4/8/22

# Execution of a complete instruction

Execution of a complete instructions:

* fetch information from memory to CPU
* Store " to CPU register to memory
* Transfer of data b/w CPU registers
* Perform arithmetic or logic operation and store the result in CPU registers

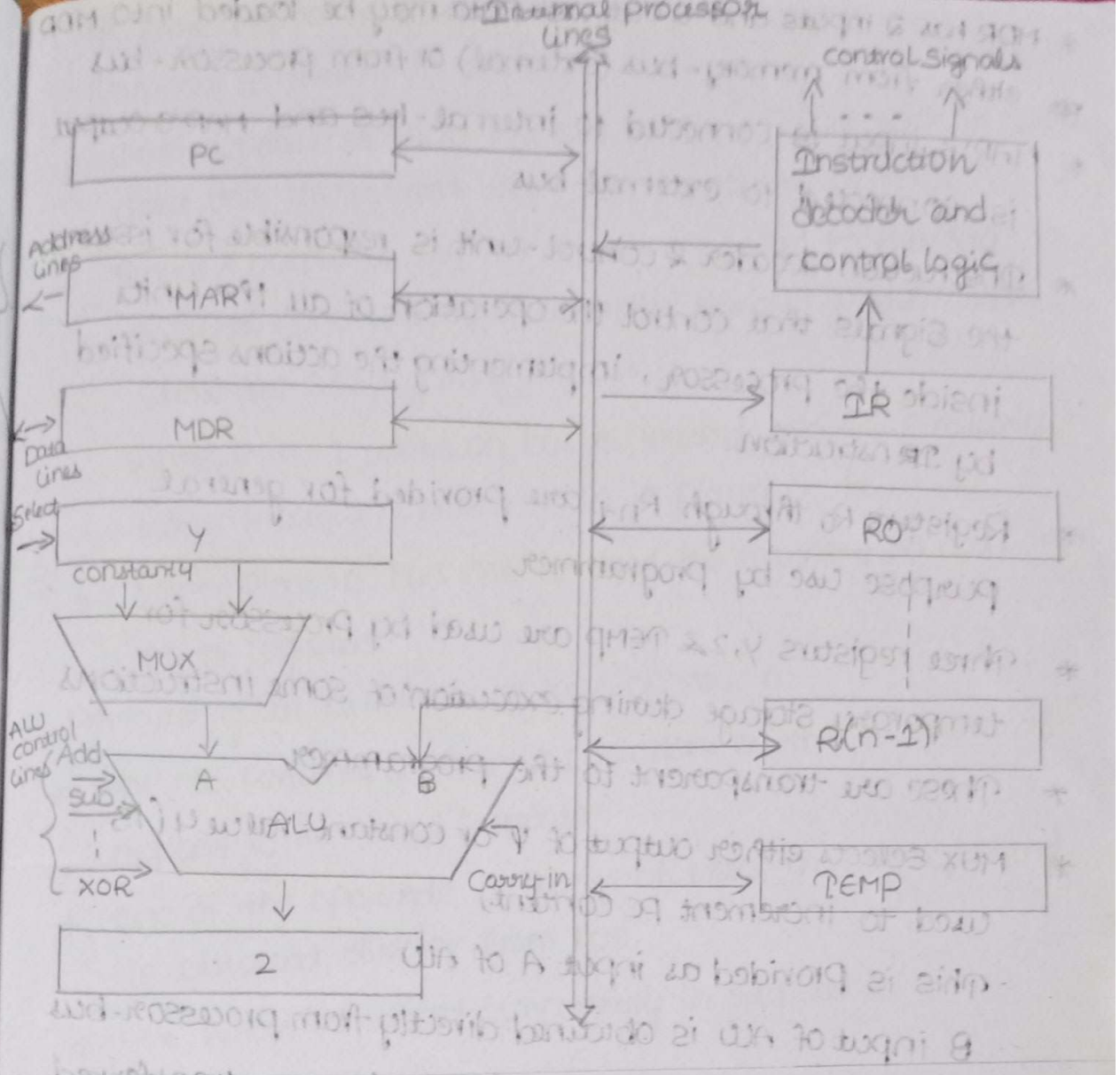
To execute an instruction, processor has to perform following 3 steps

1. Fetch contents of memory-location pointed by PC.
   contents of this memory-location is an instruction to be executed
   The instructions are loaded into IR, symbolically, this operation can be written as IR ← [[PC]]
2. Increment the contents of PC
   PC ← [PC] + 4
3. Carry out the actions specified by the instruction

* The first 2 steps are referred to as fetch phase;
  Step 3 is referred to as execution phase

* Execution control-sequence is written as follows

1. PCout, MARin, Read, Select4, Add, Zin
2. Zout, PCin, Yin, WMFC
3. $MDR_{out}$, IRin
4. RS, MARin, Read
5. MDRinE, WMFC
6. MDRout, Rd, End




Ex: Add (R3), R1

1. $PC_{out}$, $MAR_{in}$, Read, Select4, Add, $Z_{in}$
2. $Z_{out}$, $PC_{in}$, $Y_{in}$, WMFC
3. $MDR_{out}$, $IR_{in}$
4. $R3_{out}$, $MAR_{in}$, Read
5. $R1_{out}$, $Y_{in}$, WMFC
6. $MDR_{out}$, SelectY, Add, $Z_{in}$
7. $Z_{out}$, $R1_{in}$, End

* MDR has 2 inputs and 2 outputs. Data may be loaded into MD
⇒ either from memory-bus (external) or from processor-bus
* MAR's input is connected to internal-bus and MAR's output is connected to external-bus
* Instruction decoder & control-unit is responsible for issuing the signals that control the operation of all the units inside the processor, implementing the actions specified by Instruction.
* Registers $R_0$ through $R_{n-1}$ are provided for general purpose use by programmer
* Three registers Y, Z & TEMP are used by processor for temporary storage during execution of some instructions
* These are transparent to the programmer
* MUX selects either output of Y or constant-value 4 (is used to increment PC content)
This is provided as input A of ALU
B input of ALU is obtained directly from processor-bus
* As instruction execution progresses, data are transferred from one register to another, often passing through ALU to perform Arithmetic or logic operation

⇒ An instruction can be executed by performing one or more of the following operations:

1. Transfer a word of data from one processor-register to another or to the ALU
2. Perform arithmetic or a logic operation and store the result in a processor-register
3. Fetch the contents of a given memory-location and load them into a processor-register

4. Store a word of data from a processor

Register Transfers:

* Instruction execution involves a sequence of steps in which data are transferred from one register to another
* Input & output of register Ri is connected to bus via switches controlled by 2 controlled signals: Riin & Riout These are called gating signals
* When Riin = 1, data on bus is loaded into Ri. Similarly, when Riout = 1, content of Ri is placed on bus
* When Riout = 0, bus can be used for transferring data from other registers

Performing an arithmetic or logic operation:

* The ALU performs arithmetic operations on the 2 operands applied to its A and B inputs
* One of the operands is output of MUX & the other operand is obtained directly from bus
* The result is stored temporarily in register 2

Ex: [R3] ← [R1] + [R2]

1. R1out, Yin // transfer the contents of R1 to Y register
2. R2 out, SelectY, Add, Zin
   // R2 contents are transferred directly to B input of ALU
   // The numbers of added. Sum stored in register 2
3. Zout, R3 in // Sum is transferred to register R3

2. MOVE (Rs), Rd

* This instruction copies the contents of memory-location pointed to by Rs into Rd. This is a memory read operation. This requires the following actions.

→ fetch the instruction

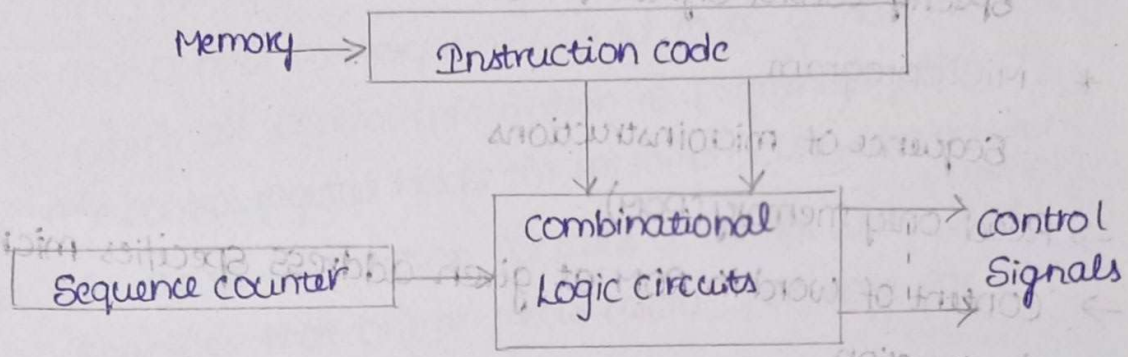
→ fetch the operand (i.e. the contents of the memory-location pointed by Rs)

# Microprogrammed control unit

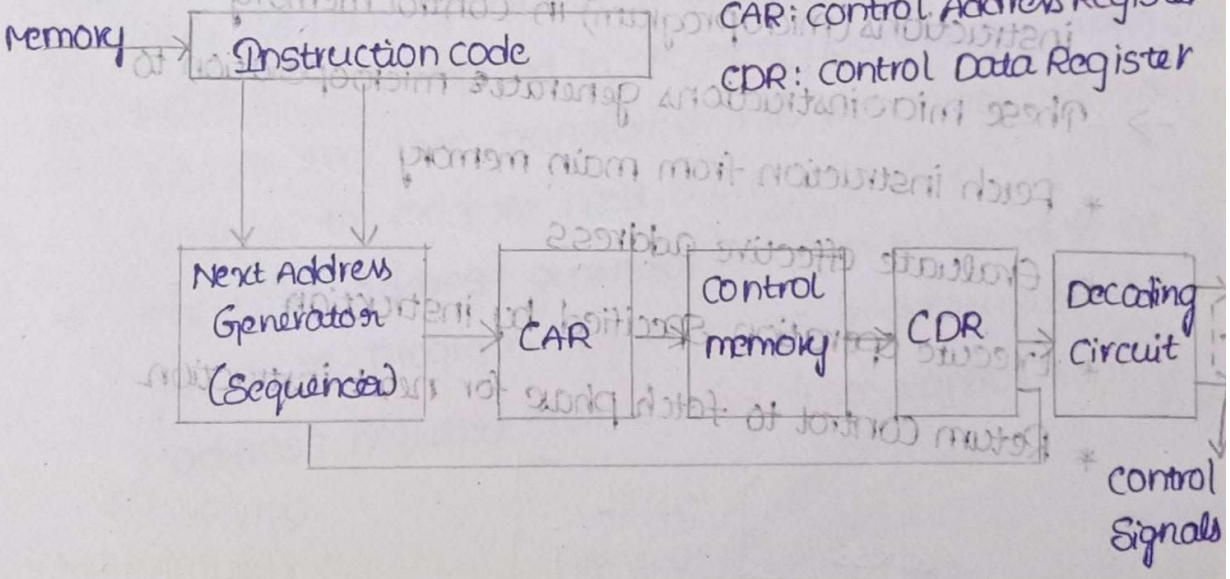
* The control unit is classified into two major categories
  - Hardwired control
  - Microprogrammed control
* The Hardwired control organization involves the control logic to be implemented with gates, flipflops, decoders, and other digital circuits.
* In Microprogrammed control, the micro-operations are performed by executing a microprogram consisting of micro-instructions

## Control unit Implementation:

### Hardwired:




### Microprogrammed:



* control signals

Group of bits used to select paths in multiplexers, decoders, arithmetic logic units

* control variables

Binary variables specify microoperations

⇒ certain microoperations initiated while others idle

* control word

String of 1's and 0's represent control variables.

* control memory

Memory contains control words

* Microinstructions

control words stored in control memory

specify control signals for execution of microoperations

* Microprogram

Sequence of microinstructions

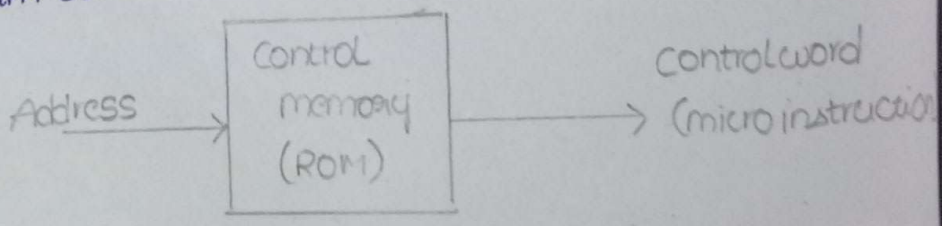
→ Read-only memory (ROM)

