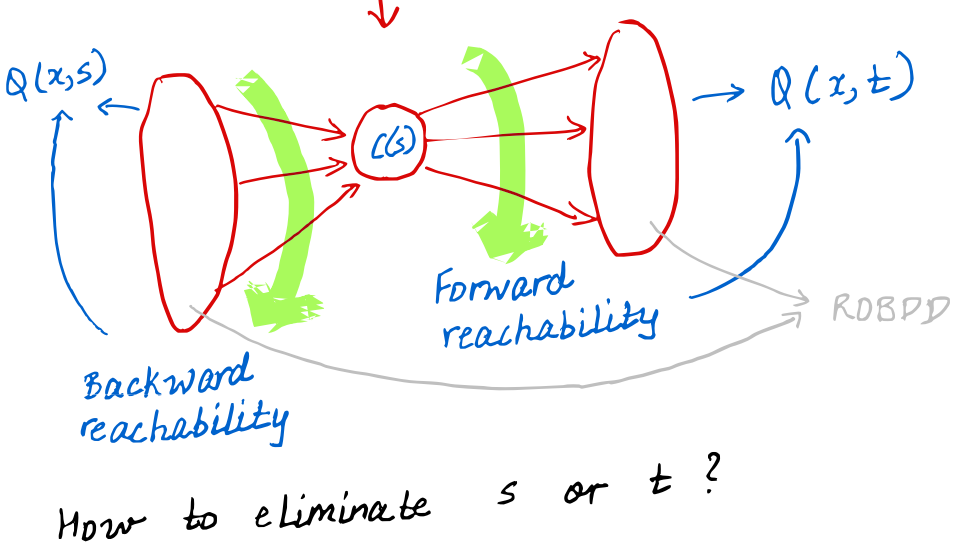
XNOR

Transition Relation $R(x, s, t)$

$C(s) \; R(x, s, t) = Q(x, s, t)$

$C(s)$: set of initial states → initially reachable states

$R(x, s, t)$: Transition relation → all valid transitions (generic) → reachable



How to eliminate s or t?

Existential quantification

$f(x_1, x_2, \ldots, x_n) = x_i f_{x_i} + \bar{x_i} f_{\bar{x_i}}$

$\exists_{x_i} f(x_1, x_2, \cdots, x_n) = f_{x_i} + f_{\bar{x_i}}$

For above example, $Q(x, s_0, s_1, s_2, t_0, t_1, t_2) \rightarrow \exists_{s_2} \exists_{s_1} \exists_{s_0} Q \rightarrow Q'(x, t_0, t_1, t_2) \rightarrow \exists_x Q' \rightarrow Q''(t_0, t_1, t_2)$

$Q(x, s, t) \rightarrow \exists_s Q(x, s, t) = Q'(x, t)$

↓

$\exists_x Q'(x, t) = Q''(t)$

$C'(s) = C(s) + Q''(t)$ ↳ replace variable t with s

↑ New set of reachable states

Now, check $\lambda_1 \equiv \lambda_2$ ↳ if '0' → stop → Not equivalent

else if $C'(s) = C(s)$ → stop → equivalent

else continue

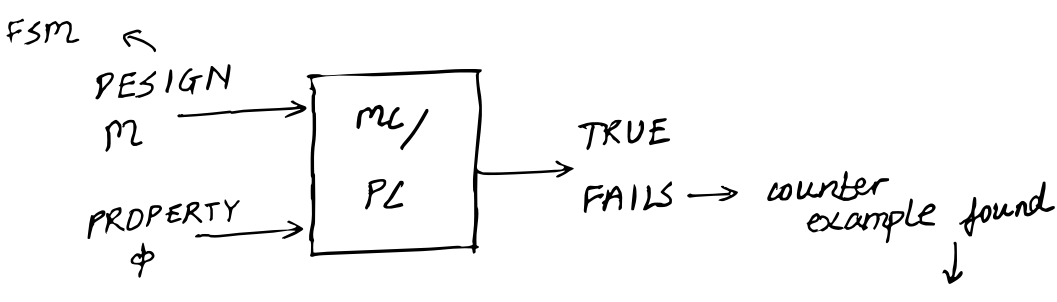
→ Property Checking

Model Checking

↳ Fully Automatic

2008 → Turing Award given to founders of Model Checking Clark & Emerson

B. Mishra (1998) found bug in FIFO implementation in "Bible of VLSI" by Mead & Conway



Is property always satisfied? $M, s_0 \overset{?}{\models} \phi$

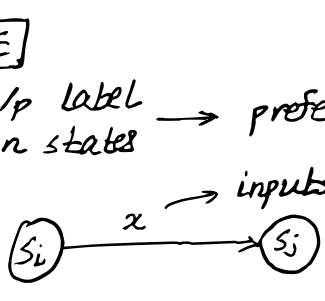
MOORE

↳ o/p label on states → preferred

→ inputs on edges, but we need to transform them to states

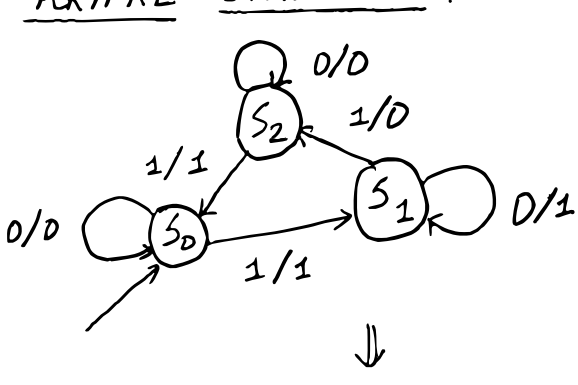
↳ so that edges contain no information

i.e. states contain all information.

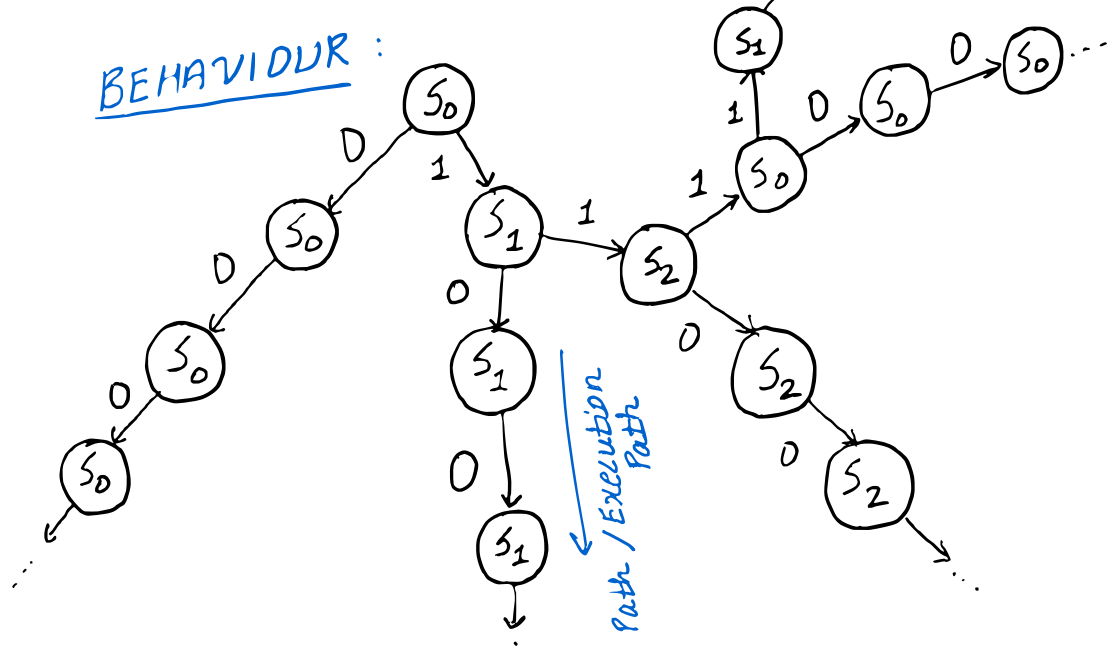


How to do this?

