Computer Architecture & Organization (6CS4-04)

Computer Science and Engineering Department

Vision of the Department

To become renowned Centre of excellence in computer science and engineering and make competent engineers & professionals with high ethical values prepared for lifelong learning.

Mission of the Department

- To impart outcome based education for emerging technologies in the field of computer science and engineering.
- To provide opportunities for interaction between academia and industry.
- To provide platform for lifelong learning by accepting the change in technologies.
- To develop aptitude of fulfilling social responsibilities.

Program Outcomes (PO):

- **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

- **Problem analysis**: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

- **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

- **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

- **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

- **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

- **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

- **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

- **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

- **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

- **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

- **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Educational Objectives (PEO):

PEO1: To provide students with the fundamentals of Engineering Sciences with more emphasis in computer science and engineering by way of analyzing and exploiting engineering challenges.

PEO2: To train students with good scientific and engineering knowledge so as to comprehend, analyze, design, and create novel products and solutions for the real life problems.

PEO3: To inculcate professional and ethical attitude, effective communication skills, teamwork skills, multidisciplinary approach, entrepreneurial thinking and an ability to relate engineering issues with social issues.

PEO4: To provide students with an academic environment aware of excellence,leadership, written ethical codes and guidelines, and the self-motivated life-long learning needed for a successful professional career.

PEO5: To prepare students to excel in Industry and Higher education by educating Students along with High moral values and Knowledge.

Program Specific Outcome (PSO):

PSO: Ability to interpret and analyze network specific and cyber security issues, automation in real word environment.
PSO2: Ability to Design and Develop Mobile and Web-based applications under realistic constraints.

Jaipur Engineering College and Research Centre

Computer Architecture and Organization (6CS4-04)

Session 2020-21

COURSE OUTCOMES: After Completion of the course, Students will be able to:

CO1: Identification of registers, micro-operations and basic computer organizations & design

CO2: Identification of computer architecture and processing

CO3: Introduction and applications of computer arithmetic operations

CO4 : Knowledge of computer Memory organization

Mapping Between CO and PO

CO-PO Mapping												
Computer Architecture 6CS4-04												
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
Co1: Ability to understand the functional units of the processor and various micro operations.	2	2	2	1	1	1	1	1	1	2	1	3
Co2: Analyze different architectural and organizational design issues that can affect the performance of a computer.	3	3	3	2	2	1	1	1	1	2	1	3
Co3. Examine the airthmetic problems and principles of computer design.	3	2	2	2	2	1	1	1	1	2	1	3
Co4. Describe and examine the concept of cache memory, Virtual memory and I/O organization.	3	2	2	2	1	1	1	1	1	1	1	3

Mapping Between CO and PSO:

CO-PSO Mapping		
Computer Architecture 6CS4-04		
	PSO1	**PSO2**
Co1: Ability to understand the functional units of the processor and various micro operations.	1	1
Co2: Analyze different architectural and organizational design issues that can affect the performance of a computer.	2	1
Co3. Examine the airthmetic problems and principles of computer design.	2	1
Co4. Describe and examine the concept of cache memory, Virtual memory and I/O organization.	1	1

RAJASTHAN TECHNICAL UNIVERSITY, KOTA

Syllabus

III Year-VI Semester: B.Tech. Computer Science and Engineering

6CS4-04: Computer Architecture and Organization

Credit: 3 **Max. Marks: 150(IA:30, ETE:120)**
3L+0T+0P **End Term Exam: 3 Hours**

SN	Contents	Hours
1	**Introduction:** Objective, scope and outcome of the course.	01
2	**Computer Data Representation:** Basic computer data types, Complements, Fixed point representation, Register Transfer and Micro-operations: Floating point representation, Register Transfer language, Register Transfer, Bus and Memory Transfers (Tree-State Bus Buffers, Memory Transfer), Arithmetic Micro-Operations, Logic Micro-Operations, Shift Micro-Operations, Arithmetic logical shift unit. Basic Computer Organization and DesignInstruction codes, Computer registers, computer instructions, Timing and Control, Instruction cycle, Memory-Reference Instructions, Input-output and interrupt, Complete computer description, Design of Basic computer, design of Accumulator Unit.	10
3	**Programming The Basic Computer:** Introduction, Machine Language, Assembly Language, assembler, Program loops, Programming Arithmetic and logic operations, subroutines, I-O Programming. Micro programmed Control: Control Memory, Address sequencing, Micro program Example, design of control Unit	7
4	**Central Processing Unit:** Introduction, General Register Organization, Stack Organization, Instruction format, Addressing Modes, data transfer and manipulation, Program Control, Reduced Instruction Set Computer (RISC)Pipeline And Vector Processing, Flynn's taxonomy, Parallel Processing, Pipelining, Arithmetic Pipeline, Instruction, Pipeline, RISC Pipeline, Vector Processing, Array Processors	8
5	**Computer Arithmetic:** Introduction, Addition and subtraction, Multiplication Algorithms (Booth Multiplication Algorithm), Division Algorithms, Floating Point Arithmetic operations, Decimal Arithmetic Unit. Input-Output Organization, Input-Output Interface, Asynchronous Data Transfer, Modes Of Transfer, Priority Interrupt, DMA, Input-Output Processor (IOP), CPUIOP Communication, Serial communication.	8
6	**Memory Organization:** Memory Hierarchy, Main Memory, Auxiliary Memory, Associative Memory, Cache Memory, Virtual Memory. Multipreocessors: Characteristics of Multiprocessors, Interconnection Structures, Inter-processor Arbitration, Inter-processor Communication and Synchronization, Cache Coherence, Shared Memory Multiprocessors.	8
	Total	42

Office of Dean Academic Affairs
Rajasthan Technical University, Kota

Jaipur Engineering College and Research Centre

Computer Architecture and Organization (6CS4-04)

Session 2020-21

Computer Architecture and Organization (6CS4-04)

LECTURE PLAN:

Unit No./ Total Lecture Reqd.	Topics	Lect. Reqd.	Lect. No.
Unit-I (10)	1. Objective, scope and outcome of the course. Basic computer data types, Complements, Fixed point representation, Register Transfer	1	1
	2. Micro-operations:		
	2a. Floating point representation, Register Transfer language	1	2
	2b. Register Transfer, Bus and Memory Transfers (Tree-State Bus Buffers, Memory Transfer)	1	3
	2c. Arithmetic Micro-Operations, Logic Micro-Operations	1	4
	2d. Shift Micro-Operations, Arithmetic logical shift unit	1	5
	3. Basic Computer Organization and Design Instruction codes	1	6
	4. Computer registers, computer instructions	1	7
	5. Timing and Control, Instruction cycle,	1	8
	6. Memory-Reference Instructions, Input-output and interrupt	1	9
	7. Complete computer description, Design of Basic computer, design of Accumulator Unit	1	10
BC-1	**Von Neuman Architecture**	1	11
Unit-II (7)	1. Introduction, machine language, Assembly language	1	12
	2. Assembler, Program loops	1	13
	3. Programming Arithmetic and logic operations	1	14
	4. Subroutines and I-O Programming	1	15
	5. Control Memory	1	16
	6. Address Sequencing	1	17
	7. Micro program Example	1	18
	8. Design of control unit	1	19
UNIT III (8)	1. Introduction General Register Organization	1	20
	2. Stack Organization, Instruction Format	1	21
	3. Addressing Modes, Data transfer and manipulation	1	22
	4. Program Control, Reduced Instruction set computer (RISC) pipeline	1	23
	5. Vector Processing and Flynn's taxonomy	1	24
	6. Parallel processing, pipeline	1	25
	7. Arithmetic pipeline, Instruction pipeline	1	26
	8. RISC pipeline, Vector Processing and Array Processing	1	27
Unit-IV (8)	1. Introduction, Addition and subtraction	1	28
	2. Multiplication Algorithms (Booth Multiplication Algorithm)	1	29
	3. Division Algorithms, Floating Point Arithmetic operations	1	30
	4. Decimal Arithmetic Unit, Input-Output Organization	1	31
	5. Input-Output Interface, Asynchronous Data Transfer	1	32
	6. Modes of Transfer, Priority Interrupt	1	33
	7. DMA, Input-Output Processor (IOP)	1	34

	8. CPU-IOP Communication, Serial communication	1	35
BC-2	**Interaction of computer with hardware**	1	36
Unit- V (8)	1. Memory Hierarchy	1	37
	2.Main Memory, Auxiliary memory	1	38
	3. Associative memory,	1	39
	4. Cache Memory, Virtual Memory	1	40
	5. Microprocessor: Characteristics of microprocessor	1	41
	6. Interconnection Structure	1	42
	7. Inter-processer Arbitration, Intercrosses communication and synchronization	1	43
	8. Cache Coherence, Shared Memory Multiprocessor	1	44
BC-3	**Different Operating System Architectures and their relation to Computer System Architectures**	1	45

This schedule is tentative and is subject to minimal changes during teaching.