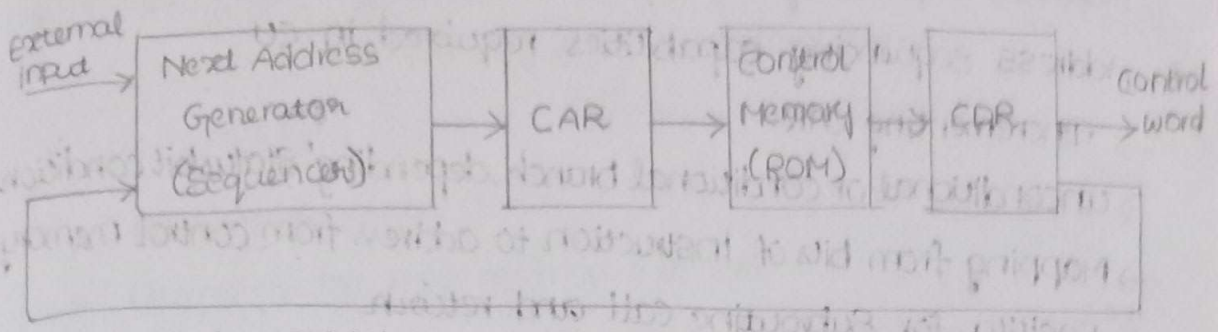
→ content of word in ROM at given address specifies micro instruction

→ Each user computer instruction initiates series of micro instructions (microprogram) in control memory

→ these microinstructions generate microoperation to

* fetch instruction from main memory
* Evaluate effective address
* Execute operation specified by instruction
* Return control to fetch phase for next instruction






Control memory:

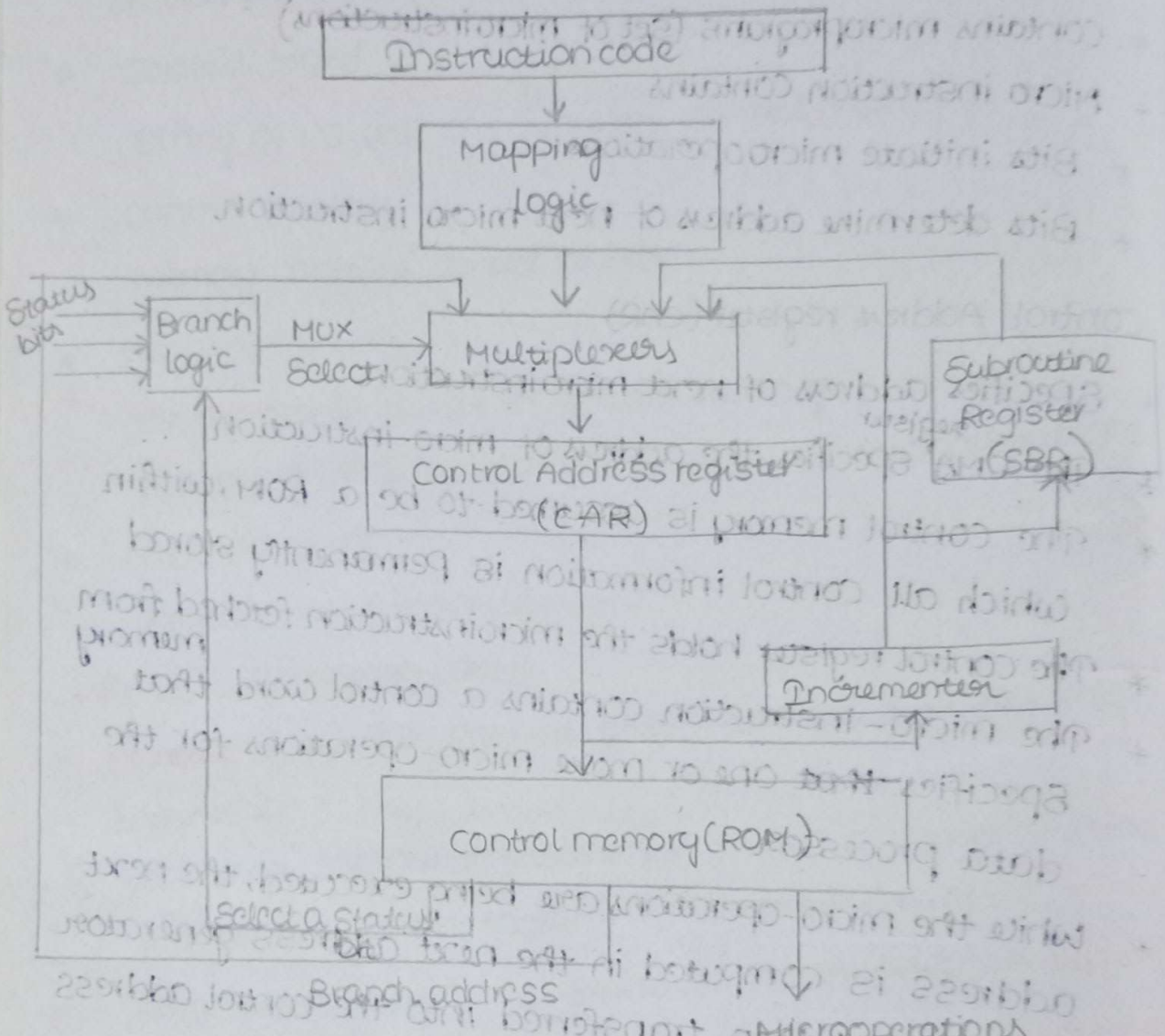
* Contains microprograms (set of micro instructions)
* Micro instruction contains
  * Bits initiate micro operations
  * Bits determine address of next micro instruction

Control Address register (CAR)

* Specifies address of next micro instruction
* The CMA register specifies the address of micro-instruction
* The control memory is assumed to be a ROM, within which all control information is permanently stored
* The control register holds the microinstruction fetched from memory
* The micro-instruction contains a control word that specifies one or more micro-operations for the data processor
* While the micro-operations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next micro instruction
* The next address generator is often referred to as a micro-program Sequencer, as it determines the address Sequence that is read from control memory

Address sequencing:

* Address sequencing capabilities required in CU
  - → Incrementing CAR
  - → unconditional or conditional branch, depending on Status bit conditions
  - → Mapping from bits of instruction to address from control memory
  - → Facility for Subroutine call and return



* Next address generator (microprogram sequencer)

Determines address sequence for control memory

* Microprogram sequencer functions (Incrementer)

