Chapter 4

CPU DESIGN

Assistant Professor

Er. Shiva Ram Dam

Contents:

1. Introduction to CPU
2. Specifying a CPU
3. Design and Implementation of a Very Simple and Relatively Simple CPU
4. Instruction Execution, Fetch, Decode , Data path
5. ALU Design
6. Designing Hardwired Control Unit
7. Design Verification

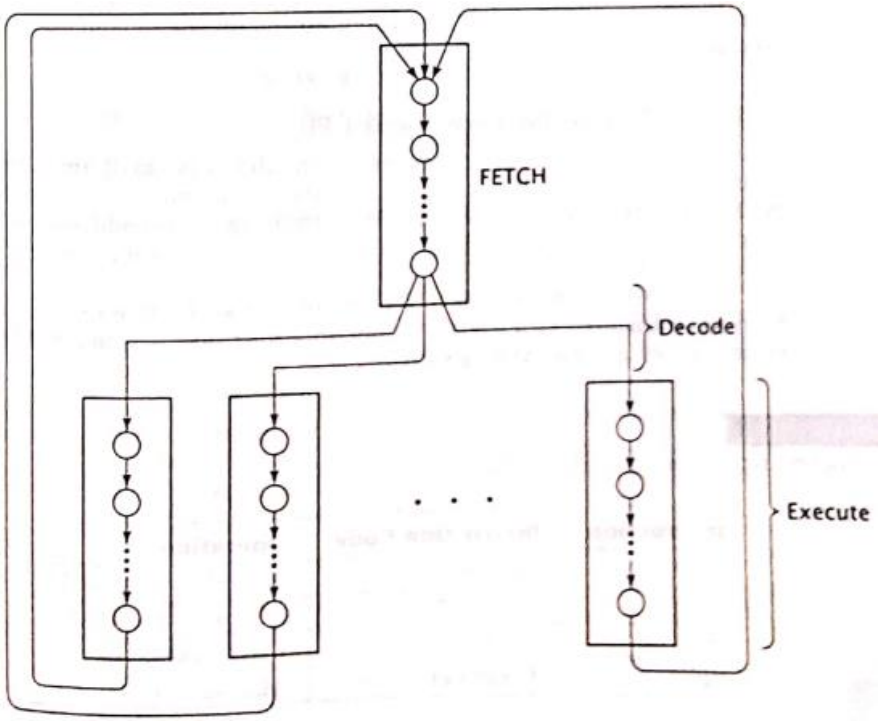
4.1 Introduction to CPU

- ALU + Registers + Control Unit
- Performs sequences of microoperations needed to perform fetch, decode and execute cycles of every instructions in CPU instruction set.

4.2 Specifying a CPU

- The first step in designing CPU is to **determine its applications**. The key is to match the capabilities of the CPU to the tasks it will perform.
- Second step is to **design a Instruction Set Architecture** (ISA) which can handle the given tasks.
- Third step is to **design State diagram for** CPU. Here the different microoperation are shown which are performed during each stat and the conditions that cause CPU to go from one state to another. CPU is a complex Finite State Machine.
- Then, specifying the steps the CPU must perform in order to fetch, decode and execute every instructions in its instruction set.

- In general, a CPU performs following sequence of operations.
  1. **Fetch Cycle:** Fetching an instruction from memory, then go to decode cycle.
  2. **Decode Cycle:** Decoding an instruction- that is, determine which instruction has been fetched.
  3. **Execute cycle:** Executing the instruction . Then go to the fetch cycle and fetch the next instruction.



Generic CPU state diagram

4.3 Design and Implementation of a Very Simple and Relatively Simple CPU

- To illustrate the CPU design process,
- lets us consider:
  - A small and somewhat impractical CPU.
  - It can access 64 bytes of memory, each byte being 8 bits wide.
  - The CPU does this by outputting a 6-bit address on its output pins A[5..0], and reading in the 8-bit value from memory on its inputs D[7…0].
  - The CPU has: one programmable register, 8-bit accumulator (AC).
  - It has only four instructions in its instruction set.

Instruction	Instruction Code	Operation
ADD	00AAAAAA	AC←AC + M[AAAAAA]
AND	01AAAAAA	AC←AC ^ M[AAAAAA]
JMP	10AAAAAA	GOTO AAAAAA
INC	11XXXXXX	AC←AC + 1

Instruction set for the Very Simple CPU

Design and Implementation of a Very Simple and Relatively Simple CPU

- In addition to AC, the CPU contains the following registers:
  1. **Address Register (AR):**
     - 6-bit
     - Supplies an address to memory via A[5..0].
  2. **Program Counter(PC):**
     - 6-bit
     - Contains the address of the next instruction to be executed.
  3. **Data Register(DR):**
     - 8-bit
     - Receives instructions and data from memory via D[7…0]
  4. **Instruction Register(IR):**
     - 2-bit
     - Stores the opcode portion of the instruction code fetched from memory.

4.4 Instruction Execution, Fetch, Decode and Datapath

- Fetching instructions from memory
- Decoding instructions
- Executing instructions
- Constructing Datapath

1. Fetching instructions from Memory.

- The address of instruction to be fetched is stored in the PC. Since, address register (AR) supplies an address to memory through A [5……0] pins
- Firstly, copy the content of PC to AR.

  Fetch 1: **AR ← PC**

- The CPU reads the instructions from the memory and store in Data Register (DR).

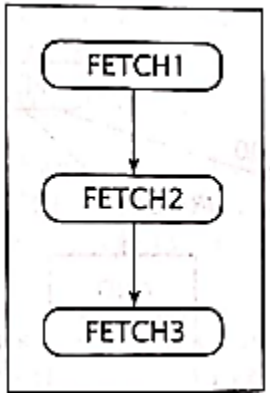
  Fetch 2: **DR ←M**

  PC ←PC + 1

- Finally, two higher order bits of DR are copied into Instruction Register (IR); these two bits indicate which instruction is to be executed, and the CPU copies the lower order 6-bit of DR into AR.

  Fetch 3: **IR ←DR [7,6]**

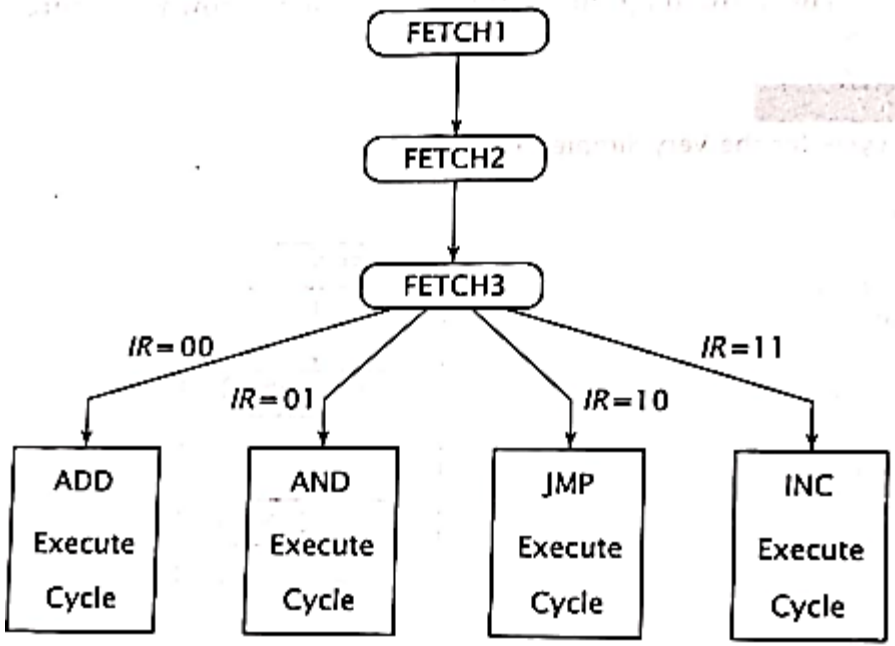
  AR ← DR [5….0]



