UNIT – II

Custom Single Purpose Processors →

A single purpose processor is a digital system intended to solve a specific computational task. The processor may be a standard one, intended for use in a wide variety of applications in which the same task must be performed. The manufacturer of such an off-the-shelf processor sells the device in large quantities.

On the other hand the Processor may be a custom one, build by a designer to implement a task specific to a particular application.

Benifits →

(1) Performance of custom single Purpose Processor may be fast due to fewer clock cycles resulting from a customized datapath, and due to shorter clock cycles resulting from simpler functional units, less multiplexors, or simple control logic

(2) Size of custom signal Purpose Processor may be small due to a simpler datapath and no program memory. In fact, the Processor may be faster and smaller than a standard one implementing the same functionality, since we can optimize the implementation for our particular task.

Processor

↓

– Digital circuit that performs a computational tasks should contain controller and datapath

↓

General purpose
– variety of computation tasks

Single-purpose
– one particular computation task

↓

Custom Single purpose
– non-standard task.

– A custom Single-purpose processor may be fast, small, low power device having high NRE, longer time to market, less flexible

NRE – non recurring Engineering (one time cost to Research & Development)

Datapath → The datapath Consists of the Circuitry for transforming data and for storing temporary data. the datapath contains an airthmetic-logic Unit (ALU) Capable of transforming data through operations such as addition, Subtraction logical AND, logical OR, inverting and shifting

The ALU also generates status signals, often stored in an status register indicating

particular data conditions. Such conditions include indicating whether data is zero, or whether an addition of two data items generates a carry.
- The data path also contain registers capable of storing temporary data. Temporary data may include data brought in from memory but not yet sent through the ALU, data comming from the ALU that will be needed for later ALU operations or will be sent back to memory, and data that must be moved from one memory location to another.

The internal data bus is the bus over which data travels within the datapath, while the external data bus is the bus over which data is brought to and from the data memory.

FSMD → A finite state machine with datapath is a mathematical abstraction that is sometimes used to design digital logic or computer programs.
- An FSMD is a digital system composed of a finite-state machine, which controls the program flow, and a datapath which performs data processing operations.
- fSMDs are essentially sequential programs in which statements has been scheduled into states thus resulting in more complex state diagrams.

FSM → A finite state machine (FSM) is a sequential circuit with random Next-state logic

– The derivation of An FSM starts with a more abstract Model, such as a state diagram or an algorithm state Machine (ASM) chart. Both show the interaction and transitions between the internal states in graphical formats.

– formally an FSM is specified by five entities
(1) symbolic states
(2) input signal
(3) output signal
(4) Next-state function
(5) output function.

FSM Representation →

The design of an FSM normally starts with an abstract, graphic description, such as a state diagram or An ASM chart. Both descriptions utilize Symbolic state Notations, show the transition among the states and indicate the output values Under various conditions.

A state diagram or An ASM chart Can Capture All the needed information (i.e state, input, output, next-state function, and output function) in a single graph.

State diagram → A state diagram Consists of nodes, which are draws as circles (Also Known

as bubbles) and one-direction transition arc.

— A Node represents a unique state of the FSM and it has a unique symbolic name.

— An Arc represents a transition from one state to another and is labeled with the condition that will cause the transition. The condition is expressed as a logic expression composed of input signals. An arc will be taken when the corresponding logic expression is evaluated to be logic '1'

— The output values are also specified on the state diagram. The moore output is a function of state and thus is naturally placed inside the state bubble. On the other hand, the mealy output depends on both state and input and thus is placed under the condition expression of the transition arc.

ASM Chart —

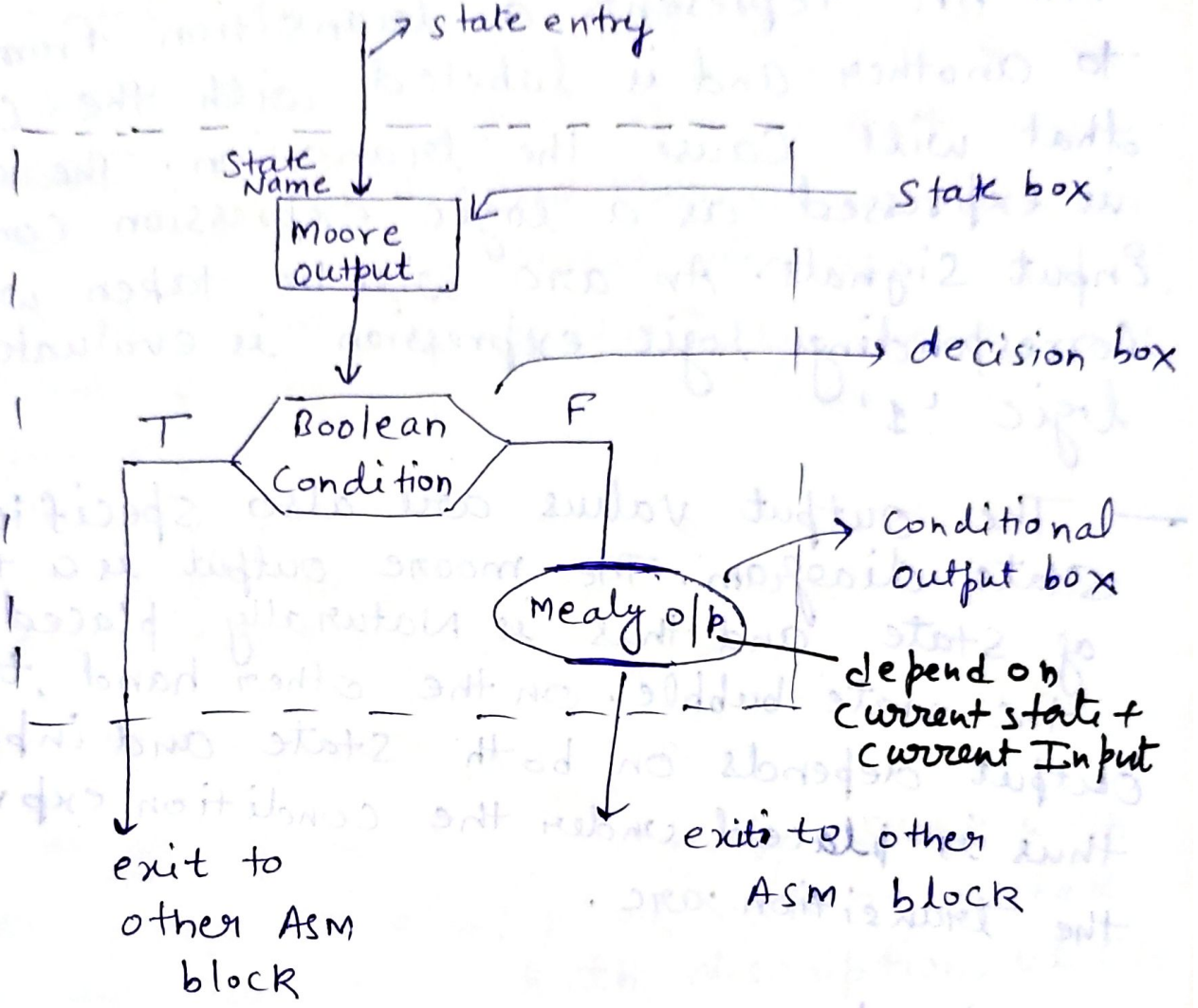
An algorithmic state machine (ASM) chart is an alternative method for representing an FSM Although an ASM chart contains the same Amount of information as a state diagram, it is more discriptive