KRIPKE STRUCTURE:



BEHAVIOUR:



Infinite behaviours each of infinite length.

↳ sequence of states based on sequence of inputs

i.e. Infinite infinitely long behaviours

Property:

→ SAFETY

Desirable behaviour should always occur

undesirable behaviour should never occur

→ LIVENESS

Desirable behaviour would eventually be achieved

→ FAIRNESS

Desirable behaviour should be achieved repeatedly

Require Temporal operators alongside Static operators to specify behaviour (property)

G (global) F (future) N (Next) U (until)

For a request req,

F grant

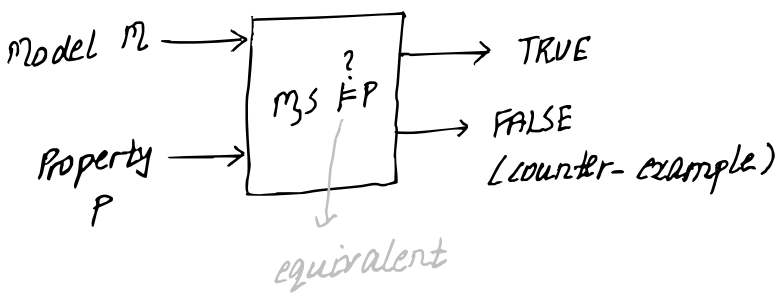
⇓

in the future, request is granted.

? ? ?

Property Checking:

(Model Checking)



SPECIFICATION $\overset{?}{\equiv}$ IMPLEMENTATION

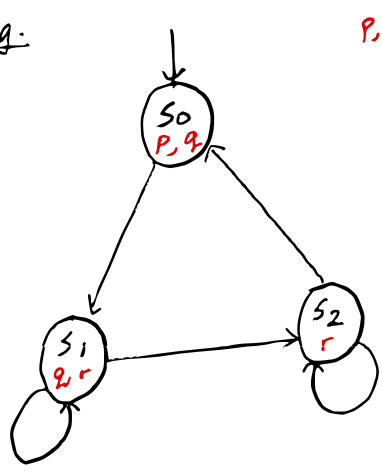
SPECIFICATION ↑ Property
- Safety
- Liveness
- Fairness

IMPLEMENTATION ↑ Model (M) FSM ↑ KRIPKE STRUCTURE (everything defined in states)

Behaviour → (in a property)
- static → Boolean operators (NOT, AND, OR)
- dynamic → Temporal Behaviour → X, F, G, U (U ↑ UNTIL)

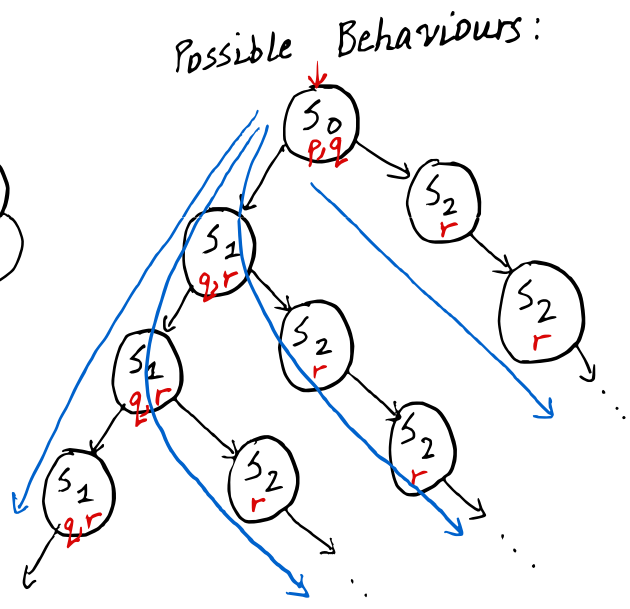
X ↑ NEXT, F ↑ FUTURE, G ↑ GLOBAL

e.g.:



p, q, r → variables
↳ some inputs, some outputs

Possible Behaviours:



In S_0 → p, q true
S_1 → q, r true
S_2 → r true

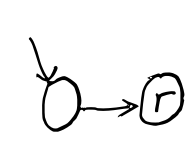
——— → Execution paths

Xr → next state, r should be true

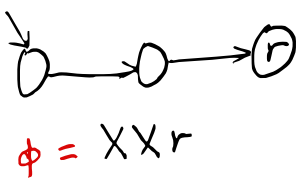
XXr → next-to-next state, r should be true

F.grant → sometime in the future, grant

req → F.grant → sometime in the future, grant req.

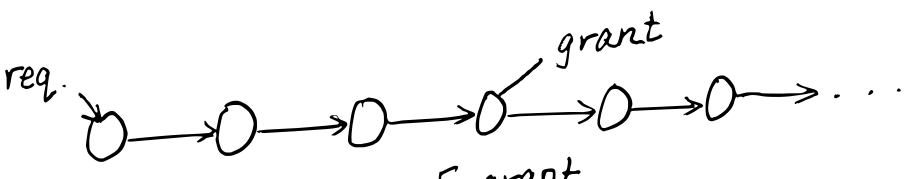


ϕ = Xr



ϕ = XXr

ϕ → formula

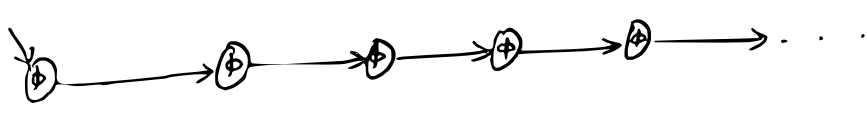


ϕ = req → F. grant

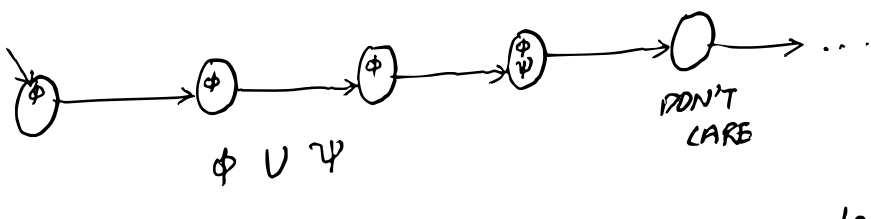
Gϕ → formula 'ϕ' holds good for all states

ϕ U ψ → formula 'ϕ' holds good until 'ψ' is asserted, after that, DON'T CARE.

