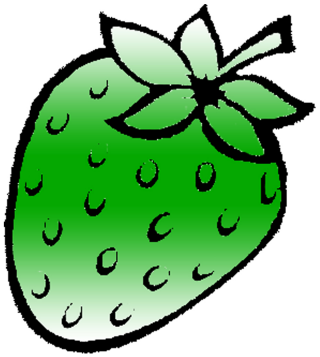
STRAWBERRY



/strawberrydevelopers

/strawberry_app

For more visit:

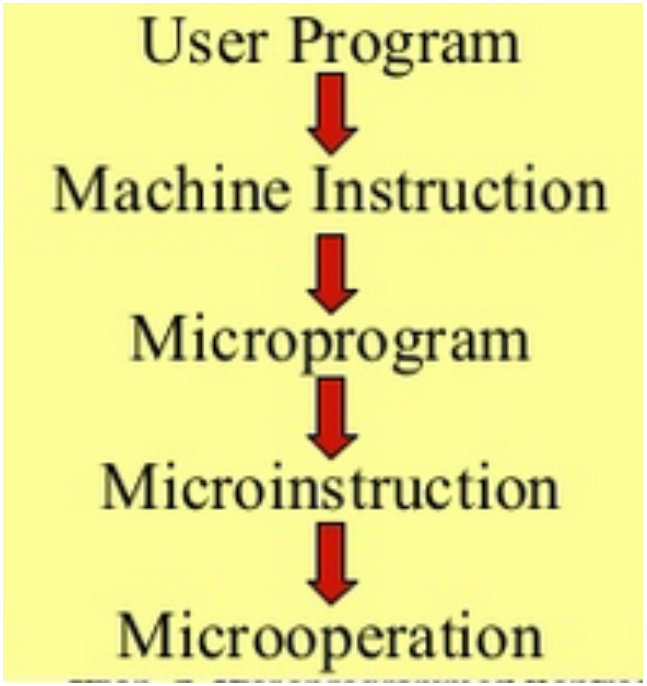
Strawberrydevelopers.weebly.com

UNIT VI: CONTROL UNIT

Agenda

- Control Unit micro operations
- Control Unit hardwired implementation
- Micro Programmed control
- Micro Instruction Format
- Applications of microprogramming

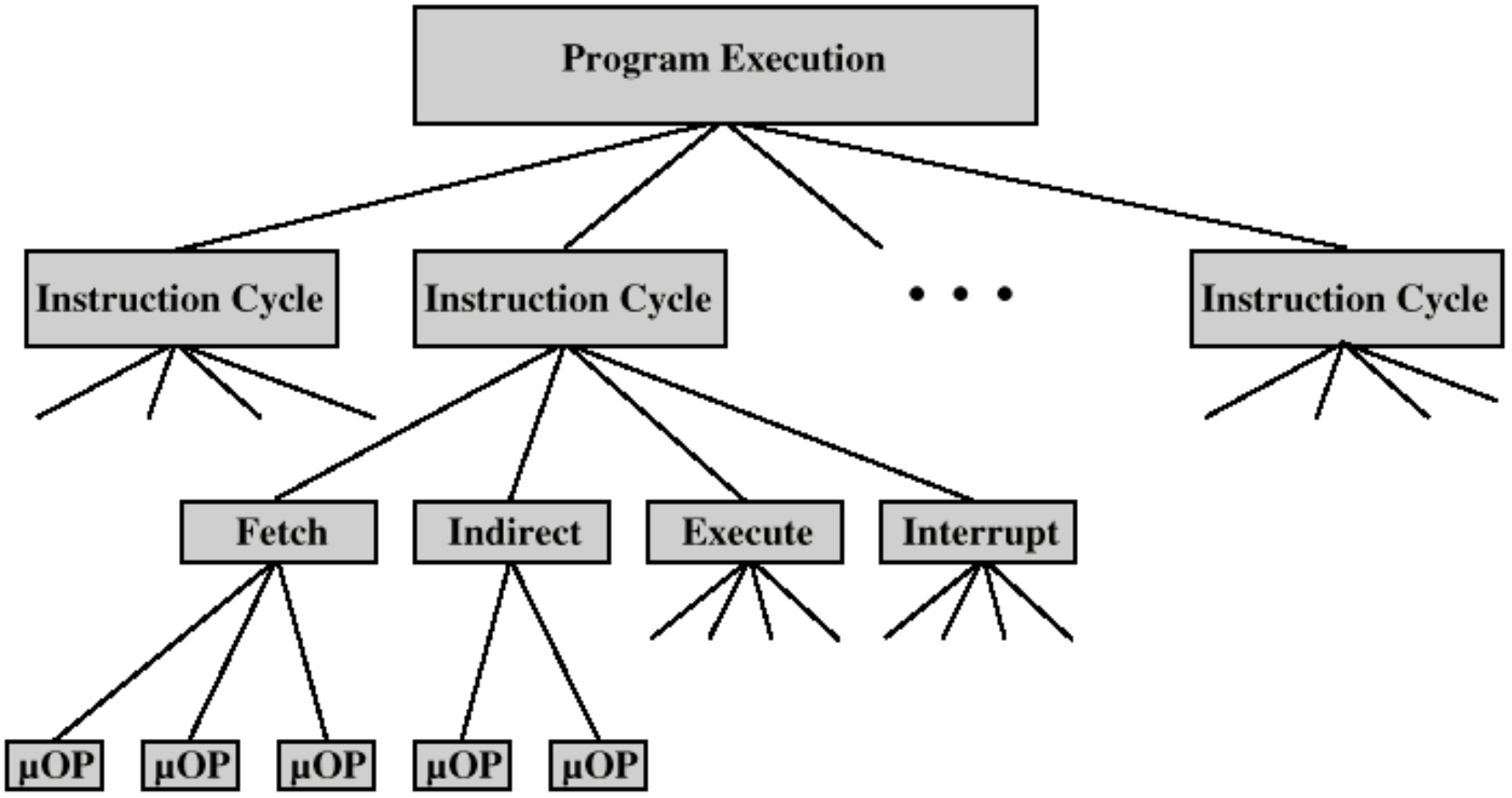
Basic Concept



Micro-Operations

- Instruction execution
  - execution of a sequence of steps, i.e., cycles
- Fetch, Indirect, Execute & Interrupt cycles
- Cycle - a sequence of micro-operations
- Micro-operations
  - data transfer between registers
  - transfer between a register & an external bus
  - ALU operation
- CU causes the processor to step through a series of micro-operations in the proper sequence
- CU generates the control signals that cause each micro-operation to be executed
- Micro-Operations are the atomic operations of the Processor

Constituent Elements of Program Execution



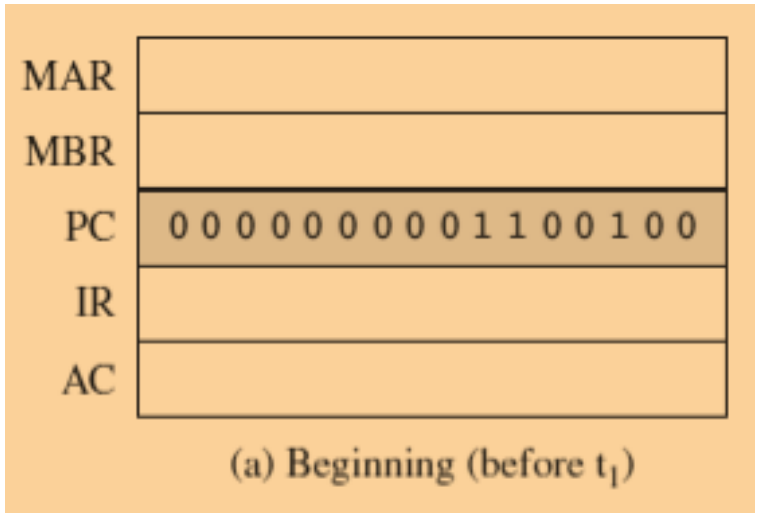
Registers

- Memory Address Register (MAR)
  - Connected to address lines of system bus
  - Specifies address for read or write operation
- Memory Buffer Register (MBR)
  - Connected to data lines of system bus
  - Holds data to write or last data read
- Program Counter (PC)
  - Holds address of next instruction to be fetched
- Instruction Register (IR)
  - Holds last instruction fetched

Fetch Sequence

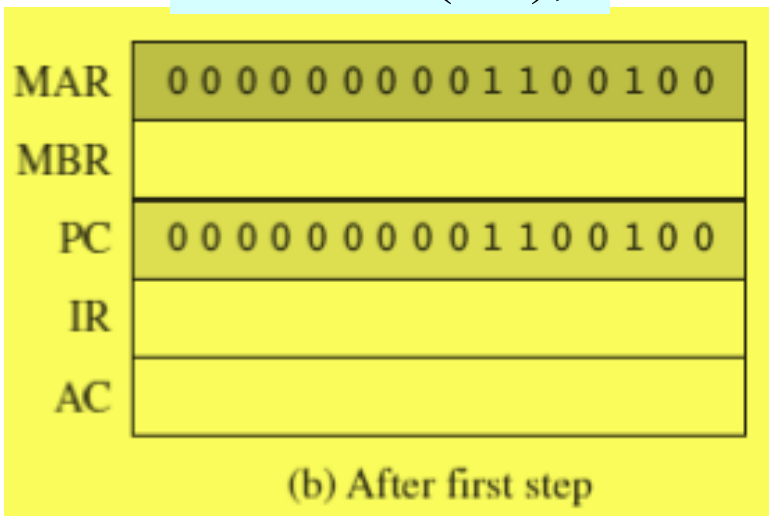
- Address of next instruction is in PC
- Address (MAR) is placed on address bus
- Control unit issues READ command
- Result (data from memory) appears on data bus
- Data from data bus copied into MBR
- PC incremented by 1 (in parallel with data fetch from memory) [**micro-code RISC, length == 1**]
- Data (instruction) moved from MBR to IR
- MBR is now free for further data fetches

Fetch Cycle: Sequence of events



(a) Beginning (before t_1)