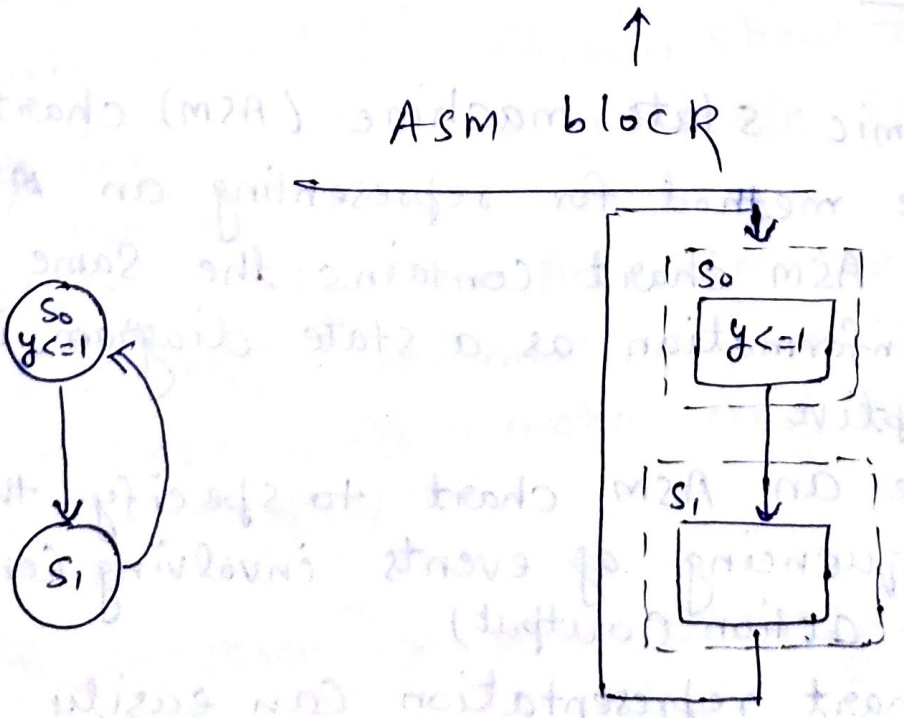
we can use an ASM chart to specify the complex sequencing of events involving commands (input) and action (output)

— An ASM chart representation can easily be transformed to VHDL code

— An Asm block consists of one state box and An optional Network of decision boxes and conditional output boxes.

— The state box represents a state in an FSM





Example of state diagram and Asm chart Conversion

~~State machine~~ Clocked sequential circuits →

→ In general a synchronous sequential circuit which is also referred to as a state machine is represented schematically by circuit in fig (a)

The ckt has n input and m outputs. it has memory devices in the feedback loop which ensure that the next state of the circuit is affected not only by the inputs but also its present output

→ The signal value at the output of each memory element is ~~refff~~ referred to as the state variable

→ The combination of values at the outputs of the K memory elements $y_1, y_2 \cdots y_K$ defines the present internal state of the circuit

the external input $x_1, x_2 \cdots x_n$ and the state variables $y_1, y_2 \cdots y_K$ are supplied to the combinational circuit which in turn produces the output $Z_1, Z_2, \cdots Z_m$ and the values $Y_1 Y_2 \cdots Y_K$

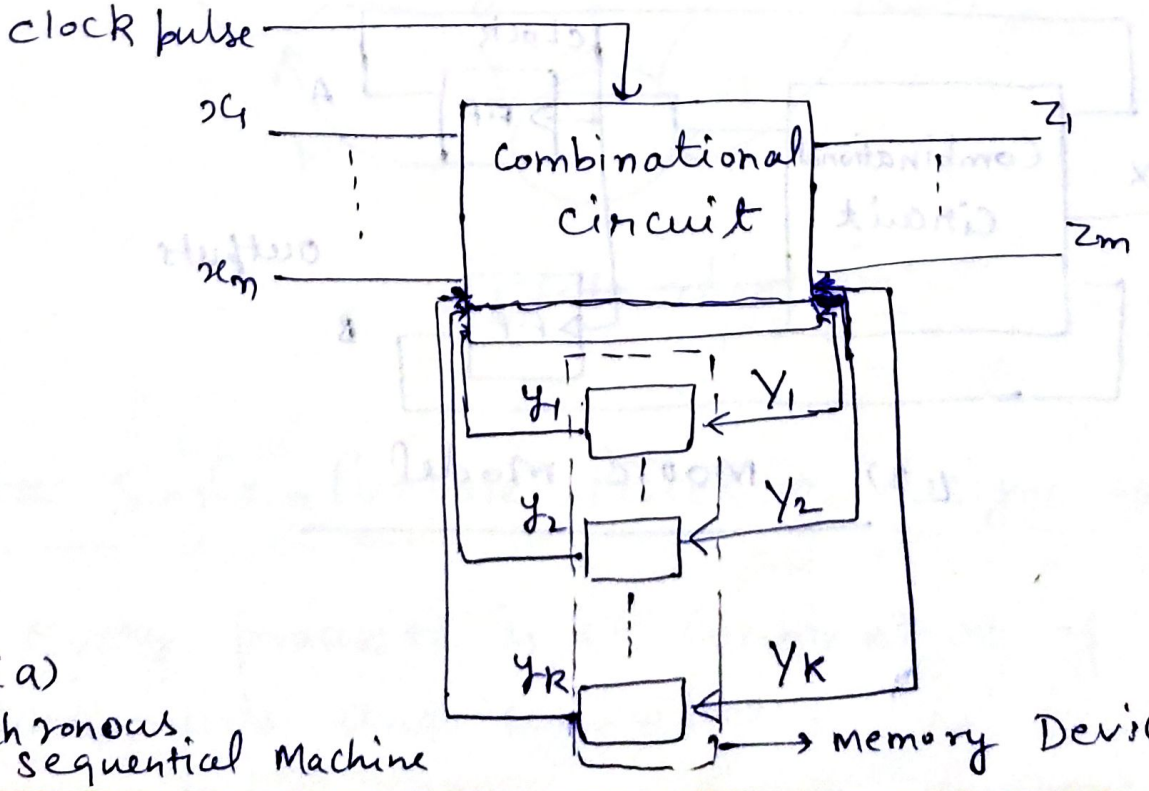
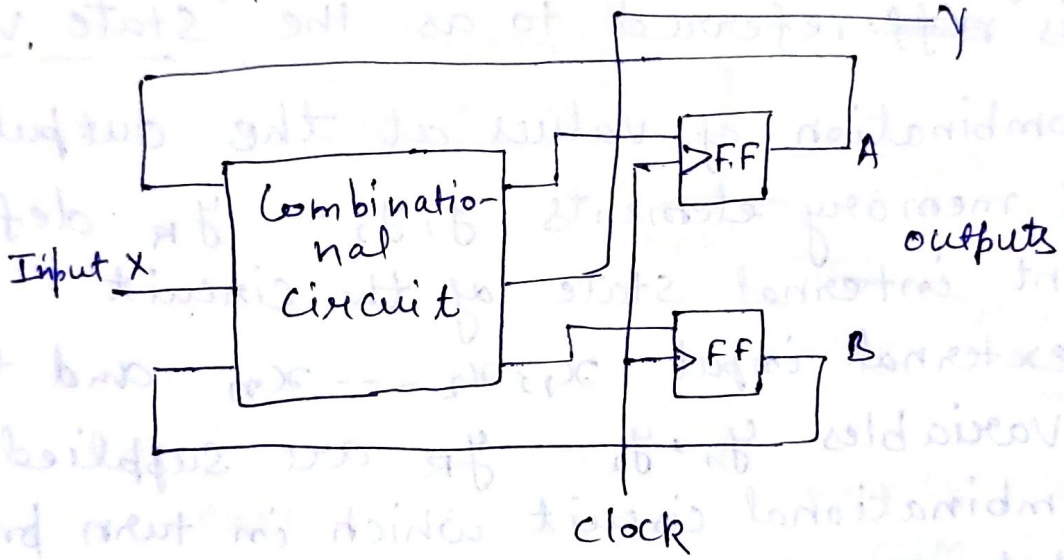


fig (a)
Synchronous sequential Machine

— The memory devices are the flip flop which are clocked simultaneously. The outputs and the Next state of a clocked sequential circuit depend upon the external inputs and the present state of the circuit. There are two commonly used Models of these sequential circuits they are called the Mealy model and the Moore model.

