Lecture 13-14: March 12, 2021

Computer Architecture and Organization-I

Biplab K Sikdar

Control Unit

Other than the ALU the important block of a CPU is control unit (CU).

CU interprets CPU instruction to determine control signals to be issued for instruction execution. Input outputs of a CU are shown in Figure 1.

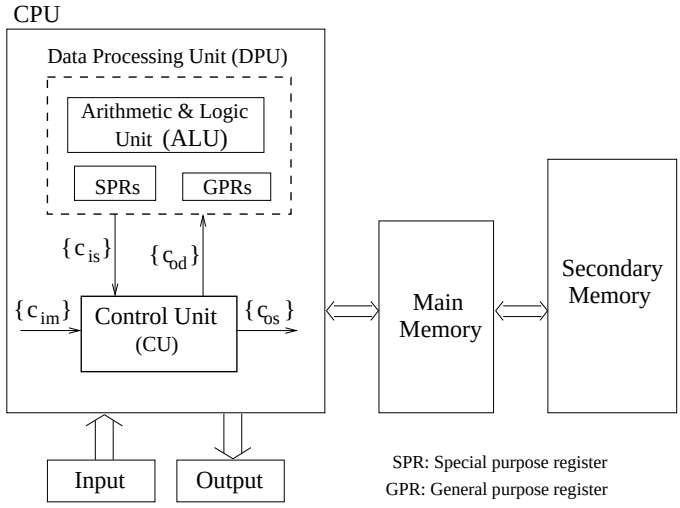
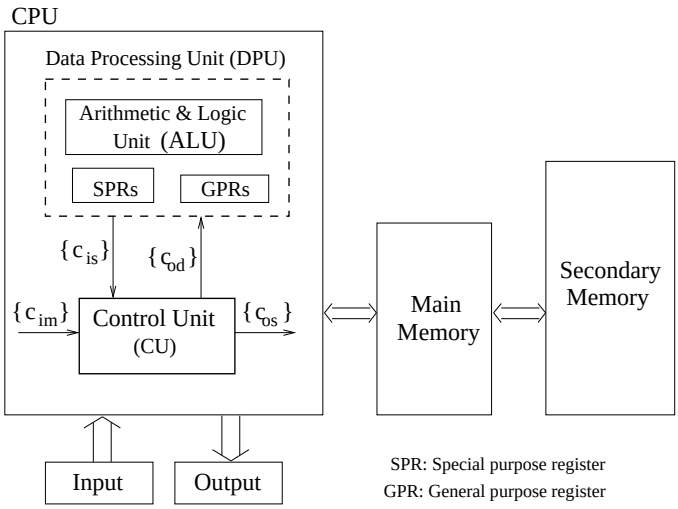


Figure 1: Input output signals of a control unit

- c_{od}s directly control operations of DPU (data processing unit). Main function of a CU is to generate c_{od}s.
- c_{is}s are status dignals of CU. These allow CU to make data dependent decision, such as errors, overflow in DPU etc.
- c_{os} is transmitted to other CUs and indicates status conditions such as 'busy' or 'operation completed'.
- c_{im}s are received from other CUs that is, signals from a supervisor.

0.1 Control Design



The main functions of a control unit for a CPU are to

i) Fetch an instruction from memory -that is, instruction sequencing,

ii) Interpret instruction in determining control signals to be sent to DPUs -that is, instruction interpretation.

0.1.1 Instruction sequencing

For instruction sequencing, the simplest method can be the storing of next executable instruction address with in the current instruction (Figure 2).

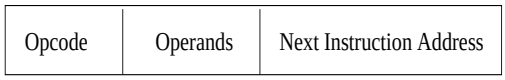


Figure 2: Instruction format containing next instruction address

This technique was followed in early designs (example EDVAC).

Inclusion of next instruction address in instruction format increases instruction length.

Alternative scheme: use of PC or IAR (insruction address register).

PC stores next executable instruction address.

While CPU executing an instruction I_i, PC is incremented by 1 (or k) to point to next executable instruction I_j.

$$PC \leftarrow PC + 1$$

or

$$PC \leftarrow PC + k$$

where k memory word is required to store the current instruction I_i in execution.

