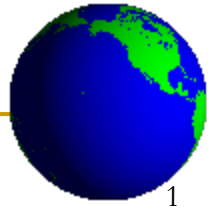
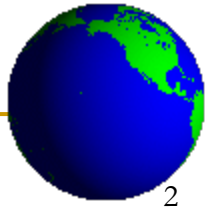
MICROPROGRAMMED CONTROL

- **Control Memory**
- **Address Sequencing**
- **Microprogram Example**
- **Design of Control Unit**



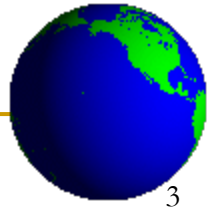
Types of control unit

- Two major types of Control Unit
  - Hardwired Control
  - The control logic is implemented with gates, F/Fs, decoders, and other digital circuits
    - + Fast operation, - Wiring change (if the design has to be modified)
  - Microprogrammed Control
    - The control information is stored in a control memory, and the control memory is programmed to initiate the required sequence of microoperations
    - + Any required change can be done by updating the microprogram in control memory, - Slow operation



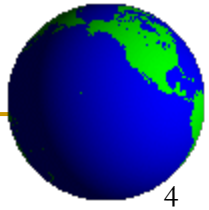
Hardwired Control

- Complex
- Fast
- Difficult to design
- Difficult to modify
  - - Lots of optimization done at implementation phase



Micro-programmed Control

- Implement "execution engine" inside CPU
  - execute one micro-instruction at a time
- What to do now?
  - micro-instruction
    - control signals
  - stored in micro-instruction control memory
    - micro-program, firmware
- What to do next?
  - micro-instruction program counter
    - default next micro-instruction
    - jumps or branches

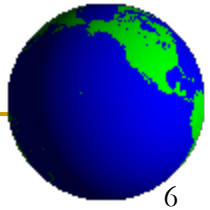


Hardwired vs. micro programmed

- Once the control unit of a hard-wired computer is designed and built, it is virtually impossible to alter its architecture and instruction set. In the case of a micro-programmed computer, however, we can change the computer's instruction set simply by altering the micro program stored in its control memory.

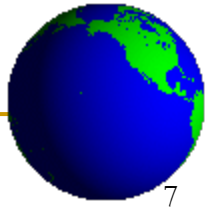
Control function

- The control function that specifies a micro operation is a binary variable. When its in the active state, the corresponding microoperation is executed
- In a bus organized system, the control signals that specify microoperations are groups of bits that select the paths in multiplexers, decoders, and ALUs



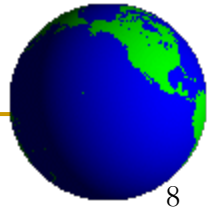
Control word

- The control unit initiates a series of sequential steps of microoperations.
- The control variables at any given time can be represented by a string of 1's and 0's called a control word.
- A control unit whose binary control variables are stored in memory is called a microprogrammed control unit



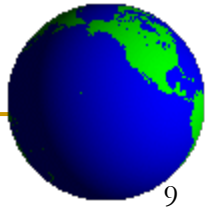
Microprogram

- Each word in control memory contains within it a microinstruction.
- The microinstruction specifies one or more microoperations for the system
- A sequence of microinstructions constitutes a microprogram
- Since alterations of the microprogram are not needed once the control unit is in operation, the control memory can be a ROM.



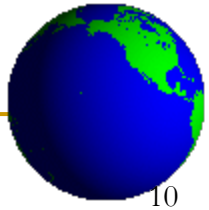
TERMINOLOGY

- **Microinstruction**
  - **Contains a control word and a sequencing word**
    - **Control Word - All the control information required for one**
    - **Sequencing Word - Information needed to decide the next microinstruction address**
  - **Vocabulary to write a microprogram**
- **Control memory**
  - **Storage in the microprogrammed control unit to store the microprogram**



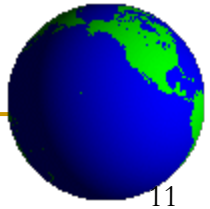
Dynamic programming

- It permits a micrprogram to be loaded initially from an auxiliary memory such as a magnetic disk.
- Control units that use dynamic microprogramming employ a writable control memory.



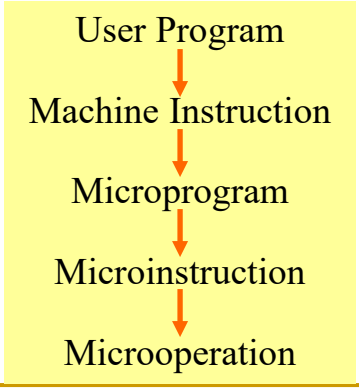
Control memory and main memory

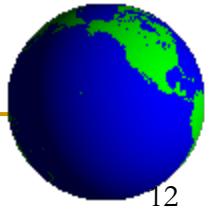
- The main memory is available to the user for storing the programs. The contents of the main memory may alter when the data are manipulated and every time that the program is changed.
- Control memory holds a fixed microprogram that cannot be altered by the occasional user. The microprogram consists of microinstructions that specify various internal control signals for execution of register microoperations.



