# UNIT-III CONTROL UNIT DESIGN

- INTRODUCTION
- CONTROL TRANSFER
- FETCH CYCLE
- INSTRUCTION INTERPRETATION AND EXECUTION
- HARDWIRED CONTROL
- MICROPROGRAMMED CONTROL

1

# INTRODUCTION

- Important component of CPU is the control unit
  - The control unit generates the signals for sequencing the operations in the datapath
    - A sequential circuit with states that *dictate the control signals* for the system
    - Using status conditions and control inputs, the sequential control unit *determines the next state* in which additional microoperations are activated.
  - Hardwired Control
    - The control unit is implemented to provide a particular digital function
  - Microprogrammed Control
    - The control unit's binary control values are stores as words in a microprogrammed control (usually ROM).
    - Each word in the control contains a microinstruction
    - A sequence of microinstructions constitutes a microprogram

2

# TERMINOLOGY Used in Control Unit

**Microprogram**
- Program stored in memory that generates all the control signals required to execute the instruction set correctly.
  - Consists of microinstructions
  - Microinstruction
    - Contains a control word and a sequencing word

**Control Word** - Control information required for one clock cycle
**Sequencing Word** - Information needed to decide the next microinstruction address

**Control Memory (Control Storage: CS)**
- Storage in the microprogrammed control unit to store the microprogram

**Writeable Control Memory (Writeable Control Storage: WCS)**
- CS whose contents can be modified
  - -> Allows the microprogram can be changed
  - -> Instruction set can be changed or modified

**Dynamic Microprogramming**
- Computer system whose control unit is implemented with a microprogram in WCS
- Microprogram can be changed by a systems programmer or a user

3

# TERMINOLOGY…

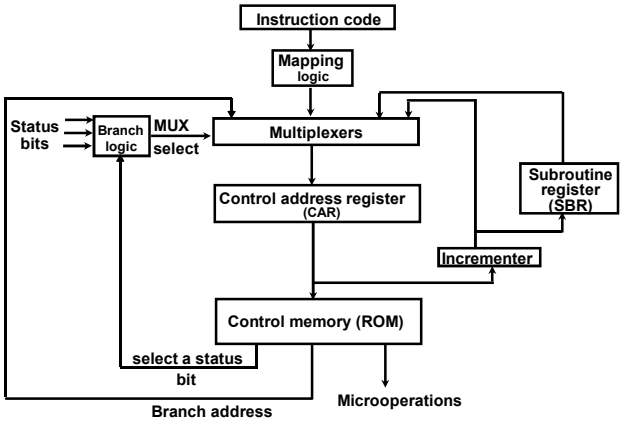
***Sequencer (Micro program Sequencer)***

A Micro program based Control Unit that determines the Microinstruction address to be executed in the next clock cycle

- In-line Sequencing
- Branch
- Conditional Branch
- Subroutine
- Loop
- Instruction OP-code mapping

4

# MICROINSTRUCTION SEQUENCING

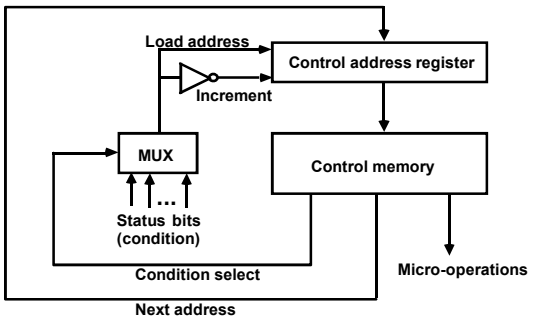



**Sequencing Capabilities Required in a Control Storage**

- Incrementing of the control address register
- Unconditional and conditional branches
- A mapping process from the bits of the machine instruction to an address for control memory
- A facility for subroutine call and return

*Sequencing*

# CONDITIONAL BRANCH




**Conditional Branch**

If *Condition* is true, then *Branch* (address from the next address field of the current microinstruction) else *Fall Through*

Conditions to Test: O(overflow), N(negative), Z(zero), C(carry), etc.

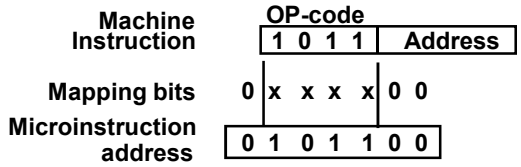
**Unconditional Branch**

Fixing the value of one status bit at the input of the multiplexer to 1

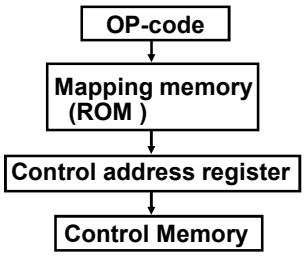
*Sequencing*

## MAPPING OF INSTRUCTIONS TO MICROROUTINES

**Mapping from the OP-code of an instruction to the address of the Microinstruction which is the starting microinstruction of its execution microprogram**