Fetch cycle for the Very Simple CPU

2. Decoding instructions

- After CPU has fetched an instruction from memory, it must determine which instruction it has fetched so that it may invoke the correct **execute routine.**
- For this CPU, there are four instructions and thus, four executer routines.
- The value in IR (00,01,10 or 11) determines which execute routine is invoked.



Fetch and decode cycles for the Very Simple CPU

3. Executing instructions

- To complete the state diagram for this CPU, we must develop the state diagram for each execute routine.
- Now we design the portion of the state diagram for each routine and the overall design for the CPU.
- The state diagrams for the individual execute routines are fairly simple, so they are only included in the diagram of the finite state machine for the entire CPU.

a) ADD instruction

- To perform ADD instruction, CPU must :
- First, fetch one operand from memory.
- Then add this operand to current contents in AC and store result back in the AC.

ADD1: DR←M

ADD2: AC←AC+DR

b) AND instruction

- Virtually similar to ADD instruction.
- First, fetch an operand from memory.
- Logically AND the two values and store it in AC.

AND1: DR←M

AND2: AC← AC^DR

c) JMP instruction

- The address to which the CPU must jump is copied into the PC.
- Then CPU fetches the next instruction.

JMP1: PC← DR [5..0]

d) INC instruction

- CPU simply adds 1 to contents of AC and goes to fetch routine.

INC1: AC← AC + 1

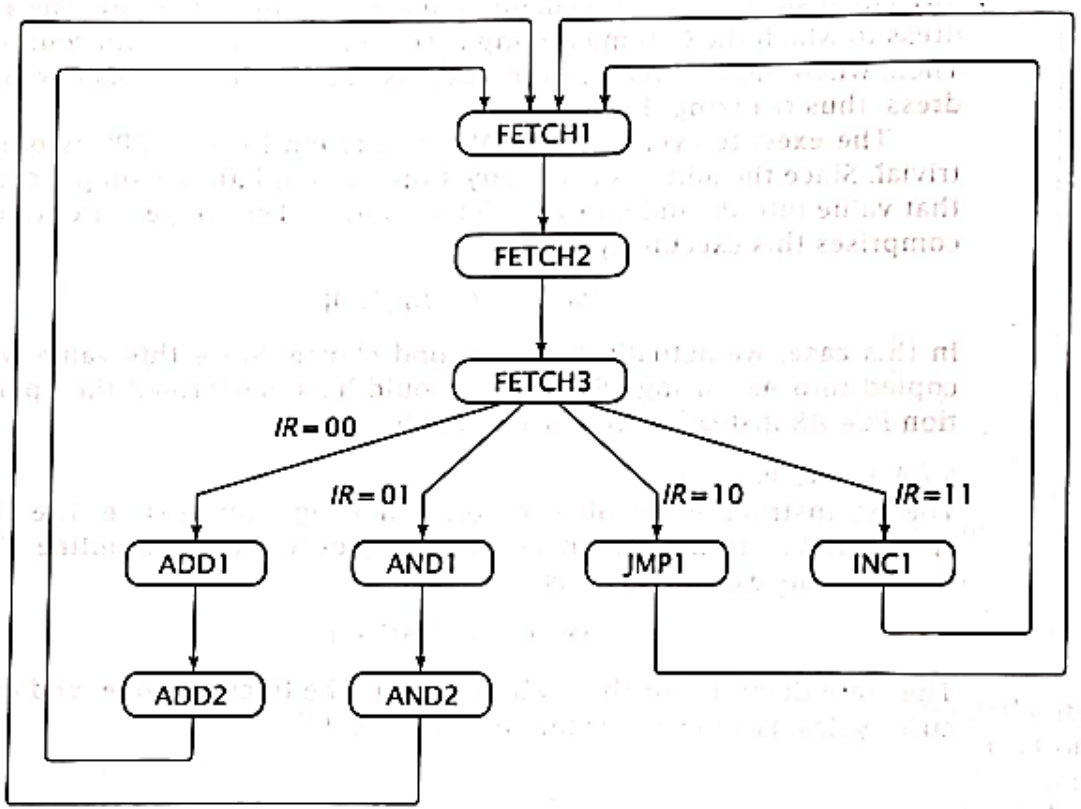
Establishing required Data paths

- The state diagram and register transfers specify what must be done in order to realize this CPU.
- Now we must design the CPU so that it actually does these things.
- First, we look at what data transfers can take place and design the internal data paths of the CPU so this can be done.

- The operations associated with each state for this CPU are:

FETCH1: $AR \leftarrow PC$
FETCH2: $DR \leftarrow M$, $PC \leftarrow PC + 1$
FETCH3: $IR \leftarrow DR[7..6]$, $AR \leftarrow DR[5..0]$
ADD1: $DR \leftarrow M$
ADD2: $AC \leftarrow AC + DR$
AND1: $DR \leftarrow M$
AND2: $AC \leftarrow AC \wedge DR$
JMP1: $PC \leftarrow DR[5..0]$
INC1: $AC \leftarrow AC + 1$

FETCH1: $AR \leftarrow PC$
FETCH2: $DR \leftarrow M$, $PC \leftarrow PC + 1$
FETCH3: $IR \leftarrow DR[7..6]$, $AR \leftarrow DR[5..0]$
ADD1: $DR \leftarrow M$
ADD2: $AC \leftarrow AC + DR$
AND1: $DR \leftarrow M$
AND2: $AC \leftarrow AC \wedge DR$
JMP1: $PC \leftarrow DR[5..0]$
INC1: $AC \leftarrow AC + 1$



Complete state diagram for the Very Simple CPU

- To design the data paths, we can take one of two approaches.
  - Create **direct paths** between each pair of components that transfer data.
  - Create a **bus** within the CPU and route data between components via the bus.

