The CPU

Review Notes

Yul Williams, D.Sc.

Goals

- At the conclusion of this session, the student should be able to:
- Identify the major portions of a CPU
- Understand the CPU design process
- Understand the process of designing CPU components
- Implement control signals for the CPU

Outline

- Introduction to CPU Design
- Specifying a CPU
- The Very Simple CPU

Introduction to CPU Design

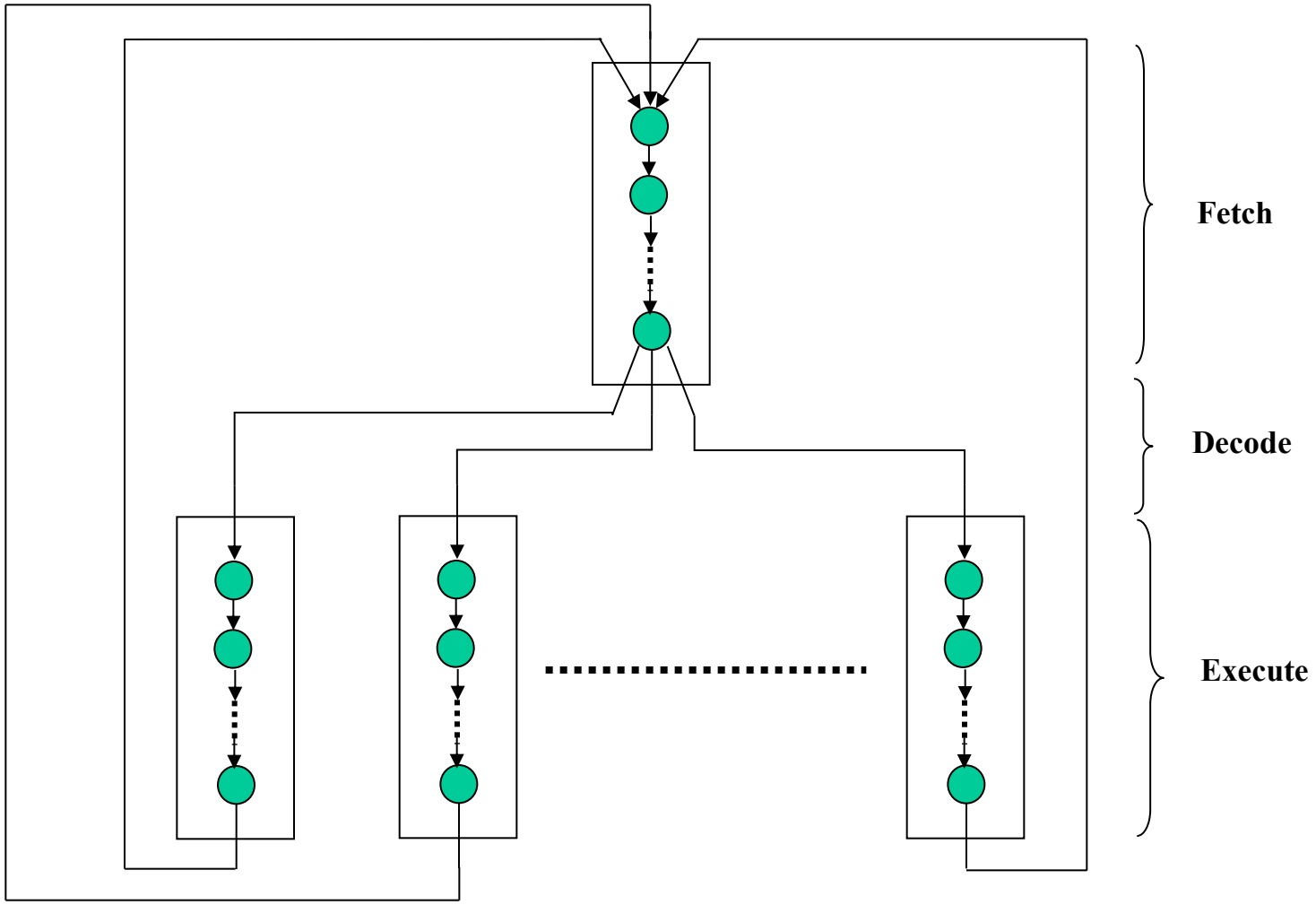
The CPU

- A CPU is composed of 3 major components:
  - The Register section
  - The Arithmetic and Logic Unit (ALU)
  - The Control Unit
- These components work together to perform the fetch, decode and execute cycles for all of the CPU's instructions.

Specifying a CPU

The following material is excerpted from the book: Computer Systems Organization and Architecture by John Carpinelli.

CPU State Diagram



Operation Sequence

- Fetch Cycle – fetch an instruction from memory and go to the decode cycle
- Decode Cycle – decode the instruction and go to the execute cycle
- Execute Cycle – execute the instruction then go to the fetch cycle

The Very Simple CPU

A Detailed View of CPU Design

Very Simple CPU Instruction Set

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

The Accumulator (AC)

- The accumulator is a programmer accessible register
- It stores the results of each operation
- It is 8 bits wide

The Program Counter (PC)

- The program counter contains the address of the next instruction to be executed
- It is a 6 bit register