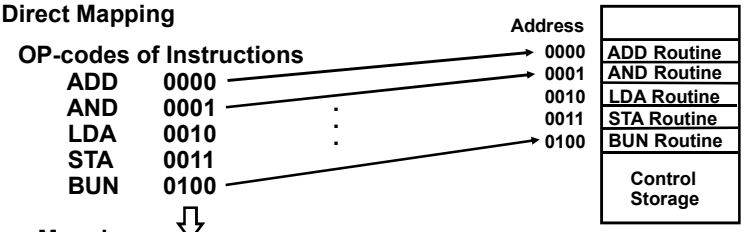

**Mapping function implemented by ROM**

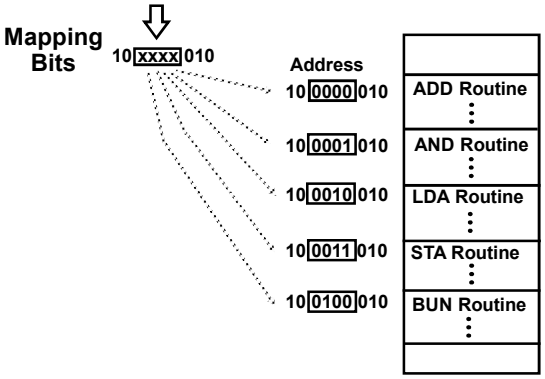



*Sequencing*

## MAPPING OF INSTRUCTIONS

**Direct Mapping**

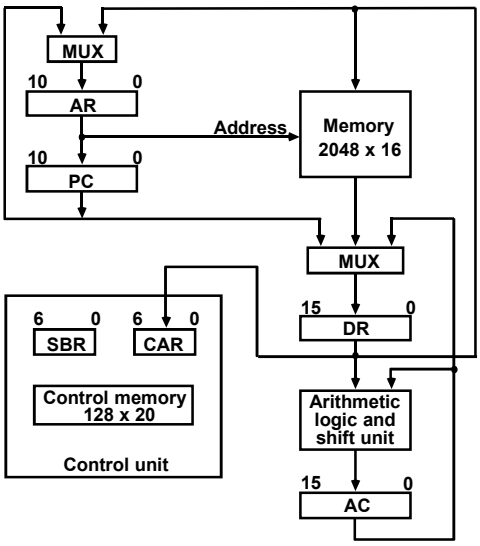


*Microprogram*

# MICROPROGRAM EXAMPLE

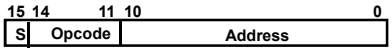
**Example Computer Configuration**




*Microprogram*

# MACHINE INSTRUCTION FORMAT

**Machine instruction format**

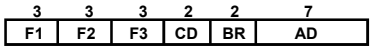



**Sample machine instructions**

| Symbol | OP-code | 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 |

EA is the effective address

**Microinstruction Format**




F1, F2, F3: Microoperation fields
CD: Condition for branching
BR: Branch field
AD: Address field

## MICROINSTRUCTION FIELD DESCRIPTIONS - F1,F2,F3

| 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 | AR ← DR(0-10) | DRTAR |
| 110 | AR ← 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 ← shl AC | SHL |
| 100 | AC ← shr AC | SHR |
| 101 | PC ← PC + 1 | INCPC |
| 110 | PC ← AR | ARTPC |
| 111 | Reserved | |



## MICROINSTRUCTION FIELD DESCRIPTIONS - CD, BR

| 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 MICROINSTRUCTIONS

- **Symbols are used in microinstructions as in assembly language**
- **A symbolic microprogram can be translated into its binary equivalent by a microprogram assembler.**