[english until → after 't', ϕ does not happen
logical until → after 't', don't care about ϕ]



Gϕ



ϕ U ψ

If behaviour definition uses temporal operators, it's called Temporal Logic.
- Linear Temporal Logic (LTL) [along one execution path]
- Computation Tree Logic (CTL) [possible paths from a state]

Path Modalities
- A → for all
- E → there exists a path

Now, operators:

X	AX	EX
F	AF	EF
G	AG	EG
U	AU	EU

example:

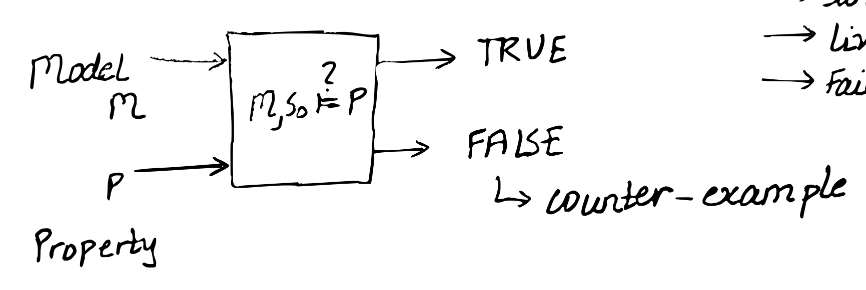
A (req → F. grant)

In all paths, request should be granted sometime in the future.

english: I studied (p) until I turned 30 (q)

logic: A[(p U q) · (q → Gp̄)]

Model Checking:



Properties:
→ Safety
→ Liveness
→ Fairness

State transition system → states are labelled by formula

Temporal logic → LTL / CTL

Boolean operators : ~, ·, +

Temporal operators : X, F, G, U
(◇) (□)

Path Modality : A, E
(∀) (∃)

} AX EX, AF EF, AG EG, AU EU

Do we need all 8?

$EX\phi \equiv \sim AX(\sim\phi)$

$EF\phi \equiv \sim AF(\sim\phi) \equiv \sim AG(\sim\phi)$

$EG\phi \equiv \sim AG(\sim\phi) \equiv \sim AF(\sim\phi)$

$E(\lambda U\phi) \equiv \sim A \sim (\lambda U \phi)$

$AF\phi = A[T U \phi]$ $EF\phi = E[TU\phi]$
↓
Tautology ?
(always true :
not expecting anything)
until ϕ, no expectations

$AG\phi \equiv \sim EF\sim\phi \overset{?}{\equiv} \sim E[TU\sim\phi]$

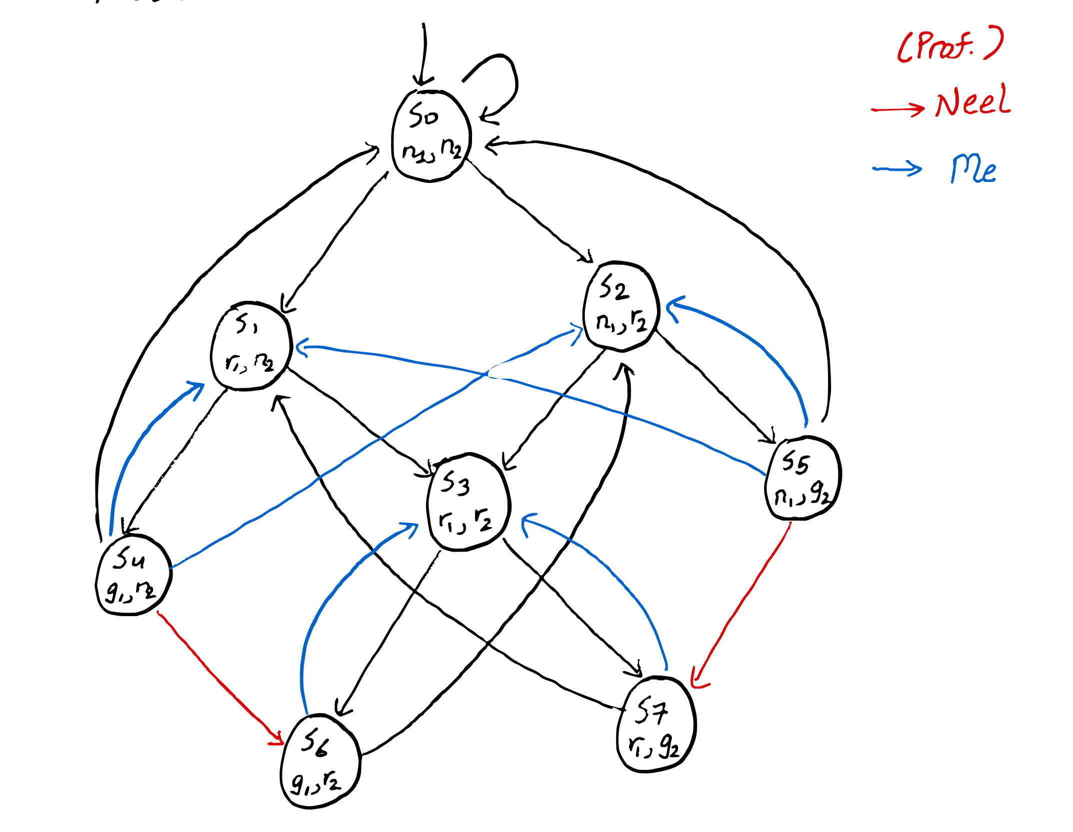
$EG\phi \equiv \sim AF\sim\phi \overset{?}{\equiv} \sim A[TU\sim\phi]$

EVERY OPERATOR MUST HAVE A PATH MODALITY

e.g. Common Resource :
Access of the common resource should be given to only one person at a time.
variables : request (r), grant (g), no request (n)
(access request) (of request) (absence of request)

For user$_{1,2}$: n_1, r_1, g_1
n_2, r_2, g_2

Model M :



S_0 : $n_1, n_2 = 1$
$r_1, r_2, g_1, g_2 = 0$
No. req → remain at S_0

Properties :

1] Safety : $AG(\overline{g_1 \cdot g_2})$

2] Liveness : $AG(r_1 \rightarrow AF g_1) \cdot AG(r_2 \rightarrow AF g_2)$?

3] Non-blocking:
User should be allowed to make a request anytime it is not in r_1 / r_2.
$AG(n_1 \rightarrow EX r_1) \cdot AG(n_2 \rightarrow EX r_2)$
If no request, then in some path, request made in next state.
i.e. There should always be a path to raise request when no request.
(i.e. absence of request)
But not all.
↳ next state no request also possible

X → special case of F
$n_1 \rightarrow EX r_1$ more strict than $n_1 \rightarrow EF r_1$

Verifying the properties :

1] $\overline{g_1 g_2} = \phi_1$ → labelled in all states (true)
property 1 is satisfied.