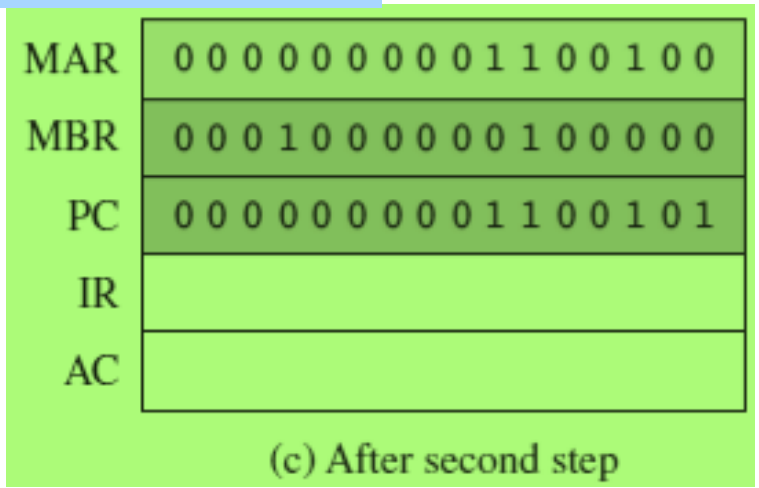
MAR←(PC);



(b) After first step

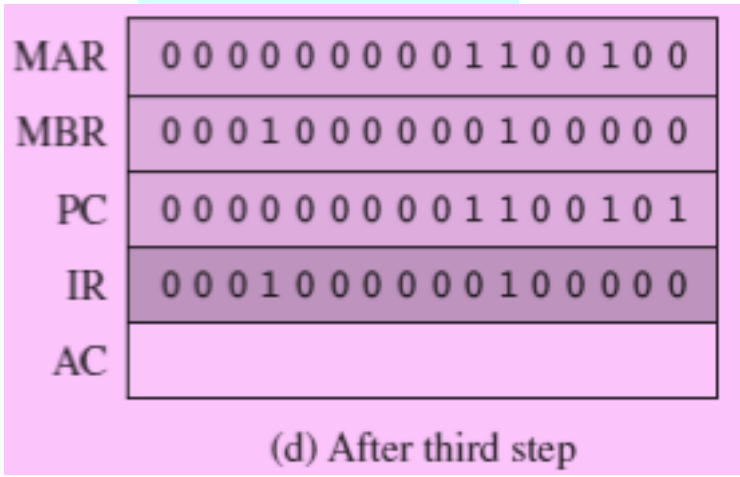
MBR <- (memory)

PC <- (PC) +1



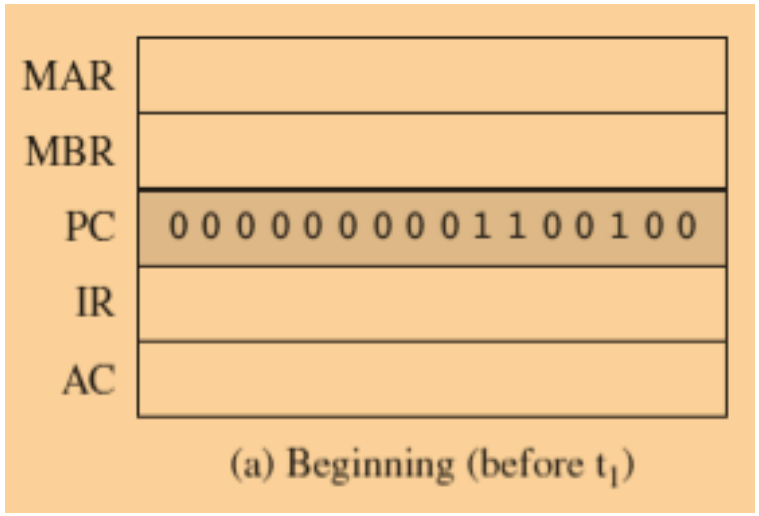
(c) After second step

IR <- (MBR)

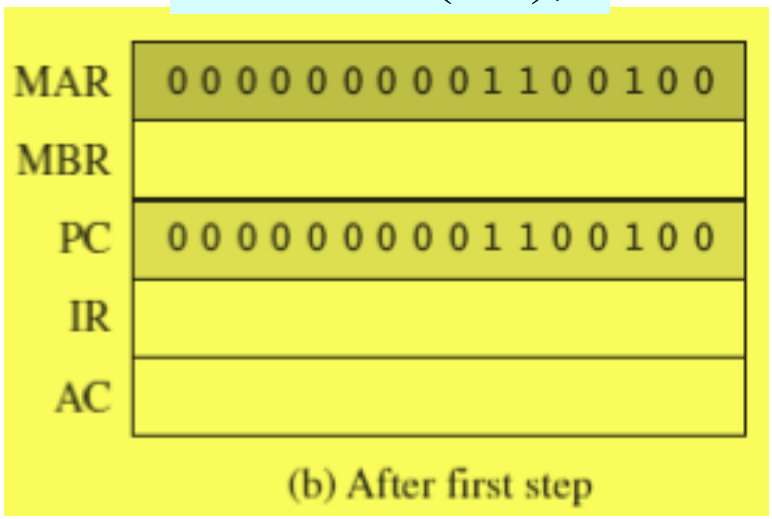


(d) After third step

Fetch Cycle: Sequence of events

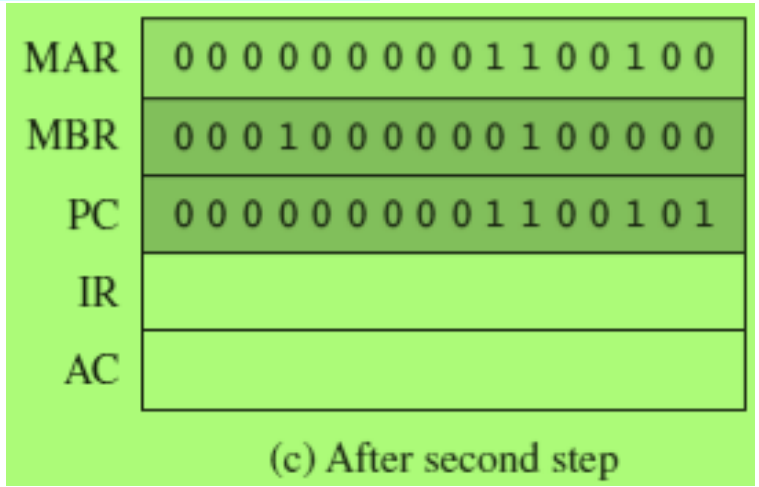


MAR←(PC);

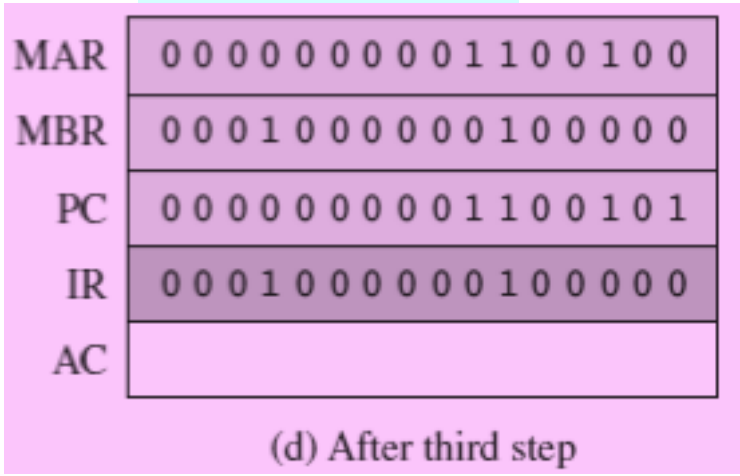


PC <- (PC) +1

MBR <- (memory)



IR <- (MBR)



Fetch Sequence (symbolic)

- t1: MAR <- (PC); CU issues READ command
- t2: MBR <- (memory) simultaneously
  PC <- (PC) +I
- t3: IR <- (MBR)
  where tx refers to the time unit/clock cycle

---------- or ----------

- t1: MAR <- (PC)
- t2: MBR <- (memory)
- t3: PC <- (PC) +1
  IR <- (MBR)

Rules for Clock Cycle Grouping

- Proper sequence must be followed
  - MAR <- (PC) must precede MBR <- (memory)
- Conflicts must be avoided
  - Must not read & write same register at same time
  - MBR <- (memory) & IR <- (MBR) must not be in same cycle
- Also: PC <- (PC) +1 involves addition
  - Must use ALU
  - Hence, may need additional micro-operations

Indirect Cycle

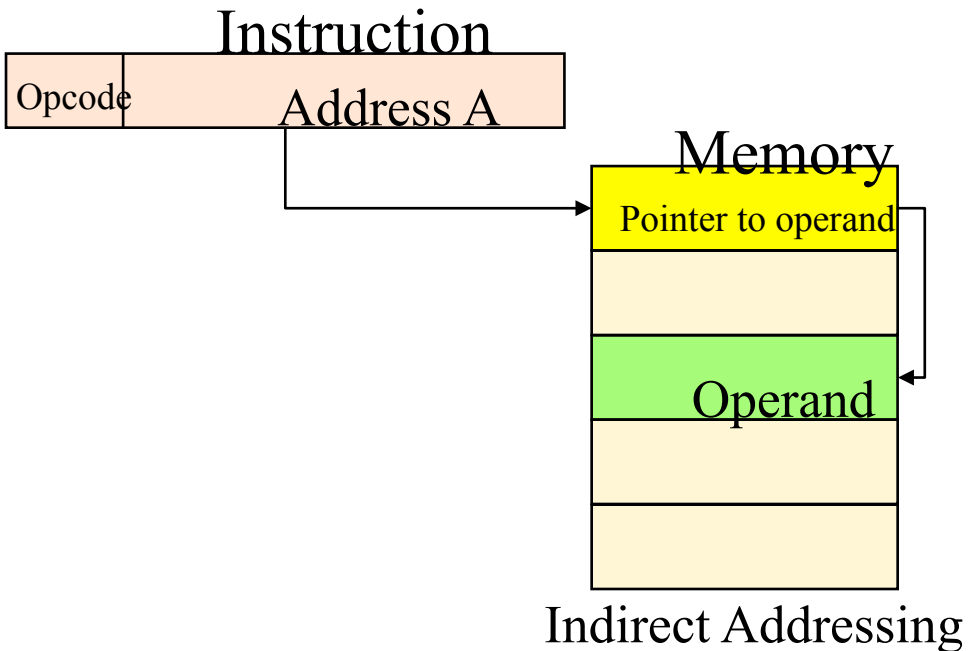
- Once an instruction is fetched, the next step is to fetch source operands.
- If the instruction specifies an indirect address, then an indirect cycle must precede the execute cycle

t1: MAR ← (IR(Address))

t2: MBR ← Memory

t3: IR(Address) ← (MBR(Address))

- MAR contains an indirect address
- MBR contains a direct address
- Result: IR is placed in same state as if direct addressing had been used originally



Indirect Addressing

Indirect Cycle

- The address field of the instruction is transferred to the MAR. This is then used to fetch the address of the operand.
- Finally, the address field of the IR is updated from the MBR, so that it now contains a direct rather than an indirect address.
- The IR is now in the same state as if indirect addressing had not been used, and it is ready for the execute cycle.