**Sample Format**

**five fields:** **label; micro-ops; CD; BR; AD**

**Label:** **may be empty or may specify a symbolic address terminated with a colon**

**Micro-ops: consists of one, two, or three symbols separated by commas**

**CD:** **one of {U, I, S, Z}, where** **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}**



# SYMBOLIC MICROPROGRAM - FETCH ROUTINE

**During FETCH, Read an instruction from memory and decode the instruction and update PC**

**Sequence of microoperations in the fetch cycle:**

```
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 the fetch cycle:**

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

**Binary equivalents translated by an assembler**

| 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 |

*Microprogram*

# SYMBOLIC MICROPROGRAM

- Control Storage: 128 20-bit words
- The first 64 words: Routines for the 16 machine instructions
- The last 64 words: Used for other purpose (e.g., fetch routine and other subroutines)
- Mapping: OP-code XXXX into 0XXXX00, the first address for the 16 routines are 0(0 0000 00), 4(0 0001 00), 8, 12, 16, 20, ..., 60

**Partial Symbolic Microprogram**

| Label | Microops | CD | BR | AD |
|---|---|---|---|---|
| | ORG 0 | | | |
| ADD: | 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 | |
| INDRCT: | READ | U | JMP | NEXT |
| | DRTAR | U | RET | |

*Microprogram*

# BINARY MICROPROGRAM

| | Address | | Binary Microinstruction | | | | | |
|---|---|---|---|---|---|---|---|---|
| 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 | 01 | 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 | 001 | 000 | 000 | 00 | 00 | 0001110 |
| | 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 | 100 | 000 | 00 | 00 | 1000100 |
| | 68 | 1000100 | 101 | 000 | 000 | 00 | 10 | 0000000 |

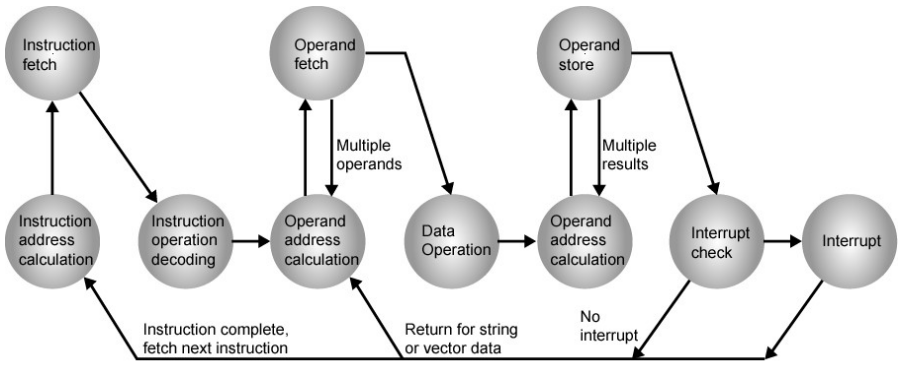
**This microprogram can be implemented using ROM**

# CPU functions

1. Fetch and instruction sequencing (fetch cycle)-Generates control signal to fetch instruction from memory and the sequence of operations involved in processing an instruction
2. Instruction interpretation and execution (execution cycle)-Tasks involved are
   - » Interpreting the operand addressing mode implied in the operation code and fetching the operands
   - » Sequencing the successive micro operations on the data path to execute the operation code specified in the instruction
3. Interrupt processing (interrupt cycle)-Process interrupt. Tasks are
   - » Suspend execution of current program
   - » Save context
   - » Set PC to start address of interrupt handler routine
   - » Process interrupt
   - » Restore context and continue interrupted program

17

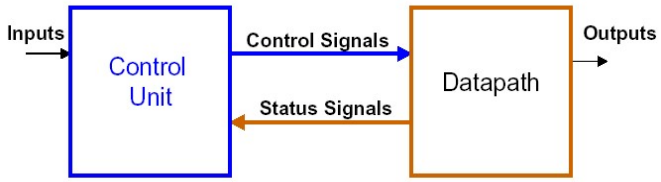
# Instruction Cycle (with Interrupts) - State Diagram




18

# Interaction Between Data and Control Units

- Digital Systems can be partitioned in a **Datapath** and **Control Unit**




- Selects the microoperation
- Determines the sequence (based on status and input signals)
- Design: State diagram or ASM:
  - Microoperations
  - Sequence

- Registers
- MUXes, ALU, Shifters, Comb. Circuits and buses
- Implements microoperations (under control of the control unit)

19

# Control Unit

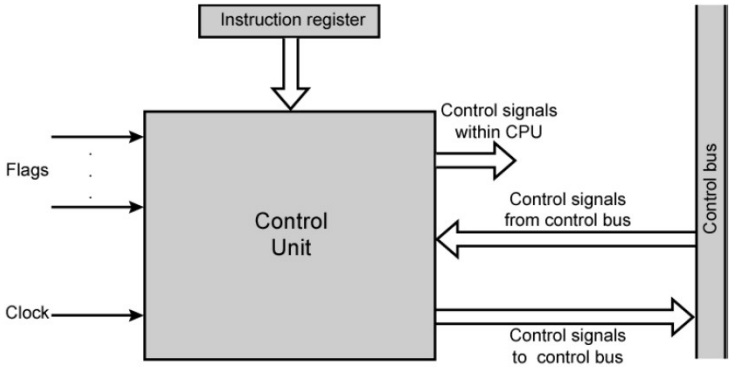



Figure 16.4 Block Diagram of the Control Unit

20

# CONTROL TRANSFER

- CPU fetches and executes each instruction of the program
- Successively goes through fetch, execute and interrupt cycles
- Two flip-flops marked F and E identify each of these 3 cycles
- Two general operations
  - Instruction control transfer
  - Program control transfer

CPU Cycles

| F | E | Comments |
|---|---|---|
| 0 | 0 | Not used |
| 1 | 0 | Fetch Cycle |
| 0 | 1 | Execute Cycle |
| 1 | 1 | Interrupt Cycle |

21

# Instruction Control Transfer

- PC holds the address of the instruction to be executed
- During fetch cycle PC is incremented to hold the address of the next instruction
- Assuming each instruction has a length of one word, then PC incremented by one, that is, `PC <= PC + 1;`
- Instruction length needs to be specified in some bits of the operation code field of instruction
- Transfer of control to non sequential instruction occurs during execution cycle of conditions branch with specified condition satisfied or by an unconditional branch instruction

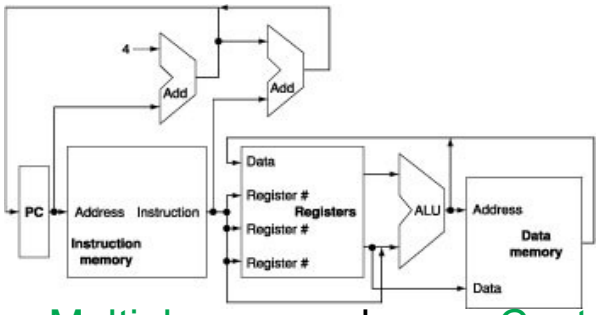
Example

- Jump X (Unconditional branch to X) or
- Jump ZX (Branch to X if the result of last arithmetic operation is zero)
- X stored in the operand field of instruction register

22

# Instruction Control Transfer

- For most instructions: fetch instruction, fetch operands, execute, store.
- An abstract view of the implementation of the MIPS subset showing the major functional units and the major connections between them