2] $r_1 \rightarrow AF g_1$
$S_1 \rightarrow S_4$ ✓
$S_1 \rightarrow S_3 \rightarrow S_6$ ✓
But,
↓
$S_1 \rightarrow S_3 \rightarrow S_7 \rightarrow S_1 \rightarrow S_3 \rightarrow S_7 \rightarrow \dots$
X
Fails

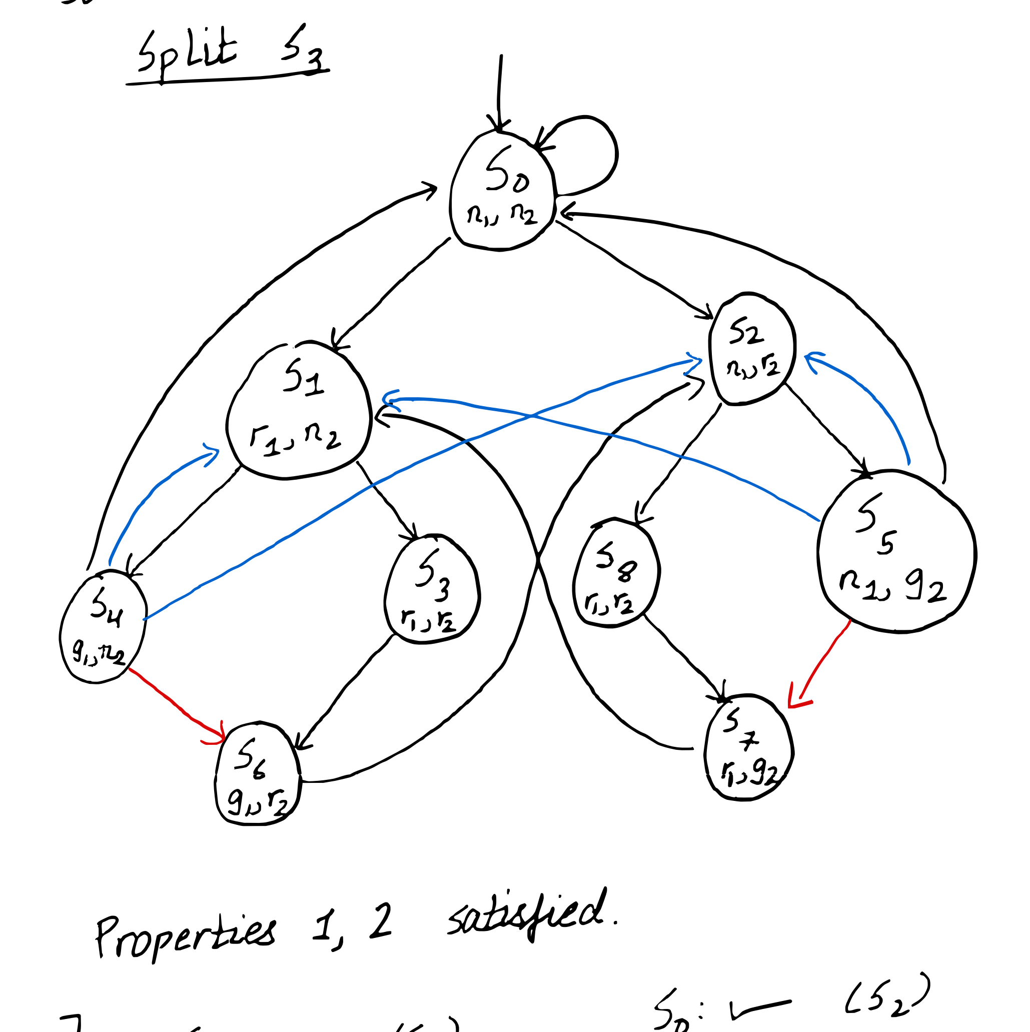
$r_2 \rightarrow AF g_2$
$S_2 \rightarrow S_5$ ✓
$S_2 \rightarrow S_3 \rightarrow S_7$ ✓
But,
$S_2 \rightarrow S_3 \rightarrow S_6 \rightarrow S_2 \rightarrow S_3 \rightarrow S_6 \dots$
X
Fails

property 2 fails

Counter-examples :
$S_0 \rightarrow S_1 \rightarrow S_3 \rightarrow S_7 \rightarrow S_1 \rightarrow S_3 \rightarrow S_7 \rightarrow \dots$
$S_0 \rightarrow S_2 \rightarrow S_3 \rightarrow S_6 \rightarrow S_2 \rightarrow S_3 \rightarrow S_6 \rightarrow \dots$

Problem ?
Common state S_3 can lead to one request being granted repeatedly while the other is ignored.
(i.e. we lose the order of arrival of requests)

Solution :
Split S_3



Properties 1, 2 satisfied.

3] S_0 : ✓ (S_1)
S_2 : ✓ (S_8)
S_5 : ✓ (S_8) (S_1)
⏟ n_1

S_0 : ✓ (S_2)
S_1 : ✓ (S_3)
S_4 : ✓ (S_6) (S_2)
⏟ n_2

∴ Property 3 satisfied.

All properties satisfied.

Minimum number of operators needed :
→ EU
→ Any one from (AX, EX)
→ Any one from (AU, AF, EF, AG, EG)

e.g. (EU, EX, AF)

Why ?
They are related!

$AX\phi = \sim EX \sim\phi$

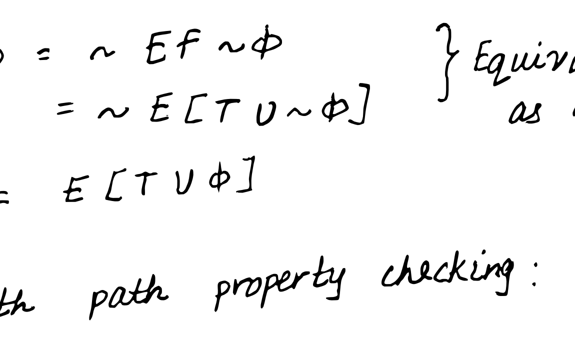
$AG\phi = \sim EF\sim\phi$
$= \sim E[TU\sim\phi]$

} Equivalent as result is same

$EF\phi = E[TU\phi]$

∞ length path property checking :

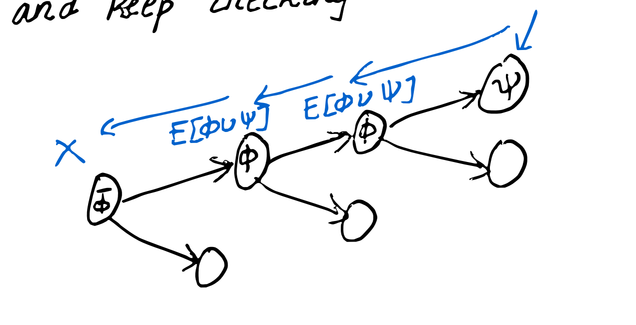
e.g. $EX\phi$
Find state where ϕ is true, go to it's pre-image and mark it as $EX\phi$.



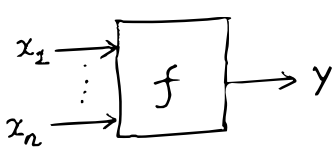
e.g. $AF\phi$
Find state where ϕ is true, go to it's pre-image check if all images of this pre-image satisfy ϕ.
If yes, mark state as $AF\phi$.

Current state $EF\phi \Longrightarrow$ all pre-images $EF\phi$

e.g. $E[\phi U \psi]$
Find state where ψ is true, go to it's pre-images and keep checking for ϕ.