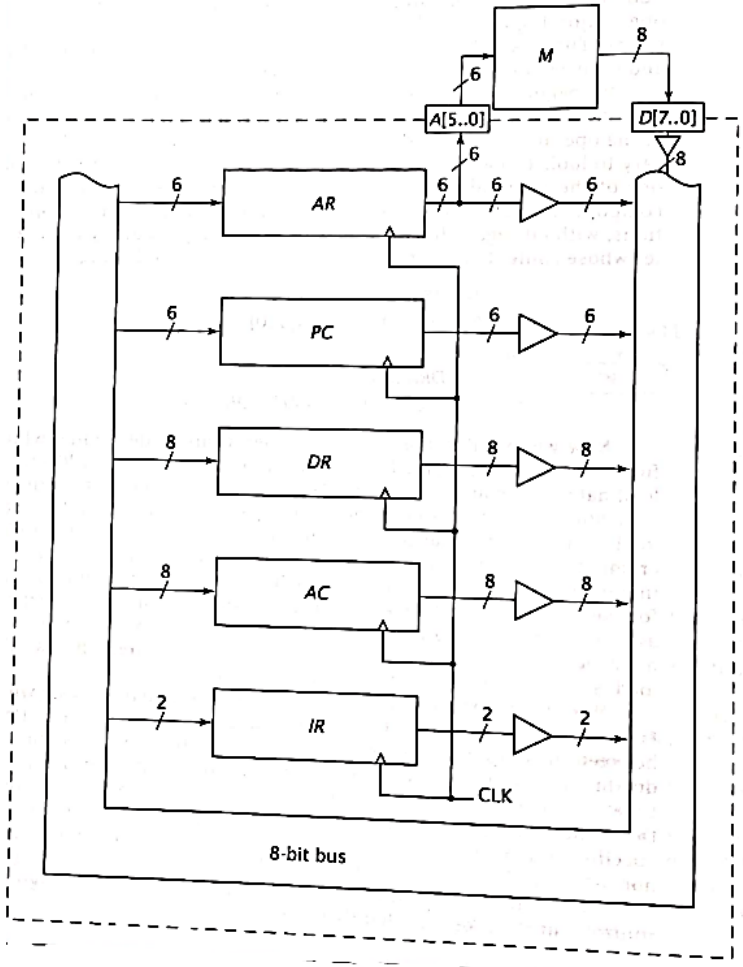
- Now we regroup the operations, without regard for the cycles in which they occur, by the register whose contents they modify. This result in the following:

FETCH1: $AR \leftarrow PC$
FETCH2: $DR \leftarrow M$, $PC \leftarrow PC + 1$
FETCH3: $IR \leftarrow DR[7..6]$, $AR \leftarrow DR[5..0]$
ADD1: $DR \leftarrow M$
ADD2: $AC \leftarrow AC + DR$
AND1: $DR \leftarrow M$
AND2: $AC \leftarrow AC \wedge DR$
JMP1: $PC \leftarrow DR[5..0]$
INC1: $AC \leftarrow AC + 1$

AR: $AR \leftarrow PC$; $AR \leftarrow DR[5..0]$
PC: $PC \leftarrow PC + 1$; $PC \leftarrow DR[5..0]$
DR: $DR \leftarrow M$
IR: $IR \leftarrow DR[7..6]$
AC: $AC \leftarrow AC + DR$; $AC \leftarrow AC \wedge DR$; $AC \leftarrow AC + 1$

AR: $AR \leftarrow PC$; $AR \leftarrow DR[5..0]$
PC: $PC \leftarrow PC + 1$; $PC \leftarrow DR[5..0]$
DR: $DR \leftarrow M$
IR: $IR \leftarrow DR[7..6]$
AC: $AC \leftarrow AC + DR$; $AC \leftarrow AC \wedge DR$; $AC \leftarrow AC + 1$

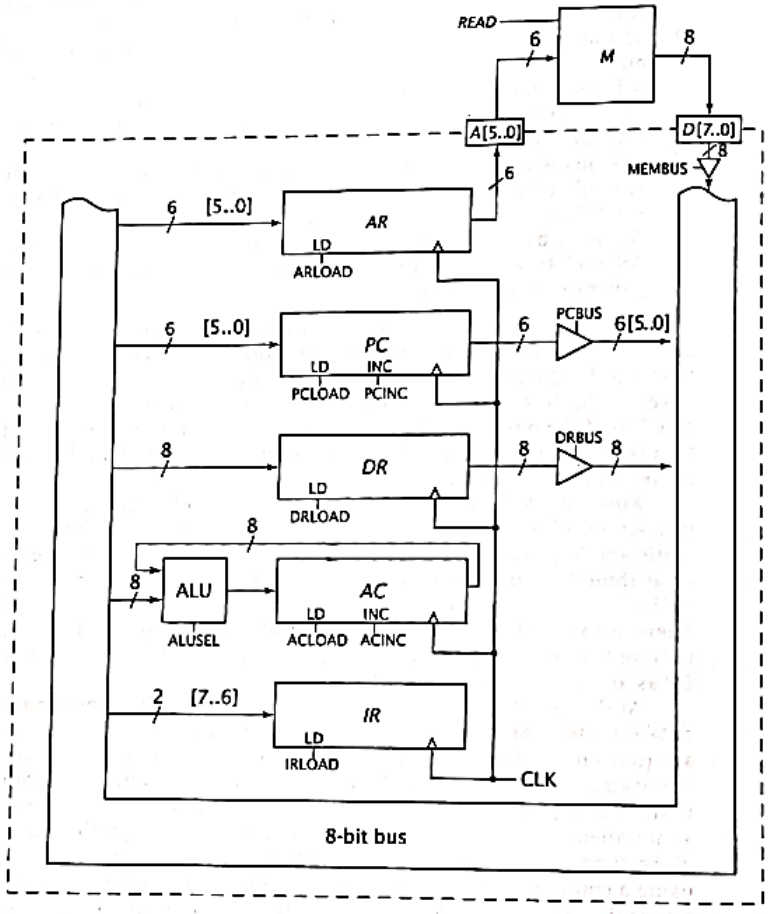
- AR, DR and IR always load data from some other component.
- We connect every component to the system bus, as shown in figure.
- We include tri-state buffers to restrict data being placed all the times by everyone.
- Also, the output of AR are connected to pins A[5..0], as required in the CPU specification.