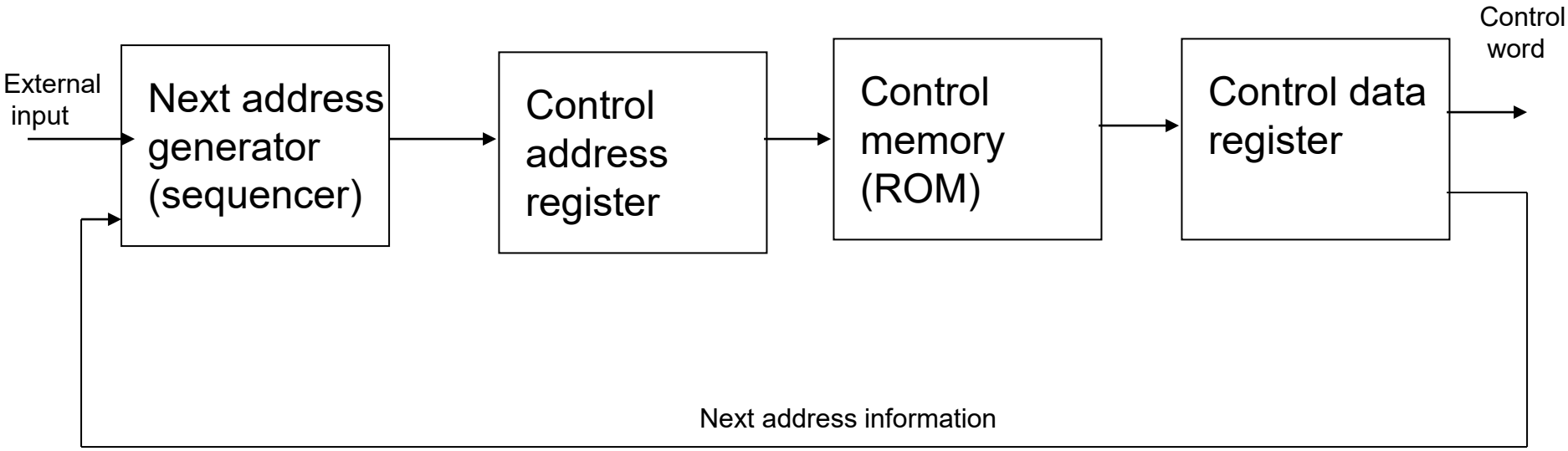
Machine instruction

- Each machine instruction initiates a series of microinstructions in control memory. These microinstructions generate the microoperations to fetch the instruction from main memory, evaluate the effective address, execute the operation specified by the instruction and to return to the fetch phase in order to repeat the cycle for next instruction.



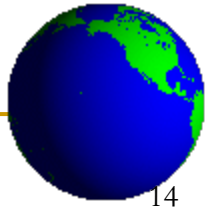


Microprogrammed control organization



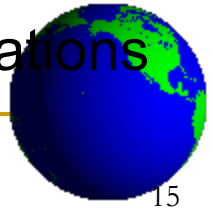
Microprogrammed control organization

- Microprogrammed control Organization :
  - 1) Control Memory
    - A memory is part of a control unit : *Microprogram*
  - 2) Control Address Register
    - Specify the address of the microinstruction
  - 3) Control data register (*pipeline register*)
    - Holds the instruction read from memory
    - Allows the execution of the microoperations specified by the control word simultaneously with the generation of the next microinstruction
    - This configuration requires a two phase clock, with one clock applied to the address register and the other to the data register
  - 4) Sequencer (= *Next Address Generator*)



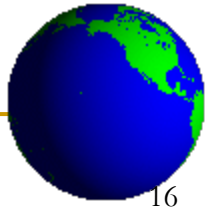
Sequencer

- The next address generator is sometimes called the sequencer, as it determines the address sequence that is read from control memory. The address can be specified in several ways, depending on the sequencer inputs.
- Typical functions of the sequencer are:-
  - Loading into the control address register an address from control memory
  - Transferring an external address
  - Loading an initial address to start the control operations



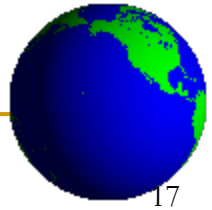
Address sequencing

- Microinstructions are stored in control memory in groups, with each group specifying a routine
- Each computer instruction has its own microprogram routine in control memory to generate the microoperations that execute the instruction
- The hardware that controls the address sequencing of the control memory must be capable of sequencing the microinstructions within a routine and be able to branch from one routine to another



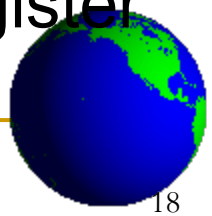
Steps of address sequencing

- An initial address is loaded into the control register when computer is turned on. This address is actually the address of the first instruction that activates the instruction fetch routine
- The fetch routine may be sequenced by incrementing the control address register through the rest of the microinstructions.
- After the fetch routine, the instruction is in the instruction register of the computer.



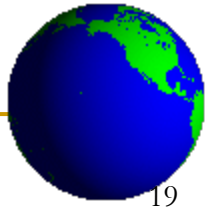
Steps of address sequencing cont…

- The control memory next must go through the routine that determines the effective address of the operand.
- A machine instruction may have bits that specify the various addressing modes.
- The effective address computation routine can be reached thru a branch instruction, which is conditioned on the status of the mode bits of the instruction
- After the completion of the routine, the address of the operand is avail in the memory address register