Digital Circuit Testing:

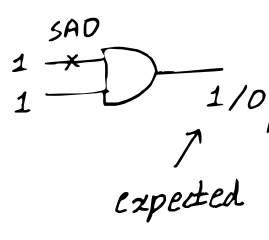


Generate compact test set to apply in reasonable time (few seconds)

$$y = f(x_1, x_2, \cdots, x_n)$$

Generating test vector → Algorithmic method
Generating test vector → Algebraic method

Faults: Stuck at 0 / Stuck at 1
(SA0) (SA1)



If fault is present, and distinguishable

$$(OG\ f^n) \oplus (Faulty\ f^n) = 1$$

different outputs

$$f(x_1, x_2, \cdots, x_n) \oplus f_\alpha(x_1, x_2, \cdots, x_n) = 1$$

OG f^n Faulty f^n

say fault : $SA0|_{x_i}$ $(x_i = 0)$

$$f = x_i f_{x_i} + \bar{x}_i f_{\bar{x}_i} \equiv x_i f_{x_i} \oplus \bar{x}_i f_{\bar{x}_i} \quad (\text{exclusive})$$

$x_i \rightarrow SA0 \rightarrow f_\alpha = 0 \cdot f_{x_i} + 1 \cdot f_{\bar{x}_i}$

$$f_\alpha = f_{\bar{x}_i}$$

$\therefore f \oplus f_\alpha = 1$

$\Rightarrow (x_i f_{x_i} \oplus \bar{x}_i f_{\bar{x}_i}) \oplus f_{\bar{x}_i} = 1$

$\Rightarrow x_i f_{x_i} \oplus (1 \oplus \bar{x}_i) f_{\bar{x}_i} = 1$

$\Rightarrow x_i (f_{x_i} \oplus f_{\bar{x}_i}) = 1$

$(f_{x_i} \oplus f_{\bar{x}_i})$ → Boolean Difference $(\partial f / \partial x_i)$

$$\therefore x_i \cdot \frac{\partial f}{\partial x_i} = 1$$

Thus to distinguish $(SA0)|_{x_i}$

$x_i = 1 \longrightarrow$ fault should be activated

$\frac{\partial f}{\partial x_i} = 1 \longrightarrow$ fault should get propagated

$$\therefore x_i \cdot (f_{x_i} \oplus f_{\bar{x}_i}) = 1$$

x_i → fault sensitization or activation condition

$(f_{x_i} \oplus f_{\bar{x}_i})$ → fault propagation condition

For $SA1|_{x_i}$:

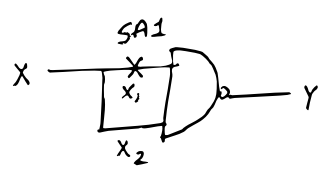
$$\bar{x}_i \cdot \frac{\partial f}{\partial x_i} = 1$$

$$\therefore \bar{x}_i \cdot (f_{x_i} \oplus f_{\bar{x}_i}) = 1$$

$\bar{x}_i$ → activation

$(f_{x_i} \oplus f_{\bar{x}_i})$ → propagation

e.g.



$x_1 = x, \quad x_2 = x$

$SA1|_{x_1} \longrightarrow$ Is it distinguishable ?
Can it be tested ?

$$\bar{x}_1 \cdot (f_{x_1} \oplus f_{\bar{x}_1}) = 1$$

$f = \overline{x_1 x_2} = \bar{x}_1 + \bar{x}_2$

$f_{x_1} = \bar{x}_2$

$f_{\bar{x}_1} = 1$

$\bar{x}_1 (\bar{x}_2 \oplus 1) = 1$

$\bar{x}_1 x_2 = 1$

$x_1 = x_2 = x$

$\therefore \bar{x} x = 1$

$0 = 1$ → not possible

i.e. $\bar{x}_1 \cdot \frac{\partial f}{\partial x_1} \neq 1 \longrightarrow SA1|_{x_1}$ cannot be tested as fault is not propagated

Redundant fault → No test vectors

Fault at intermediate points in the circuit:

Fault at some point h

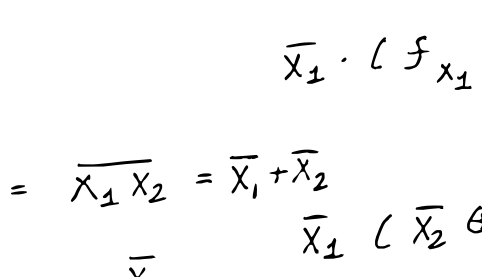
Write $Y = f(x_1, \ldots, x_n, h)$

$\therefore$ Checking for distinguishability:

$SA0|_h$: $h \cdot \frac{\partial f}{\partial h} = 1$

$SA1|h$: $\bar{h} \cdot \frac{\partial f}{\partial h} = 1$

e.g.



Assume one fault at a time.

$SA0|_{h_1}$ or $SA1|h_2$

$SA0|_{h_1}$

$Y = \overline{X_1 \cdot \overline{h_1 X_2}} = \bar{X}_1 + h_1 X_2$

$f_{h_1} = \bar{X}_1 + X_2$

$f_{\bar{h}_1} = \bar{X}_1$

Check :

$h_1 \cdot (f_{\bar{h}_1} \oplus f_{h_1}) = 1$

$h_1 \cdot (\bar{X}_1 \oplus (\bar{X}_1 + X_2)) = 1$

$(h_1 = X_1)$

$\therefore X_1 (X_1 X_2) = 1$

$\therefore X_1 X_2 = 1$

Test vector: $X_1\ X_2$ = 1 1

$Y = \overline{X_1 h_2} = \bar{X}_1 + \bar{h}_2$

$f_{h_2} = \bar{X}_1$

$f_{\bar{h}_2} = 1$

Check:

$\bar{h}_2 \cdot (f_{\bar{h}_2} \oplus f_{h_2}) = 1$

$\bar{h}_2 \cdot (1 \oplus \bar{X}_1) = 1$

$(h_2 = \overline{X_1 X_2})$

$\therefore X_1 X_2 \cdot (X_1) = 1$

$\therefore X_1 X_2 = 1$

Test vector: $X_1\ X_2$ = 1 1

$SA1|_{h_1}$

$\bar{h}_1 (f_{\bar{h}_1} \oplus f_{h_1}) = 1$

$\bar{X}_1 (X_1 X_2) = 1$

$0 = 1$

X

redundant
no test vectors

$SA0|_{h_2}$

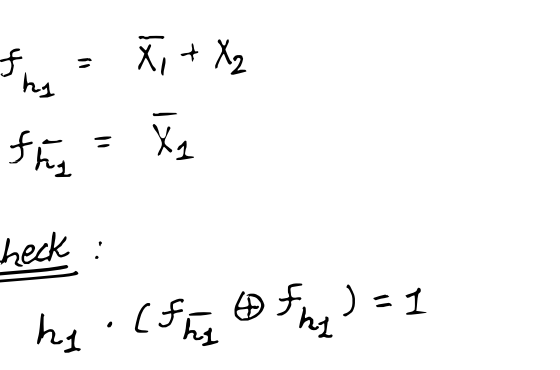
$h_2 (f_{\bar{h}_2} \oplus f_{h_2}) = 1$

$(\bar{X}_1 + \bar{X}_2) X_1 = 1$

$\therefore X_1 \bar{X}_2 = 1$

Test vector: $X_1\ X_2$ = 1 0

e.g.



$f = X_1 X_2 + X_3$

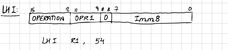
$\frac{\partial f}{\partial X_2} = X_3 \oplus (X_1 + X_3)$

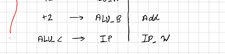
$= \bar{X}_3 X_1$

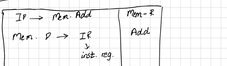
$\therefore \frac{\partial f}{\partial X_2} = 1 \Rightarrow X_1 \bar{X}_3 = 1$

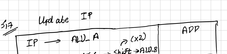
$\therefore SA0|_{X_2}$ → 1 1 0 ($X_1\ X_2\ X_3$)

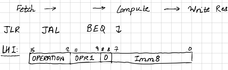
$SA1|_{X_2}$ → 1 0 0 ($X_1\ X_2\ X_3$)











224 Project:

Instructions:

ADD SUB ADI AND ORA MUL? IMP?