PROGRAMMING THE BASIC COMPUTER

Introduction

Machine Language

Assembly Language

Assembler

Program Loops

Programming Arithmetic and Logic Operations

Subroutines

Input-Output Programming

INTRODUCTION

Those concerned with computer architecture should have a knowledge of both hardware and software because the two branches influence each other.

Instruction Set of the *Basic Computer*

Symbol	*Hexa code*	*Description*
AND	0 or 8	AND M to AC
ADD	1 or 9	Add M to AC, carry to E
LDA	2 or A	Load AC from M
STA	3 or B	Store AC in M
BUN	4 or C	Branch unconditionally to m
BSA	5 or D	Save return address in m and branch to m+1
ISZ	6 or E	Increment M and skip if zero
CLA	7800	Clear AC
CLE	7400	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right E and AC
CIL	7040	Circulate left E and AC
INC	7020	Increment AC, carry to E
SPA	7010	Skip if AC is positive
SNA	7008	Skip if AC is negative
SZA	7004	Skip if AC is zero
SZE	7002	Skip if E is zero
HLT	7001	Halt computer
INP	F800	Input information and clear flag
OUT	F400	Output information and clear flag
SKI	F200	Skip if input flag is on
SKO	F100	Skip if output flag is on
ION	F080	Turn interrupt on
IOF	F040	Turn interrupt off

m: effective address
M: memory word (operand) found at m

MACHINE LANGUAGE

Program

A list of instructions or statements for directing the computer to perform a required data processing task

Various types of programming languages

- Hierarchy of programming languages

- **Machine-language**
  - **- Binary code**
  - **- Octal or hexadecimal code**
- **Assembly-language (Assembler)**
  - **- Symbolic code**
- **High-level language (Compiler)**

COMPARISON OF PROGRAMMING LANGUAGES

• **Binary Program to Add Two Numbers**

Location	Instruction Code
0	0010 0000 0000 0100
1	0001 0000 0000 0101
10	0011 0000 0000 0110
11	0111 0000 0000 0001
100	0000 0000 0101 0011
101	1111 1111 1110 1001
110	0000 0000 0000 0000

• **Hexa program**

Location	Instruction
000	2004
001	1005
002	3006
003	7001
004	0053
005	FFE9
006	0000

• **Program with Symbolic OP-Code**

Location	Instruction		Comments
000	LDA	004	Load 1st operand into AC
001	ADD	005	Add 2nd operand to AC
002	STA	006	Store sum in location 006
003	HLT		Halt computer
004	0053		1st operand
005	FFE9		2nd operand (negative)
006	0000		Store sum here

• **Assembly-Language Program**

```
      ORG   0     /Origin of program is location 0
      LDA   A     /Load  operand from location A
      ADD   B     /Add  operand from location B
      STA   C     /Store sum in location C
      HLT         /Halt computer
A,    DEC   83    /Decimal operand
B,    DEC   -23   /Decimal operand
C,    DEC   0     /Sum stored in location C
      END         /End of symbolic program
```

• **Fortran Program**

```
INTEGER  A, B, C
DATA  A,83 / B,-23
C = A + B
END
```

ASSEMBLY LANGUAGE

Syntax of the BC assembly language

Each line is arranged in three columns called fields

***Label* field**

- **May be empty or may specify a symbolic address consists of up to 3 characters**
- **Terminated by a comma**

***Instruction* field**

- **Specifies a machine or a pseudo instruction**
- **May specify one of**
  - *** Memory reference instr. (MRI)**
    MRI consists of two or three symbols separated by spaces.
    ADD OPR (direct address MRI)
    ADD PTR I (indirect address MRI)
  - *** Register reference or input-output instr.**
    Non-MRI does not have an address part
  - *** Pseudo instr. with or without an operand**
    Symbolic address used in the instruction field must be defined somewhere as a label

***Comment* field**

- **May be empty or may include a comment**

PSEUDO-INSTRUCTIONS

ORG N

Hexadecimal number N is the memory loc.
for the instruction or operand listed in the following line

END

Denotes the end of symbolic program

DEC N

Signed decimal number N to be converted to the binary

HEX N

Hexadecimal number N to be converted to the binary

Example: Assembly language program to subtract two numbers

```
        ORG  100        / Origin of program is location 100
        LDA  SUB        / Load subtrahend to AC
        CMA             / Complement AC
        INC             / Increment AC
        ADD  MIN        / Add minuend to AC
        STA  DIF        / Store difference
        HLT             / Halt computer
MIN,    DEC  83         / Minuend
SUB,    DEC  -23        / Subtrahend
DIF,    HEX  0          / Difference stored here
        END             / End of symbolic program
```

TRANSLATION TO BINARY

Hexadecimal Code		*Symbolic Program*	
Location	*Content*		
			ORG 100
100	2107		LDA SUB
101	7200		CMA
102	7020		INC
103	1106		ADD MIN
104	3108		STA DIF
105	7001		HLT
106	0053	MIN,	DEC 83
107	FFE9	SUB,	DEC -23
108	0000	DIF,	HEX 0
			END

ASSEMBLER - FIRST PASS -

Assembler

Source Program - Symbolic Assembly Language Program

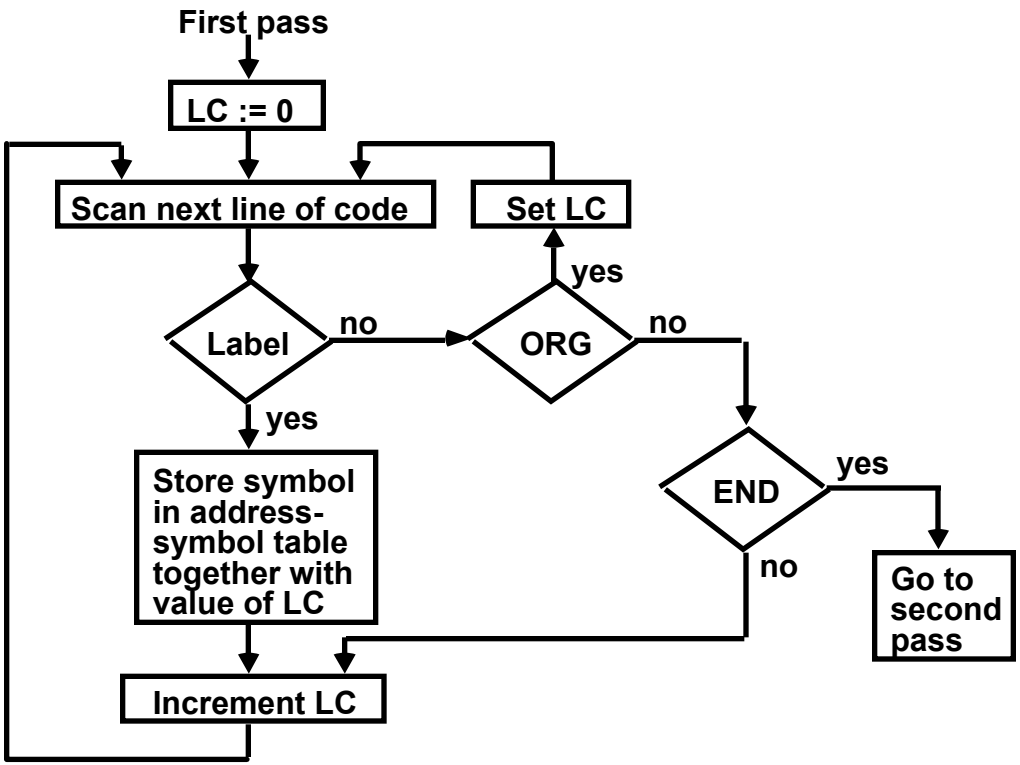
Object Program - Binary Machine Language Program