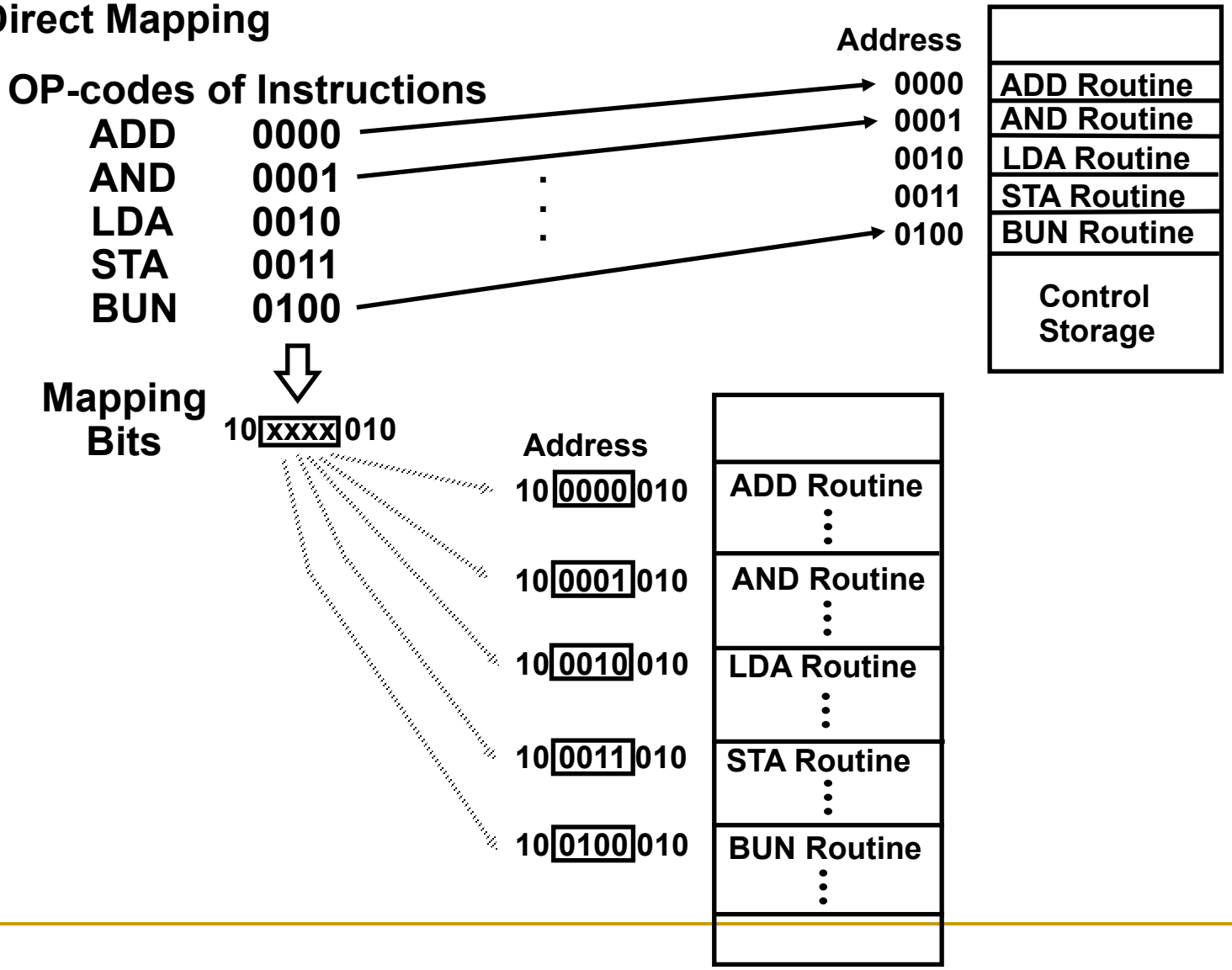
Steps of address sequencing cont…

- The next step is to generate the microoperations that execute the instruction fetched from memory which depends upon the op code.
- Each instruction has its own microprogram routine stored in a given location of control memory
- The transformation from the instruction code bits to an address in control memory where the routine is located is called *MAPPING*



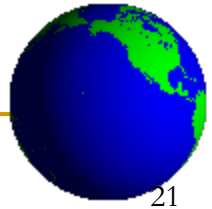
MAPPING OF INSTRUCTIONS

Direct Mapping



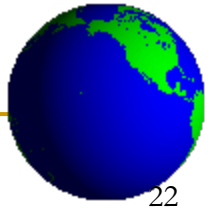
Branching

- Once the required routine is reached, the microinstruction that execute the instruction may be sequenced by
  - Incrementing the control address register
  - Sequence of microoperations will depend on values of certain status bits in processor registers
- When the execution of the instruction is completed, the control must return to the fetch routine by executing an unconditional branch microinstruction to the first address of the fetch routine

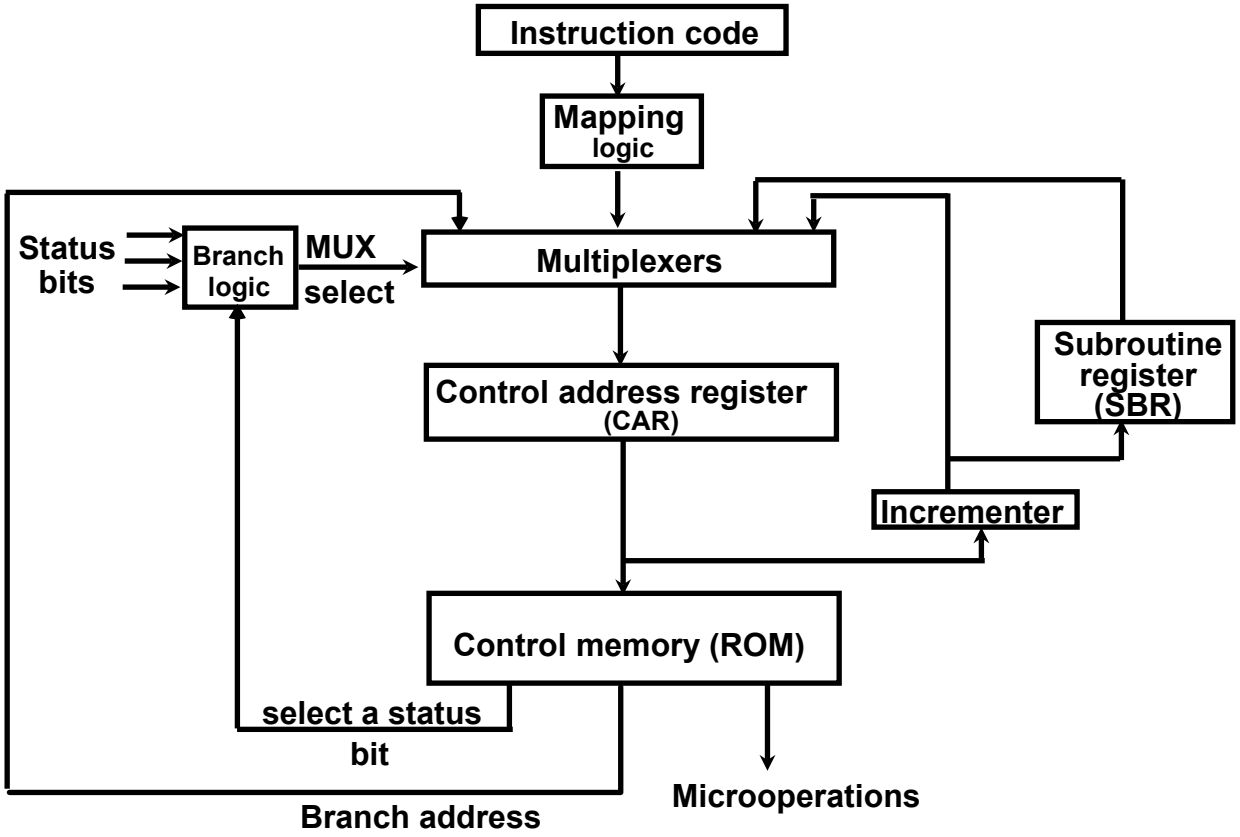


Address sequencing capabilities

- The Address sequencing capabilities required in a control memory are:-
  - Incrementing of the control address register
  - Unconditional branch or conditional branch, depending on status bit conditions
  - A mapping process from the bits of the instruction to an address for control memory
  - A facility for subroutine call and return