Fetch → Read → Execute → Write Result

S_1 S_2 S_3 S_4

LW

Fetch → Read → Compute Address → Read Memory → Write Result

S_5 S_6 S_7 S_8 S_9

SW

Fetch → Read → Compute Address → Write Memory

S_{10} S_{11} S_{12} S_{13}

LHI LLI

Fetch → → Compute → Write Result

JLR JAL BEQ J

LHI:

15 – 12	11 – 9	8	8 – 7 – 0
OPERATION	OPR1	0	Imm8

LHI R1, 54

SS1:

Fetch and update IP

IP → Mem_A	Mem_R
Mem_D → IR	IR_W
IP → ALU_A	
+2 → ALU_B	Add
ALU_C → IP	IP_W

SS2:

Understand and read

SS3:

Compute ZE8 + Shift

IR_{0-7} → ZE8_in	
ZE8_out → Shift_in	SR
Shift_out → T1	T1_W

→ LLI: OR or OL

SS4:

Write Result

$IR_{9-11} \rightarrow RF_A3$ $T1 \rightarrow RF_D3$	RF_W

JAL:

JAL R1, 54

15 – 12	11 – 9	8 – 0
OPERATION	OPR1	IMM9

Fetch
Write IP in R1
Compute IP + Imm*2
Update IP to IP + 2 + Imm*2

SS1

Fetch

$IP \rightarrow Mem_A$ $Mem_D \rightarrow IR$	Mem_R

SS2

Compute

$IR_{0-8} \rightarrow ZE9_in$ $ZE9_out \rightarrow Shift_in$ $Shift_out \rightarrow ALU_A$ $IP \rightarrow ALU_B$	Add

ALU_C → T1		T1_W

S3 Write result

IR_{9-11} → RF_A3		
IP → RF_D3		RF_W
T1 → ALU_A		
+2 → ALU_B		Add
ALU_C → IP		IP_W

?

JLR

JLR R_1, R_2

15–12	11–9	8–6	5–0
OPERATION	OPR1	OPR2	0 0 0 0 0 0

Fetch

IP → Mem_A	
Mem_D → IR	Mem_R

Read

$IR_{6-8} \rightarrow RF_A1$ $RF_D1 \rightarrow T1$	$T1_W$

Write Result

?

$IR_{9-11} \rightarrow RF_A3$ $IP \rightarrow RF_D3$ $T1 \rightarrow IP$	RF_W IP_W

BEQ → Pasting what was done in class

BEQ $R_1, R_2, 1D$

If $R_1 == R_2$ THEN

$IP = \underbrace{IP + 2}_{\text{next instruction}} + 1D \times 2$

→ IF $R_1 == R_2$ then

$IP = IP + 2 + \underline{\underline{Imm6 \times 2}}$

Why 2?

↳ 1 instruction = 2 byte

Imm6 → no. of instructions

15 – 12	11 – 9	8 – 6	5 – 0
OPERATION	OPR1	DEST	IMM6

16 bits

1. Fetch Instruction and update IP.
2. Understand and Read Operands (R_1 & R_2)
3. Compute (Compare R_1 & R_2) if $(R_1 - R_2) == 0$
   $Z = 1$
4. If $(Z == 1)$ THEN $IP = IP + Imm6 \times 2$

S14 Fetch

IP → Mem. Add	Mem-R
Mem. D → IR (→ inst. reg.)	Add
IP → ALU_A	IP_W
+2 → ALU_B	IR_W
ALU_C → IP	

S15 Operand Read (OR)

IR 9-11 → RF_A1	T1_W
IR 6-8 → RF_A2	T2_W
RF_D1 → T1	
RF_D2 → T2	

→ register file

T2 → temporary register

S16 Execute

T1 → ALU_A	SUB
T2 → ALU_B	Z_W
ALU_Z → Z	

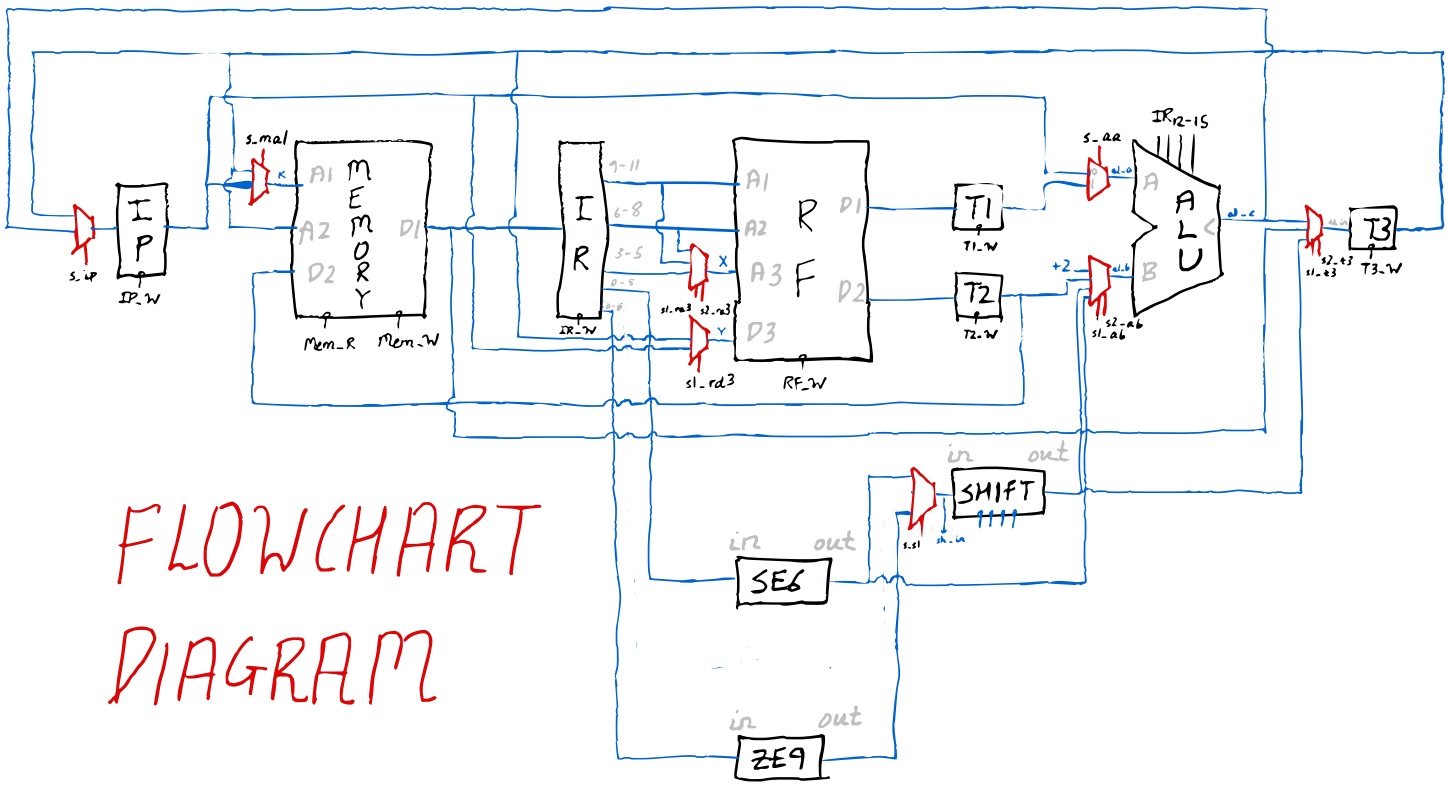
S17 Update IP

IP → ALU_A	ADD
Immd → SE6 → Shift (x2) → ALU_B	
IF(Z == 1)	IP_W
ALU_C → IP	
else	
IP → IP	