For a branch instruction, say JMP X, PC will be loaded with X -that is,

$$PC \leftarrow X \text{ for JMP X}$$

For conditional branch, it is

$$PC \leftarrow X$$

if branch condition is true. Otherwise,

$$PC \leftarrow PC + 1(k).$$

Instruction sequencing in subroutine call (CALL X (and RET)) is implemented as

$$\text{CALL X} \equiv \text{PUSH PC}, \quad \text{RET} \equiv \text{POP PC}.$$

0.1.2 The Design

Two basic techniques are used to design a control unit for the CPU.

A. **Hardwired control**: Follows design of a sequential logic circuit to generate specific fixed sequence of control signals. Once constructed, changes can only be implemented by redesigning and physically viewing the unit.

B. **Microprogrammed control**: Execute sequence of micro-instructions for a particular macro-instruction (machine assembly instruction).

A micro-instruction is a set of micro-operations.

A micro-operation performs a data transfer between registers, data transfer between a register and a bus, or simple arithmatic/logical operation.

Execution of a macro-instruction is performed by fetching micro-instructions one at a time from specially designed control memory (CM).

Decoding of micro-instruction enables activation of control lines (signals).

Control signals are implemented in software rather than the hardware and, therefore, design changes are easy.

For design changes, we need to alter content of control memory.

Micro-programmed control unit is slower than hardwared design.

0.2 Hardwired Control Design

Design methodologies considered for harddwired design of CU - (i) State-table method, (ii) Delay element method, and (iii) Sequence counter method.

0.2.1 State-table method

CU is a finite state machine (FSM). Design steps follow logic of designing an FSM.

This technique is suitable for small CU. There are several practical disadvantages -

(a) Number of states & input condition may be very large and thus state table size & amount of computation needed become excessive;

(b) Very complex circuit design.

Design debugging and subsequent maintainance of circuit become more difficult.

0.2.2 Delay element method

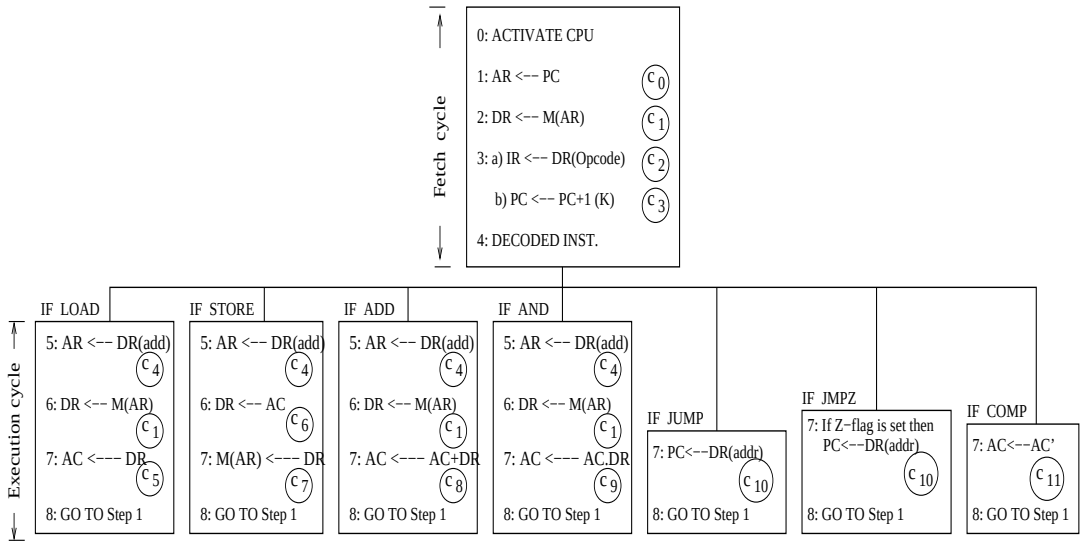


Figure 3: The sequence of control signals

Consider the function of CU shown in Figure 3, of a CPU with 7 macro-instructions.

A simplest version showing only sequence of control signals is in Figure 4.