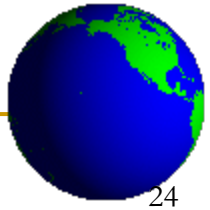


MICROINSTRUCTION SEQUENCING



Control address register

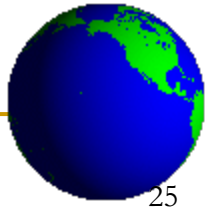
- There are 4 different paths from which the CAR receives the address:-
  - Incrementer increments the CAR by 1, to select the next microinstruction in sequence
  - Branching is achieved by specifying the branch address in one of the fields of the microinstruction
  - Conditional branching is obtained by using a part of the microinstruction to select specific status bits in order to determine its condition
  - An external address in transferred into control memory via a mapping logic circuit
  - The return address for a subroutine is stored in a special register whose value is then used when the microprogram wishes to return from the subroutine



Sequencer (Microprogram Sequencer)

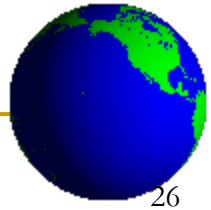
A Microprogram Control Unit that determines the Microinstruction Address to be executed in the next clock cycle

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



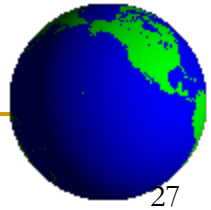
Conditional branching

- The branch logic provides decision making capabilities in the control unit. The status bits in the system that provide parameter information such as
  - Carry out of an adder
  - The sign bit of a number
  - The mode bits of an instruction
  - Input or output status conditions



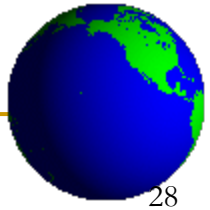
Branch logic

- The simplest way to implement the branch logic hardware is to test the specified condition and branch to the indicated address if the condition is met; otherwise the address register is incremented.
- This can be implemented with a MUX.

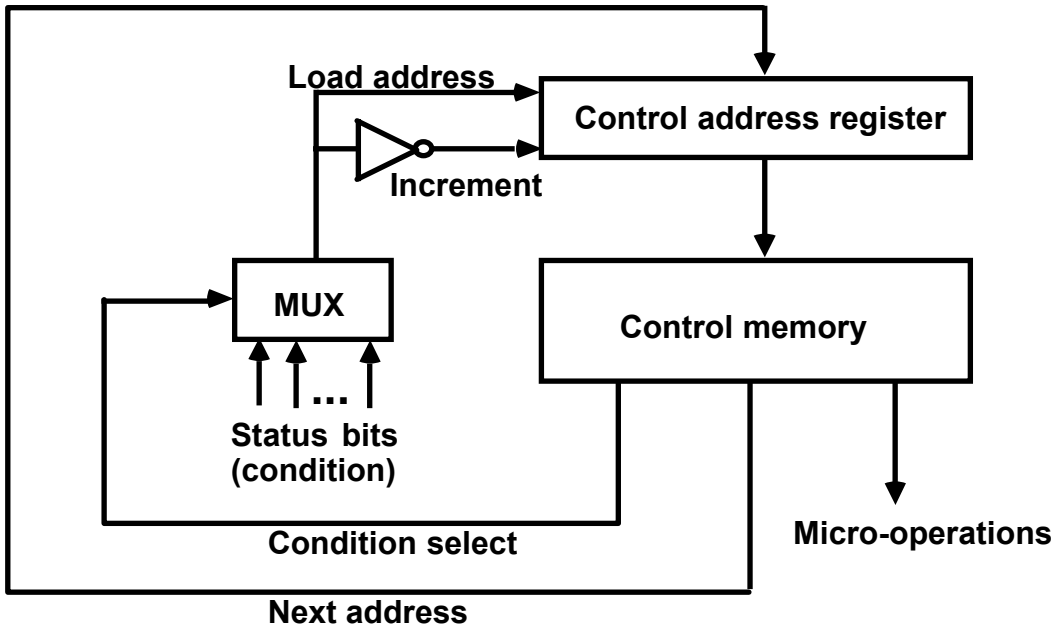


Branch logic cont…

- Suppose there are 8 status bit conditions in the system
- 3 bits in the microinstruction are used to specify any of the 8 status bit conditions. These three bits provide the selection variables for the multiplexer.
- If the selected status bit is in the 1 state, the output of the MUX is 1,otherwise its 0.
- A 1 output generates a control signal to transfer the branch address from the microinstruction into the control address register
- A 0 output causes the address register to be incremented



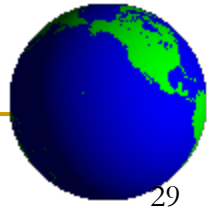
CONDITIONAL BRANCH



Conditional Branch

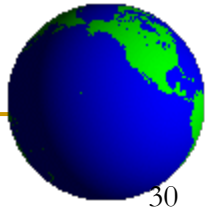
If *Condition* is true, then *Branch* (address from the next address field of the current microinstruction) else *Fall Through*

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



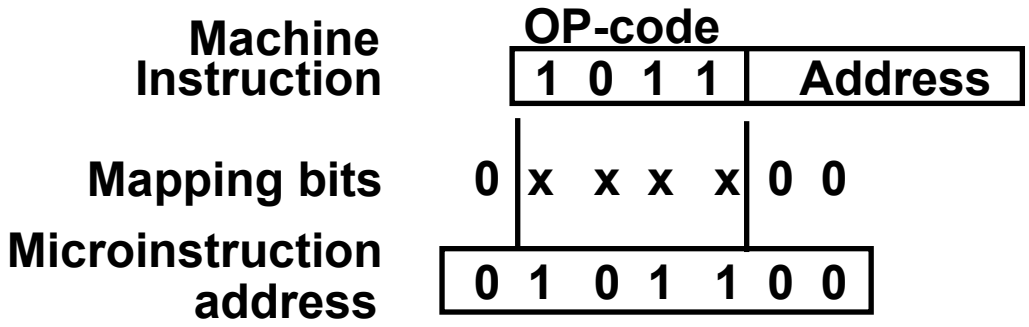
Unconditional branch

- It can be implemented by loading the branch address from the control memory into the control address register.
- This can be accomplished by fixing the value of one status bit at the input of the MUX, so its always equal to 1

