

# CPU Architecture :

# Specification

Determine the application, the tasks to be fulfilled

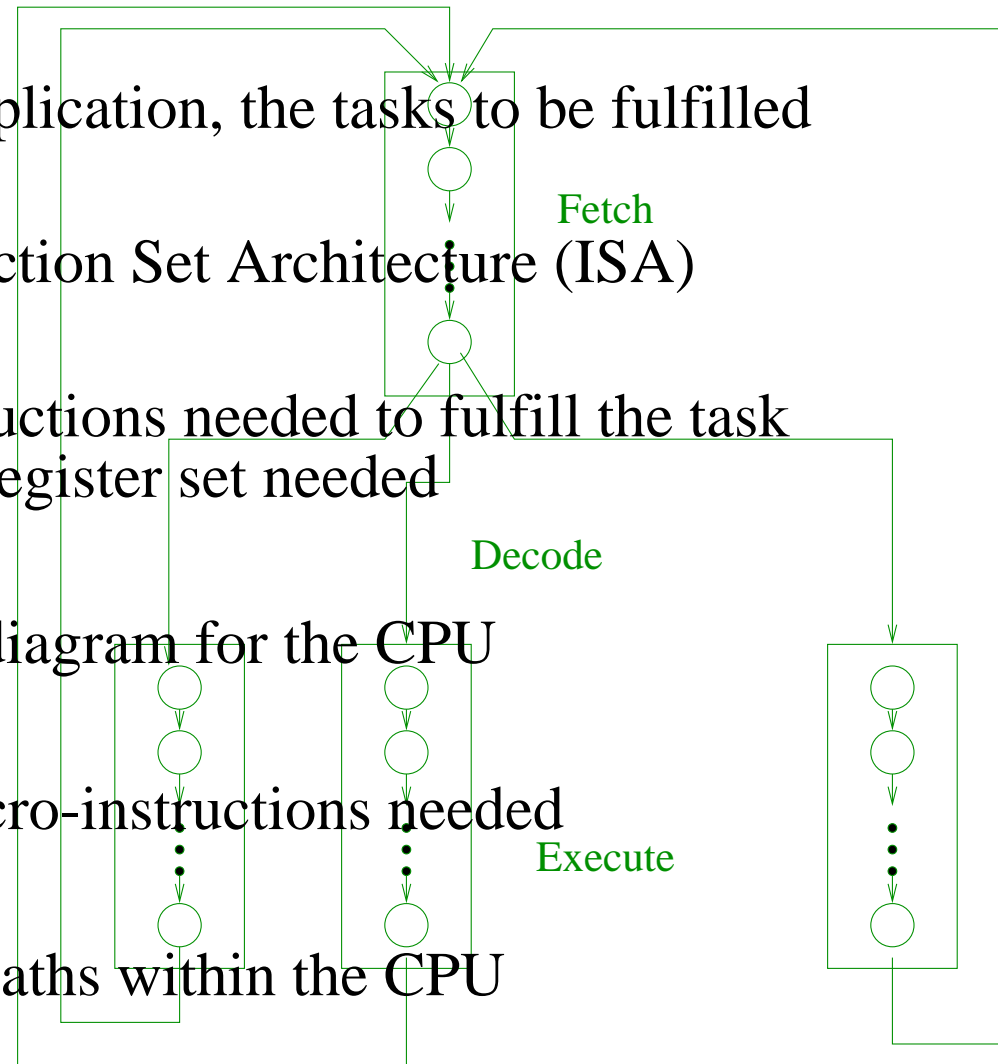
Design the Instruction Set Architecture (ISA)

- select the instructions needed to fulfill the task
- decide on the register set needed

Design the state diagram for the CPU

- specify the micro-instructions needed

Identify the datapaths within the CPU

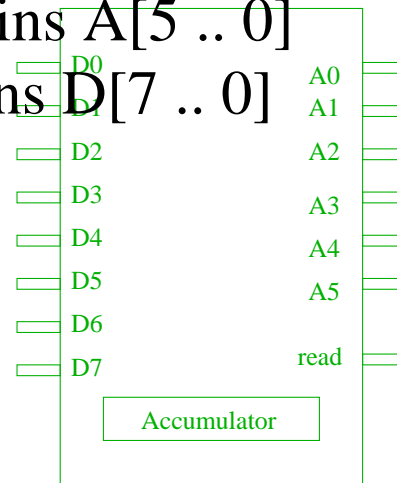


# CPU Architecture : Carpinelli's very simple CPU.

---

It can access 64 words of memory. Each word is 8 bits wide.