Two pass assembler

1st pass: generates a table that correlates all user defined (address) symbols with their binary equivalent value

2nd pass: binary translation

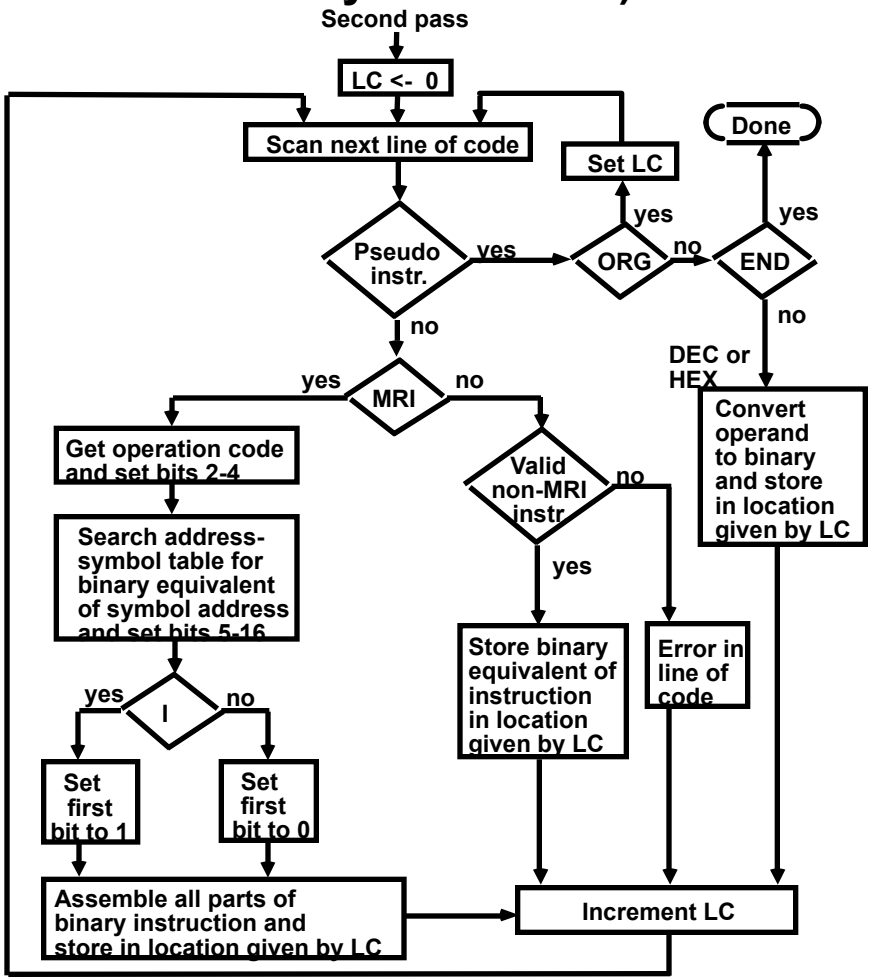
First pass



ASSEMBLER - SECOND PASS -

Second Pass

Machine instructions are translated by means of table-lookup procedures;
(1. Pseudo-Instruction Table, 2. MRI Table, 3. Non-MRI Table
4. Address Symbol Table)



PROGRAM LOOPS

Loop: A sequence of instructions that are executed many times, each with a different set of data

Fortran program to add 100 numbers:

```
      DIMENSION  A(100)
      INTEGER  SUM,  A
      SUM = 0
      DO  3  J = 1,  100
3     SUM = SUM + A(J)
```

Assembly-language program to add 100 numbers:

```
          ORG  100          / Origin of program is HEX 100
          LDA  ADS          / Load first address of operand
          STA  PTR          / Store in pointer
          LDA  NBR          / Load -100
          STA  CTR          / Store in counter
          CLA               / Clear AC
LOP,      ADD  PTR  I       / Add an operand to AC
          ISZ  PTR          / Increment pointer
          ISZ  CTR          / Increment counter
          BUN  LOP          / Repeat loop again
          STA  SUM          / Store sum
          HLT               / Halt
ADS,      HEX  150          / First address of operands
PTR,      HEX  0            / Reserved for a pointer
NBR,      DEC  -100         / Initial value for a counter
CTR,      HEX  0            / Reserved for a counter
SUM,      HEX  0            / Sum is stored here
          ORG  150          / Origin of operands is HEX 150
          DEC  75           / First operand
            .
            .
            .
          DEC  23           / Last operand
          END               / End of symbolic program
```

PROGRAMMING ARITHMETIC AND LOGIC OPERATIONS

Implementation of Arithmetic and Logic Operations

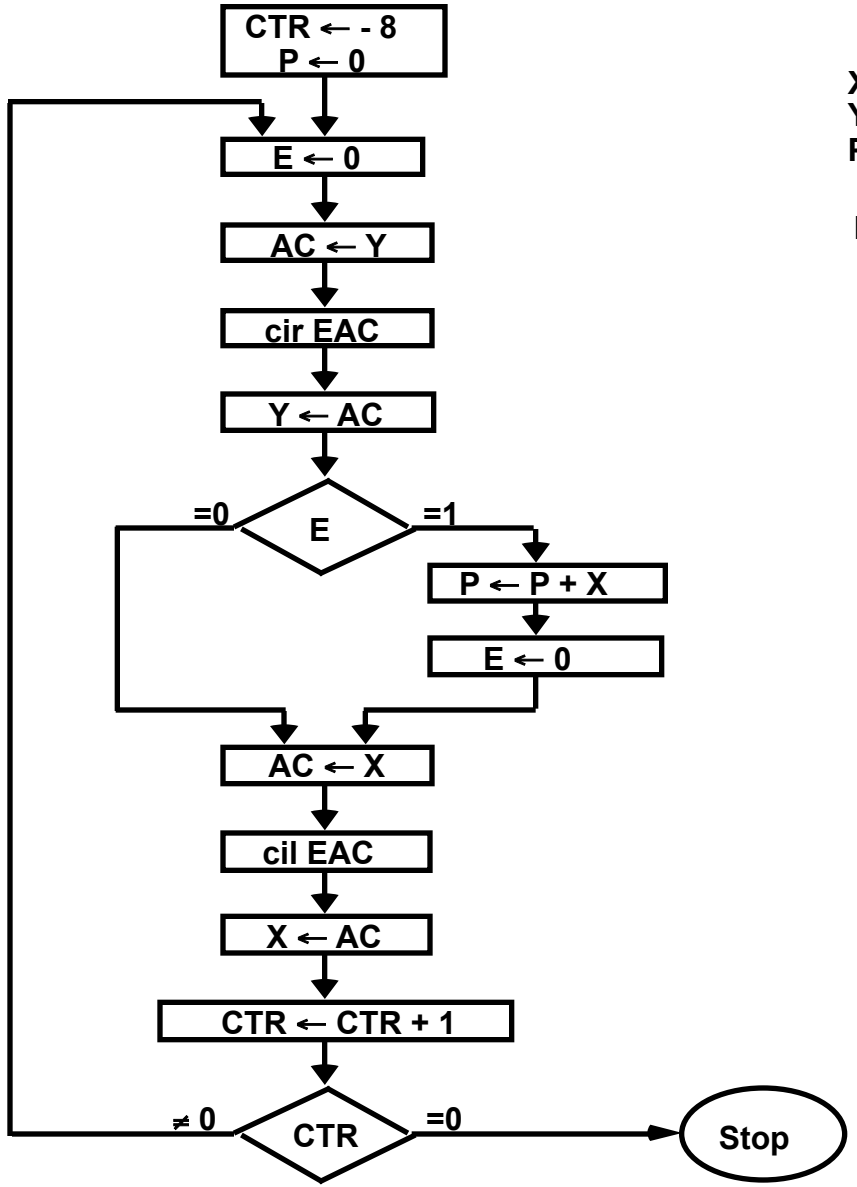
- **Software Implementation**
  - **Implementation of an operation with a program using machine instruction set**
  - **Usually when the operation is not included in the instruction set**