Preliminary register section for the Very Simple CPU

AR: AR←PC; AR←DR[5..0]
PC: PC←PC + 1; PC←DR[5..0]
DR: DR←M
IR: IR←DR[7..6]
AC: AC←AC + DR; AC←AC ^ DR; AC←AC + 1

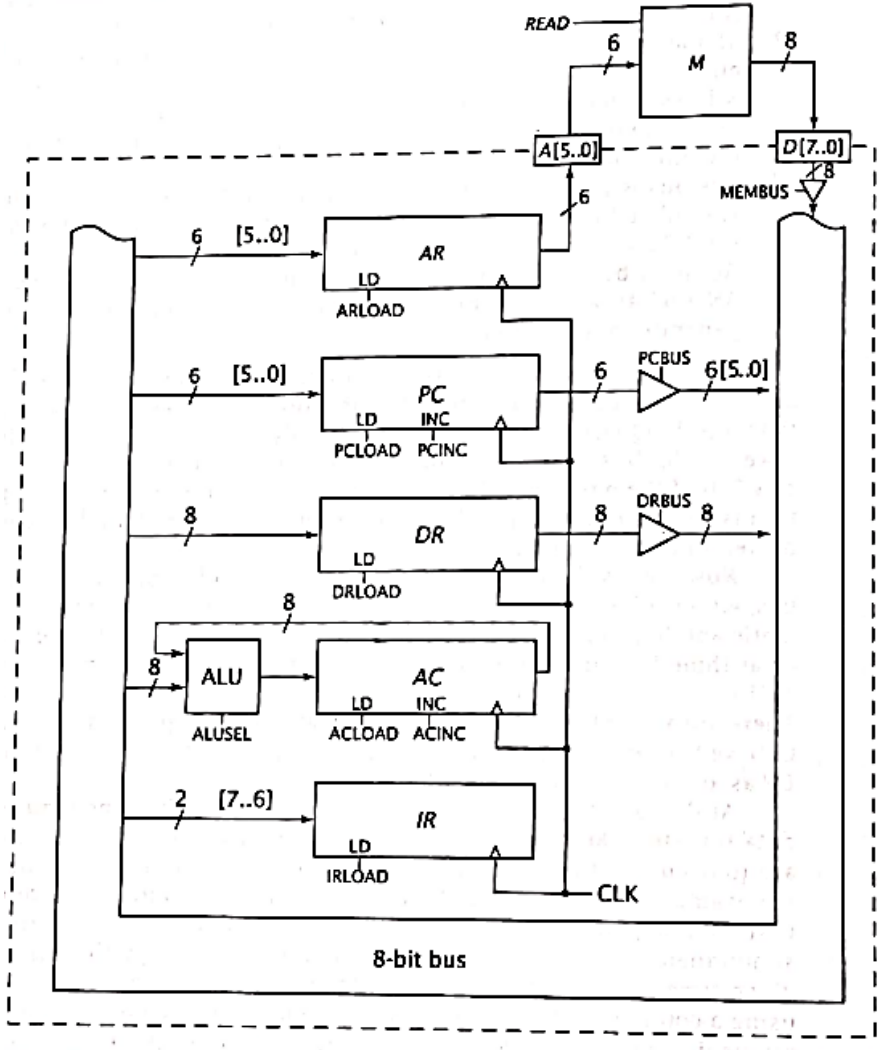
- AC can load in one of two values, either AC+DR or AC^DR.
- The CPU must incorporate some arithmetic and logic circuitry to generate these values.
  - When ALUSEL = 0, the output of ALU is arithmetic sum of its input
  - When ALUSEL = 1, the output of ALU is logical AND of its input.



Final register section for the Very Simple CPU

4.5 ALU Design

- The circuit diagram for the relatively simple CPU is shown in figure below:



ALU for relatively simple CPU

- The ALU for this CPU performs only two functions:
  - Adds its two inputs or
  - logically ANDs its two inputs.
- The simplest way to design this ALU is to create separate hardware to perform each function and then use a multiplexer to output one of the two results.
  - The addition is implemented using a standard 8-bit parallel adder.
  - The logical AND operation is implemented using eight bits 2-input AND gates.
  - The outputs of the parallel adder and the AND gates are input to an 8-bit 2 to 1 multiplexer.
  - The control unit of the MUX is called S (for select).

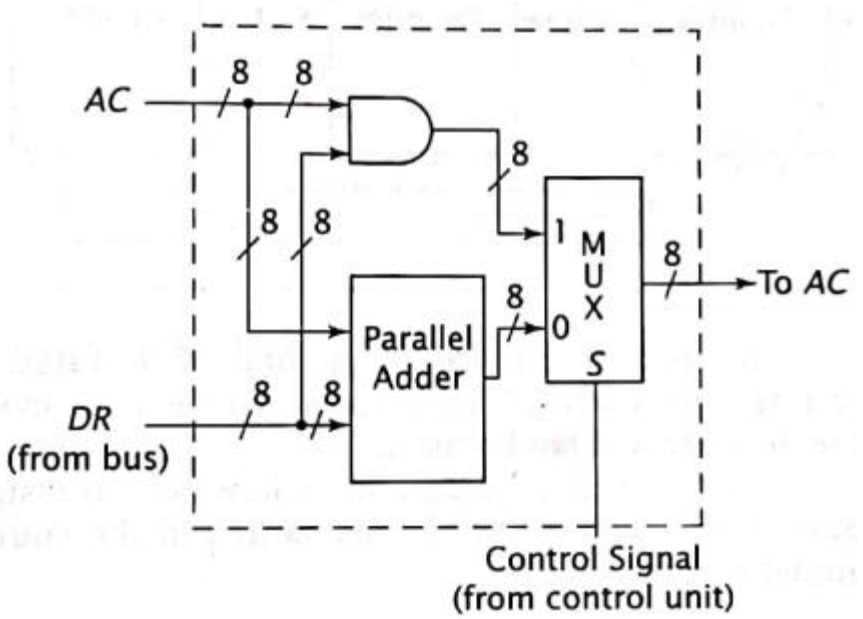
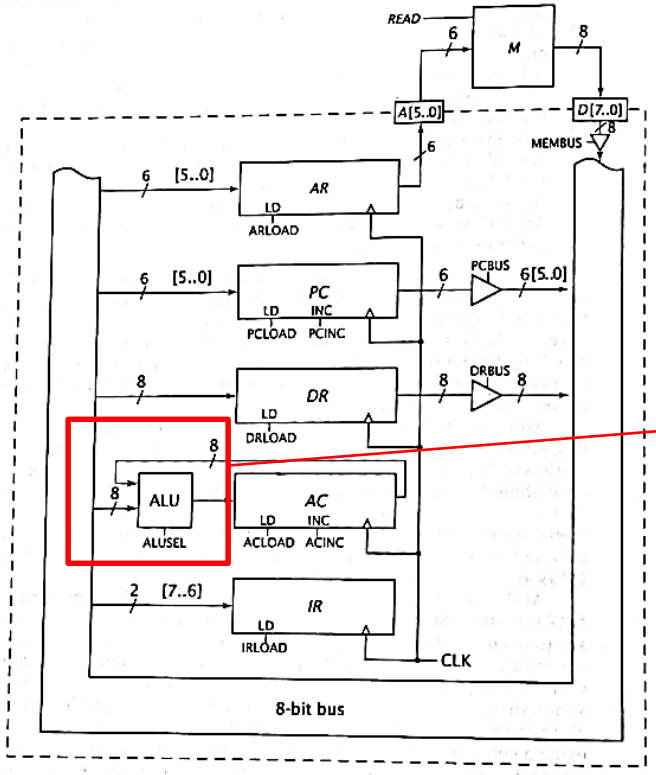
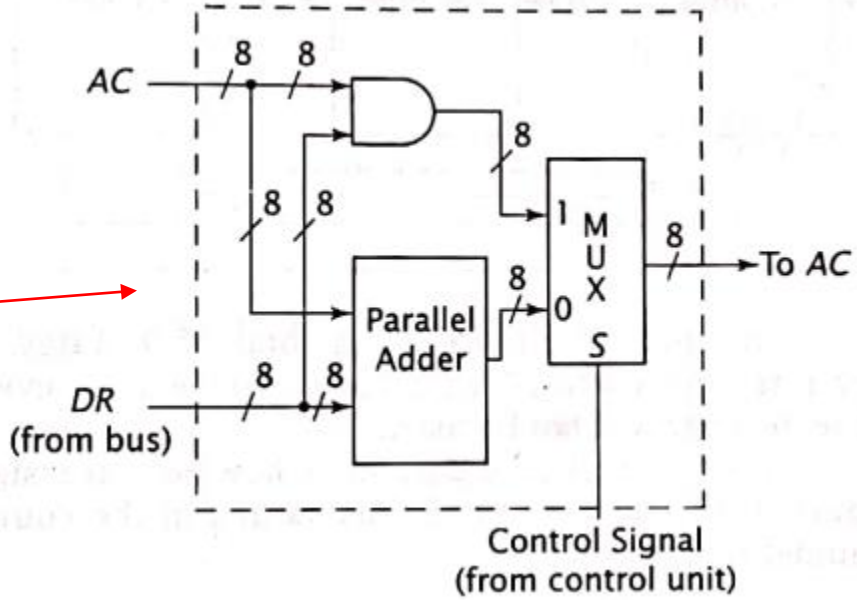