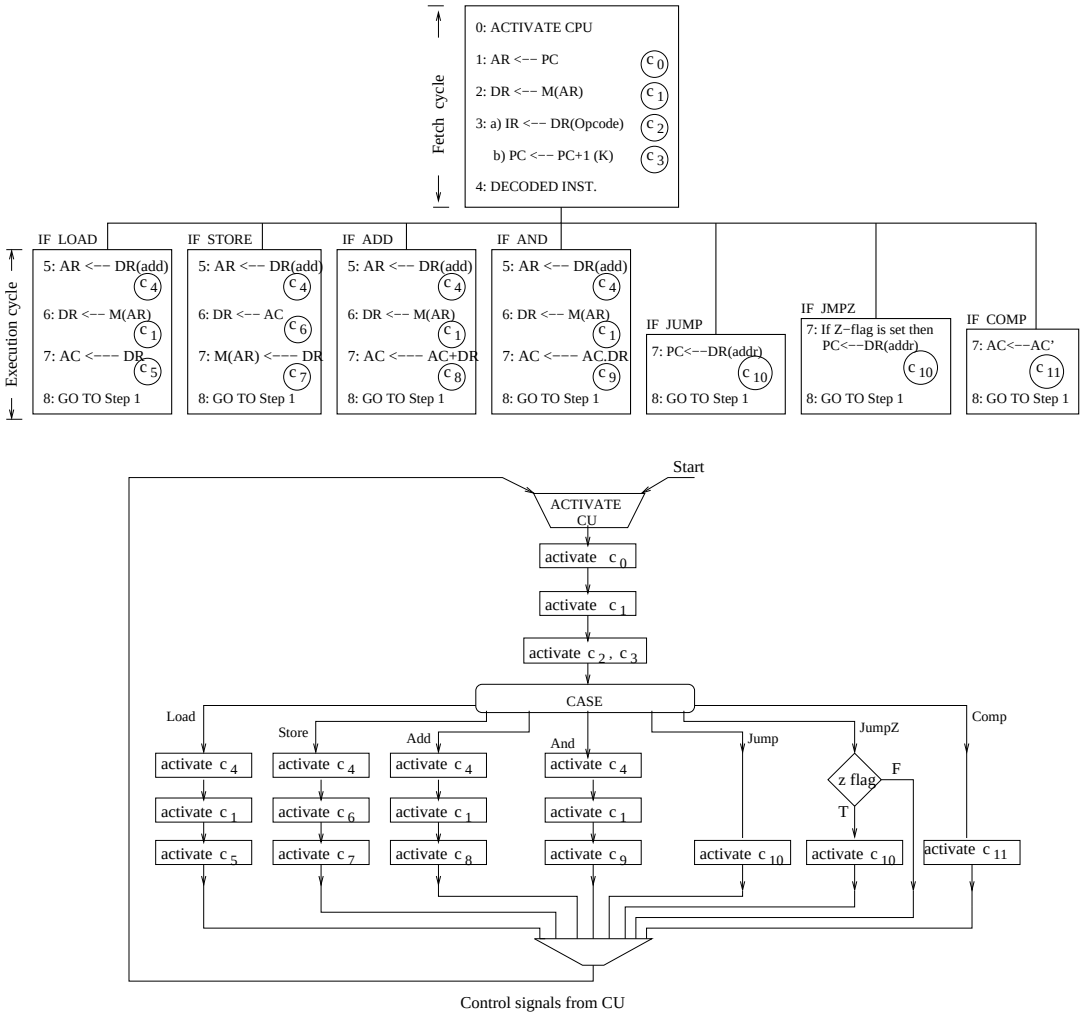
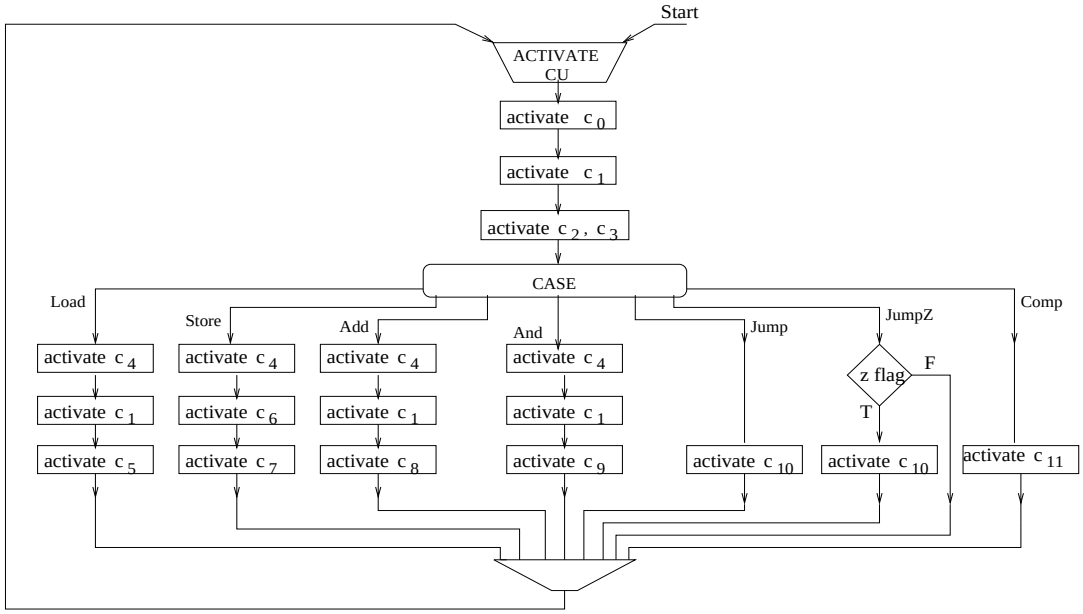