- **Hardware Implementation**
  - **Implementation of an operation in a computer with one machine instruction**

Software Implementation example:

* **Multiplication**
- **For simplicity, unsigned positive numbers**
- **8-bit numbers -> 16-bit product**

FLOWCHART OF A PROGRAM - Multiplication -



X holds the multiplicand
Y holds the multiplier
P holds the product

Example with four significant digits

		P
X =	0000 1111	
Y =	0000 1011	0000 0000
	0000 1111	0000 1111
	0001 1110	0010 1101
	0000 0000	0010 1101
	0111 1000	1010 0101
	1010 0101	

ASSEMBLY LANGUAGE PROGRAM - Multiplication -

```
        ORG  100
LOP,    CLE          / Clear E
        LDA  Y       / Load multiplier
        CIR          / Transfer multiplier bit to E
        STA  Y       / Store shifted multiplier
        SZE          / Check if bit is zero
        BUN  ONE     / Bit is one; goto ONE
        BUN  ZRO     / Bit is zero; goto ZRO
ONE,    LDA  X       / Load multiplicand
        ADD  P       / Add to partial product
        STA  P       / Store partial product
        CLE          / Clear E
ZRO,    LDA  X       / Load multiplicand
        CIL          / Shift left
        STA  X       / Store shifted multiplicand
        ISZ  CTR     / Increment counter
        BUN  LOP     / Counter not zero; repeat loop
        HLT          / Counter is zero; halt
CTR,    DEC  -8      / This location serves as a counter
X,      HEX  000F    / Multiplicand stored here
Y,      HEX  000B    / Multiplier stored here
P,      HEX  0       / Product formed here
        END
```

ASSEMBLY LANGUAGE PROGRAM
- Double Precision Addition -

```
LDA   AL      / Load A low
ADD   BL      / Add B low, carry in E
STA   CL      / Store in C low
CLA           / Clear AC
CIL           / Circulate to bring carry into AC(16)
ADD   AH      / Add A high and carry
ADD   BH      / Add B high
STA   CH      / Store in C high
HLT
```

ASSEMBLY LANGUAGE PROGRAM
- Logic and Shift Operations -

- **Logic operations**
  - **BC instructions : AND, CMA, CLA**
  - **Program for OR operation**

```
LDA   A      / Load 1st operand
CMA          / Complement to get A'
STA   TMP    / Store in a temporary location
LDA   B      / Load 2nd operand B
CMA          / Complement to get B'
AND   TMP    / AND with A' to get A' AND B'
CMA          / Complement again to get A OR B
```

- **Shift operations - BC has *Circular Shift* only**
  - **Logical shift-right operation**

```
CLE
CIR
```

  - **Logical shift-left operation**

```
CLE
CIL
```

  - **Arithmetic right-shift operation**

```
CLE     / Clear E to 0
SPA     / Skip if AC is positive
CME     / AC is negative
CIR     / Circulate E and AC
```

SUBROUTINES

Subroutine

- A set of common instructions that can be used in a program many times.
- Subroutine *linkage* : a procedure for branching to a subroutine and returning to the main program

Example

Loc.			
		ORG 100	/ Main program
100		LDA X	/ Load X
101		BSA SH4	/ Branch to subroutine
102		STA X	/ Store shifted number
103		LDA Y	/ Load Y
104		BSA SH4	/ Branch to subroutine again
105		STA Y	/ Store shifted number
106		HLT	
107	X,	HEX 1234	
108	Y,	HEX 4321	
			/ Subroutine to shift left 4 times
109	SH4,	HEX 0	/ Store return address here
10A		CIL	/ Circulate left once
10B		CIL	
10C		CIL	
10D		CIL	/ Circulate left fourth time
10E		AND MSK	/ Set AC(13-16) to zero
10F		BUN SH4 I	/ Return to main program
110	MSK,	HEX FFF0	/ Mask operand
		END	

SUBROUTINE PARAMETERS AND DATA LINKAGE

Linkage of Parameters and Data between the Main Program and a Subroutine

- **via Registers**
- **via Memory locations**
- **….**

Example: Subroutine performing *LOGICAL OR operation*; Need two parameters

Loc.			
200		ORG 200	
201		LDA X	/ Load 1st operand into AC
202		BSA OR	/ Branch to subroutine OR
203		HEX 3AF6	/ 2nd operand stored here
204		STA Y	/ Subroutine returns here
205	X,	HLT	
206	Y,	HEX 7B95	/ 1st operand stored here
207	OR,	HEX 0	/ Result stored here
208		CMA	/ Subroutine OR
209		STA TMP	/ Complement 1st operand
20A		LDA OR I	/ Store in temporary location
20B		CMA	/ Load 2nd operand
20C		AND TMP	/ Complement 2nd operand
20D		CMA	/ AND complemented 1st operand
20E		ISZ OR	/ Complement again to get OR
20F		BUN OR I	/ Increment return address
210	TMP,	HEX 0	/ Return to main program
		END	/ Temporary storage

SUBROUTINE - Moving a Block of Data -

```
                        / Main program
        BSA  MVE        / Branch to subroutine
        HEX  100        / 1st address of source data
        HEX  200        / 1st address of destination data
        DEC  -16        / Number of items to move
        HLT
MVE,    HEX  0          / Subroutine MVE
        LDA  MVE  I     / Bring address of source
        STA  PT1        / Store in 1st pointer
        ISZ  MVE        / Increment return address
        LDA  MVE  I     / Bring address of destination
        STA  PT2        / Store in 2nd pointer
        ISZ  MVE        / Increment return address
        LDA  MVE  I     / Bring number of items
        STA  CTR        / Store in counter
        ISZ  MVE        / Increment return address
LOP,    LDA  PT1  I     / Load source item
        STA  PT2  I     / Store in destination
        ISZ  PT1        / Increment source pointer
        ISZ  PT2        / Increment destination pointer
        ISZ  CTR        / Increment counter
        BUN  LOP        / Repeat 16 times
        BUN  MVE  I     / Return to main program
PT1,    --
PT2,    --
CTR,    --
```

- **Fortran subroutine**

```
      SUBROUTINE  MVE (SOURCE, DEST, N)
      DIMENSION  SOURCE(N), DEST(N)
      DO  20  I = 1, N
20    DEST(I) = SOURCE(I)
      RETURN
      END
```

INPUT OUTPUT PROGRAM

Program to Input one Character(Byte)

```
CIF,    SKI          / Check input flag
        BUN  CIF     / Flag=0, branch to check again
        INP          / Flag=1, input character
        OUT          / Display to ensure correctness
        STA  CHR     / Store character
        HLT
CHR,    --           / Store character here
```