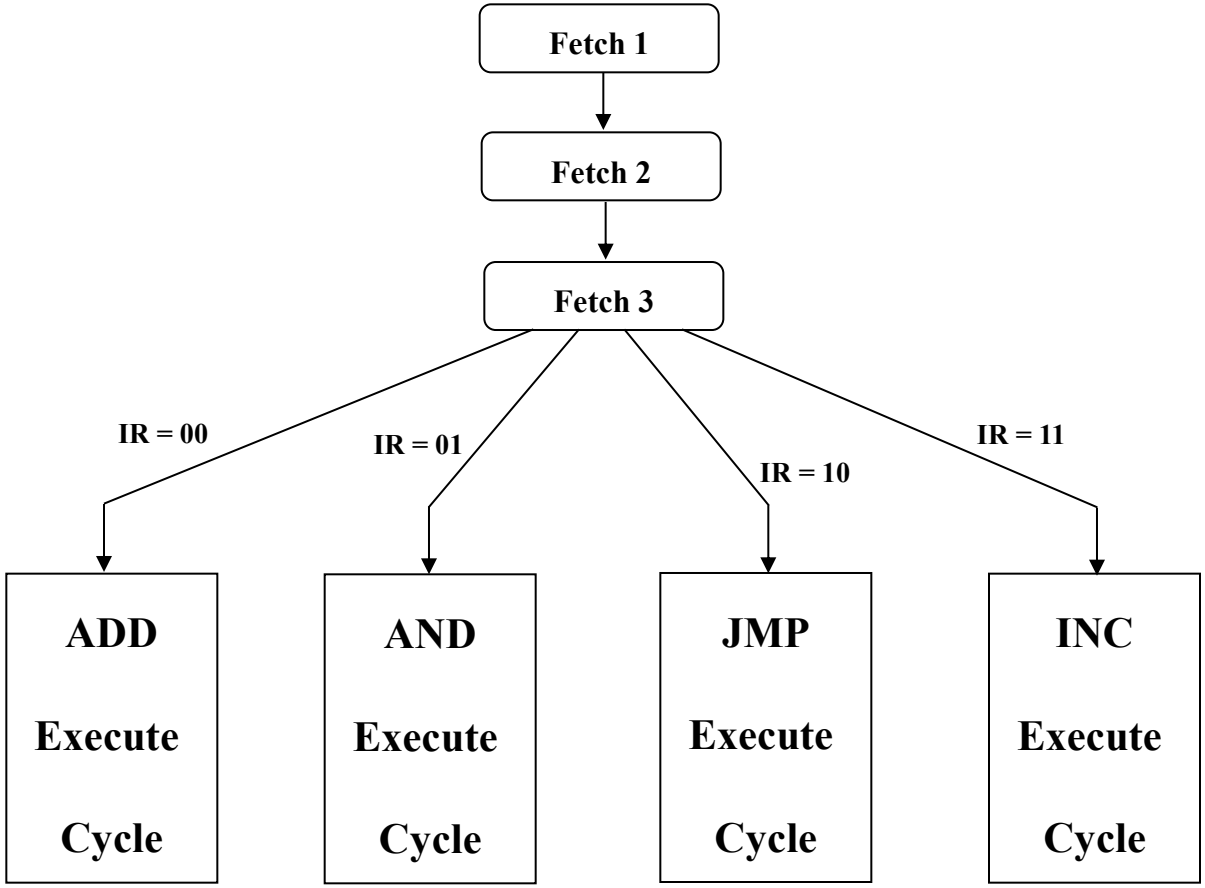
The Data Register (DR)

- The Data Register receives instructions and data from memory
- It is an 8 bit register
- It is tied to the data lines D[7..0]

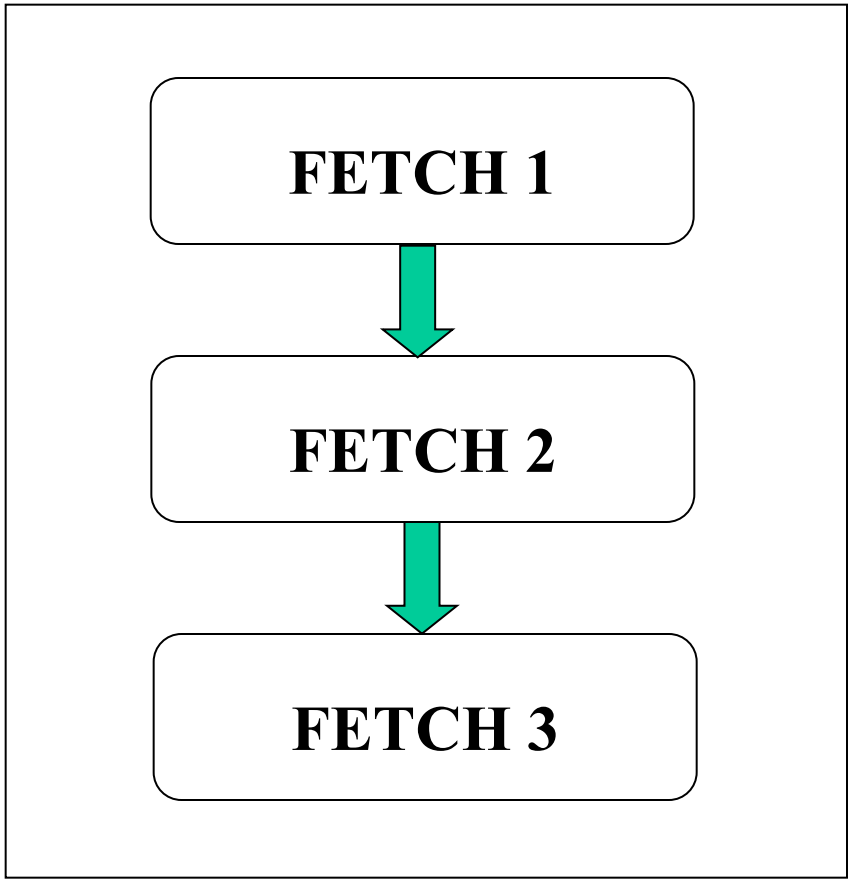
The Instruction Register (IR)

- The Instruction Register stores the opcode portion of the instruction code fetched from memory
- It is two bits wide. As such, it can represent only four instructions

Very Simple CPU Fetch, Decode and Execute Cycles



The Fetch Cycle



Fetching Instructions from Memory

- In order for the CPU to execute an instruction, it must first fetch it from memory
- Fetching instructions from memory follows a specific sequence of steps

Fetching Instructions

- An address is sent to memory by loading it into the AR
- The memory places the requested data onto its output pins (that are connected to the data bus D[7..0]
- The data is copied from the data bus and stored in the DR
- The PC register is incremented to point to the next instruction to be executed
- The 2 most significant bits of DR are copied to the IR and the 6 remaining bits are copied to AR