Extend to 16-bit

ALUL
1
IP
IP
0
Z

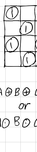
Memory
FF7
FFD
8 bits
PC
+
2
1 byte



FLOWCHART DIAGRAM

Erdem

Q1]



$A \oplus B \oplus C \oplus D$
or
$A \odot B \odot C \odot D$

Q2]

$$\frac{\partial f}{\partial x_i} = f_{x_i} \oplus f_{\overline{x}_i}$$

$$\frac{\partial (f+g)}{\partial x_i} = (f_{x_i} + g_{x_i}) \oplus (f_{\overline{x}_i} + g_{\overline{x}_i})$$

$$= (f_{x_i} + g_{x_i})(\overline{f_{\overline{x}_i} + g_{\overline{x}_i}}) + \overline{(f_{x_i} + g_{x_i})}(f_{\overline{x}_i} + g_{\overline{x}_i})$$

$$= f_{x_i}\overline{f}_{\overline{x}_i}\overline{g}_{\overline{x}_i} + f_{x_i}$$

$$\overline{g} \oplus \frac{\partial g}{\partial x_i} = \overline{g} \oplus (g_{x_i} \oplus g_{\overline{x}_i})$$

$$\overline{g} = 1 \oplus g = 1 \oplus g_{\overline{x}_i} \oplus x_i \oplus (\overline{x}_i \oplus g_{\overline{x}_i})$$

$f(x_1, x_2, \ldots, x_n) = f_{\overline{x}_i} \oplus x_i(f_{\overline{x}_i} \oplus f_{x_i})$

$$\therefore \overline{g} \oplus \frac{\partial g}{\partial x_i} = 1 \oplus g_{x_i} \oplus \overline{x}_i \frac{\partial g}{\partial x_i} \oplus \frac{\partial g}{\partial x_i}$$

$$= \overline{g}_{\overline{x}_i} \oplus \overline{x}_i \frac{\partial g}{\partial x_i} \oplus x_i \frac{\partial g}{\partial x_i}$$

$$g_{\overline{x}_i} \oplus x_i = g_{x_i}$$

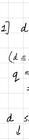
$$\left(\overline{f} \oplus \overline{x}_i \frac{\partial f}{\partial x_i} \oplus \frac{\partial f}{\partial x_i}\right) = \overline{g}_{\overline{x}_i} \frac{\partial f}{\partial x_i}$$

$$1 \oplus f_{x_i} \oplus 0 \quad f_{\overline{x}_i} \frac{\partial g}{\partial x_i} \oplus g_{\overline{x}_i} \frac{\partial f}{\partial x_i}$$

$$\overline{f} \frac{\partial g}{\partial x_i} \oplus g \frac{\partial f}{\partial x_i}$$

=

$A \equiv G$
$C \equiv E$
$B \equiv$



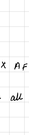
$(A\,B\,C\,D\,E\,F\,G\,H)$ D-equivalent
↓
$(A(GH)\,C\,B\,C(E\,E\,F))$ 1-equivalent
$(AH)\,(CG)\,(BDF)\,(BC)$ 2-equivalent
$(AH)\,(CG)\,(BDF)\,(CEC)$ 3-equivalent

As 2-equivalent states are identical to 3-equivalent states, (AH) (BDF) and (CE) are ∞-equivalent

∴ As $A \equiv H$
$B \equiv F$
$C \equiv E$
$D \equiv F$, M_1 is contained in machine M_2.

Starting conditions under which the machines are equivalent:

M_1: A, B, D, C
M_2: H, F, F, E



Q4?] 1] d saD

$(A \equiv a)$

$$a = a \oplus (d+b) \oplus \overline{b}(d+b)$$
$$= a\overline{d}\,\overline{b} + \overline{a}d + \overline{a}b + \overline{b}d$$

d saD
↓

$$d\,\frac{\partial a}{\partial d} = 1$$

$$d\,((\overline{a} + \overline{b} + \overline{a}b) \oplus (a\overline{b} + \overline{a}b)) = 1$$
$$d\,(\overline{a} + \overline{b}) \oplus a \oplus b) = 1$$

$(B \equiv a)$

$$\therefore a((\overline{a} + \overline{b}) \oplus a \oplus b) = 1$$
$$\therefore a\overline{b} \oplus a \oplus ab = 1$$
$$\therefore a(\overline{a} \oplus \overline{b} \oplus b)$$
$$\therefore a(b \oplus b)$$
$$= 0 = 1$$

redundant ↩

2] m saD

$$a = a \oplus m + \overline{b}m$$
$$= a\overline{m} + \overline{a}m + \overline{b}m$$

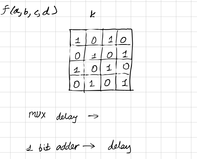
$$\therefore m\,\frac{\partial a}{\partial m} = m(\overline{a} + \overline{b}) \oplus a)$$
$$= m(\overline{a}(\overline{a} + \overline{b}) + a(ab))$$
$$= m(\overline{a} + \overline{a}\overline{b} + ab)$$
$$= m(\overline{\overline{a} + ab})$$
$$= m(a\overline{b})$$
$$\overline{a} + b$$
$$= (a+b)(\overline{a}+b)$$
$$= b \qquad = 1$$

$m = a+b$

AX AF ϕ

In all paths, in next state
↳ in all paths, future ϕ

AF AX ϕ
In all paths, in future, all next ϕ



F(a,b,c,d) k

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

mux delay →

1 bit adder → delay

$S2 = \overline{X_3}(\overline{X_0} + X_2 + X_1)$
$S1 = \overline{X_2}(X_3 + \overline{X_1}X_0)$
$S_0 = X_3X_1 + \overline{X_3}\,\overline{X_2}\,\overline{X_1}X_0 + X_3X_2X_0$

for S2-0

$S4\_i2 = X_3\overline{X_2}X_1$
$S4\_i1 = \overline{X_3} + \overline{X_2}\,\overline{X_1}$
$S4\_i0 = \overline{X_3}$

$S3\_i0 = \overline{X_3} + \overline{X_2}\,\overline{X_1} + \overline{X_1}X_0 + X_1\overline{X_0}$
$S3\_i1 = X_3X_0(X_2 + X_1)$