It outputs a 6 bit address on pins A[5 .. 0]  
and reads an 8 bit value on pins D[7 .. 0]



It has four instructions :

- ADD
- AND
- JMP
- INC

It has one programmer accessible register :

- AC an 8-bit accumulator

## **CPU Architecture : Carpinelli's very simple CPU.**

---

It will require the following register set

- 8 bit Accumulator, AC
- 6 bit Program Counter, PC
- 6 bit Address Register, AR
- 8 bit Data Register, DR
- 2 bit Instruction Register, IR

# CPU Architecture : Fetch - Execute.

---

Instruction	Op-code	Operation
ADD	00AAAAAA	$AC \leftarrow AC + M[AAAAAA]$
AND	01AAAAAA	$AC \leftarrow AC \wedge M[AAAAAA]$
JMP	10AAAAAA	GOTO AAAAAA
INC	11AAAAAA	$AC \leftarrow AC + 1$

---

Fetch Instruction :

- address to memory, on address pins
- mem ready, so signal read, read data pins

Decode is effected by the value in IR

Separate states will represent the instruction execution

# CPU Architecture : Fetch cycle. 1

---

The fetch phase will take three states :

FETCH1 :  $AR \leftarrow PC$

As AR will be connected to A[5..0], this places the address on the address pins

FETCH2 :  $DR \leftarrow M, \quad PC \leftarrow PC + 1$

Assert 'read', data available on D[7..0], store in DR. Increment the Program Counter.

## CPU Architecture : Fetch - cycle. 2

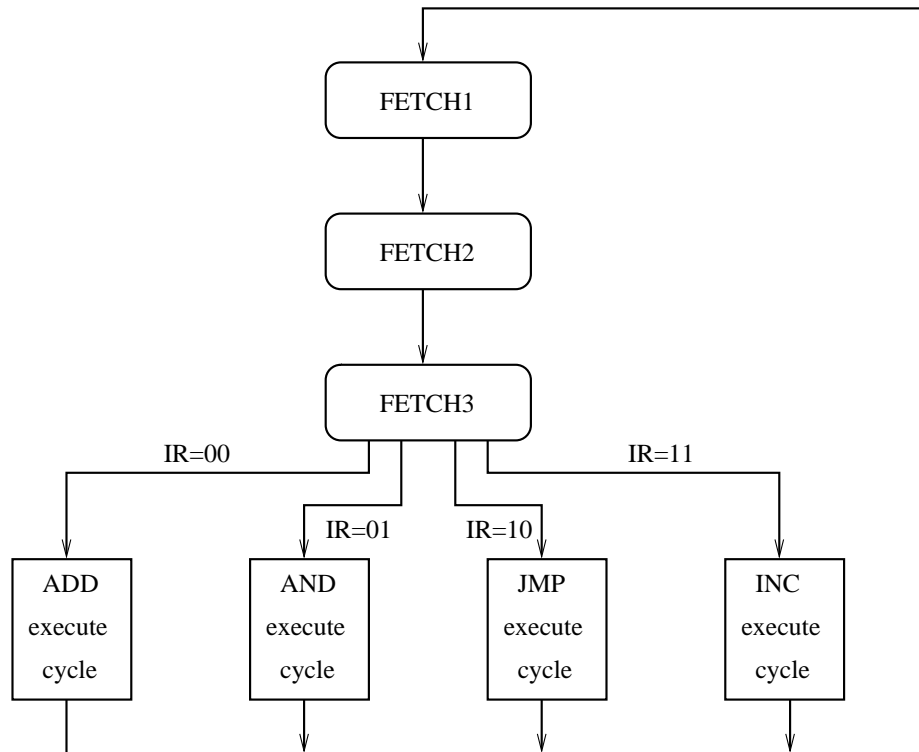
---

FETCH3:  $IR \leftarrow DR[7..6], AR \leftarrow DR[5..0]$

- Copy the instruction into the IR.
  - Copy the lower 6 bits into the AR as if it's an
    - ADD
    - or an
    - AND
- then this will be the address to get the data from.