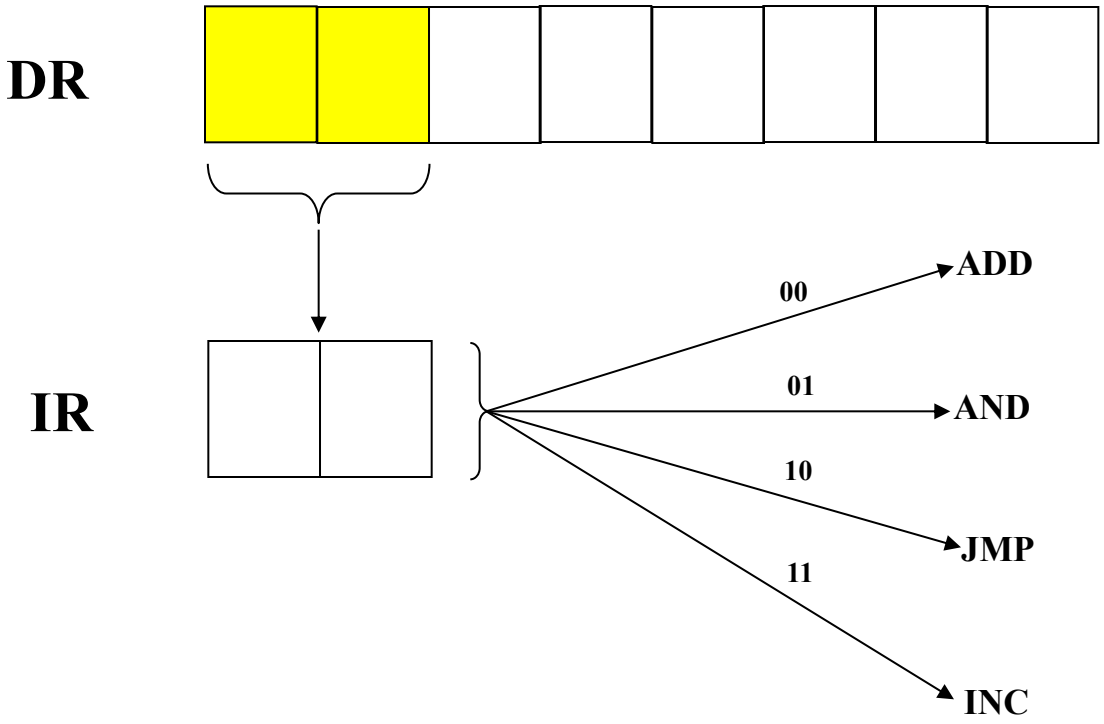
Fetch Cycle Illustration

FETCH 1: AR ← PC

FETCH 2: DR ← M, PC ← PC + 1

FETCH 3: IR ← DR[7..0], AR ← DR[5..0]

Instruction Decoding



The Address Register (AR)

- In our Very Simple CPU model, the address register supplies an address to memory
- The address register is 6 bits
- We denote its range as A[5..0]
- The CPU may address 64 bytes of memory

Executing Instructions

- In order to complete the state diagram for the CPU, each execute routine must be defined
- Each of them is fairly straightforward
- We will look at them one at a time

ADD Execute Routine

- In order to execute the ADD instruction, the operand must be fetched from memory
- The operand may then be added to the contents already stored in the accumulator

ADD Execute Routine

ADD1: DR ←M

ADD2: AC ← AC + DR

AND Execute Routine

- Execution of the AND instruction is very similar to that of the ADD instruction
- The operand must be fetched from memory and placed in the DR
- The contents of the DR and the AC are logically ANDed together and the results are stored in the AC

AND Execute Routine

AND1: DR ← M

AND2: AC ← AC ^ DR

JMP Execute Routine

- The JMP instruction is executed by copying the target address into the PC
- In the previous Fetch cycle, the address is already stored in the DR
- When the CPU fetches the next instruction, it copies the contents of the DR[5..0] into the PC and jumps to the target address

JMP Execute Routine

JMP1: PC ← DR[5..0]

INC Execute Routine

- Like the JMP instruction, the INC can be executed in a single step
- The CPU adds 1 to the accumulator and goes back to the fetch state