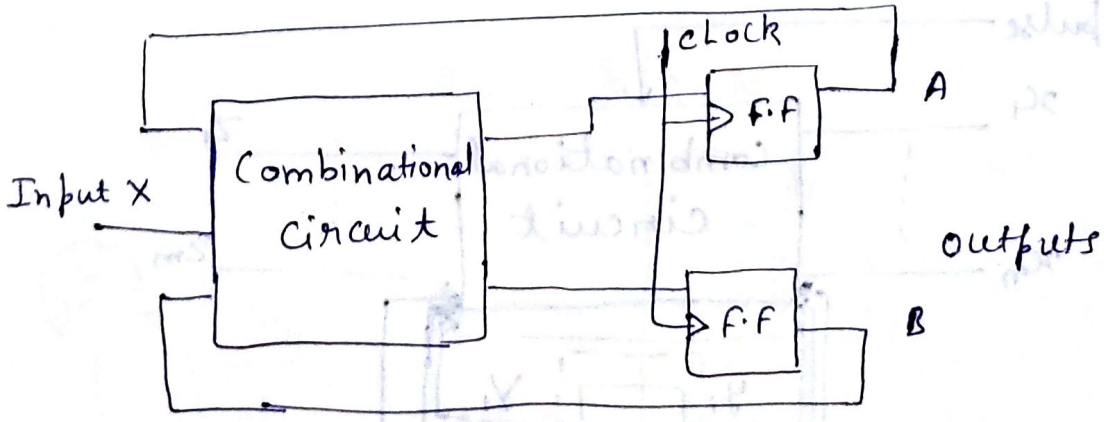
→ In the Mealy Model, the o/p are a function of both the present state and inputs, in the Moore Model the o/p are a function of the present state only.

(1)



(1) Mealy Model

(2)

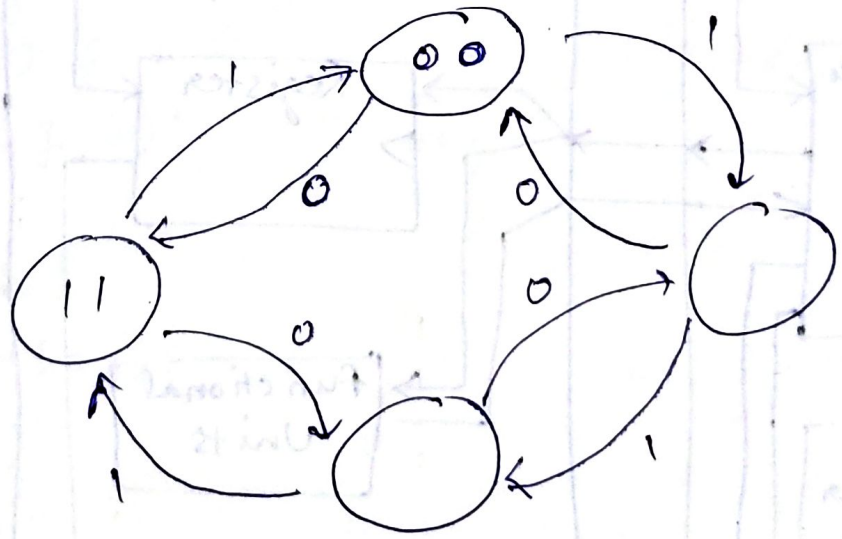


(2) Moore Model

Que If the state table of a sequential circuit is shown draw its state diagram

Present state		Next state			
		X = 0		X = 1	
A	B	A	B	A	B
0	0	1	1	0	1
0	1	0	0	1	0
1	0	0	1	1	1
1	1	1	0	0	0

Sol

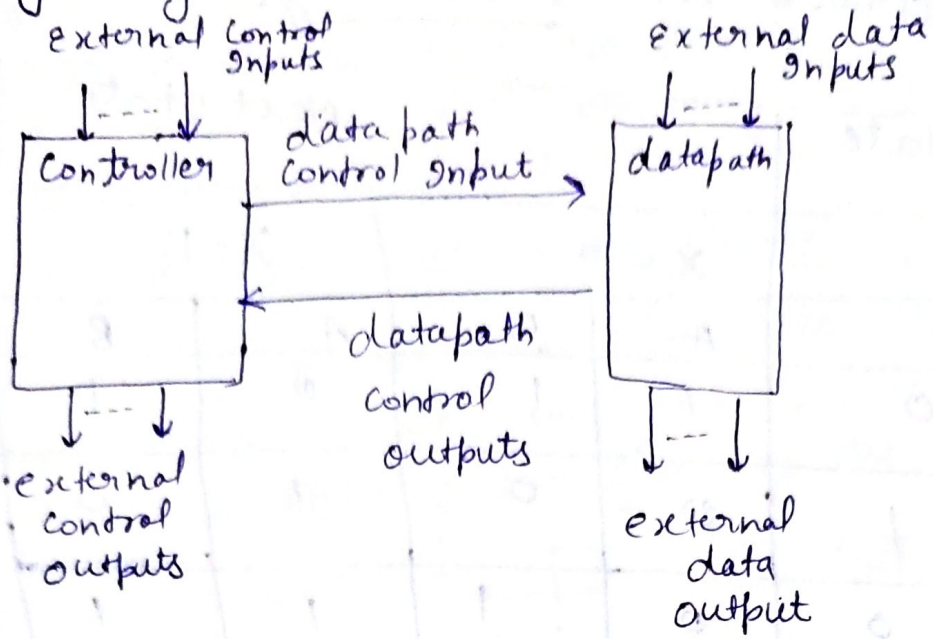


State diagram

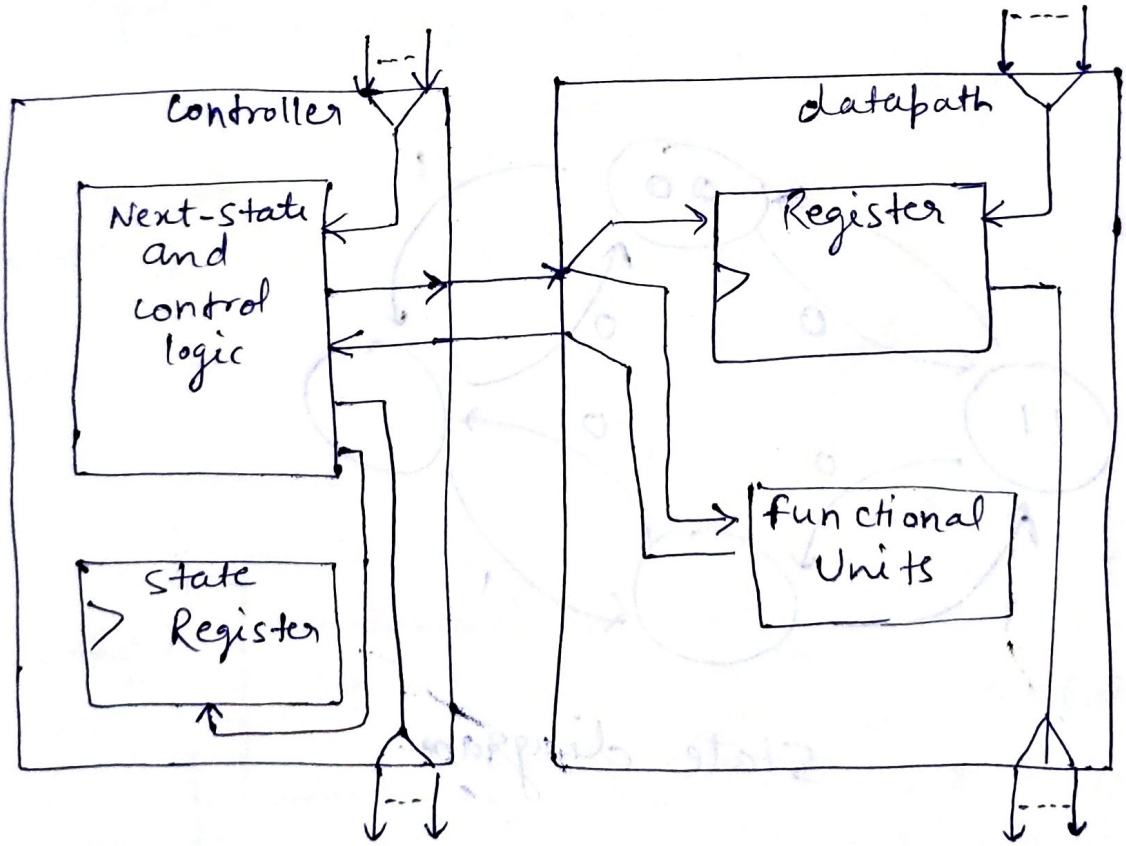
Custom Single-Purpose Processor design →