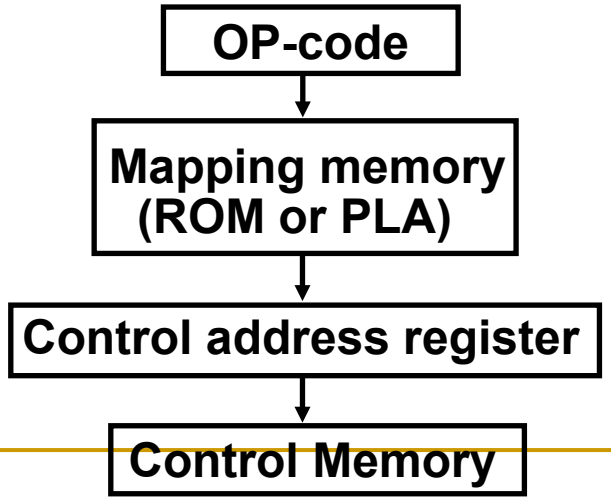


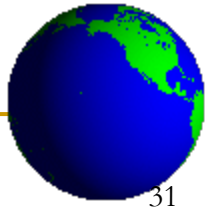
MAPPING OF INSTRUCTIONS TO MICROROUTINES

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



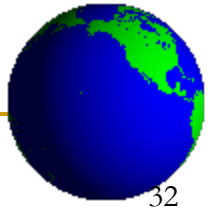
Mapping function implemented by ROM or PLA





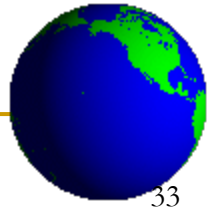
Subroutines

- Many microprograms contain identical sections of code. Microinstructions can be saved by employing subroutines that use common sections of microcode.
  - Example: the sequence of microoperations to generate the effective address of an operand is common to all MRIs
- Microprograms that use subroutines must have a provision for storing the return address during a subroutine call and restoring the address during a subroutine return. This is done by subroutine register

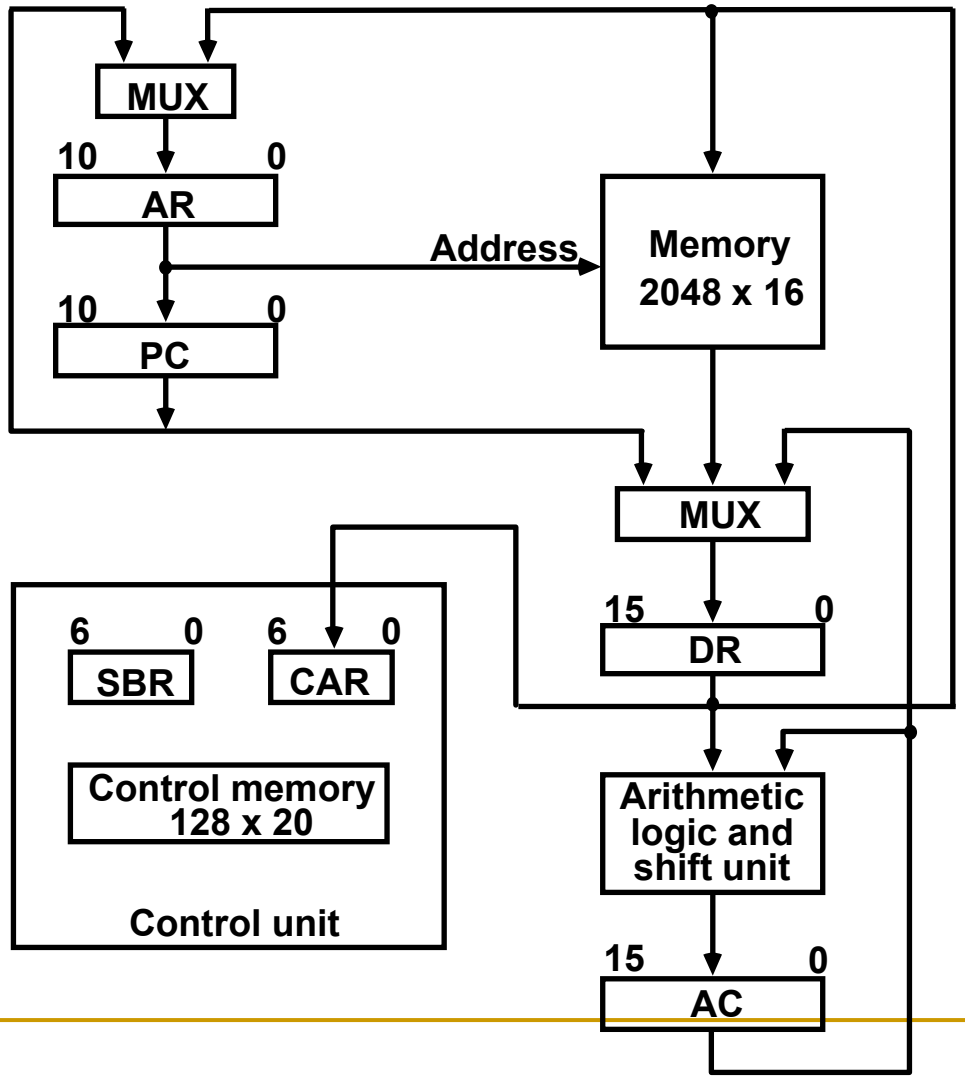


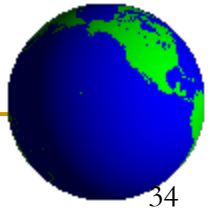
MICROPROGRAMMING

- Once the config of the computer and its microprogrammed control unit is done, the designer's task it to generate the microcode for the control memory. This code generation is called microprogramming



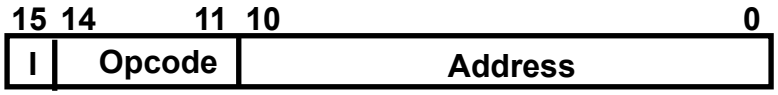
Computer Configuration





MACHINE INSTRUCTION FORMAT

Machine instruction format

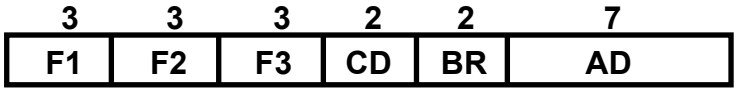


Sample machine instructions

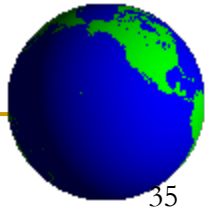
Symbol	OP-code	Description
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	if (AC < 0) then (PC ← EA)
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC

EA is the effective address

Microinstruction Format



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

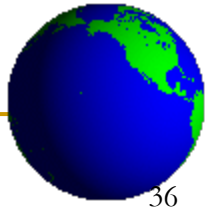


MICROINSTRUCTION FIELD DESCRIPTIONS - F1,F2,F3

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

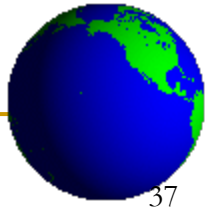
F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC'	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	



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

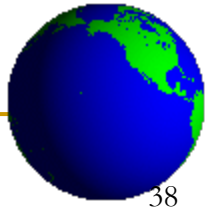
Micro-ops: consists of one, two, or three symbols separated by commas

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:

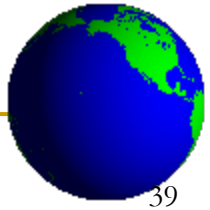
```
AR ← PC
DR ← M[AR], PC ← PC + 1
AR ← DR(0-10), CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0
```

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

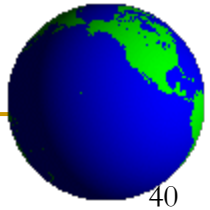


SYMBOLIC MICROPROGRAM

- 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

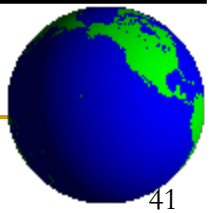
Label	Microops	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	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	



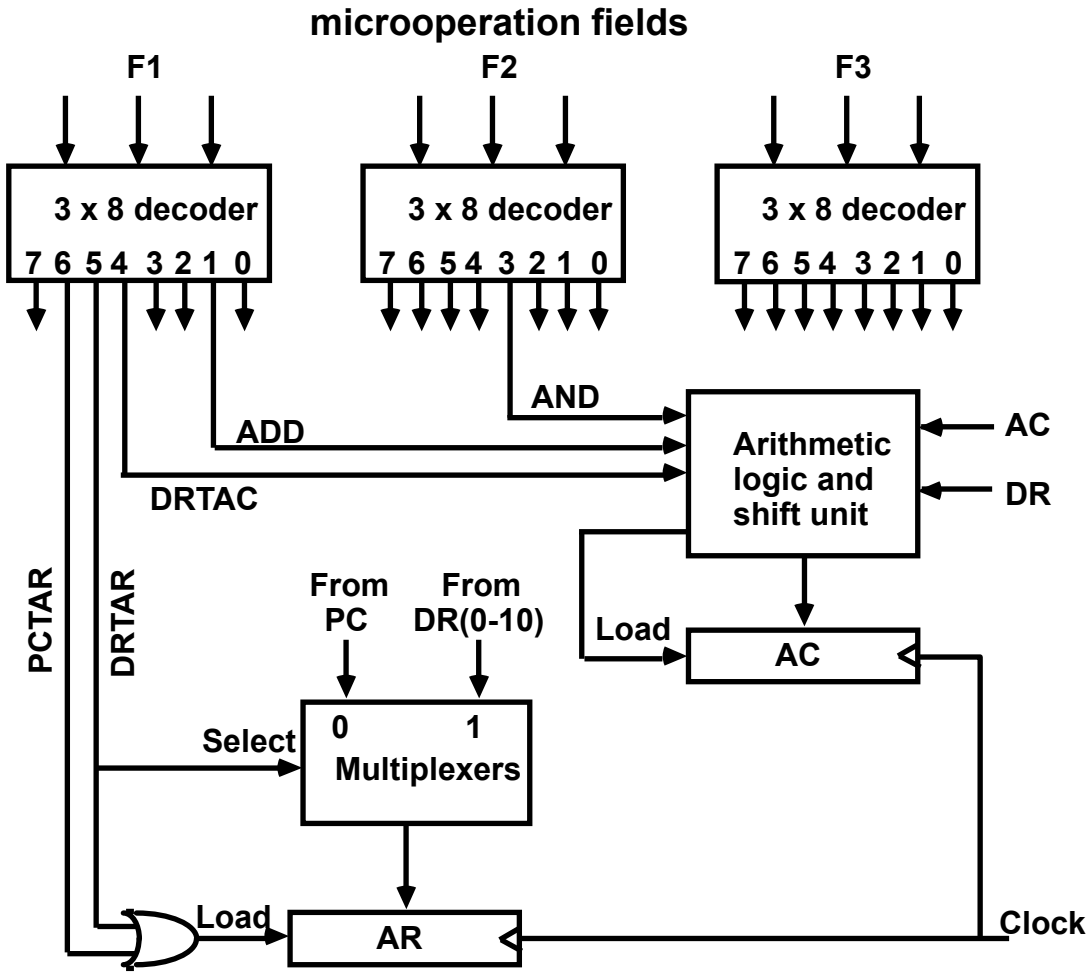
BINARY MICROPROGRAM

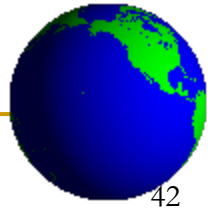
Micro Routine	Address		Binary Microinstruction					
	Decimal	Binary	F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	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

