UNIT-III

Syllabus:

Central Processing Unit: General Register Organization, STACK Organization, Instruction formats, Addressing modes, Data Transfer and manipulation, Program control, Reduced Instruction Set Computer

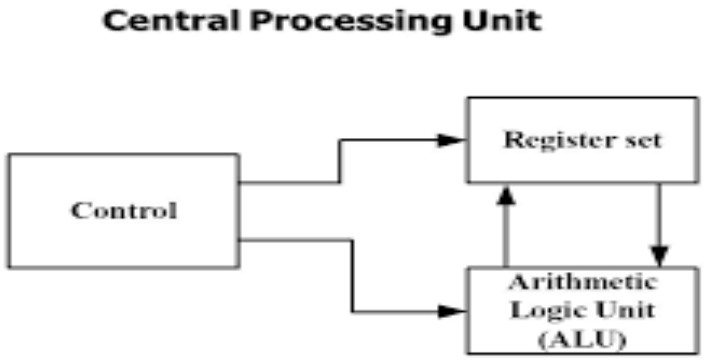
Micro Programmed Control: Control memory, Address sequencing, micro program example, design of control unit.

Central Processing Unit

Introduction

The part of the computer that performs the bulk of data processing operations is called the central processing unit.CPU is the main brain of the computer that accepts data, performs operations on the data and sends out the result. Information from an input device or from the computer's memory is communicated via the bus to the Central Processing Unit (CPU), which is the part of the computer that translates commands and runs programs.

CPU consists of Arithmetic Logic Unit (ALU) and Control Unit (CU). In addition, CPU also has a set of registers which are temporary storage areas for holding data, and instructions. ALU performs the arithmetic and logic operations on the data that is made available to it. CU is responsible for organizing the processing of data and instructions. CU controls and coordinates the activity of the other units of computer. CPU uses the registers to store the data, instructions during processing.



Major components of CPU

General Register Organizations

A set of flip-flops forms a register. A register is a unique high-speed storage area in the CPU. They include combinational circuits that implement data processing. The information is always defined in a register before processing. The registers speed up the implementation of programs.

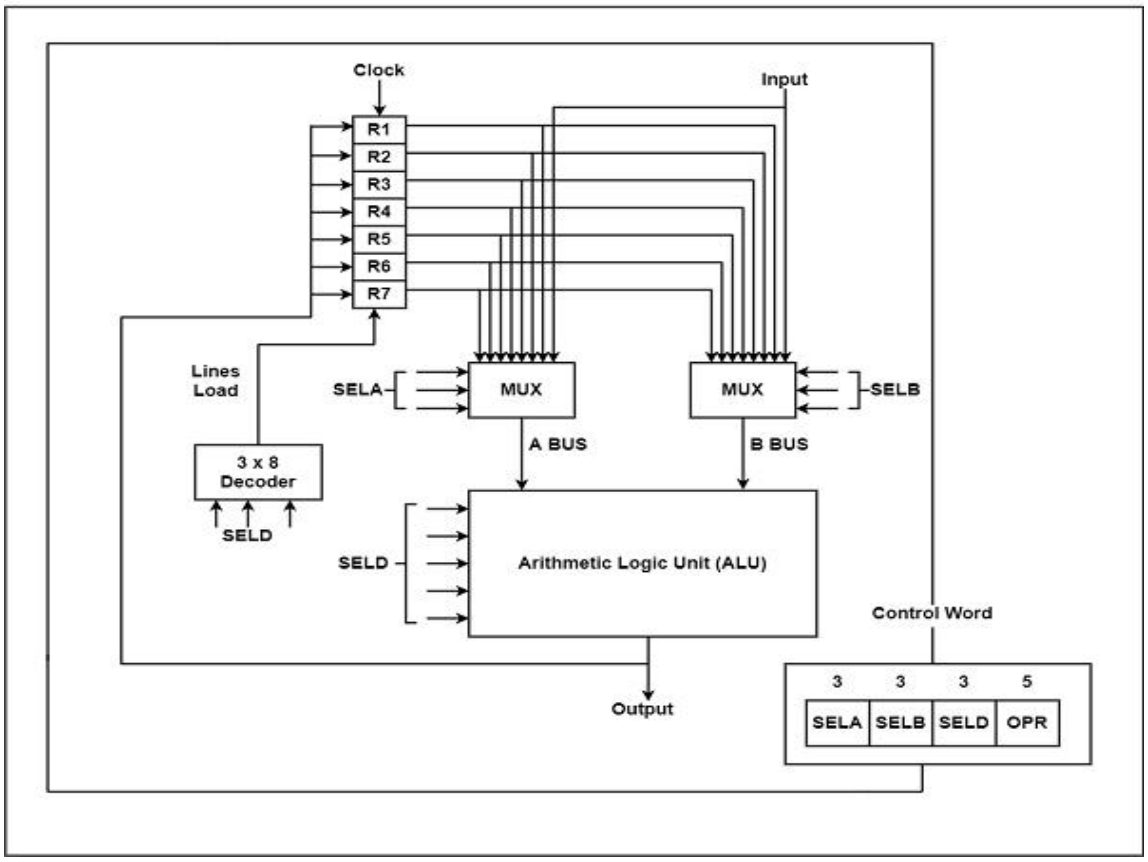
Registers implement two important functions in the CPU operation are as follows −

- It can support a temporary storage location for data. This supports the directly implementing programs to have fast access to the data if required.
- It can save the status of the CPU and data about the directly implementing program.

Example − Address of the next program instruction, signals get from the external devices and error messages, and including different data is saved in the registers.

If a CPU includes some registers, therefore a common bus can link these registers. A general organization of seven CPU registers is displayed in the figure.

General Organization of Registers



The CPU bus system is managed by the control unit. The control unit explicit the data flow through the ALU by choosing the function of the ALU and components of the system.

Consider **R1 ← R2 -R3**, the following are the functions implemented within the CPU −

MUX A Selector (SELA) − It can place R2 into bus A.

MUX B Selector (SELB) − It can place R3 into bus B.

ALU Operation Selector (OPR) − It can select the arithmetic addition (ADD).

Decoder Destination Selector (SELD) − It can transfers the result into R1.

The multiplexers of 3-state gates are performed with the buses. The state of 14 binary selection inputs determines the control word.

The 14-bit **control word** defines a micro-operation.

The encoding of register selection fields is specified in the table.

Encoding of Register Selection Field

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

There are several micro-operations are implemented by the ALU. Few of the operations implemented by the ALU are displayed in the table.

Encoding of ALU Operations

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Subtract A - B	SUB
00110	Decrement A	DECA
01000	ADD A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

There are some ALU micro-operations are shown in the table.