This saves one cycle during execute.

# CPU Architecture : establish the execute steps.



ADD1: DR  $\leftarrow$  M

ADD2: AC  $\leftarrow$  AC + DR

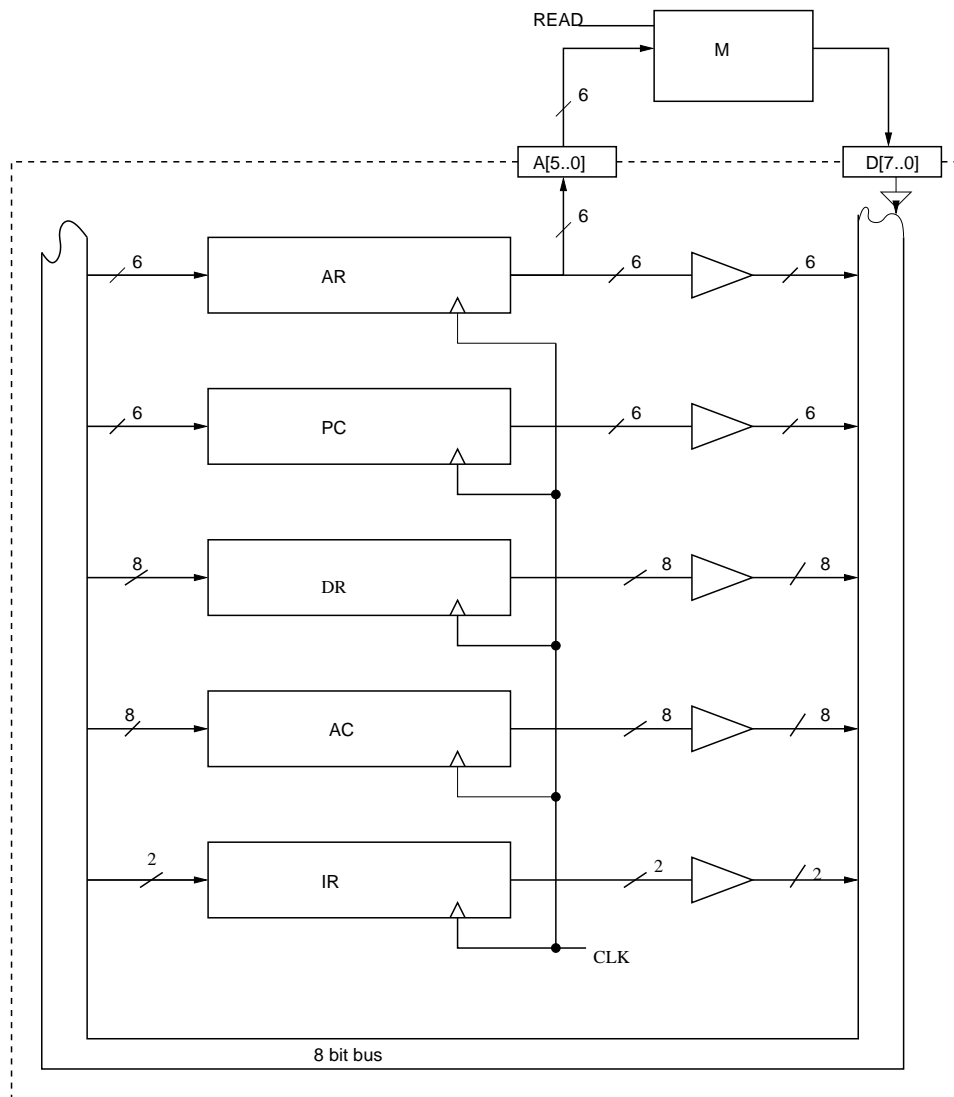
AND1: DR  $\leftarrow$  M

AND2: AC  $\leftarrow$  AC ^ DR

JMP1: PC  $\leftarrow$  DR[5..0]