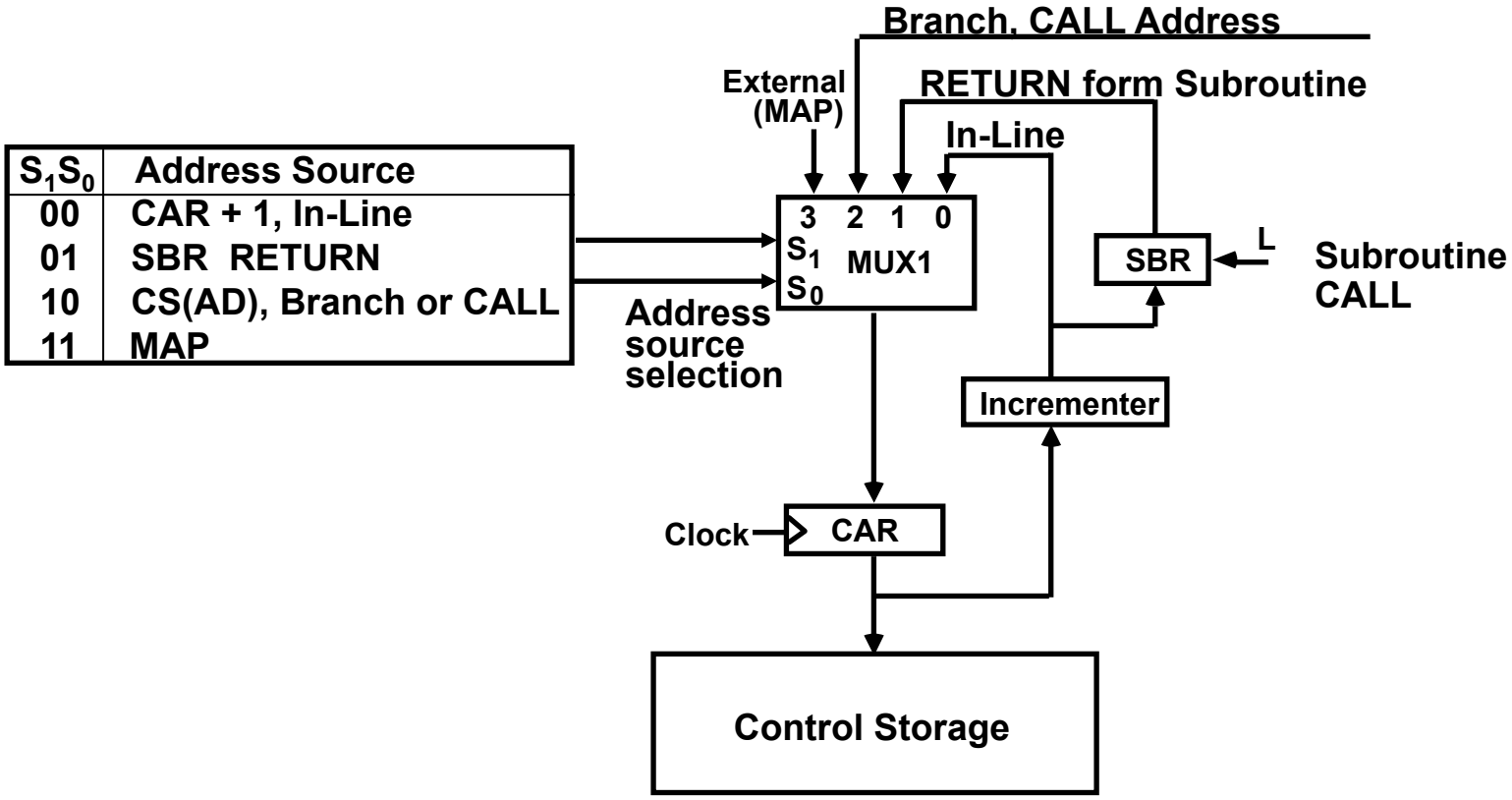


DESIGN OF CONTROL UNIT
- DECODING ALU CONTROL INFORMATION -





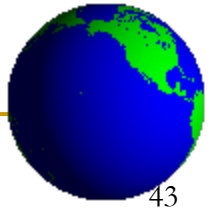
MICROPROGRAM SEQUENCER
- NEXT MICROINSTRUCTION ADDRESS LOGIC -



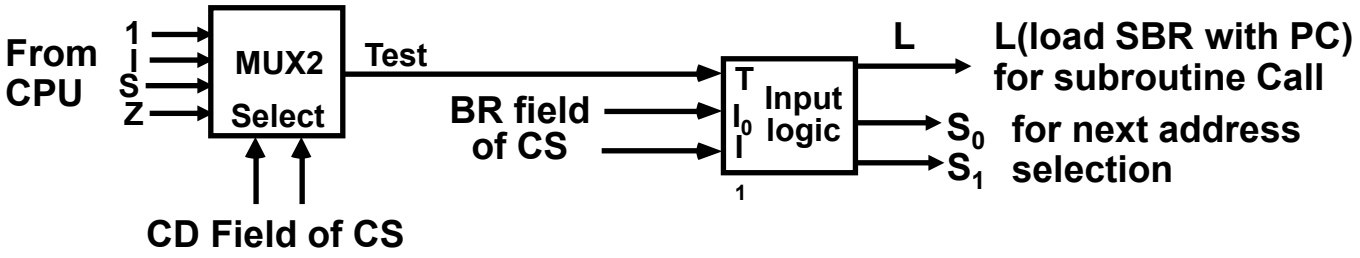
S_1S_0	Address Source
00	CAR + 1, In-Line
01	SBR RETURN
10	CS(AD), Branch or CALL
11	MAP

MUX-1 selects an address from one of four sources and routes it into a CAR

- **In-Line Sequencing → CAR + 1**
- **Branch, Subroutine Call → CS(AD)**
- **Return from Subroutine → Output of SBR**
- **New Machine instruction → MAP**