ALU Micro-Operations

Micro-operation	SELA	SELB	SELD	OPR	Control Word			
R1 ← R2 – R3	R2	R3	R1	SUB	010	011	001	00101
R4 ← R4 v R5	R4	R5	R4	OR	100	101	100	01010
R6 ← R6 + R1	-	R6	R1	INCA	110	000	110	00001
R7 ← R1	R1	-	R7	TSFA	001	000	111	00000
Output ← R2	R2	–	None	TSFA	010	000	000	00000
Output ← Input	Input	-	None	TSFA	000	000	000	00000
R4 ← shl R4	R4	-	R4	SHLA	100	000	100	11000
R5 ← 0	R5	R5	R5	XOR	101	101	101	01100

Stack Organization

A stack is an ordered linear list in which all insertions and deletions are made at one end, called top. It uses Last In First Out (LIFO) access method which is the most popular access method in most of the CPU. A register is used to store the address of the topmost element of the stack which is known as Stack pointer (SP) because its value always points at the top item in the stack. In this organization, ALU operations are performed on stack data.

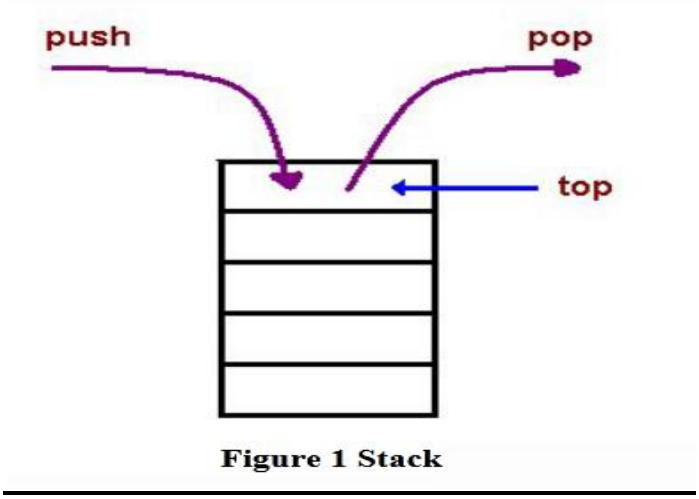


Figure 1 Stack

The main two operations that are performed on the operands of the stack are **Push and Pop.** These two operations are performed from one end only.

- **Push operation** - The operation of inserting an item onto a stack is called push operation.
- **Pop operation**- The operation of deleting an item onto a stack is called pop operation.

Applications

- Evaluation of mathematical expressions using Reverse Polish Notation
- To reverse a word. A given word is pushed to stack - letter by letter - and then popped out letters from the stack. There are two types of stack organization which are used in the computer hardware.
- **Register stack-** It is built using register
- **Memory stack-** It is a logical part of memory allocated as stack. The logically partitioned part of RAM is used to implement stack.

Register stack-

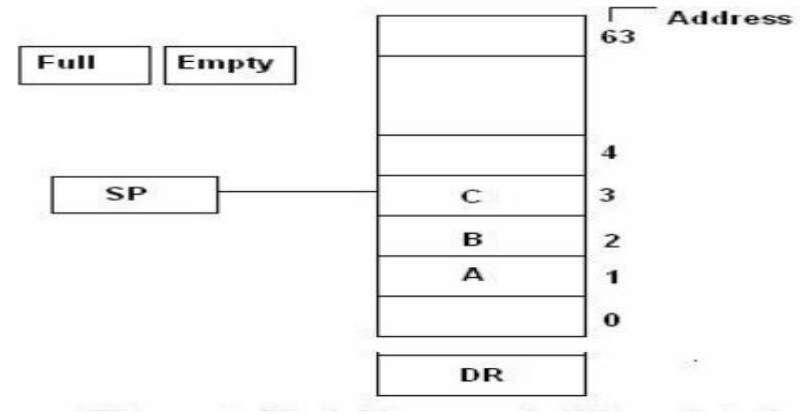


Figure 2: a. 64 word Register stack

There are 64 registers used to make a register stack. The numbers 0, 1, 2, 3,..... Upto 63 denote the address of different registers. SP is a pointer which points to the top of the stack i.e. it currently points to the item at the top. The two more registers called FLAG and EMPTY are used. The Full and EMPTY registers are made up of flip-flops. It indicates whether the stack is full or not.

If FULlL = 1, then EMPTY = 0 it means that stack is full.

If FULlL = 0, then EMPTY= 1 it means that stack is empty.

Similarly the EMPTY flag states whether the stack is empty or not because while retrieving the data from the stack it may happen that stack is empty.

DR is the data register through which data is transferred to and fro from the stack.

Operations

● **Push Operation**

SP ← SP + 1

//stack pointer is incremented by one

M[SP] ← DR

// the content of the data register is stored at the current memory pointed by the stack pointer.

IF (SP = 0) then (FULL ←1)

(EMPTY ←0)

// The values of flag registers are set.

● **Pop operation**

DR ←M[SP]

// the content of current memory pointed by the stack pointer is stored to data register

SP = SP – 1 //stack pointer is decremented by one

If (SP = 0) then (EMPTY ←1) FULL ←0

Since the address of 64 registers are numbered from 0, 1, 2, 3,..... Upto 63. The binary value of 63 is 111111 which is of 6- bits. When stack pointer is incremented after 63, it becomes 64 corresponding to the binary value 1000000 which is of 7 - bits. But in this case the address of each register is of 6- bits, hence 1 is discarded from the binary value of 64. The stack pointer points to the 000000 address register. It implies that the stack is full.

Zero address instructions are used in registers stack organization. The address that does not contain the address of the operands.

Example: (a + b) * (c + d)

Following are the instructions to execute it

Push a Push b

Add. // Zero address instruction

Push c Push d

Add. // Zero address instruction

Mul. // Zero address instruction

Memory stack - The RAM is divided into three logical parts.

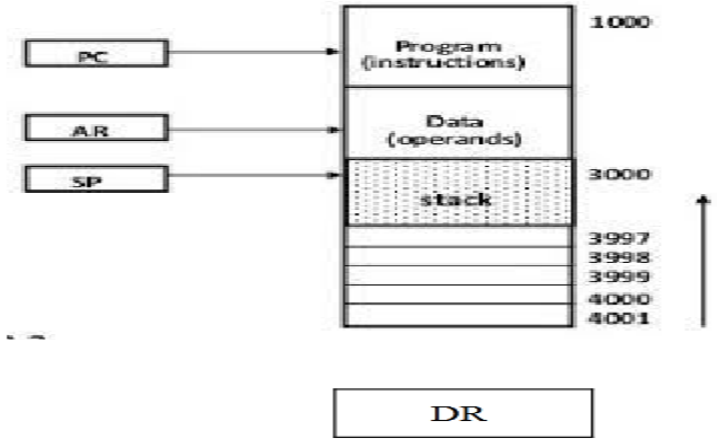


Figure 3: Memory Stack

Program- The logical part of RAM where programs are stored. The program section address starts from 1000.

● Data - It is the logical part of RAM where data (operands) are stored.The data section address starts from 2000.

● **Stack** - It is the part of RAM used to implement stack. It's address starts from 3000 and continues upto 4001.