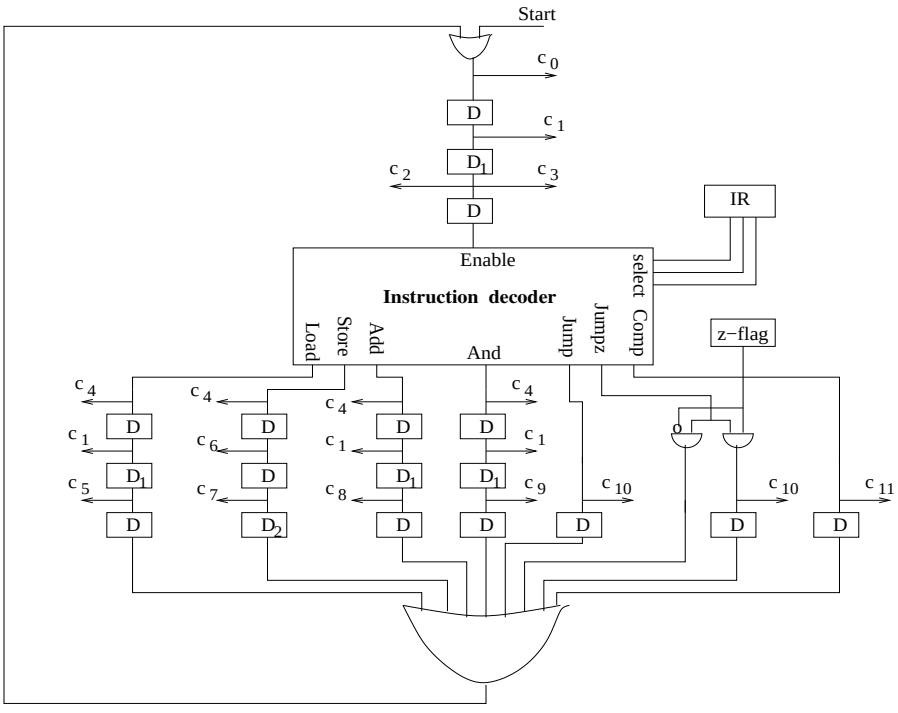
Figure 4: Sequence of control signals

CU designed with delay elements is shown in Figure 5.

CU using delay elements is designed directly from the diagram.



(a) Control signals from CU



(b) CU designed in dealy element method

Figure 5: Example design with delay element method

Following rules are followed for such a design:

a) In between two successive control signals (micro-instructions), a delay element is to be inserted.

   That is, an *assignment box* in control flow diagram is replaced by a edge and an *edge* is replaced by a delay element (Figure 6(a)).

b) The *k to 1 connector* is replaced by a k-input OR gate (Figure 6(b)).

c) *Decision box* is replaced by two 2-input AND gates (Figure 6(c)).