Interrupt Cycle

- At the completion of the execute cycle, a test is made to determine whether any enabled interrupts have occurred. If so, the interrupt cycle occurs. The nature of this cycle varies greatly from one machine to another.

t1: MBR ← (PC)

t2: MAR ← Save_Address for PC content
PC ← Routine_Address

t3: Memory ← MBR (actual saving of the PC contents)

- In the first step, the contents of the PC are transferred to the MBR, so that they can be saved for return from the interrupt.
- Then the MAR is loaded with the address at which the contents of the PC are to be saved, and the PC is loaded with the address of the start of the interrupt-processing routine. These two actions may each be a single micro-operation. the final step is to store the MBR, which contains the old value of the PC, into memory. The processor is now ready to begin the next instruction cycle.

Execute Cycle (ADD)

- **Different sequence of micro-operations for each instruction**
- **ADD R1, X** - add the contents of location X to Register 1 , place the result in R1
- t1: MAR ← (IR(address(X)))
- t2: MBR ← (Memory)
- t3: R1 ← (R1) + MBR(location X)

- **Note: there is no overlap of micro-operations**

Execute Cycle (ISZ)

- ISZ X - increment and skip if zero
  - t1: MAR ← (IR(address(x))
  - t2: MBR ← (memory)
  - t3: MBR ← (MBR) + 1
  - **t4: memory ← (MBR)**

    if (MBR) == 0 then PC ← (PC) + 1

    test & action operation is one micro op performed during time unit t4

Execute Cycle (BSA) – subroutine call instruction

- **BSA X - Branch and save address**
  - **Address of instruction following BSA is saved in X;**
    it will be used to return from the subroutine
  - **Execution continues from X+1**
  - t1: MAR ← (IR(**address(X)**)
  - MBR ← (PC) - **address of next instruction in the sequence**

    BSA X branches to X+1 after saving return address to location X
  - t2: PC ← (IR(**address(X)**)
  - memory ← (MBR) - **save PC contents in memory**
  - t3: PC ← (PC) + 1 - **start processing from X+1**

X : return address
X+1: start of subroutine
…
…
X+n: return from subroutine

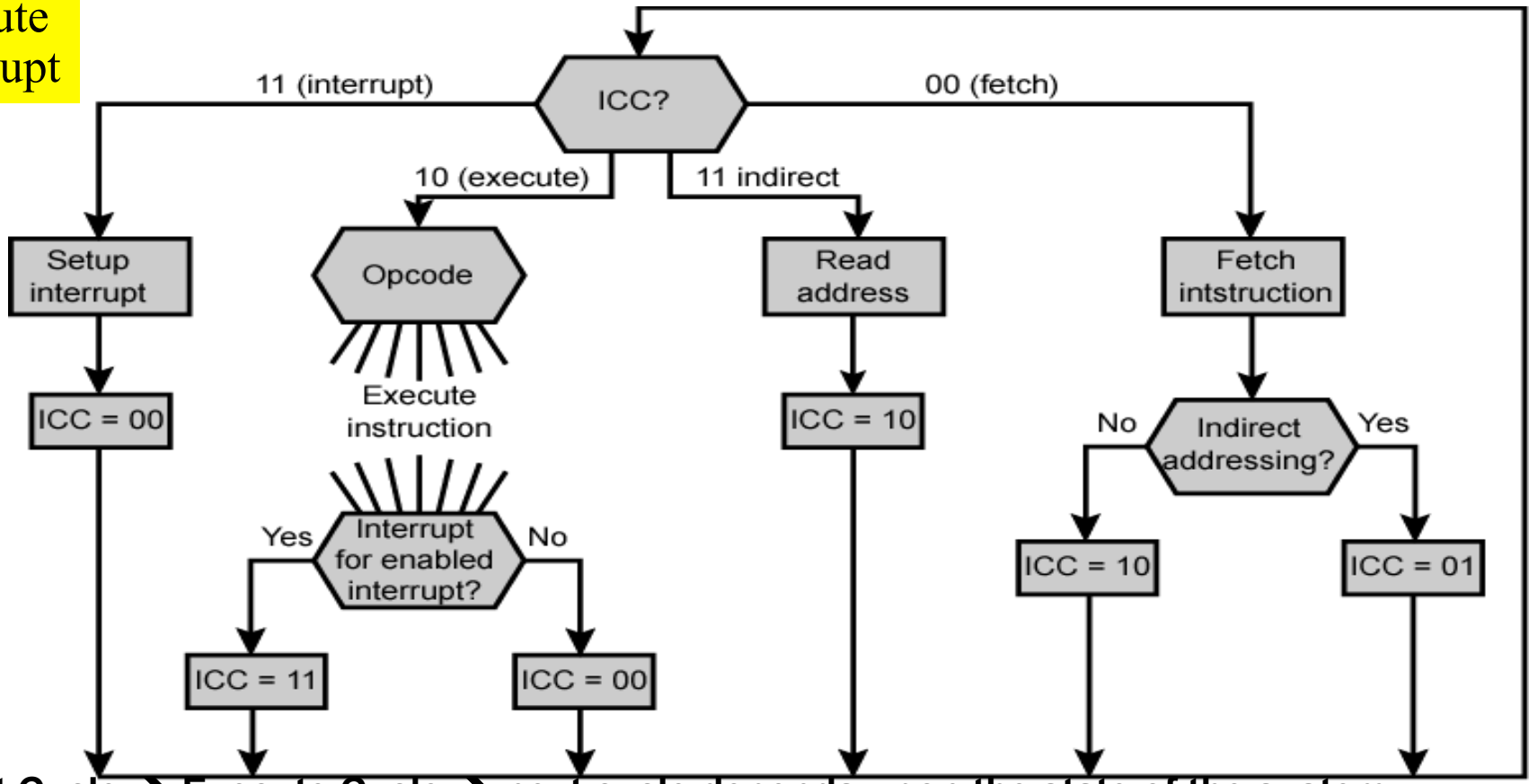
Instruction Cycle

- Each phase decomposed into sequence of elementary micro-operations
- E.g. fetch, indirect, and interrupt cycles
- Execute cycle
  - One sequence of micro-operations for each opcode
- Need to tie sequences together
- Assume new 2-bit register
  - Instruction cycle code (ICC) designates which part of cycle processor is in
    - 00: Fetch
    - 01: Indirect
    - 10: Execute
    - 11: Interrupt

Flowchart for Instruction Cycle (Code)

Operation of the Processor ←→ Performance of a Sequence of Micro-Operations

00: Fetch
01: Indirect
10: Execute
11: Interrupt

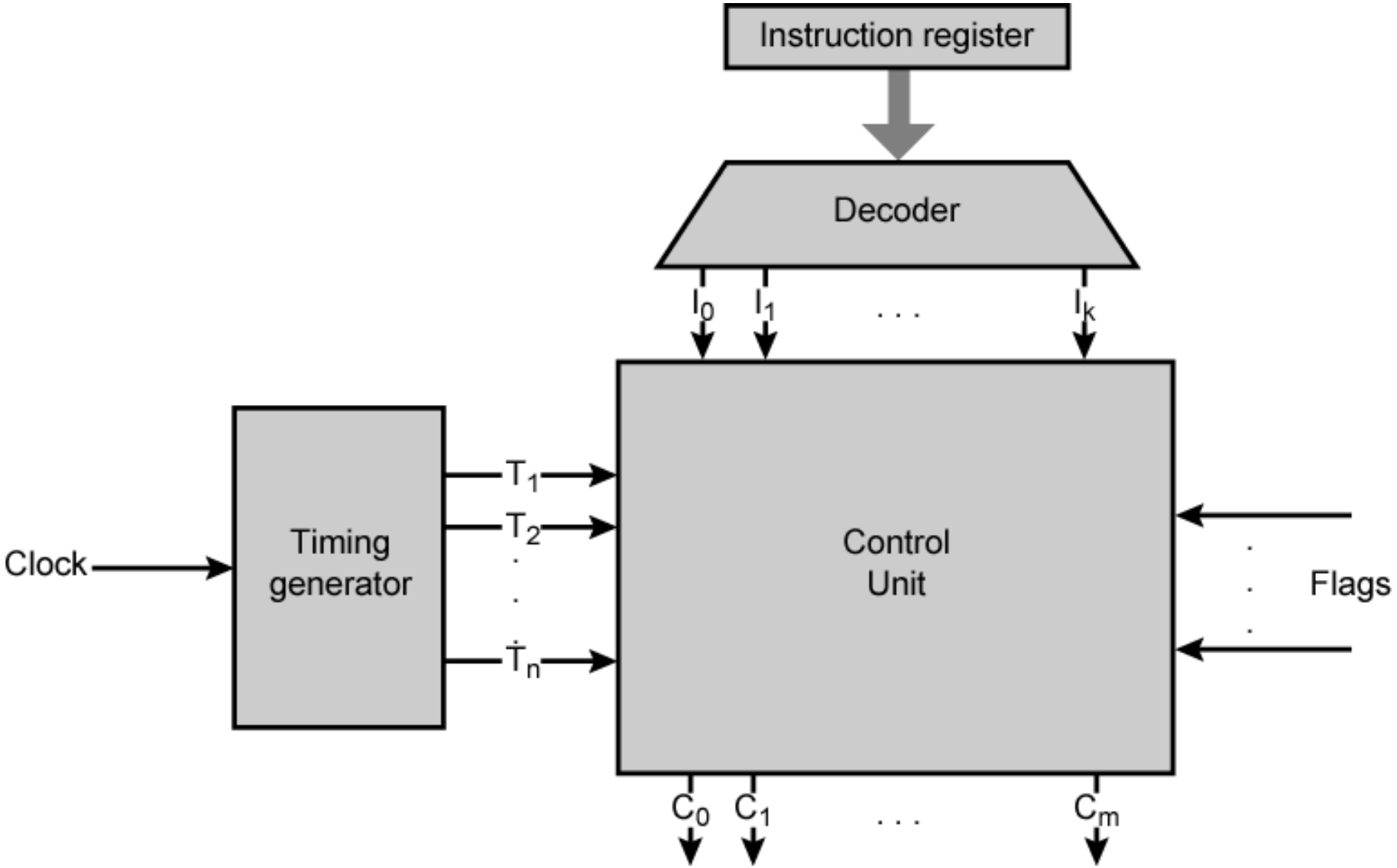


Indirect Cycle ➔ Execute Cycle ➔ next cycle depends upon the state of the system
Interrupt Cycle ➔ Fetch Cycle ➔ next cycle depends upon the state of the system

Hardwired Implementation

- Control unit inputs
- Flags and control bus
  - Each bit means something
- Instruction register
  - Op-code causes different control signals for each different instruction
  - Unique logic for each op-code
  - Decoder takes encoded input and produces single output
  - *n* binary inputs and 2^n outputs
- Clock
  - Repetitive sequence of pulses
  - Useful for measuring duration of micro-ops
  - Must be long enough to allow signal propagation
  - Different control signals at different times within instruction cycle
  - Need a counter with different control signals for t1, t2 etc.