every processor is a combination of datapath components and controllers.

Let us design a single purpose processor for computing a greatest common divisor (GCD)

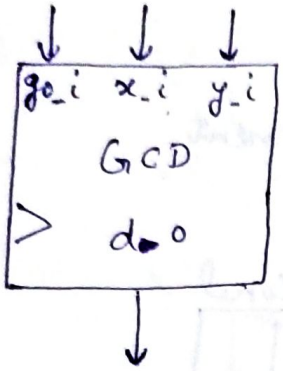


Controller and datapath



A view inside the controller and datapath

→ first create algorithm
→ Convert algorithm to 'complex' state machine
  - Known as FSMD: finite state machine with data Path
  - Can use templates to perform such Conversion



Black-box view

Algo 1

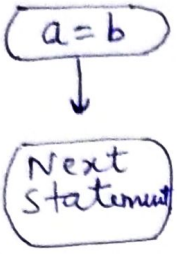
```
0:  int x, y;
1:  while (1) {
2:  while (! go_i);
3:    x = x_i ;
4:    y = y_i ;
5:  while (x != y) {
6:    if (x < y)
7:      y = y - x;
      else
8:      x = x - y;
    }
9:    d_o = x;
  }
```

desired functionality

State diagram templates —

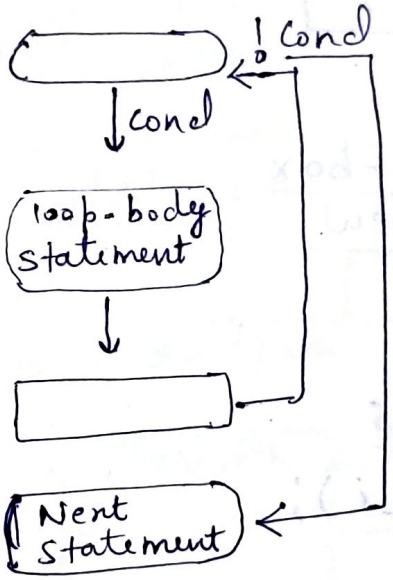
(1) Assignment statement

```
a = b
next statement
```



(2) Loop statement

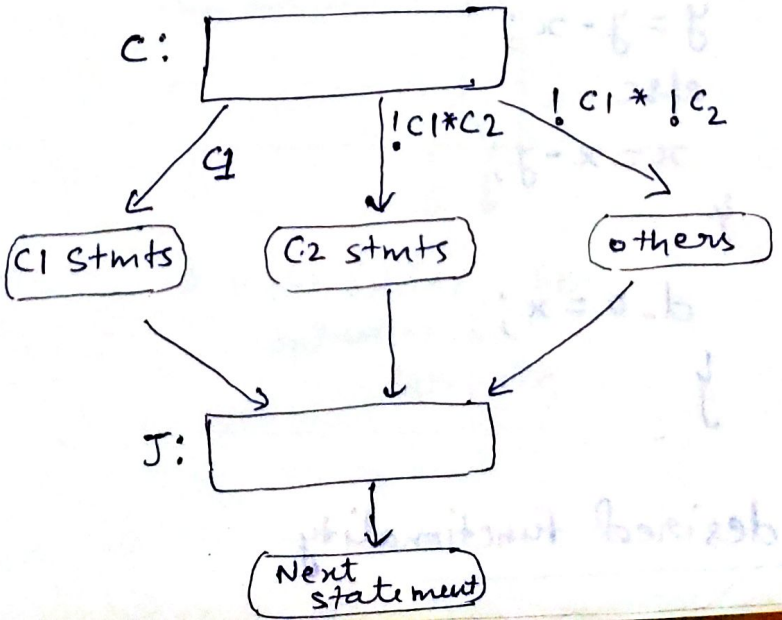
```
while ( cond)
  {
  loop-body-
    statements
  }
Next statement
```



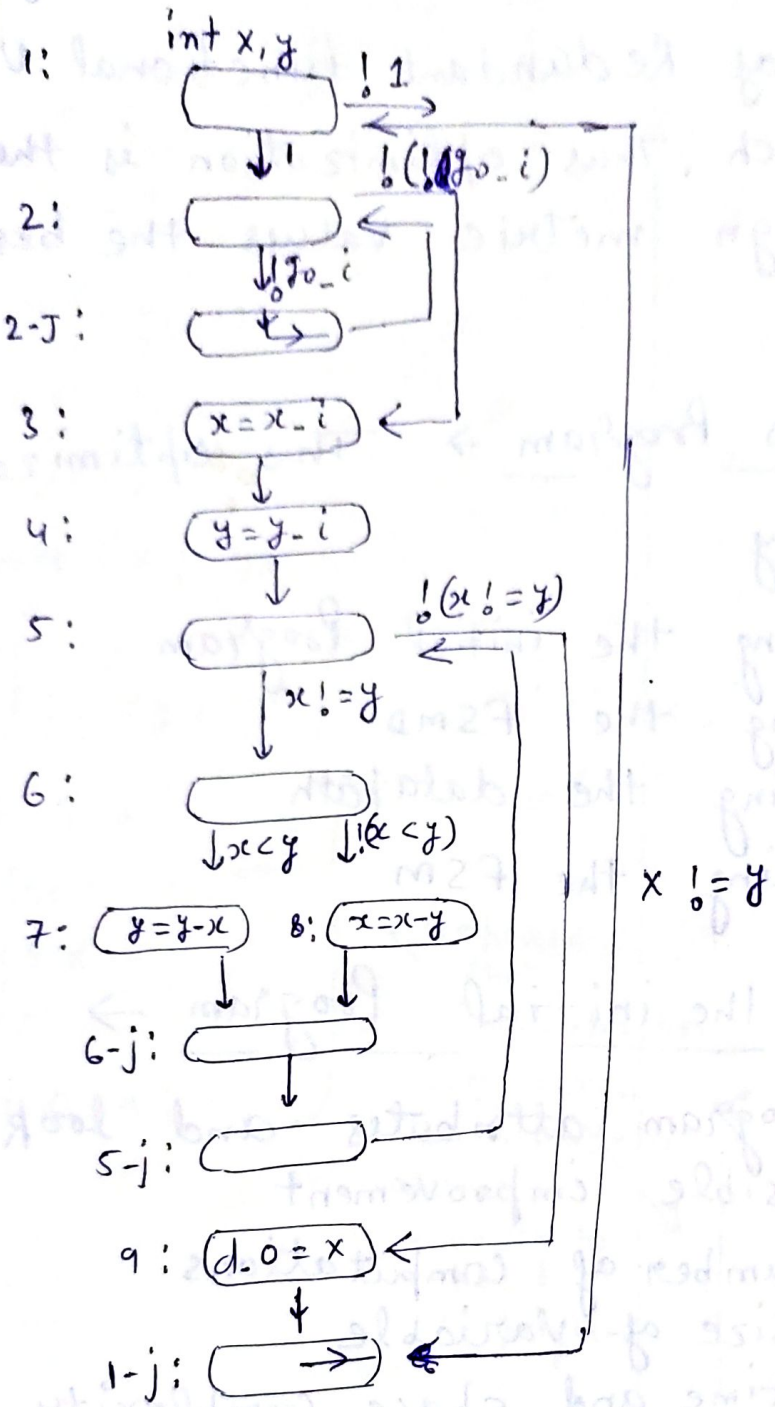
(3) Branch statement

```
if (c1)
   C1 stmts
else if C2
    C2 stmts

else
    other stmts
Next statement
```