It is not necessary that the sequence and address of each part is fixed. It is logically decided by the manufacturer.

Program counter (PC) is used to point the program counter.

Data register is used to store data

Data is pointed by the address pointer or register.

There is also a stack pointer.l

Push operation.	**POP operation**
SP ← SP - 1.	DR ← M[SP]
M[SP]. ← DR.	SP ← SP + 1

Consider its application - Evaluation of mathematical expressions using Reverse Polish Notation. The common mathematical method of writing arithmetic expressions imposes difficulties when evaluated by a computer. The Polish mathematician Lukasiewicz showed that arithmetic expressions can be represented in prefix notation as well as postfix notation.

Infix	Prefix or Polish.	Postfix or reverse Polish notation
A + B	+AB	AB+

Example :. (3 * 4) * (5 * 6). 3 4* 5 6* *

						→	6				
→	4			→	5		5	→	30		
3	3	→	12		12		12		12	→	360

Actually, this is how a computer evaluates the mathematical expressions. It is one of the applications of stack.

Instruction Formats:

The physical and logical structure of computers is normally described in reference manuals provided with the system. Such manuals explain the internal construction of the CPU, including the processor registers available and their logical capabilities.

The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. A computer instruction is often divided into three parts

OP-code field - specifies the operation to be performed

Address field - designates memory address or a processor register

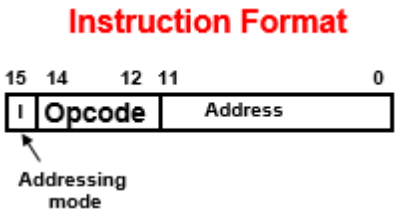
Mode field - determines how the address field is to be interpreted (to) get effective address or the operand)

• The number of address fields in the instruction format depends on the internal organization of CPU.

• In the Basic Computer, since the memory contains 4096 (= 212) words, we needs 12 bit to specify which memory address this instruction will use

• In the Basic Computer, bit 15 of the instruction specifies the addressing mode (0: direct addressing, 1: indirect addressing)

• Since the memory words, and hence the instructions, are 16 bits long, that leaves 3 bits for the instruction's opcode



Computers may have instructions of several different lengths containing varying number of addresses. The number of address fields in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organizations:

1 Single accumulator organization.

2 General register organizations.

3 Stack organizations.

An example of an accumulator-type organization is the basic computer. All operations are performed with an implied accumulator register. The instruction format in this type of computer uses one address field. For example, the instruction that specifies an arithmetic addition is defined by an assembly language instruction as ADD.

Where X is the address of the operand. The ADD instruction in this case results in the operation AC ← AC +M[X]. AC is the accumulator register and M[X] symbolizes the memory word located at address X.

An example of a general register type of organization . The instruction format in this type of computer needs three register address fields. Thus the instruction for an arithmetic addition may be written in an assembly language as ADD R1, R2, R3 To denote the operation R1 ← R2 + R3. The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers.

An example of stack-organization . Computers with stack organization would have PUSH and POP instructions which require an address field. Thus the instruction PUSH X

Will push the word at address X to the top of the stack. The stack pointer is updated automatically. Operation-type instructions do not need an address field in stack-organized computers. This is because the operation is performed on the two items that are on top of the stack. The instruction ADD

The three most common CPU organizations:

Single accumulator organization:

ADD X AC ← AC + M[X]

General register organization:

ADD R1, R2, R3 R1 ← R2 + R3
ADD R1, R2 R1 ← R1 + R2
MOV R1, R2 R1 ← R2
ADD R1, X R1 ← R1 + M[X]

Stack organization:

PUSH X TOS <--M[X]
ADD

Three-Address Instructions
Program to evaluate X = (A + B) * (C + D) :

```
ADD   R1,   A,   B :   R1 <--M[A] + M[B]
ADD   R2,   C,    D :  R2 <--M[C] + M[D]
MUL   X,    R1,   R2 :  M[X] <--R1 * R2
```

Results in short programs , Instruction becomes long (many bits)

Two-Address Instructions
Program to evaluate X = (A + B) * (C + D) :

```
MOV   R1,   A    R1 <--M[A]
ADD   R1,   B    R1 <--R1 + M[A]
MOV   R2,   C    R2 <--M[C]
ADD   R2,   D    R2 <--R2 + M[D]
MUL   R1,   R2   R1 <--R1 * R2
MOV   X,    R1   M[X] <--R1
```

One-Address Instructions
Use an implied AC register for all data manipulation

Program to evaluate X = (A + B) * (C + D) :

```
LOAD   A   AC <--M[A]
ADD    B   AC <--AC + M[B]
STORE  T   M[T] <--AC
LOAD   C   AC <--M[C]
ADD    D   AC <--AC + M[D]
MUL    T   AC <--AC * M[T]
STORE  X   M[X] <--AC
```

Zero-Address Instructions
Can be found in a stack-organized computer
Program to evaluate X = (A + B) * (C + D) :

```
PUSH   A   TOS <--A
PUSH   B   TOS <--B
ADD        TOS <--(A + B)
PUSH   C   TOS <--C
PUSH   D   TOS <--D
ADD        TOS <--(C + D)
MUL        TOS <--(C + D) * (A + B)
POP    X   M[X] <--TOS
```

Addressing modes:

Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:

1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.
2. To reduce the number of bits in the addressing field of the instruction.

PC holds the address of the instruction to be executed next and is incremented each time an instruction is fetched from memory. An example of an instruction format with a distinct addressing mode field is shown in Fig(27).

Figure 27 Instruction format with mode field.

Immediate Mode: In this mode the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field.

Register Mode: In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction.

Register Indirect Mode: In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.

Auto-increment or Auto-decrement Mode: This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.

The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode.

Direct Address Mode: In this mode the effective address is equal to the address part of the instruction.

Indirect Address Mode: In this mode the address field of the instruction gives the address where the effective address is stored in memory.

The effective address in these modes is obtained from the following computation:

effective address = address part of instruction + content of CPU register

Relative Address Mode: In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.

Indexed Addressing Mode: In this mode the content of an index register is added to the

address part of the instruction to obtain the effective address.

Base Register Addressing Mode: In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.

Numerical Example:

1- The two-word instruction at address 200 and 201 is a "load to AC" instruction with an address field equal to 500.

2- The first word of the instruction specifies the operation code and mode, and the second word specifies the address part.

3- PC has the value 200 for fetching this instruction.

4- The content of processor register Rl is 400, and the content of an index register XR is 100.

AC receives the operand after the instruction is executed. The Fig(28) lists a few pertinent addresses and shows the memory content at each of these addresses.