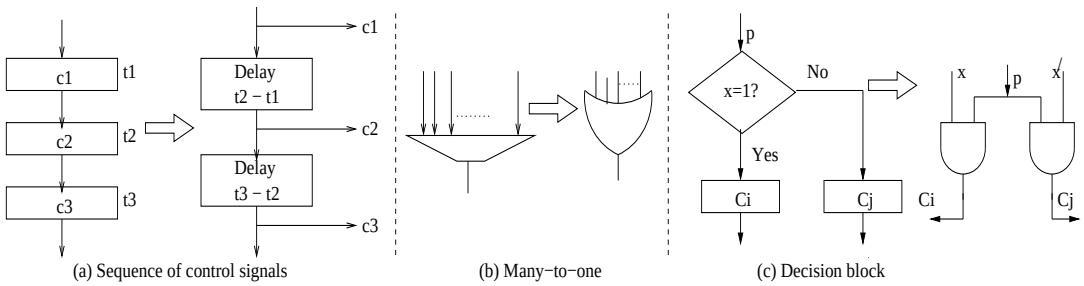


Figure 6: Control flow to logic conversion

Output of a delay element is to be a signal pulse of precise magnitude and duration.

In real design requirement, duration of different control signals may not be the same.

In example design of Figure 5(b), we assume all the control signals except c_1 and c_7 are to be of same duration (D).

Both c_1 and c_7 manage memory access (read/write) and, therefore, are to be activated for larger period of time.

In example, delays are shown as D_1 and D_2 respectively.

Limitations

Main limitation of control design following delay element method is the number of delay elements needed (approximately equal to the number of states O(N)).

Synchronization of so many widely distributed delay elements is difficult.

Design of a huge number of delay elements with specific delays is very expensive.

0.2.3 Sequence counter method

Design of CU using sequence counter method is based on the fact that control circuits perform a relatively small number of tasks repeatedly.

CU of Figure 5 repeats 6-tasks (that is, activate c_0, c_1, (c_2, c_3), c_4, c_1, c_5; if CPU executes a series of Load instructions).

A better representation of tasks performed repeatedly is shown in Figure 7.

Each pass in loop of Figure 7 constitutes an CPU instruction cycle.

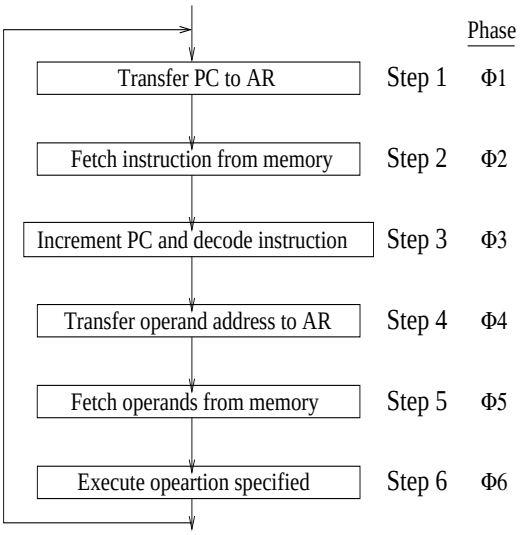


Figure 7: CPU behaviour represented as a single closed loop

If each step in loop is performed in an appropriately chosen clock period (ϕ), a CU can be built around a single modulo-6 sequence counter.

It is assumed that each step of Figure 7 including memory access is performed in one clock period.