- Missing Multiplexers, and some Control lines for read and write.

23

# PROGRAM CONTROL TRANSFER

- While program P1 is running ,CPU is under control of P1
- Transfer of program control may occur in 2 situations
  1. There is an instruction in P1 which calls a subroutine denoted as program P2, For example,
     JMPX, where X denotes starting address of P2
  2. While an instruction in P1 is getting fetched and executed ,an interrupt flag gets set, where X denotes starting address of ISR(P2). After execution of ISR restore the execution which is saved in a temporary register.
- Prior to transferring control to P2 ,the CPU status of P1 is stored in stack (push and pop operation)

24

# FETCH CYCLE

- The instruction whose address is determined by the PC is obtained from the memory
- Loaded into the IR.
- The PC is then incremented to point to the next instruction and switch over to execution cycle
- CPU enters fetch cycle if F=1 and E=0

$C_{01}$: MAR ⇐ PC
$C_{02}$ $C_{03}$:MDR ⇐ M(MAR);PC ⇐ PC+1
$C_{04}$:IR ⇐ MDR
$C_{05}$:F ⇐ 1; E ⇐ 0

Sequence of micro operations of fetch cycle

25

# INSTRUCTION INTERPRETATION AND EXECUTION

- Instruction loaded into Instruction Register (IR)
- Processor interprets/decodes instruction and performs required actions
  - Different for each instruction
- e.g. ADD X, R1 - add the contents of location X to Register R1 , result in X
- $C_{61}$: MAR <- ($IR_{address}$)
- $C_{62}$: MDR <- (memory)
- $C_{63}$: X <- R1 + (MDR)
- $C_{64}$: F ← 1; E ← 0
- Note no overlap of micro-operations

26

# Recap of basic Concepts of Control Unit

- To execute instructions, the processor must generate the necessary control signals in proper sequence.
- Hardwired control:
  - Control unit is designed as a finite state machine.
  - Inflexible but fast.
  - Appropriate for simpler machines (e.g. RISC machines)
- Microprogrammed control:
  - Control path is designed hierarchically using principles identical to the CPU design.
  - Flexible, but slow.
  - Appropriate for complex machines (e.g. CISC machines)

27

# Hardwired control

| Step | Action |
|---|---|
| 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 |

- Each step in this sequence is completed in one clock cycle.
- A counter may be used to keep track of the control steps.
- Each state or count, of this counter, corresponds to one control step.

28

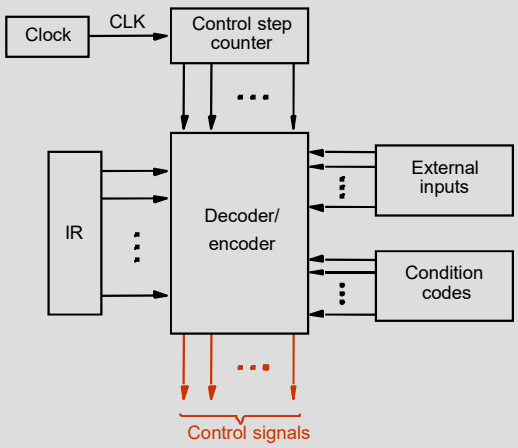
# Hardwired control (contd..)

- Required control signals are determined by the following information:
  - Contents of the control step counter.
    - Determines which step in the sequence.
  - Contents of the instruction register.
    - Determines the actual instruction
  - Contents of the condition code flags.
    - Used for example in a BRANCH instruction.
  - External input signals such as Memory Flow control (MFC).

29

# Hardwired control (contd..)

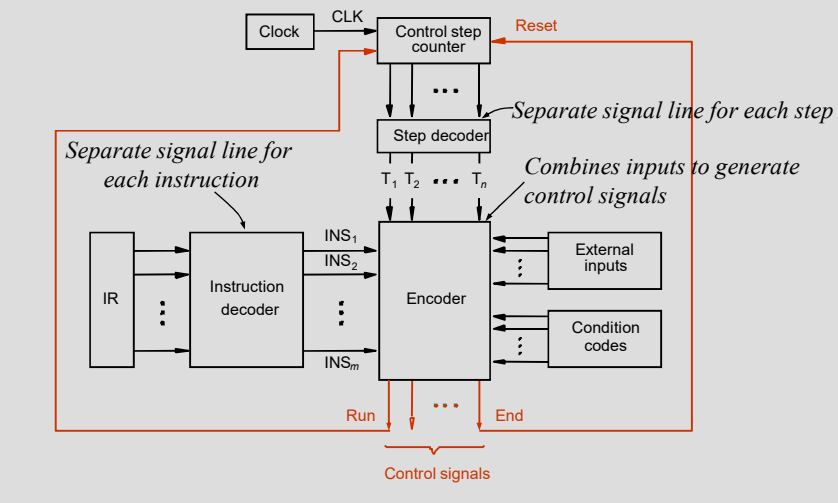
**Control unit organization**




*•Control unit consists of a decoder/encoder block to accept the following inputs:*
*- Control step counter.*
*- Instruction Register. IR*
*- Condition codes*
*- External inputs.*
*•Generates control signals.*

30

# Hardwired control (contd..)

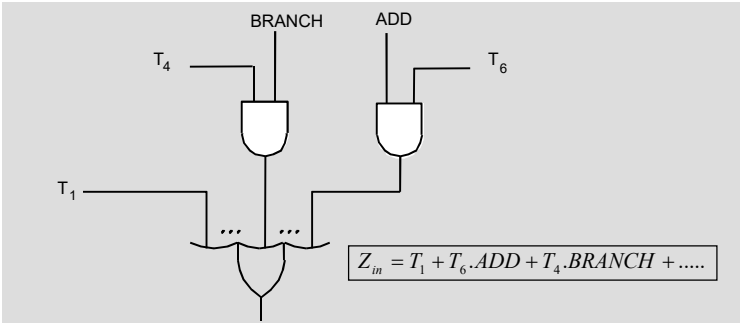



31

# Hardwired control (contd..)

Control signals such as $Z_{in}$, $PC_{out}$, $ADD$ are generated by encoder block

Suppose if $Z_{in}$ is asserted:
- During $T_1$ for all instructions.
- During $T_6$ for $ADD$ instruction.
- During $T_4$ for unconditional $BRANCH$ instruction.




32

# Hardwired control (contd..)

- Control hardware can be viewed as a state machine:
  - Changes state every clock cycle depending on the contents of the instruction register, condition codes, and external inputs.
- Outputs of the state machine are control signals:
- Sequence of control signals generated by the machine is determined by wiring of logic elements, hence the name "hardwired control".
- Speed of operation is one of the advantages of hardwired control is its speed of operation.
- Disadvantages include:
  - Little flexibility

33

# Microprogrammed control

- Hardwired control generates control signals using:
  - A control step counter.
  - Decoder/encode circuit.
- Microprogrammed control:
  - Control signals are generated by a ***program*** similar to machine language programs.

34

# Microprogrammed control (contd..)

Control Word (**CW**) is a word whose individual **bits** represent various control signals.

| Step | Action |
|---|---|
| 1 | $PC_{out}$ , $MAR_{in}$ , Read, Select4, Add, $Z_{in}$ |
| 2 | $Z_{out}$ , $PC_{in}$ , $Y_{in}$ , WMF C |
| 3 | $MDR_{out}$ , $IR_{in}$ |
| 4 | $R3_{out}$ , $MAR_{in}$ , Read |