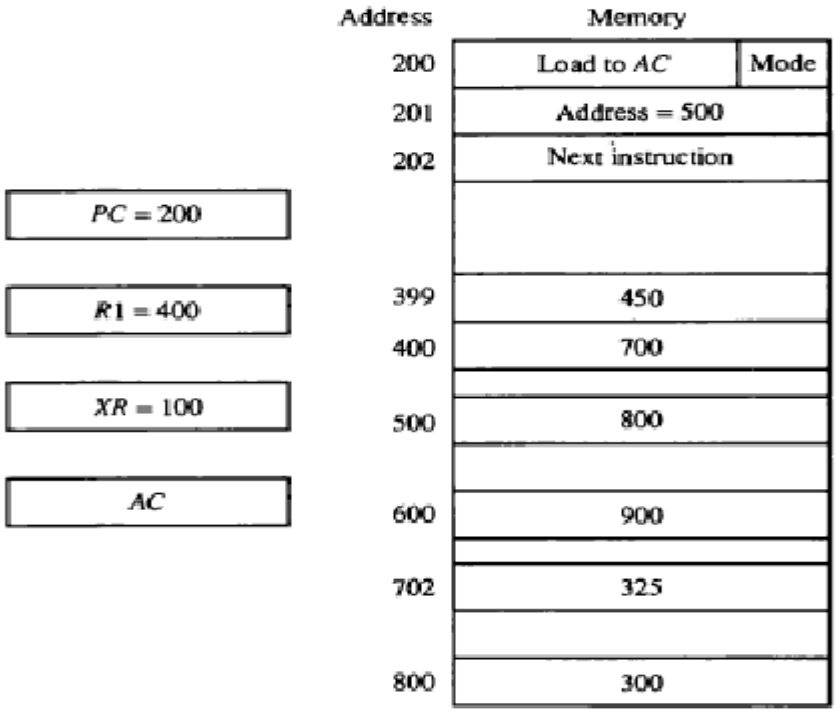


Figure 28 Numerical example for addressing modes.

Answer:

1- In the direct address mode the effective address is the address part of the instruction 500 and the operand to be loaded into AC is 800.

2- In the immediate mode the second word of the instruction is taken as the operand rather than an address, so 500 is loaded into AC. (The effective address in this case is 201)

3- In the indirect mode the effective address is stored in memory at address 500. Therefore, the effective address is 800 and the operand is 300.

4- In the relative mode the effective address is 500 + 202 = 702 and the operand is 325. (Note that the value in PC after the fetch phase and during the execute phase is 202)

5- In the index mode the effective address is XR + 500 = 100 + 500 = 600 and the operand is 900.

6- In the register mode the operand is in Rl and 400 is loaded into AC. (There is no effective address in this case)

7- In the register indirect mode the effective address is 400, equal to the content of Rl and the operand loaded into AC is 700.

8- The auto-increment mode is the same as the register indirect mode except that Rl is incremented to 401 after the execution of the instruction.

9- The auto-decrement mode decrements Rl to 399 prior to the execution of the instruction. The operand loaded into AC is now 450.

Table (16) lists the values of the effective address and the operand loaded into AC for the nine addressing modes.

TABLE 16 Tabular List of Numerical Example

Addressing Mode	Effective Address	Content of AC
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

Data Transfer and Manipulation Instructions

Data Transfer Instructions:

Data transfer instructions cause transfer of data from one location to another without changing the binary information. The most common transfer are between the ·

- Memory and Processor registers
- Processor registers and input output devices
- Processor registers themselves

Typical Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data manipulation Instructions

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. These instructions perform arithmetic, logic and shift operations.

Three Basic Types:

- *Arithmetic instructions*
- *Logical and bit manipulation instructions*
- *Shift instructions*

Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

Logical and Bit Manipulation Instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

4. Logical Instructions

The logical instructions define the set of operations performed by the processor Arithmetic Logic Unit (ALU). In this section we will discuss only the four key operations of logical instructions: **AND, OR, XOR and NOT**

Basic format of instructions

No.	Instruction	Basic Format
1	AND	AND operand1, operand2
2	OR	OR operand1, operand2
3	XOR	XOR operand1, operand2
4	NOT	NOT operand1

Table 1: Logical Instructions and their basic formats

The first operand in most cases refers to a register or memory. The second operand could be either in a register, memory or a constant value.

AND instruction

The AND instruction is used for performing operations on bits.

Binary Inputs		
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Table 2: Possible binary states for two inputs & Output of AND logic with two inputs

Example 1

- **MOV AX, 10100011 ;** Copy the binary value 10100011 to the accumulator
- MOV BX, 00111101 ; Copy the binary value 00111101 to register BX
- AND AX, BX ; Perform an AND operation on the values in registers AX and BX. Store the output in the accumulator

---

The AND operation is performed as shown below:
Operand1 (AX): 1010 0011
Operand2 (BX): 0011 1101
_____-

Operand1 (AX): 0010 0001

The accumulator AX will have the value 00100001 after the AND operation is executed. This value will be stored in the 8-bit low-order register AL of the accumulator.

NOTE: AX (16-Bits) = AH(Higher Order 8-bits) + AL (Lower Order 8-bits)

OR instruction

The OR instruction is used for performing operations on bits.

Binary Inputs		
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Table 3: Possible binary states for two inputs & Output of OR logic with two inputs

Example 2

- **MOV AX, 10100011**
- MOV BX, 00111101
- OR AX, BX

The AND operation is performed as shown below:
Operand1 (AX): 1010 0011
Operand2 (BX): 0011 1101
_____-

Operand1(AX): 1011 1111

The accumulator AX will have the value 10111111 after the OR operation is executed. This value will be stored in the 8-bit low order register AL of the accumulator.

XOR instruction

The XOR instruction is used for performing operations on bits.

Binary Inputs		
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Table 4: Possible binary states for two inputs & Output of XOR logic with two inputs

Example 3

- **MOV AX, 10100011**
- MOV BX, 00111101

- XOR AX, BX

The XOR operation is performed as shown below:
Operand1 (AX): 1010 0011
Operand2 (BX): 0011 1101
_____-
Operand1(AX): 1001 1110
The accumulator AX will have the value 10011110 after XOR operation is executed. This value will be stored in the 8-bit low order register AL of the accumulator.

NOT instruction

The NOT instruction is used for performing operations on bits.

Binary Inputs	
A	Output
0	1
1	0

Table 5: Possible binary states for one input & Output of NOT logic with one input

---

Example 4
• MOV AL, 10100011
• NOT AL
The NOT operation is performed as shown below:
Operand1 (AL): 1010 0011
_____-
Operand1(AL): 0101 1100
The accumulator AL will have the value 0101 1100 after the NOT operation is executed.

NOTE:

Bitwise **Logical instructions** are the most primitive operations needed by every computer architecture. At a minimum, an architecture could provide a NAND operations, since all other logical functions can be derived from NAND operations.

Program Control Instructions

Instructions are always stored in successive memory locations. When processed in the CPU, the instructions are fetched from consecutive memory locations and executed.