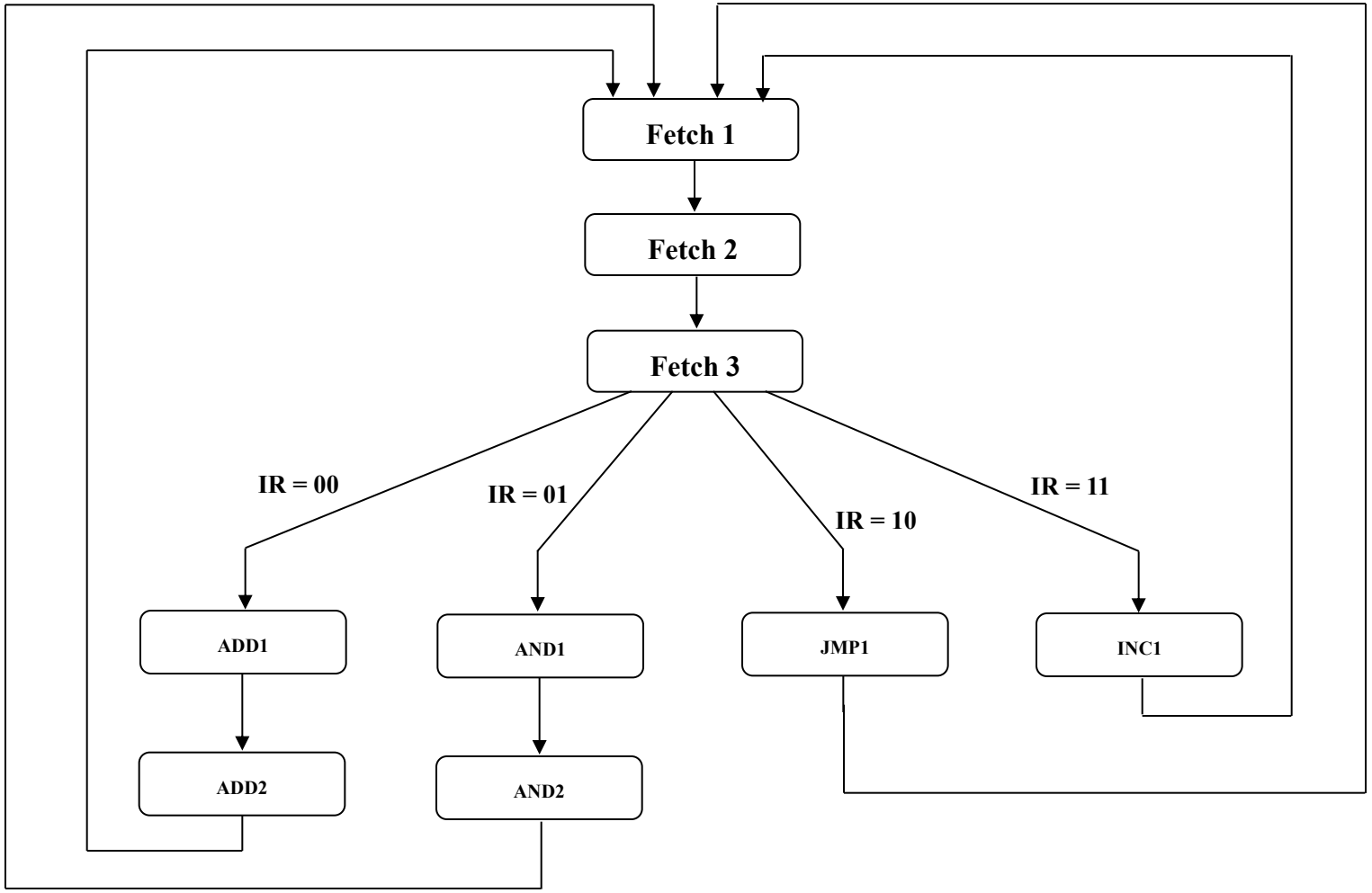
INC Execute Routine

INC1: AC ← AC + 1

Constructing the Final State Diagram for the Very Simple CPU

- Once all of the instruction execution routines are defined, the final state diagram for the Very Simple CPU may be formed
- We simply replace the high-level execution boxes from the original diagram with the more detailed execution steps required by each instruction

State Diagram for the Very Simple CPU



Establishing the Data Paths

- The state diagram and register transfers tell us the required behavior and actions of the Very Simple CPU
- The interconnection of components must be established to make it complete
- The data path must be designed so that the CPU can realized to perform the desired actions

Defined CPU Operations

Instruction	Operations
FETCH1	**AR <-- PC**
FETCH2	**DR <--M, PC <-- PC + 1**
FETCH3	**IR <-- DR[7..6], AR <-- DR[5..0]**
ADD1	**DR <-- M**
ADD2	**AC <-- AC + DR**
AND1	**DR <-- M**
AND2	**AC < -- AC ^ DR**
JMP1	**PC <-- DR[5..0]**
INC1	**AC <-- AC + 1**

Data Path Decisions

- Since interaction must take place between several registers and memory, a direct path approach would be cumbersome
- A better approach would be to use a bus approach

Review Data Transfer Operations

- Since we are going to use the bus approach to route the data, we must analyze operations that load data into each component
- Group the operations by the register whose content they modify

Analysis of Operations

Modified Register	Operations
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

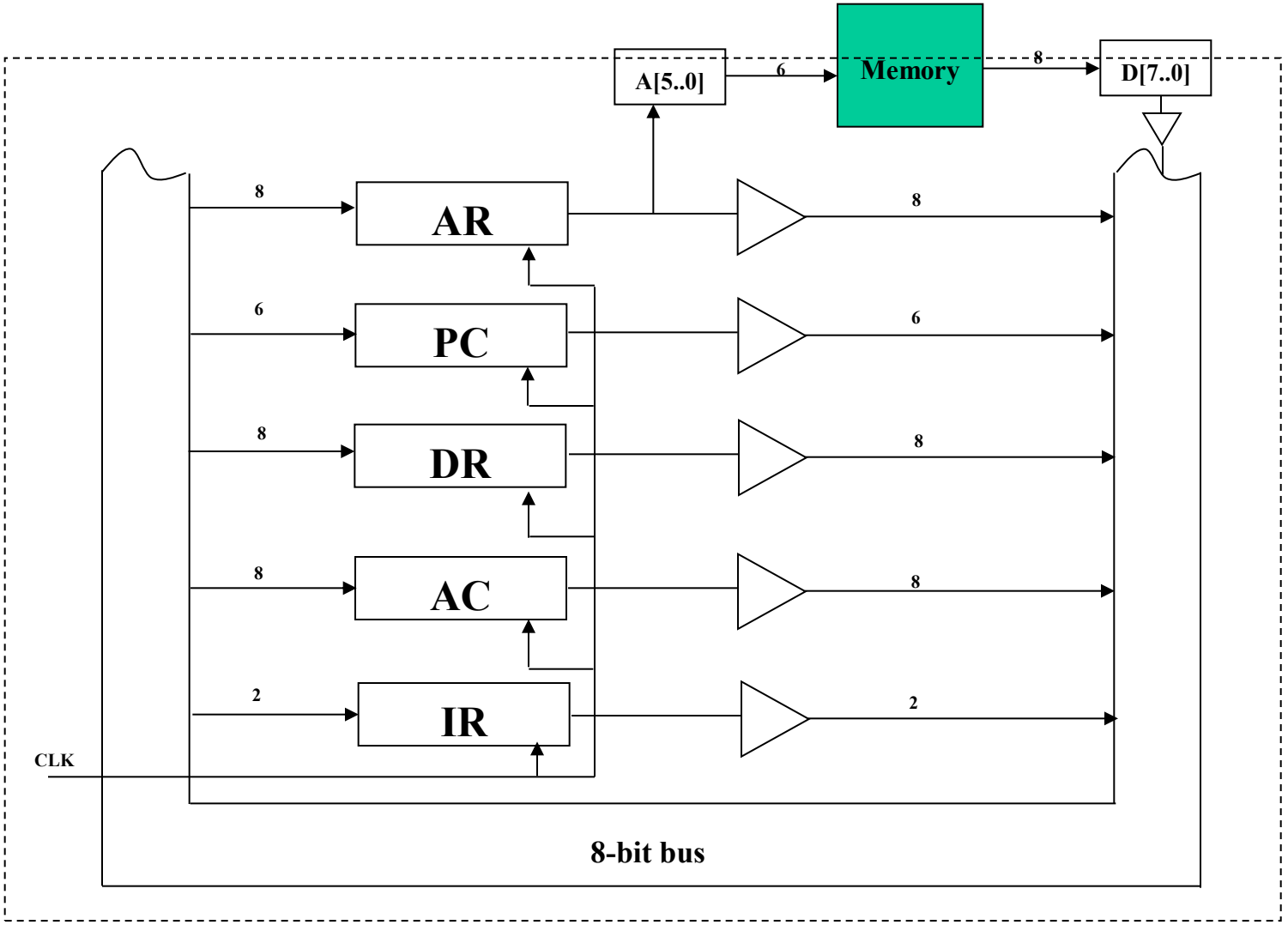
Results of Analyses

- AR, DR, and IR always load data from some other component
  - They all need to be able to perform a parallel load
- PC and AC can load data from external sources but they also need to be able to increment their contained values
  - Instead of designing the registers as counters eliminates the need to design special hardware to perform the increments

Connecting the Components

- After we have made the determination of how the components need to operate, we may proceed to actually connecting the components
- As planned, we will use the bus structure to perform the connections
- The next illustration is a preliminary design of the register section of the CPU