Figure: ALU Design


READ
6
M
8
A[5..0]
D[7..0]
8
MEMBUS
6
6
[5..0]
AR
LD
ARLOAD
6
[5..0]
PC
LD
INC
PCLOAD
PCINC
6
PCBUS
6[5..0]
8
DR
LD
DRLOAD
8
8
DRBUS
8
8
ALU
ALUSEL
AC
LD
INC
ACLOAD
ACINC
2
[7..6]
IR
LD
IRLOAD
CLK
8-bit bus


AC
8
8
8
8
8
8
1
M
U
X
0
S
8
To AC
Parallel
Adder
8
DR
(from bus)
8
8
Control Signal
(from control unit)

4.6 Designing Hardwired Control Unit

- After designing the CPU for every operation necessary to fetch, decode and execute the entire instruction set, the next step is **to design the circuit to generate control signals to cause operation to occur in proper sequence.** This is control unit.
- There are two methods:
  - Hardwired control
  - Microprogrammed control

- Hardwired control:
  - It uses sequential and combinational logic to generate control signals.

- Microprogrammed control:
  - It uses a lookup memory to output the control signals.

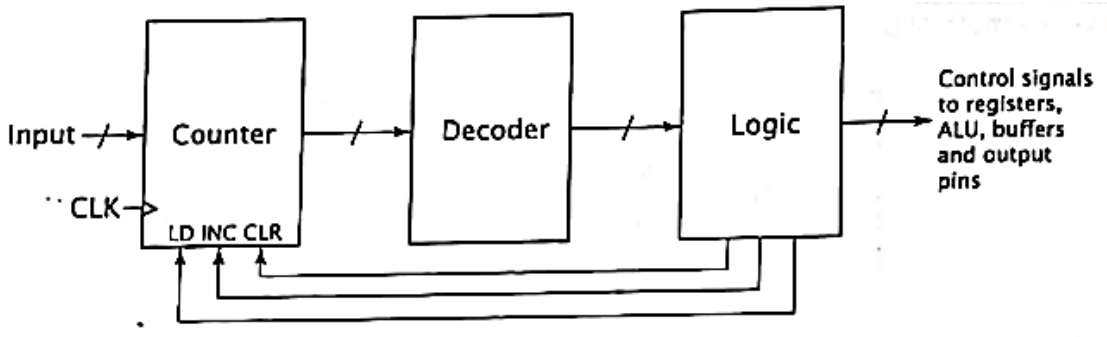
4.6 Designing Hardwired Control Unit (Contd.)

- Here, we focus on **hardwired control.**
- The simple control unit has three components:
  - **a)** **Counter:** it contain current state.
  - **b)** **Decoder:** It takes current state and generates individual signals for each state.
  - **c)** **Combinational Logic:** It takes the individual state signals and generate the control signals for each component as well as the signals to control the counter.
- These signals cause control unit to traverse the states in proper order.

Generic Hardwired Control Unit

- The CPU that we have designed has 9 total states. So, a 4-bit counter and 4-to-16 decoder are needed. Here, seven of the outputs of the decode will not be used.

- **Input:** in input, states are provided.
- **CLK:** a clock signal
- **CLR:** CLR input is used to reach the state needed.
- **INC:** use INC input of counter to traverse the states.
- L**D:** use LD input to reach the proper execute routine.



Mapping Logic

- To make the function simple as possible:
- Consider the possible mapping 10 IR [1…0] i.e. if IR=00, the input to counter is 1000. For IR=01, the input is 1001 and so on.
- So, the counter values will be:

IR	Counter value	state
00	10**00 (8)**	ADD1
01	10**01 (9)**	AND1
10	10**10 (10)**	JMP1
11	10**11 (11)**	INC1

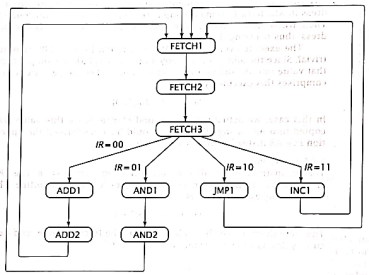
Counter values for the proposed mapping function

- Here, the problem arises. Since state ADD1 has a counter value of 8 and state AND1 has a counter value of 9. There is no counter value that has to be assigned to ADD2. (not best solution)

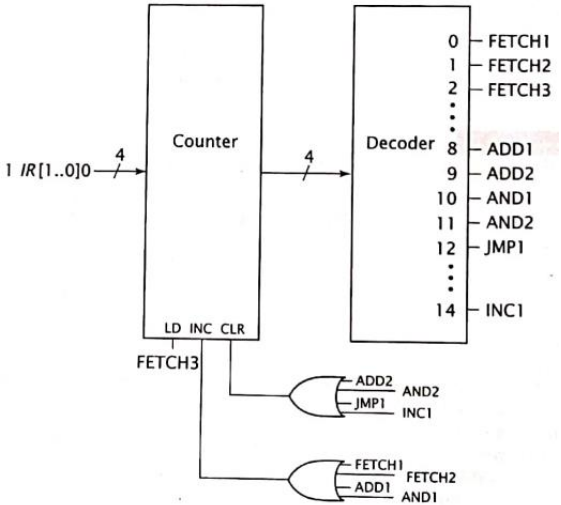
- Now if we mapping function as: 1IR [1… 0]0, which results in counter values of:

IR	Counter	Value	state
00	1 **0 0** 0 (8)	8	ADD1
01	1 **0 1** 0 (10)	10	AND1
10	1 **1 0** 0 (12)	12	JMP1
11	1 **1 1** 0 (14)	14	INC1

- Now, we can assign counter value 9 to ADD2 and 11 to AND2



- So, a complete hardwired control unit is:



Hardwired control unit for the Very Simple CPU

- For Counter, INC, CLR and LD signals are needed.
- INC is asserted when control unit is traversing sequential states, during Fetch1, Fetch2, JMP1 and INC1
- LD is asserted at end of fetch cycle during stateFetch3.
- CLR is asserted at the end of each execute cycle to return to fetch cycle. This happens during ADD2, AND2, JMP1 and INC1