state diagram of Algo 1 →



State diagram

Optimizing Single-Purpose Processors →

optimization of SPP is necessary to meet the design challenges. This involves removing some

- Unnecessary states from FSMD to simplify the design. Removal of Redundant functional units can be another approach. Thus optimization is the task of making the design metric values the best possible.

optimizing GCD Program → This optimization can be carried by

(1) optimizing the initial Program
(2) optimizing the FSMD
(3) optimizing the datapath
(4) optimizing the FSM

(1) optimizing the initial Program →

- Analyze Program attributes and look for areas of possible improvement
  - Number of computations
  - Size of variable
  - time and space complexity
  - operations used :- multiplication and division are very expensive

Original program

```
0: int x, y;
1: while (1)
   {
2: while (!go_i);
3: x = x_i;
4: y = y_i;
5: while (x != y)
   {
6: if (x < y)
7: y = y - x;
   else
8: x = x - y;
   }
9: d_o = x;
   }
```

replace the subtraction operation(s) with modulo operation in order to speed up program

optimized program

```
0: int x, y, r;
1: while (1)
   {
2: while (!go_i);
   // x must be the large Number
3: if (x_i >= y_i)
   {
4: x = x_i;
5: y = y_i;
   }
6: else {
7: x = y_i;
8: y = x_i;
   }
9: while (y != 0)
   {
10: r = x % y;
11: x = y;
12: y = r;
   }
13: d_o = x;
   }
```

GCD(42, 8) - 9 iteration to ~~compute~~ complete the loop
x and y values as follows
(42,8), (43,3), (26,8), (18,6)
(10,8), (2,8), (2,6), (2,4)
(2,2)

GCD (42,2) - 3 iteration to complete the loop
x and y values evaluated as follows
(42,2), (8,2), (2,0)

This ~~above~~ algorithm makes use of Modulo operation % and uses fewer steps and far more ~~di~~ efficient in terms in time. The choice of Algorithm can have the biggest impact on the efficiency of the designed Processor.

(2) Optimizing the FSMD → Template based Procedure to convert a Program into FSMD may result in an inefficient FSMD, as this Procedure result in many unnecessary states.

Scheduling is the task of assigning operations from the original program to states in an FSMD. The Scheduling obtained using template-based method (shown in the fig) can be improved

Some states can be merged into one, when there are no loop operations between them. Unwanted states can be removed whose outgoing transitions have constant values.

The optimized (reduced) FSMD with only 6 states (from 13 states) is shown in fig

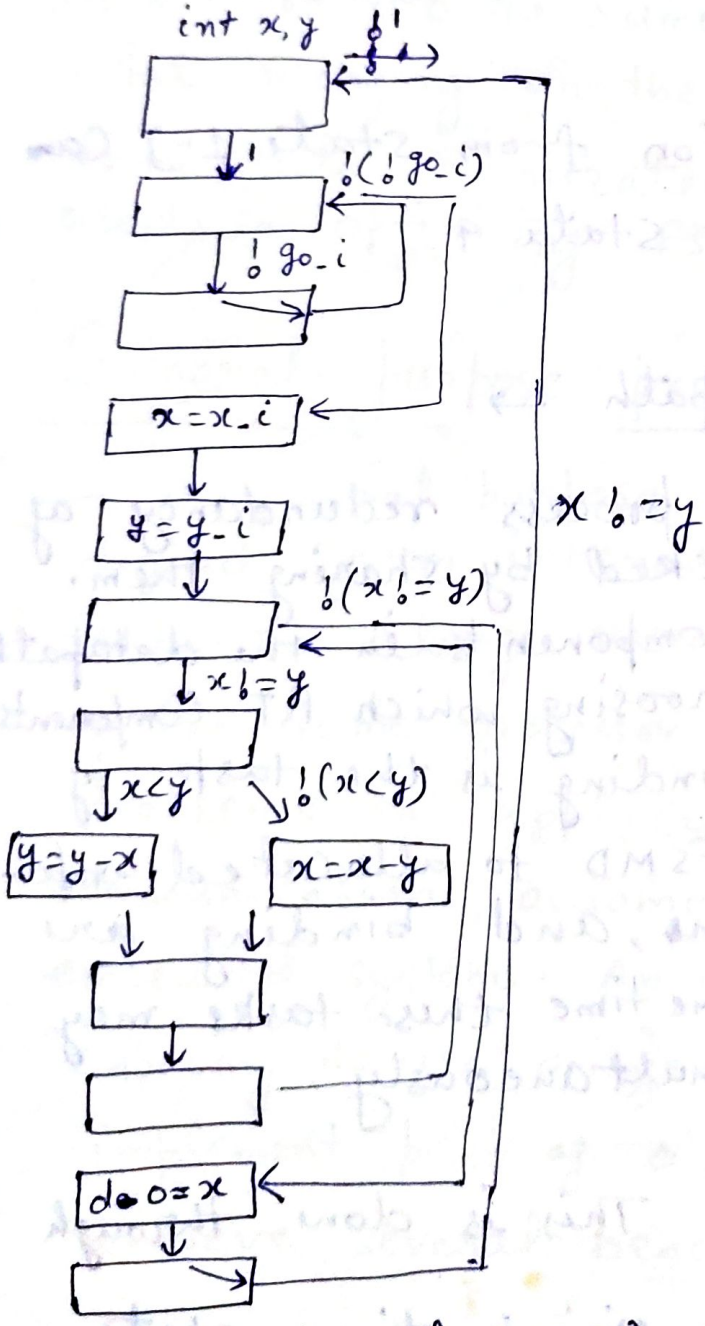
In deciding the Number of states in an FSMD the consequent hardware constraint must also be looked into. for example a particular program statement had the operation $a = b*c*d*e$. generating a single state for this operation will require usage of three multipliers in the datapath multipliers are expensive. To avoid such usage, the

the operation can be broken down into smaller operations like

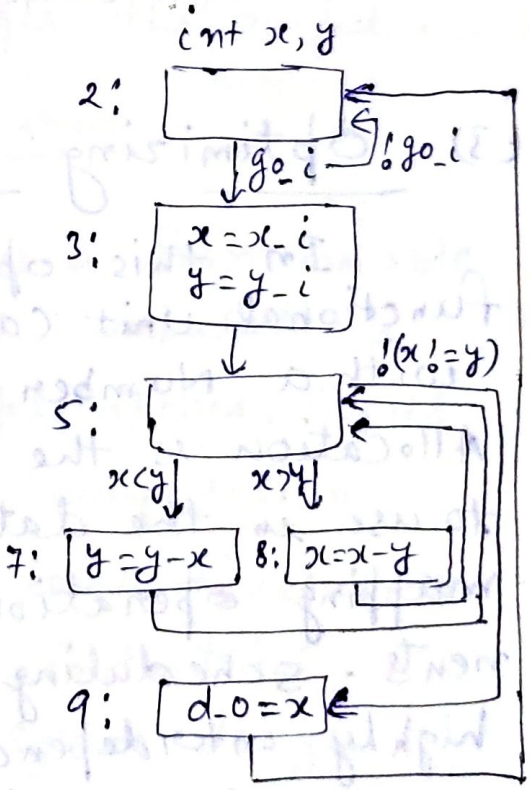
$t_1 = b * c$,

$t_2 = d * e$,

And $a = t_1 * t_2$ with each smaller operation having its own state, Then only one multiplier would be needed in the data path. Multiplication operations can share the multiplier. while optimizing time constraints must also should be considered.