| 5 | $R1_{out}$ , $Y_{in}$ , WMF C |
| 6 | $MDR_{out}$ , SelectY, Add, $Z_{in}$ |
| 7 | $Z_{out}$ , $R1_{in}$ , End |

At every step, some control signals are asserted (=1) and all others are 0.

***Control Signals:***
***$PC_{out}$***
***$PC_{in}$***
***$MAR_{in}$***
***Read***
***$MDR_{out}$***
***$IR_{in}$***
***$Y_{in}$***
***SelectY***
***Select4***
***Add***
***$Z_{in}$***
***$Z_{out}$***
***$R1_{out}$***
***$R1_{in}$***
***$R3_{out}$***
***WMFC***
***End .....***

35

# Microprogrammed control (contd..)

| Micro - instruction | .. | $PC_{in}$ | $PC_{out}$ | $MAR_{in}$ | Read | $MDR_{out}$ | $IR_{in}$ | $Y_{in}$ | Select | Add | $Z_{in}$ | $Z_{out}$ | $R1_{out}$ | $R1_{in}$ | $R3_{out}$ | WMFC | End | .. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 2 | | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | |
| 3 | | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 4 | | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| 5 | | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | |
| 6 | | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 7 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | |

- *At every **step**, a Control Word needs to be generated.*
- *Every **instruction** will need a sequence of CWs for its execution.*
- *Sequence of **CWs** for an **instruction** is the **microroutine** for the instruction.*
- *Each **CW** in this microroutine is referred to as a **microinstruction**.*

*(SelectY is represented by Select=0, & Select4 by Select=1)*

36

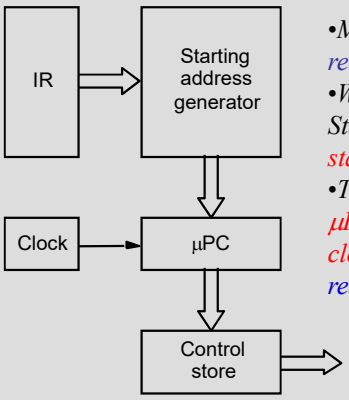
# Microprogrammed control (contd..)

- Every instruction will have its own microroutine which is made up of microinstructions.
- Microroutines for **all** instructions in the instruction set of a computer are stored in a special memory called Control Store.
- Recall that the Control Unit generates the control signals:
  - Sequentially reading the CWs of the corresponding microroutine from the control store.

37

# Microprogrammed control (contd..)

**Basic organization of a microprogrammed control unit.**




*•Microprogram counter (μPC) is used to read CWs from control store sequentially.*
*•When a new instruction is loaded into IR, Starting address generator generates the starting address of the microroutine.*
*•This address is loaded into the μPC.*
*μPC is automatically incremented by the clock, so successive microinstructions are read from the control store.*

38

# Microprogrammed control (contd..)

- Basic organization of the microprogrammed control unit cannot check the status of condition codes or external inputs to determine what should be the next microinstruction.
- Recall that in the hardwired control, this was handled by an appropriate logic function.
- How to handle this in microprogrammed control:
  - Use conditional branch microinstructions.
  - These microinstructions, in addition to the branch address also specify which of the external inputs, condition codes or possibly registers should be checked as a condition for branching.

39

# Microprogrammed control (contd..)

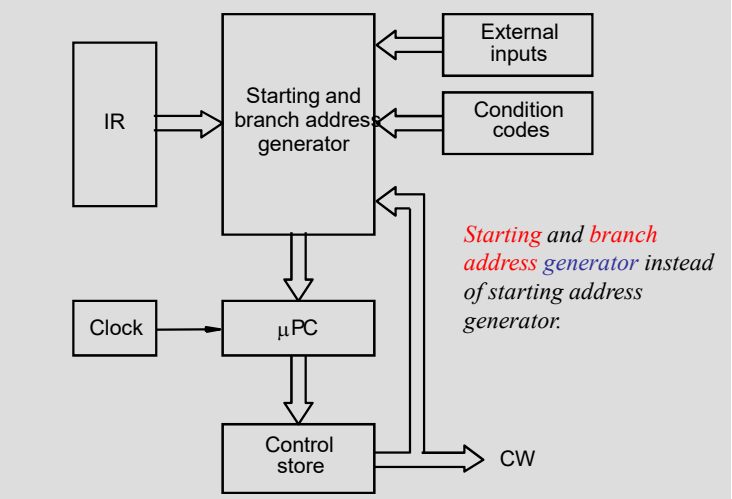
| Address | Microinstruction | |
|---|---|---|
| 0 | $PC_{out}$ , $MAR_{in}$ , Read, Select4, Add, $Z_{in}$ | Fetch *BRANCH<0* instruction, microroutine is at address 25. |
| 1 | $Z_{out}$ , $PC_{in}$ , $Y_{in}$ , WMFC | |
| 2 | $MDR_{out}$ , $IR_{in}$ | |
| 3 | Branch to starting address of appropriate microroutine | Branch to address 25. |
| ..... | | |
| 25 | If N=0, then branch to microinstruction 0 | Test the *N* bit of the condition codes<br>If 0, go to 0 and get new instr.<br>Else execute microinstruction located at 26 and put the branch target address into Register *Z*. *(*Microinstruction at location 27*)*. |
| 26 | Offset-field-of-$IR_{out}$ , SelectY, Add, $Z_{in}$ | |
| 27 | $Z_{out}$ , $PC_{in}$ , End | |

Address 25 is the output of starting address generator and is loaded into the microprogram counter (*µPC*).

40

# Microprogrammed control (contd..)

Control Unit with Conditional Branching in the Microprogram




41

# Microprogrammed control (contd..)

- Starting and branch address generator accepts as inputs:
  - Contents of the Instruction Register **IR** (as before).
  - External inputs
  - Condition codes
- Generates a new address and loads it into microprogram counter ($\mu PC$) when a microinstruction instructs it do so.
- $\mu PC$ is incremented every time a microinstruction is fetched except:
  - New instruction is loaded into IR, $\mu PC$ is loaded with the starting address of the microroutine for that instruction.
  - Branch instruction is encountered and branch condition is satisfied, $\mu PC$ is loaded with the branch address.