Some Few Examples:

Example 1:

Instruction	Instruction Code	Operation
JMP1	00AAAAAA	PC←AAAAAA + 1
INC2	01XXXXXX	AC←AC + 2
ADD1	10AAAAAA	AC←AC + M[AAAAAA] + 1
SKIP	11XXXXXX	PC←PC + 1

Datapath

- Fetch 1: AR ← PC
- Fetch 2: DR ←M, PC ← PC + 1
- Fetch 3: IR ← DR[7,6], AR ← DR[5….0]
- JMP11: PC ←AR
- JMP12: PC ← PC +1
- INC21: AC ←AC +1
- INC22: AC ←AC +1
- ADD11: DR ← M, AC ←AC + 1
- ADD12: AC ← AC + DR
- SKIP1: PC ← PC + 1

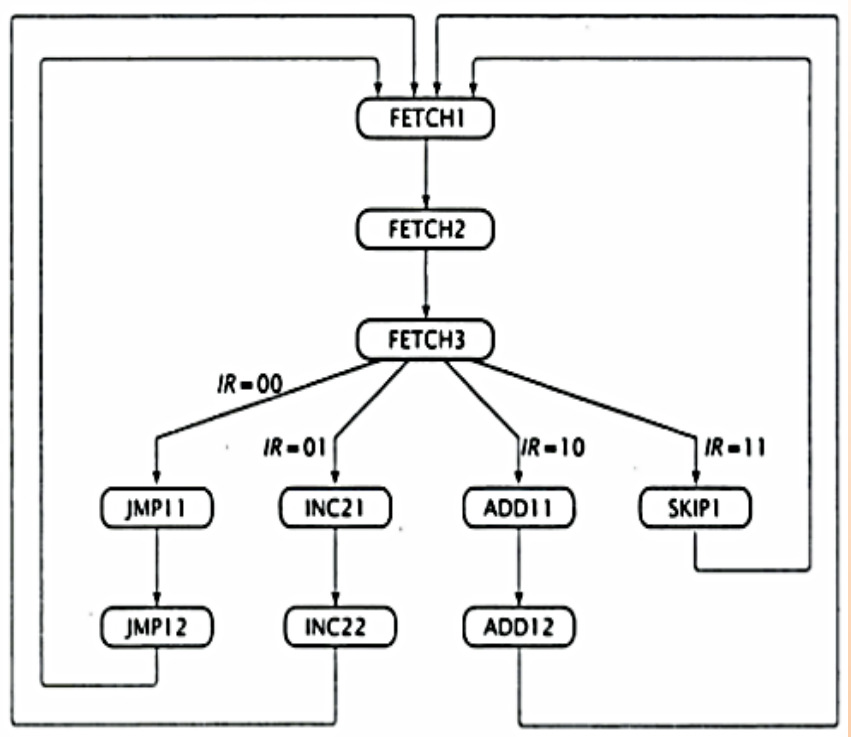


Figure: State diagram

- Students are required to design:
  - Datapath
  - ALU

Hardwired Control Unit

Datapath

- **Fetch 1: AR ← PC**
- **Fetch 2: DR ←M, PC ← PC + 1**
- Fetch 3: IR ← DR[7,6], AR ← DR[5….0]
- **JMP11: PC ←AR**
- JMP12: PC ← PC +1
- **INC21: AC ←AC +1**
- INC22: AC ←AC +1
- **ADD11: DR ← M, AC ←AC + 1**
- ADD12: AC ← AC + DR
- SKIP1: PC ← PC + 1

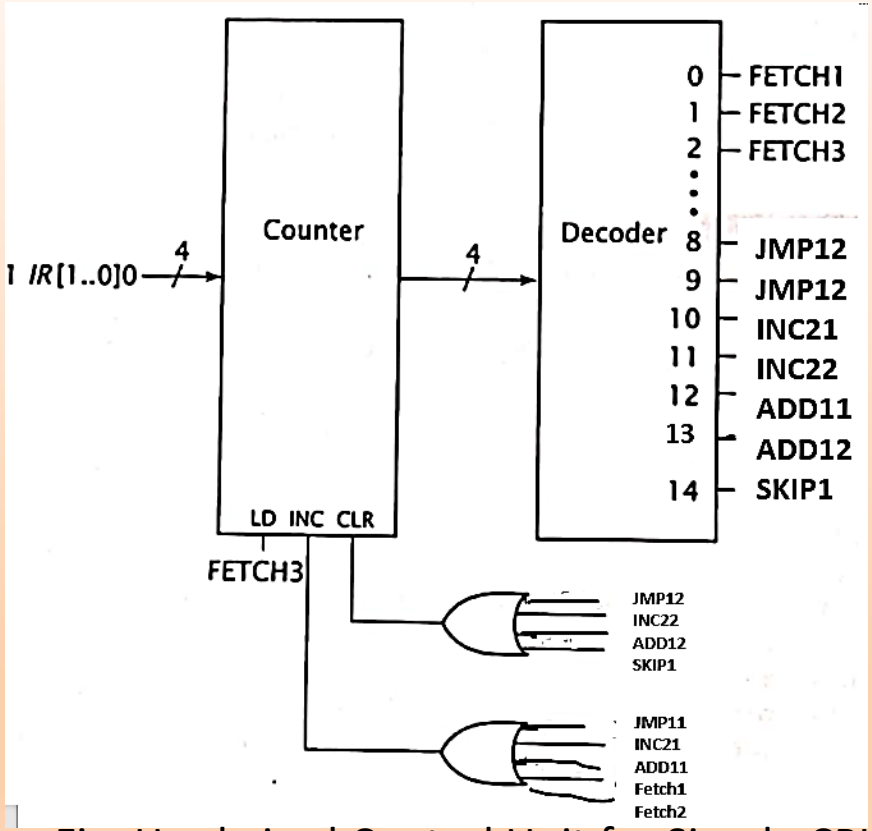


Fig: Hardwired Control Unit for Simple CPU

Design a CPU that meets the following specifications.