Control Unit with Decoded Inputs



Problems With Hard Wired Designs

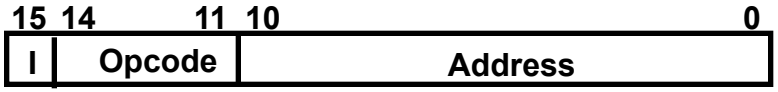
- Complex sequencing & micro-operation logic
- Difficult to design and test
- Inflexible design
- Difficult to add new instructions

Microinstruction

- An instruction that controls data flow and instruction-execution sequencing in a processor at a more fundamental level than machine instructions.
- A series of microinstructions is necessary to perform an individual machine instruction.

Microprogram Example

Computer instruction format

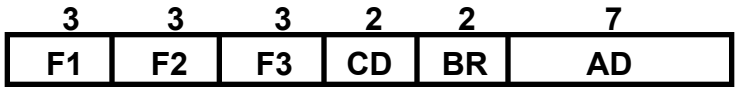


Four computer instructions

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

Micro-instruction Types

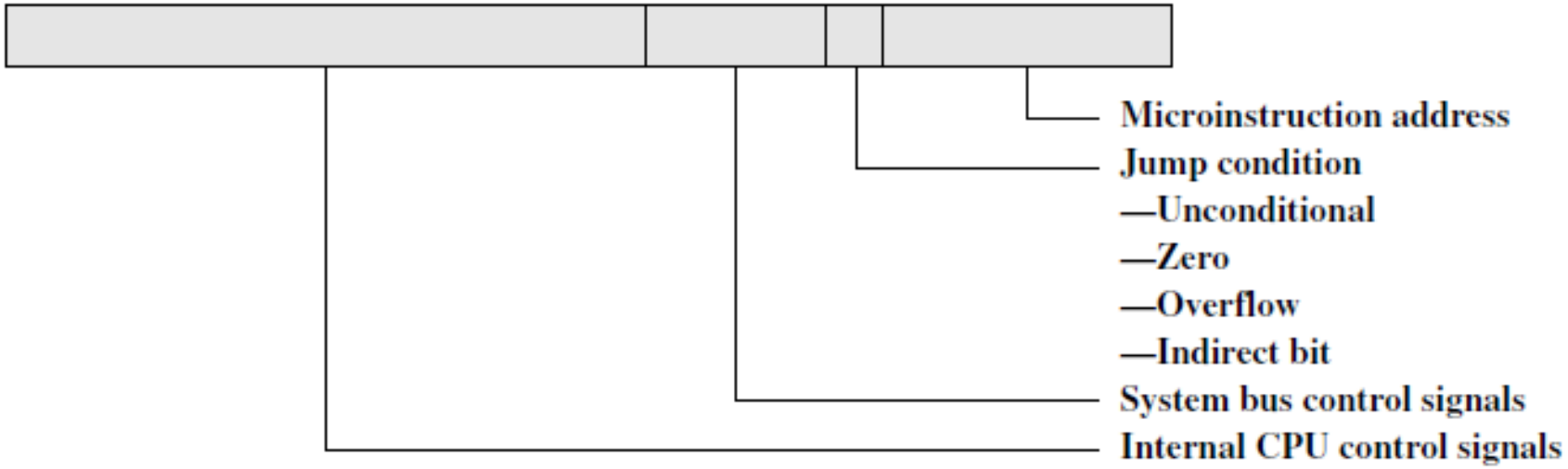
- Each micro-instruction specifies single (or few) micro-operations to be performed
  - (*vertical* micro-programming)
- Each micro-instruction specifies many different micro-operations to be performed in parallel
  - (*horizontal* micro-programming)

Horizontal Micro-programming

- Wide memory word
- High degree of parallel operations possible
- Little encoding of control information

Microinstruction Format: Horizontal

- **1.** To execute this microinstruction, turn on all the control lines indicated by a 1 bit; leave off all control lines indicated by a 0 bit. The resulting control signals will cause one or more micro-operations to be performed.
- **2.** If the condition indicated by the condition bits is false, execute the next microinstruction in sequence.
- **3.** If the condition indicated by the condition bits is true, the next microinstruction to be executed is indicated in the address field.

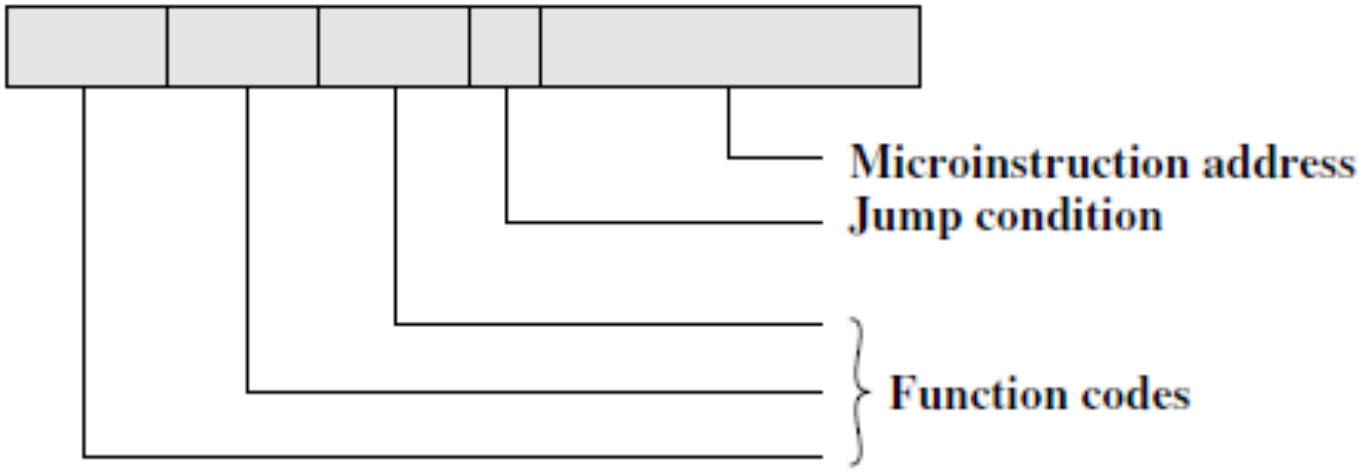


(a) Horizontal microinstruction

Microinstruction Format: Vertical

Vertical Microinstruction:

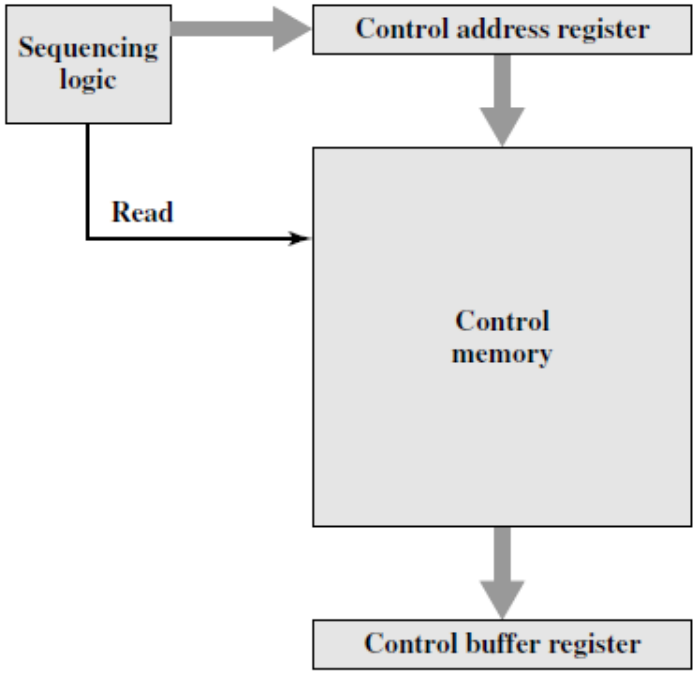
- Width is narrow
- Limited ability to express parallelism
- Considerable encoding of control information requires external memory word decoder to identify the exact control line being manipulated
- In a vertical microinstruction, a code is used for each action to be performed [e.g MAR ⟵ PC]



(b) Vertical microinstruction

Microprogrammed Control Unit

- The set of microinstructions is stored in the *control memory.*
- The *control address register* contains the address of the next microinstruction to be read.
- When a microinstruction is read from the control memory, it is transferred to a *control buffer register.*
- A sequencing unit that loads the control address register and issues a read command.