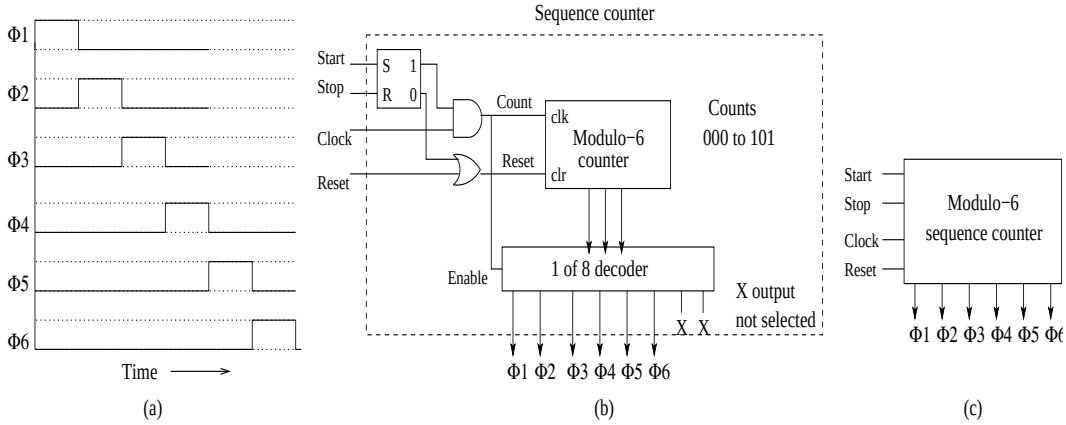


Figure 8: Modulo-6 sequence counter

A modulo-6 counter counts from 000 to 101 and then sets to 000.

Modulo-6 sequence counter consists of mod-6 counter + 1-of-8 decoder (Fig. 8).

Most significant two outputs of 1-of-8 decoder remain deactivated (X) all the time.

If a memory access requires two clock period, then the CU needs a modulo-8 sequence counter (in an iteration, maximum number of memory accesses is 2).

Φ_is effectively divide the time required for one complete cycle by k (6) equal parts.

Input lines Start/Stop and a flip-flop of Figure 8 are to turn the counter on and off.

A pulse on Start line causes counter to start counting its states.

A pulse on Stop line disconnects clock and resets the counter.

Design: Consider the single addressed CPU with following 7 macro instructions.

Mnemonic	*Description*
Load X	AC$\leftarrow$ M(X)
Store X	M(X)$\leftarrow$ AC
Add X	AC$\leftarrow$ AC + M(X)
And X	AC$\leftarrow$ AC $\wedge$ M(X)
Jump X	PC$\leftarrow$ X (unconditional branch)
Jumpz X	if AC = 0 then PC$\leftarrow$ X (conditional branch)
Comp	AC$\leftarrow AC'$ (complement accumulator)

The control signals that execute all the micro instructions are

$Control\ signal$	$Operation\ controlled$	
c_0	$AR \leftarrow PC$	
c_1	$DR \leftarrow M(AR)$	$Read\ Memory$
c_2	$PC \leftarrow PC + 1(k)$	
c_3	$IR \leftarrow DR(Opcode)$	
c_4	$AR \leftarrow DR(address)$	
c_5	$AC \leftarrow DR$	
c_6	$DR \leftarrow AC$	
c_7	$M(AR) \leftarrow DR$	$Write\ Memory$
c_8	$AC \leftarrow AC + DR$	
c_9	$AC \leftarrow AC \wedge DR$	
c_{10}	$PC \leftarrow DR(address)$	
c_{11}	$AC \leftarrow AC'$	$Complement\ Accumulator$

Figure 3 describes instruction fetch cycle.

Execution of a macro-instruction is then 6-step operation as depicted in Figure 7.

Figure 9 is the general structure of hardwired CU.