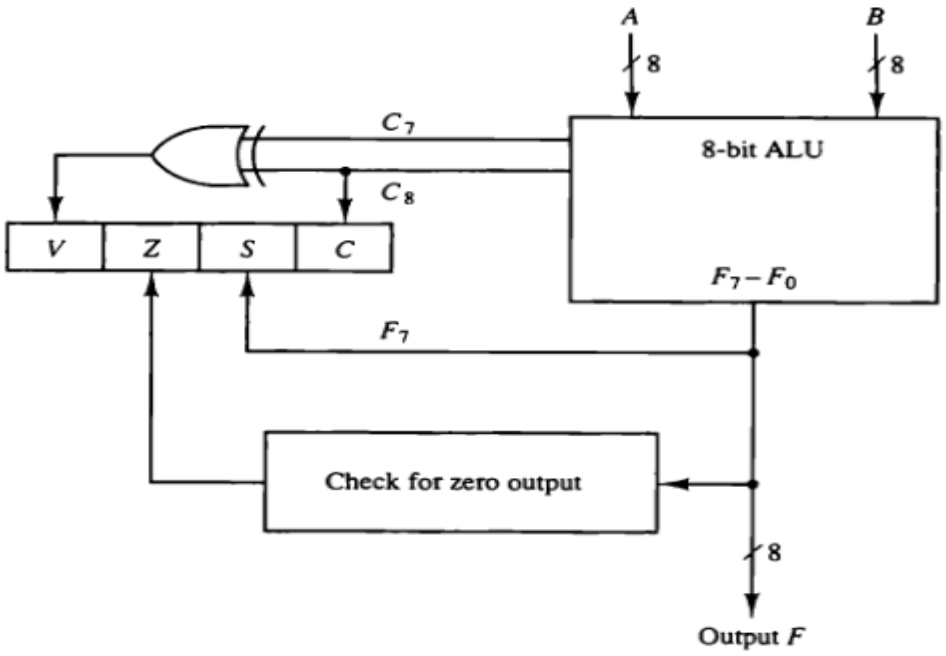
Typical Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

Status Bit Conditions

It is sometimes convenient to supplement the ALU circuit in the CPU with a status register where status bit conditions can be stored for further analysis. Status bits are also called condition-code bits or flag bits. The four status bits are symbolized by C, S, Z, and V. The bits are set or cleared as a result of an operation performed in the ALU.

1. Bit C (carry) is set to 1 if the end carry C8 is 1. It is cleared to 0 if the carry 0.
2. Bit S (sign) is set to 1 if the highest-order bit F? is 1. It is set to 0 if the bit is 0.
3. Bit Z (zero) is set to 1 if the output of the ALU contains all 0's. It is cleared to 0 otherwise. In other words, Z = 1 if the output is zero and Z = 0 if the output is not zero.
4. Bit V (overflow) is set to 1 if the exclusive-OR of the last two carries is equal to 1, and Cleared to 0 otherwise. This is the condition for an overflow when negative numbers are in 2's complement. For the 8-bit ALU, V = 1 if the output is greater than +127 or less than -128.



Status register bits.

Conditional Branch Instructions

Conditional Branch Instructions

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 1$
BNZ	Branch if not zero	$Z = 0$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$
	Unsigned compare conditions ($A - B$)	
BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$
	Signed compare conditions ($A - B$)	
BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

Some computers consider the C bit to be a borrow bit after a subtraction operation A — B. A Borrow does not occur if A ^ B, but a bit must be borrowed from the next most significant Position if A < B. The condition for a borrow is the complement of the carry obtained when the subtraction is done by taking the 2's complement of B. For this reason, a processor that considers the C bit to be a borrow after a subtraction will complement the C bit after adding
the 2's complement of the subtrahend and denote this bit a borrow.

Subroutine Call and Return

- A subroutine is a self-contained sequence of instructions that performs a given computational task.

- The instruction that transfers program control to a subroutine is known by different names. The most common names used are call subroutine, jump to subroutine, branch to subroutine, or branch and save address.

- The instruction is executed by performing two operations:

(1) The address of the next instruction available in the program counter (the return address) is Stored in a temporary location so the subroutine knows where to return

(2) Control is transferred to the beginning of the subroutine.

Different computers use a different temporary location for storing the return address.

- Some store the return address in the first memory location of the subroutine, some store it in a fixed location in memory, some store it in a processor register, and some store it in a memory stack. The most efficient way is to store the return address in a memory stack. The advantage of using a stack for the return address is that when a succession of subroutines is called, the sequential return addresses can be pushed into the stack. The return from subroutine instruction causes the stack to pop and the contents of the top of the stack are transferred to the program counter.

- A subroutine call is implemented with the following micro operations:

SP ←SP - 1 Decrement stack pointer

M [SP] ←PC Push content of PC onto the stack

PC ← effective address Transfer control to the subroutine

- If another subroutine is called by the current subroutine, the new return address is pushed into The stack and so on. The instruction that returns from the last subroutine is implemented by the Micro operations:

PC←M [SP] Pop stack and transfer to PC
SP ←SP + 1 Increment stack pointer

Program Interrupt

- Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request. Control returns to the original program after the service program is executed.

- The interrupt procedure is, in principle, quite similar to a subroutine call except for three Variations:

(1) The interrupt is usually initiated by an internal or external signal rather than from the Execution of an instruction (except for software interrupt as explained later);

(2) The address of the interrupt service program is determined by the hardware rather than from the address field of an instruction.

(3) An interrupt procedure usually stores all the information

- The state of the CPU at the end of the execute cycle (when the interrupt is recognized) is determined

From:

1. The content of the program counter
2. The content of all processor registers
3. The content of certain status conditions

- **Program status word the** collection of all status bit conditions in the CPU is sometimes called a program status word or PSW. The PSW is stored in a separate hardware register and contains the status information that characterizes the state of the CPU.

Types of Interrupts

- There are three major types of interrupts that cause a break in the normal execution of a Program. They can be classified as:

1. External interrupts 2. Internal interrupts 3. Software interrupts

- External interrupts come from input-output (I/O) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source.

- Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps. Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation.

- A software interrupt is initiated by executing an instruction. Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program.

Reduced Instruction Set Computer (RISC)

RISC meaning reduced instruction set. They are processors that use a small instruction set and simple addressing mode so that their instructions can be executed much faster within the CPU with less referring to the memory. This type of processor is classified as a reduced instruction set computer (RISC).

RISC instructions:

Example: Evaluate X=(A+B)*(C+D)

```
LOAD    R1,  A        R1←M[A]
LOAD    R2,  B        R2←M[B]
LOAD    R3,  C        R3←M[C]
LOAD    R4,  D        R4←M[D]
ADD     R1,  R1, R2   R1←R1+R2
ADD     R3,  R3, R2   R3←R3+R4
MUL     R1,  R1, R3   R1←R1*R3
STORE   X,   R1       M[X] ←R1
```

Characteristics of the RISC

- They contain a large number of registers.
- They use overlapped register windows to speed up procedure call and return.
- They have an efficient instruction pipeline.
- They have compiler support for efficient translation of high level language programs into machine language programs

Characteristics of CISC

- Variety of addressing modes.
- Larger number of instructions.
- Variable length of instruction formats.
- Several cycles may be required to execute one instruction.
- Instruction-decoding logic is complex.
- One instruction is required to support multiple addressing modes.