Control Unit Microarchitecture

Microprogrammed Control Unit: Functioning

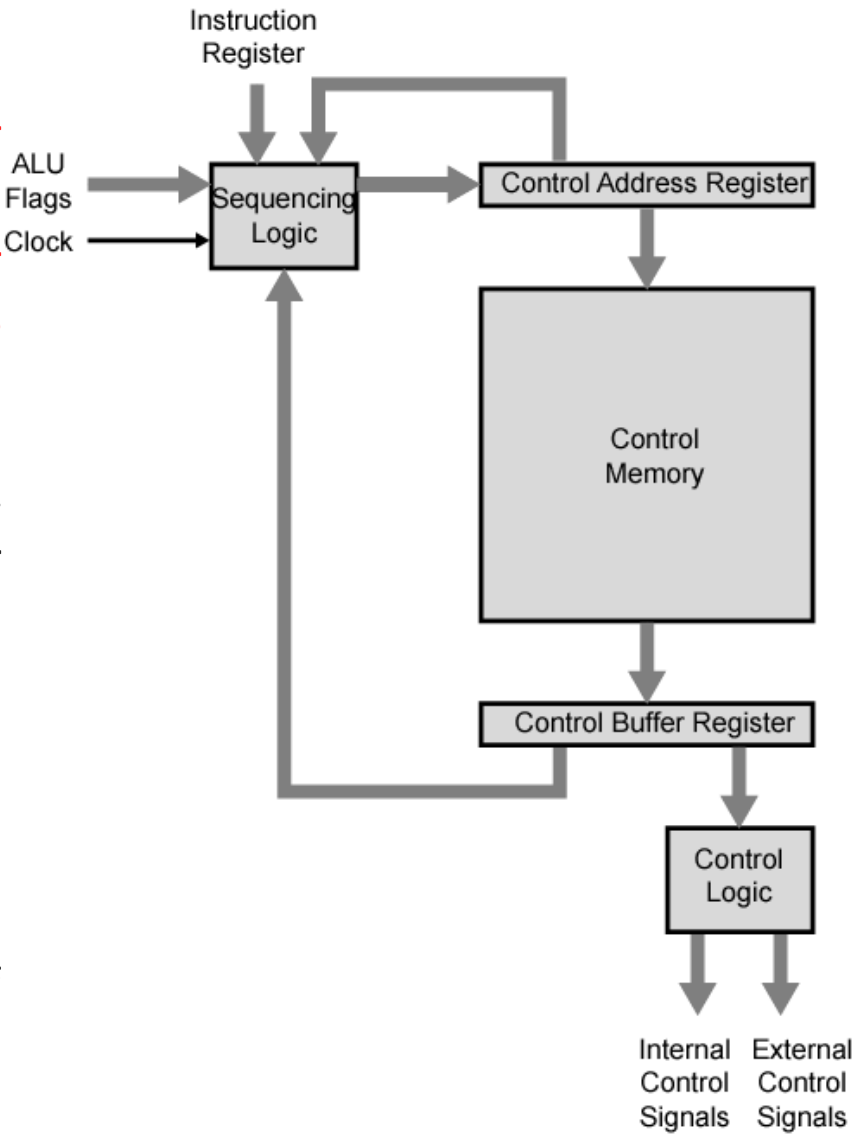
1. To execute an instruction, the sequencing logic unit issues a READ command to the control memory.

2. The word whose address is specified in the control address register is read into the control buffer register.

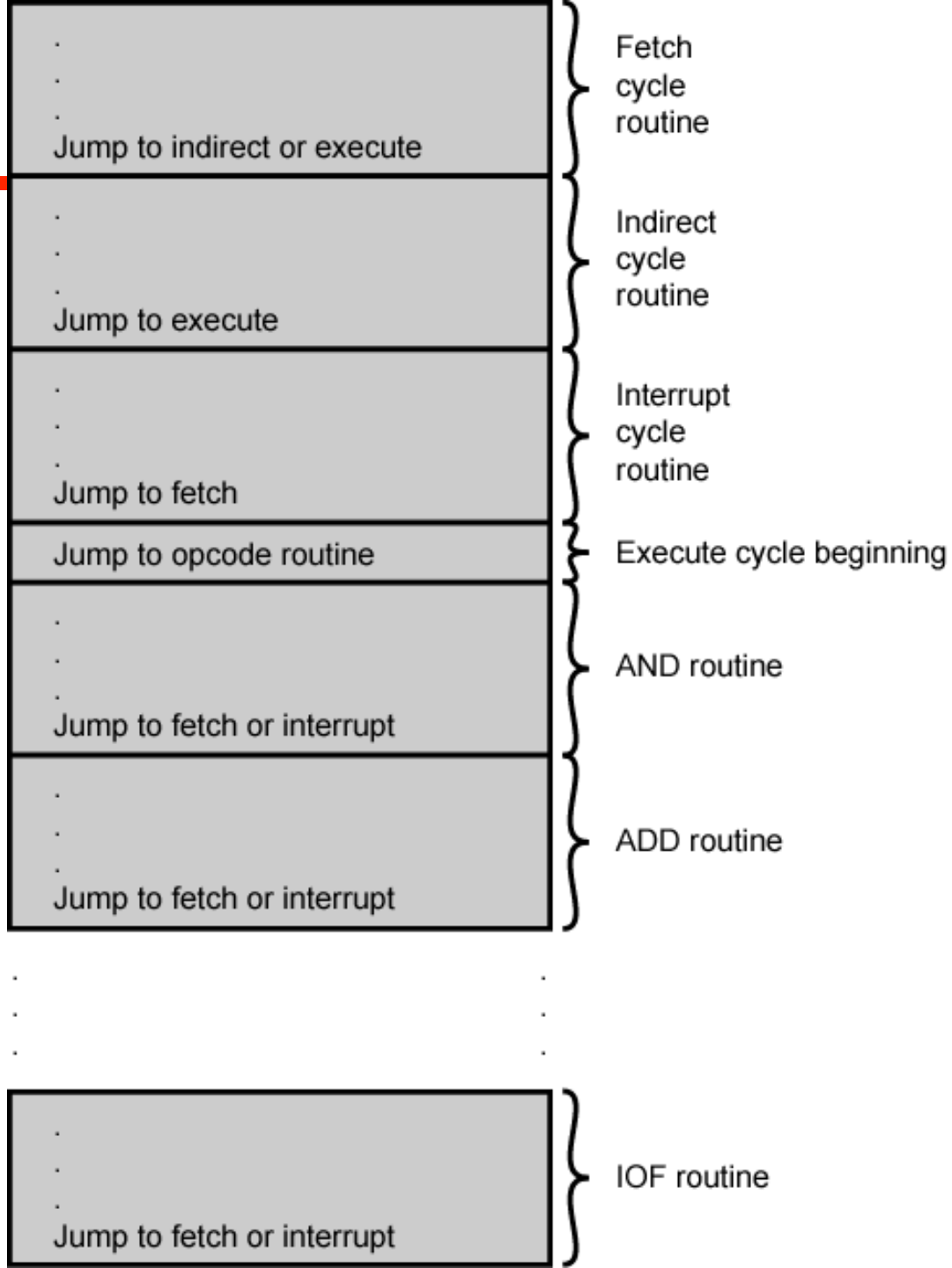
3. The content of the control buffer register generates control signals and nextaddress information for the sequencing logic unit.

4. The sequencing logic unit loads a new address into the control address register based on the next-address information from the control buffer register and the ALU flags. Depending on the value of the ALU flags and the control buffer register, one of three decisions is made based on the opcode in the IR.

- **Get the next instruction:** Add 1 to the control address register.
- **Jump to a new routine based on a jump microinstruction:** Load the address field of the control buffer register into the control address register. (Interrupt)
- **Jump to a machine instruction routine:** Load the control address register(Indirect addressing)



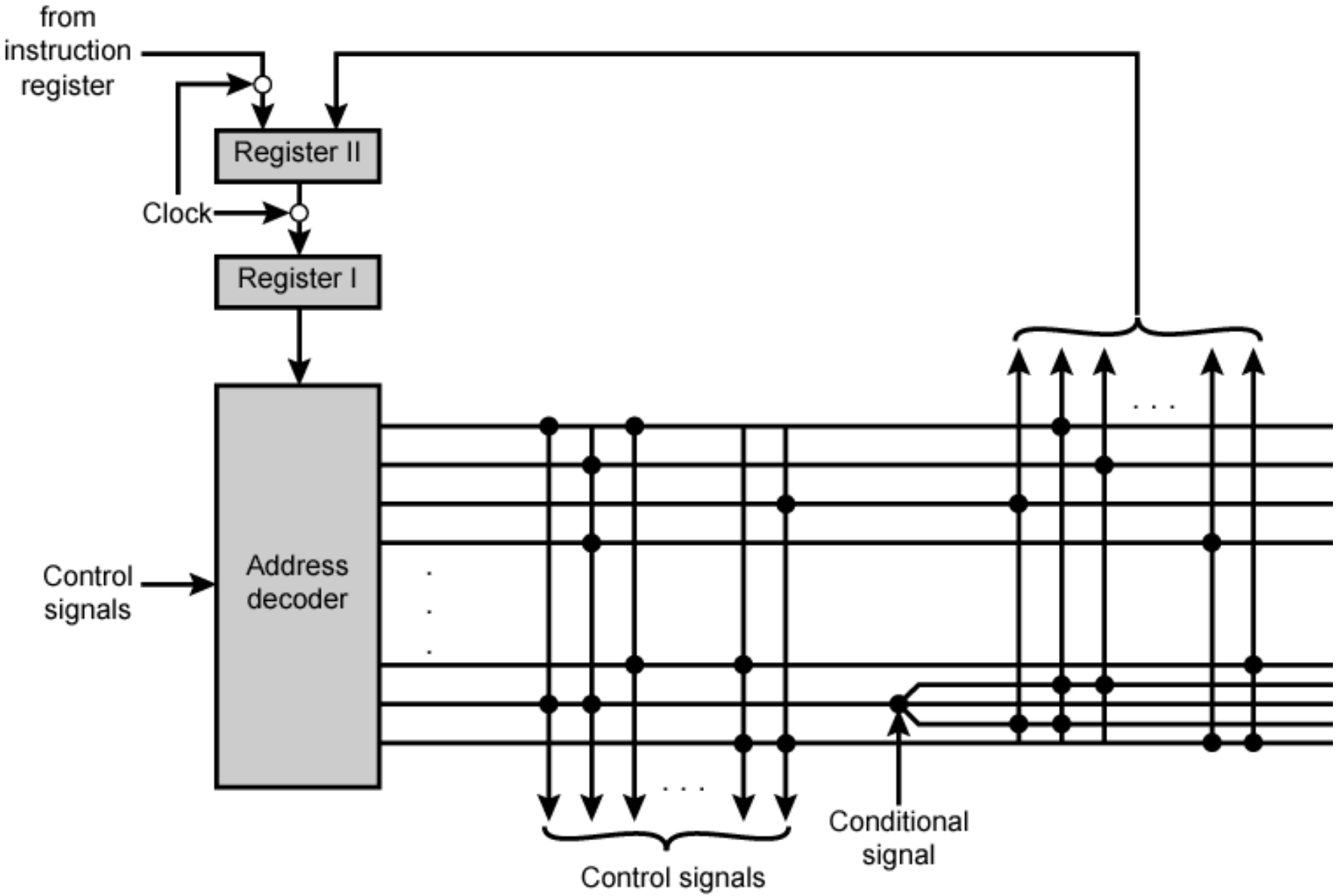
Organization of Control Memory



Wilkes: Microprogrammed Control Unit

- Proposed by Wilkes in 1951
- Matrix partially filled with diodes
- During cycle, one row activated
  - Generates signals where diode present
  - First part of row generates control
  - Second generates address for next cycle

Wilkes's Microprogrammed Control Unit



Advantages and Disadvantages of Microprogramming

- Simplifies design of control unit
  - Cheaper
  - Less error-prone
- Slower

Tasks Done By Microprogrammed Control Unit

- **Microinstruction sequencing:** Get the next microinstruction from the control memory.
- **Microinstruction execution:** Generate the control signals needed to execute the microinstruction.