Preliminary Design



Actual Data Transfers

- AR only supplies its data to memory. There is no need to connect it to the internal bus
- IR does not supply its data to any other component on the internal bus
- AC, likewise, does not supply its data to any of the components on the internal bus
- These connections can be removed from the preliminary design drawing

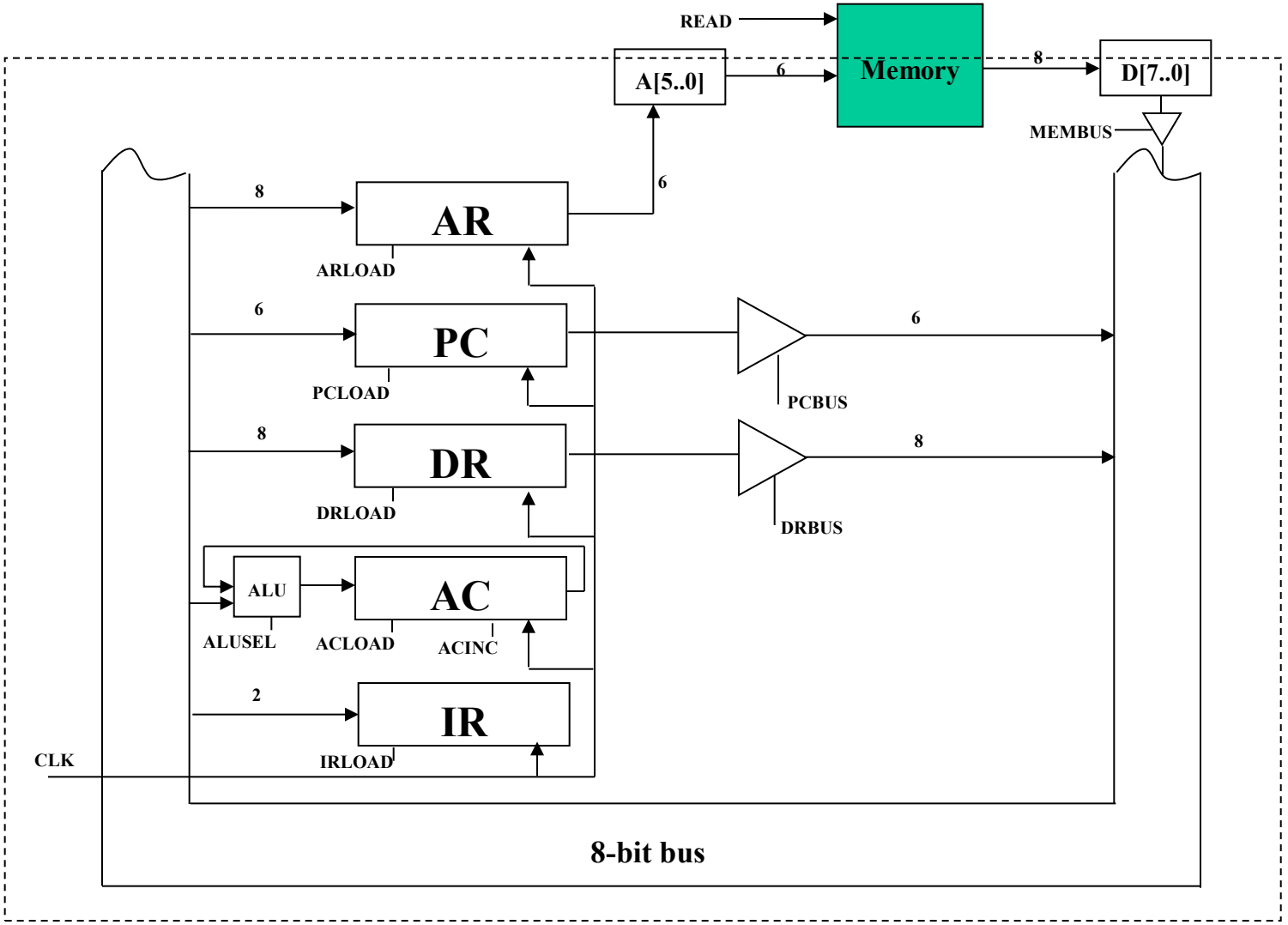
Actual Data Transfers, Cont'd

- The internal bus is 8 bits wide
  - Some data transfers require 6 bits and 2 bits
  - We must specify which components receive data from which bits of the internal bus
- The AC must be able to load the sum of AC + DR and the logical AND of AC ^ DR
  - An ALU is required to facilitate these functions

A Final Model

- After the suggested changes are implemented, the CPU has the capability to perform all of the required data transfers
- The next tasks are to define the ALU and the control unit
- The final model is shown next