INC1: AC  $\leftarrow$  AC + 1

# CPU Architecture : Establish data paths.



Sort the register transfers by destination

AR : AR  $\leftarrow$  PC  
 AR  $\leftarrow$  DR[5..0]

PC : PC  $\leftarrow$  PC + 1  
 PC  $\leftarrow$  DR[5..0]

AC : AC  $\leftarrow$  AC + DR  
 AC  $\leftarrow$  AC ^ DR  
 AC  $\leftarrow$  AC + 1

IR : IR  $\leftarrow$  DR[7..6]

## **CPU Architecture : Register Selection.**

---

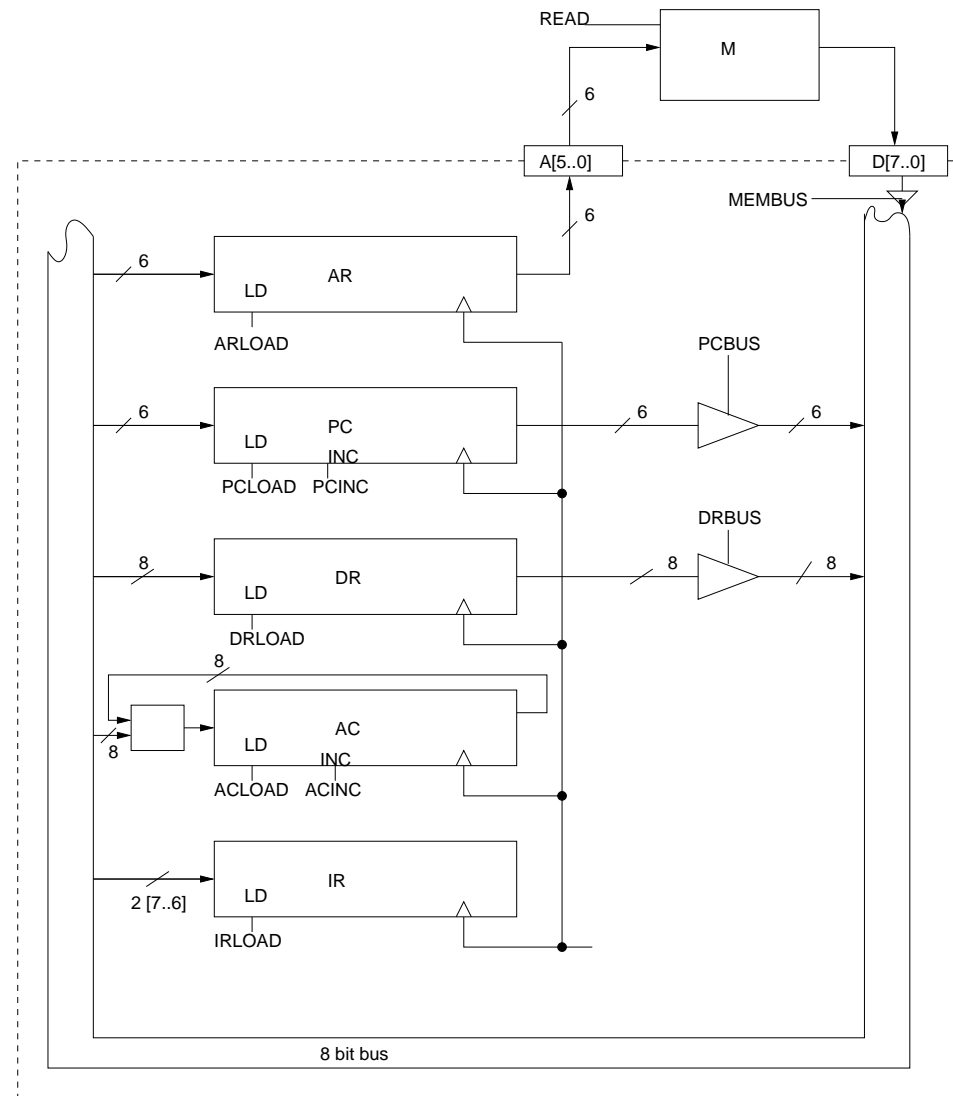
- AR only supplies data to memory, not other components
- IR doesn't supply data to any component via the internal bus
- AC does not supply data to any component
- Bus is 8 bits wide, but not all registers are, so specify the width of connections
- AC must load sum of AC and DR, also logical AND of AC and DR so we need an ALU