  - End instruction is encountered, $\mu PC$ is loaded with the address of the first CW in the microroutine for the instruction fetch cycle.

42

# Microprogrammed control (contd..)

## Microinstruction format

- Simple approach is to allocate one bit for each control signal
  - Results in long microinstructions, since the number of control signals is usually very large.
  - Few bits are set to 1 in any microinstruction, resulting in a poor use of bit space.

- Reduce the length of the microinstruction by taking advantage of the fact that most signals are not needed simultaneously, and many signals are mutually exclusive.

- For example:
  - Only one ALU function is active at a time.
  - Source for a data transfer must be unique.
  - *Read* and *Write* memory signals cannot be active simultaneously.

43

# Microprogrammed control (contd..)

## Microinstruction format

- Group mutually exclusive signals in the same group.
- At most one microperation can be specified per group.
- Use binary coding scheme to represent signals within a group.

Examples:
- If ALU has 16 operations, then 4 bits can be sufficient.
- Group register output signals into the same group, since only one of these signals will be active at any given time (Why?)

If the CPU has 4 general purpose registers, then $PC_{out}$, $MDR_{out}$, $Z_{out}$, $Offset_{out}$, $R0_{out}$, $R1_{out}$, $R2_{out}$, $R3_{out}$ and $Temp_{out}$ can be placed in a single group, and 4 bits will be needed to represent these.

44

# Microprogrammed control (contd..)

Microinstruction

| F1 | F2 | F3 | F4 | F5 |
|---|---|---|---|---|

| F1 (4 bits) | F2 (3 bits) | F3 (3 bits) | F4 (4 bits) | F5 (2 bits) |
|---|---|---|---|---|
| 0000: No transfer<br>0001: $PC_{out}$<br>0010: $MDR_{out}$<br>0011: $Z_{out}$<br>0100: $R0_{out}$<br>0101: $R1_{out}$<br>0110: $R2_{out}$<br>0111: $R3_{out}$<br>1010: $TEMP_{out}$<br>1011: $Offset_{out}$ | 000: No transfer<br>001: $PC_{in}$<br>010: $IR_{in}$<br>011: $Z_{in}$<br>100: $R0_{in}$<br>101: $R1_{in}$<br>110: $R2_{in}$<br>111: $R3_{in}$ | 000: No transfer<br>001: $MAR_{in}$<br>010: $MDR_{in}$<br>011: $TEMP_{in}$<br>100: $Y_{in}$ | 0000: Add<br>0001: Sub<br>⋮<br>1111: XOR<br>16 ALU functions | 00: No action<br>01: Read<br>10: Write |

| F6 | F7 | F8 | ... |
|---|---|---|---|

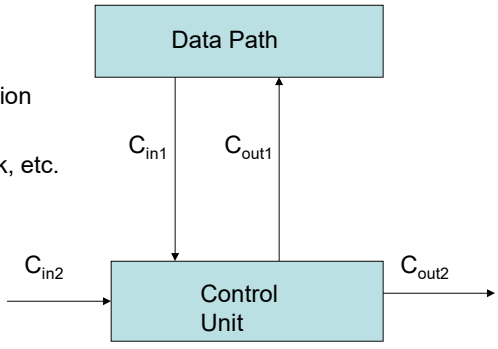
| F6 (1 bit) | F7 (1 bit) | F8 (1 bit) |
|---|---|---|
| 0: SelectY<br>1: Select4 | 0: No action<br>1: WMFC | 0: Continue<br>1: End |

- Each group occupies a large enough field to represent all the signals.
- Most fields must include one inactive code, which specifies no action.
- All fields do not have to include inactive code.

45

# Input-Output behavior of a control Unit

- Set of signals $C_{in1}$ input to the Control Unit from the datapath includes those from the operation code of IR, various flag registers of ALU, interrupt , etc. dictate the flow of control for the algorithm implemented on the datapath for an operation code
- Other input signals $C_{in2}$ are those control which control the operation of the CPU itself, such as start, stop, clock, etc.
- $C_{out1}$ fed at appropriate point on the datapath to sequence and control the micro operations associated with operation code



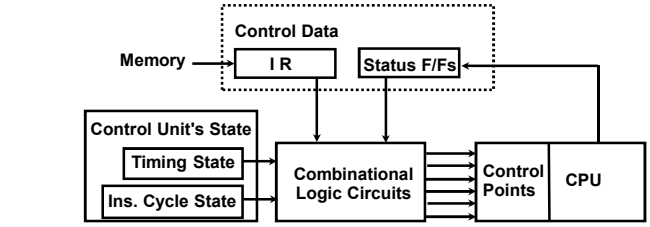

- Other control signal $C_{out2}$ cover signals such as busy, operation complete, etc., transmitted to other units.

46

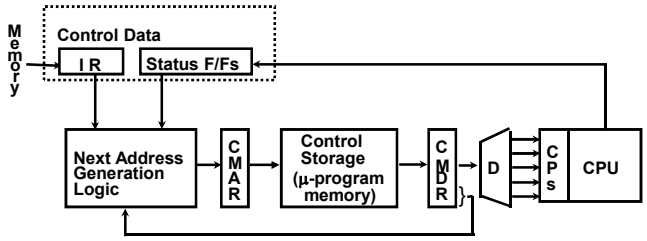
# COMPARISON OF CONTROL UNIT IMPLEMENTATIONS

**Control Unit Implementation**

**Sequencial Logic Circuits (Hard-wired)**




**Microprogram**




47

# HARDWIRED CONTROL

- The control unit is designed as a sequential logic circuit which generates the specific sequence of control signals as its primary output
- The sequence of 4 control signals $C_0$, $C_1$, $C_2$, $C_3$ can be developed by using a 2-bit sequence counter

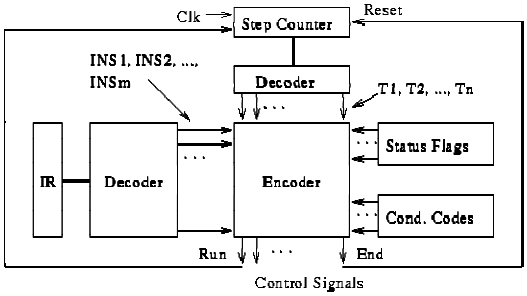
Sequence of micro operations of fetch cycle

$C_0=C_{01}$: MAR ← PC
$C_1=C_{02}\ C_{03}$: MDR ← M(MAR);PC ← PC+1
$C_2=C_{04}$: IR ← MDR
$C_3=C_{05}$: F ← 0; E ← 1

- Its output is decoded to derive the desired control signals in the given sequence.

48

# Control Unit Block Diagram




49

# Detailed control sequencing

- ADD (R3), R1 **High-level execution sequence:**
- Fetch instruction.
- Fetch memory operand.
- Perform addition.
- Store result in memory R3.

**Detailed control sequencing**:

1: PCout, MARin, Read, Clear Y, Set carry-in to ALU, Add, Zin
2: Zout, PCin, WMFC
3: MDRout, IRin
4: R3out, MARin, Read
5: R1out, Yin, WMFC
6: MDRout, Add, ZinZout, R1in, END

50

## Control Unit specification for the Fetch Sequence

- Address of next instruction is in PC (s01)
- Content of PC loaded into MAR (c01) ,s02
- Address (MAR) is placed on address bus