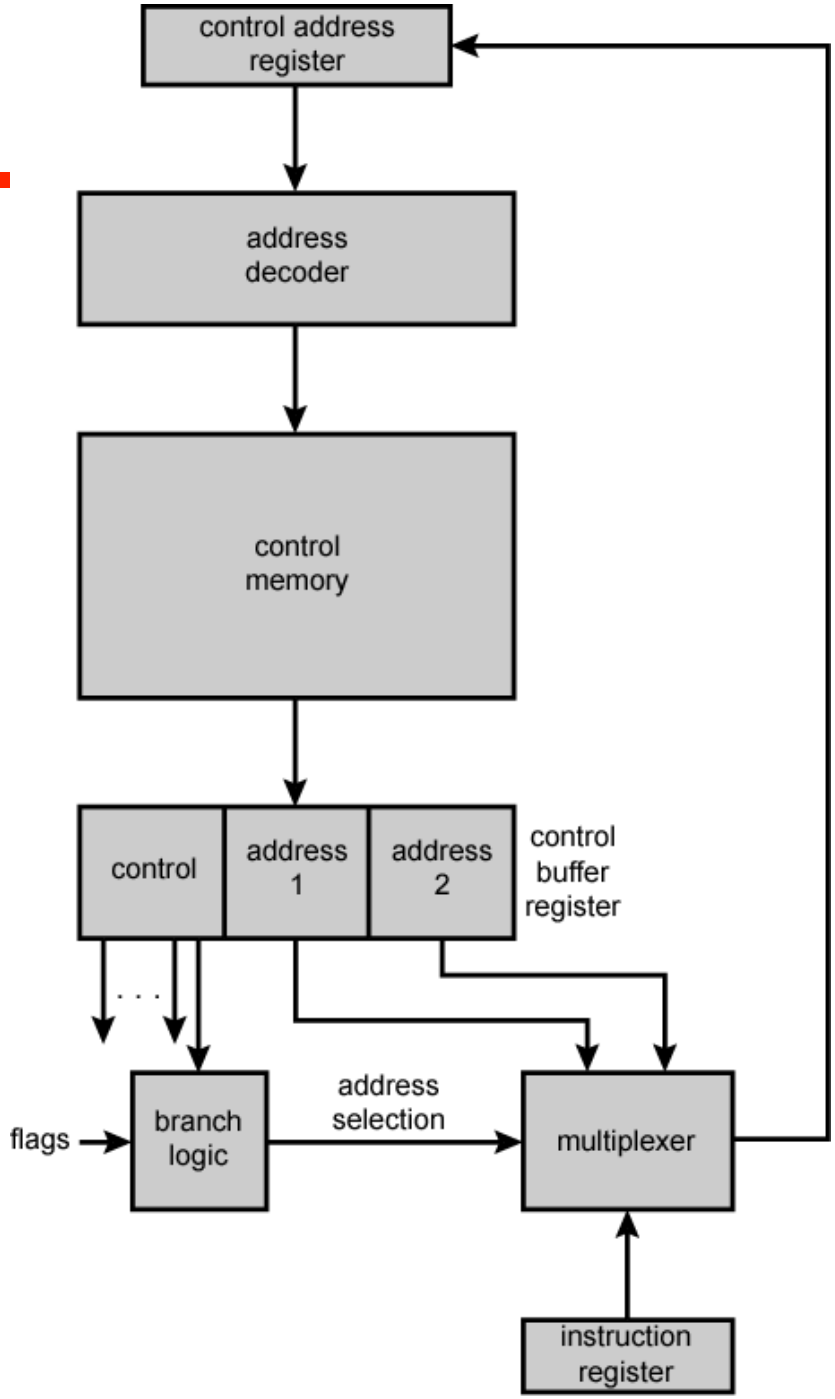
Microinstruction sequencing: Design Considerations

- Size of microinstructions
- Address generation time
  - Determined by instruction register
    - Once per cycle, after instruction is fetched
  - Next sequential address
    - Common in most designed
  - Branches
    - Both conditional and unconditional

Sequencing Techniques

- Based on current microinstruction, condition flags, contents of IR, control memory address must be generated
- Based on format of address information
  - Two address fields
  - Single address field
  - Variable format

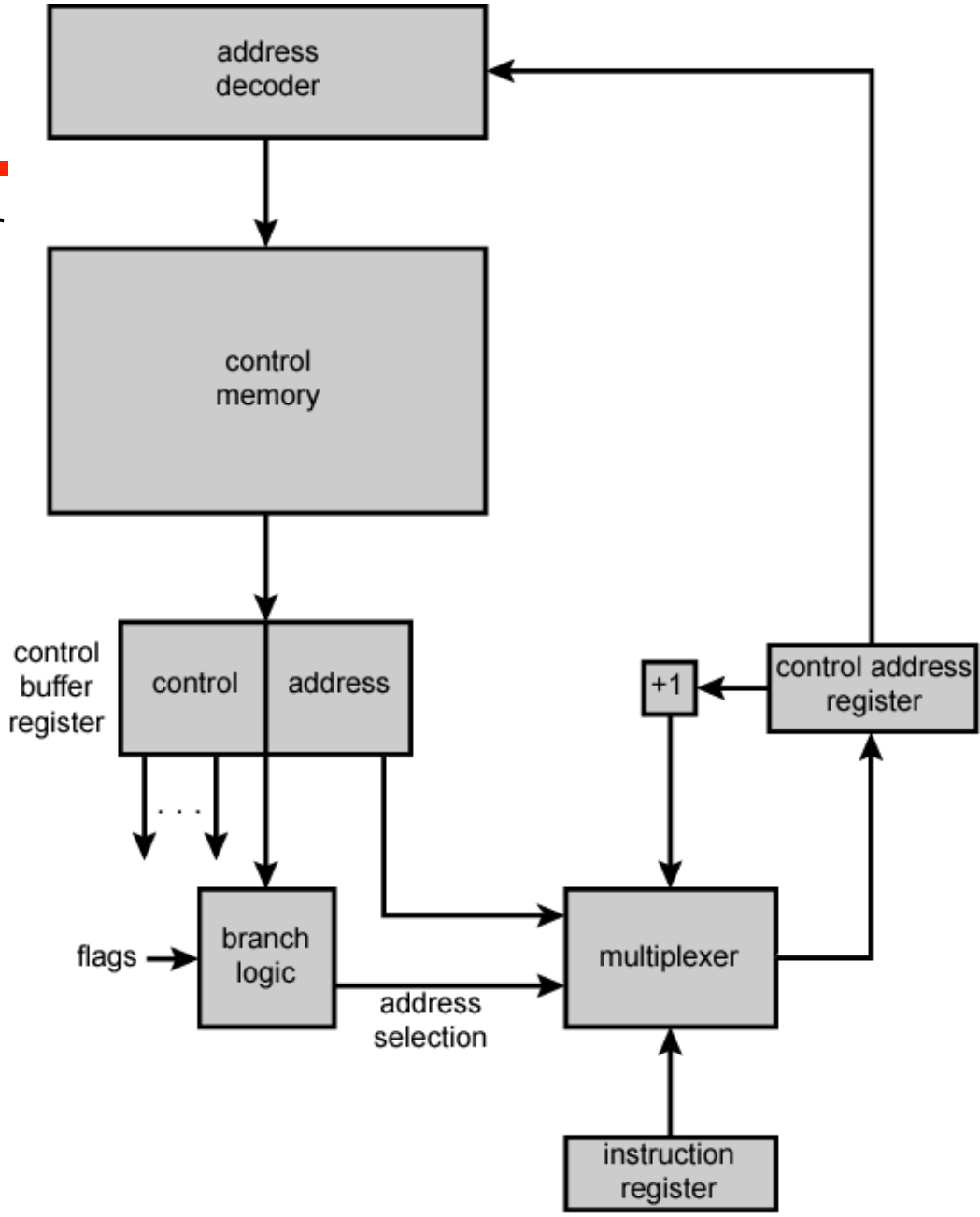
Branch Control Logic: Two Address Fields



Branch Control Logic: Single Address Field

With this approach, the options for next address are as follows:

- Address field
- Instruction register code
- Next sequential address

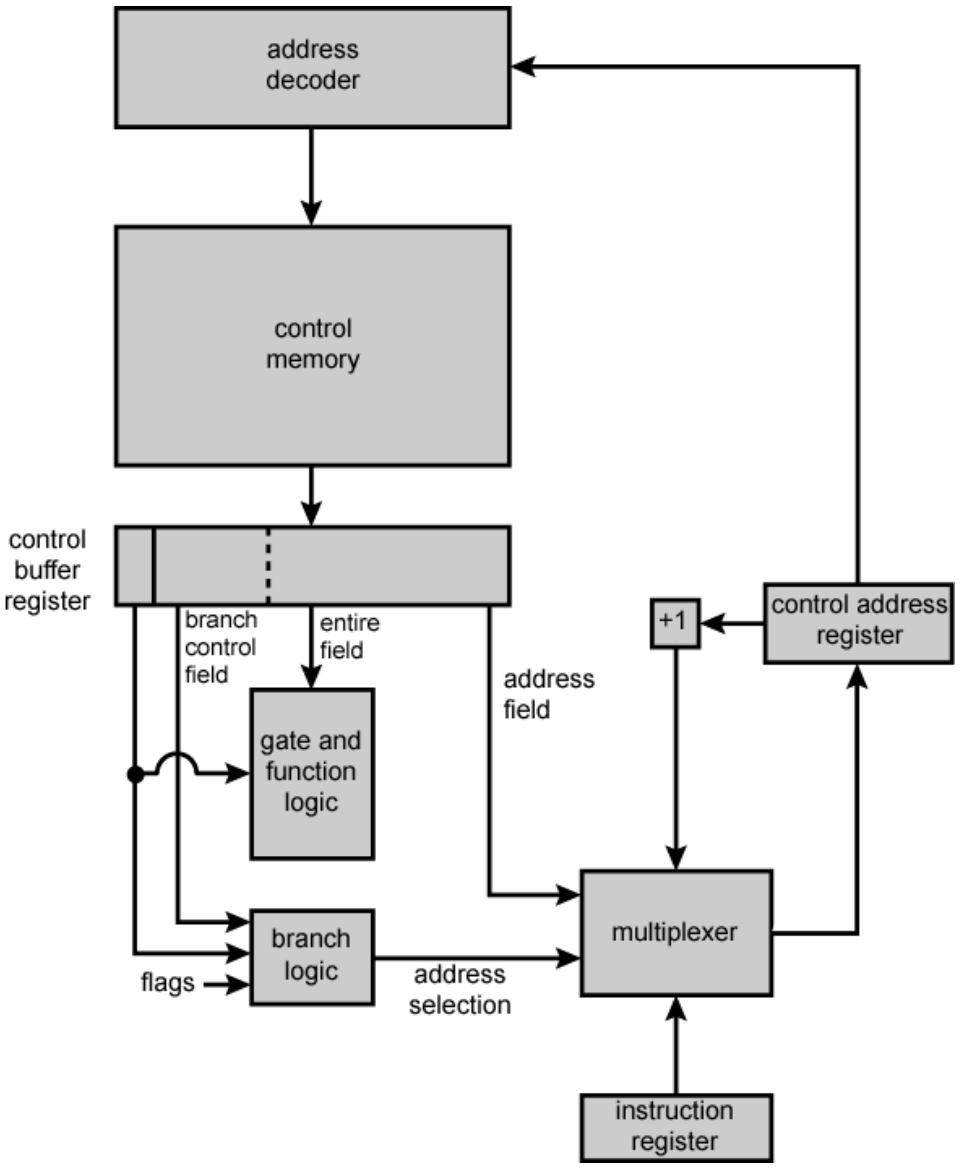


Branch Control Logic: Variable Format

Another approach is to provide for two entirely different microinstruction formats

- One bit designates which format is being used.
- In one format, the remaining bits are used to activate control signals.
- The next address is either the next sequential address or an address derived from the instruction register.
- In the other format, some bits drive the branch logic module, and the remaining bits provide the address.
- With the second format, either a conditional or unconditional branch is being specified.

One disadvantage of this approach is that one entire cycle is consumed with each branch microinstruction. With the other approaches, address generation occurs as part of the same cycle.



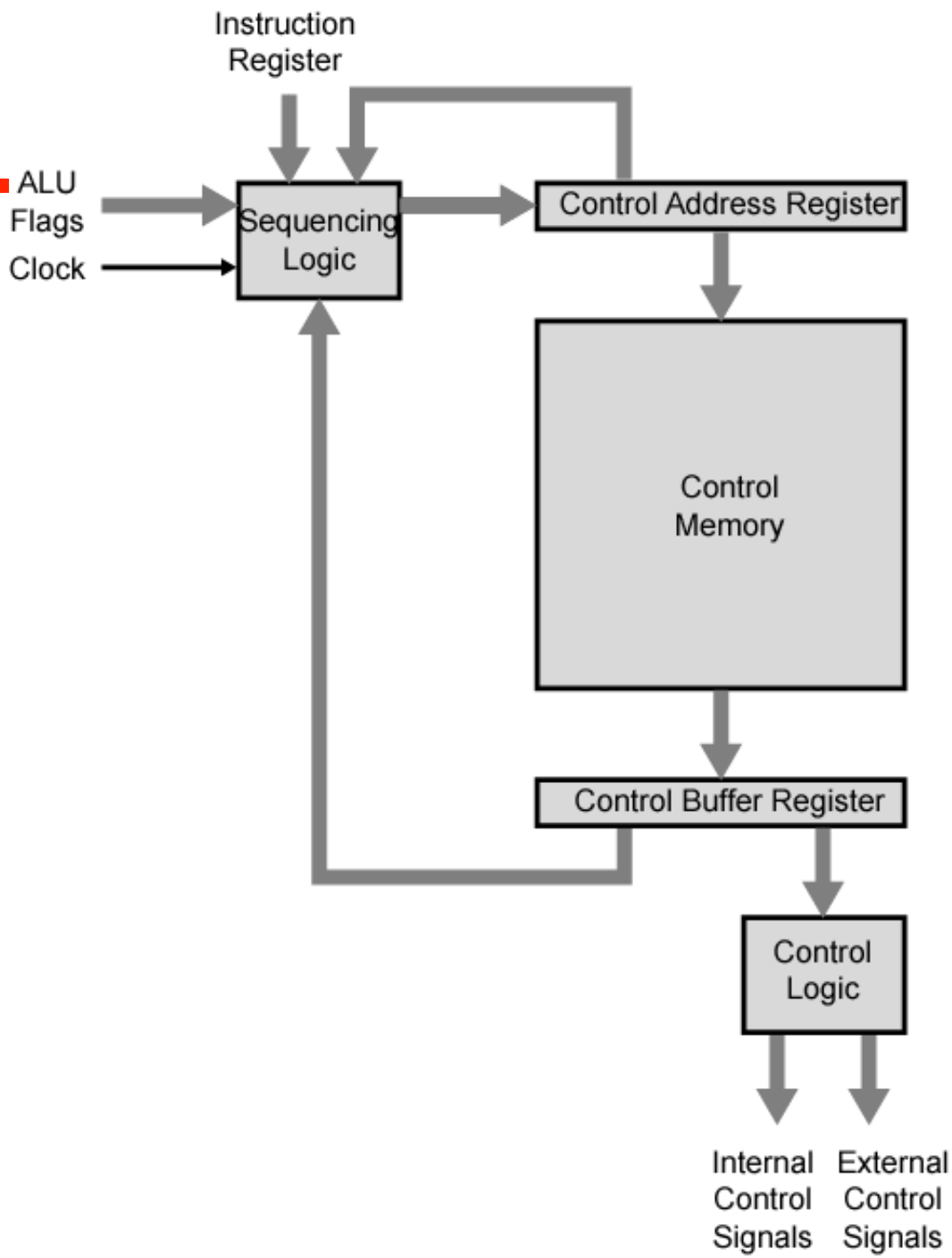
Address Generation

Explicit	Implicit
Two-field	Mapping
Unconditional Branch	Addition
Conditional branch	Residual control

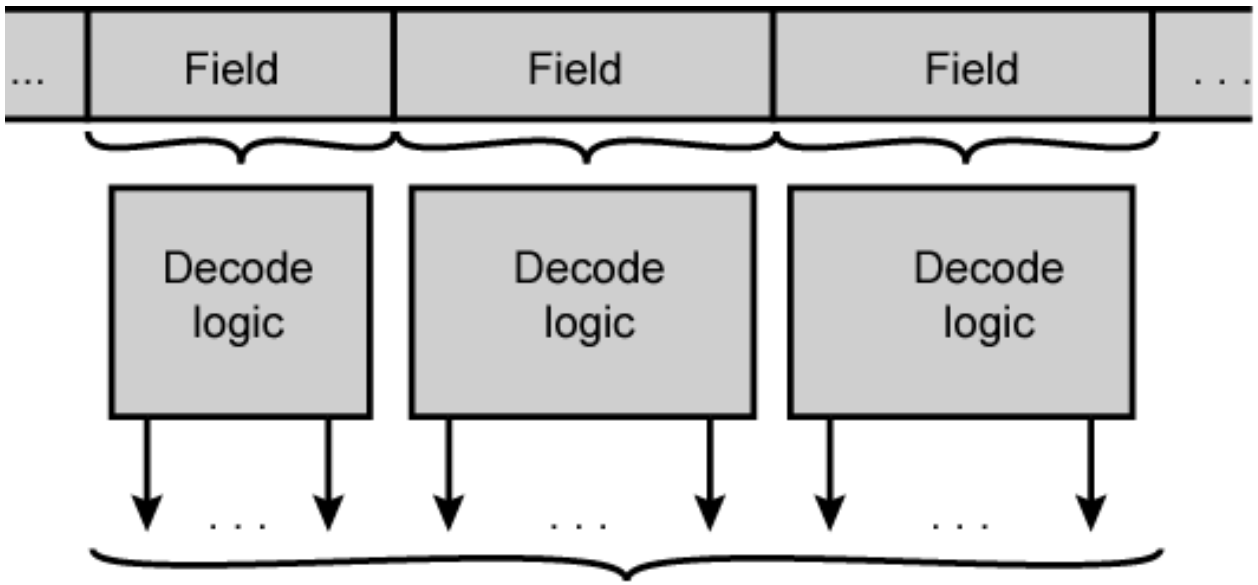
Microinstruction Execution

- The cycle is the basic event
- Each cycle is made up of two events
  - Fetch
    - Determined by generation of microinstruction address
  - Execute
    - Effect is to generate control signals
    - Some control points internal to processor
    - Rest go to external control bus or other interface