(original FSMD)



optimized FSMD

Steps : →

— eliminate state 1 → transitions have constant values

— merge State 2 and State 2J — no loop operation in between them

— merge state 3 and state 4 — assignment operation are independent of one another

— merge state 5 and state 6 — transitions from State 6 can be done in state 5

— eliminate 5J and 6J — transition from each state can be done from state 7 and state 8 respectively.

— eliminate 1-J - transition from state 1-J can be done directly from state 9

(3) Optimizing the datapath →

In this optimization process redundancy of functional unit can be checked by sharing them, with a Number of RT components in the datapath Allocation is the task of choosing which RT Components to use in the data path. Binding is the task of mapping operations from FSMD to allocated components. scheduling allocations, and binding are highly interdependent. Some time these tasks may have to be considered Simultaneously.

(4) Optimizing FSM → This is done through State encoding and state minimization. state

encoding is the task of assigning a unique bit pattern to each state in an FSM. CAD Tools can be of great aid in searching for the best encoding that decides the size of the state Register and the size of the combinational logic

State minimization is the task of merging equivalent states into a single state. Two states are equivalent if for all possible input combination those two states generate the same output and transition to the same next state.

State merging on the other hand is different from state minimization. state merging that is used in optimizing FSMD changes the output.

General-purpose processors →

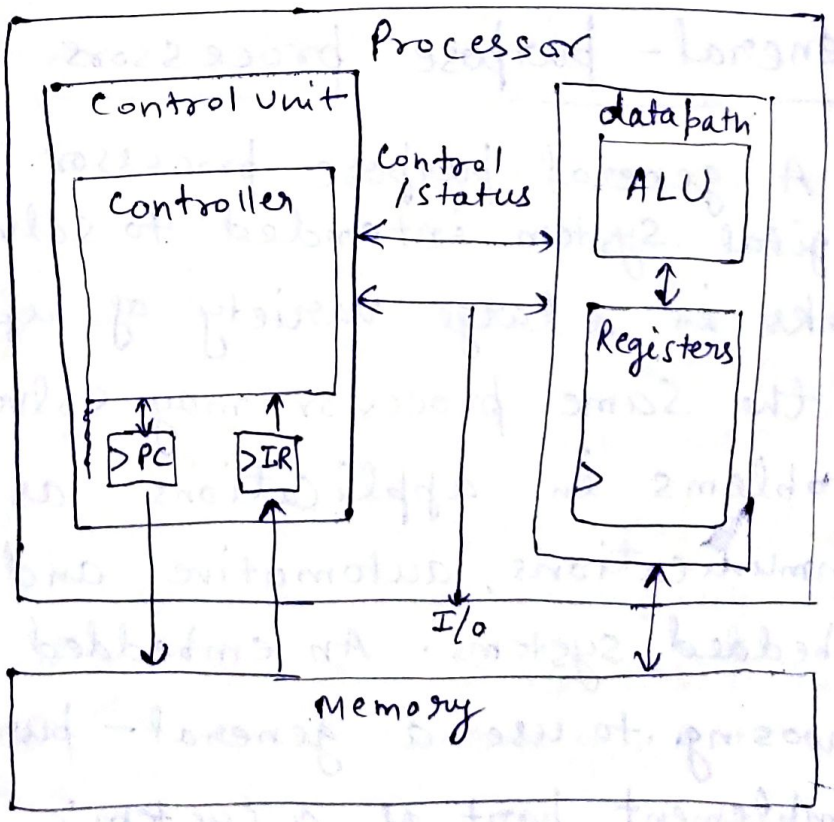
— A general purpose processor is a Programmable digital system intended to solve computations tasks in a large variety of applications. Copies of the same processor may solve computation Problems in applications as ~~doing~~ diverse as communications, automotive and industrial embedded systems. An embedded system designer choosing to use a general-purpose processor to implement part of a system's functionality may achieve several benefits.

— first the unit cost of the processor may be very low, often a few dollars or less

— second, because the processor manufacturer can spread NRE cost over large numbers of units the manufacturer can afford to invest large NRE cost into the processor's design, without significantly increasing the unit cost

— Third the embedded system designer may incur low NRE cost, since the designer need only write software, and then apply a complier and /or an Assembler, both of which are mature and low-cost technologies.

Basic Architecture →



— Control unit and datapath →
Not similar to single purpose Processor

Key Differences →

→ Data path is general

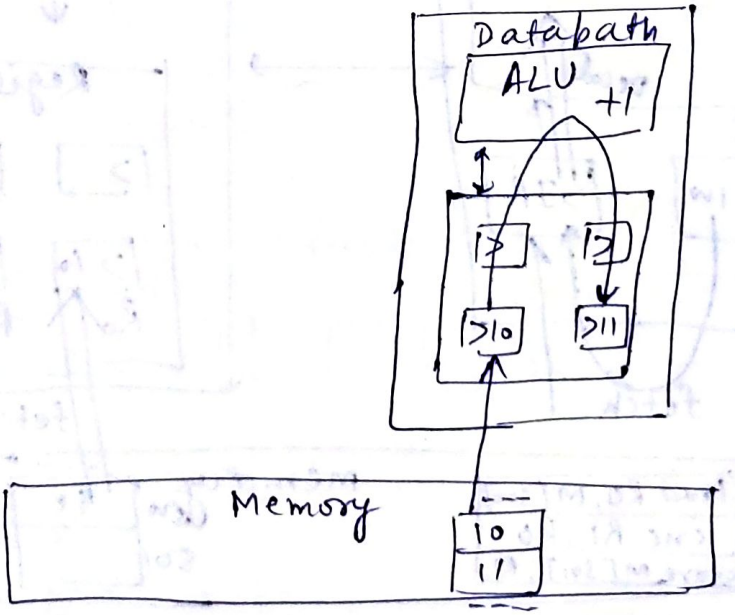
— control Unit doesn't store the Algorithm — the Program Algorithm is "Programmed" into the memory.