- Control unit issues READ command
- Result (data from memory) appears on data bus
- Data from data bus copied into MDR (co2),s03
- PC incremented by 1 (in parallel with data fetch from memory) (c03),s03
- Data (instruction) moved from MDR to IR (c04), S04
- MDR is now free for further data fetches (F← 0 ; E←1) (c05) , s05

51

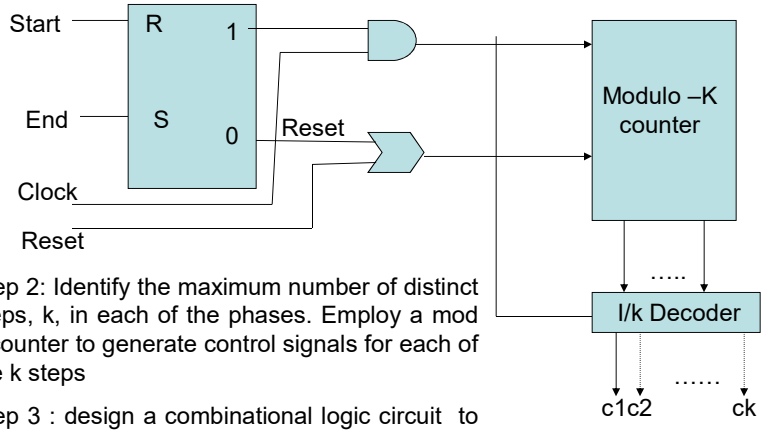
## Methods for systematic design of Hardwired Control Logic

- Sequence counter method: To design control unit of moderate complexity.
- Delay element method: Depends on the use of clocked delay elements for generating the sequence of control signals.
- State table method: Employs the algorithmic approach to sequential circuit design using classical state table method.

52

# Sequence counter method

- Step 1: Identify the distinct phases in the flowchart. Employ log p number of flip flops to handle p number distinct phases



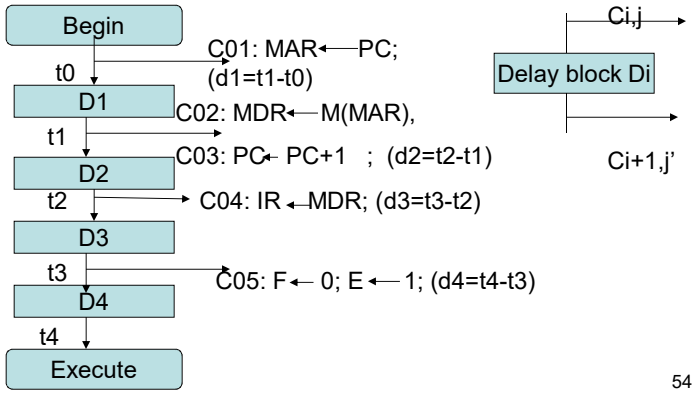

Step 2: Identify the maximum number of distinct steps, k, in each of the phases. Employ a mod k counter to generate control signals for each of the k steps

Step 3 : design a combinational logic circuit to generate the sequence of control signals to control the micro operations of each phase

53

# Delay element method

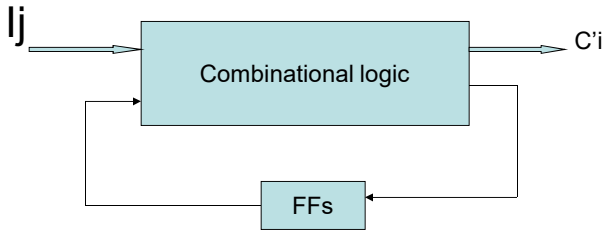
## Control unit based on delay element method for the fetch cycle




54

# State table method

- The state Si (i=1,2…) has been marked above each block of the flowchart.
- This the state of the control unit which generates the control signals to control the micro operations in the data path




55

# Steps to design the Control structure realizing flowchart

- State Assignment: States are assigned as s1,s2,s3…;each such assignment specifies a particular state of the control unit at the specific time step. State table derived from state assignment
- State Minimization: A set of states {Sa,Sb,…Sc} can be merged to a single state S' if Si & Sj is pair wise compatible
- State Encoding: State variables are defined and states are encoded in terms of state variables

56

## Hard-wired Control Unit-advantages

1. Minimizes the average number of clock cycles needed per instruction
2. occupies a relatively small area (typically 10%) of the CPU chip area
3. High efficiency in terms of operation speed
4. is to minimize cost of the circuit

57

## Problems With Hard Wired Designs

- Complex sequencing & micro-operation logic
- Difficult to design and test
- Inflexible design
- Large design turn around time for complex design
- Difficult to add new instructions

58

# Tasks Done By Micro programmed Control Unit

- Microinstruction sequencing
- Microinstruction execution
- Must consider both together

59

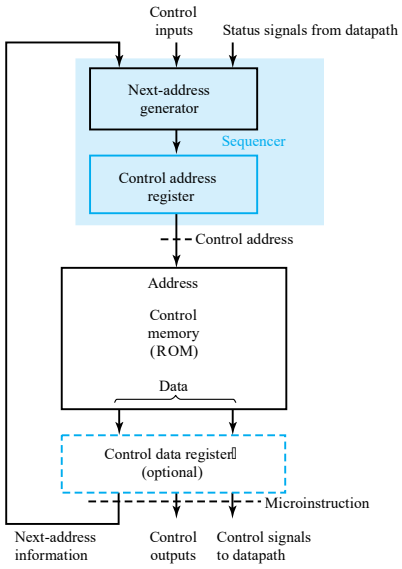
# Microprogrammed Control

The concept of micro programmed control, employ the following steps:

1. Any instruction to be executed by a CPU can be broken down into a set of sequential micro operations – each specifying a RTL operation on the data path. The set of micro operations to be executed on the RTL components at any time step is referred as *microinstructions*.
2. The sequence of control signals necessary to execute the sequential microinstructions stored in ROM called *control ROM*
3. To implement an instruction on the data path, the control signals stored in the ROM can be accessed
4. The control signals read from the ROM are used to control the micro operations associated with a microinstruction to be executed at any time step
5. The address of the next micro instruction is generated
6. The steps 3,4 and 5 are repeated till the set of sequential microinstructions associated with the instruction is executed

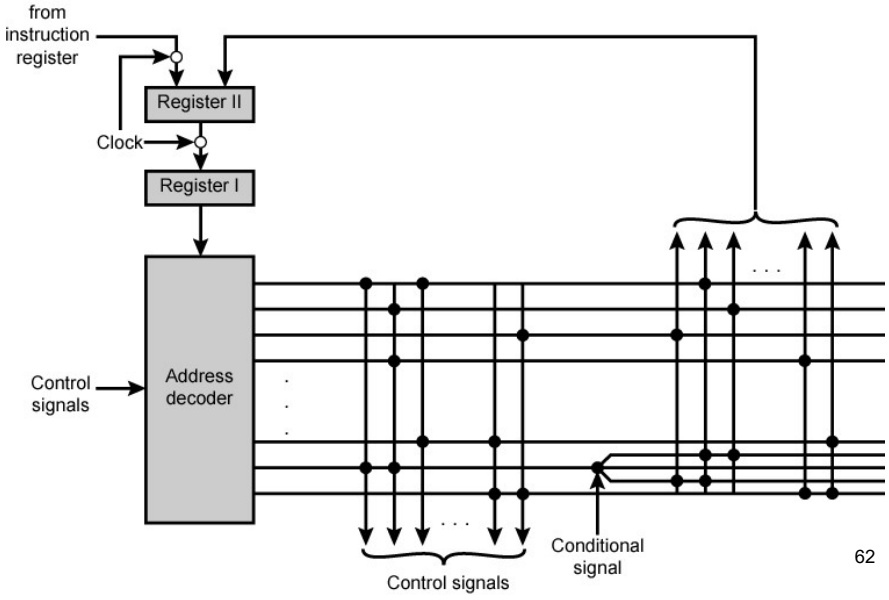
60

# Microprogrammed Control **(continued)**




61

# Wilkes's Microprogrammed Control Unit




62

## MICROINSTRUCTION FORMAT

**Information in a Microinstruction**
- **Control Information**
- **Sequencing Information**
- **Constant**
  **Information which is useful when feeding into the system**

**These information needs to be organized in some way for**
- **Efficient use of the microinstruction bits**
- **Fast decoding**

**Field Encoding**

- **Encoding the microinstruction bits**
- **Encoding slows down the execution speed due to the decoding delay**
- **Encoding also reduces the flexibility due to the decoding hardware**

63

## Encoding of control signals

- Different formats of microinstruction depending on the encoding of control signals.
- The signals are divided into multiple control fields in a ROM word.
  1. Horizontal/vertical formats
  2. Functional encoding
  3. Resource encoding
  4. Direct versus indirect encoding

64

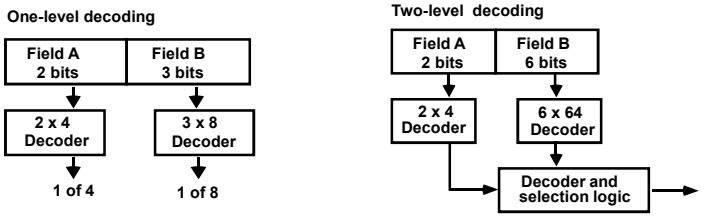