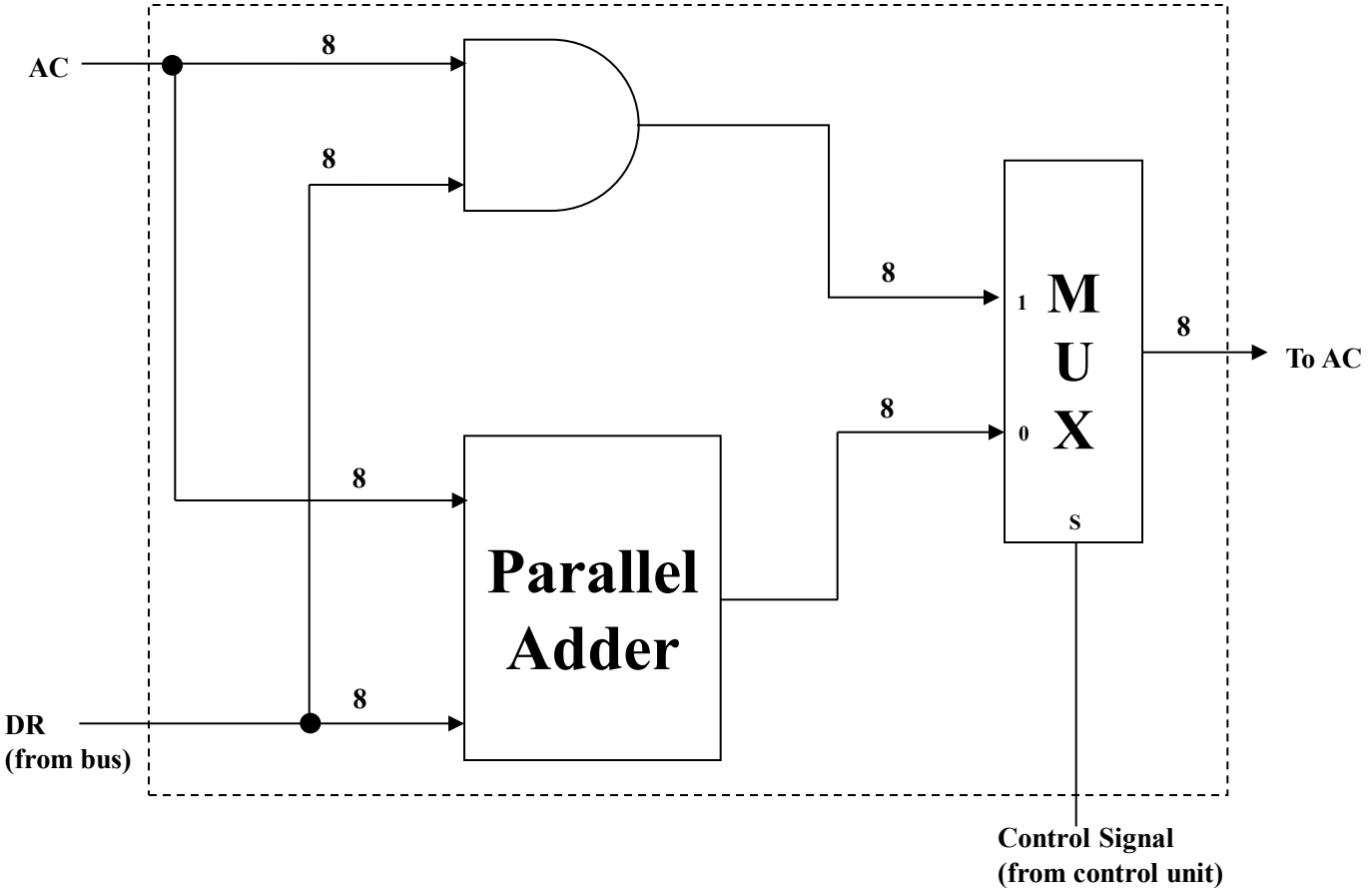
Final Register Section Design



ALU Design

- The ALU must implement the logical AND and the arithmetic ADD operations
- The AND operation is provided through a network of AND gates (represented by the 16 input AND gate in the following illustration
- The ADD operation is provided through the use of a parallel adder circuit
- Both operations are performed each time the AC and DR inputs are provided. The final output, however, is multiplexed and the select line (S) determines which result (AND or ADD) is stored back into the AC

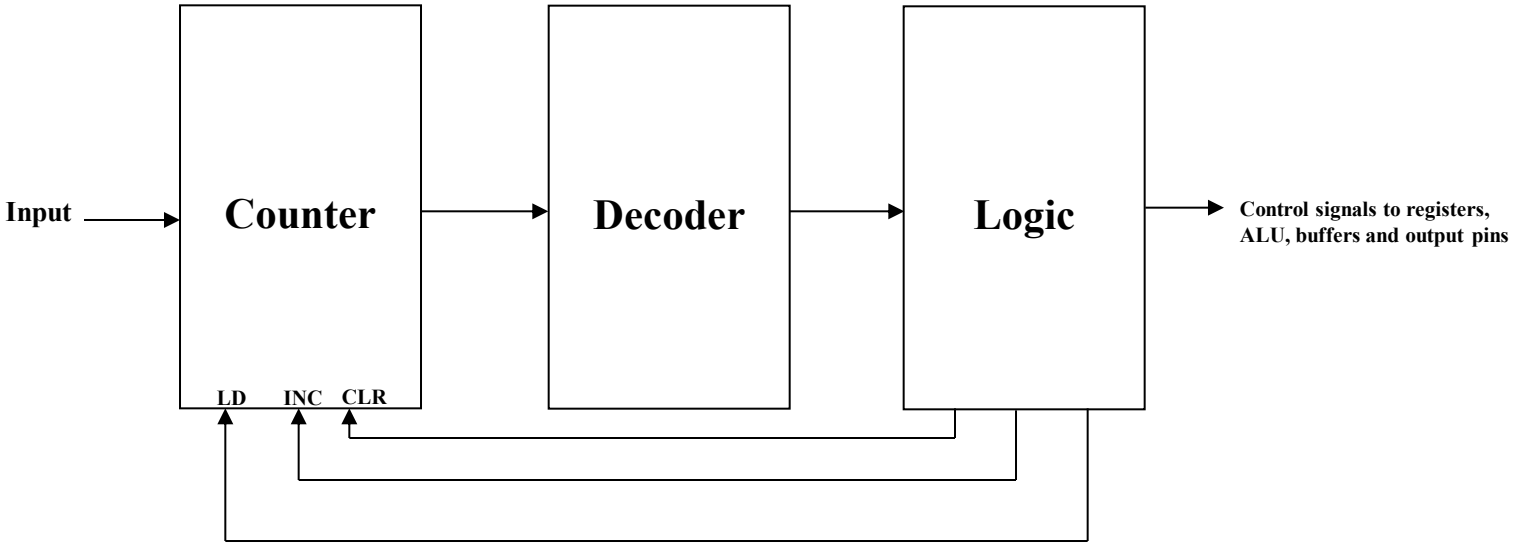
The ALU Design



Control Unit Design

- The CPU is currently able to fetch, decode, and execute all of the instructions in the instruction set
- The circuitry that provides the control signals must now be constructed. This circuitry comprises the control unit
- The control unit is constructed using three components:
  - A counter, a decoder and some combinational logic to generate the actual signals for the components and the counter
  - This will be a hardwired control unit design

The Generic Control Unit



The Counter and Decoder

- The Very Simple CPU has 9 states
- In order for the counter to cover those 9 states, it must have an adequate number of bits to cover all states
- We will use a 4-bit counter as the input to the decoder
- Since the decoder accepts the 4-bit counter output, it can represent up to 16 states. Seven of the 16 states will not be used
- The decoder is a 4 to 16 decoder

Assigning States to Decoder Outputs

- Assign FETCH1 to counter 0. Use clear to reach this state
- Assign sequential states (FETCH2 and FETCH3) to counter values 1 and 2, respectively.
- Assign ADD1 and ADD2 to consecutive counter values
- Assign AND1 and AND2 to consecutive counter values
- Assign the 1st state of each execute routine based on instruction opcodes and the maximum number of states in the routine