RISC Vs. CISC

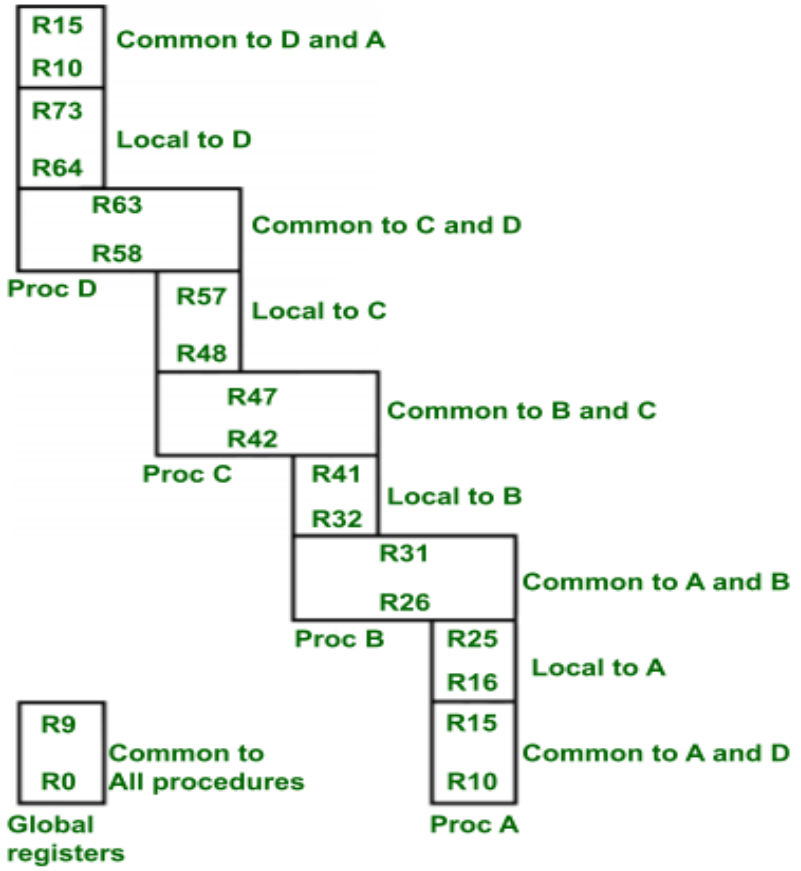
Parameter	RISC	CISC
Instruction Types	Simple	Complex
No of Instructions	Reduced(30-40)	Extended(100-200)
Duration of an Instruction	One Cycle	More Cycles (4-120)
Instruction Format	Fixed	Variable
Instruction execution	In Parallel(Pipeline)	Sequential
Addressing Modes	Simple	Complex
Instruction Accessing the memory	Two: Load and Store	Almost all from set
Register set	Multiple	Unique
Complexity	In compiler	In CPU

Register organization in RISC CPU :

- **RISC** CPUs is to use an overlapped register window that provides passing of parameters to called procedure and stores the result to the calling procedure.
- For each procedure call, the new register window is assigned from register file used by the new procedure.
- Each procedure call activates the new register window by increment a pointer and the return statement decrements the pointer which causes the activation of the previous window.
- Windows of adjacent procedures have overlapping registers that are shared to provide passing of parameters and storage of results.

Overlapped register window of RISC CPU

- In this organization, the RISC CPU contains 74 registers. Registers R0 to R9 are global registers that contain parameters which are common to all procedures.
- The remaining registers(R10 to R73) are divided into four windows to contain procedures A, B, C and D.
- Each window contains 10 local registers and two sets of 6 registers common to consecutive windows. Local registers contain local variables and common overlapped registers help to pass parameters between adjacent procedures without actual movement of data.
- One of the register window is activated at a time. The high registers of the calling procedure overlap the low registers of the called procedures, therefore parameters are passed from calling to called procedure easily.



For example:

- Procedure C called procedure D. Therefore, registers R58 to R63 are common to both procedures C and D. Therefore, procedure C stores parameters for procedure D in these registers. Procedure D uses registers R64 to R73 to store the local variables.
- When procedure B returns after performing its computation, then the result from registers(R26 to R31) is transferred back to window A.
- Registers R10 to R15 are common to procedures A and D also the four windows have a circular organization.
- As R0 to R9(i.e. 10 registers) is available all procedures. Therefore, a procedure contains 32 registers while procedure it is active(which includes 10 global ,10 local, 6 low overlapping registers and 6 high overlapping registers) .

- The organization of register windows has the relationship as
- Number of global registers = g
- Number of local registers present in each window = l
- Number of registers common to two adjacent windows = c
- Number of windows = w

Then the number of registers available for each window is calculated by

Window size = l+2c+g

The total number of registers required in the processor is

Register file = (l + c)w+ g

For example :

- In the above figure we have g= 10, l= 10, c= 6, w= 4, then
- the window size = 10+2×6+4 = 36 and register file size = (10+6)×4 + 10 = 74

Micro Programmed Control: Control memory, Address sequencing, micro program example, design of control unit.

Introduction:

- The function of the control unit in a digital computer is to initiate sequence of micro operations.
- Control unit can be implemented in two ways
  (i) Hardwired control
  (ii) Micro programmed 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
  (i) High speed of operation
  (ii) Expensive
  (iii) Relatively complex
  (iv) No flexibility of adding new instructions
- Examples of CPU with hardwired control unit are Intel 8085, Motorola 6802, Zilog 80, and any RISC CPUs.

Micro programmed Control:

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

Control Memory:

- ✓ A memory that is part of a control unit is referred to as a **control memory**.
- ✓ The control function that specifies a micro operation is called as **control variable.**
- ✓ When control variable is in one binary state, the corresponding micro operation 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 micro operations 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.
- ✓ **Micro program control unit**: A control unit whose binary control variables are stored in memory is called a micro program 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 micro program.
- ✓ The control unit consists of control memory used to store the micro program.
- ✓ 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.