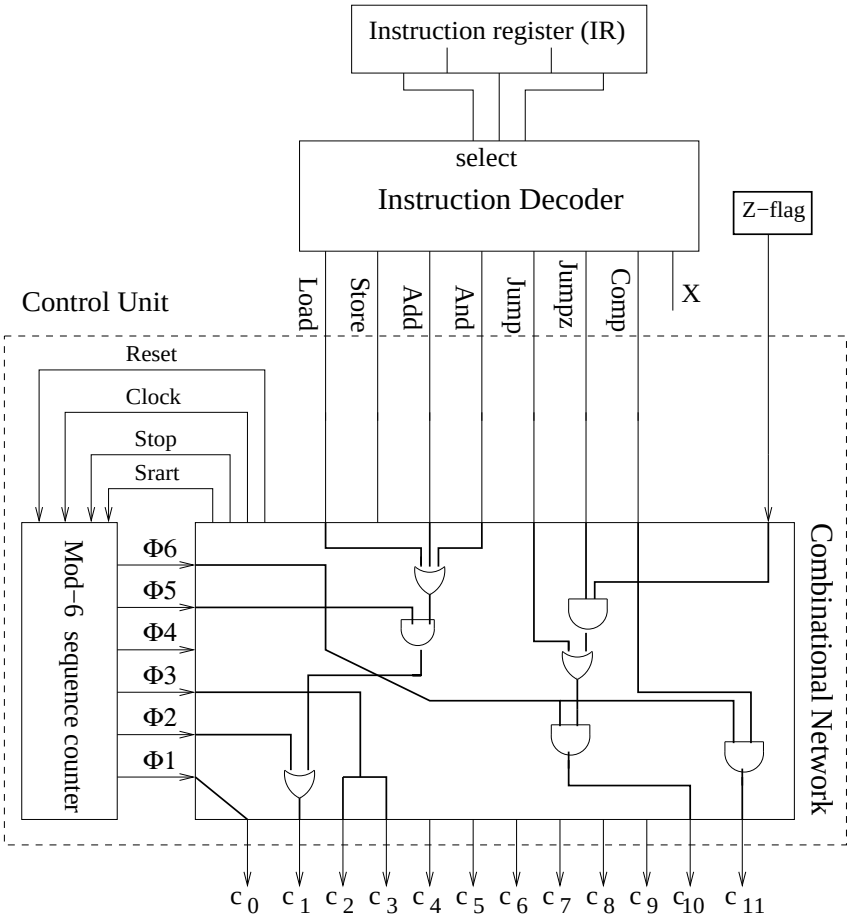


Figure 9: Hardwired CPU control unit around a sequence counter

Example: c_1 which causes a memory read, is activated when Φ_2 =1.

It is also activated when Φ_5 =1 during operand fetching for Load, Add, or And.

Therefore, c_1 can be defined as (follow Figure 10)

$$c_1 = \Phi_2 + \Phi_5 \text{ (Load + Add + And)}$$

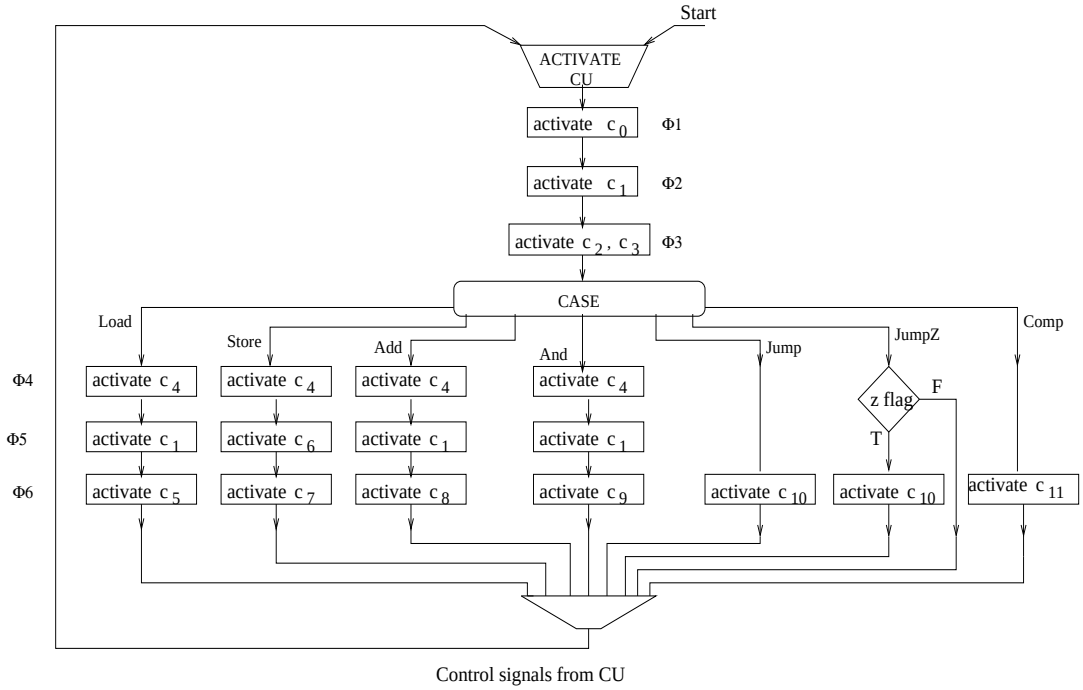


Figure 10: Control signal

The control signal c_i can be defined as:

$c_0 = \Phi_1$

c_1 = Φ_2 + Φ_5(Load + Add + And)

c_2 = Φ_3

c_3 = Φ_3

c_4 = Φ_4 (Load + Store + Add + And)

c_5 = Φ_6 . Load

c_6 = Φ_5 . Store

c_7 = Φ_6 . Store

c_8 = Φ_6 . Add

c_9 = Φ_6 . And

c_{10} = Φ_6 (Jump + Jumpz . z-flag)

c_{11} = Φ_6 . Comp

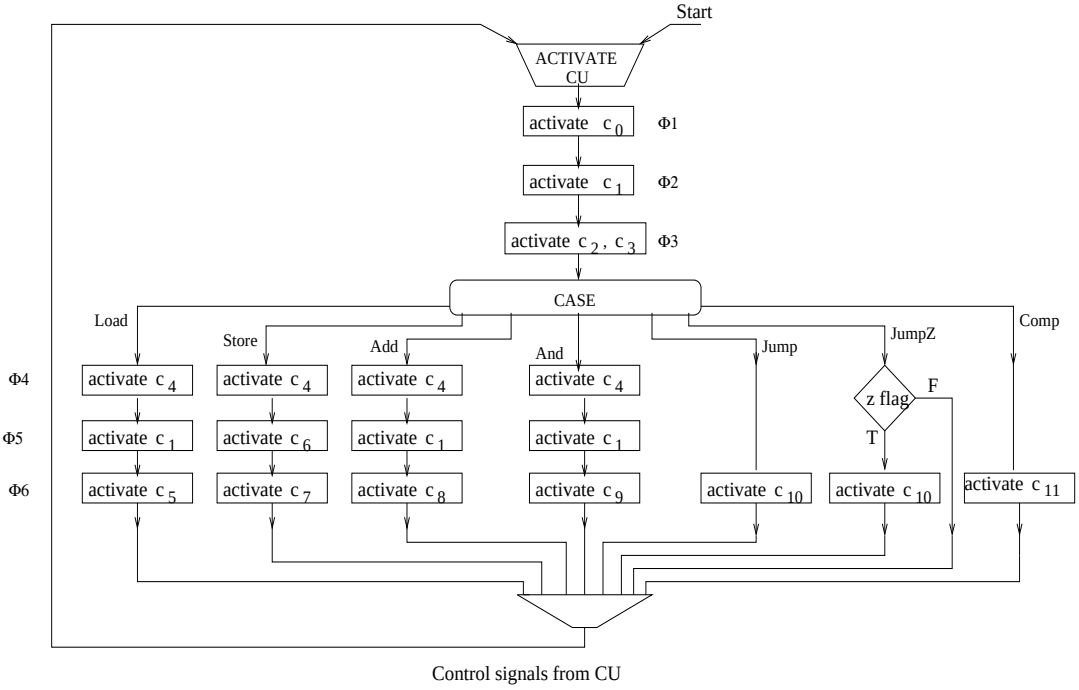
Thus the combinational circuit of Figure 9 is to be such that the c_is are realized.

Hardwired CU design is inflexible. It is extremely difficult to introduce any modifications and requires a huge design effort even for a moderate change in design.

Example: Assume introduction of a new instruction *Clear*.

Micro-instructions to be executed for *Clear* macro-instruction are

$$\begin{aligned} &\text{DR} \leftarrow \text{AC} && < c_6 >, \\ &\text{AC} \leftarrow \text{AC'} && < c_{11} >, \\ &\text{AC} \leftarrow \text{AC} \wedge \text{DR} && < c_9 >. \end{aligned}$$



CU of new CPU with 8 macro-instructions (7 + *Clear*), has also 12 control signals.

Modification needed is the change in combinational network.

Consider X output of instruction decoder (Figure 9) denotes *Clear*.

Then expressions for control signals c_6 and c_{11} are to be modified as

c_6 = Φ_5.Store + Φ_4.Clear [in unmodified c_6 = Φ_5 Store],
c_{11} = Φ_6.Comp + Φ_5.Clear [in unmodified c_{11} = Φ_6 Comp],
c_9 = Φ_6 (And + Clear) [in unmodified c_9 = Φ_6 And].

Such a minor change in logic expression may demand massive design effort to resolve different isuues (routing/placement/ power dissipation) almost afresh.