Program to Output a Character

```
        LDA  CHR     / Load character into AC
COF,    SKO          / Check output flag
        BUN  COF     / Flag=0, branch to check again
        OUT          / Flag=1, output character
        HLT
CHR,    HEX  0057    / Character is "W"
```

CHARACTER MANIPULATION

Subroutine to Input 2 Characters and pack into a word

```
IN2,    --              / Subroutine entry
FST,    SKI
        BUN  FST
        INP             / Input 1st character
        OUT
        BSA  SH4        / Logical Shift left 4 bits
        BSA  SH4        / 4 more bits
SCD,    SKI
        BUN  SCD
        INP             / Input 2nd character
        OUT
        BUN  IN2  I     / Return
```

PROGRAM INTERRUPT

Tasks of Interrupt Service Routine

- **Save the Status of CPU**
  Contents of processor registers and Flags

- **Identify the source of Interrupt**
  Check which flag is set

- **Service the device whose flag is set**
  (Input Output Subroutine)

- **Restore contents of processor registers and flags**

- **Turn the interrupt facility on**

- **Return to the running program**
  Load PC of the interrupted program

INTERRUPT SERVICE ROUTINE

```
Loc.
  0     ZRO,    -           / Return address stored here
  1             BUN  SRV    / Branch to service routine
100             CLA         / Portion of running program
101             ION         / Turn on interrupt facility
102             LDA  X
103             ADD  Y      / Interrupt occurs here
104             STA  Z      / Program returns here after interrupt

                            / Interrupt service routine
200     SRV,    STA  SAC    / Store content of AC
                CIR         / Move E into AC(1)
                STA  SE     / Store content of E
                SKI         / Check input flag
                BUN  NXT    / Flag is off, check next flag
                INP         / Flag is on, input character
                OUT         / Print character
                STA  PT1  I / Store it in input buffer
                ISZ  PT1    / Increment input pointer
        NXT,    SKO         / Check output flag
                BUN  EXT    / Flag is off, exit
                LDA  PT2  I / Load character from output buffer
                OUT         / Output character
                ISZ  PT2    / Increment output pointer
        EXT,    LDA  SE     / Restore value of AC(1)
                CIL         / Shift it to E
                LDA  SAC    / Restore content of AC
                ION         / Turn interrupt on
                BUN  ZRO  I / Return to running program
        SAC,    -           / AC is stored here
        SE,     -           / E is stored here
        PT1,    -           / Pointer of input buffer
        PT2,    -           / Pointer of output buffer
```

MICROPROGRAMMED CONTROL

Contents:

- ✓ Control memory
- ✓ Address Sequencing
- ✓ Microprogram Example
- ✓ Design of Control Unit

Introduction:

- The function of the control unit in a digital computer is to initiate sequence of microoperations.
- Control unit can be implemented in two ways
  - Hardwired control
  - Microprogrammed control

Hardwired Control:

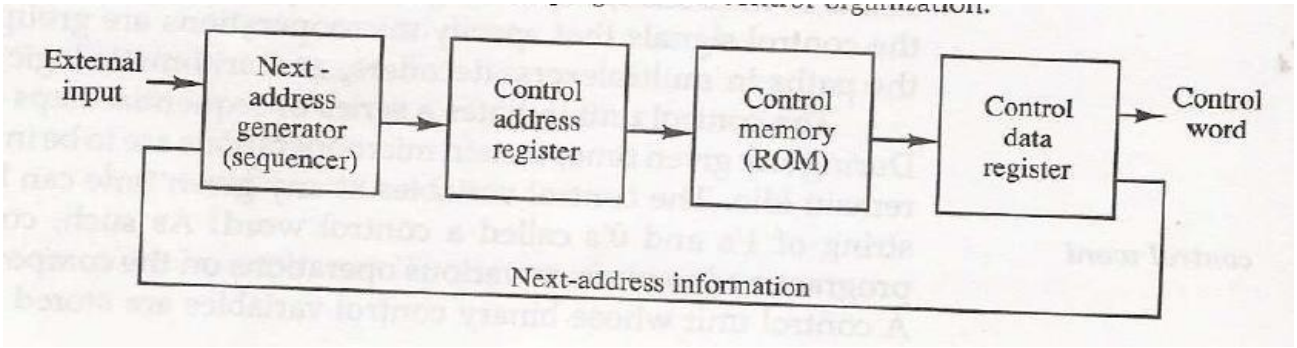
- ✓ When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be *hardwired.*
- ✓ The key characteristics are
  - High speed of operation
  - Expensive
  - Relatively complex
  - No flexibility of adding new instructions
- ✓ Examples of CPU with hardwired control unit are Intel 8085, Motorola 6802, Zilog 80, and any RISC CPUs.

Microprogrammed Control:

- ✓ Control information is stored in control memory.
- ✓ Control memory is programmed to initiate the required sequence of micro-operations.
- ✓ The key characteristics are
  - Speed of operation is low when compared with hardwired
  - Less complex
  - Less expensive
  - Flexibility to add new instructions
- ✓ Examples of CPU with microprogrammed control unit are Intel 8080, Motorola 68000 and any CISC CPUs.

1. Control Memory:

- The control function that specifies a microoperation is called as ***control variable.***
- When control variable is in one binary state, the corresponding microoperation is executed. For the other binary state the state of registers does not change.
- The active state of a control variable may be either 1 state or the 0 state, depending on the application.
- For bus-organized systems the control signals that specify microoperations are groups of bits that select the paths in multiplexers, decoders, and arithmetic logic units.
- ***Control Word:*** The control variables at any given time can be represented by a string of 1's and 0's called a control word.
- All control words can be programmed to perform various operations on the components of the system.
- ***Microprogram control unit:*** A control unit whose binary control variables are stored in memory is called a microprogram control unit.
- The control word in control memory contains within it a *microinstruction.*
- The microinstruction specifies one or more micro-operations for the system.
- A sequence of microinstructions constitutes a *microprogram*.
- The control unit consists of control memory used to store the microprogram.
- Control memory is a permanent i.e., read only memory (ROM).
- The general configuration of a micro-programmed control unit organization is shown as block diagram below.



- The control memory is ROM so all control information is permanently stored.
- The control memory address register (CAR) specifies the address of the microinstruction and the control data register (CDR) holds the microinstruction read from memory.
- The next address generator is sometimes called a microprogram sequencer. It is used to generate the next micro instruction address.
- The location of the next microinstruction may be the one next in sequence or it may be located somewhere else in the control memory.
- So it is necessary to use some bits of the present microinstruction to control the generation of the address of the microinstruction.
- Sometimes the next address may also be a function of external input conditions.
- The control data register holds the present microinstruction while next address is computed and read from memory. The data register is times called a *pipeline register.*