Datapath operations →

— Load —
  - Read memory location into Register

— ALU operation
  — Input certain registers through ALU, store back in register

— Store
  — write register to memory location

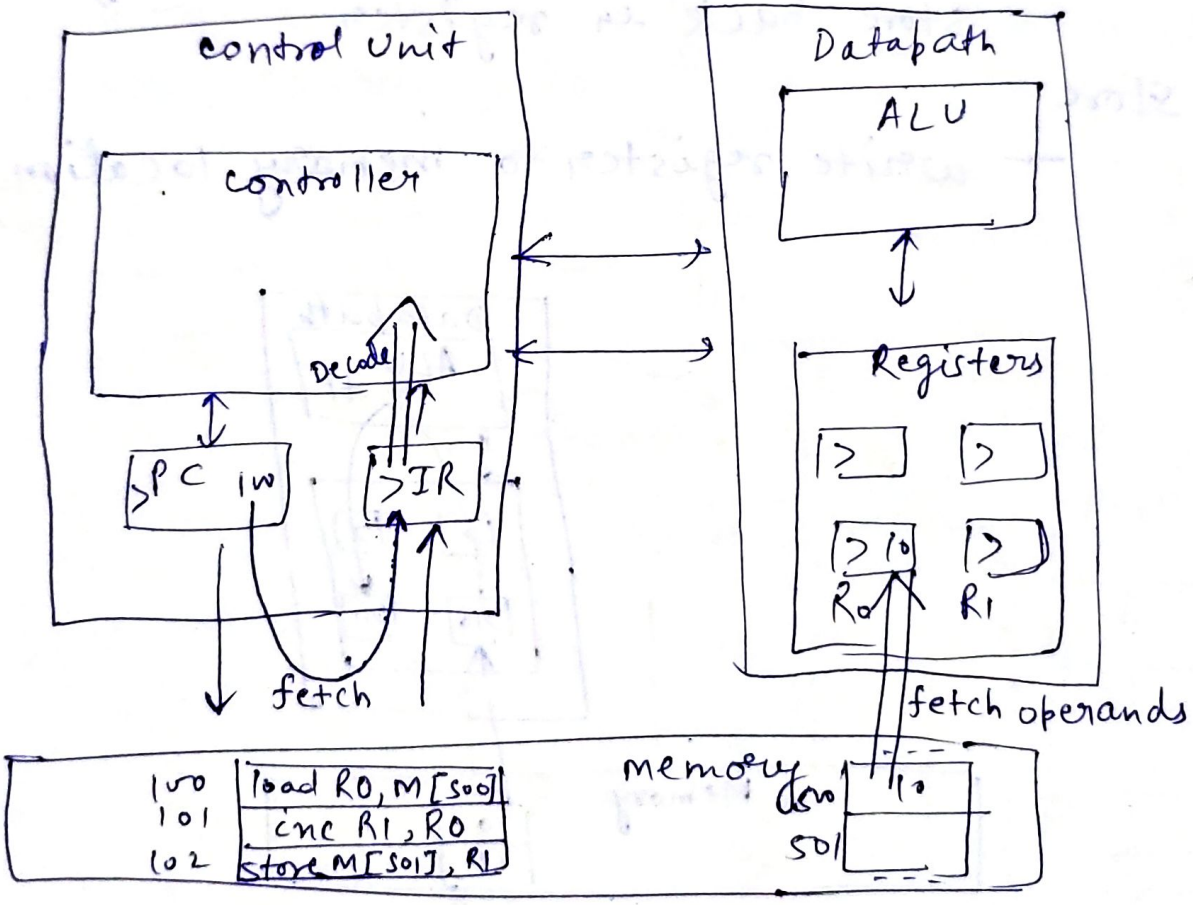


Control unit → control unit: configures the data path operations

— Sequence of desired operations ("instructions") stored in memory — Program

— Instruction cycle — broken into several sub-operations, each one have clock cycle e.g.

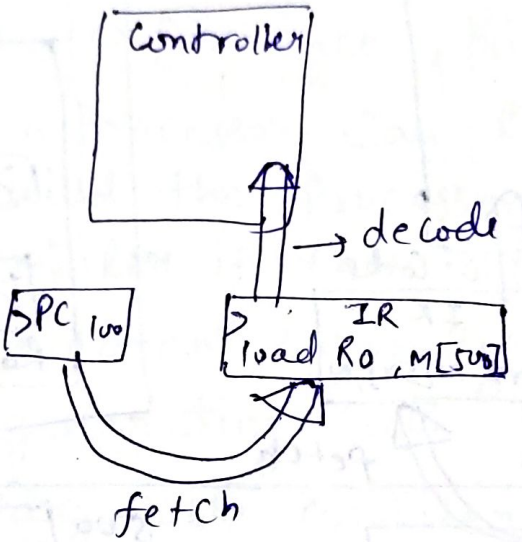
- fetch: get Next instruction into IR
- decode: Determine what the instruction means
- fetch operands: move data from memory to data path register
- execute: Move data through the ALU
- store results: write data from Register to memory



Instruction cycles →

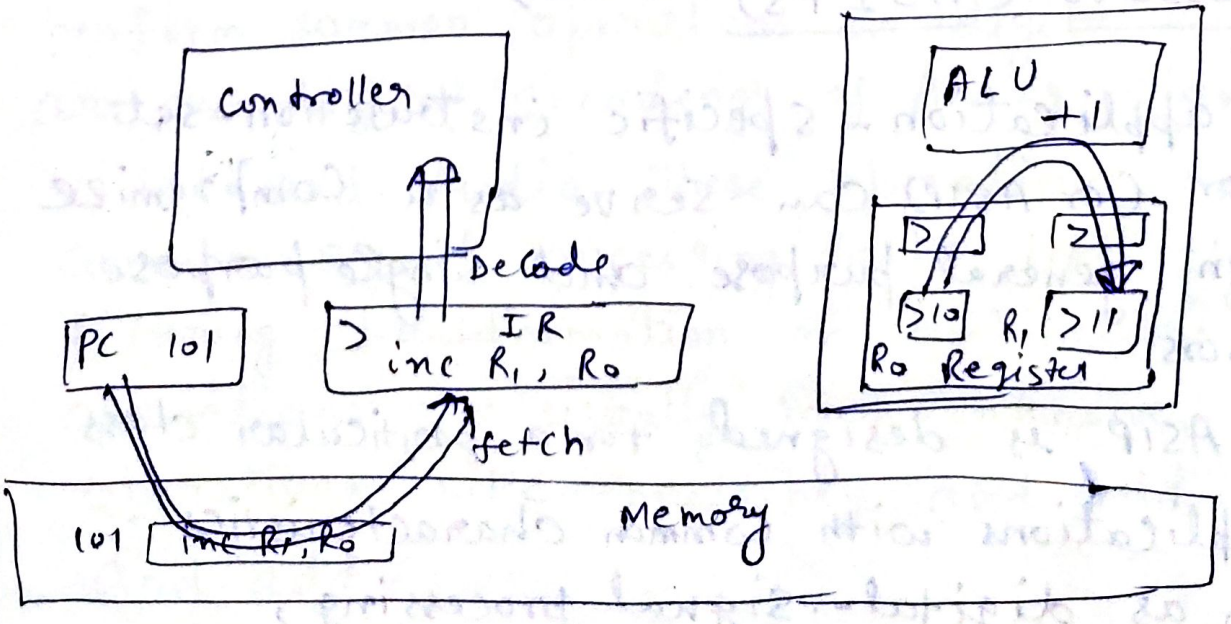
(1)