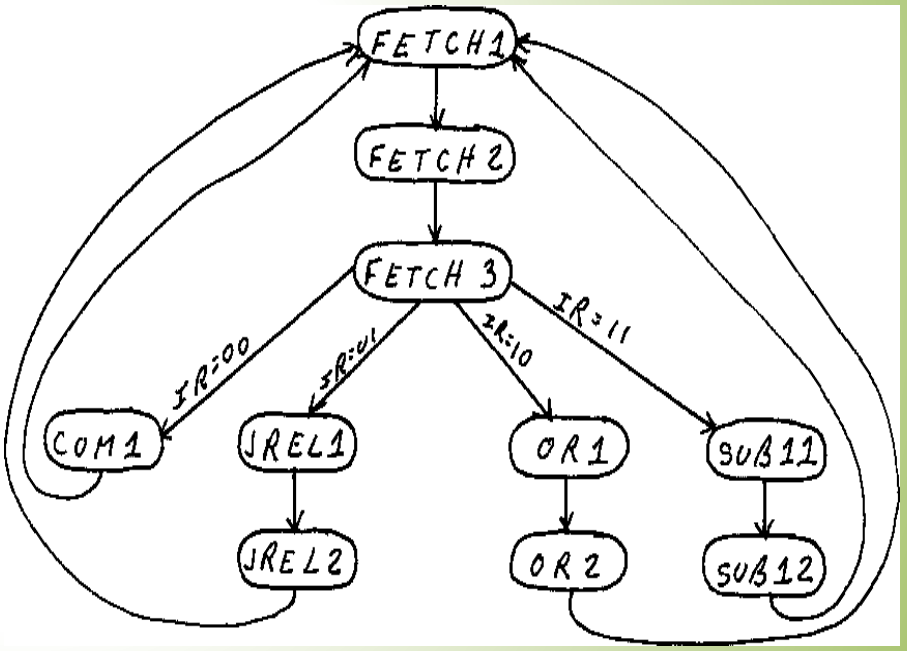
- It can access 64 words of memory, each word being 8 bits wide. The CPU does this by outputting a 6-bit address on its output pins A[5..0] and reading in the 8-bit value from memory on its inputs D[7..0].
- The CPU contains a 6-bit address register (*AR*) and program counter (*PC*); an 8-bit accumulator (*AC*) and data register (*DR*); and a 2-bit instruction register (*IR*).
- The CPU must realize the following instruction set.

Instruction	Instruction Code	Operation
COM	00XXXXXX	$AC \leftarrow AC'$
JREL	01AAAAAA	$PC \leftarrow PC + 00AAAAAA$
OR	10AAAAAA	$AC \leftarrow AC \vee M[00AAAAAA]$
SUB1	11AAAAAA	$AC \leftarrow AC - M[00AAAAAA] - 1$

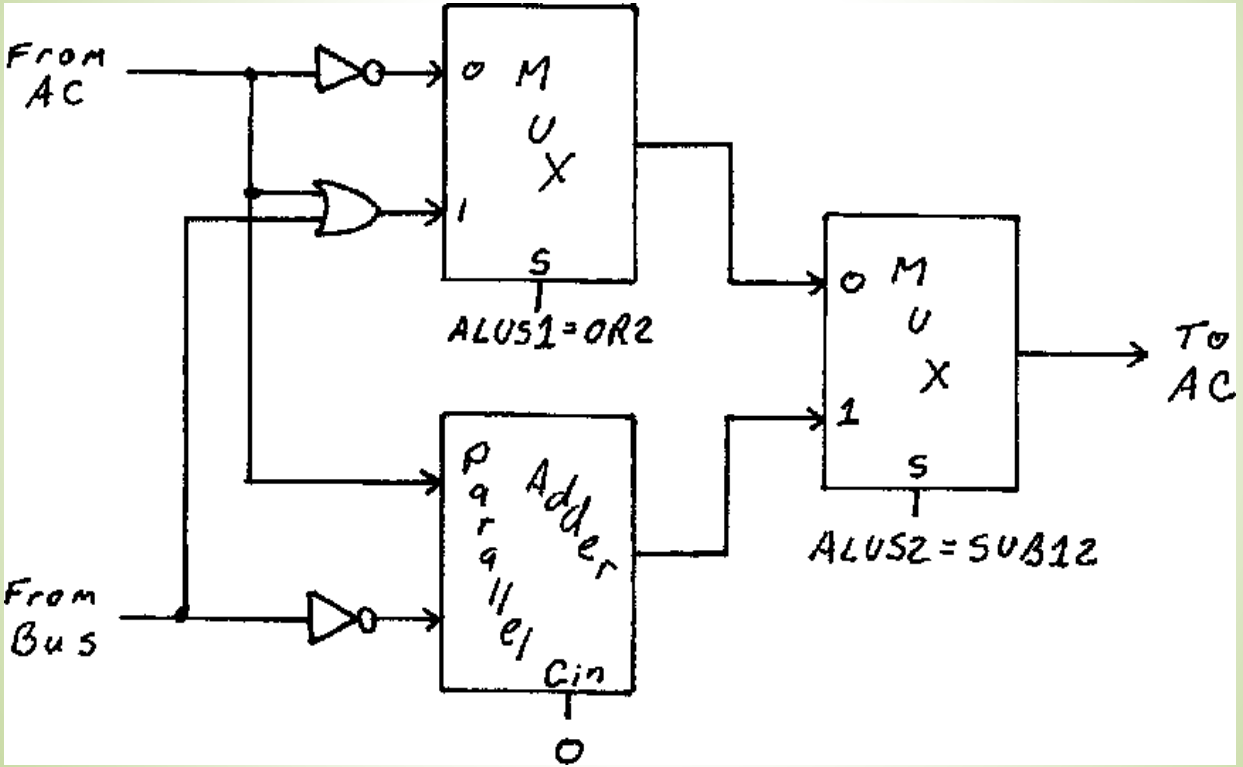
Solution:

State diagram and RTL code:

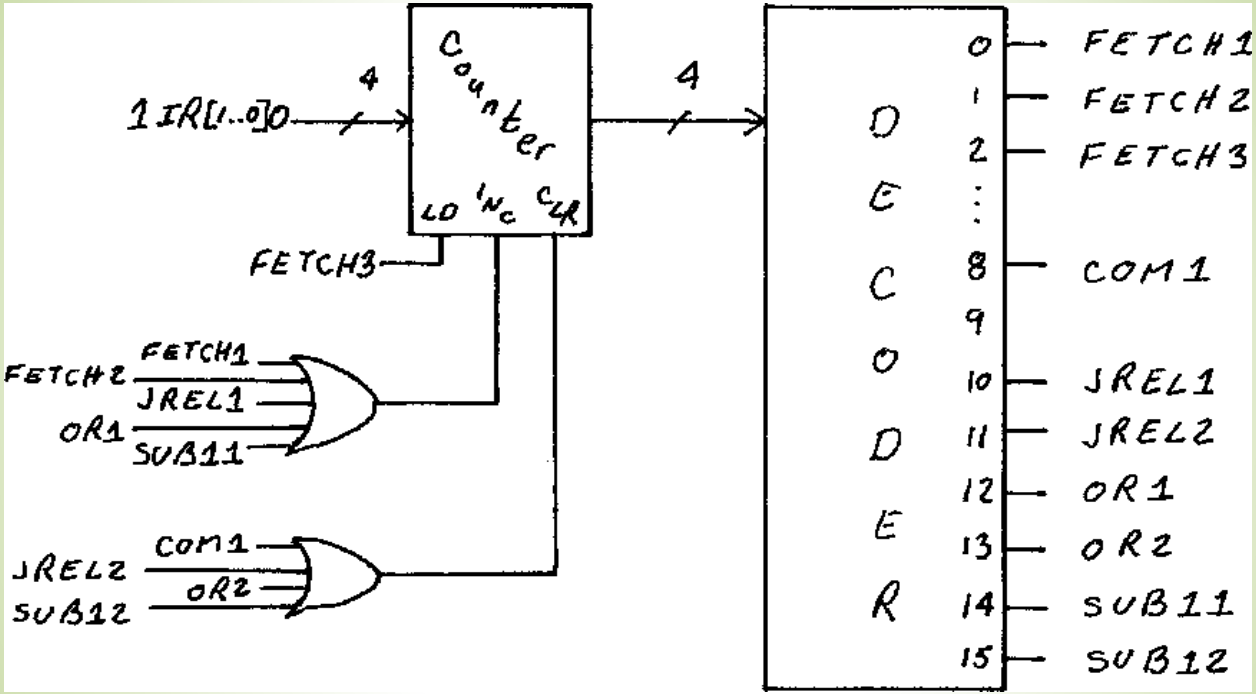
- FETCH1: $AR \leftarrow PC$
- FETCH2: $DR \leftarrow M$, $PC \leftarrow PC + 1$
- FETCH3: $IR \leftarrow DR[7..6]$, $AR \leftarrow DR[5..0]$
- COM1: $AC \leftarrow AC'$
- JREL1: $DR \leftarrow M$
- JREL2: $PC \leftarrow PC + DR[5..0]$
- OR1: $DR \leftarrow M$
- OR2: $AC \leftarrow AC$ Ú DR
- SUB11: $DR \leftarrow M$
- SUB12: $AC \leftarrow AC + DR'$