## **CPU Architecture : Control Signals.**

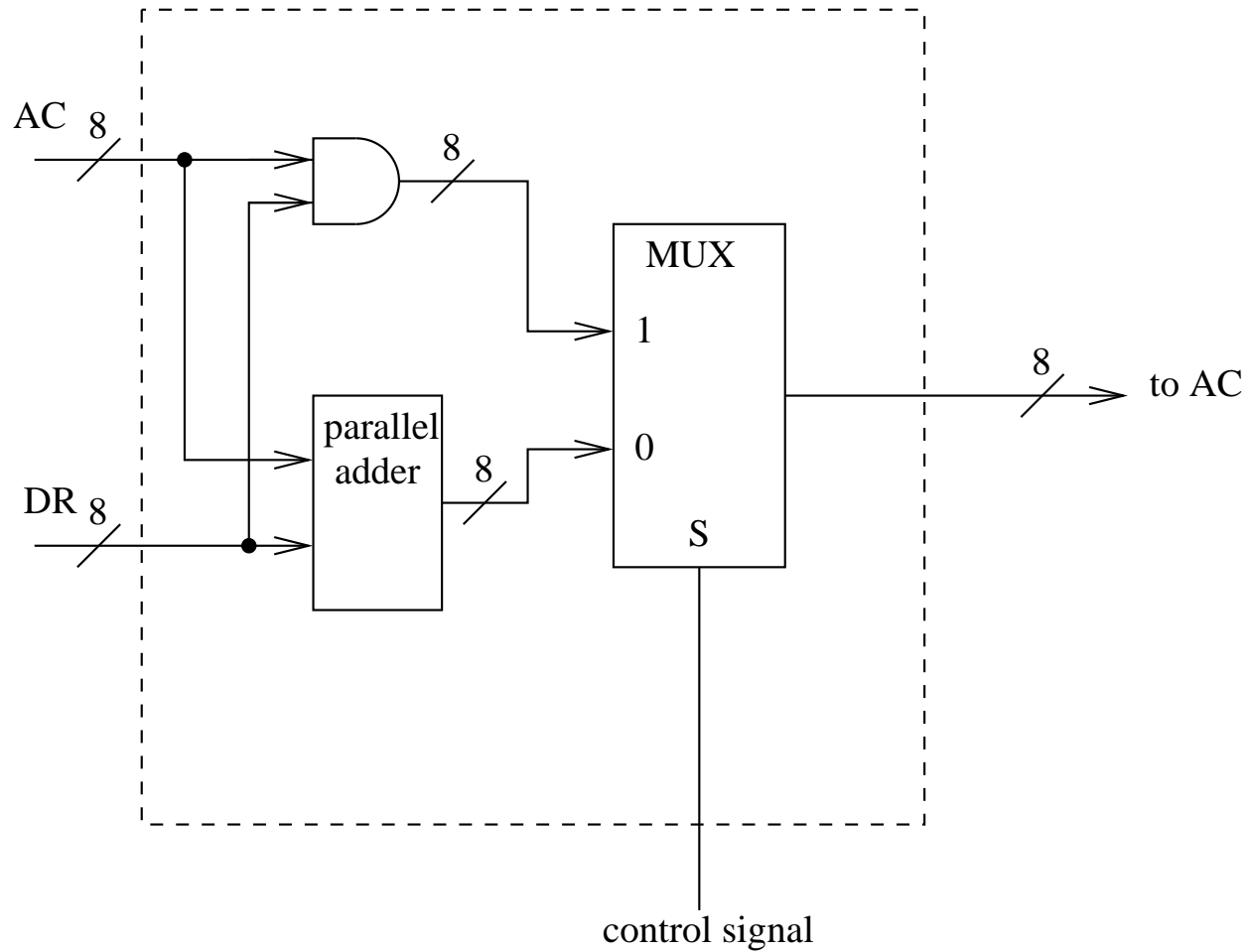
---

- We need a load signal for all registers that act as destinations for data
- We need bus enable signals for those registers that output data to the internal bus
- We need to ensure that all transfers that take place in the same state can occur simultaneously