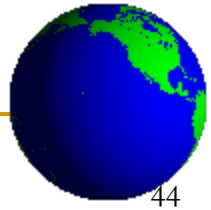
MICROPROGRAM SEQUENCER
- CONDITION AND BRANCH CONTROL -



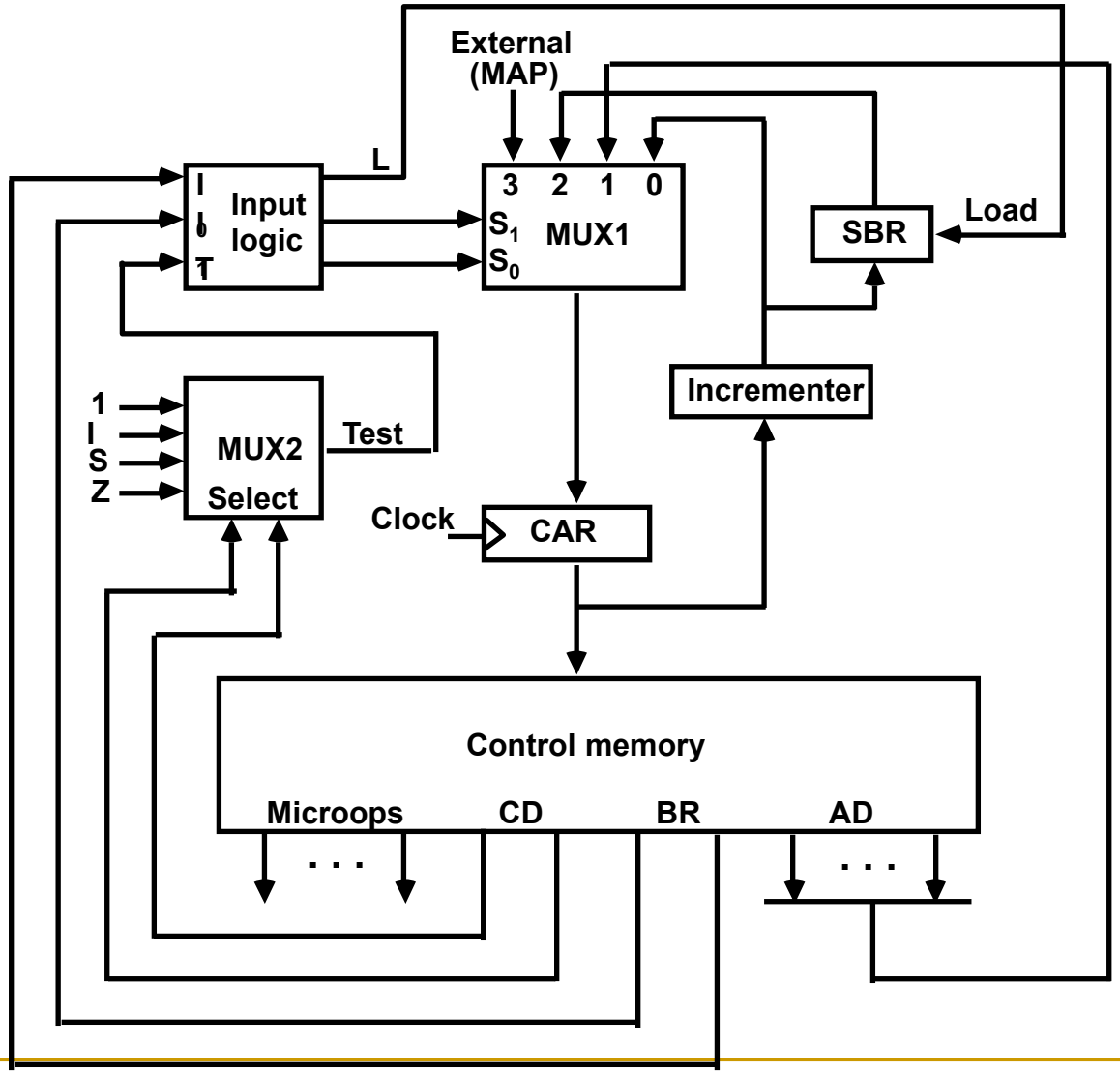
Input Logic

I_0I_1T	Meaning	Source of Address	S_1S_0	L
000	In-Line	CAR+1	00	0
001	JMP	CS(AD)	10	0
010	In-Line	CAR+1	00	0
011	CALL	CS(AD) and SBR <- CAR+1	10	1
10x	RET	SBR	01	0
11x	MAP	DR(11-14)	11	0

$$S_0 = I_0$$
$$S_1 = I_0I_1 + I_0'T$$
$$L = I_0'I_1T$$



MICROPROGRAM SEQUENCER



MICROINSTRUCTION FORMAT

Information in a Microinstruction

- **Control Information**
- **Sequencing Information**
- **Constant**

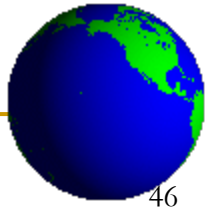
  Information which is useful when feeding into the system

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**



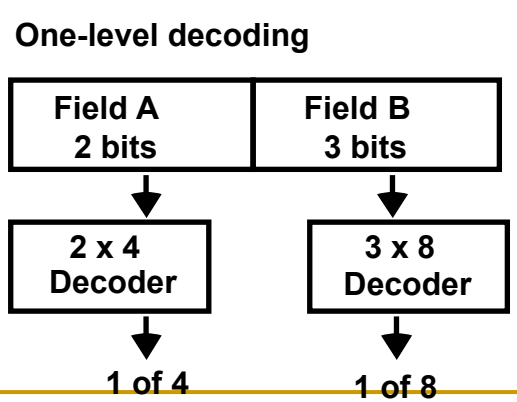
HORIZONTAL AND VERTICAL MICROINSTRUCTION FORMAT

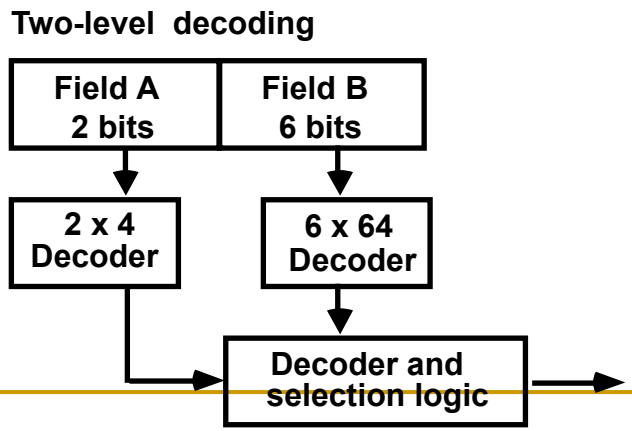
Horizontal Microinstructions

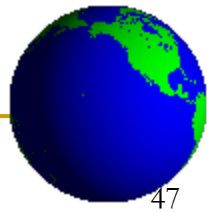
- **Each bit directly controls each micro-operation or each control point**
- ***Horizontal* implies a long microinstruction word**
- **Advantages: Can control a variety of components operating in parallel.**
  --> Advantage of efficient hardware utilization
- **Disadvantages: Control word bits are not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







NANOSTORAGE AND NANOINSTRUCTION

The decoder circuits in a vertical microprogram storage organization can be replaced by a ROM

=> Two levels of control storage

First level - *Control Storage*

Second level - *Nano Storage*

Two-level microprogram

First level

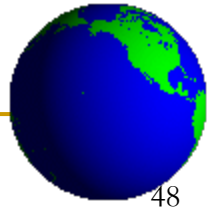
-*Vertical* format Microprogram

Second level

-*Horizontal* format Nanoprogram

- Interprets the microinstruction fields, thus converts a vertical microinstruction format into a horizontal nanoinstruction format.

Usually, the microprogram consists of a large number of short microinstructions, while the nanoprogram contains fewer words with longer nanoinstructions.



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)