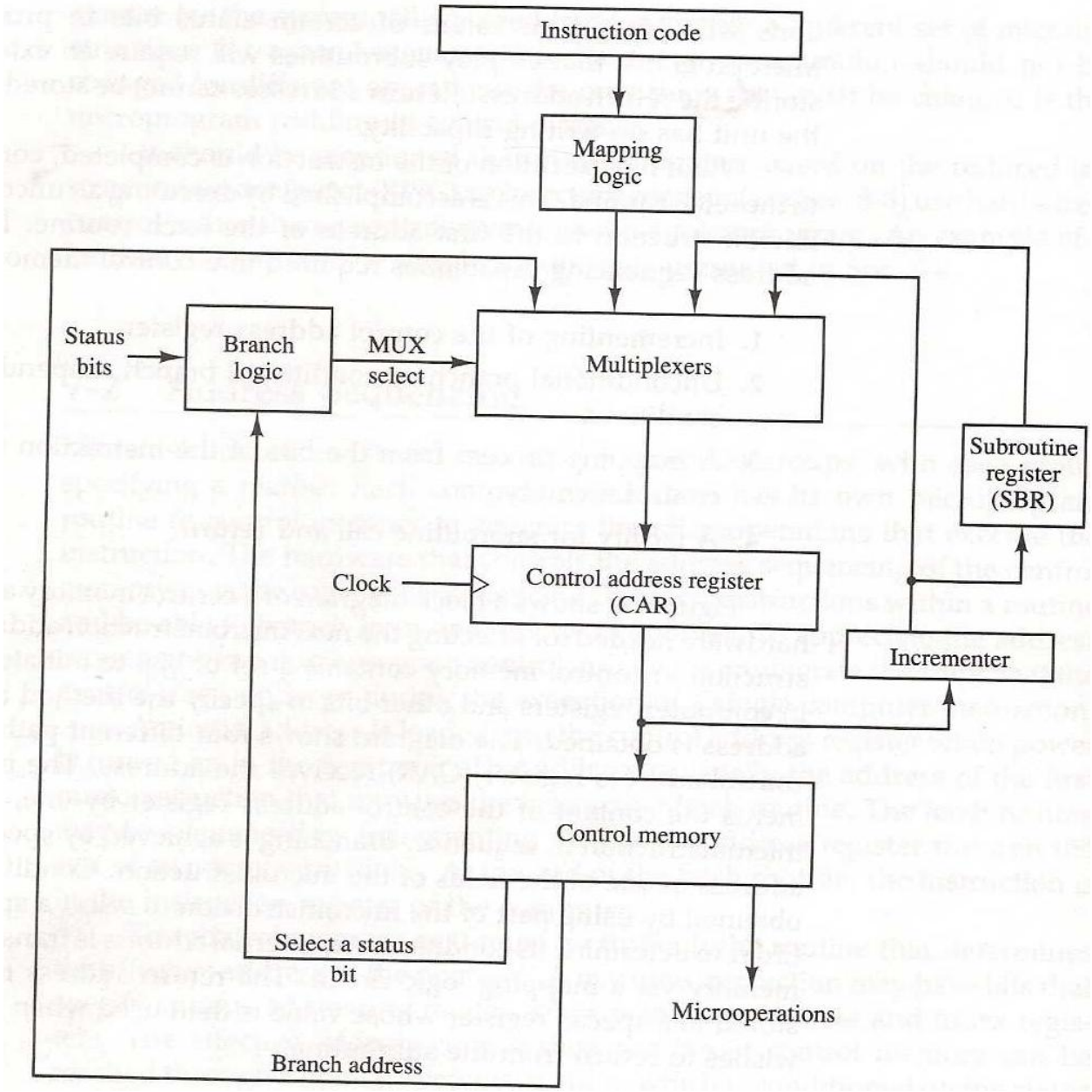
- A computer with a microprogrammed control unit will have two separate memories: a main memory and a control memory
- The microprogram consists of microinstructions that specify various internal control signals for execution of register microoperations
- These microinstructions generate the microoperations to:
    - fetch the instruction from main memory
    - evaluate the effective address
    - execute the operation
    - return control to the fetch phase for the next instruction

2. Address Sequencing:

- Microinstructions are stored in control memory in groups, with each group specifying a *routine.*
- Each computer instruction has its own microprogram routine to generate the microoperations.
- 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 the control must undergo during the execution of a single computer instruction:
    - Load an initial address into the CAR when power is turned on in the computer. This address is usually the address of the first microinstruction that activates the instruction fetch routine – IR holds instruction
    - The control memory then goes through the routine to determine the effective address of the operand – AR holds operand address
    - The next step is to generate the microoperations that execute the instruction by considering the opcode and applying a *mapping process.*
        - *The transformation of the instruction code bits to an address in control memory where the routine of instruction located is referred to as mapping process.*
    - After execution, control must return to the fetch routine by executing an unconditional branch
- In brief 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.

- The below figure shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address.



- The microinstruction in control memory contains a set of bits to initiate microoperations in computer registers and other bits to specify the method by which the next address is obtained.
- In the figure four different paths form which the control address register (CAR) receives the address.
  - The incrementer increments the content of the control register address register by one, 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 part of the microinstruction to select a specific status bit in order to determine its condition.
  - An external address is transferred into control memory via a mapping logic circuit.
  - The return address for a subroutine is stored in a special register, that value is used when the micoprogram wishes to return from the subroutine.

Conditional Branching:

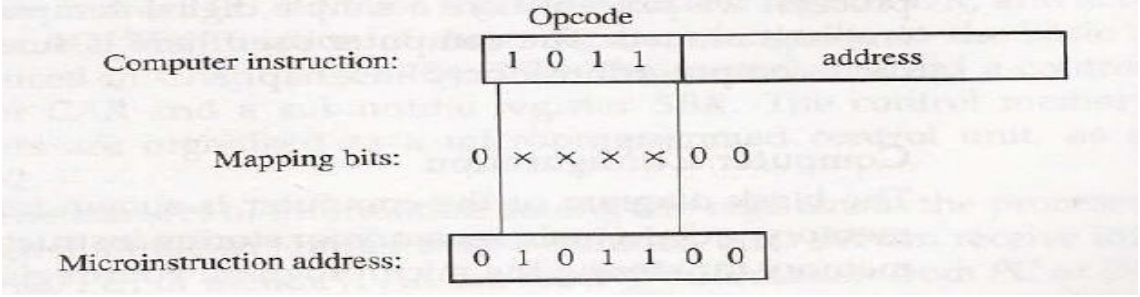
- Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition.
- The status conditions are special bits in the system that provide parameter information such as the carry-out of an adder, the sign bit of a number, the mode bits of an instruction, and i/o status conditions.
- The status bits, together with the field in the microinstruction that specifies a branch address, control the branch logic.

- The branch logic tests the condition, if met then branches, otherwise, increments the CAR.
- If there are 8 status bit conditions, then 3 bits in the microinstruction are used to specify the condition and provide the selection variables for the multiplexer.
- For unconditional branching, fix the value of one status bit to be one load the branch address from control memory into the CAR.

Mapping of Instruction:

- A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine is located.
- The status bits for this type of branch are the bits in the opcode.
- Assume an opcode of four bits and a control memory of 128 locations. The mapping process converts the 4-bit opcode to a 7-bit address for control memory shown in below figure.

Figure 7-3 Mapping from instruction code to microinstruction address.



- Mapping consists of placing a 0 in the most significant bit of the address, transferring the four operation code bits, and clearing the two least significant bits of the control address register.
- This provides for each computer instruction a microprogram routine with a capacity of four microinstructions.