- We need to be able to increment the PC register and the AC register.

# CPU Architecture : Final Register Selection.

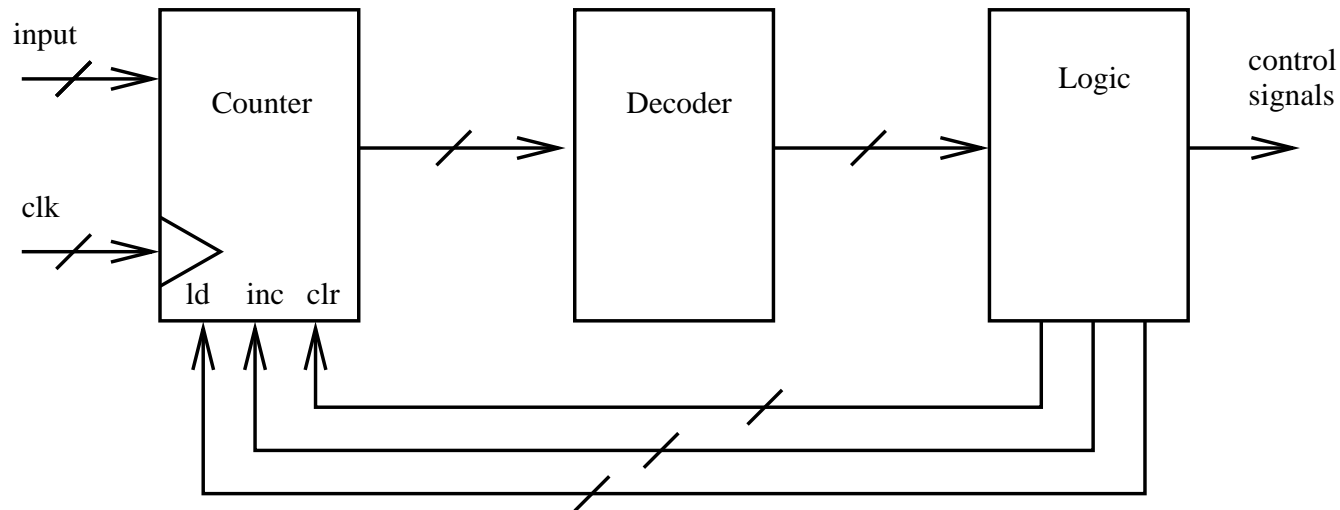


# CPU Architecture : Simple ALU.



# CPU Architecture : Hardwired Control.

---



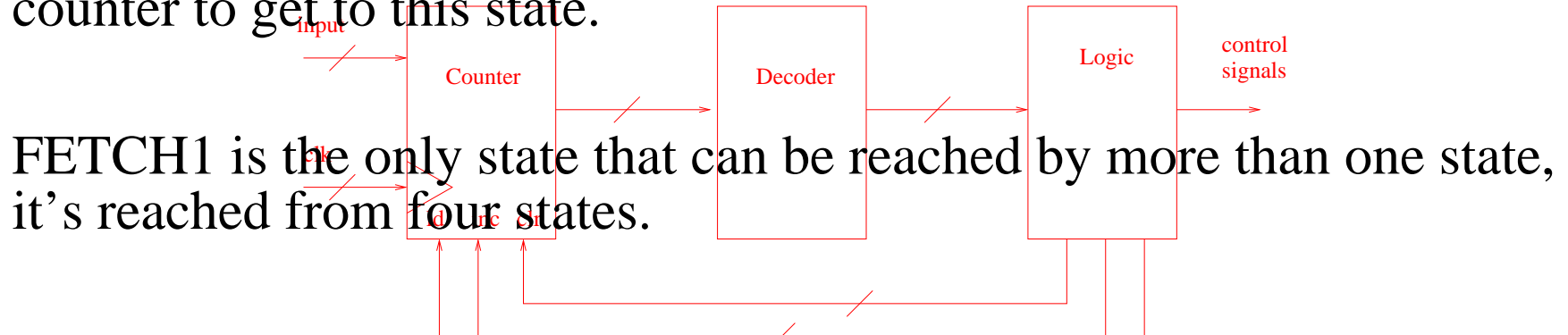
This CPU has 9 states therefore we need :