Assigning States to Decoder Outputs cont'd

- Use the opcodes to generate the data input to the counter and the Load (LD) to reach the proper execute routine
- The opcode is mapped to the correct execute routine using this process

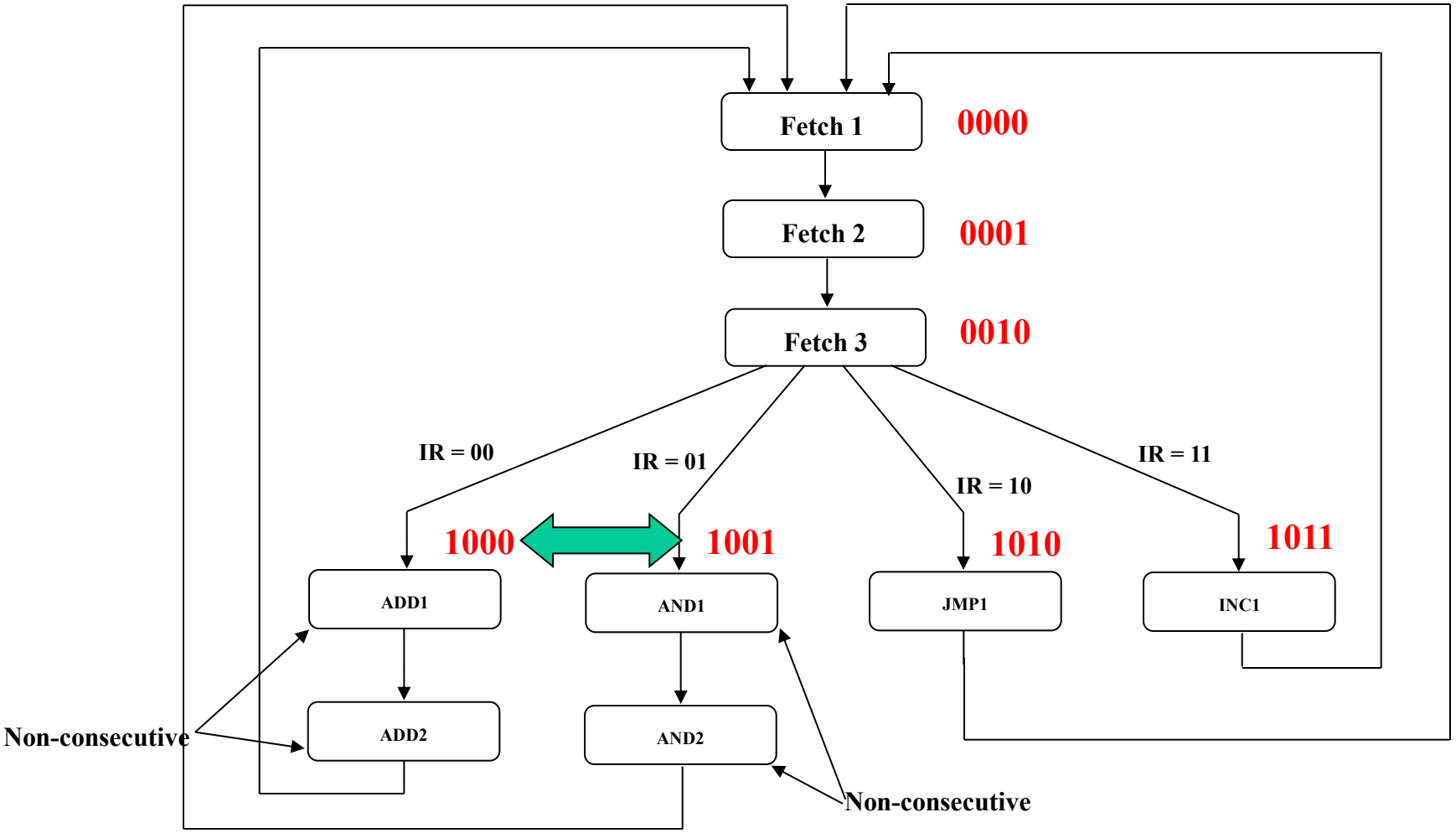
Instructions and Their First States

Instruction	First State	IR
ADD	ADD1	00
AND	AND1	01
JMP	JMP1	10
INC	INC1	11

Counter Values for Proposed Mapping Function

IR[1..0]	Counter Value	State
00	1000 (8)	ADD1
01	1001 (9)	AND1
10	1010 (10)	JMP1
11	1011 (11)	INC1