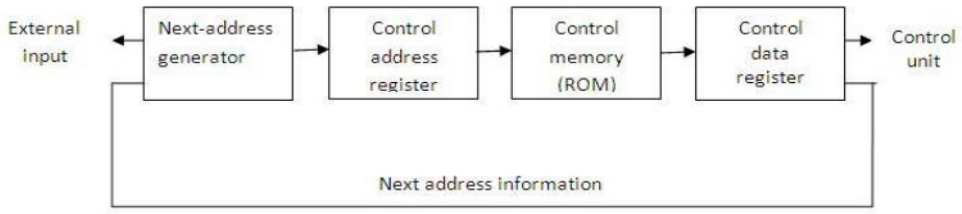


figure 4.1: Micro-programmed control organization

- ✓ 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 micro program 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 micro programmed control unit will have two separate memories: a main memory and a control memory
- ✓ The micro program consists of microinstructions that specify various internal control signals for execution of register micro operations
- ✓ These microinstructions generate the micro operations to:
  - fetch the instruction from main memory
  - evaluate the effective address
  - execute the operation
  - return control to the fetch phase for the next instruction

Address sequencing:

- ✓ Microinstructions are stored in control memory in groups, with each group specifying a routine.
- ✓ Each computer instruction has its own micro program routine to generate the micro operations.
- ✓ 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:

  (i) 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

  (ii) The control memory then goes through the routine to determine the effective address of the operand

  – AR holds operand address

  (iii) The next step is to generate the micro operations 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.

  (iv) 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:

  (i) Incrementing of the control address register.

  (ii) Unconditional branch or conditional branch, depending on status bit conditions.

  (iii) A mapping process from the bits of the instruction to an address for control memory.

  (iv) 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.

selection of address for control memory

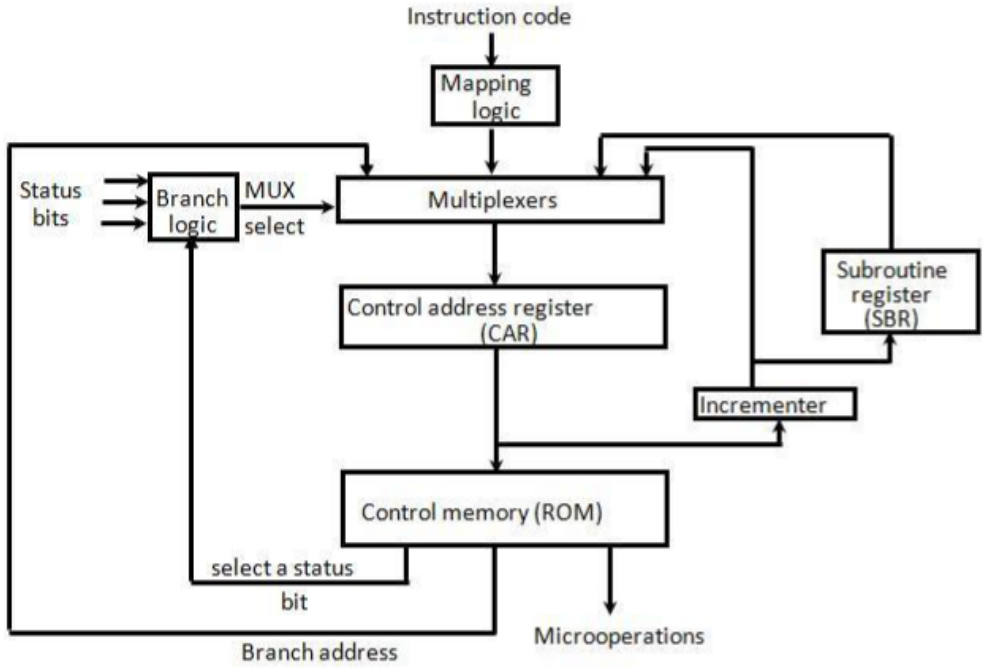


Figure 4.2: Selection of address for control memory

- ✓ The microinstruction in control memory contains a set of bits to initiate micro operations 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.

  (i) The incrementer increments the content of the control register address register by one, to select the next microinstruction in sequence.

  (ii) Branching is achieved by specifying the branch address in one of the fields of the microinstruction.

  (iii) Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition.

  (iv) An external address is transferred into control memory via a mapping logic circuit.

  (v) The return address for a subroutine is stored in a special register, that value is used when the micro program wishes to return from the subroutine.

Conditional Branching:

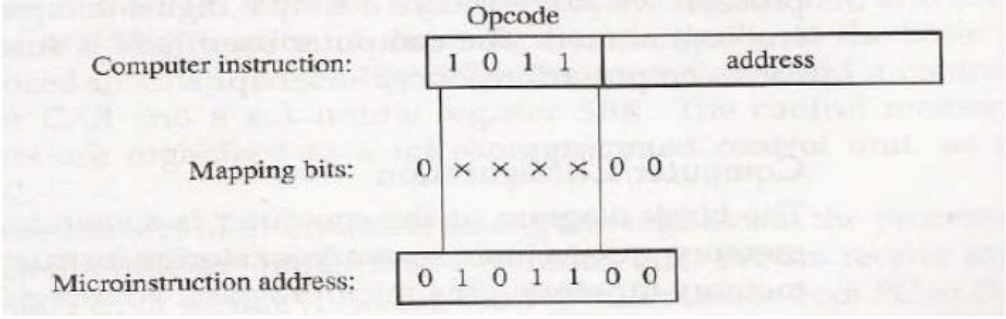
- ✓ Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition.

- ✓ 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 micro program 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 micro program 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 micro program.
- ✓ Frequently many micro programs contain identical section of code.
- ✓ Microinstructions can be saved by employing subroutines that use common sections of microcode.
- ✓ Micro programs 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

Micro program 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 micro program
- ✓ 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.

Computer Hardware Configuration

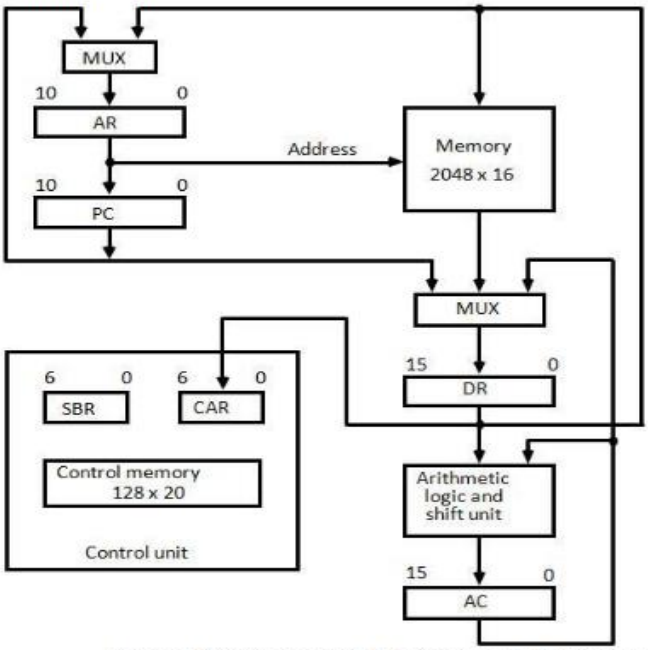
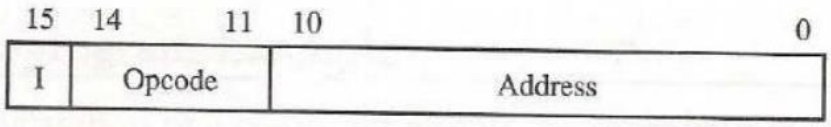


Figure 4.4: Computer hardware configuration

- 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

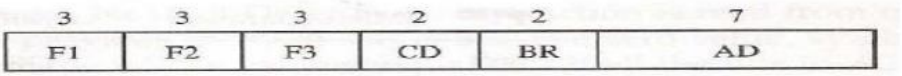
- The example will only consider the following 4 of the possible 16 memory instructions

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 micro operations 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 micro operation fields can specify one of seven possibilities.
- No more than three micro operations 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 micro operations 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 eacl 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:

$$AR \leftarrow PC$$

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

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

- 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

Design of control Unit:

- The control memory out of each subfield must be decoded to provide the distinct micro operations.
- 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.