→ Increment CAR by one

→ Transfer external address into CAR

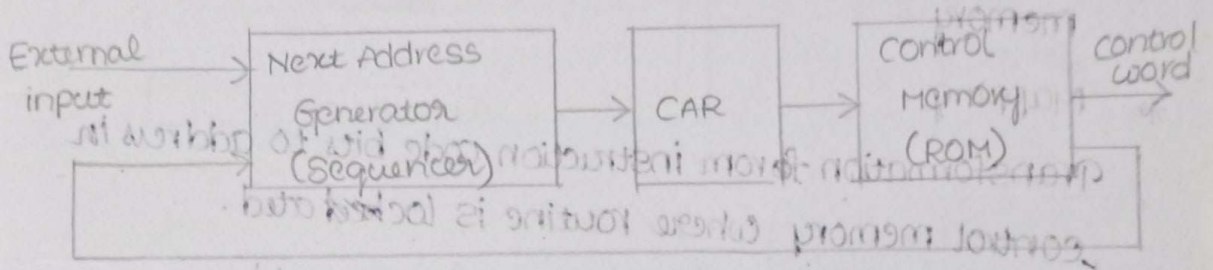
→ Load initial address into CAR to start control operations

* Control data register (CDR) - or pipeline register

→ Holds microinstruction read from control memory

→ Allows execution of micro operations specified by control word simultaneously with generation of next micro-instruction

* Control unit can operate without CDR




* Routine

Group of microinstructions stored in control memory

* Each computer instruction has its own microprogram routine to generate microoperations that execute the instruction

* Subroutine

Sequence of microinstructions used by other routines to accomplish particular task

Ex: Subroutine to generate effective address of operand for memory reference instruction

* Subroutine register (SBR)

Stores return address during subroutine call

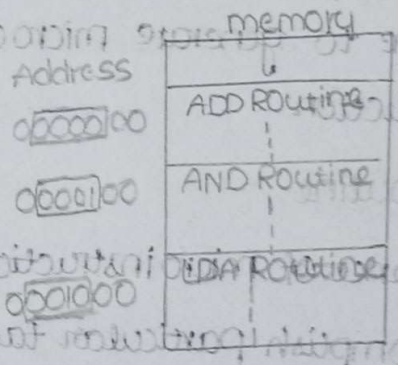
* Branching from one routine to another depends on status bit conditions
* Status bits provide parameter info such as
  → carry-out of adder
  → Sign bit of number
  → Mode bits of instruction
* Info in status bits can be tested and actions initiated based on their conditions: 1 or 0
* Unconditional branch
  Fix value of Status bit to 1
* Each computer instruction has its own microprogram routine stored in a given location of the control memory
* Mapping
  Transformation from instruction code bits to address in control memory where routine is located.

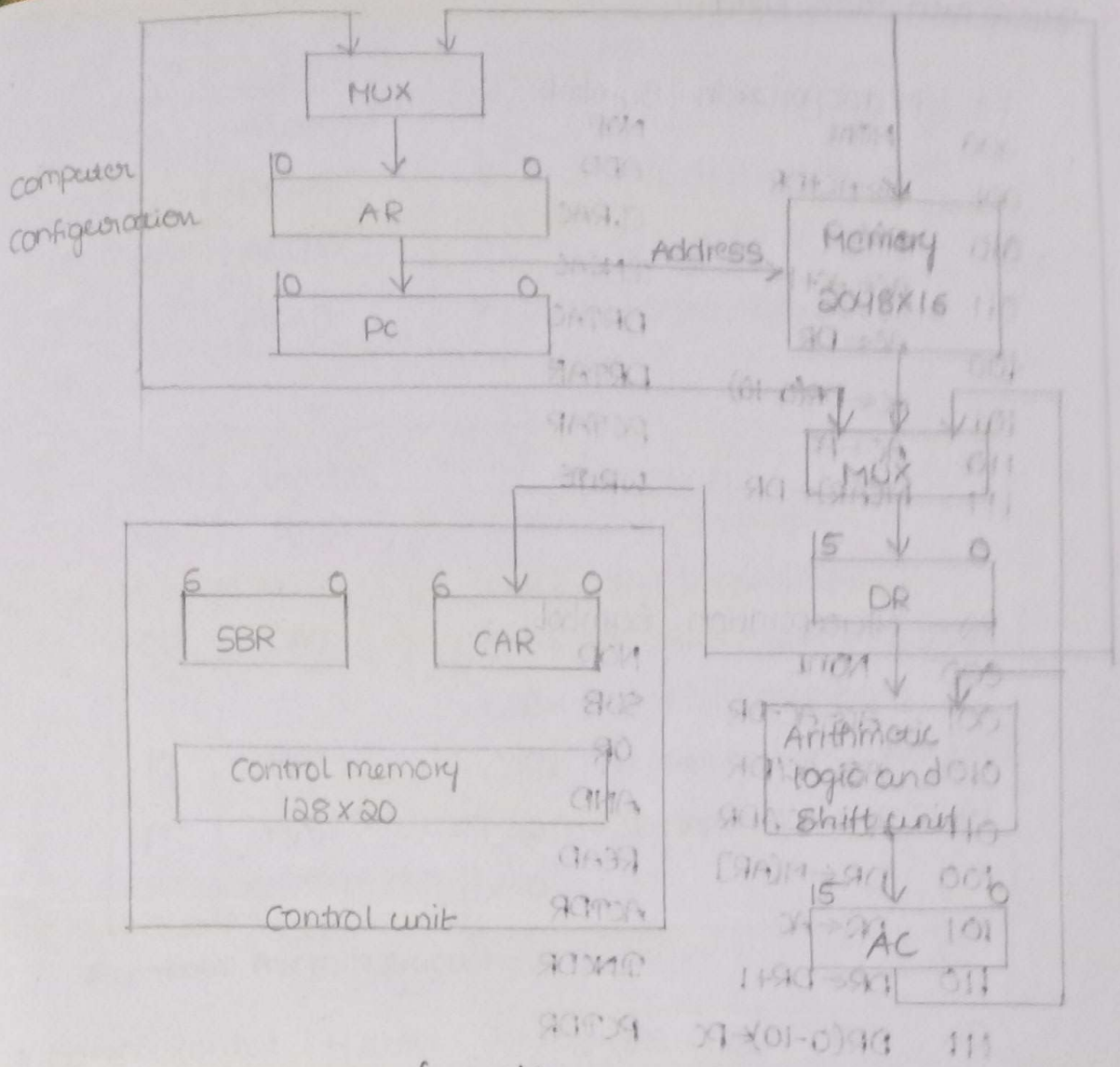
Ex: Mapping 4-bit operation code to 7-bit address

op-codes of instructions

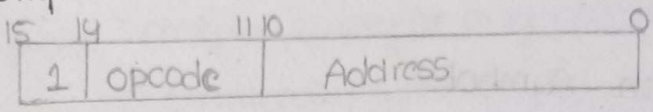
ADD 0000
AND 0001
LDA 0010

Mapping bits: XXXX00





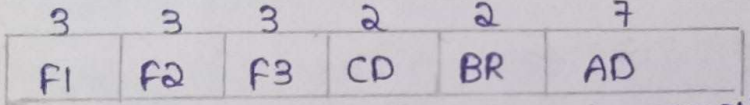

Computer instruction format:




Four computer instructions:

| Symbol | opcode | Description |
|---|---|---|
| ADD | 0000 | AC ← AC + M[EA] |
| BRANCH | 0001 | if(AC < 0) then (PC ← EA) |
| STORE | 0010 | M[EA] ← AC |
| EXCHANGE | 0011 | AC ← M[EA], M[EA] ← AC |

Micro instruction format:




F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

Microinstruction fields:

| F1 | Microoperation | Symbol |
|---|---|---|
| 000 | None | NOP |
| 001 | AC←AC+DR | ADD |
| 010 | AC←0 | CLRAC |
| 011 | AC←AC+1 | INCAC |
| 100 | AC←DR | DRTAC |
| 101 | AC←DR(0-10) | DRTAR |
| 110 | AC←PC | PCTAR |
| 111 | M[AR]←DR | WRITE |

| F2 | Microoperation | Symbol |
|---|---|---|
| 000 | None | NOP |
| 001 | AC←AC-DR | SUB |
| 010 | AC←AC∨DR | OR |
| 011 | AC←AC∧DR | AND |
| 100 | DR←M[AR] | READ |
| 101 | DR←AC | ACTDR |
| 110 | DR←DR+1 | INCDR |
| 111 | DR(0-10)←PC | PCTDR |

| F3 | Microoperation | Symbol |
|---|---|---|
| 000 | None | NOP |
| 001 | AC←AC⊕DR | XOR |
| 010 | AC←AC' | COM |
| 011 | AC←shlAC | SHL |
| 100 | AC←shrAC | SHR |
| 101 | PC←PC+1 | INCPC |
| 110 | PC←AR | ARTPC |
| 111 | Reserved | |

| CD | condition | Symbol | Comments |
|---|---|---|---|
| 00 | Always=1 | U | : unconditional branch |
| 01 | DR(15) | I | Indirect Address bit |
| 10 | AC(15) | S | Sign bit of AC |
| 11 | AC=0 | Z | zero value in AC |

| BR | Symbol | Function |
|---|---|---|
| 00 | JMP | CAR ← AD if condition = 1 |
| | | CAR ← CAR+1 if condition = 0 |
| 01 | CALL | CAR ← AD, SBR ← CAR+1 if condition = 1 |
| | | CAR ← CAR+1, if condition = 0 |
| 10 | RET | CAR ← SBR (return from subroutine) |
| 11 | MAP | CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0 |

Symbolic microinstruction:

* Sample format :

| Label: | Micro-OPS | CD | BR | AD |
|---|---|---|---|---|

* Label may be empty or may specify symbolic address terminated with colon
* Micro-OPS consists of 1, 2 or 3 symbols separated by commas
* CD one of {U, I, S, Z}
  - U: unconditional branch
  - I: Indirect address bit
  - S: Sign of AC
  - Z: zero value in AC
* BR one of {JMP, CALL, RET, MAP}
* AD one of {Symbolic address, NEXT, empty}

* Fetch routine

- Read instruction from memory
- Decode instruction and update PC

Microinstructions for fetch routine:

AR ← PC

DR ← M[AR], PC ← PC+1

AR ← DR(0-10), CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

Symbolic microprogram for fetch routine:

```
        ORG 64
FETCH:  PCTAR     U   JMP   NEXT
        READ,INCPC U  JMP   NEXT
        DRTAR     U   MAP
```

Binary microprogram for fetch routine:

| Binary address | F1 | F2 | F3 | CD | BR | AD |
|---|---|---|---|---|---|---|
| 1000000 | 110 | 000 | 000 | 00 | 00 | 1000001 |
| 1000001 | 000 | 100 | 101 | 00 | 00 | 1000010 |
| 1000010 | 101 | 000 | 000 | 00 | 11 | 0000000 |

Symbolic microprogram:

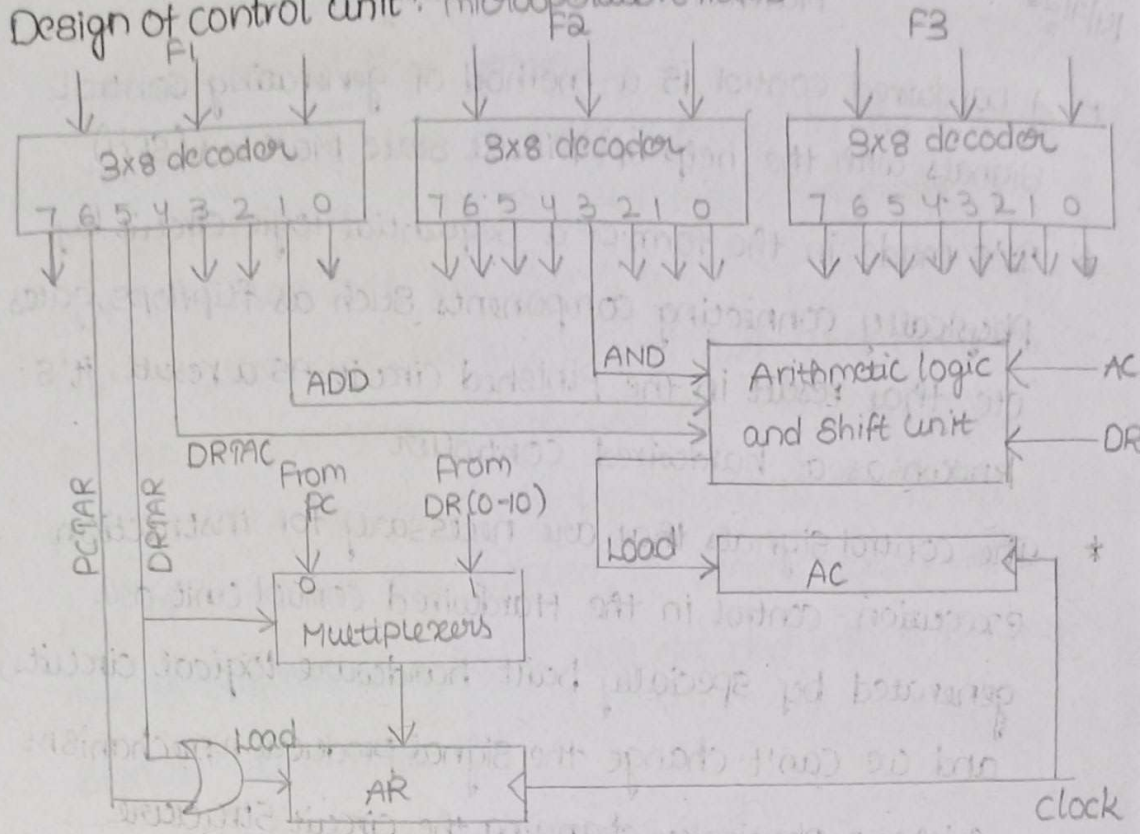
* control memory: 128 20-bit words
* FIRST 64 words: Routines for 16 machine instruction
* Last 64 words: used for other purpose (e.g. fetch routine and other Subroutines)
* Mapping: op-code xxxx into 0xxxx00, first address for 16 routines are 0 (0 0000 00), 4 (0 0001 00), 8, 12, 16, 20, ---, 60