Control Unit Organization

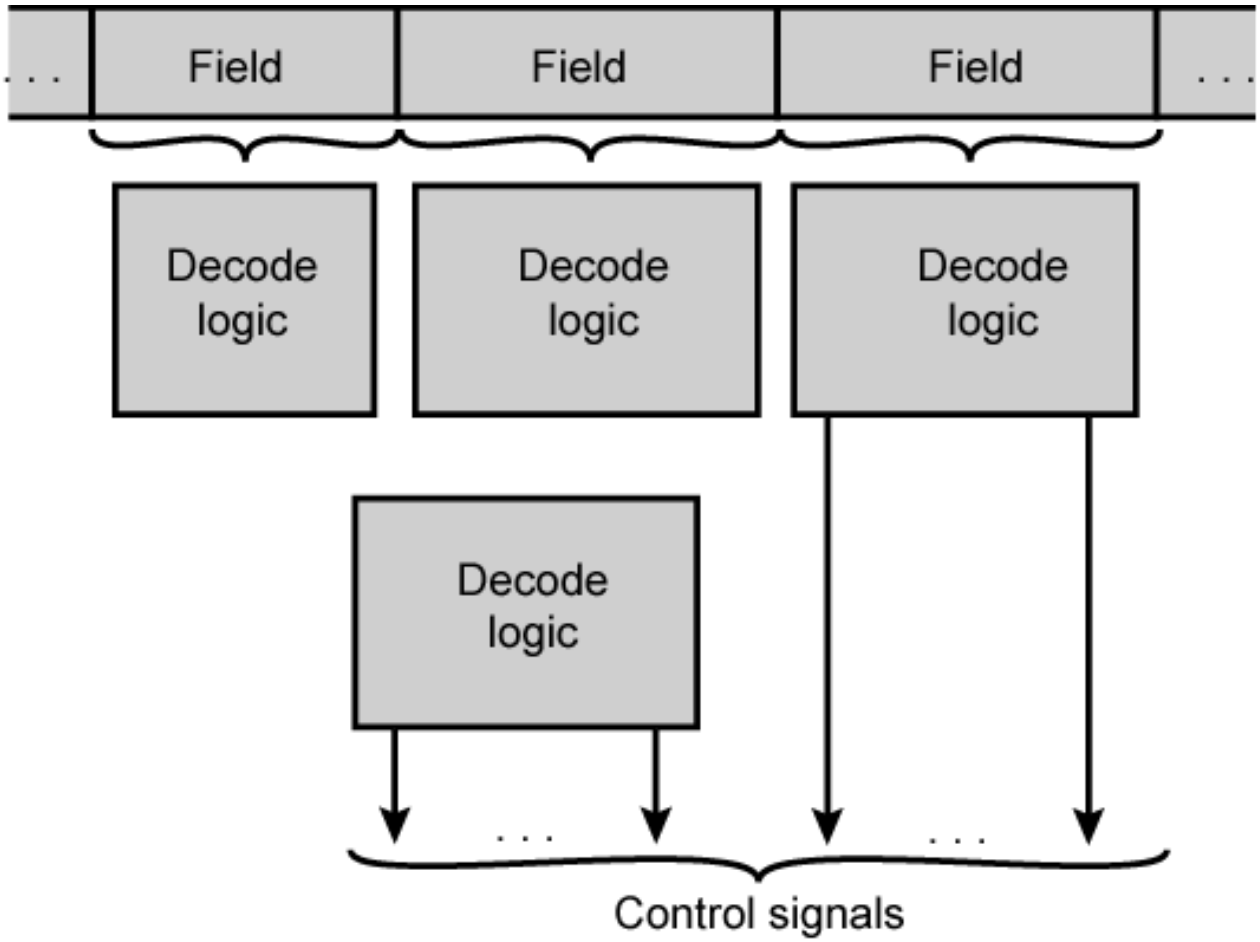


Microinstruction Encoding: Direct Encoding



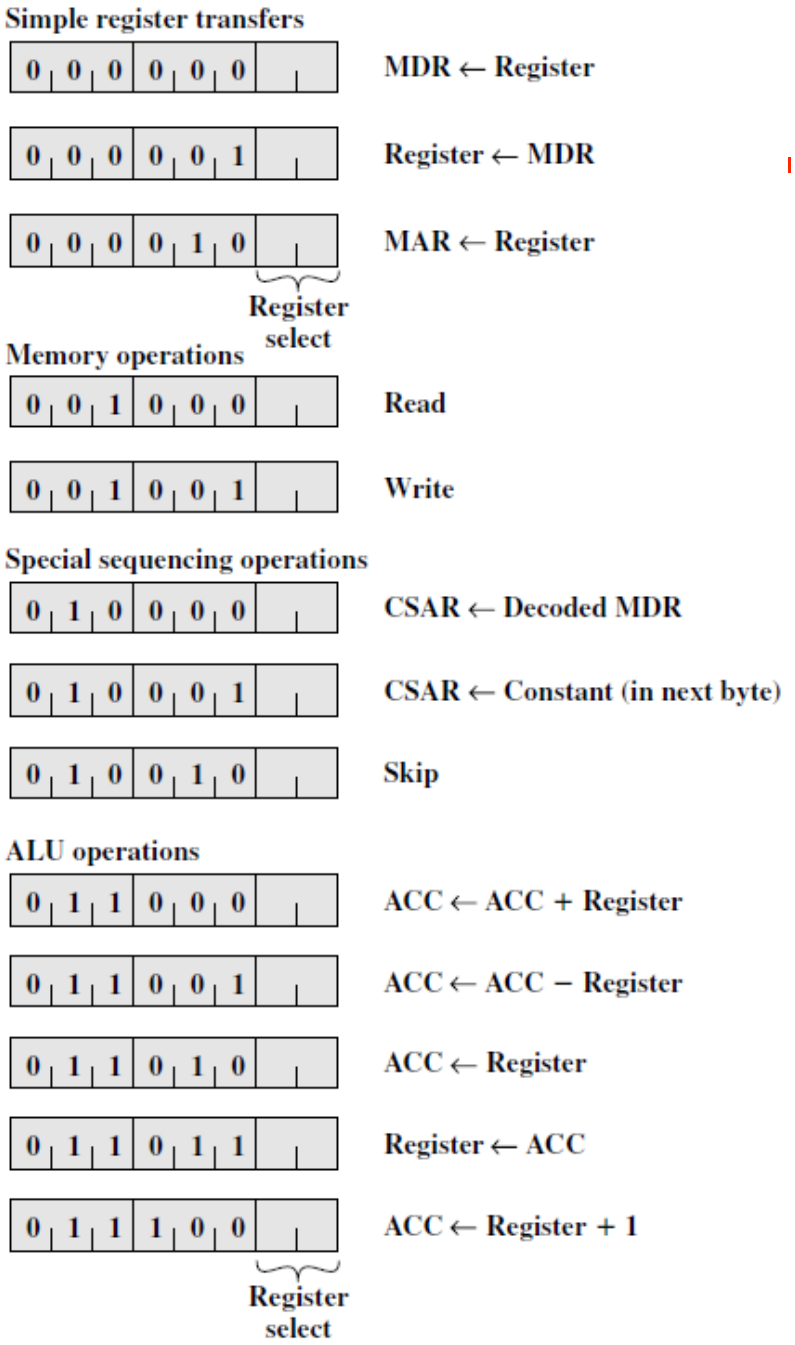
(a) Direct encoding

Microinstruction Encoding: Indirect Encoding



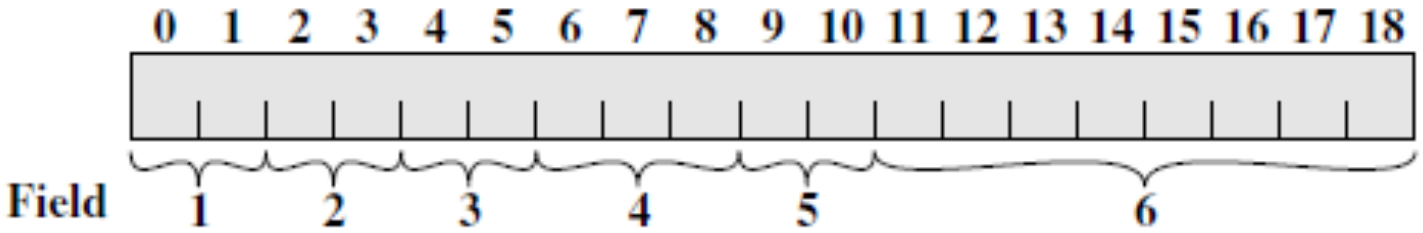
(b) Indirect encoding

Microinstruction Format: Vertical



(a) Vertical microinstruction format

Microinstruction Format: Horizontal

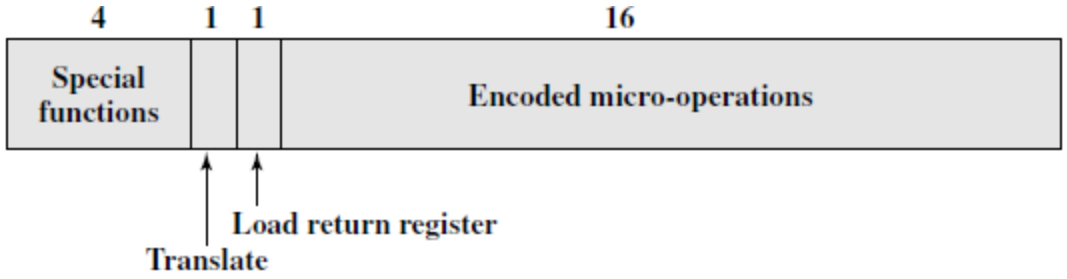


Field definition

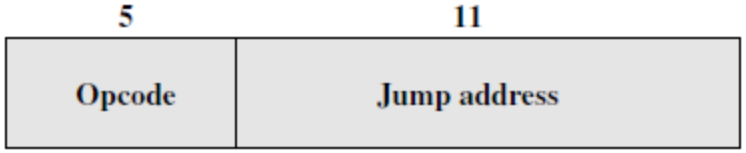
1—register transfer	4—ALU operation
2—memory operation	5—register selection
3—sequencing operation	6—Constant

(b) Horizontal microinstruction format

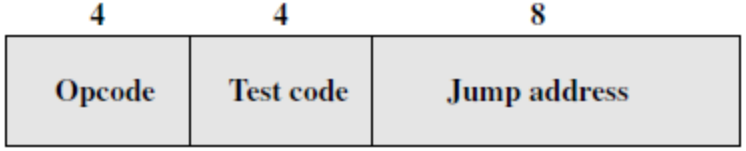
Example: LSI-I1 Microinstruction Format



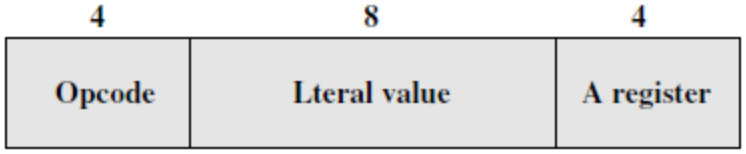
(a) Format of the full LSI-11 microinstruction



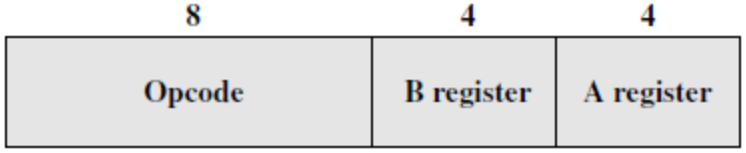
Unconditional jump microinstruction format



Conditional jump microinstruction format



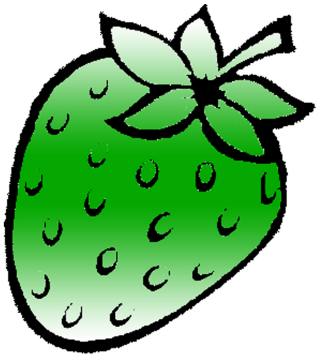
Literal microinstruction format



Register jump microinstruction format

Format of the encoded part of the LSI-11 microinstruction

STRAWBERRY



/strawberrydevelopers

/strawberry_app

For more visit:

Strawberrydevelopers.weebly.com