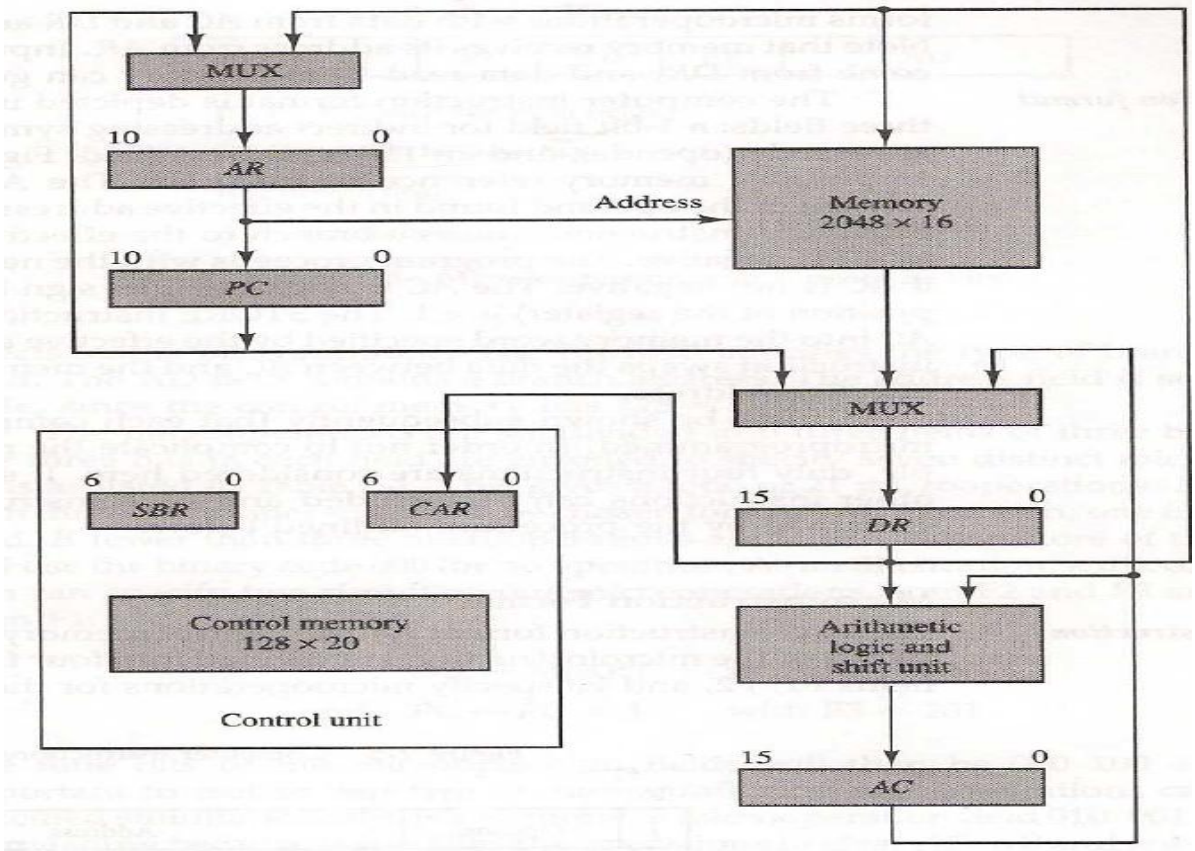
Subroutines:

- Subroutines are programs that are used by other routines to accomplish a particular task and can be called from any point within the main body of the microprogram.
- Frequently many microprograms contain identical section of code.
- Microinstructions can be saved by employing subroutines that use common sections of microcode.
- 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.
- A subroutine register is used as the source and destination for the addresses

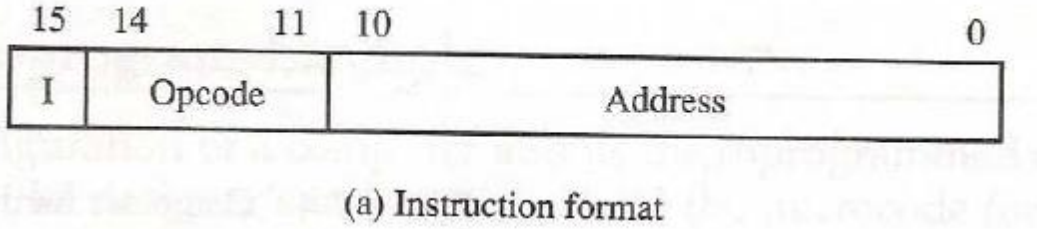
3. Microprogram Example:

- The process of code generation for the control memory is called *microprogramming.*
- The block diagram of the computer configuration is shown in below figure.
- Two memory units:
  - Main memory – stores instructions and data
  - Control memory – stores microprogram
- Four processor registers
  - Program counter – PC
  - Address register – AR
  - Data register – DR
  - Accumulator register - AC
- Two control unit registers
  - Control address register – CAR
  - Subroutine register – SBR

- Transfer of information among registers in the processor is through MUXs rather than a bus.



- The computer instruction format is shown in below figure.



(a) Instruction format

- Three fields for an instruction:
  - 1-bit field for indirect addressing
  - 4-bit opcode
  - 11-bit address field

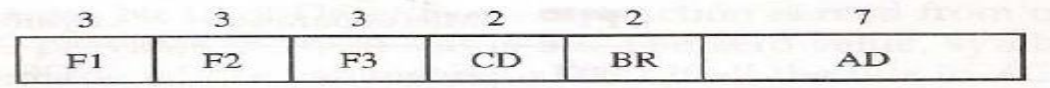
- The example will only consider the following 4 of the possible 16 memory instructions

Symbol	Opcode	Description
ADD	0000	$AC \leftarrow AC + M\,[EA]$
BRANCH	0001	If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M\,[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA],\ M[EA] \leftarrow AC$

EA is the effective address

(b) Four computer instructions

- The microinstruction format for the control memory is shown in below figure.



F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

Figure 7-6 Microinstruction code format (20 bits).

- The microinstruction format is composed of 20 bits with four parts to it
  - Three fields F1, F2, and F3 specify microoperations for the computer [3 bits each]
  - The CD field selects status bit conditions [2 bits]
  - The BR field specifies the type of branch to be used [2 bits]
  - The AD field contains a branch address [7 bits]
- Each of the three microoperation fields can specify one of seven possibilities.
- No more than three microoperations can be chosen for a microinstruction.
- If fewer than three are needed, the code 000 = NOP.
- The three bits in each field are encoded to specify seven distinct microoperations listed in below table.

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{–}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

- Five letters to specify a transfer-type microoperation
  - First two designate the source register
  - Third is a 'T'
  - Last two designate the destination register

  AC ← DR F1 = 100 = DRTAC
- The condition field (CD) is two bits to specify four status bit conditions shown below

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

- The branch field (BR) consists of two bits and is used with the address field to choose the address of the next microinstruction.

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD$, $SBR \leftarrow CAR + 1$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2\text{–}5) \leftarrow DR(11\text{–}14)$, $CAR(0,1,6) \leftarrow 0$

- Each line of an assembly language microprogram defines a symbolic microinstruction and is divided into five parts
    1. The label field may be empty or it may specify a symbolic address. Terminate with a colon (:).
    2. The microoperations field consists of 1-3 symbols, separated by commas. Only one symbol from each field. If NOP, then translated to 9 zeros
    3. The condition field specifies one of the four conditions
    4. The branch field has one of the four branch symbols
    5. The address field has three formats
        a. A symbolic address – must also be a label
        b. The symbol NEXT to designate the next address in sequence
        c. Empty if the branch field is RET or MAP and is converted to 7 zeros
- The symbol ORG defines the first address of a microprogram routine.
- ORG 64 – places first microinstruction at control memory 1000000.

Fetch Routine:

- The control memory has 128 locations, each one is 20 bits.
- The first 64 locations are occupied by the routines for the 16 instructions, addresses 0-63.
- Can start the fetch routine at address 64.
- The fetch routine requires the following three microinstructions (locations 64-66).
- The microinstructions needed for fetch routine are:

$AR \leftarrow PC$

$DR \leftarrow M[AR]$, $PC \leftarrow PC + 1$

$AR \leftarrow DR(0\text{–}10)$, $CAR(2\text{–}5) \leftarrow DR(11\text{–}14)$, $CAR(0,1,6) \leftarrow 0$

- It's Symbolic microprogram:

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

- It's Binary microprogram:

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

4. Design of control Unit:

- The control memory out of each subfield must be decoded to provide the distinct microoperations.
- The outputs of the decoders are connected to the appropriate inputs in the processor unit.
- The below figure shows the three decoders and some of the connections that must be made from their outputs.