Partial Symbolic Microprogram:

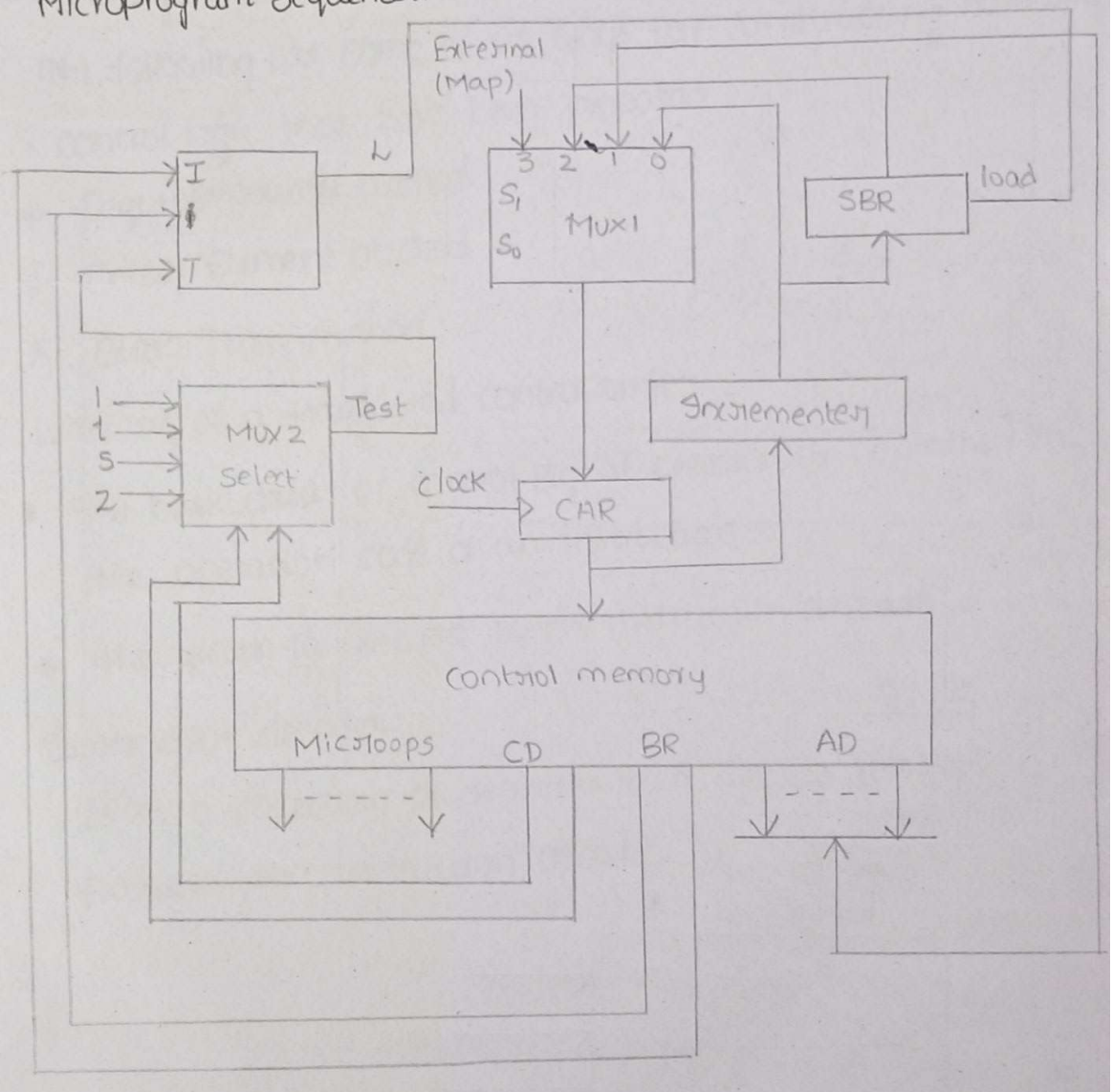
| Label | Microops | CD | BR | AD |
|---|---|---|---|---|
| ADD: | ORG 0 | | | |
| | NOP | I | CALL | INDRCT |
| | READ | U | JMP | NEXT |
| | ADD | U | JMP | FETCH |
| | ORG 4 | | | |
| BRANCH: | NOP | S | JMP | OVER |
| | NOP | U | JMP | FETCH |
| OVER: | NOP | I | CALL | INDRCT |
| | ARTPC | U | JMP | FETCH |
| | ORG 8 | | | |
| STORE: | NOP | I | CALL | INDRCT |
| | ACTDR | U | JMP | NEXT |
| | WRITE | U | JMP | FETCH |
| | ORG 12 | | | |
| EXCHANGE: | NOP | I | CALL | INDRCT |
| | READ | U | JMP | NEXT |
| | ACTDR, DRTAC | U | JMP | NEXT |
| | WRITE | U | JMP | FETCH |
| | ORG 64 | | | |
| FETCH: | PCTAR | U | JMP | NEXT |
| | READ, INCPC | U | JMP | NEXT |
| | DRTAR | U | MAP | |
| | | | JMP | NEXT |
| INDRCT | READ | U | | |
| | DRTAR | U | RET | |