- a 4 bit counter
- a 4 to 16 decoder of which 7 outputs will not be used.

# CPU Architecture : Assigning states.

---

- 1) Assign FETCH1 to counter value 0 and use the CLR input to the counter to get to this state.



FETCH1 is the only state that can be reached by more than one state, it's reached from four states.

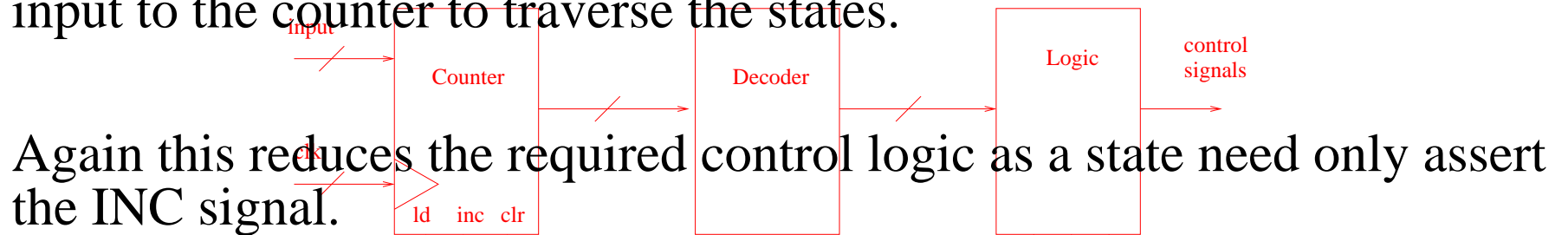
Branches to FETCH1 can then be made by clearing the counter, this is easy to implement in the control unit logic.

All the other states can only be reached from one other state.

# CPU Architecture : Sequential States.

---

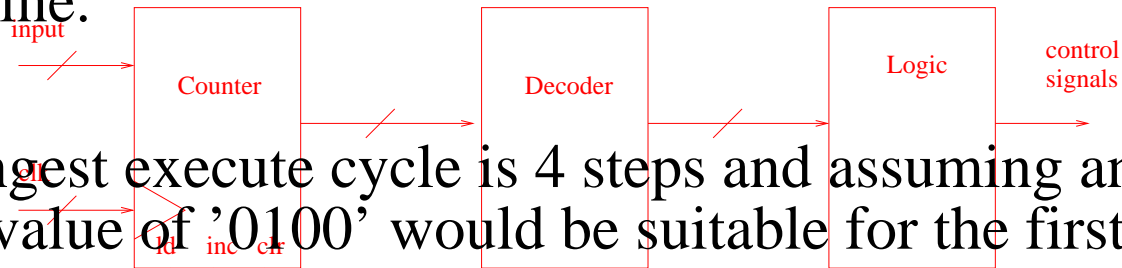
- 2) Assign sequential states to sequential counter values, using the INC input to the counter to traverse the states.



eg. Assign counter value 1 to FETCH2 and counter value 2 to FETCH3. Similarly AND1 and AND2 would be assigned consecutive values.

## CPU Architecture : Execute States.

- 3) Use the instruction opcode to assign the first state of each execute routine, allowing for the maximum number of steps required for an execute routine.



eg. if the longest execute cycle is 4 steps and assuming an opcode '01' then a state value of '0100' would be suitable for the first step.