ALU:

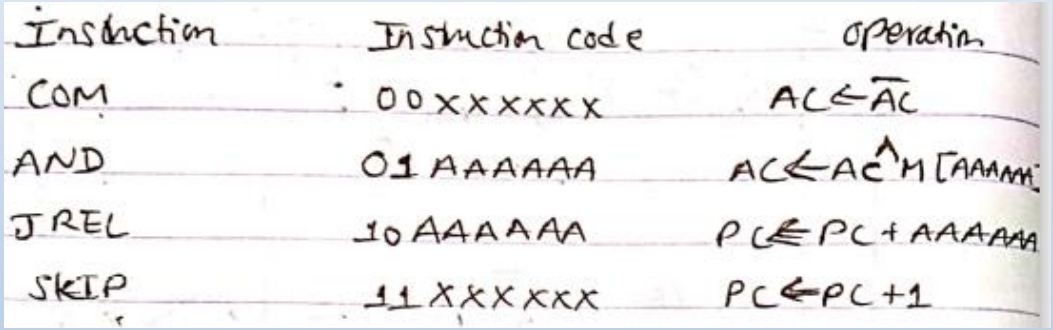

From AC
0
M
U
X
1
S
ALUS1=OR2
From Bus
Parallel Adder
Cin
0
0
M
U
X
1
S
ALUS2=SUB12
To AC

Control unit:



Assignment:

- **Design a CPU that has 6-bit address register (AR), 6-bit program counter (PC), 8-bit data register (DR) and 2-bit instruction register (IR). The CPU must execute the following instructions:**



Instruction	Instruction code	Operation
COM	00xxxxxx	$AC \leftarrow \overline{AC}$
AND	01 AAAAAA	$AC \leftarrow AC \wedge M[AAAAAA]$
JREL	10 AAAAAA	$PC \leftarrow PC + AAAAAA$
SKIP	11 xxxxxx	$PC \leftarrow PC + 1$

- Show the RTL code for fetch and execute cycles for each instruction and draw state diagram.
- Show the final register section
- Show the ALU Design
- Show hardwired control unit for same CPU.

Solution:

- Show the RTL code for fetch and execute cycles for each instruction and
- Draw state diagram.
- Show the final register section and
- Design the ALU
- Show hardwired control unit for same CPU.

- **RTL code:**

Fetch1 : AR← PC
Fetch2 : DR← M, PC← PC+1
Fetch3 : IR← DR [7-6], AR←DR[5…0]
COM1 : AC← AC’
AND1 : DR←M
AND2 : AC← AC^DR
JREL1 : AR←DR [5..0]
JREL2 : PC←PC+AR
SKIP1 : PC←PC+1

Exam Questions:

1. **a) There is a Very Simple CPU for the given set of Instructions: [PU 2017 Fall]**
   - LDA (Load Accumulator with a value from memory) with opcode 00;
   - AND (performing and operation to the content of accumulator with the value at some memory address) with opcode 01;
   - SKIP (skipping an instruction) with opcode 10;
   - DEC (decrementing the value at accumulator) with opcode 11;

   Let the instruction width be 8 bits and address is 6 bits. Design the CPU's Register section, state diagram and ALU.

   b) Design the hardwired control unit for the CPU described in above question. [PU 2017 Fall]

2. a) There is a very simple CPU for the given set of Instruction: [PU 2017 Spring]

Instruction	Instruction Code	Micro-operation
ADD	00 AAAAAA	AC←AC+M[AAAAAA]
SUB	01 AAAAAA	AC←AC-M[AAAAAA]
SKIP	10 XXXXXX	PC←PC+1
DEC2	11 XXXXXX	AC←AC-2

Let the instruction width be 8 bits and address is 6 bits. Design the CPU's register section, state diagram and ALU.

b) Design the hardwired control unit for the CPU described in above question. [PU 2017 Spring]

3. a) There is a very simple CPU for the given set of Instruction: [PU 2018 Fall]

Instruction	Instruction Code	Operations
STA	00 AAAAAA	M[AAAAAA}←AC
XNOR	01 AAAAAA	AC←AC⊙M[AAAAAA]
JMP	10 AAAAAA	GOTO AAAAAA
SKIP	11 XXXXXX	PC←PC+1

Let the instruction width be 8 bits and address is 6 bits. Design the CPU's register section, state diagram and ALU.

b) Design the hardwired control unit for the CPU described in above question. [PU 2018 Fall]

4. a) There is a very simple CPU for the given set of Instruction: [PU 2018 Spring]

Instruction	Instruction Code	Operations
STR	00 AAAAAA	AC→M[AAAAAA]
NAND	01 AAAAAA	AC←(AC^M[AAAAAA])`
JMP	10 AAAAAA	PC← AAAAAA
INC2	11 XXXXXX	AC←AC+2

Let the instruction width be 8 bits and address is 6 bits. Design the CPU's register section, state diagram and ALU.

b) Design the hardwired control unit for the CPU described in above question. [PU 2018 Spring]

5. a) There is a very simple CPU for the given set of Instruction: [PU 2019 Spring]

Instruction	Instruction Code	Operations
STR	00 AAAAAA	AC→M[AAAAAA]
AND	01 AAAAAA	AC←AC^M[AAAAAA]
JMPR	10 AAAAAA	PC←PC+AAAAAA
DEC2	11 XXXXXX	AC←AC-2

Let the instruction width be 8 bits and address is 6 bits. Design the CPU's register section, state diagram and ALU.

b) Design the hardwired control unit for the CPU described in above question. [PU 2019 Spring]

6. For a very simple CPU, has the following instruction set. Show the state diagram, register section, ALU design. [PU 2020 Spring]

Operation	Instruction Code	Instruction
M[AAAAAA] <- AC	00AAAAAA	STA
AC<- M[AAAAAA]	01AAAAAA	LDA
AC<- AC+2	10XXXXXX	INC
AC<- AC'	11 XXXXXX	COM

End of chapter 4