| | Address | | Binary Micro instruction | | | | | |
|---|---|---|---|---|---|---|---|---|
| Micro Routine | Decimal | Binary | F1 | F2 | F3 | CD | BR | AD |
| ADD | 0 | 0000000 | 000 | 000 | 000 | 01 | 01 | 1000011 |
| | 1 | 0000001 | 000 | 100 | 000 | 00 | 00 | 0000010 |
| | 2 | 0000010 | 001 | 000 | 000 | 00 | 00 | 1000000 |
| | 3 | 0000011 | 000 | 000 | 000 | 00 | 00 | 1000000 |
| BRANCH | 4 | 0000100 | 000 | 000 | 000 | 10 | 00 | 0000110 |
| | 5 | 0000101 | 000 | 000 | 000 | 00 | 00 | 1000000 |
| | 6 | 0000110 | 000 | 000 | 000 | 01 | 01 | 1000011 |
| | 7 | 0000111 | 000 | 000 | 110 | 00 | 00 | 1000000 |
| STORE | 8 | 0001000 | 000 | 000 | 000 | 01 | 00 | 1000011 |
| | 9 | 0001001 | 000 | 101 | 000 | 00 | 00 | 0001010 |
| | 10 | 0001010 | 111 | 000 | 000 | 00 | 00 | 1000000 |
| | 11 | 0001011 | 000 | 000 | 000 | 00 | 00 | 1000000 |
| EXCHANGE | 12 | 0001100 | 000 | 000 | 000 | 01 | 01 | 1000011 |
| | 13 | 0001101 | 000 | 000 | 000 | 00 | 00 | 0001100 |
| | 14 | 0001110 | 100 | 101 | 000 | 00 | 00 | 0001111 |
| | 15 | 0001111 | 111 | 000 | 000 | 00 | 00 | 1000000 |
| FETCH | 64 | 1000000 | 110 | 000 | 000 | 00 | 00 | 1000001 |
| | 65 | 1000001 | 000 | 100 | 101 | 00 | 00 | 1000010 |
| | 66 | 1000010 | 101 | 000 | 000 | 00 | 11 | 0000000 |
| INDRCT | 67 | 1000011 | 000 | 000 | 000 | 00 | 00 | 1000100 |
| | 68 | 1000100 | 101 | 000 | 000 | 00 | 10 | 0000000 |

Design of control unit: microoperation fields



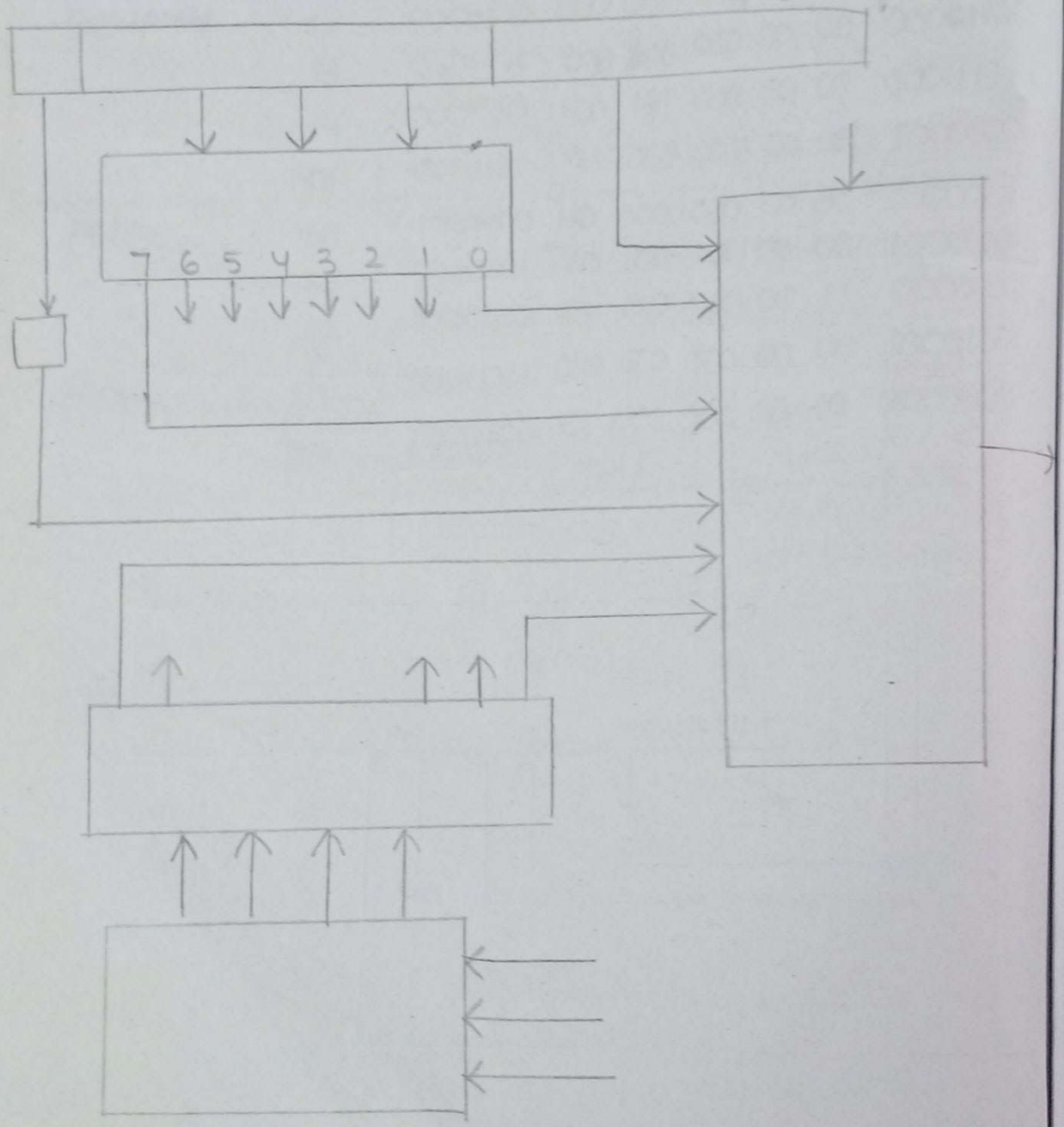

Microprogram Sequencer:



14/7/22

# Hardwired control unit

* A hardwired control is a method of generating control signals with the help of Finette State Machines (SFM)
* It's made in the form of a sequential logic circuit by physically connecting components such as flipflops, gates, etc that result in the finished circuit. As a result, it's known as a hardwired controller
* The control signals that are necessary for instruction execution control in the Hardwired control unit are generated by specially built hardware logical circuits, and we can't change the signal production mechanism without physically changing the circuit structure



* It consists of two decoders, a sequence
number of control logic gates
* An instruction read from memory is placed in the Instruction register (IR) where it is decoded into three parts
I bit, operation code and bits 0 to 11
* The operation code bit is decoded with 3x8 decoder producing 8 outputs D0 through D7
* Bit 15 of the instruction is transferred to a flip-flop I
* And operand bits are applied to control logic gates
* The 16 outputs of 4-bit SC are decoded into 16 timing signals T0 through T15

⇒ Designing of Hardwired control unit:

The following are some of the ways for constructing hardwired control logic that have been proposed

* Sequence counter method
* Delay Element method
* State Table method.

Working of a Hardwired control unit:-

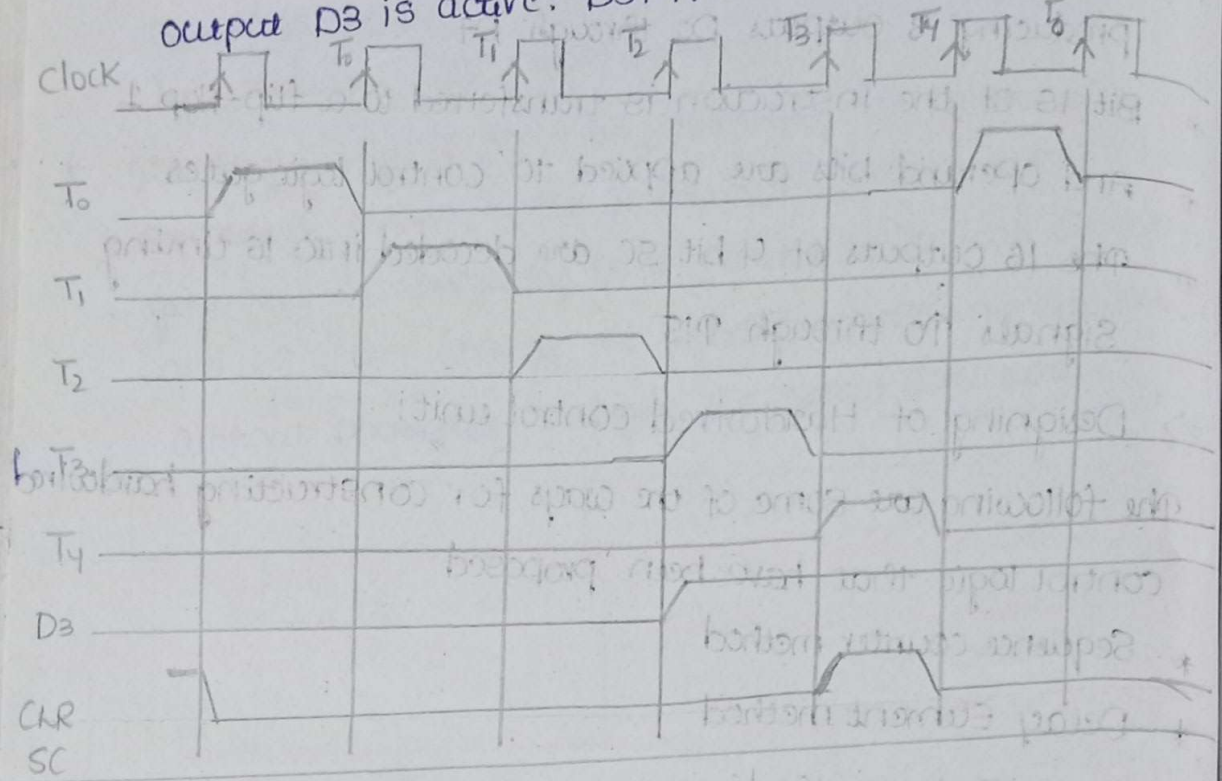
* The basic data for control signal creation is contained in the operation code of an instruction
* The opcode is decoded in the instruction decoder.

Instruction decoder:

It is a collection of decoders that decode various fields of the instruction opcode.

Timing signals of Hardwired control unit:

* Generated by 4-bit sequence counter and 4x16 decoder
* The SC can be incremented or cleared
  Ex: T0, T1, T2, T3, T4, T0, T1 - - - - -
* Assume: At time T4, SC is cleared to 0 if decoder output D3 is active: D3T4: SC ← 0




Advantages:

1. Because of the use of combinational circuits to generate signals, Hardwired control unit is fast
2. It depends on no. of gates, how much delay can occur in generation of control signals
3. It can be optimized to produce the fast mode of operation
4. Faster than micro-programmed control unit

Disadvantages:

1. The complexity of the design increases as we require more control signals to be generated
2. Modifications in the control signals are very difficult because it requires rearranging of wires in the hardware circuit

3. Adding a new feature is difficult & complex
4. Difficult to test & correct mistakes in the original design
5. It is Expensive

| Hardwired control unit | Microprogrammed control unit |
| --- | --- |
| * It generates the control signals using logic circuits (needed for the processor) | * Generates control signals with the help of micro-instructions stored in control memory |
| * It is faster as the required control signals are generated with the help of hardware | * This is slower than the other as micro instructions are used for generating signals here |
| * Difficult to modify as the control signals that need to be generated are hard wired | * Easy to modify as the modification need to be done only at the instruction level |
| * More costlier as everything has to be realised in terms of logic gates | * Less costlier than hardwired control as only micro instructions are used for generating control signals |
| * It cannot handle complex instructions as the circuit design for it becomes complex | * It can handle complex instructions |
| * Only limited no. of instructions are used due to the hardware implementation | * control signals for many instructions can be generated |
| * used in computer that makes use of RISC | * used in computer that makes use of CISC |

Comparison:

| Attributes | Hardwired control | Microprogramming control |
|---|---|---|
| Speed | Fast | Slow |
| Cost of Implementation | More | Cheaper |
| Flexibility | Difficult to modify | Flexible |
| Ability to Handle complex Instruction | Difficult | Easier |
| Decoding | complex | Easy |
| Application | RISC | CISC |
| Instruction set size | Small | Large |
| control memory | Absent | Present |