## HORIZONTAL AND VERTICAL MICROINSTRUCTION FORMAT

**Horizontal Microinstructions**

- **Each bit directly controls each micro-operation or each control point**
- ***Horizontal* implies a long microinstruction word**
- **Advantages: Can control a variety of components operating in parallel.**
  **--> Advantage of efficient hardware utilization**
- **Disadvantages: Control word bits are not fully utilized**
  **--> CS becomes large --> Costly**

**Vertical Microinstructions**

- **A microinstruction format that is not horizontal**
- ***Vertical* implies a short microinstruction word**
- **Encoded Microinstruction fields**
  **--> Needs decoding circuits for one or two levels of decoding**




65

# Functional Encoding

- The micro operations executed in a data path of a CPU may be categorized under different function names such as Shift function, Add function, Logical functions, Input-Output, etc.
- Multiple control bits associated to control a specified function

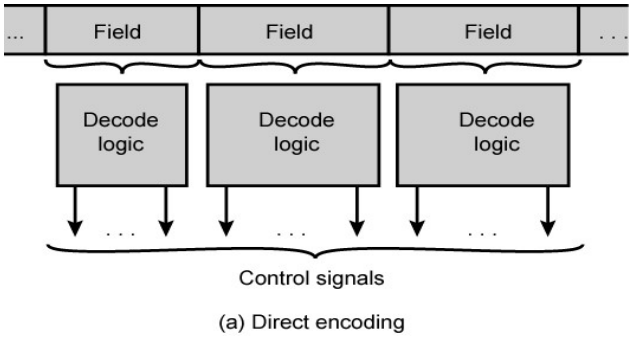
66

# Resource encoding

- The CPU data path consists of a set of interconnected RTL components.
- Each of these components or a subset of such components viewed as a resource.
- If control signals associated with such a resource are mutually exclusive, then they can be encoded in a single control field.

67

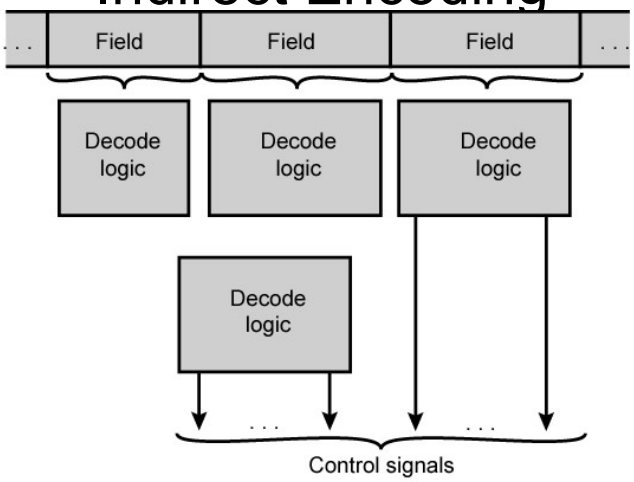
# Microinstruction Encoding Direct Encoding




(a) Direct encoding

68

# Microinstruction Encoding Indirect Encoding




(b) Indirect encoding

69

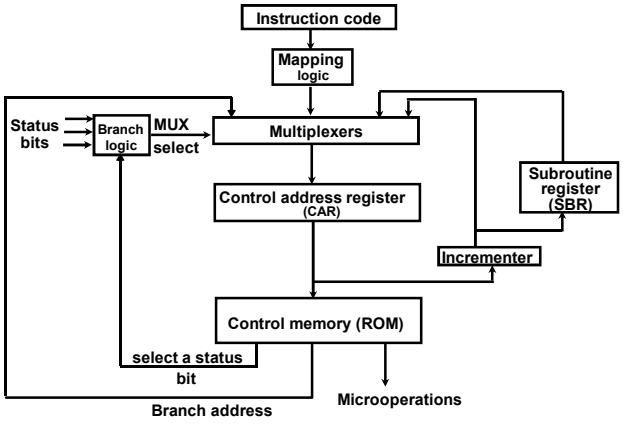
# Micro instruction execution

Two basic actions

1. Generating next microinstruction address and fetching micro program instruction from control memory by this address
2. Executing the micro operations controlled by different signals encoded in various control fields of the microinstruction
    1. Decoding and applying control signals on the CPU data path
    2. Executing the intended micro operations controlled by the signals
    3. Storing the output in the destination register specified in the micro operation on the CPU data path

70

# MICROINSTRUCTION SEQUENCING




**Sequencing Capabilities Required in a Control Storage**

- **Incrementing of the control address register**
- **Unconditional and conditional branches**
- **A mapping process from the bits of the machine instruction to an address for control memory**
- **A facility for subroutine call and return**

71

# Mono phase and Poly phase operation

- In a micro operation cycle, a micro program control unit implements the sequence of actions under 4 different phases.
- If the micro instruction cycle is controlled by a single clock pulse which synchronizes all the control signals ,then the mode is termed as *mono phase operation*
- If each of the phases is controlled by a different phase of a clock ,the mode is referred as poly phase operation

72

# NANOSTORAGE AND NANOINSTRUCTION

**The decoder circuits in a vertical micro program storage organization can be replaced by a ROM**

**=> Two levels of control storage**

**First level - *Control Storage***

**Second level - *Nano Storage***

**Two-level micro program**

**First level**

**-*Vertical* format Micro program**

**Second level**

**-*Horizontal* format Nano program**

**- Interprets the microinstruction fields, thus converts a vertical microinstruction format into a horizontal nano instruction format.**

**Usually, the micro program consists of a large number of short microinstructions, while the nano program contains fewer words with longer nano instructions.**

73

# TWO-LEVEL MICROPROGRAMMING - EXAMPLE

*** Microprogram: 2048 microinstructions of 200 bits each**

*** With 1-Level Control Storage: 2048 x 200 = 409,600 bits**

*** Assumption:**

**256 distinct microinstructions among 2048**

*** With 2-Level Control Storage:**

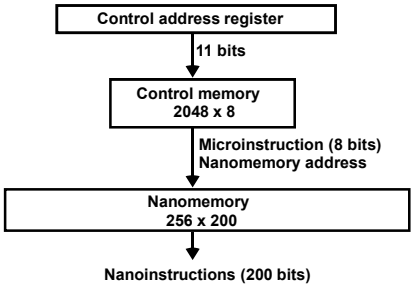
**Nano Storage: 256 x 200 bits to store 256 distinct nanoinstructions**

**Control storage: 2048 x 8 bits**

**To address 256 nano storage locations 8 bits are needed**

*** Total 1-Level control storage: 409,600 bits**

**Total 2-Level control storage: 67,584 bits (256 x 200 + 2048 x 8)**




74

## Micro-programmed Control Unit - Advantages

1. A micro-programmed control unit is flexible and allows designers to incorporate new and more powerful instructions as VLSI technology increases the available chip area for the CPU
2. allows any design errors discovered during the prototyping stage to be removed

75

## Microprogrammed Control Unit - Disadvantages

1. requires several clock cycles to execute each instruction, due to the access time of the microprogram memory
2. Occupies a large portion (typically 55%) of the CPU chip area

76