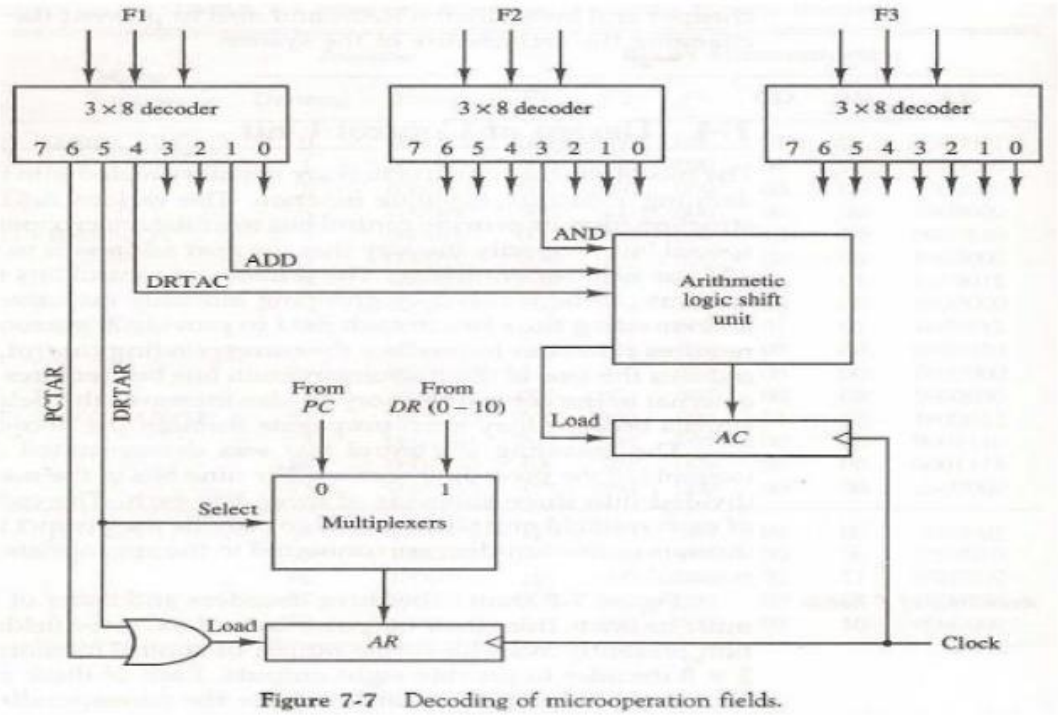


Figure 7-7 Decoding of microoperation fields.

- 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 micro operation 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

Micro program Sequencer:

- The basic components of a micro programmed control unit are the control memory and the circuits that select the next address.
- The address selection part is called a micro program sequencer.
- The purpose of a micro program 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 micro program 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.

  (i) The first multiplexer selects an address from one of four sources and routes it into control address register CAR.

  (ii) 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

  (i) The address field of the present microinstruction

  (ii) From the out of SBR

  (iii) 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.

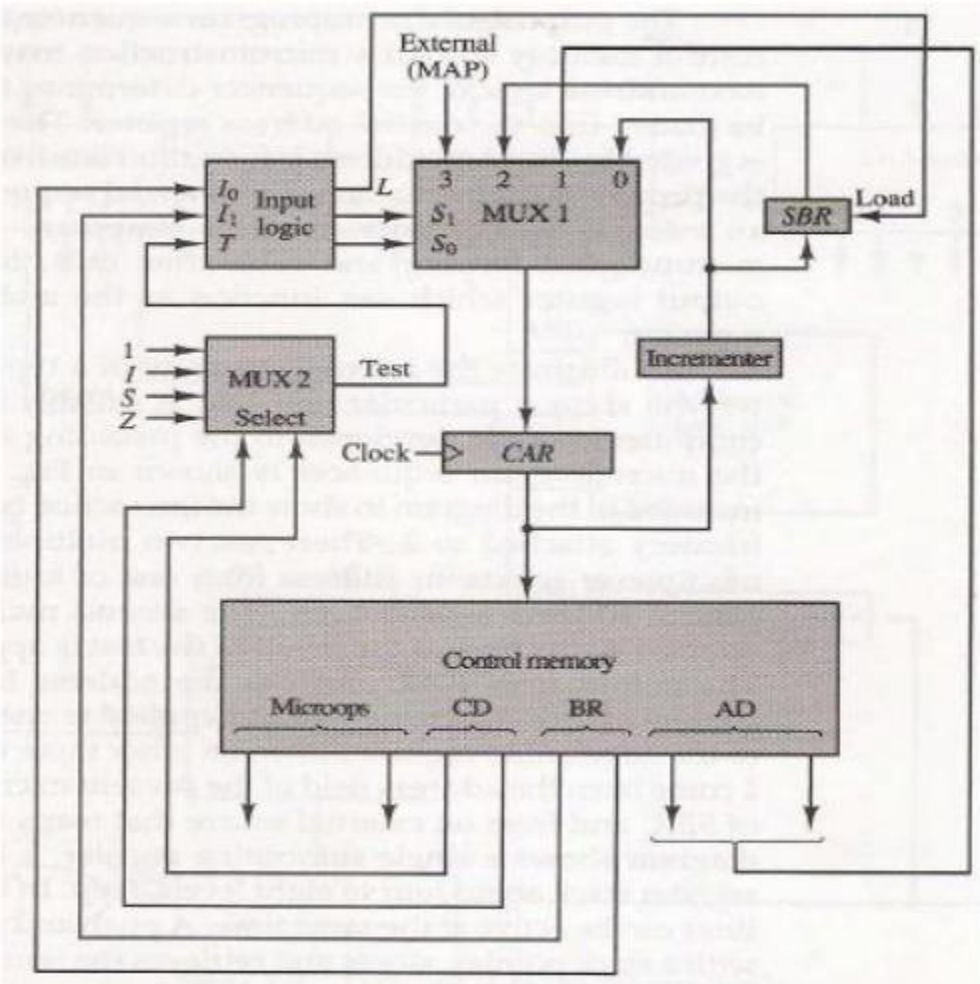


Figure 7-8 Microprogram sequencer for a control memory.

- 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:

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