$S5\_01 = \overline{X_0}$
$S5\_00 = X_1X_0$

$S7\_i0 = X_0$

$S4\_01 = X_3X_2X_0$
$S4\_00 = X_3X_2\overline{X_0}$

$S7\_00 = \overline{X_2}$

$S3\_00 = X_3X_2\overline{X_0}$

Precious ☆

S1: $X_3\overline{X_2}\,\overline{X_0}$ } shift
S0 = X_2

S1 S0
0L - 00 - LLI
2L - 01 - JAL, BEQ
8L - 10 - LHI

	X_3	X_2	X_1	X_0	Z			
Add	0	0	0	0	0	0	0	0
API	0	0	0	1	0	0	0	0
Sub	0	0	1	0	0	0	0	0
MUL	0	0	1	1	0	0	0	0
AND	0	1	0	0	0	0	0	0
ORA	0	1	0	1	0	0	0	0
IMP	0	1	1	0	0	0	0	0
X	0	1	1	1	0	X	X	X
LHI	1	0	0	0	0	0	0	0
LLI	1	0	0	1	0	0	0	0
LLW	1	0	1	0	0	0	0	0
SLW	1	0	1	1	0	0	0	0
BEQ	1	1	0	0	0	0	0	0
JAL	1	1	0	1	0	0	1	0
X	1	1	1	0	0	X	X	X
JLR	1	1	1	1	0	X	X	X
Add	0	0	0	0	1			
API	0	0	0	1	1			
Sub	0	0	1	0	1			
MUL	0	0	1	1	1			
AND	0	1	0	0	1			
ORA	0	1	0	1	1			
IMP	0	1	1	0	1			
X	0	1	1	1	1			
LHI	1	0	0	0	1			
LLI	1	0	0	1	1			
LLW	1	0	1	0	1			
SLW	1	0	1	1	1			
BEQ	1	1	0	0	1	1	0	0
JAL	1	1	0	1	1	0	1	0
X	1	1	1	0	1	X	X	X
JLR	1	1	1	1	1	X	X	X

Re-exam:

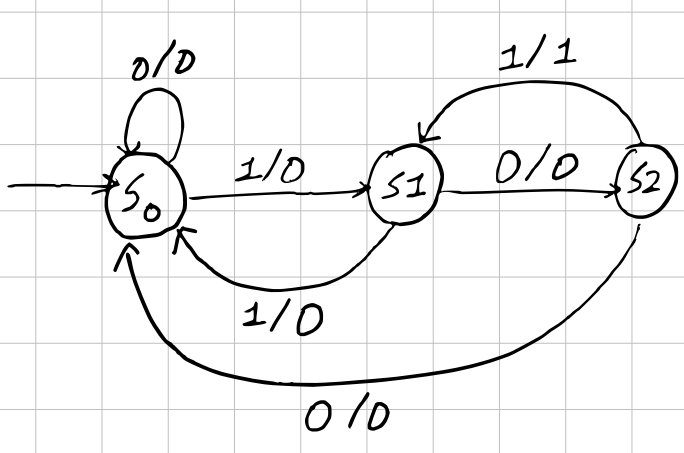
Q9]

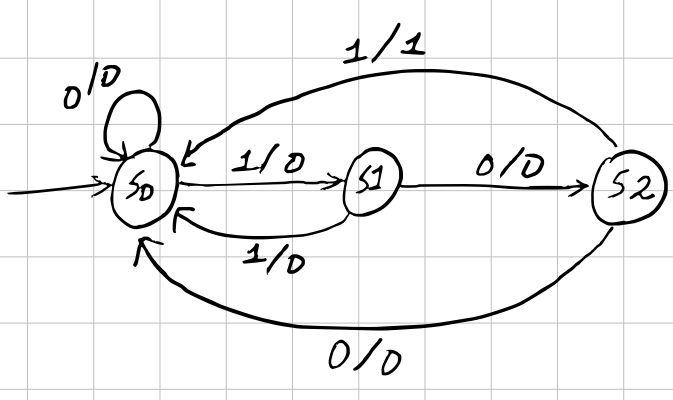
		0000
G0	0	0000
	1	0001
G1	8	1000
	3	0011
	5	0101
	6	0110
G2	10	1010
	7	0111
G3	14	1110
G4	15	1111

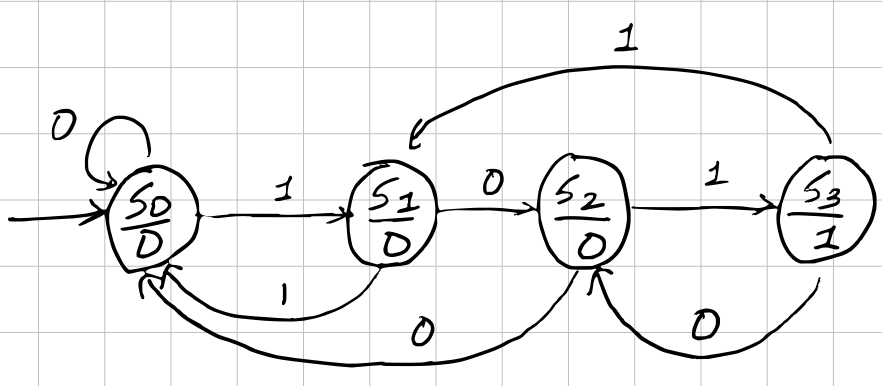
0000
0001
0011
0101
0110
0111
1000
1010
1110
1111

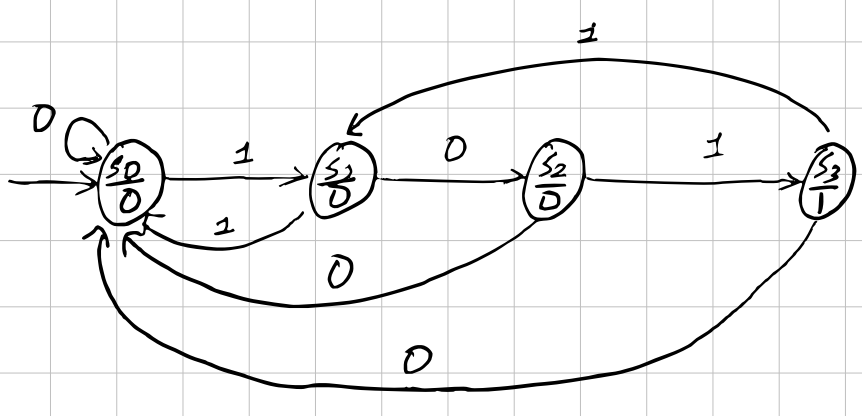
(0, 1) 000_
(0, 8) _000
~~(1, 3) 00_1~~
(8, 10) 10_0
~~(3, 7) 0_11~~
~~(5, 7) 01_1~~
~~(6, 7) 011_~~
~~(6, 14) _110~~
(10, 14) 1_10
~~(7, 15) _111~~
~~(14, 15) 111_~~

(1, 3, 5, 7) 0__1
(C, 7, 14, 15) _11_
(6, 14, 7, 15) _11_









$D = \overline{\overline{AB} \cdot m} = AB + \overline{m}$

$D_m = AB$

$D_{\overline{m}} = 1$

$\frac{\partial D}{\partial m} = 1 \oplus AB$

$= \overline{AB}$

$= \overline{A} + \overline{B}$

$\therefore \quad m \cdot \frac{\partial D}{\partial m} = 1$

SAD|m :

$m = \overline{\overline{B}C} = B + \overline{C}$

$(B + \overline{C}) \angle (\overline{A} + \overline{B}) = 1$

$\therefore \quad A = 0 \quad B = X \quad C = 0$

$B = 1$