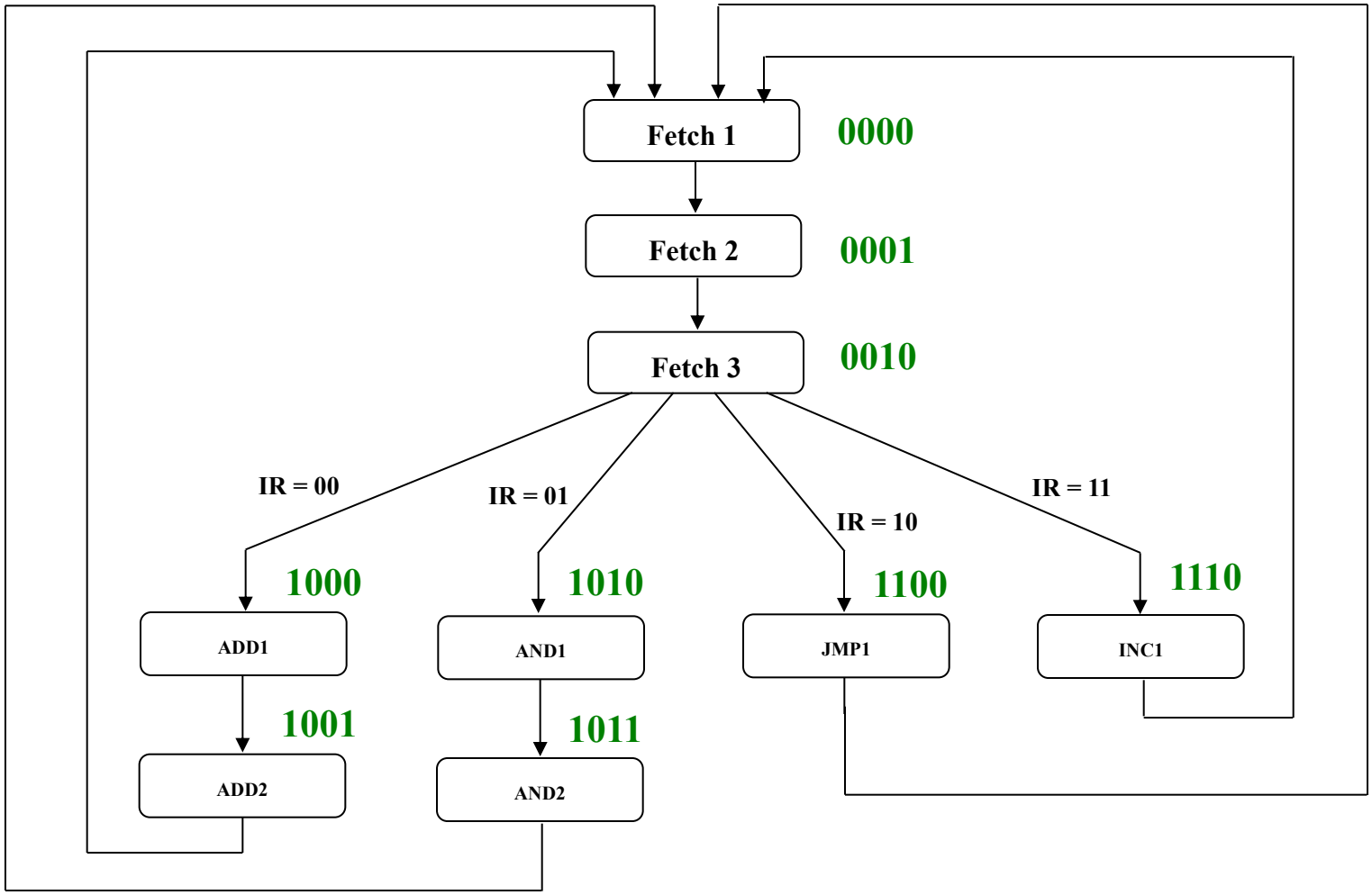
Current Mapping of States



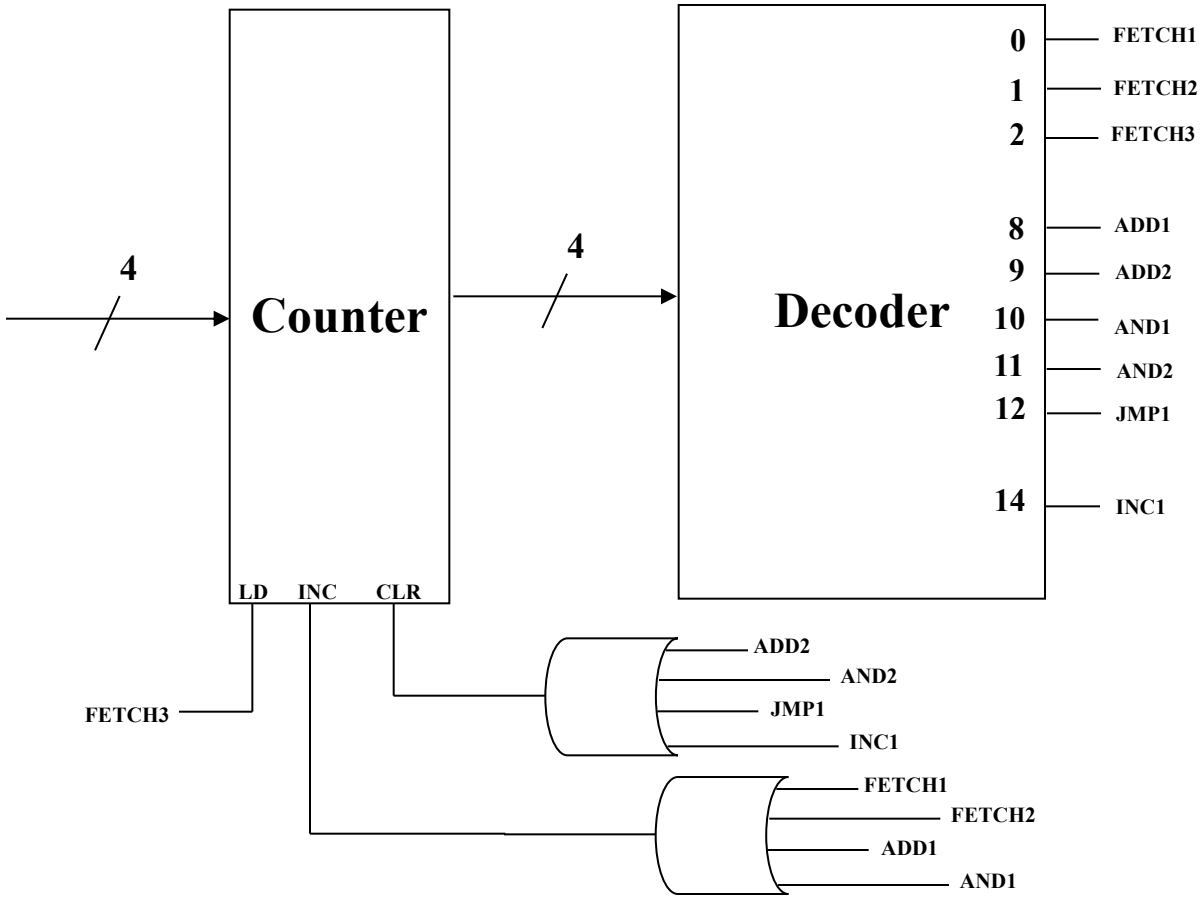
Let's Redefine our Mapping

- Each execute routine hs no more than 2 states
- Let's alter the counter values by two
- We do this by shifting the IR[1..0] to the left
- This results in the counter values 8, 10, 12, and 14 for ADD1, AND1, JMP and INC1, respectively
- Now we can assign ADD2 = 9, AND2 = 11 to yield consecutive values
- Now, let's take a look at the new assignments

Final Mapping of States



Hardwired Control Unit for The Very Simple CPU



Generation of Control Signals

PCLOAD = JMP1

PCINC = FETCH2

DRLOAD = FETCH2 v ADD1 v AND1

ACLOAD = ADD2 v AND2

ACINC = INC1

IRLOAD = FETCH3

MEMBUS = FETCH2 v ADD1 v AND1

PCBUS = FETCH1

READ = FETCH2 v ADD1 v AND1

Summary

- Designing a CPU can be tedious and time consuming
- It is very important to not skip steps in the design process
- The design process will be iterative for portions of the design
- The success of your CPU design will depend greatly on the amount of work you put into the initial CPU specification