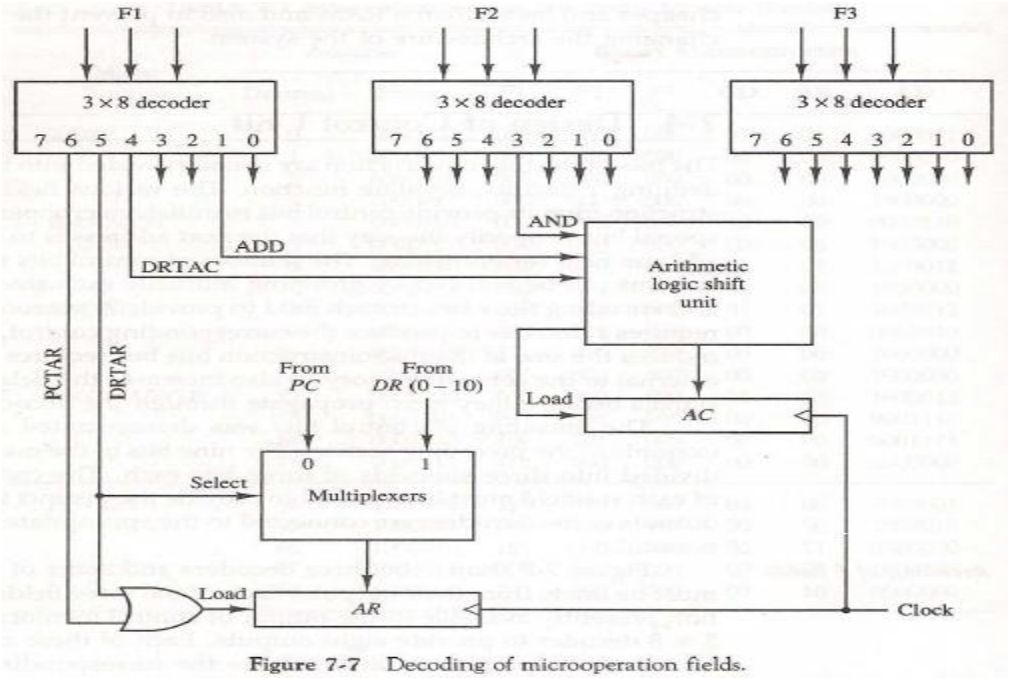


Figure 7-7 Decoding of microoperation fields.

- The three fields of the microinstruction in the output of control memory are decoded with a 3x8 decoder to provide eight outputs.
- Each of the output must be connected to proper circuit to initiate the corresponding microoperation as specified in previous topic.
- When F1 = 101 (binary 5), the next pulse transition transfers the content of DR (0-10) to AR.
- Similarly, when F1= 110 (binary 6) there is a transfer from PC to AR (symbolized by PCTAR). As shown in Fig, outputs 5 and 6 of decoder *F1* are connected to the load input of AR so that when either one of these outputs is active, information from the multiplexers is transferred to AR.
- The multiplexers select the information from DR when output 5 is active and from PC when output 5 is inactive.
- The transfer into AR occurs with a clock transition only when output 5 or output 6 of the decoder is active.
- For the arithmetic logic shift unit the control signals are instead of coming from the logical gates, now these inputs will now come from the outputs of AND, ADD and DRTAC respectively.

Microprogram Sequencer:

- The basic components of a microprogrammed control unit are the control memory and the circuits that select the next address.
- The address selection part is called a microprogram sequencer.
- The purpose of a microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.
- The next-address logic of the sequencer determines the specific address source to be loaded into the control address register.
- The block diagram of the microprogram sequencer is shown in below figure.
- The control memory is included in the diagram to show the interaction between the sequencer and the memory attached to it.
- There are two multiplexers in the circuit.
    - The first multiplexer selects an address from one of four sources and routes it into control address register *CAR.*
    - The second multiplexer tests the value of a selected status bit and the result of the test is applied to an input logic circuit.
- The output from CAR provides the address for the control memory.

- The content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine register *SBR.*
- The other three inputs to multiplexer come from
  - The address field of the present microinstruction
  - From the out of SBR
  - From an external source that maps the instruction
- The CD (condition) field of the microinstruction selects one of the status bits in the second multiplexer.
- If the bit selected is equal to 1, the T variable is equal to 1; otherwise, it is equal to 0.
- The *T* value together with two bits from the BR (branch) field goes to an input logic circuit.
- The input logic in a particular sequencer will determine the type of operations that are available in the unit.

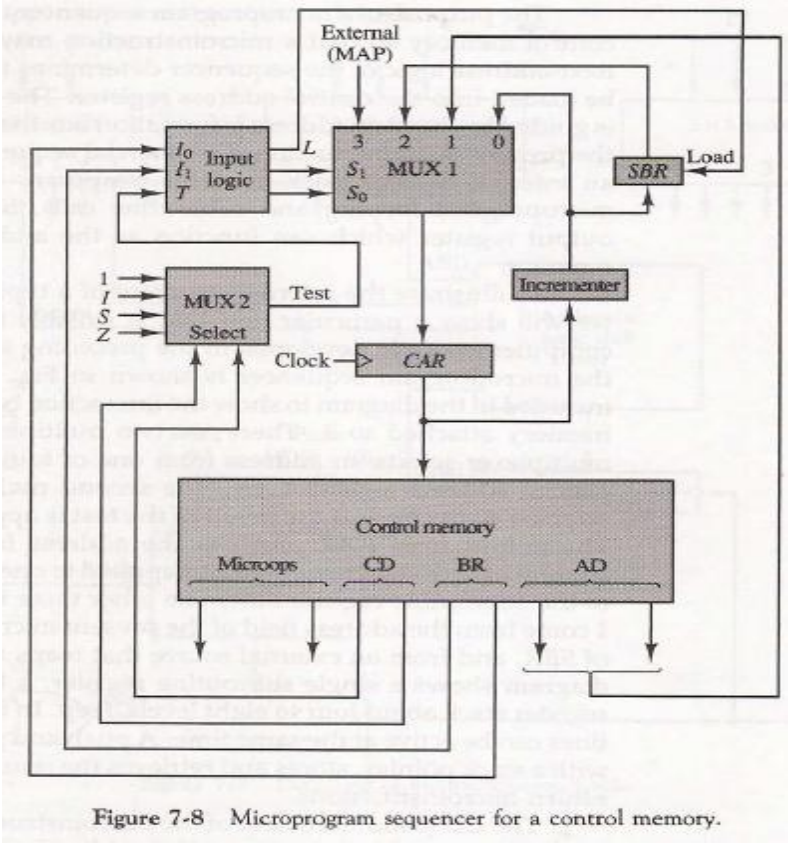


Figure 7-8 Microprogram sequencer for a control memory.

- The input logic circuit in above figure has three inputs I_0, I_1, and T, and three outputs, S_0, S_1, and L.
- Variables S_0 and S_1 select one of the source addresses for CAR. Variable L enables the load input in SBR.
- The binary values of the selection variables determine the path in the multiplexer.
- For example, with S_1S_0 = 10, multiplexer input number 2 is selected and establishes transfer path from SBR to CAR.
- The truth table for the input logic circuit is shown in Table below.

BR Field	Input			MUX 1		Load *SBR*
	I_1	I_0	T	S_1	S_0	L
0 0	0	0	0	0	0	0
0 0	0	0	1	0	1	0
0 1	0	1	0	0	0	0
0 1	0	1	1	0	1	1
1 0	1	0	×	1	0	0
1 1	1	1	×	1	1	0

- Inputs I_1 and I_0 are identical to the bit values in the BR field.
- The bit values for S_1 and S_0 are determined from the stated function and the path in the multiplexer that establishes the required transfer.
- The subroutine register is loaded with the incremented value of CAR during a call microinstruction (BR = 01) provided that the status bit condition is satisfied (T = 1).
- The truth table can be used to obtain the simplified Boolean functions for the input logic circuit:

$$S_1 = I_1$$
$$S_0 = I_1 I_0 + I'_1 T$$
$$L = I'_1 T I_0$$