Use the opcode as data input to the counter and load (ld) this value into the counter as the start point for the execute cycle

load 0100 ->inc, 0101 -> inc, 0110 -> inc, 0111 ->clr, 0000

# CPU Architecture : States attempt 1

---

- The input to the counter needs to be a function of IR
- The function needs to be as simple as possible

- 1 0 IR[1..0]

- This gives load values of

IR	Counter value	State
00	1000	ADD1
01	1001	AND1
10	1010	JMP1
11	1011	INC1

## CPU Architecture : States attempt 2

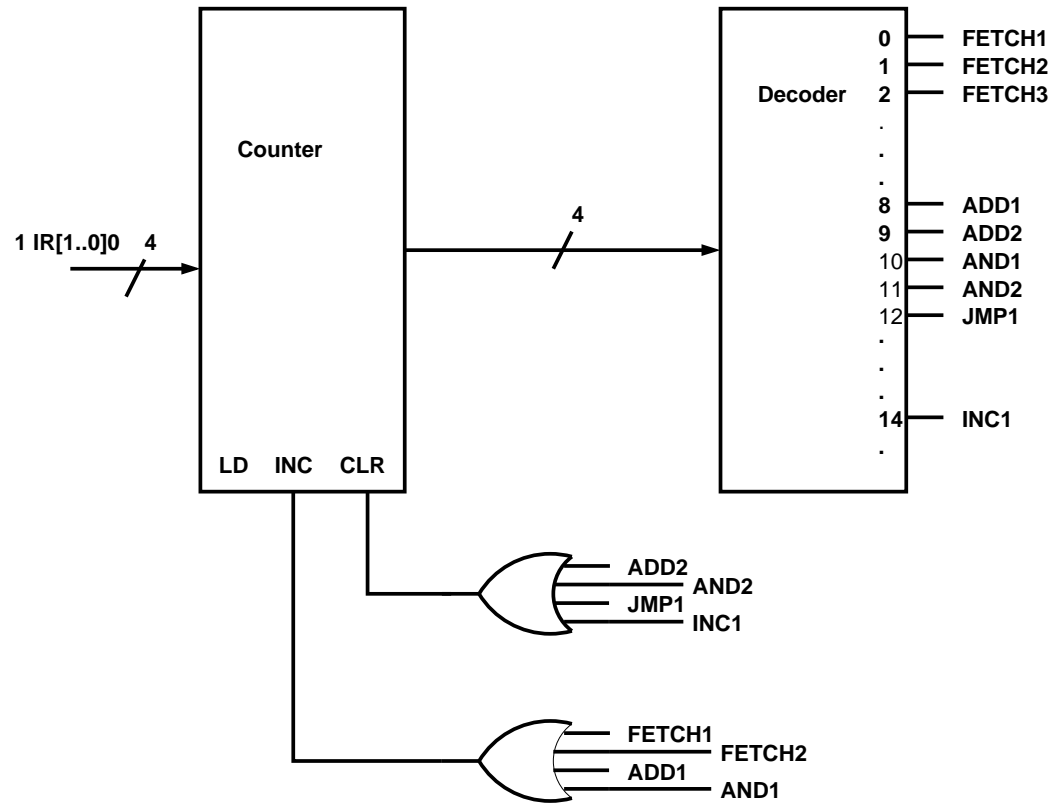
---

■ Given that there are a maximum of two states for an execute

- 1 IR[1..0] 0 gives a better mapping
- 

IR	Counter value	State
00	1000	ADD1
	1001	ADD2
01	1010	AND1
	1011	AND2
10	1100	JMP1
11	1110	INC1

# CPU Architecture : Control Signals



## CPU Architecture : Control Signals 2

---

PCLOAD = JMP1  
PCINC = FETCH2  
DRLOAD = FETCH2 \ / ADD1 \ / AND1  
ACLOAD = ADD2 \ / AND2  
ACINC = INC1  
IRLOAD = FETCH3

MEMBUS = FETCH2 \ / ADD1 \ / AND1  
PCBUS = FETCH1  
READ = FETCH2 \ / ADD1 \ / AND1

# CPU Architecture : States attempt 1

---