PC = 100



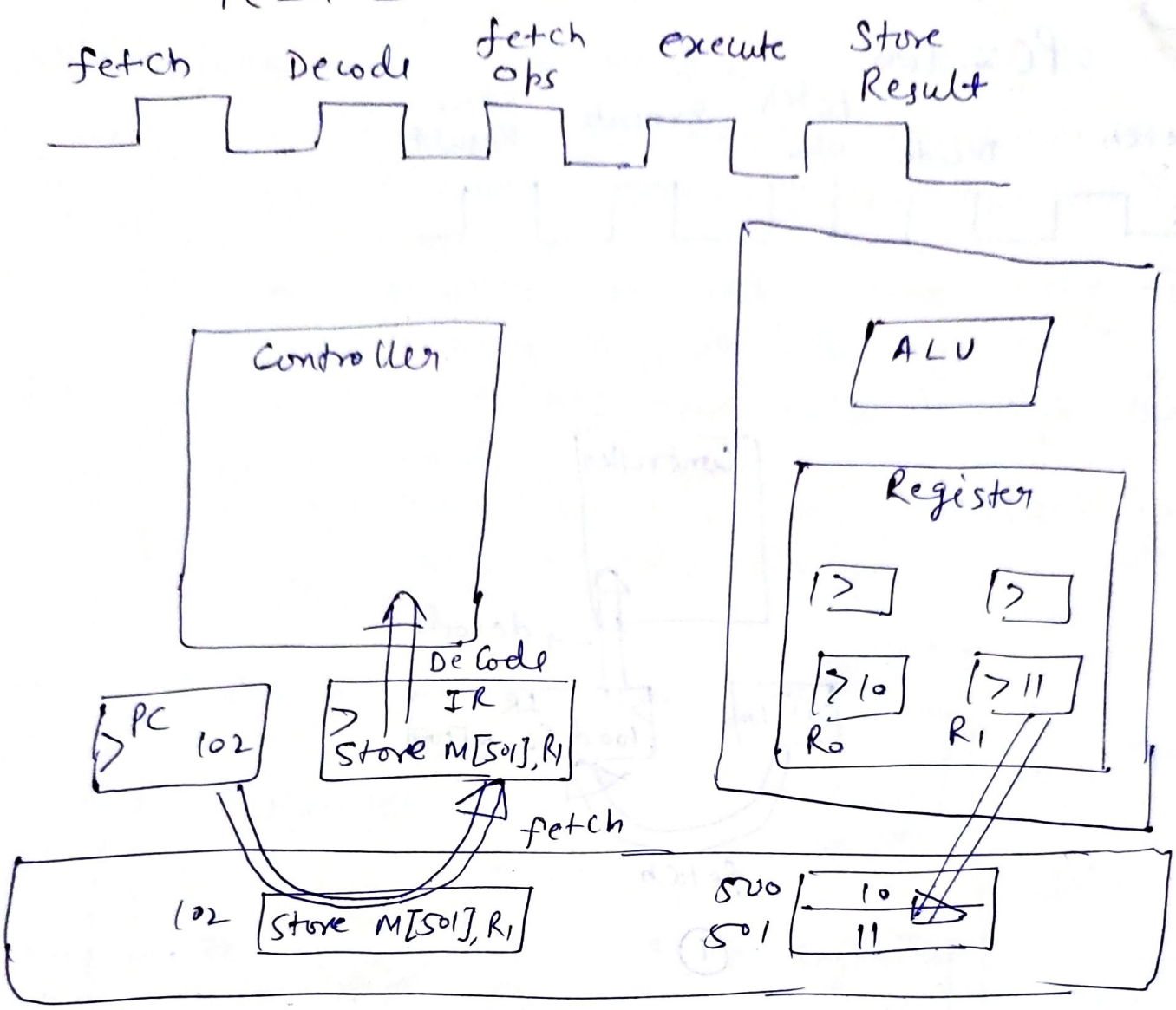
(1)

(2) PC = 101



(2)

(3)



(3)

Application-specific Instruction set Processors (ASIPs) ⟶

⟶ An application-specific instruction-set Processor (or ASIP) can serve as a compromize between general purpose and single purpose processors.

⟶ An ASIP is designed for a particular class of applications with common characteristics such as digital-signal processing,

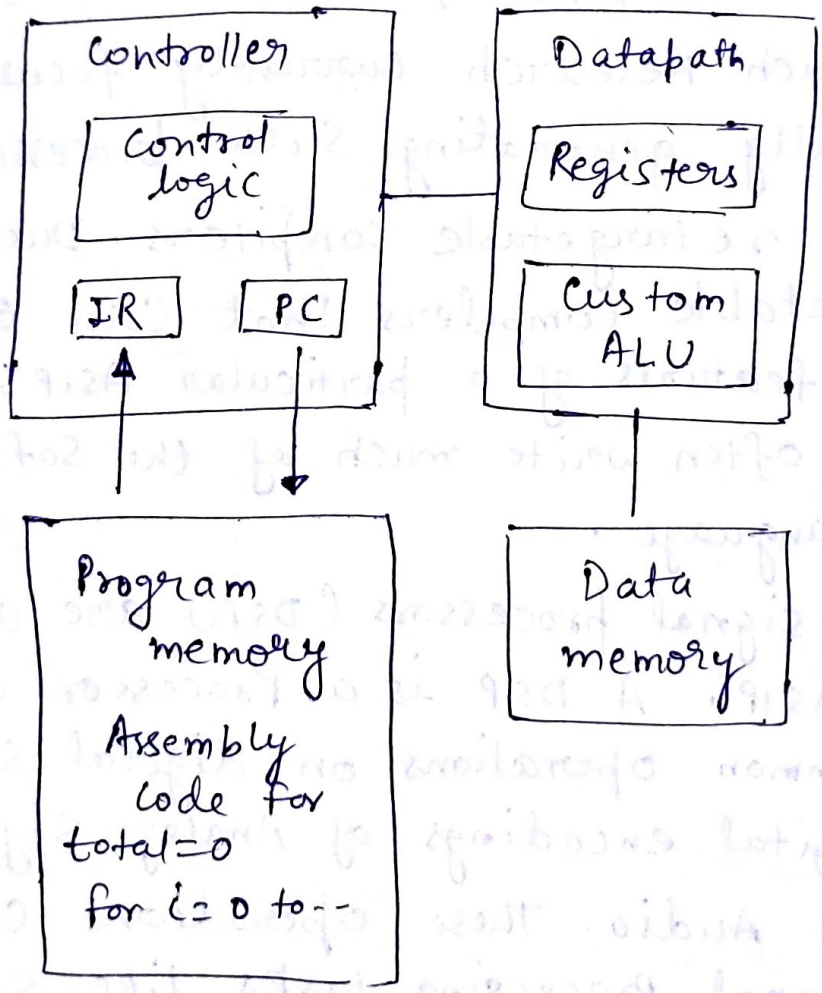
Tele communication, embedded control etc.

— The designer of such a processor can optimized the datapath for the application class, perhaps adding special functional units for common operations and eliminating other infrequently used units.

— using an ASIP in an embedded system can provide the benifits of flexibility while still achieving good performance, power and size. However such processor can require large NRE cost to build the processor itself, and to build a compiler if these items don't already exist. Much Research currently focuses on automatically generating such processors and associated retargetable compilers. Due to the lack of retargetable compilers that can exploit the unique features of a particular ASIP, designers using ASIPs often write much of the software in assembly language.

— Digital-signal processors (DSPs) are a common class of ASIP. A DSP is a Processor designed to perform common operations on digital signals, which are the digital encodings of Analog signals like video and Audio. These operations carry out common signal processing tasks like signal filtering; transformation or combination. Such operations are usually math-intensive, including operations like multiply and add or shift and add.

To support such operations, a DSP may have special purpose datapath components such a multiply-Accumulate unit. which can perform a computation like T = T + M[i] * k using only one instruction. Because DSP Programs often manipulate large array of data. A DSP may also include special hardware to fetch sequential data memory locations in parallel with other operations. to further speed execution.



Application Specific processors

DSP Chips →

DSP Processors are microprocessors designed to perform digital signal processing - the Mathematical manipulation of digitally represented signals.

Digital signal processing is one of the core technology in rapidly growing application area such as wireless communication, audio and video processing and industrial control.

- Most DSP processors share some common basic features designed to support high-performance repetitive, numerically intensive tasks.
- The most often cited of these features is the ability to perform one or more multiply accumulate operation is useful in DSP algorithm that involve computing a vector dot product.

  Such as digital filters, correlation and fourier transforms
- DSP processors integrate multiply-accumulated hardware into the main data path of the Processor
- Some Recent DSP processors provide two or more multiply-Accumulate Units. allowing multiply-Accumulate operations to be performed in parallel.

— DSP Processors generally provide extra "guard" bits in the accumulator. for example the Motorola DSP Processor family ~~examined~~

— A second feature shared by DSP processors is the ability to complete several accesses to memory in a single instruction cycle. this allows the processor to fetch an instruction while simultaneously fetching operands and/or storing the result of a previous instruction to memory.